Solving the kinematic problems with Python/Sympy

February 22, 2024

1 The Direct Kinematic Problem - Done by Sympy

[1]:
```
import sympy as sp
```

Let Python know some symbolic variables:

[2]:
```
t1, t2, t3 = sp.symbols('t1 t2 t3')
```

[3]:
```
t1
```

[3]: t_1

Computations with the symbolic variables will be symbolic ;-)

[3]:
```
t1+t2*t3**2
```

[3]: $t_1 + t_2 t_3^2$

We should also have symbolic names for the lengths of the upper and lower arm:

[4]:
```
l1, l2 = sp.symbols('l1 l2')
```

[5]:
```
c1, c2, t = sp.symbols('c1 c2 t')
```

[6]:
```
sp.Matrix([[l1,l2],[c1,c2]])
```

[6]: $\begin{bmatrix} l_1 & l_2 \\ c_1 & c_2 \end{bmatrix}$

We need rotations around some center (c_1, c_2) and will represent it by the homogeneous matrix

$$Rot(c_1, c_2, t) = \begin{pmatrix} \cos(t) & -\sin(t) & u \\ \sin(t) & \cos(t) & v \\ 0 & 0 & 1 \end{pmatrix}.$$

where

$$\begin{pmatrix} u \\ v \end{pmatrix} = \left(\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} - \begin{pmatrix} \cos(t) & -\sin(t) \\ \sin(t) & \cos(t) \end{pmatrix} \right) \cdot \begin{pmatrix} c_1 \\ c_2 \end{pmatrix} = \begin{pmatrix} 1-\cos(t) & \sin(t) \\ -\sin(t) & 1-\cos(t) \end{pmatrix} \cdot \begin{pmatrix} c_1 \\ c_2 \end{pmatrix}.$$

Even if the latter computation is very easy, we will submit it to Sympy:

[8]:
```
sp.simplify(sp.Matrix([[1-sp.cos(t),sp.sin(t)],[-sp.sin(t),1-sp.cos(t)]]) * sp.
↪Matrix([[c1],[c2]]))
```

[8]:
$$\begin{bmatrix} -c_1 \cos(t) + c_1 + c_2 \sin(t) \\ -c_1 \sin(t) - c_2 \cos(t) + c_2 \end{bmatrix}$$

The `simplify`-command tries to make the expression simpler. We will use the result in the following definition of the function Rot

[9]:
```
def Rot(c1,c2,t):
    return sp.Matrix([[sp.cos(t),-sp.sin(t),-c1*sp.cos(t) + c1 + c2*sp.
↪sin(t)],[sp.sin(t),sp.cos(t),-c1*sp.sin(t) - c2*sp.cos(t) + c2],[0,0,1]])
```

[10]:
```
Rot(c1,c2,t)
```

[10]:
$$\begin{bmatrix} \cos(t) & -\sin(t) & -c_1 \cos(t) + c_1 + c_2 \sin(t) \\ \sin(t) & \cos(t) & -c_1 \sin(t) - c_2 \cos(t) + c_2 \\ 0 & 0 & 1 \end{bmatrix}$$

[12]:
```
Rot(1,3,sp.pi/7)
```

[12]:
$$\begin{bmatrix} \cos\left(\frac{\pi}{7}\right) & -\sin\left(\frac{\pi}{7}\right) & -\cos\left(\frac{\pi}{7}\right) + 1 + 3\sin\left(\frac{\pi}{7}\right) \\ \sin\left(\frac{\pi}{7}\right) & \cos\left(\frac{\pi}{7}\right) & -3\cos\left(\frac{\pi}{7}\right) - \sin\left(\frac{\pi}{7}\right) + 3 \\ 0 & 0 & 1 \end{bmatrix}$$

The solution of the direct kinematic problem of the planar 3R-arm is given by the following matrix; DON'T FORGET THE OFFSET!

[13]:
```
sp.Matrix([[1,0,l1+l2],[0,1,0],[0,0,1]])
```

[13]:
$$\begin{bmatrix} 1 & 0 & l_1 + l_2 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

[14]:
```
Rot(0,0,t)
```

[14]:
$$\begin{bmatrix} \cos(t) & -\sin(t) & 0 \\ \sin(t) & \cos(t) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

[15]:
```
SolDKP = Rot(0,0,t1)*Rot(l1,0,t2)*Rot(l1+l2,0,t3)*sp.
↪Matrix([[1,0,l1+l2],[0,1,0],[0,0,1]])
SolDKP
```

[15]:
$$\begin{bmatrix} (-\sin(t_1)\sin(t_2) + \cos(t_1)\cos(t_2))\cos(t_3) + (-\sin(t_1)\cos(t_2) - \sin(t_2)\cos(t_1))\sin(t_3) & -(-\sin(t_1)\sin(t \\ (-\sin(t_1)\sin(t_2) + \cos(t_1)\cos(t_2))\sin(t_3) + (\sin(t_1)\cos(t_2) + \sin(t_2)\cos(t_1))\cos(t_3) & (-\sin(t_1)\sin(t \\ 0 & \end{bmatrix}$$

Not very nice, but true...

