

Cobotta Simulation using Mujoco on Genesis

Aug. 19, 2026

物理エンジンに興味を持った人は、PC 上で Cobotta に自分で設計したハンドを持たせて動かしてみよう。

Genesis：物理エンジンのひとつ。流体，弾性体，剛体，様々な力学現象をシミュレートできる。
<https://genesis-world.readthedocs.io/en/latest/>

MuJoCo (Multi-Joint dynamics with Contact)：多関節の剛体力学が得意な物理エンジン (or ライブラリ)。ここでは Genesis から呼び出されている。
<https://rikei-tawamure.com/entry/2025/05/24/211341>

1. インストール

以下に，ubuntu 上にインストールする方法を示す。
まず，apt を update して python3 他をインストールする。

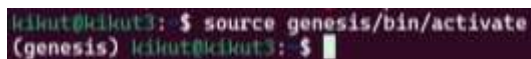
```
> sudo apt update
> sudo apt install python3
> sudo apt install python3.12-venv
> sudo apt-get install python3-tk
```

適当なディレクトリ（ここでは genesis/）を作成してそこに Genesis 用の仮想環境(genesis ディレクトリ)を構築する。

```
> python3 -m venv genesis
```

その後，genesis をアクティベートする。

```
> source genesis/bin/activate
```



```
kikut@kikut3: $ source genesis/bin/activate
(genesi) kikut@kikut3: $
```

まず，pip を upgrade する。

```
> pip install --upgrade pip
```

PyTorch (<https://pytorch.org/get-started/locally/>) をインストールする（ディープラーニング（深層学

習) や機械学習 の研究・開発で使われる オープンソースの Python ライブラリ). 以下を選択して”Run this command”をコピーして実行する.

NOTE: Latest Stable PyTorch requires Python 3.10 or later.

PyTorch Build	Stable (2.13.0)			Preview (Nightly)	
Your OS	Linux		Mac		Windows
Package	Pip		LibTorch		Source
Language	Python			C++ / Java	
Compute Platform	CUDA 12.6	CUDA 13.0	CUDA 13.2	ROCm 7.2	CPU
Run this Command:	<pre>pip3 install torch torchvision --index-url https://download.pytorch.org/whl/cpu</pre>				

> pip3 install torch torchvision --index-url https://download.pytorch.org/whl/cpu

次に, genesis 本体をインストールする. 19 Aug. 2026 現在の version は 1.3.3

> pip install genesis-world

MuJoCo をインストールする.

> pip install mujoco-py genesis

動画用ライブラリをインストールする.

> pip install imageio imageio-ffmpeg

> pip install opencv-python

2. 実行

以下が存在するディレクトリに移動して実行する.

