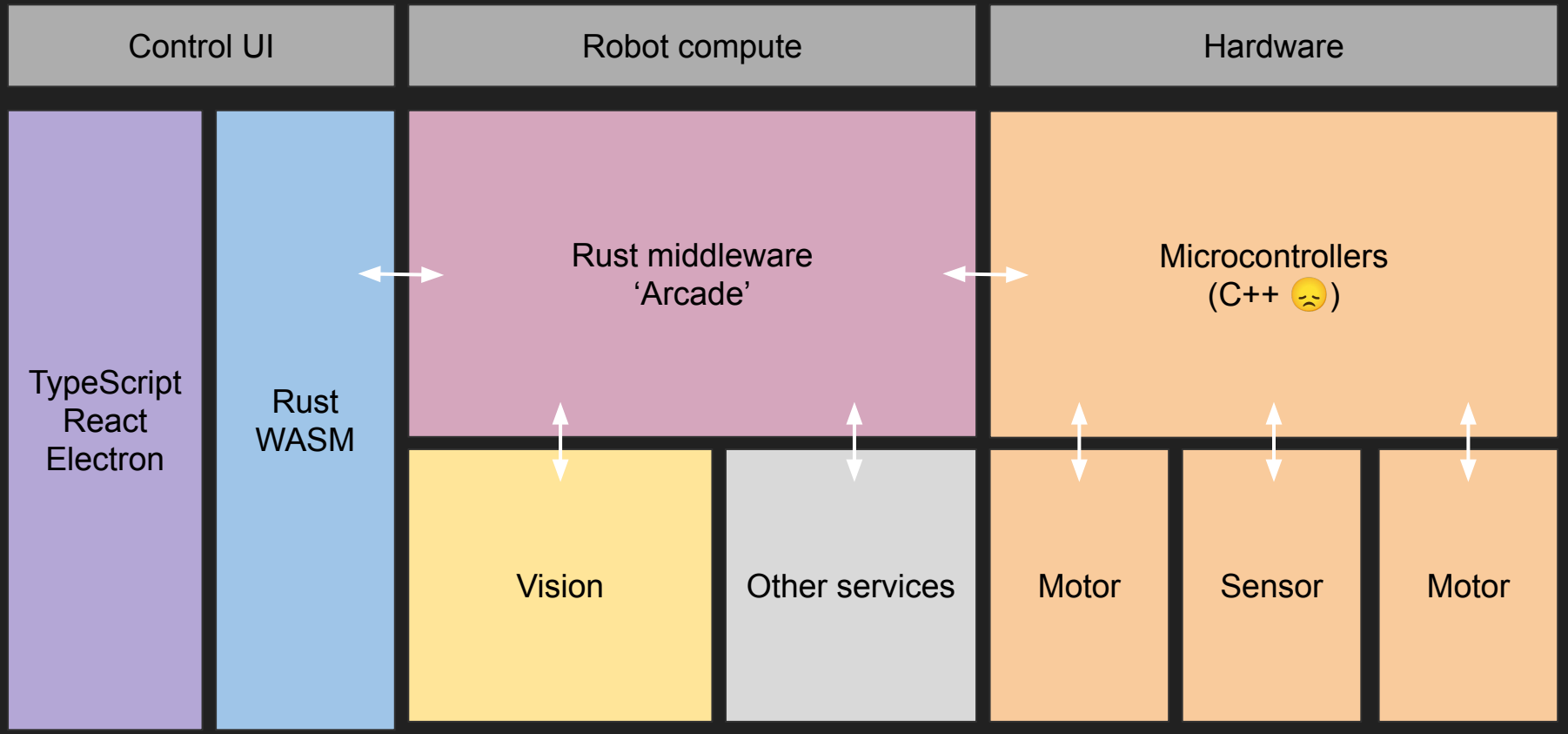
Robots that move (with Rust)

Ross Gardiner, Software Engineer @ Monumental


ENGELS
ENGELS

Control UI	Robot compute	Hardware
High level plans ‘Move the robot along this path in space’	Convert to low level streams of data ‘Send the elbow motor to 45 degrees’	Execute the instructions ‘Motor driver, start moving to 45 degrees’


Control UI
Robot compute
Hardware
TypeScript
React
Electron
Rust
WASM
Rust middleware
‘Arcade’
Microcontrollers
(C++ 😞)
Vision
Other services
Motor
Sensor
Motor

plans

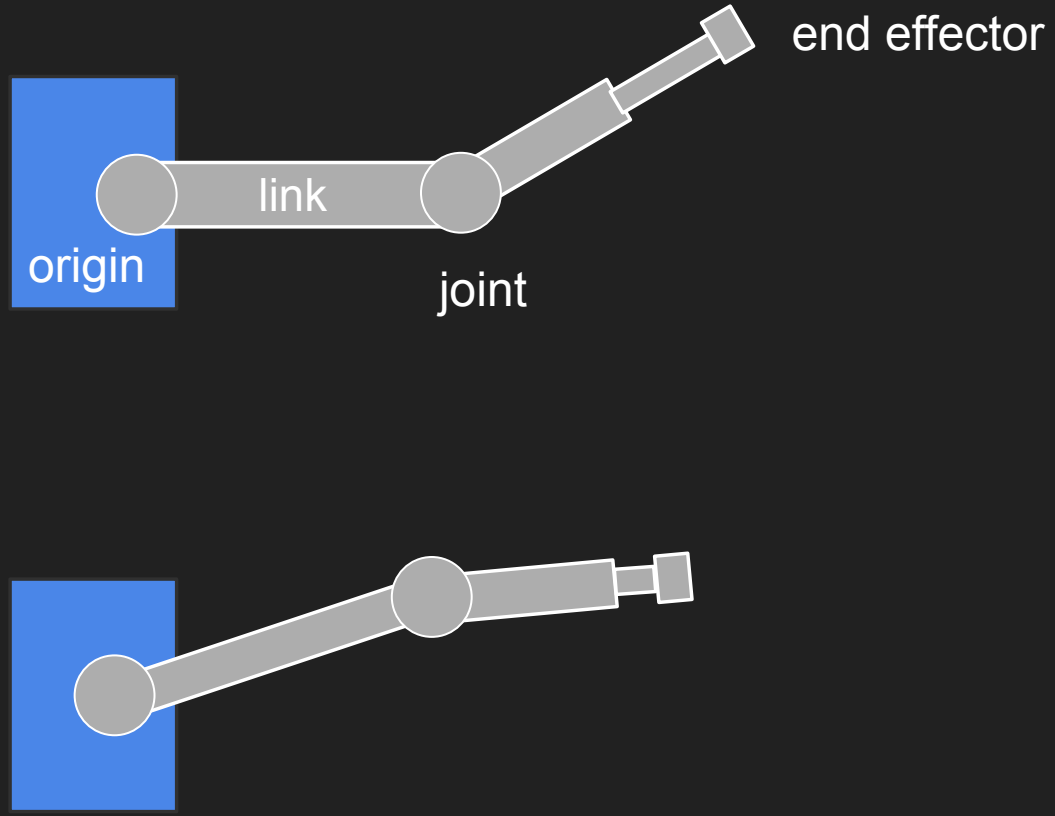
bot along
space’

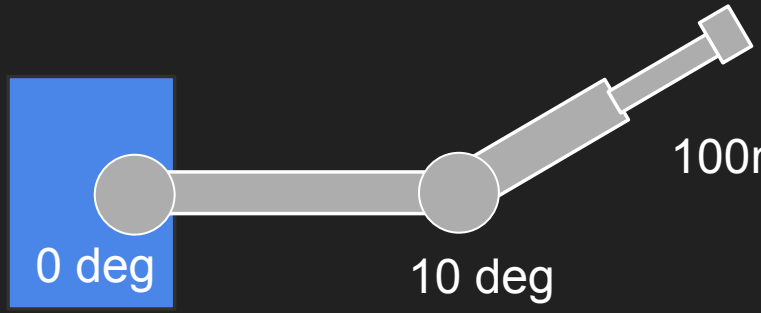
Convert to low level streams of data

‘Send the elbow motor to 45
degrees’

Exec

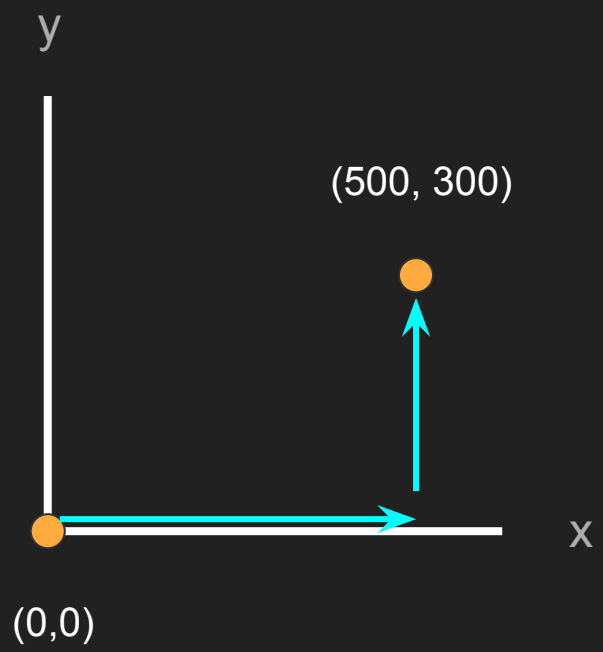
‘Motor dr


end effector
link
origin
joint


0 deg
10 deg
100mm

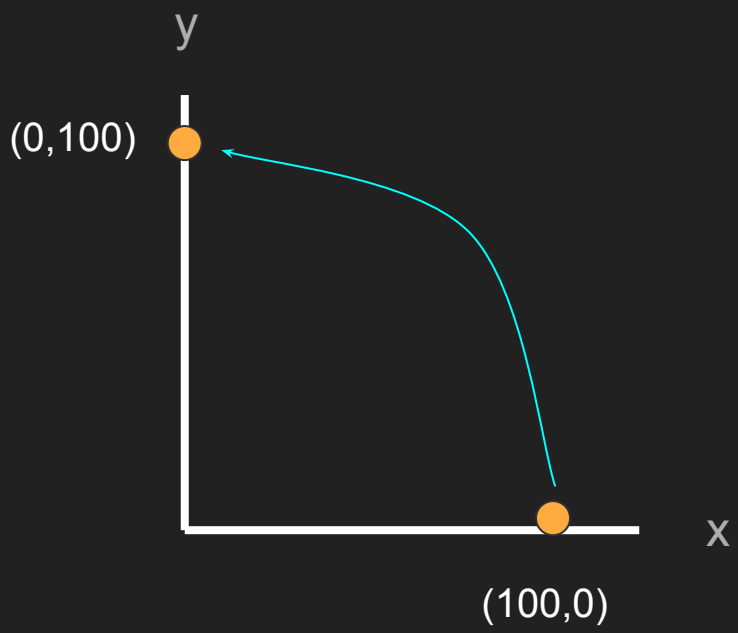

10 deg
-5 deg
20mm
where am I?