We will try to simplify the expression:

[16]:
```
SolDKP = sp.simplify(SolDKP)
SolDKP
```

[16]:
$$\begin{bmatrix} \cos(t_1+t_2+t_3) & -\sin(t_1+t_2+t_3) & l_1\cos(t_1)+l_2\cos(t_1+t_2) \\ \sin(t_1+t_2+t_3) & \cos(t_1+t_2+t_3) & l_1\sin(t_1)+l_2\sin(t_1+t_2) \\ 0 & 0 & 1 \end{bmatrix}$$

Done! The preceding expression *is* the solution of the direct kinematic problem.

Let's turn to the inverse kinematic problem.

2 The Inverse Kinematic Problem - Done by Sympy

We will solve the IKP in two different ways.

That is: solve the system

$$\text{SolDKP} = \begin{pmatrix} \cos(\gamma) & -\sin(\gamma) & u \\ \sin(\gamma) & \cos(\gamma) & v \\ 0 & 0 & 1 \end{pmatrix}.$$

2.1 Geometric Solution

Consider the translational part of the solution of the DKP:

$$\begin{bmatrix} l_1\cos(t_1)+l_2\cos(t_1+t_2) \\ l_1\sin(t_1)+l_2\sin(t_1+t_2) \end{bmatrix}$$

Observe that the following expression yields the very same result:

$$R_1 \cdot P_2 = \begin{bmatrix} \cos(t_1) & -\sin(t_1) \\ \sin(t_1) & \cos(t_1) \end{bmatrix} \cdot \begin{bmatrix} l_1+l_2\cos(t_2) \\ l_2\sin(t_2) \end{bmatrix}$$

[17]:
```
R1 = sp.Matrix([[sp.cos(t1),-sp.sin(t1)],[sp.sin(t1),sp.cos(t1)]])
R1
```

[17]:
$$\begin{bmatrix} \cos(t_1) & -\sin(t_1) \\ \sin(t_1) & \cos(t_1) \end{bmatrix}$$

[18]:
```
P2 = sp.Matrix([[l1+l2*sp.cos(t2)],[l2*sp.sin(t2)]])
P2
```

[18]:
$$\begin{bmatrix} l_1+l_2\cos(t_2) \\ l_2\sin(t_2) \end{bmatrix}$$

[19]:
```
sp.simplify(R1*P2)
```

[19]:
$$\begin{bmatrix} l_1\cos(t_1)+l_2\cos(t_1+t_2) \\ l_1\sin(t_1)+l_2\sin(t_1+t_2) \end{bmatrix}$$

Next, let

$$T = \begin{bmatrix} u \\ v \end{bmatrix}$$

denote the origin of the target system. We have to solve the equation

$$R_1 \cdot P_2 = T.$$

Since the rotation matrix R_1 will not change the norm of P_2, we get

$$\|P_2\| = \|T\| = u^2 + v^2.$$

For the norm (square of the length of a vector) we have to import another command:

[20]:
```
from sympy.vector import CoordSys3D
u, v = sp.symbols('u v')
```

The simplified term $\|P_2\|$ is given by

[22]:
```
sp.simplify(P2.dot(P2))
```

[22]: $l_1^2 + 2l_1l_2\cos(t_2) + l_2^2$

Thus, we let Sympy solve the equation

[23]:
```
EllbowAngle = sp.solve(sp.simplify(P2.dot(P2))-u**2-v**2,t2)
EllbowAngle
```

[23]:
```
[-acos(-(l1**2 + l2**2 - u**2 - v**2)/(2*l1*l2)) + 2*pi,
 acos((-l1**2 - l2**2 + u**2 + v**2)/(2*l1*l2))]
```

[24]:
```
EllbowAngle[0]
```

[24]: $-\operatorname{acos}\left(-\dfrac{l_1^2 + l_2^2 - u^2 - v^2}{2l_1l_2}\right) + 2\pi$