CAD/***.obj

/***.stl

cobotta.py

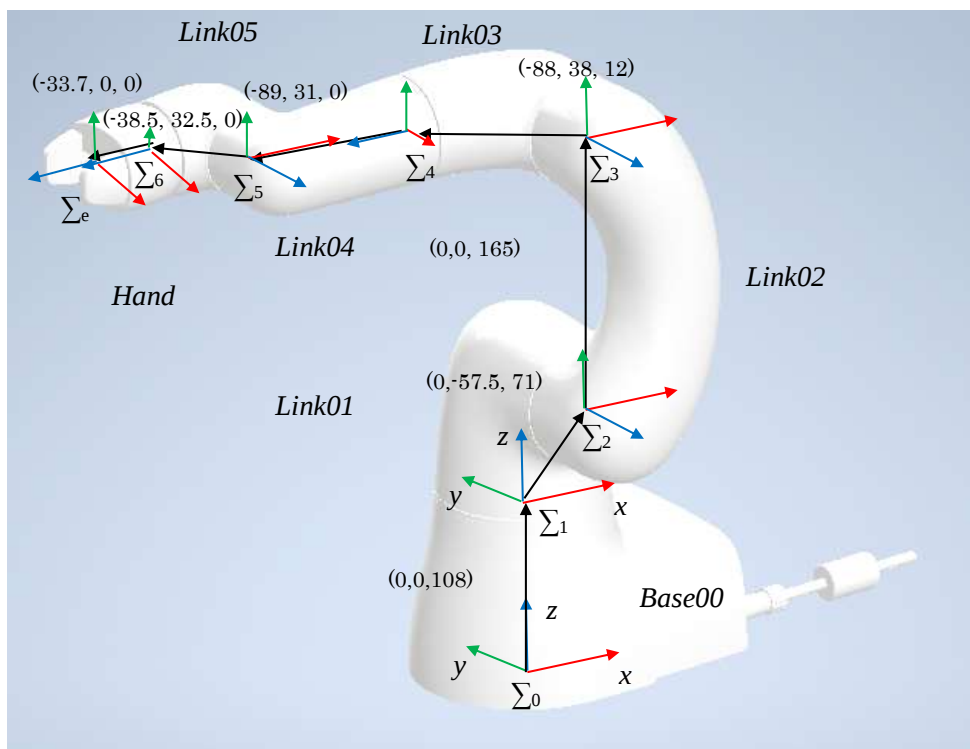
cobotta.xml

> python3 cobotta.py

以下のように表示される.

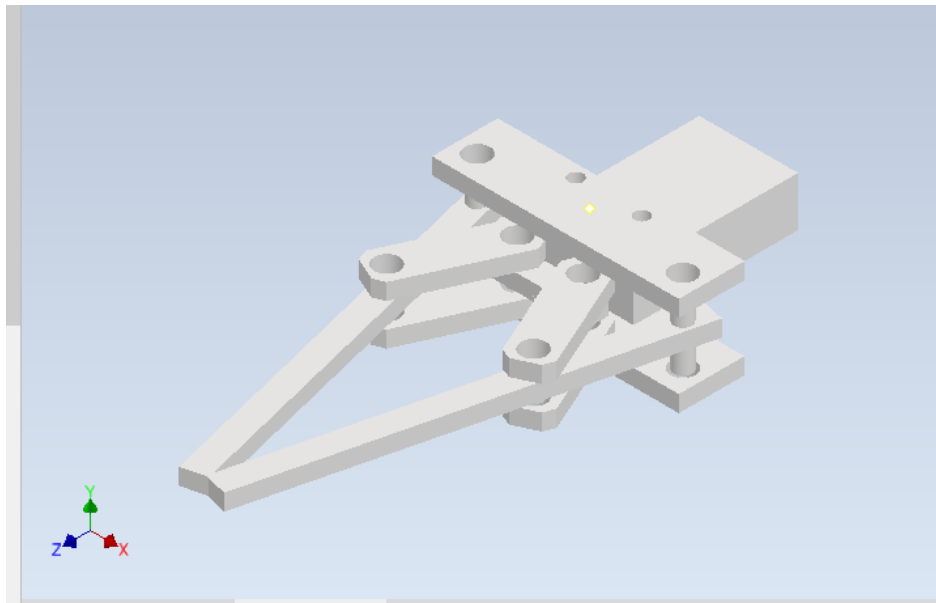


3. Cobotta の座標系, および関節の定義



4. Cobotta に自分のハンドを持たせてみる。

オリジナルハンドのアセンブリファイルの原点と座標系を確認し、obj ファイルと stl ファイルを保存する。重心位置、慣性テンソルは iProperty で確認できる。



更新(U)

密度(D) 要求される精度(Y) クリップボードにコピー(O)

1.000 g/cm³ 低

一般的なプロパティ

☐ ビード線を含む(B) ☐ 数量のオーバーライドを含める(Q)

重心

質量(S) 27.664 g (相対誤差 =  X 0.000 mm (相対誤差 = 

面積(R) 19303.199 mm² (相対誤差 =  Y -12.000 mm (相対誤差 = 

体積(V) 27664.450 mm³ (相対誤差 =  Z 3.196 mm (相対誤差 = 

慣性プロパティ

慣性モーメント(P) グローバル(G) 重心(C)