Transforming points



Translate by (500, 300)

Transforming points



Rotate by 90 degrees

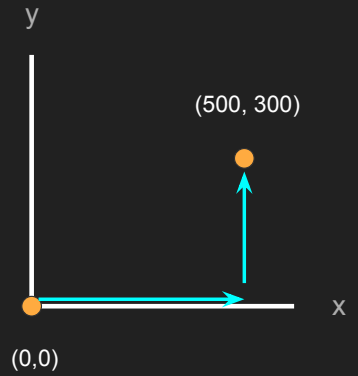
Transforming points with matrices

$$T(500, 300) = \begin{pmatrix} 1 & 0 & 500 \\ 0 & 1 & 300 \\ 0 & 0 & 1 \end{pmatrix}$$

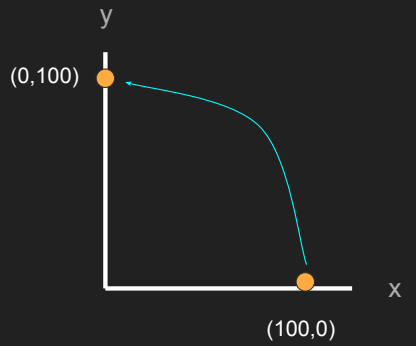
$$R(90^\circ) = \begin{pmatrix} \cos 90^\circ & -\sin 90^\circ & 0 \\ \sin 90^\circ & \cos 90^\circ & 0 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Transforming points with matrices

$$\begin{pmatrix} 1 & 0 & 500 \\ 0 & 1 & 300 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 500 \\ 300 \\ 1 \end{pmatrix}$$



$$\begin{pmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 100 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 100 \\ 1 \end{pmatrix}$$

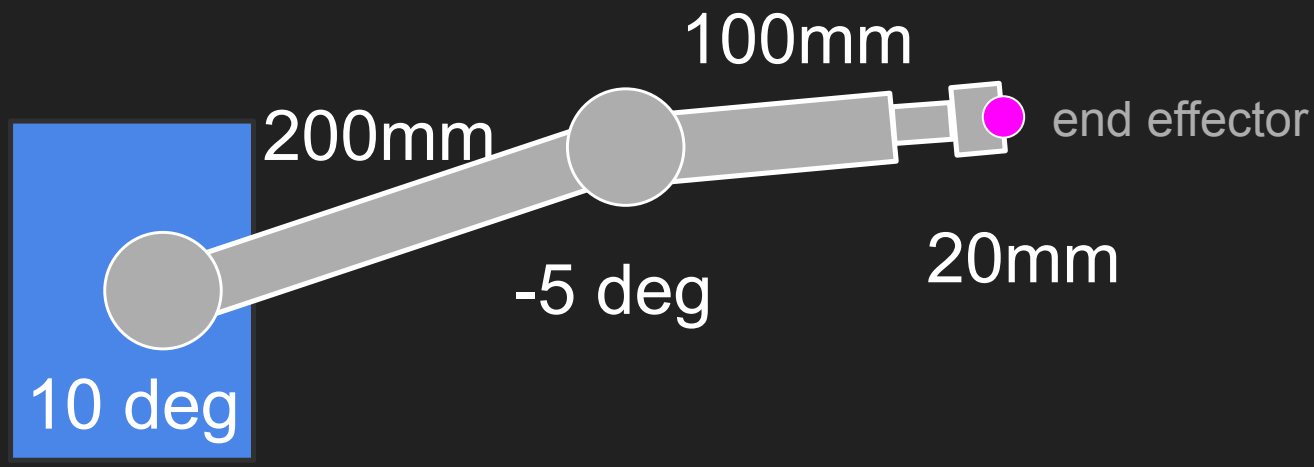


Transforming points with matrices

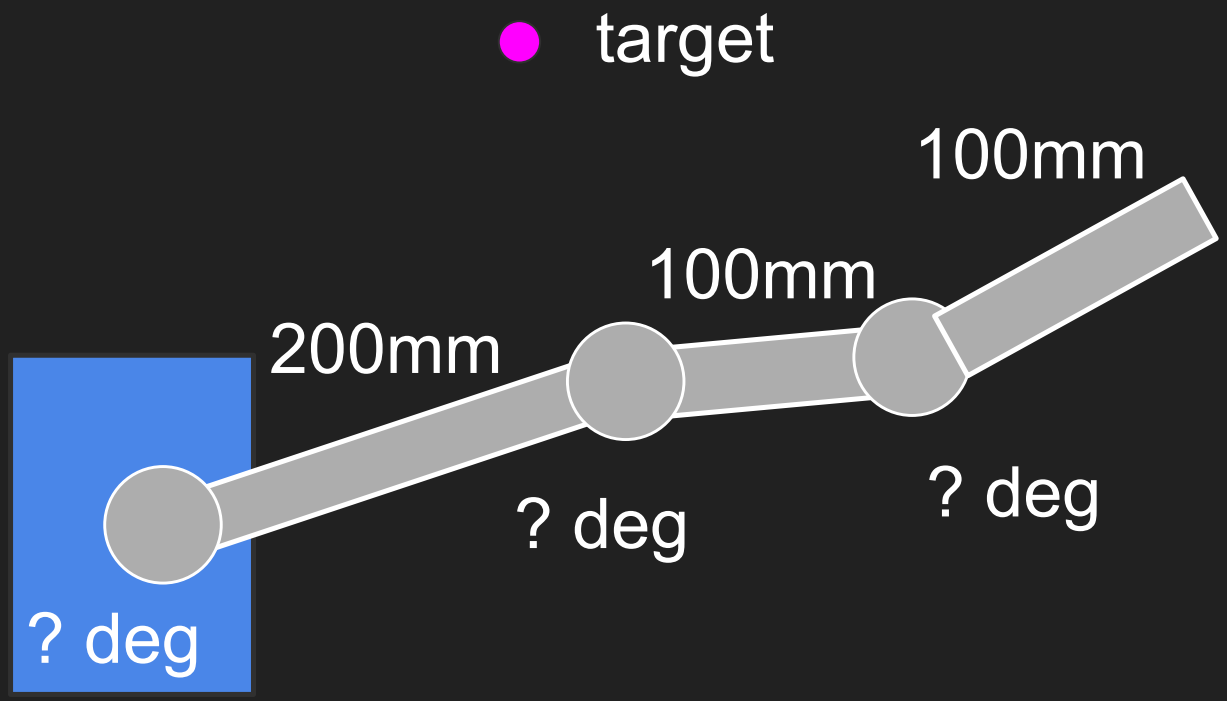
$$\begin{pmatrix} 1 & 0 & 500 \\ 0 & 1 & 300 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 500 \\ 300 \\ 1 \end{pmatrix}$$

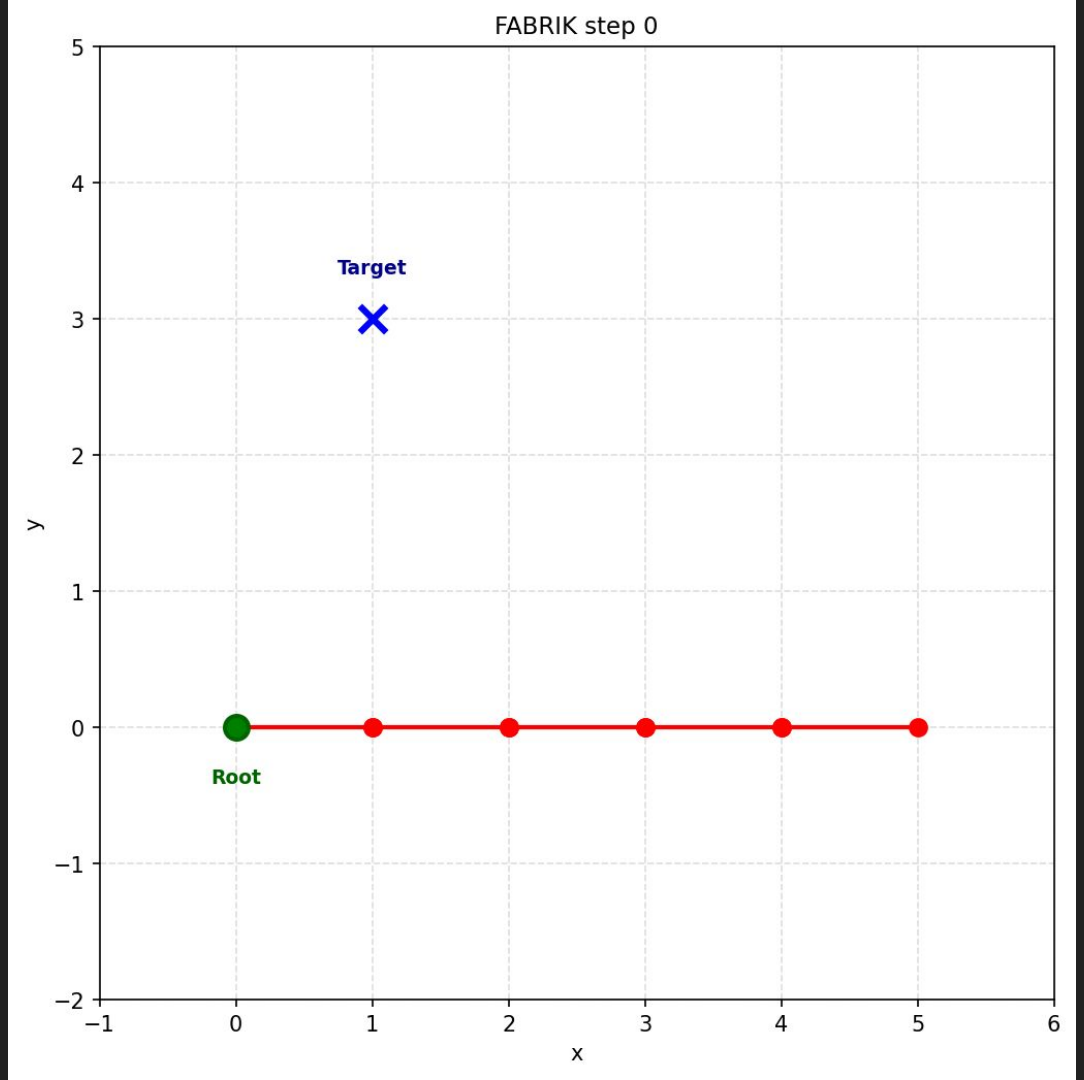
nalgebra

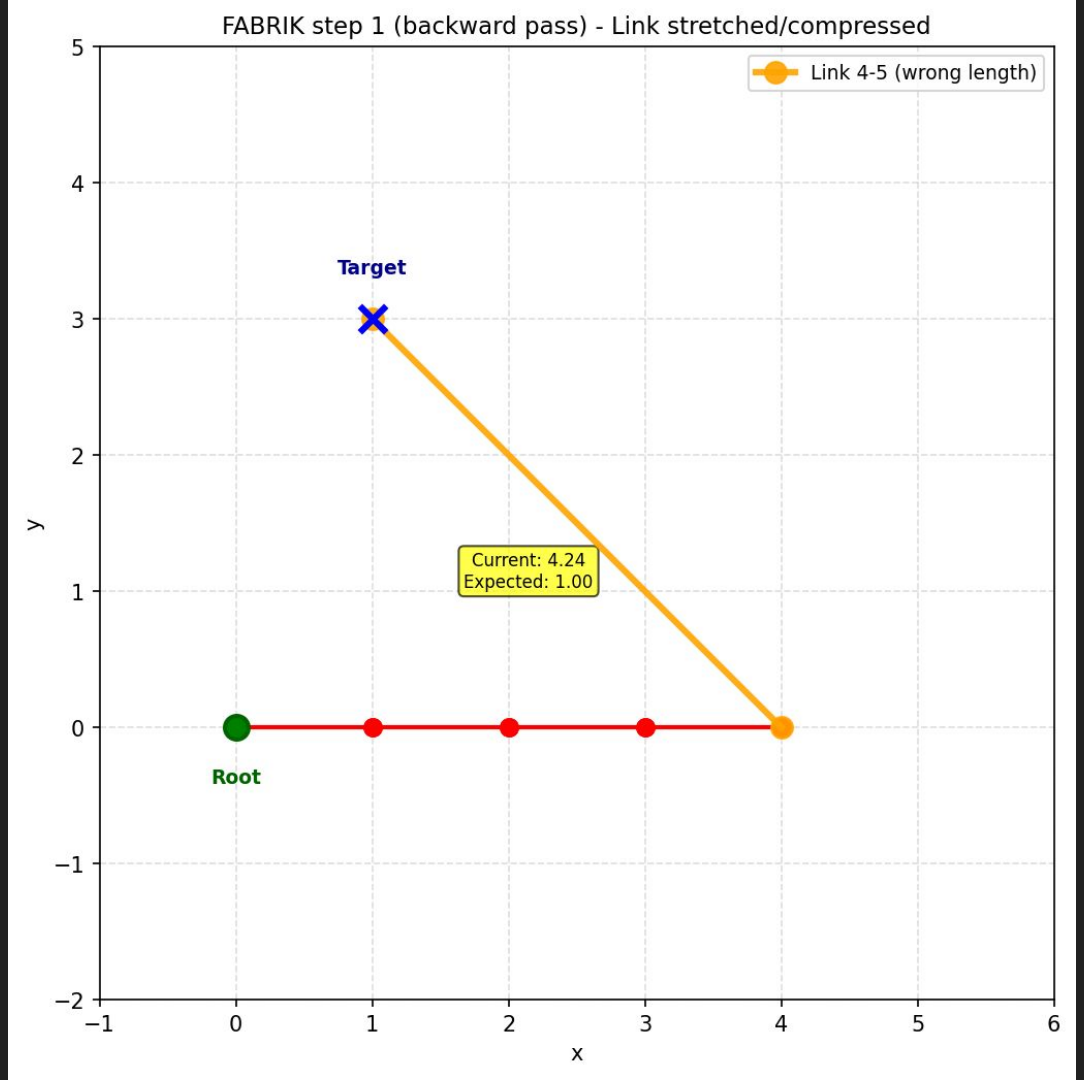
$$\begin{pmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 100 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 100 \\ 1 \end{pmatrix}$$