[25]:
```
EllbowAngle[1]
```

[25]: $\operatorname{acos}\left(\dfrac{-l_1^2 - l_2^2 + u^2 + v^2}{2l_1l_2}\right)$

For **both** angles we get some vector P_2 and have to solve $R_1 \cdot P_2 = T$, i.e., we have to rotate P_2 around the origin by the angle t_1 to T. Clearly, we may use the "intelligent" arctan `atan2` for this job. Thus, we obtain the following solution for t_1 (recall that we have to flip the coordinates for `atan2`):

[26]:
```
ShoulderAngle = sp.atan2(v,u) - sp.atan2(P2[1],P2[0])
ShoulderAngle
```

[26]: $\operatorname{atan}_2(v,u) - \operatorname{atan}_2(l_2\sin(t_2), l_1 + l_2\cos(t_2))$

We finish by plugging the specific values for t_2 into the latter expression:

[109]:
```
ShoulderAngle.subs(t2, EllbowAngle[0])
```

[109]: $\operatorname{atan}_2(v,u) - \operatorname{atan}_2\left(-l_2\sqrt{1 - \dfrac{\left(l_1^2 + l_2^2 - u^2 - v^2\right)^2}{4l_1^2l_2^2}}, l_1 - \dfrac{l_1^2 + l_2^2 - u^2 - v^2}{2l_1}\right)$

[110]:
```
ShoulderAngle.subs(t2, EllbowAngle[1])
```

[110]:
$$\operatorname{atan}_2(v,u) - \operatorname{atan}_2\left(l_2\sqrt{1-\frac{\left(-l_1^2-l_2^2+u^2+v^2\right)^2}{4l_1^2l_2^2}}, l_1+\frac{-l_1^2-l_2^2+u^2+v^2}{2l_1}\right)$$

We got everything we will need.

2.2 Algebraic Solution

Again, we are interested in the solution of the equation

$$\begin{bmatrix} l_1\cos(t_1)+l_2\cos(t_1+t_2) \\ l_1\sin(t_1)+l_2\sin(t_1+t_2) \end{bmatrix} = \begin{bmatrix} u \\ v \end{bmatrix}.$$

Putting $c_1=\cos(t_1)$, $c_2=\cos(t_1+t_2)$, $s_1=\sin(t_1)$, $s_2=\sin(t_1+t_2)$, we get

$$\begin{bmatrix} l_1c_1+l_2c_2 \\ l_1s_1+l_2s_2 \end{bmatrix} = \begin{bmatrix} u \\ v \end{bmatrix}.$$

If we can solve the latter equation for c_1, s_1, c_2, s_2, then we can recover $t_1 = \operatorname{atan2}(s_1, c_1)$ and $t_1+t_2 = \operatorname{atan2}(s_2, c_2)$. However, there are only two equations for the four variables, so we add two more, namely $c_1^2+s_1^2=1$ and $c_2^2+s_2^2=1$ (trigonometric Pythagoras). Thus, we have to solve the sollowing system of equations

$$\begin{aligned} l_1c_1+l_2c_2 &= u && (1)\\ l_1s_1+l_2s_2 &= v && (2)\\ c_1^2+s_1^2 &= 1 && (3)\\ c_2^2+s_2^2 &= 1 && (4) \end{aligned}$$

It is not too hard to derive the solution by paper and pencil, but we want to use Sympy:

[27]:
```
c1, c2, s1, s2 = sp.symbols('c1 c2 s1 s2')
```

[28]:
```
AlgSolve = sp.solve([l1*c1+l2*c2-u, l1*s1+l2*s2-v, c1**2+s1**2-1,c2**2+s2**2-1],␣
 ↪[c1,s1,c2,s2], dict = True)
AlgSolve
```