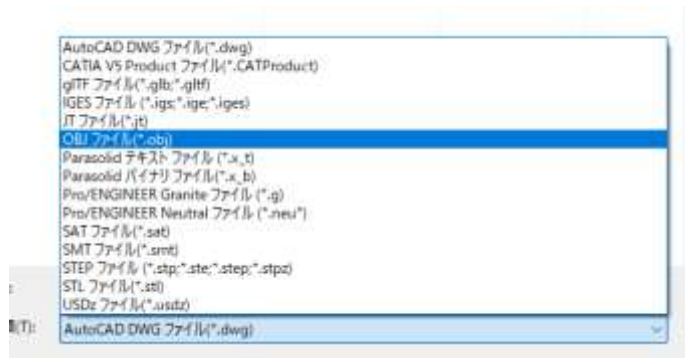
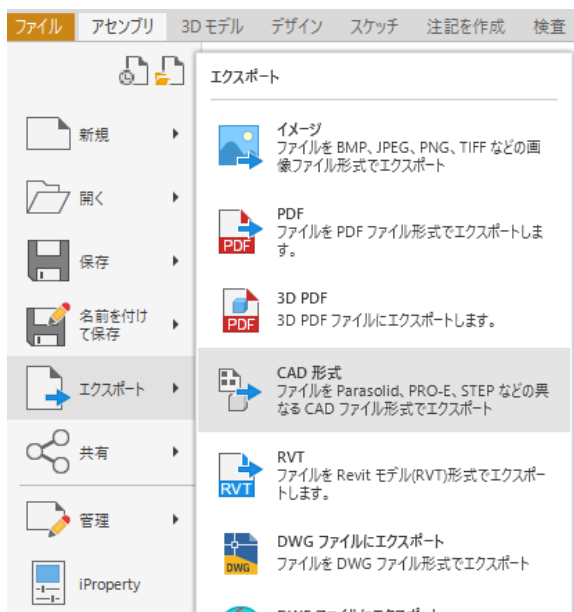
質量モーメント

Ixx 21730.699 g mm² 負の値を使って計算.

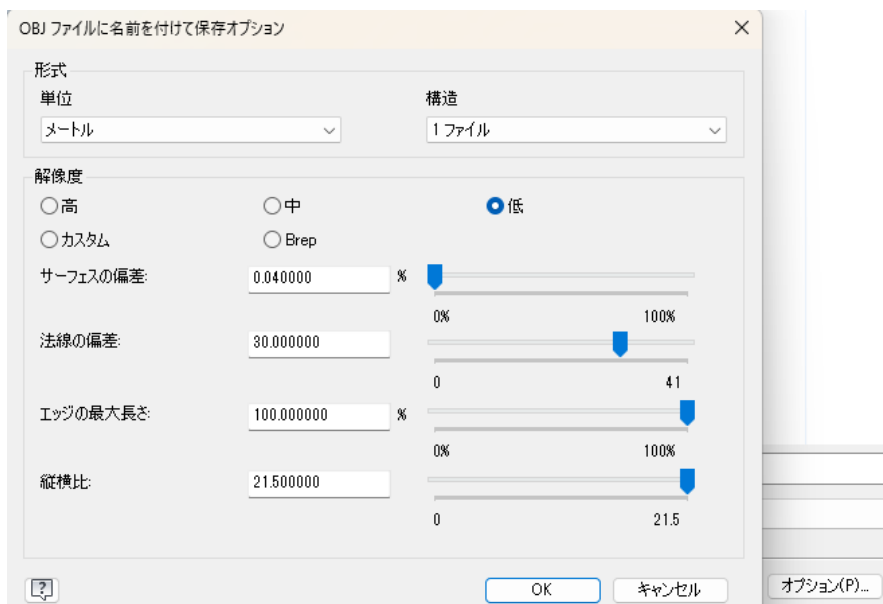
Ixy 0.000 g mm² (相対誤差 =  Iyy 25423.195 g mm² (相対誤差 = 

Ixz -0.000 g mm² (相対誤差 =  Iyz -0.000 g mm² (相対誤差 =  Izz 5637.846 g mm² (相対誤差 = 

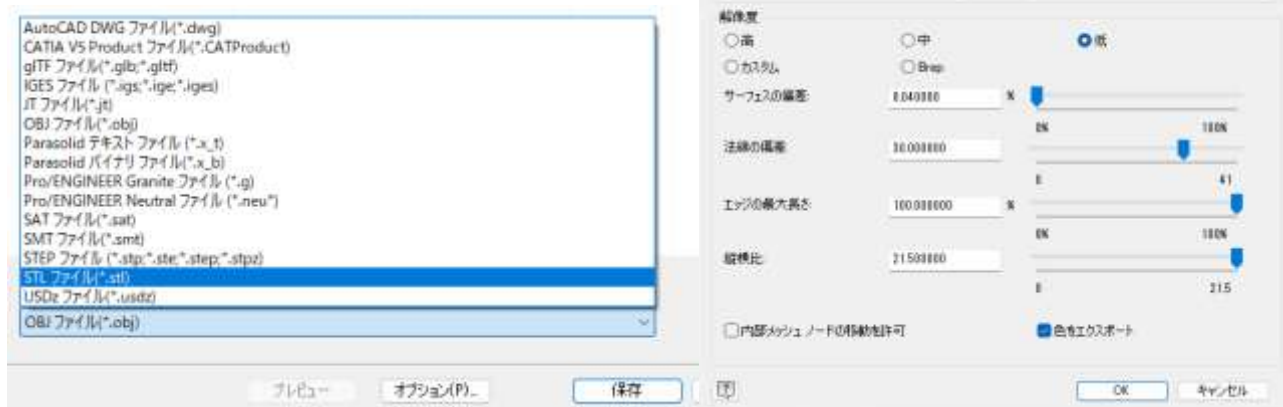
obj ファイルは、
ファイル→エクスポート→CAD 形式→OBJ ファイル (*.obj)
から保存できる。



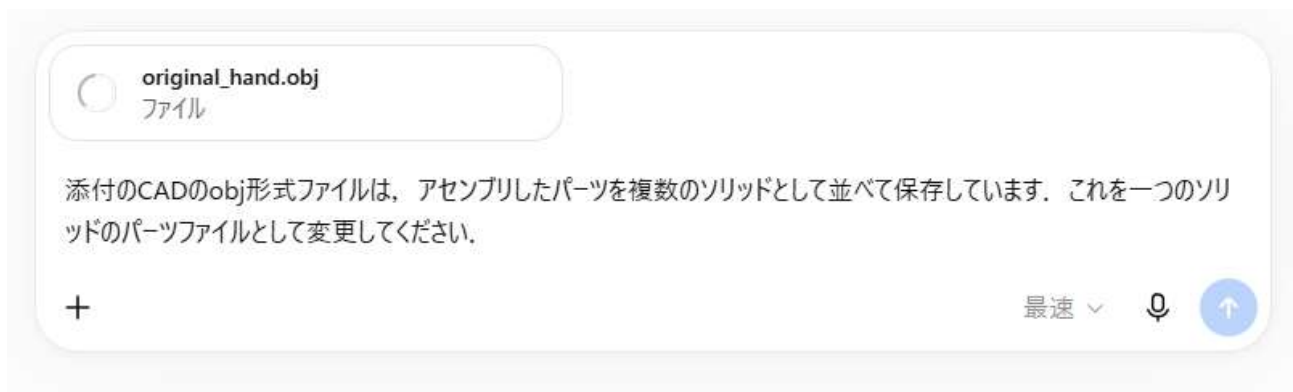
オプションで単位はメートルにしておく．ファイル名は何でもよいが，ここでは，"original_hand.obj"にしておく．



同様に stl ファイルも "original_hand.stl" として保存する．こちらもオプションで，「メートル」を選択しておく．形式は「バイナリ」にする（どちらでも読み込める）．



なお、Inventor はアセンブリファイルの場合、obj ファイルは各パーツをソリッドとして別々に保存する。これを一つのソリッドの obj ファイルに変更する。例えば、chatGPT に以下のようにして返還してもらうことができる。



結論: 添付OBJを、23個に分かれていた **g** (ソリッド) を統合し、1つのパーツ／1つのOBJオブジェクトとして扱えるファイルに変更しました。

[統合済みOBJをダウンロード](#)

Confidence: High — 元の頂点・面形状は変更せず、23個のグループ定義を1つの **o** / **g** に統合しています。

“hand_single_part.obj”として保存されたので、これを”original_hand.obj”として再保存しておく。

二つのファイルを CAD のディレクトリにコピーしておく。

CAD/original_hand.obj

/original_hand.stl

・cobotta.xml にこれらを剛体として登録する。関節自由度を持たせることもできる。

以下，朱書の部分を追加する．

```
<asset>
  <material class="cobotta" name="white" rgba="1 1 1 1"/>
  <material class="cobotta" name="off_white" rgba="0.901961 0.921569 0.929412 1"/>
  <material class="cobotta" name="black" rgba="0.25 0.25 0.25 1"/>

  <!-- Collision meshes -->
  <mesh name="base00_c" file="CAD/base00.stl"/>
  <mesh name="link01_c" file="CAD/link01.stl"/>
  <mesh name="link02_c" file="CAD/link02.stl"/>
  <mesh name="link03_c" file="CAD/link03.stl"/>
  <mesh name="link04_c" file="CAD/link04.stl"/>
  <mesh name="link05_c" file="CAD/link05.stl"/>
  <mesh name="hand_c" file="CAD/hand06.stl"/>
  <mesh name="finger00_c" file="CAD/finger00.stl"/>
  <mesh name="finger01_c" file="CAD/finger01.stl"/>
  <mesh name="original_hand_c" file="CAD/original_hand.stl"/>

  <!-- Visual meshes -->
  <mesh file="CAD/base00.obj"/>
  <mesh file="CAD/link01.obj"/>
  <mesh file="CAD/link02.obj"/>
  <mesh file="CAD/link03.obj"/>
  <mesh file="CAD/link04.obj"/>
  <mesh file="CAD/link05.obj"/>
  <mesh file="CAD/hand06.obj"/>
  <mesh file="CAD/finger00_nogroup.obj"/>
  <mesh file="CAD/finger01_nogroup.obj"/>
  <mesh file="CAD/original_hand.obj"/>
</asset>

  <body name="hand06" pos="-0.0385 0 -0.0325" euler="0.0 -1.570796 0">
    <inertial mass="0.05" pos="-0.016 0.0 0" fullinertia="0.000025 0.000038
0.000029 0.0 0.0 0.0"/>
    <joint name="joint5" type="hinge" pos="0 0 0" axis="0 0 1" limited="true"/>
    <geom mesh="hand06" material="off_white" class="visual"/>
    <geom mesh="hand_c" class="collision"/>
    <site name="Sig6" pos="0 0 0"/>

    <body name="finger00" pos="-0.016 0 0.0337"> # ハンドの開き具合を調整
      <inertial mass="0.05" pos="-0.002 0 0.0089" fullinertia="0.000001 0.000001
0.000001 0.0 0.0 0.0"/>
      <joint name="joint6" type="slide" pos="0 0 0" axis="1 0 0" limited="true"/>
      <geom mesh="finger00_nogroup" material="off_white" class="visual"/>
      <geom mesh="finger00_c" class="collision"/>
    </body>
    <body name="finger01" pos="0.016 0 0.0337">
      <inertial mass="0.05" pos="0.002 0 -0.0089" fullinertia="0.000001 0.000001
0.000001 0.0 0.0 0.0"/>
      <joint name="joint7" type="slide" pos="0 0 0" axis="-1 0 0" limited="true"/>
      <geom mesh="finger01_nogroup" material="off_white" class="visual"/>
      <geom mesh="finger01_c" class="collision"/>
    </body>

    <body name="original_hand" pos="0.0 0 0.06"> # 設置位置
      <inertial mass="0.0276" pos="0.0 -0.012 0.0032" fullinertia="0.0000217.
0.00002542 0.000005638 0.0 0.0 0.0"/> # 質量, 重心, 慣性テンソル (Ixx, Iyy, Izz, Ixy, Ixz, Iyz)
      <geom mesh="original_hand" material="off_white" class="visual"/>
```

```
<geom mesh="original_hand_c" class="collision"/>
</body>
```

```
<site name="ee" pos="0 0 0.0337"/>
</body>
```

・ cobotta.py の位置制御を書き換え、指先を停止させる。

```
for i in range(Iteration):
    t = i*DT
    cobotta.set_dofs_position(np.array([180/180*np.pi*np.sin(2*np.pi*Freq*0.5*t),
                                        45/180*np.pi*np.sin(2*np.pi*Freq*t),
                                        30/180*np.pi*np.sin(2*np.pi*Freq*t),
                                        30/180*np.pi*np.sin(2*np.pi*Freq*t),
                                        30/180*np.pi*np.sin(2*np.pi*Freq*t),
                                        30/180*np.pi*np.sin(2*np.pi*Freq*t),
                                        0.0, # 0.01*np.sin(2*np.pi*Freq*2.0*t),
                                        0.0, # 0.01*np.sin(2*np.pi*Freq*2.0*t),      #
                                        1/180*np.pi*i
                                        ]), dofs_idx)
    scene.step()
    if (i%EveryDraw)==0:
        cam.render() # 録画モードで内部的にフレーム保存

        # フレームを取得して動画に書き込み
        rgb_frame, _, _, _ = cam.render(depth=False, segmentation=False, normal=False)
        frame_bgr = cv2.cvtColor(rgb_frame, cv2.COLOR_RGB2BGR)
        video.write(frame_bgr)
```

実行すると、以下ようになる。

```
> python3 coibotta.py
```

ハンドの動きを制御するには、さらに関節の定義が必要になる。

指とハンドの衝突判定はここでは無視している。力制御なども可能である。



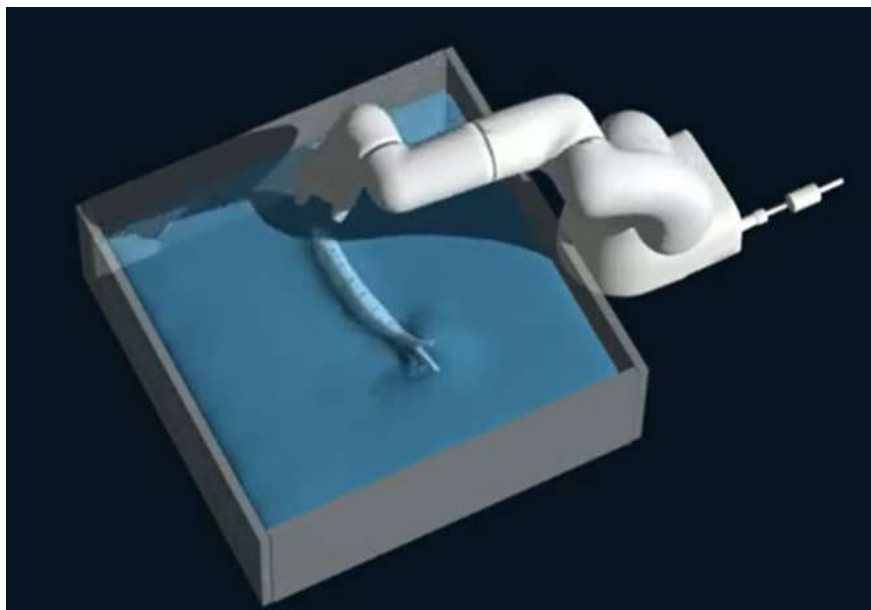
弾性体を切断したり，流体計算をしたりもできる。

布をひねる.



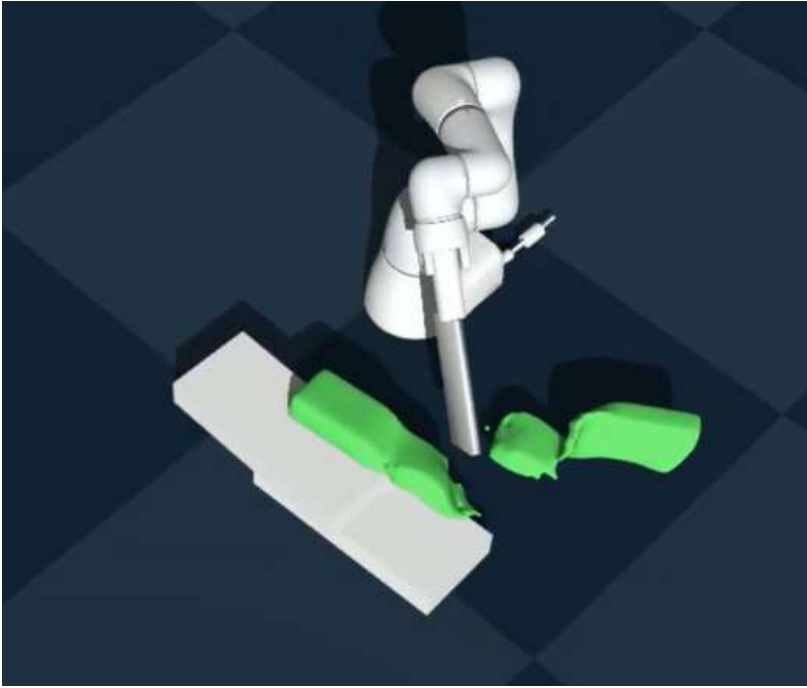
<https://youtu.be/0S1r6HorA84>

水槽の魚をつかむ.



<https://youtu.be/K0ve7XcrHGI>

切る



<https://youtu.be/j1n0QVG8WWI>