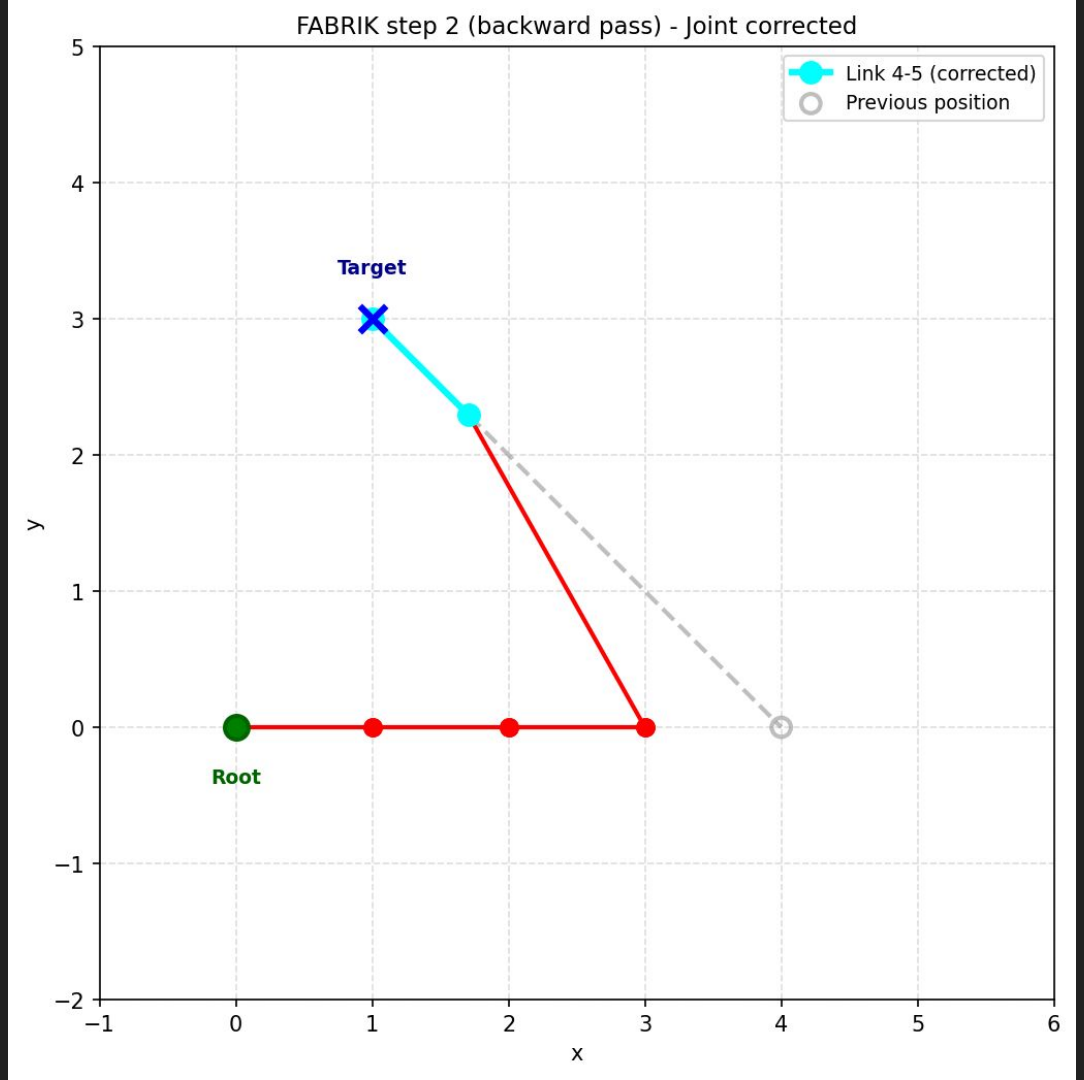


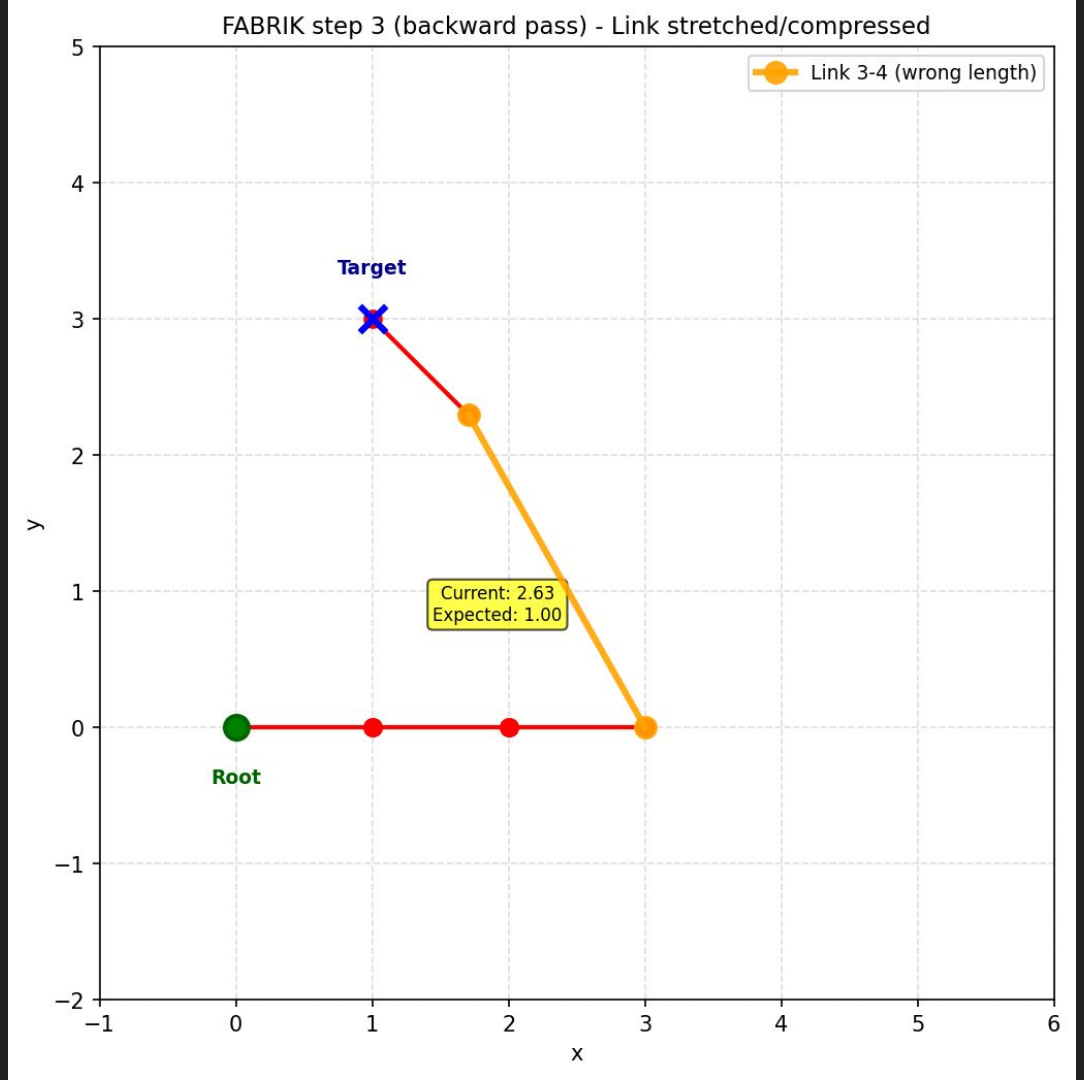
rotate 10 degrees
translate 200mm
rotate -5 degrees
translate 100mm
translate 20mm

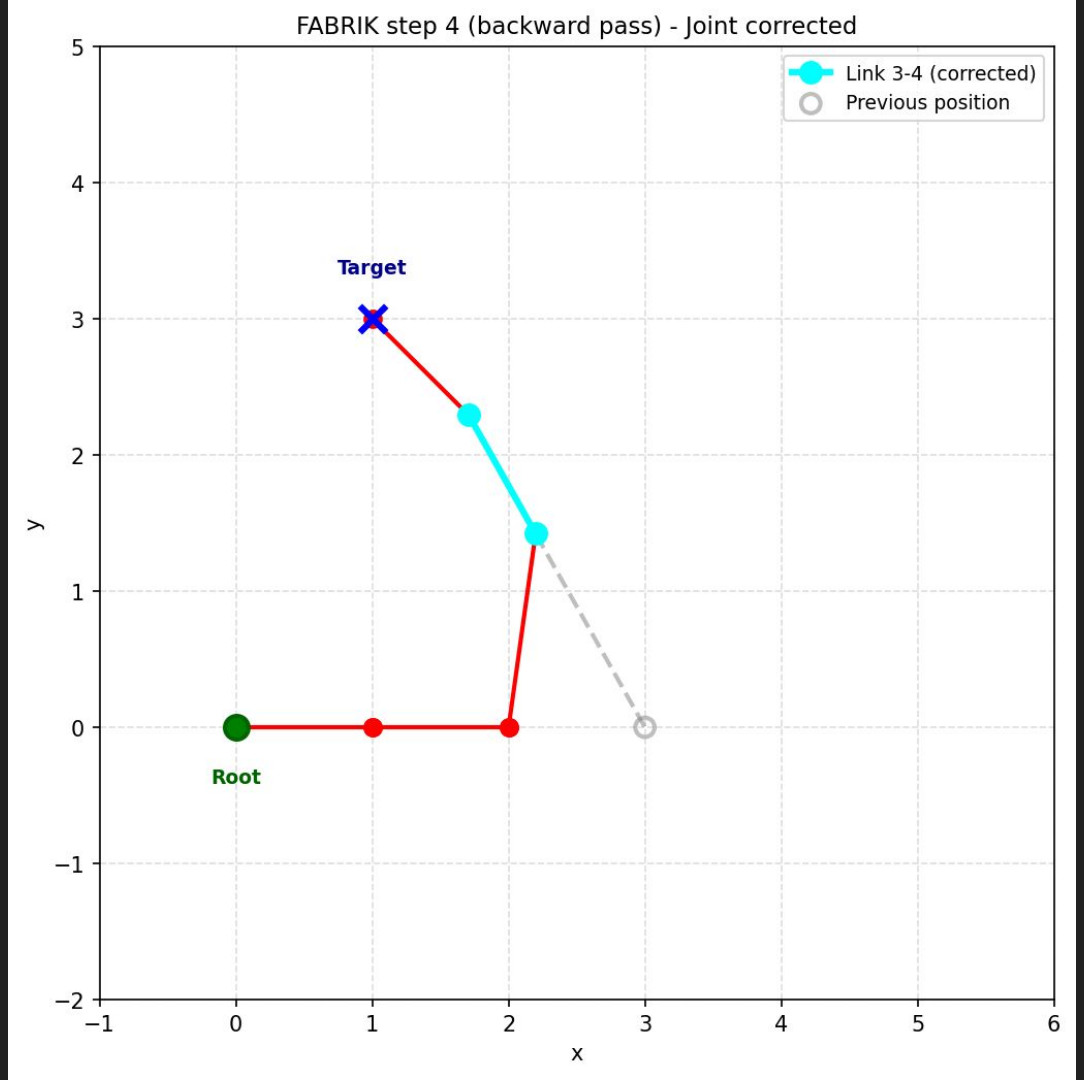

target
100mm
100mm
200mm
? deg
? deg
? deg

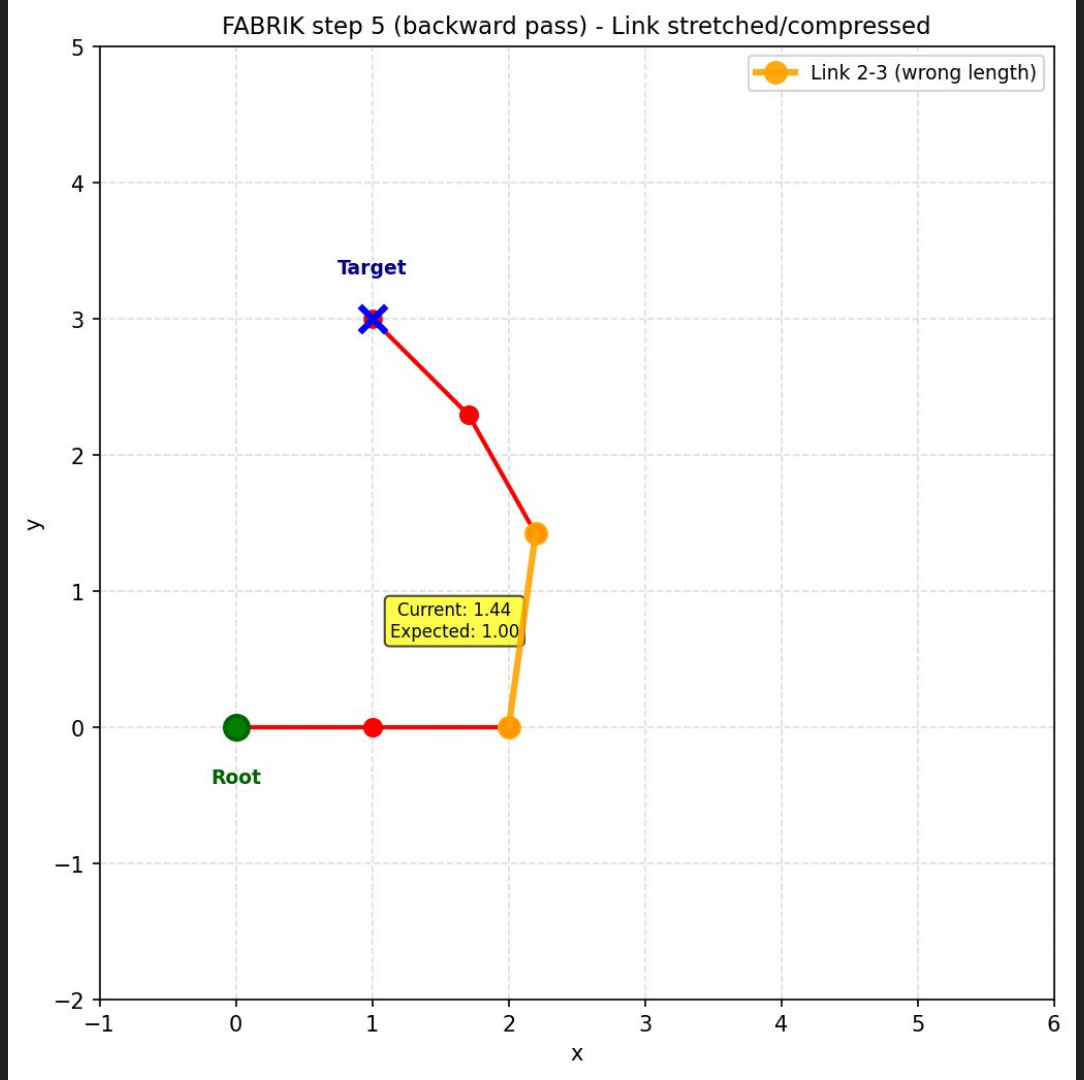

FABRIK step 0
Target
Root
x
y
5
4
3
2
1
0
−1
−2
−1
0
1
2
3
4
5
6

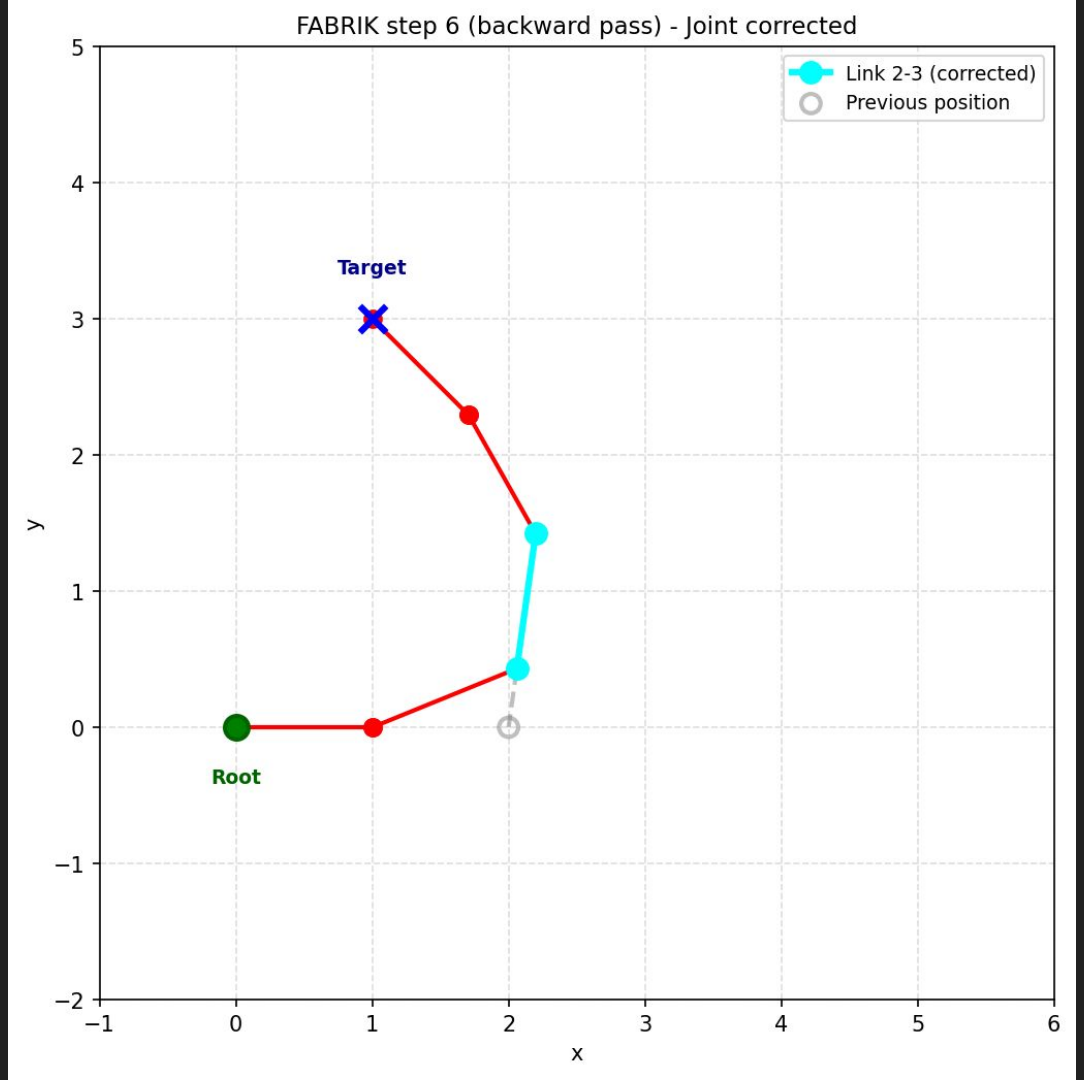

FABRIK step 1 (backward pass) - Link stretched/compressed
Link 4-5 (wrong length)
Target
Current: 4.24
Expected: 1.00
Root
x
y
5
4
3
2
1
0
−1
−2
−1
0
1
2
3
4
5
6

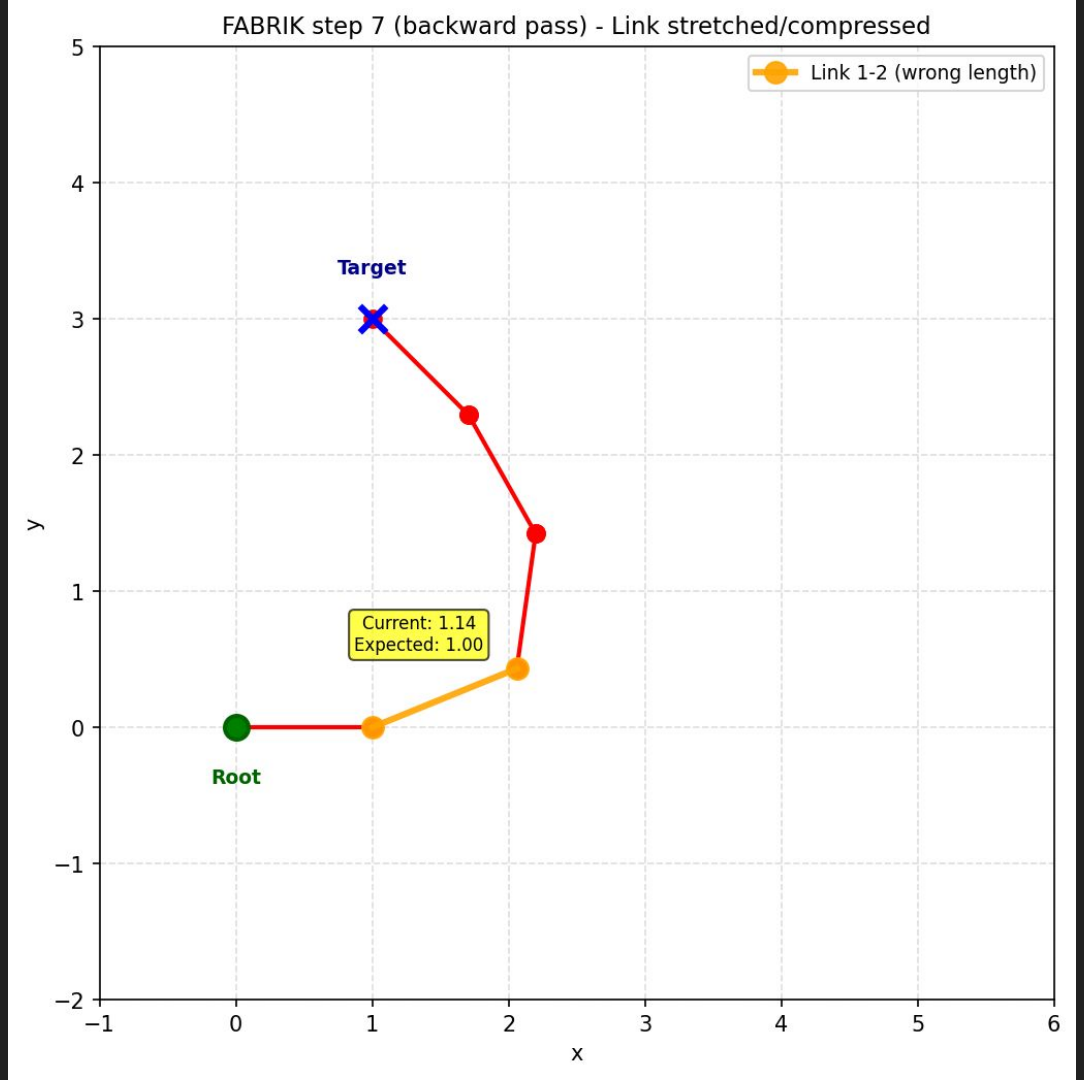

FABRIK step 2 (backward pass) - Joint corrected
Link 4-5 (corrected)
Previous position
Target
Root
x
y
5
4
3
2
1
0
−1
−2
−1
0
1
2
3
4
5
6

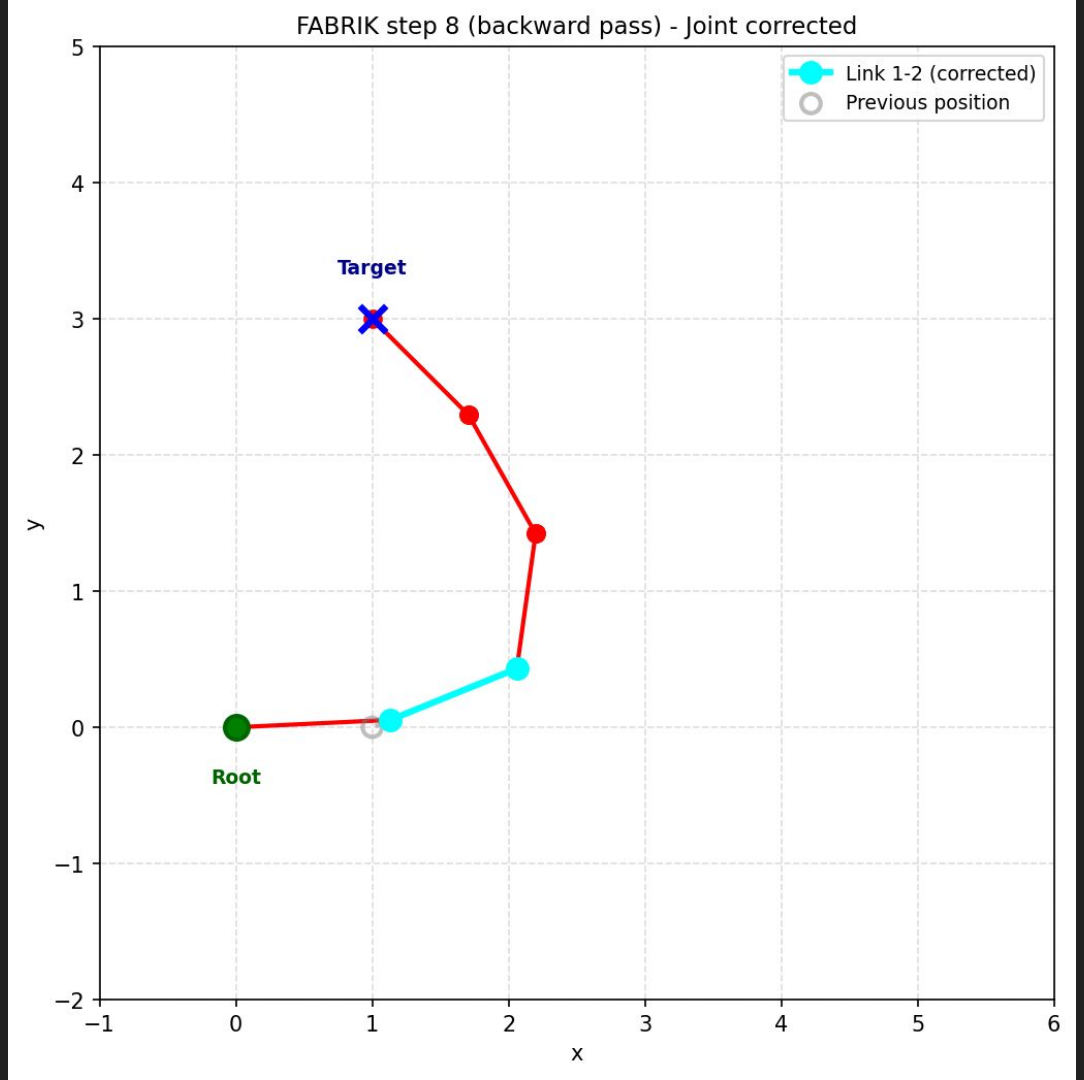

FABRIK step 3 (backward pass) - Link stretched/compressed
Link 3-4 (wrong length)
Target
Current: 2.63
Expected: 1.00
Root
x
y
5
4
3
2
1
0
−1
−2
−1
0
1
2
3
4
5
6

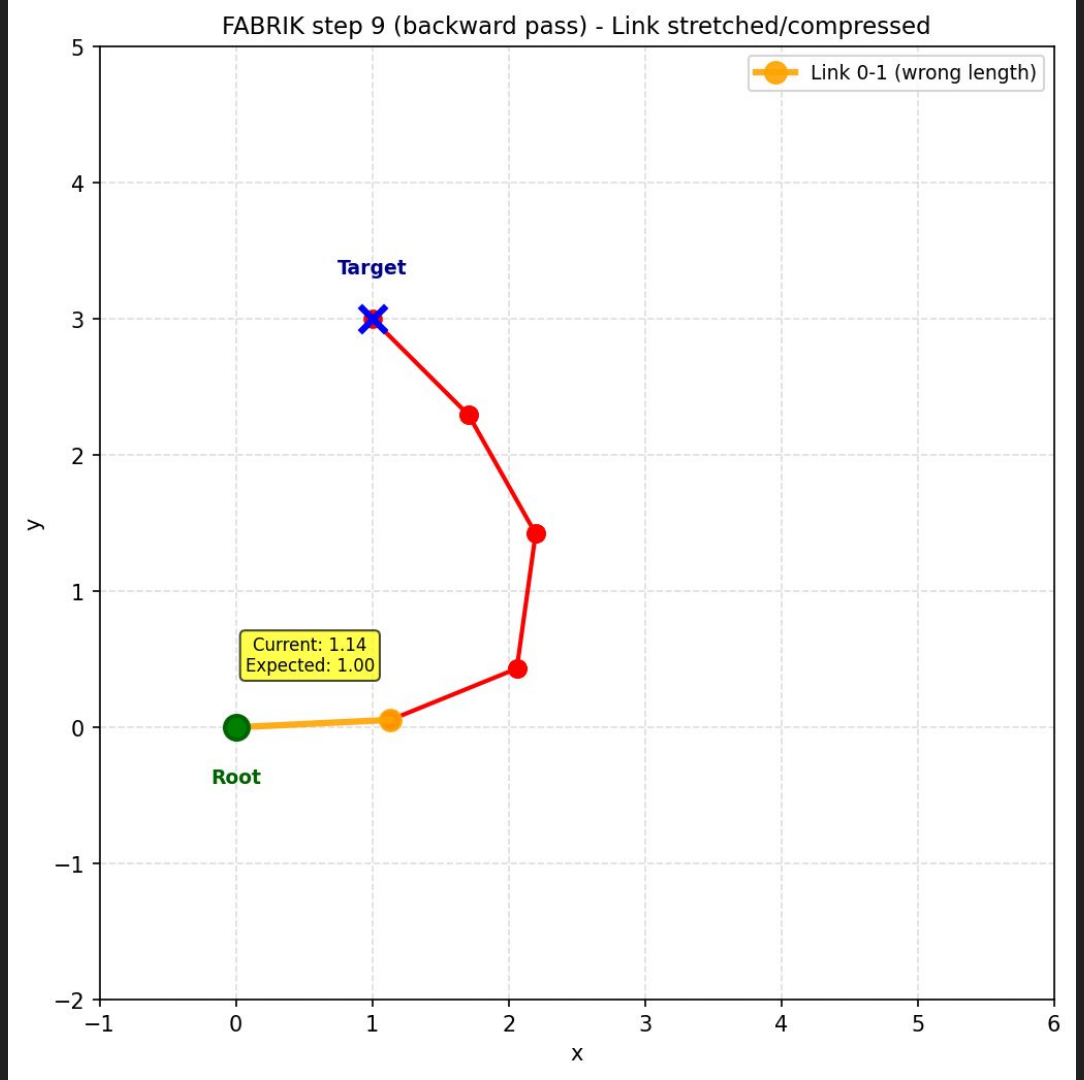

FABRIK step 4 (backward pass) - Joint corrected
Link 3-4 (corrected)
Previous position
Target
Root
x
y
5
4
3
2
1
0
−1
−2
−1
0
1
2
3
4
5
6

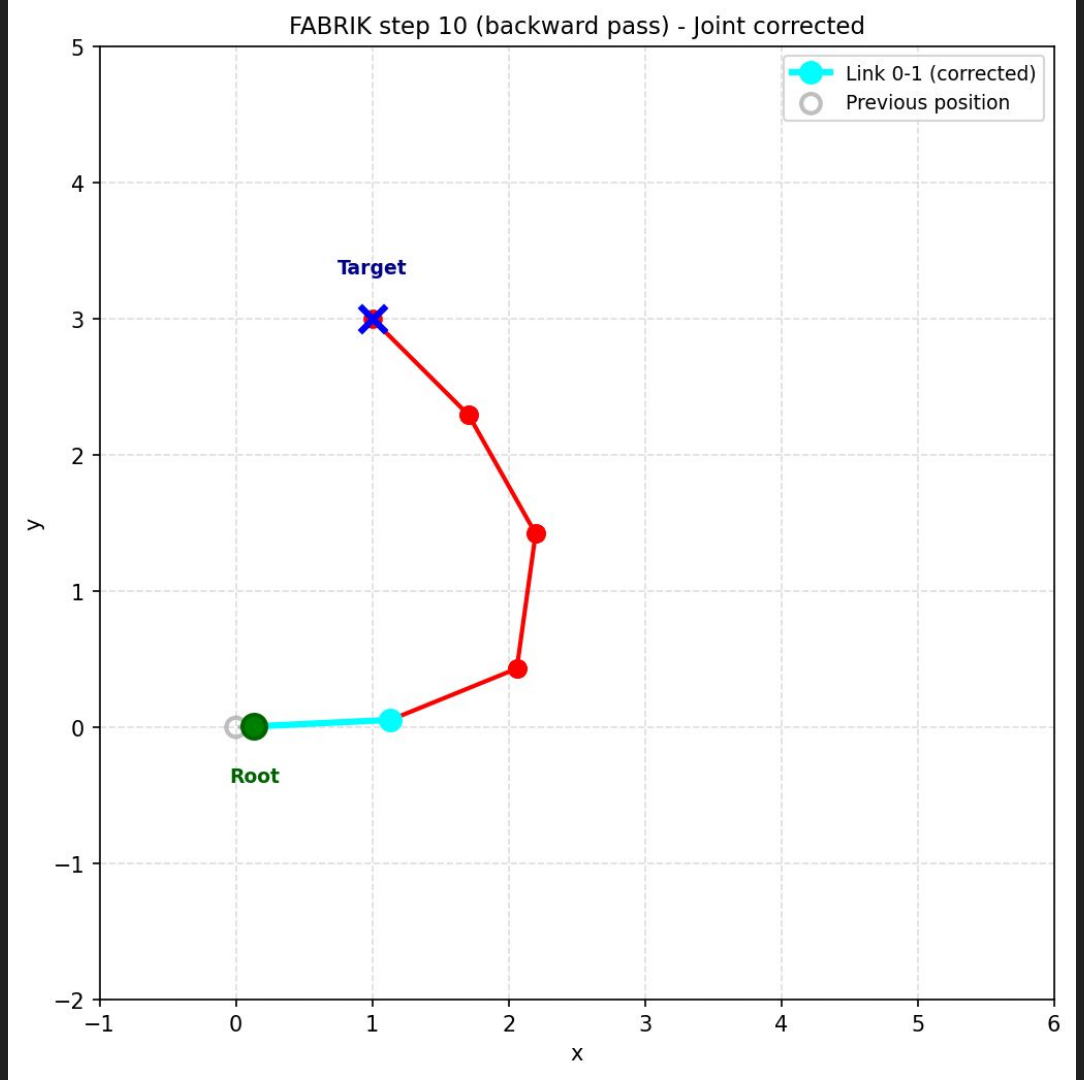

FABRIK step 5 (backward pass) - Link stretched/compressed
Link 2-3 (wrong length)
Target
Current: 1.44
Expected: 1.00
Root
x
y
5
4
3
2
1
0
−1
−2
−1
0
1
2
3
4
5
6

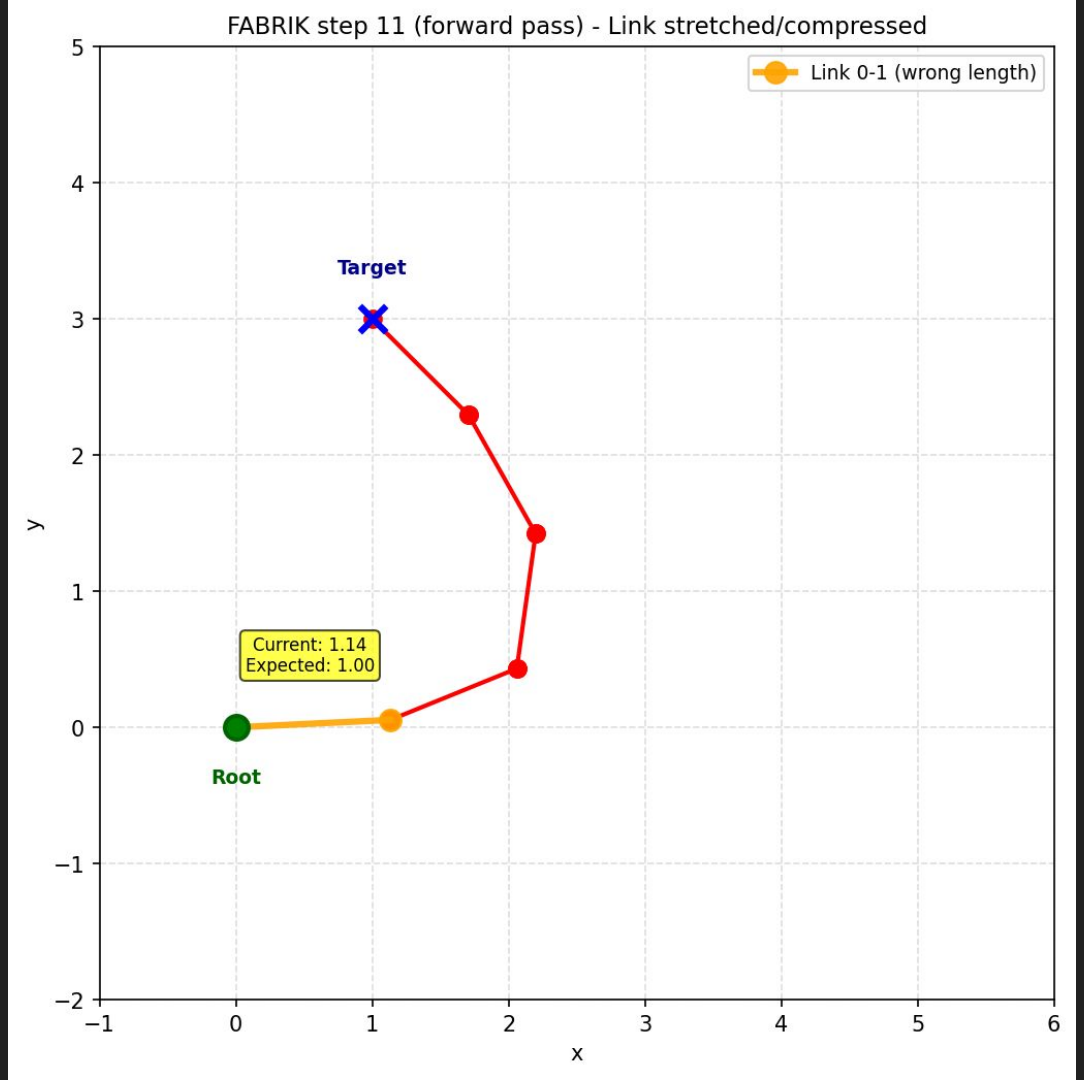

FABRIK step 6 (backward pass) - Joint corrected
Link 2-3 (corrected)
Previous position
Target
Root
x
y
5
4
3
2
1
0
−1
−2
−1
0
1
2
3
4
5
6

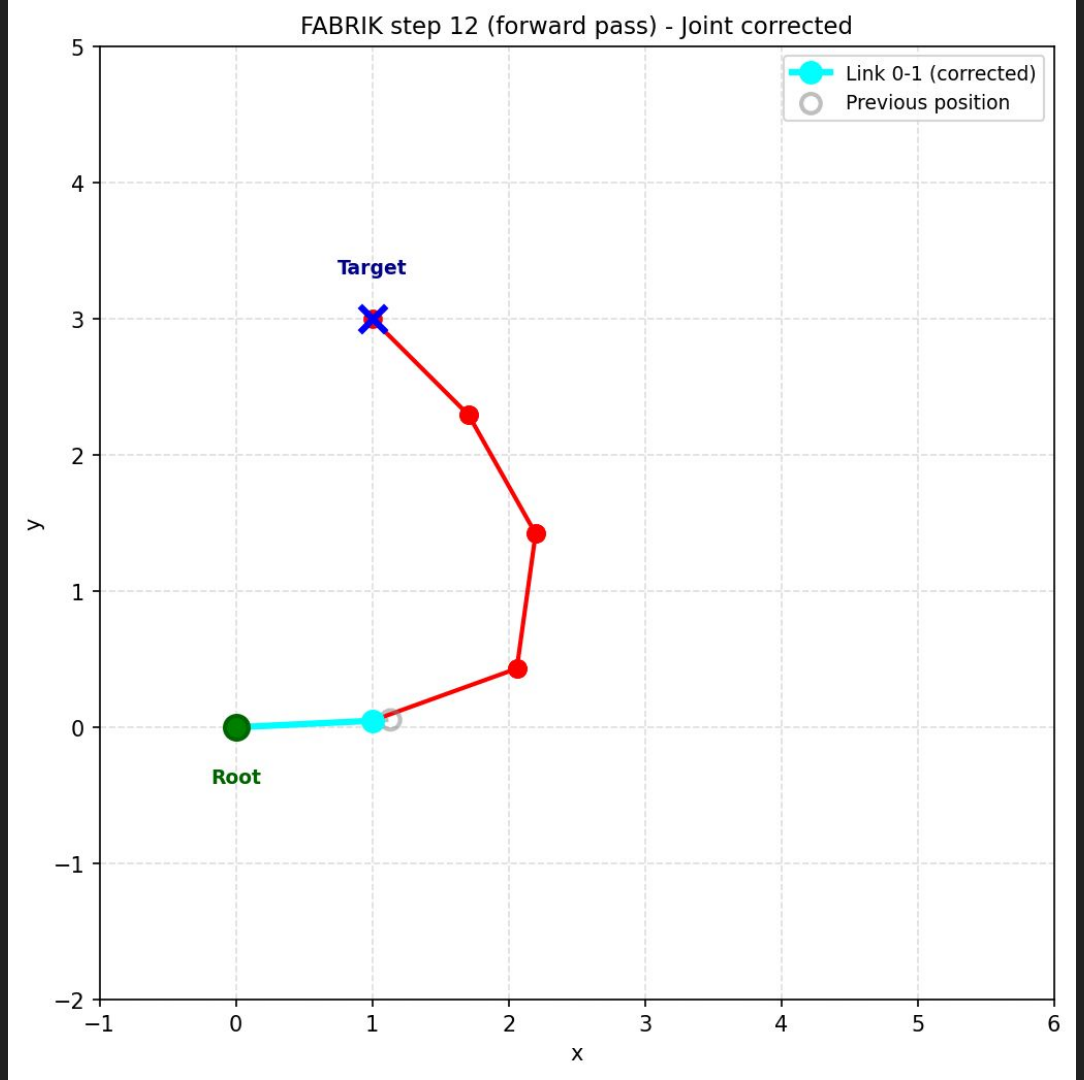

FABRIK step 7 (backward pass) - Link stretched/compressed
Link 1-2 (wrong length)
Target
Current: 1.14
Expected: 1.00
Root
x
y
5
4
3
2
1
0
−1
−2
−1
0
1
2
3
4
5
6


FABRIK step 8 (backward pass) - Joint corrected
Link 1-2 (corrected)
Previous position
Target
Root
x
y
5
4
3
2
1
0
−1
−2
−1
0
1
2
3
4
5
6


FABRIK step 9 (backward pass) - Link stretched/compressed
Link 0-1 (wrong length)
Target
Current: 1.14
Expected: 1.00
Root
x
y
5
4
3
2
1
0
−1
−2
−1
0
1
2
3
4
5
6


FABRIK step 10 (backward pass) - Joint corrected
Link 0-1 (corrected)
Previous position
Target
Root
x
y
5
4
3
2
1
0
−1
−2
−1
0
1
2
3
4
5
6


FABRIK step 11 (forward pass) - Link stretched/compressed
Link 0-1 (wrong length)
Target
Current: 1.14
Expected: 1.00
Root
x
y
5
4
3
2
1
0
−1
−2
−1
0
1
2
3
4
5
6


FABRIK step 12 (forward pass) - Joint corrected
Link 0-1 (corrected)
Previous position
Target
Root
x
y
5
4
3
2
1
0
−1
−2
−1
0
1
2
3
4
5
6

FABRIK

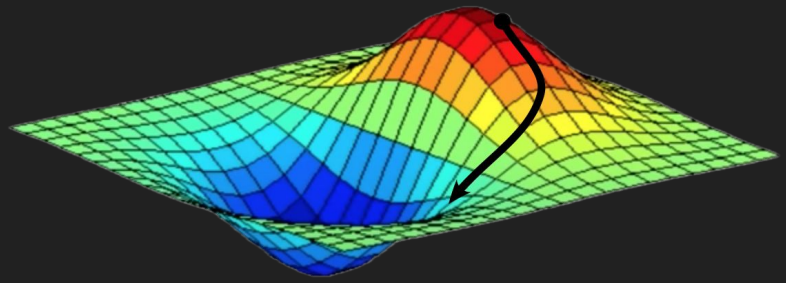
```
repeat until converged:
  # Backward pass
  joints[n-1] = target
  for i from n-2 down to 0:
      r = distance(joints[i+1], joints[i])
      t = lengths[i] / r
      joints[i] = (1-t)*joints[i+1] + t*joints[i]

  # Forward pass
  joints[0] = base
  for i from 1 to n-1:
      r = distance(joints[i], joints[i-1])
      t = lengths[i-1] / r
      joints[i] = (1-t)*joints[i-1] + t*joints[i]
```

bevy_fabrik

DIY

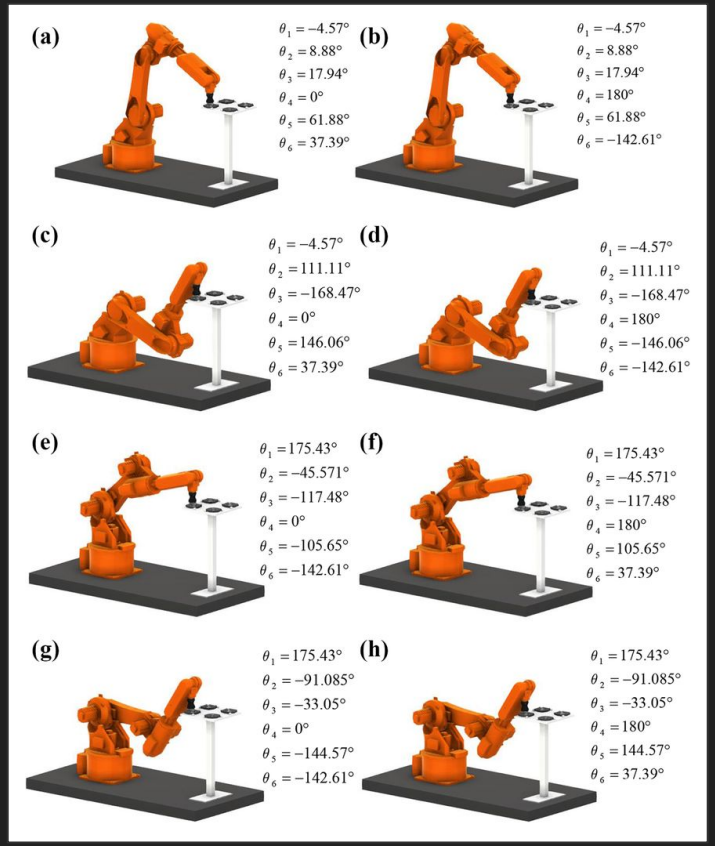
Jacobians

$$\begin{bmatrix} \Delta X \\ \Delta Y \\ \Delta Z \\ \Delta q_i \\ \Delta q_j \\ \Delta q_k \end{bmatrix} = \begin{bmatrix} \frac{\partial X}{\partial \theta_1} & \frac{\partial X}{\partial \theta_2} & \frac{\partial X}{\partial \theta_3} & \frac{\partial X}{\partial \theta_4} & \frac{\partial X}{\partial \theta_5} & \frac{\partial X}{\partial \theta_6} \\ \frac{\partial Y}{\partial \theta_1} & \frac{\partial Y}{\partial \theta_2} & \frac{\partial Y}{\partial \theta_3} & \frac{\partial Y}{\partial \theta_4} & \frac{\partial Y}{\partial \theta_5} & \frac{\partial Y}{\partial \theta_6} \\ \frac{\partial Z}{\partial \theta_1} & \frac{\partial Z}{\partial \theta_2} & \frac{\partial Z}{\partial \theta_3} & \frac{\partial Z}{\partial \theta_4} & \frac{\partial Z}{\partial \theta_5} & \frac{\partial Z}{\partial \theta_6} \\ \frac{\partial q_i}{\partial \theta_1} & \frac{\partial q_i}{\partial \theta_2} & \frac{\partial q_i}{\partial \theta_3} & \frac{\partial q_i}{\partial \theta_4} & \frac{\partial q_i}{\partial \theta_5} & \frac{\partial q_i}{\partial \theta_6} \\ \frac{\partial q_j}{\partial \theta_1} & \frac{\partial q_j}{\partial \theta_2} & \frac{\partial q_j}{\partial \theta_3} & \frac{\partial q_j}{\partial \theta_4} & \frac{\partial q_j}{\partial \theta_5} & \frac{\partial q_j}{\partial \theta_6} \\ \frac{\partial q_k}{\partial \theta_1} & \frac{\partial q_k}{\partial \theta_2} & \frac{\partial q_k}{\partial \theta_3} & \frac{\partial q_k}{\partial \theta_4} & \frac{\partial q_k}{\partial \theta_5} & \frac{\partial q_k}{\partial \theta_6} \end{bmatrix} \cdot \begin{bmatrix} \theta_1 \\ \theta_2 \\ \theta_3 \\ \theta_4 \\ \theta_5 \\ \theta_6 \end{bmatrix}$$

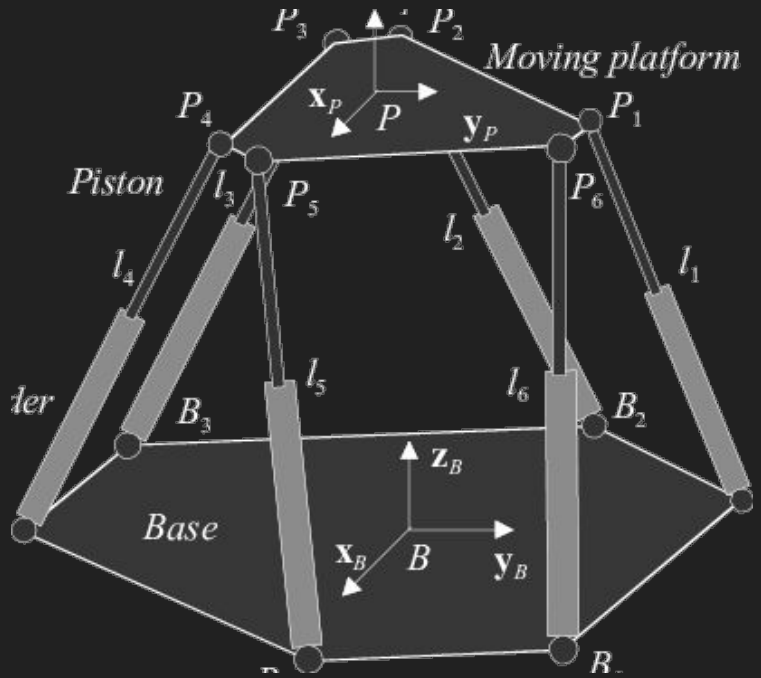


k

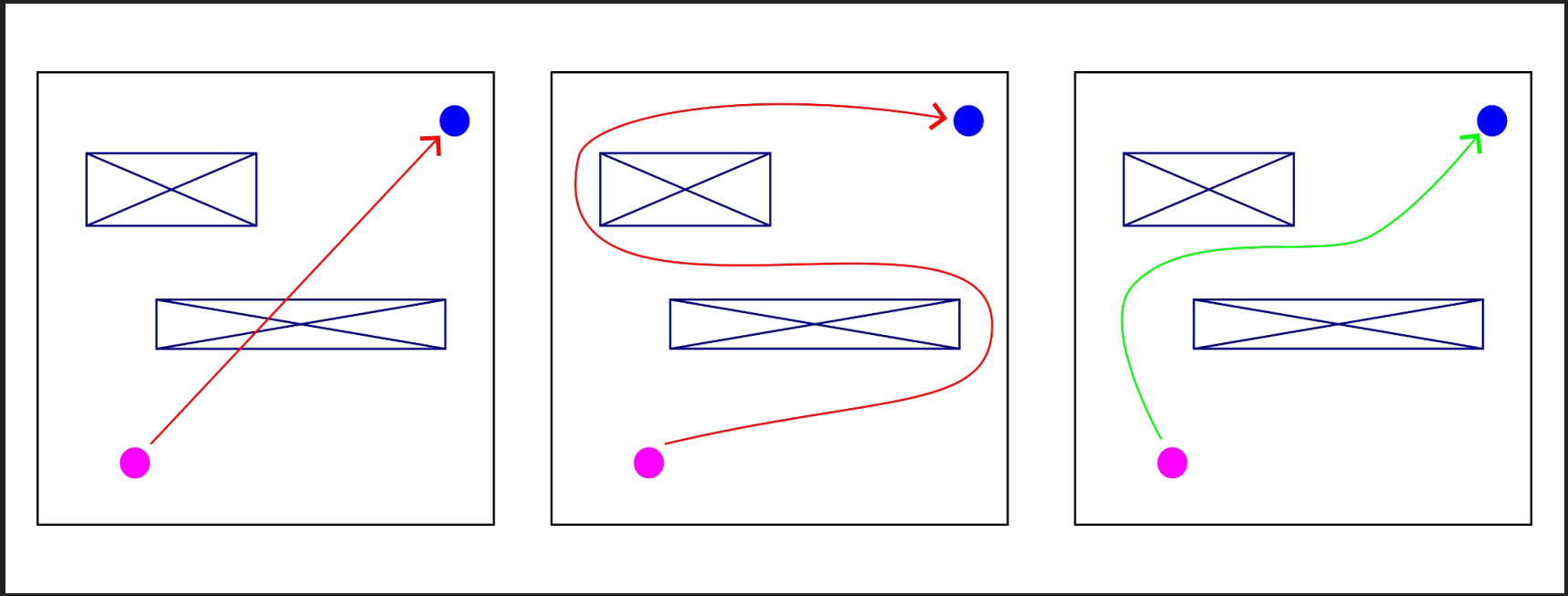
Many solutions



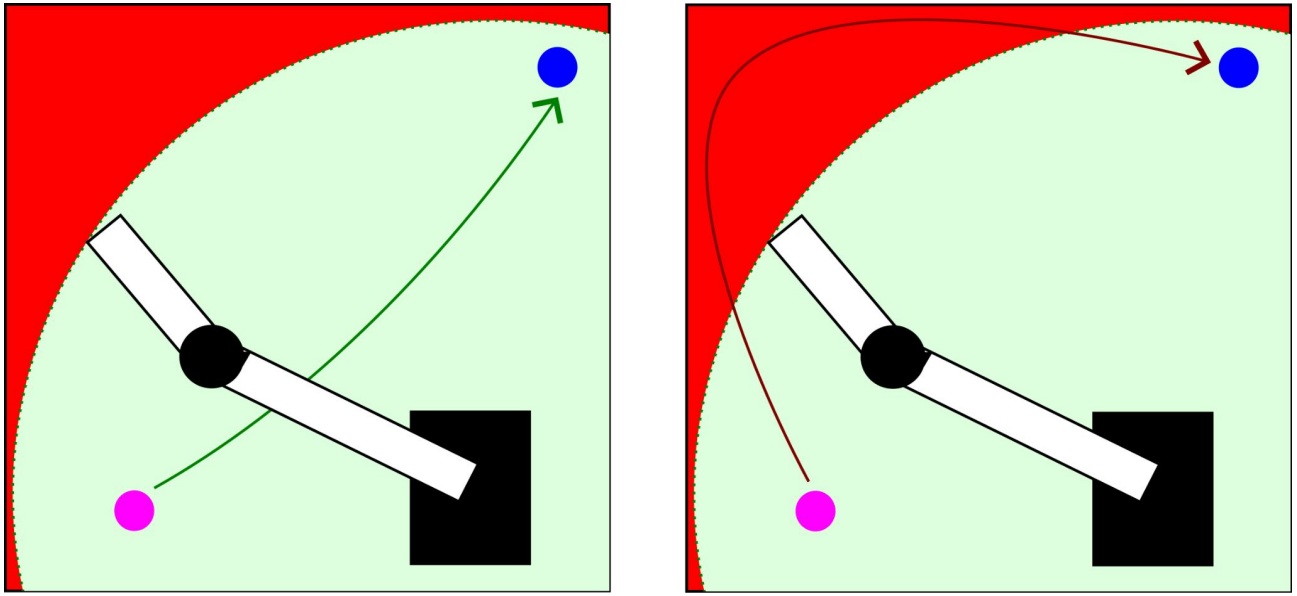
More complex cases



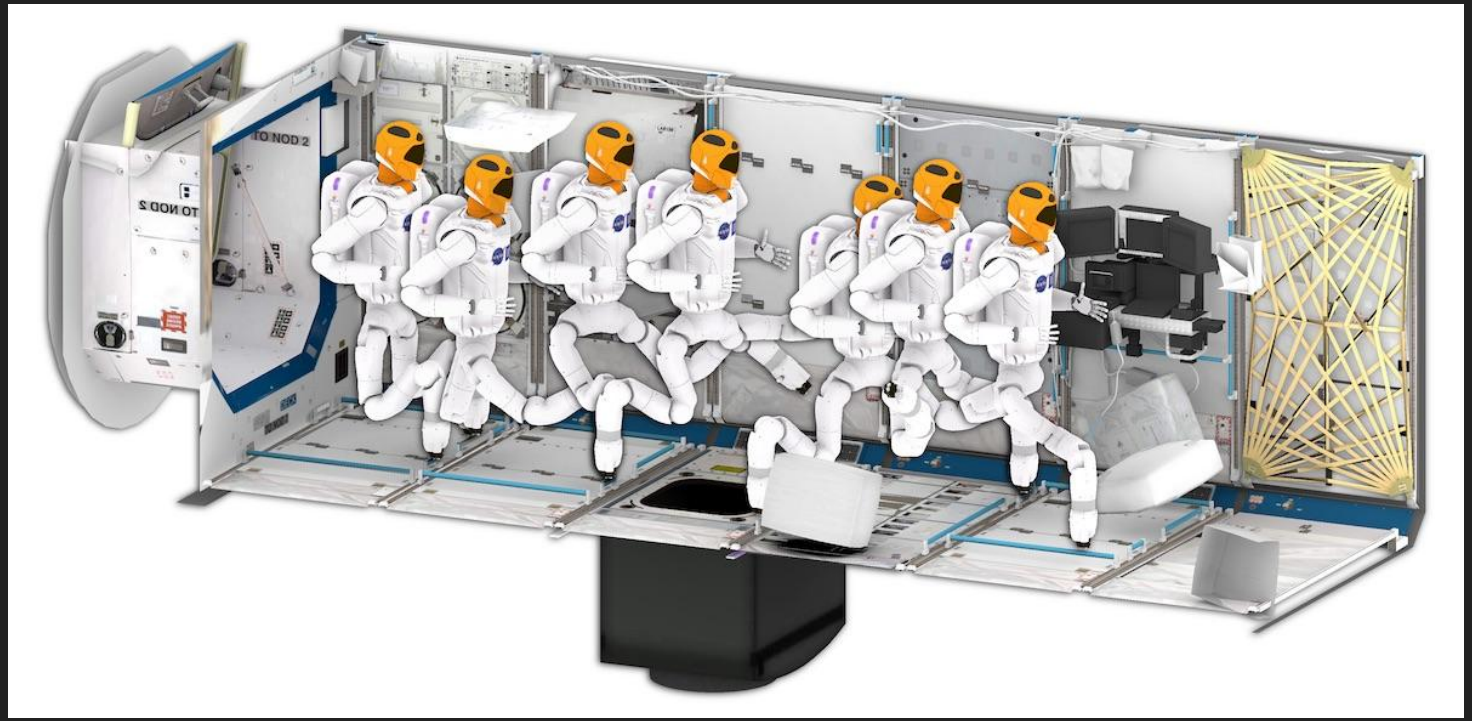
Pathfinding



Pathfinding has constraints



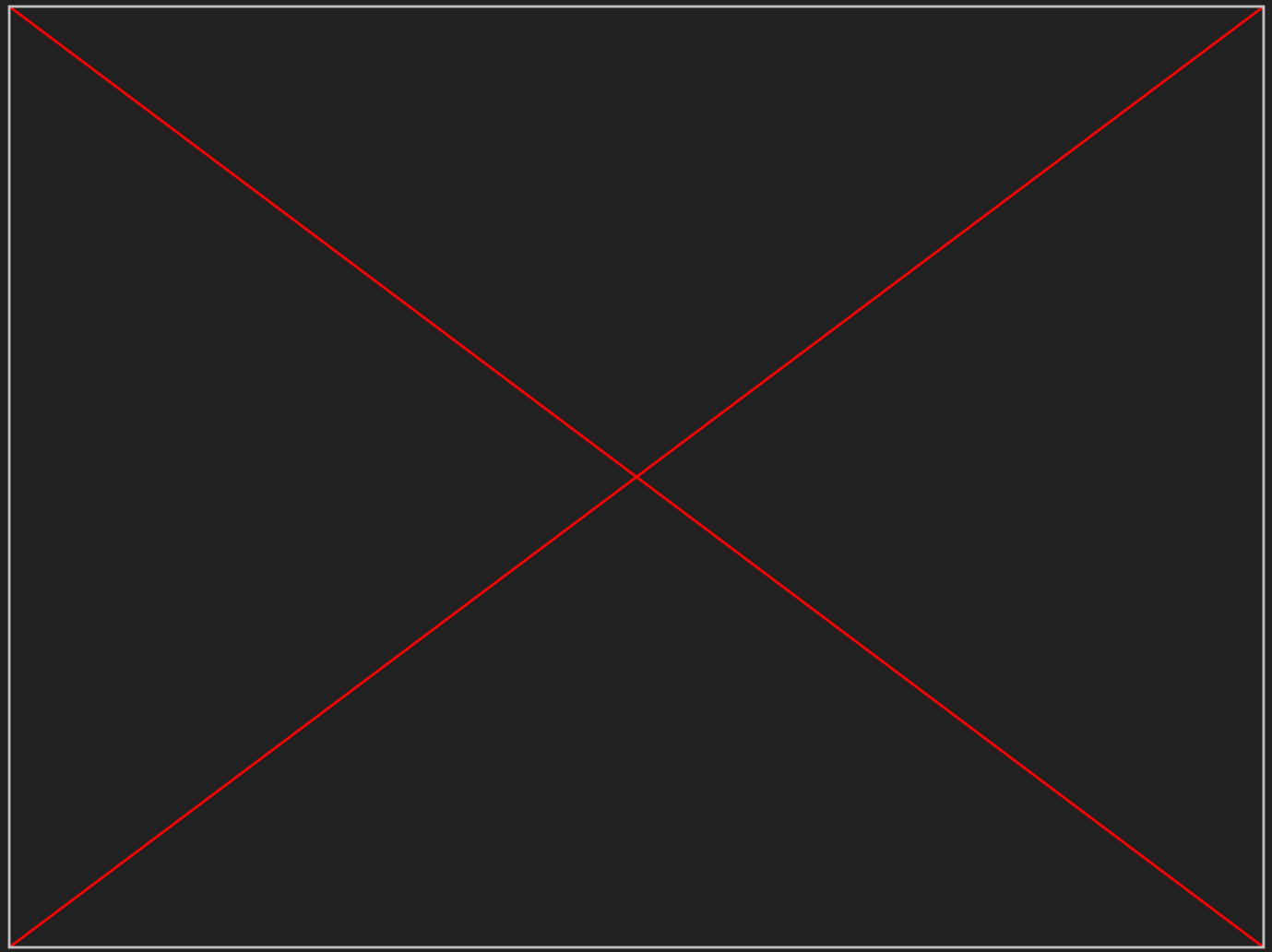
Pathfinding can be really really complicated



Pathfinding can be really really complicated

PRM
LazyPRM
PRM*
LazyPRM*
SPARS
SPARS2
RRT
RRTConnect
RRT*
RRT#
RRTX
LBTRRT
SST
T-RRT
VF-RRT
pRRT
LazyRRT
TSRRT
EST
SBL
pSBL
KPIECE
BKPIECE
LBKPIECE
STRIDE
PDST
FMT*
BFMT*
Informed RRT*
BIT*
ABIT*
AIT*
EIT*
ST-RRT*
CForest
AnytimePathShortening (APS)
RRT (control)
SST (control)
EST (control)
KPIECE (control)
PDST (control)
Syclop
SyclopRRT
SyclopEST
LTLPlanner
QRRT
QRRT*
QMP
QMP*
SPQR

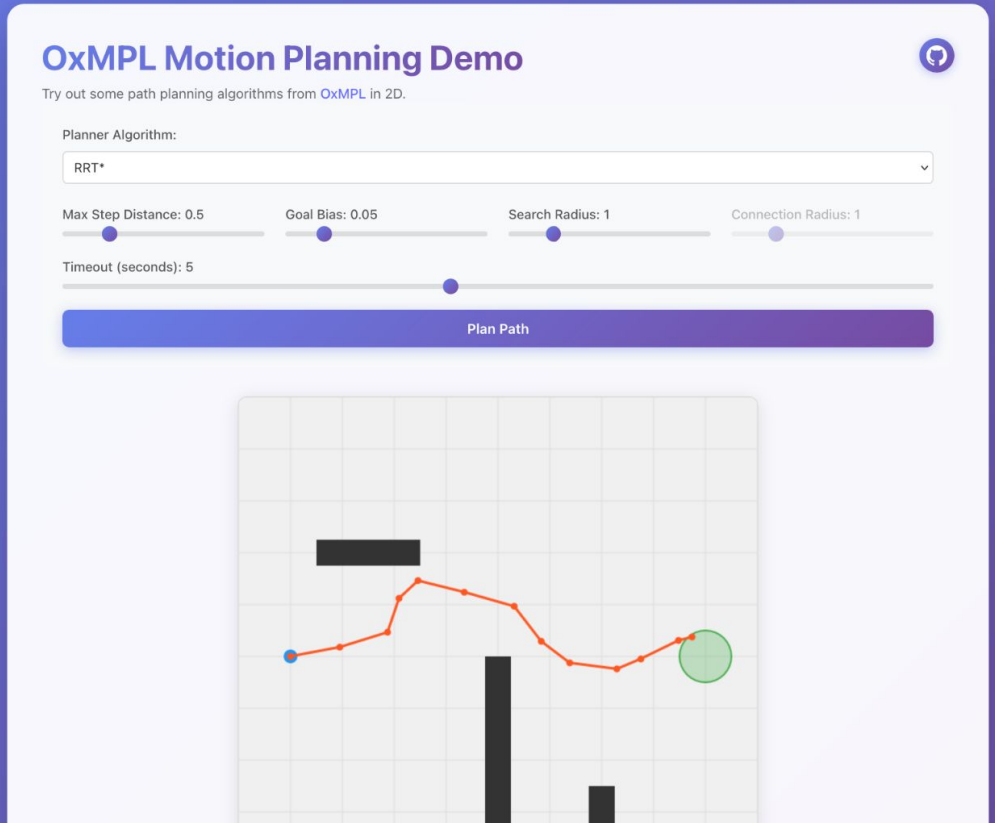
A simple approach: Rapidly Exploring Random Trees (RRT)



Path planning with Rust

 oxmpl

Try it out: **rossng.eu/oxmpl-js-demo/**



What's left?

- **Calibration**
  - Your kinematics model is a white lie - assembly errors; flex etc.
- **Motion control**
  - Motors can only accelerate and decelerate at certain speeds - how do you make the end effector actually move along the desired path?
- **Vision**
  - Where is the robot in the world? Where is the thing it needs to reach?
- and much more…