[28]:
```
[{c1: (l1**2*u - l2**2*u + u**3 + u*v**2 - v*sqrt(-l1**4 + 2*l1**2*l2**2 +
2*l1**2*u**2 + 2*l1**2*v**2 - l2**4 + 2*l2**2*u**2 + 2*l2**2*v**2 - u**4 -
2*u**2*v**2 - v**4))/(2*l1*(u**2 + v**2)),
  s1: (l1**2*v - l2**2*v + u**2*v + u*sqrt(-l1**4 + 2*l1**2*l2**2 + 2*l1**2*u**2
+ 2*l1**2*v**2 - l2**4 + 2*l2**2*u**2 + 2*l2**2*v**2 - u**4 - 2*u**2*v**2 -
v**4) + v**3)/(2*l1*(u**2 + v**2)),
  c2: (-l1**2*u + l2**2*u + u**3 + u*v**2 + v*sqrt(-l1**4 + 2*l1**2*l2**2 +
2*l1**2*u**2 + 2*l1**2*v**2 - l2**4 + 2*l2**2*u**2 + 2*l2**2*v**2 - u**4 -
2*u**2*v**2 - v**4))/(2*l2*(u**2 + v**2)),
  s2: -u*sqrt(-(-l1**2 - 2*l1*l2 - l2**2 + u**2 + v**2)*(-l1**2 + 2*l1*l2 -
l2**2 + u**2 + v**2))/(2*l2*(u**2 + v**2)) + v*(-l1**2 + l2**2 + u**2 +
v**2)/(2*l2*(u**2 + v**2))},
```

```
 {c1: (l1**2*u - l2**2*u + u**3 + u*v**2 + v*sqrt(-l1**4 + 2*l1**2*l2**2 +
2*l1**2*u**2 + 2*l1**2*v**2 - l2**4 + 2*l2**2*u**2 + 2*l2**2*v**2 - u**4 -
2*u**2*v**2 - v**4))/(2*l1*(u**2 + v**2)),
  s1: (l1**2*v - l2**2*v + u**2*v - u*sqrt(-l1**4 + 2*l1**2*l2**2 + 2*l1**2*u**2
+ 2*l1**2*v**2 - l2**4 + 2*l2**2*u**2 + 2*l2**2*v**2 - u**4 - 2*u**2*v**2 -
v**4) + v**3)/(2*l1*(u**2 + v**2)),
  c2: (-l1**2*u + l2**2*u + u**3 + u*v**2 - v*sqrt(-l1**4 + 2*l1**2*l2**2 +
2*l1**2*u**2 + 2*l1**2*v**2 - l2**4 + 2*l2**2*u**2 + 2*l2**2*v**2 - u**4 -
2*u**2*v**2 - v**4))/(2*l2*(u**2 + v**2)),
  s2: u*sqrt(-(-l1**2 - 2*l1*l2 - l2**2 + u**2 + v**2)*(-l1**2 + 2*l1*l2 - l2**2
+ u**2 + v**2))/(2*l2*(u**2 + v**2)) + v*(-l1**2 + l2**2 + u**2 +
v**2)/(2*l2*(u**2 + v**2))}]
```

We derive two (quite complicated) solutions; `dict = True` makes it easy to access the parts:

[29]:
```
AlgSolve[0][c1]
```

[29]:
$$\frac{l_1^2 u - l_2^2 u + u^3 + uv^2 - v\sqrt{-l_1^4 + 2l_1^2 l_2^2 + 2l_1^2 u^2 + 2l_1^2 v^2 - l_2^4 + 2l_2^2 u^2 + 2l_2^2 v^2 - u^4 - 2u^2 v^2 - v^4}}{2l_1\left(u^2 + v^2\right)}$$

[122]:
```
AlgSolve[0][s1]
```

[122]:
$$\frac{l_1^2 v - l_2^2 v + u^2 v + u\sqrt{-l_1^4 + 2l_1^2 l_2^2 + 2l_1^2 u^2 + 2l_1^2 v^2 - l_2^4 + 2l_2^2 u^2 + 2l_2^2 v^2 - u^4 - 2u^2 v^2 - v^4} + v^3}{2l_1\left(u^2 + v^2\right)}$$

[123]:
```
AlgSolve[0][c2]
```

[123]:
$$\frac{-l_1^2 u + l_2^2 u + u^3 + uv^2 + v\sqrt{-l_1^4 + 2l_1^2 l_2^2 + 2l_1^2 u^2 + 2l_1^2 v^2 - l_2^4 + 2l_2^2 u^2 + 2l_2^2 v^2 - u^4 - 2u^2 v^2 - v^4}}{2l_2\left(u^2 + v^2\right)}$$

[124]:
```
AlgSolve[0][s2]
```

[124]:
$$-\frac{u\sqrt{-\left(-l_1^2 - 2l_1 l_2 - l_2^2 + u^2 + v^2\right)\left(-l_1^2 + 2l_1 l_2 - l_2^2 + u^2 + v^2\right)}}{2l_2\left(u^2 + v^2\right)} + \frac{v\left(-l_1^2 + l_2^2 + u^2 + v^2\right)}{2l_2\left(u^2 + v^2\right)}$$

Clearly, one wouldn't use the solutions as they are. First of all, it suffices to compute the complicated terms for c_1, s_1 only, since

$$c_2 = \frac{u - l_1 c_2}{l_2} \text{ and } s_2 = \frac{v - l_1 s_1}{l_2}$$

are much "cheaper" to calculate. Furthermore, the radicants in the solutions for c_1, s_1 look pretty much the same. To prove that they are equal indeed, we have to extract these by cut and paste:

[127]:
```
print(AlgSolve[0][c1])
```

```
(l1**2*u - l2**2*u + u**3 + u*v**2 - v*sqrt(-l1**4 + 2*l1**2*l2**2 +
2*l1**2*u**2 + 2*l1**2*v**2 - l2**4 + 2*l2**2*u**2 + 2*l2**2*v**2 - u**4 -
2*u**2*v**2 - v**4))/(2*l1*(u**2 + v**2))
```

[128]:
```
print(AlgSolve[0][s1])
```

```
(l1**2*v - l2**2*v + u**2*v + u*sqrt(-l1**4 + 2*l1**2*l2**2 + 2*l1**2*u**2 +
2*l1**2*v**2 - l2**4 + 2*l2**2*u**2 + 2*l2**2*v**2 - u**4 - 2*u**2*v**2 - v**4)
+ v**3)/(2*l1*(u**2 + v**2))
```

[132]:
```
radc1 = -l1**4 + 2*l1**2*l2**2 + 2*l1**2*u**2 + 2*l1**2*v**2 - l2**4 +␣
↪2*l2**2*u**2 + 2*l2**2*v**2 - u**4 - 2*u**2*v**2 - v**4
radc1
```

[132]:
$$-l_1^4 + 2l_1^2l_2^2 + 2l_1^2u^2 + 2l_1^2v^2 - l_2^4 + 2l_2^2u^2 + 2l_2^2v^2 - u^4 - 2u^2v^2 - v^4$$

[134]:
```
rads1 = -l1**4 + 2*l1**2*l2**2 + 2*l1**2*u**2 + 2*l1**2*v**2 - l2**4 +␣
↪2*l2**2*u**2 + 2*l2**2*v**2 - u**4 - 2*u**2*v**2 - v**4
rads1
```

[134]:
$$-l_1^4 + 2l_1^2l_2^2 + 2l_1^2u^2 + 2l_1^2v^2 - l_2^4 + 2l_2^2u^2 + 2l_2^2v^2 - u^4 - 2u^2v^2 - v^4$$

[135]:
```
radc1 - rads1
```

[135]:
$$0$$

OK, they are the same, so we have to compute only once. Moreover, the structure may be nicer than one can see:

[138]:
```
Dummy = sp.factor(-l1**4 + 2*l1**2*l2**2 + 2*l1**2*u**2 + 2*l1**2*v**2 - l2**4 +␣
↪2*l2**2*u**2 + 2*l2**2*v**2 - u**4 - 2*u**2*v**2 - v**4)
Dummy
```

[138]:
$$-\left(-l_1^2 - 2l_1l_2 - l_2^2 + u^2 + v^2\right)\left(-l_1^2 + 2l_1l_2 - l_2^2 + u^2 + v^2\right)$$

[139]:
```
print(Dummy)
```

```
-(-l1**2 - 2*l1*l2 - l2**2 + u**2 + v**2)*(-l1**2 + 2*l1*l2 - l2**2 + u**2 +
v**2)
```

[140]:
```
(l1**2 + 2*l1*l2 + l2**2 - u**2 - v**2)*(-l1**2 + 2*l1*l2 - l2**2 + u**2 + v**2)
```

[140]:
$$\left(-l_1^2 + 2l_1l_2 - l_2^2 + u^2 + v^2\right)\left(l_1^2 + 2l_1l_2 + l_2^2 - u^2 - v^2\right)$$

[143]:
```
a = 2*l1*l2
b = l1**2+l2**2-u**2-v**2
sp.simplify(Dummy - (a+b)*(a-b))
```

[143]:
$$0$$

Thus, we derive the radicant by the following procedure:

$$a = 2l_1l_2 \tag{5}$$
$$b = l_1^2 + l_2^2 - u^2 - v^2 \tag{6}$$
$$d = (a+b)(a-b) \tag{7}$$

With a bit more effort:

$$c = l_1^2 - l_2^2 + u^2 + v^2 \tag{8}$$

$$c_1 = \frac{cu - v\sqrt{d}}{2l_1(u^2 + v^2)} \tag{9}$$

$$s_1 = \frac{cv + u\sqrt{d}}{2l_1(u^2 + v^2)} \tag{10}$$

$$\theta_1 = \text{atan2}(s_1, c_1). \tag{11}$$

This discussion finishes the explanations of how to use Sympy.

[ ]: