Modelación y Control con MuJoCo

– *Tutorial* –

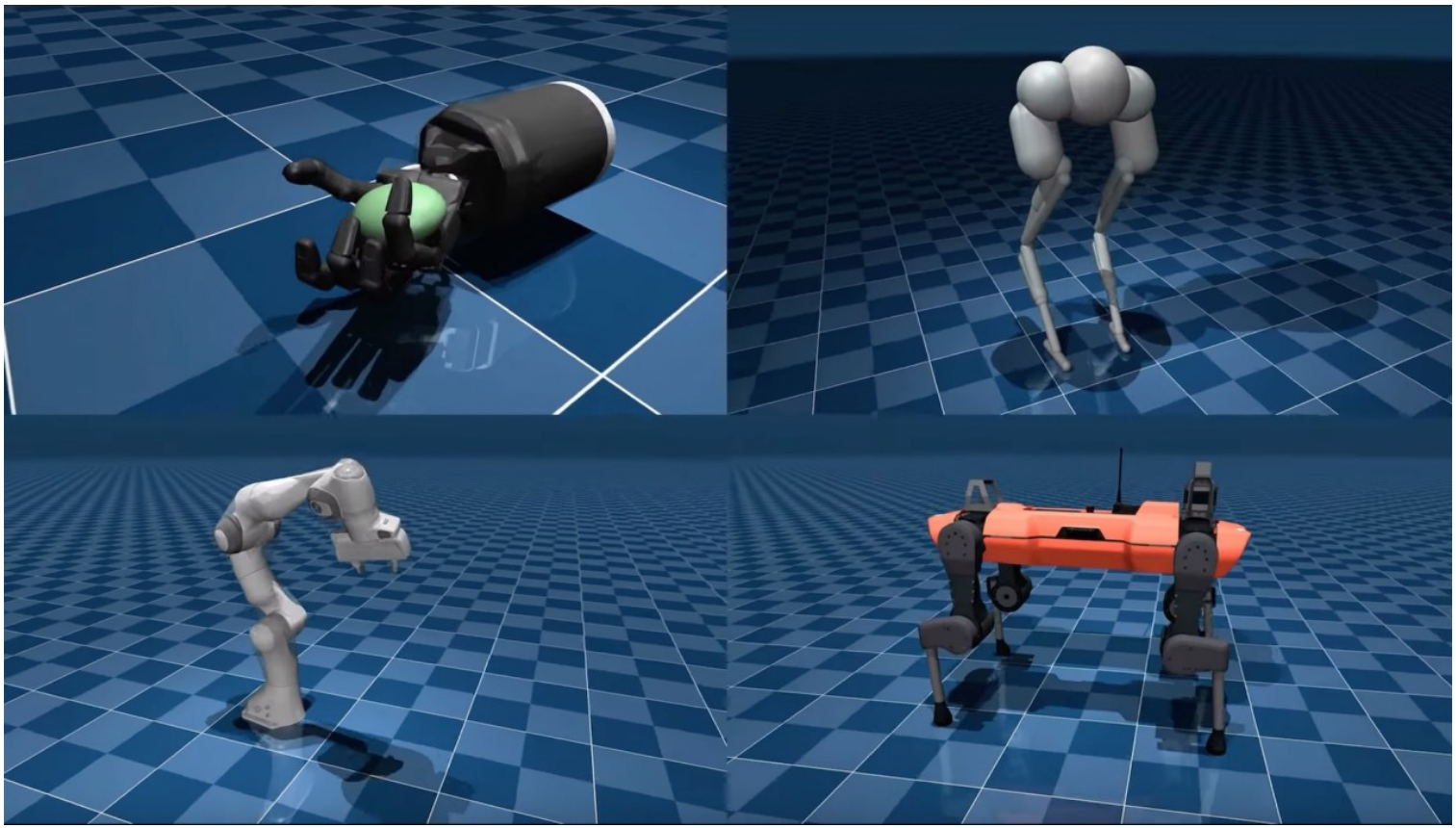
Miguel Torres-Torriti

Major de Ingeniería Robótica

IRB2001 Fundamentos de Robótica
– 2024 –

ROBOTICS AND AUTOMATION LABORATORY
Departamento de Ingeniería Eléctrica
PONTIFICIA UNIVERSIDAD CATÓLICA DE CHILE

MuJoCo: Multi-Joint dynamics with Contacts



MuJoCo: Multi-Joint dynamics with Contacts

Nociones preliminares:

- GE: "graphics engine" (motor gráfico) soportado por GPU.
  - → OpenGL o DirectX permite computar renderización/visualización 3D y acceder a GPU si está disponible.
- PE: "physics engine" (motor físico) algún día soportado por PPU.
- MuJoCo = **PE** + GE = simulador.
  - → PE=Spatial Vector Algebra/Articulated Body Algorithm (ABA) (Featherstone 1987).
  - → GE=renderizador basado en OpenGL.

Otros PEs

MuJoCo

Advanced physics simulation

Drake

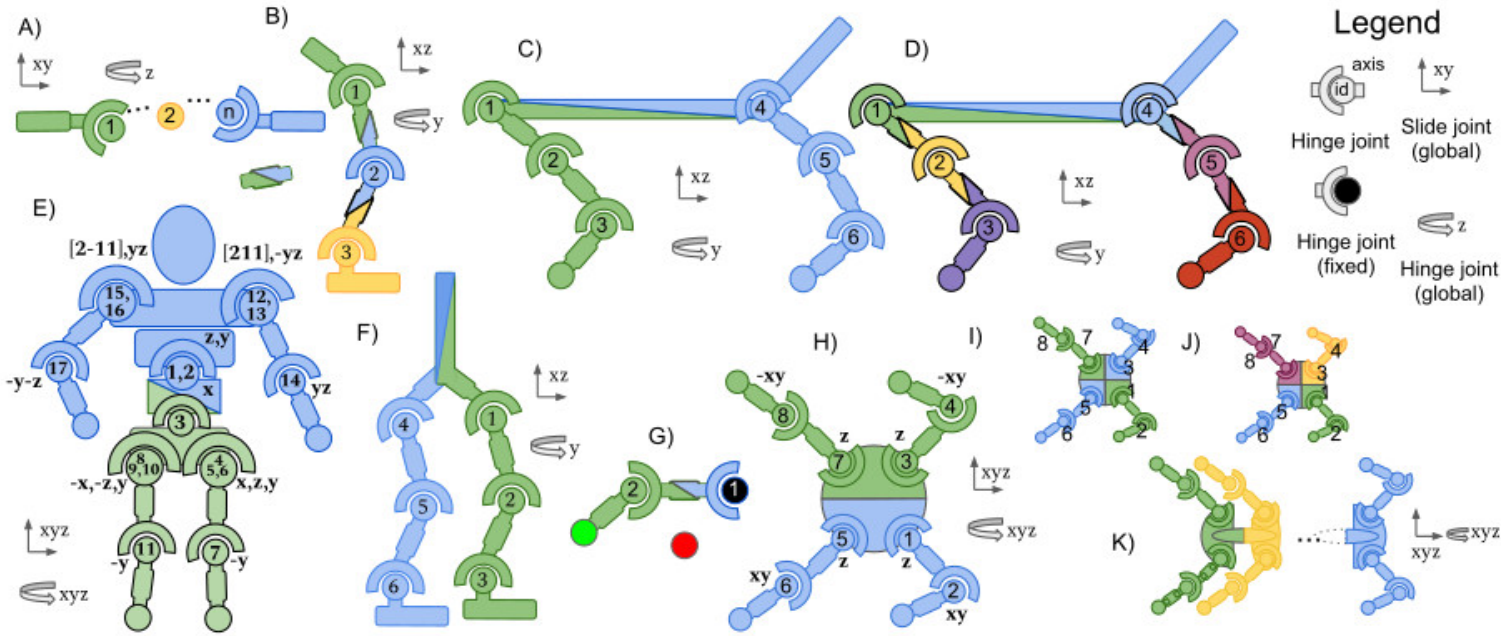
Otros PEs

- Drake es una alternativa a MuJoCo.
- Drake y MuJoCo lideran en "physical accuracy".
  - → Exactitud física en fuerzas de contacto, colisiones, dinámica y eficencia.
- Otros PEs menos realistas: Bullet, OpenODE, PhysX.
- Trade-off entre "Realismo Físico" vs. "Realismo Gráfico":
  - → Physical accuracy: dynamics for energy consumption, energy minimization, motion control, grasping.
  - → Graphical accuracy/computational efficiency: kinematics for computer games and motion planning.
- Entornos de simulación robótica+Interprocess Communication (Shared Memory/Pipes/Sockets):
  - → Player-Stage-Gazebo (PSG).
  - → V-Rep/CoppeliaSim.

Contenidos

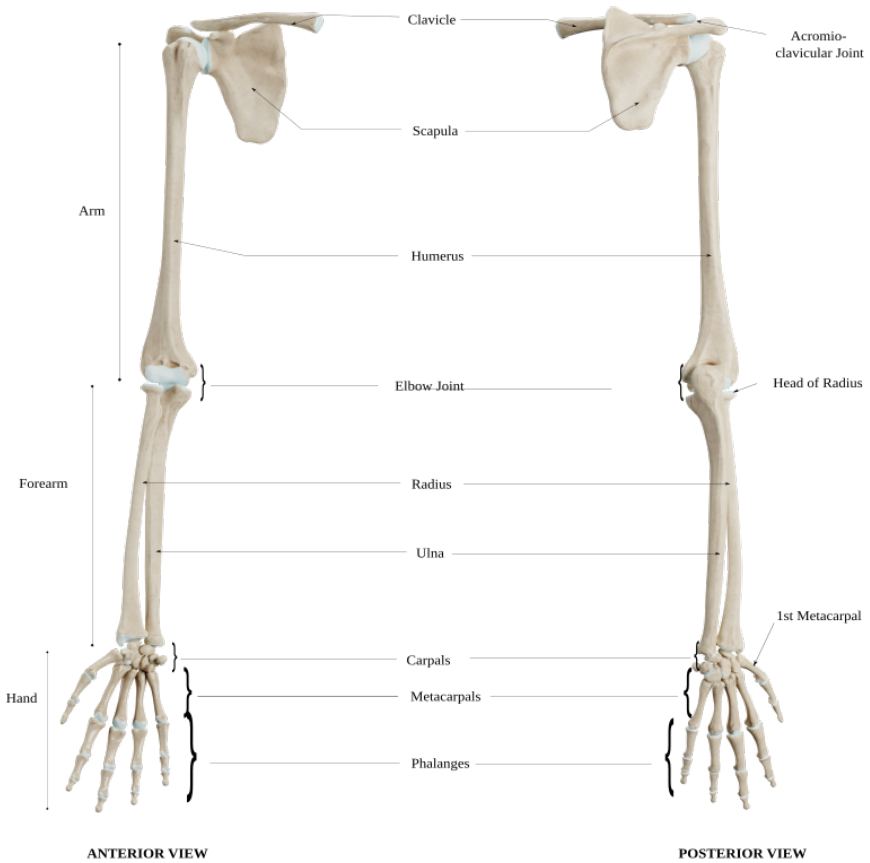
1. Modelación: Descripción del robot en archivo XML/URDF
2. Simulación: Loop
3. Gráficos: GLFW

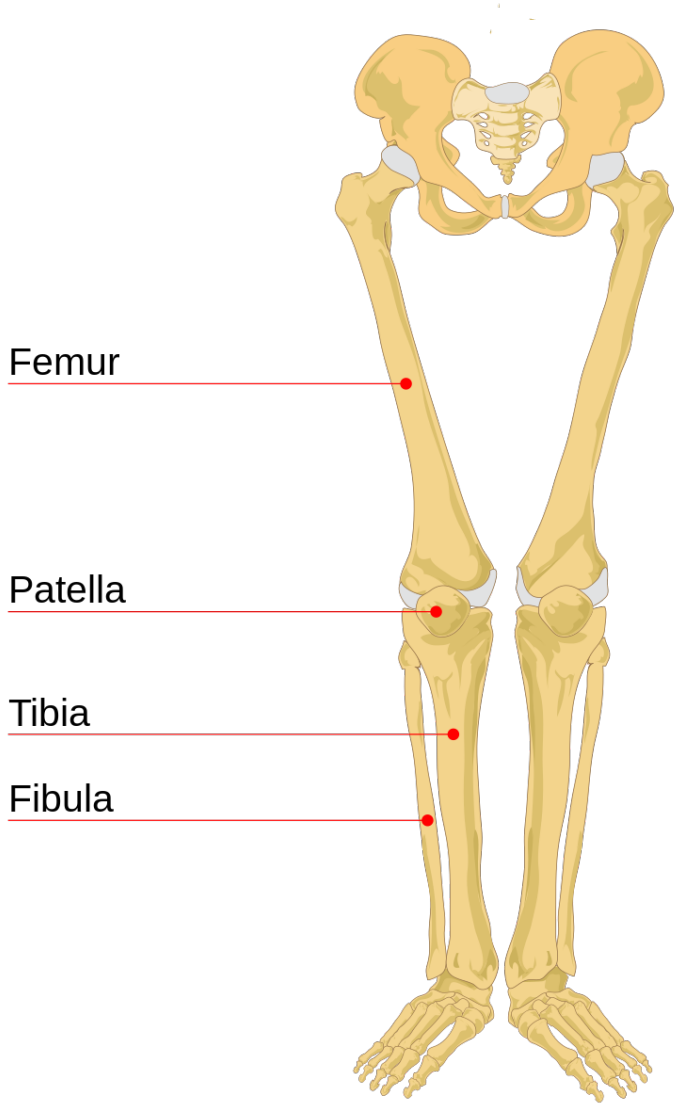
Descripción de un Sistema de Múltiples Cuerpos



https://robotics.farama.org/envs/MaMuJoCo/index.html

Humanoide


Clavicle
Acromio-
clavicular Joint
Scapula
Arm
Humerus
Elbow Joint
Head of Radius
Forearm
Radius
Ulna
1st Metacarpal
Carpals
Hand
Metacarpals
Phalanges
ANTERIOR VIEW
POSTERIOR VIEW


Femur
Patella
Tibia
Fibula

Humanoide

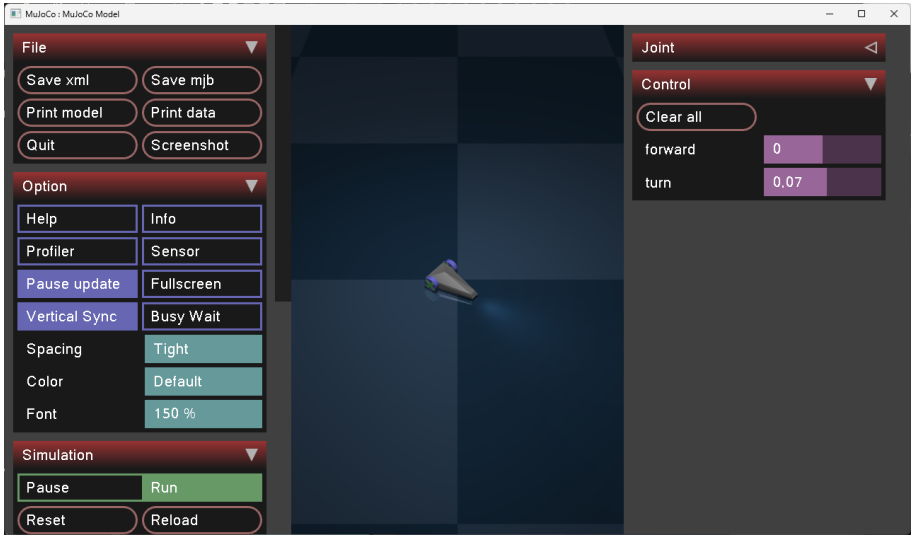
1 Body: Torso
- 1.1 Body: muslo-derecho; Joint: cadera-derecha
  - 1.1.1 Body: pierna-derecha; Joint: rodilla-derecha
    - 1.1.1.1 Body: pie-derecho; Joint: tobillo-derecho
- 1.2 Body: muslo-izquierdo; Joint: cadera-izquierda
  - 1.2.1 Body: pierna-izquierda; Joint: rodilla-izquierda
    - 1.2.1.1 Body: pie-izquierdo; Joint: tobillo-izquierdo
- 1.3 Body: cabeza; Joint: cuello
- 1.4 Body: brazo-derecho; Joint: hombro-derecho
  - 1.4.1 Body: antebrazo-derecho; Joint: codo-derecho
    - 1.4.1.1 Body: mano-derecha; Joint: muñeca-derecha
- 1.5 Body: brazo-izquierdo; Joint: hombro-izquierdo
  - 1.5.1 Body: antebrazo-izquierdo; Joint: codo-izquierdo
    - 1.4.1.1 Body: mano-izquierda; Joint: muñeca-izquierda

MuJoCo XML: (0) <mujoco> ... </mujoco>

Aspectos preliminares:

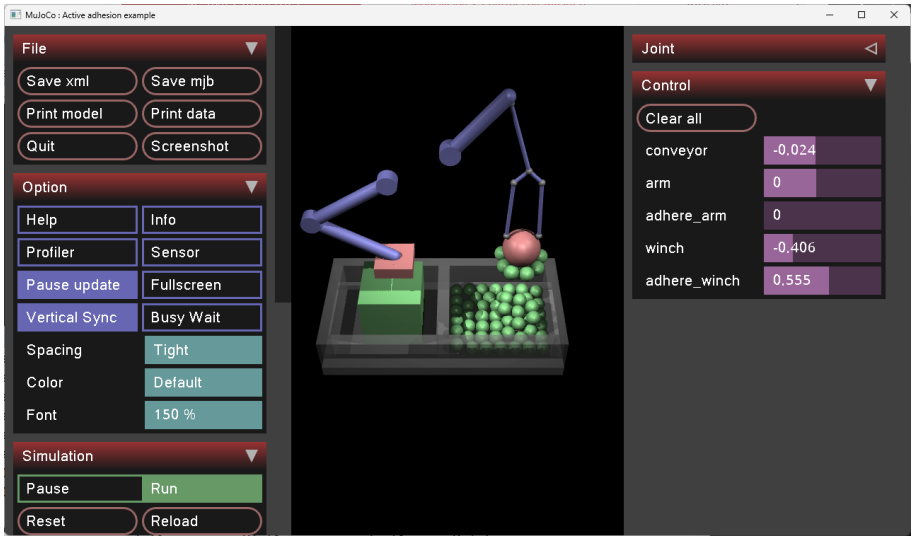
- XML (eXtensible Markup Language) es similar a HTML (Hypertext Markup Language).
- XML es una manera de guardar estructuras de datos con tags en una convención genérica <tag> ... </tag>, así como CSV (comma separated values) es una manera de guardar datos separados por commas.
- El código del modelo MuJoCO XML se inicia con <mujoco> y termina con </mujoco>.
- Los comentarios se encierran entre <!-- y -->.
- Los ejemplos pueden visualizarse con la herramienta de simulación: "../mujoco-3.1.3-windows-x86_64/bin/simulate.exe", arrastrando los modelos en:
  - mujoco-3.1.3-windows-x86_64/model
  - MuJoCo Menangerie: https://github.com/google-deepmind/mujoco_menagerie

Algunos ejemplos...



car

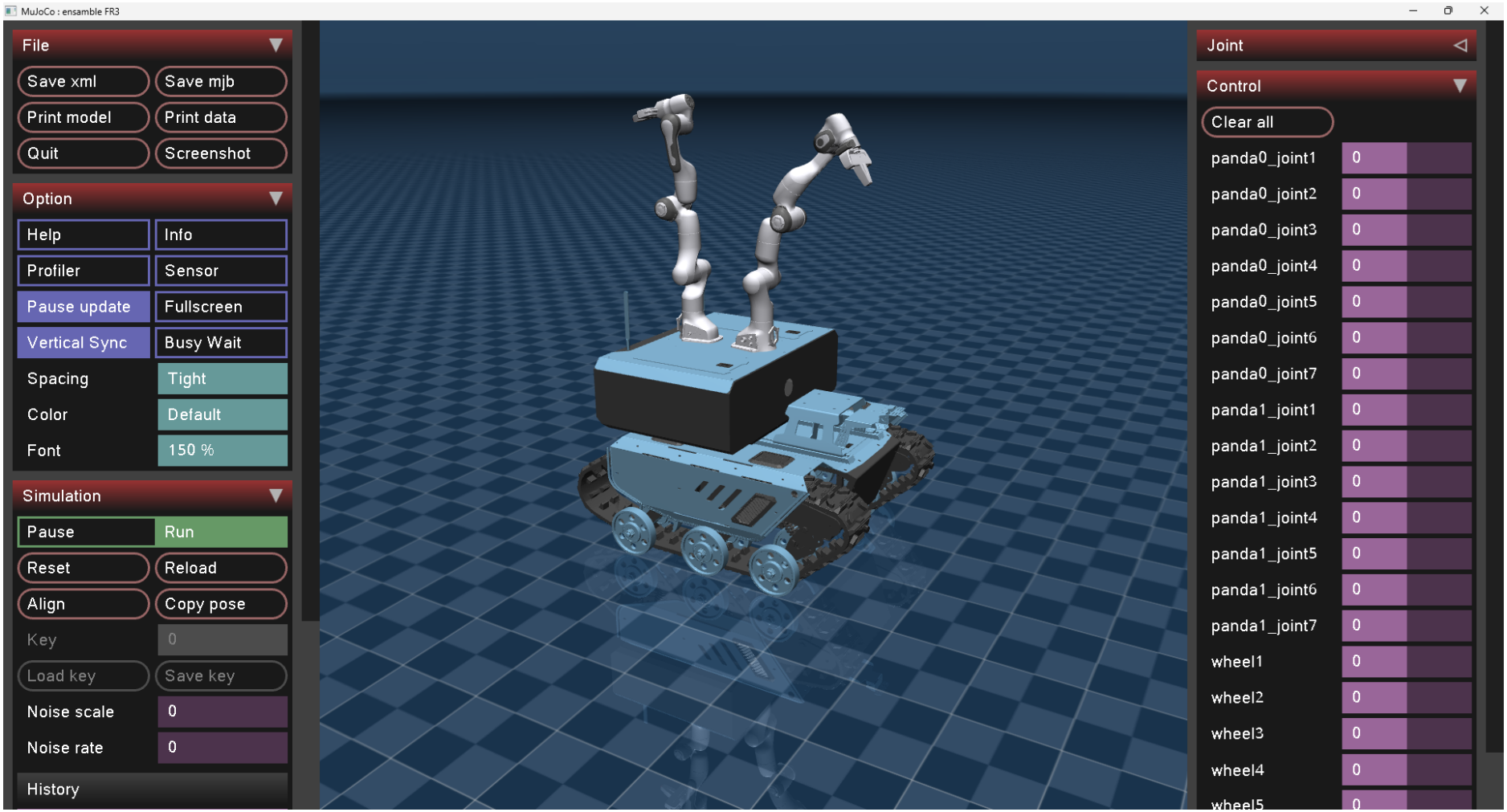
humanoid



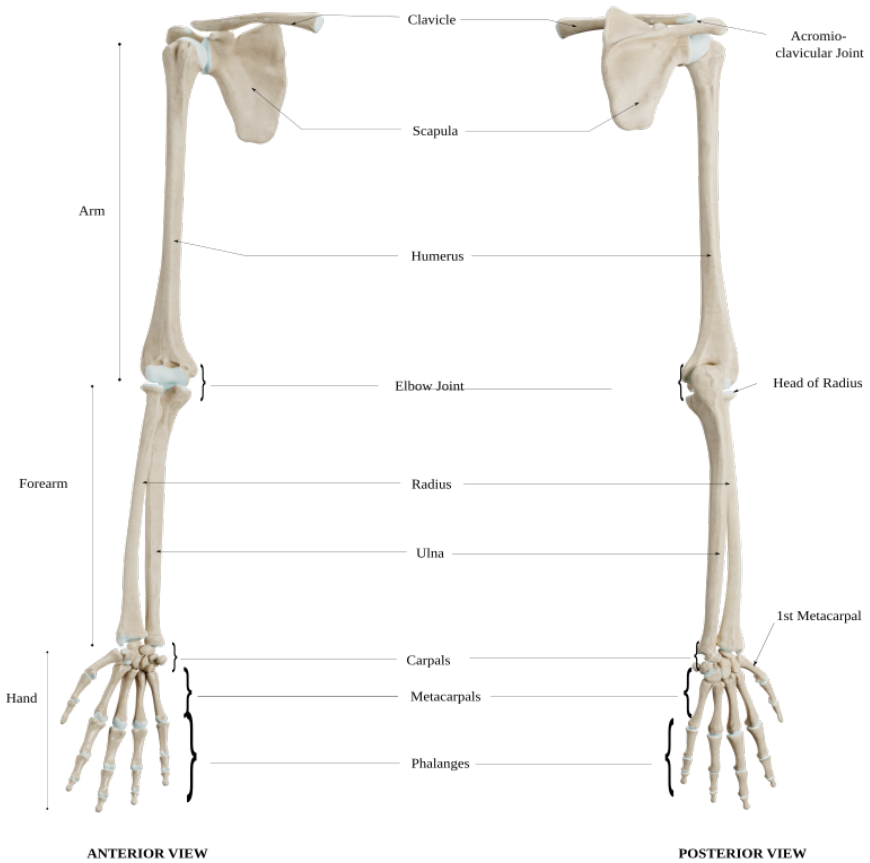
active_adhesion

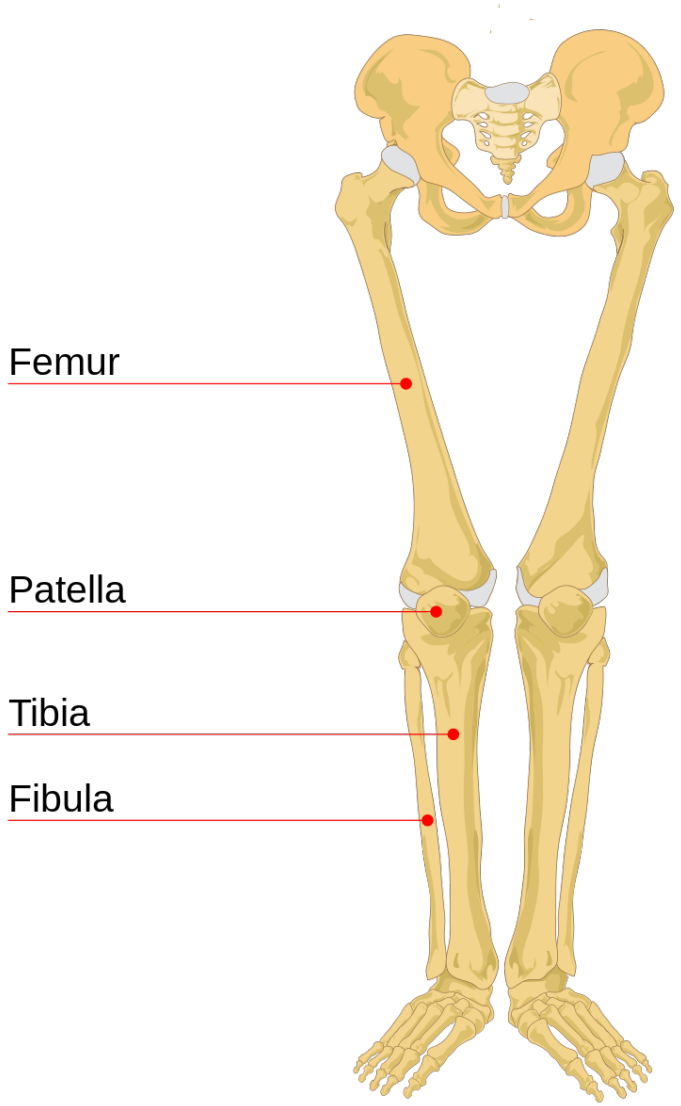
unitree go1 (Menangerie)

Dual-arm mobile manipulator



Humanoide


Clavicle
Acromio-
clavicular Joint
Scapula
Arm
Humerus
Elbow Joint
Head of Radius
Forearm
Radius
Ulna
1st Metacarpal
Carpals
Hand
Metacarpals
Phalanges
ANTERIOR VIEW
POSTERIOR VIEW


Femur
Patella
Tibia
Fibula

Partes del Modelo MuJoCo en XML

Partes principales

```
<mujoco>
  <option timestep="0.01" integrator="RK4" gravity="0 0 -9.81">
    ...
  </option>

  <asset>
    ... Aquí se definen recursos: materiales, texturas, mallas
    ...
  </asset>

  <worldbody>
    ... Aquí se define el robot
  </worldbody>

  <actuator>
    ...  Aquí se definen actuadores: pservo, vservo, motor
  </actuator>

  <sensor>
    ... Aquí se definen sensores: framepos, framequat,
    velocitmeter, accelerometer, gyro, magnetometer, rangefinder
    , ...
  </sensor>

</mujoco>
```

MuJoCo XML: (1) <option> ... </option>

- El código del modelo se inicia con <mujoco> y termina con </mujoco>.
- Los comentarios se encierran entre <!-- y -->.

<option>

```
1
2   <option timestep="0.01" integrator="RK4" gravity="0 0 -9.81">
3     <!-- <flag sensornoise="enable" energy="enable" contact="
      disable" /> -->
4     <flag sensornoise="disable" energy="enable" contact="enable
      " />
5   </option>
6
```

MuJoCo XML: (2) <asset> ... </asset>

<asset> ... </asset> es opcional:

<asset>

```
1
2   <asset>
3     <texture name="grid" type="2d" builtin="checker" width="512
      " height="512" rgb1=".9 .9 .9" rgb2=".1 .1 .1"/>
4     <material name="grid" texture="grid" texrepeat="10 10"
      texuniform="true" reflectance=".2"/>
5
6     <mesh name="tetrahedron" vertex="0 0 0  1 0 0  0 1 0  0 0
      0.3"/>
7   </asset>
8
```

MuJoCo XML: (3) <worldbody> ... </wordbody>

Con <worldbody> ... </wordbody> se define el mundo y el robot.

<worldbody>

```
1
2     <worldbody>
3         <light diffuse=".5 .5 .5" pos="0 0 3" dir="0 0 -1"/>
4     <!-- <geom type="plane" size="1 1 0.1" rgba=".9 0 0 1"/>
      -->
5
6     <geom type="plane" size="1 1 0.1" material="grid"/>
7
8     <geom type="mesh" mesh="tetrahedron" pos="2 0 0" euler="0 0
       135" rgba="1 0 0 1" />
9
10    <!-- Begin UAV -->
11    <body name="uav" pos="0 0 1" euler="0 0 0">
12      <!-- <freejoint/> -->
13      <joint name="uav_free_joint" type="free"/>
14      <geom type="box" size="0.25 0.25 0.05" rgba="0 0.9 0 1"
      mass="30"/>
15      <site name="chassis_frame" pos="0 0 0" size="0.01" />
16      <site name="rangefinder_frame" pos=".3 0 0" euler="0 90 0
      " size="0.01" />
17      <!-- Rangefinders measure along the Z axis... rotate so
      that it does not point infinite space! -->
18
19    </body>
20  </worldbody>
21
```

MuJoCo XML: (4) <actuator> ... </actuator>

- En <actuator> ... </actuator> se definen actuadores para las articulaciones.
- Un motor es un "wrench" o "spatial force" (fuerzas lineales-torques):

$$\tau = [f_x \; f_y \; f_z \; \tau_x \; \tau_y \; \tau_z]$$

- El vector gear define el factor de amplificación de cada elemento de τ.
- Ver más en: https://mujoco.readthedocs.io/en/stable/XMLreference.html#actuator

<actuator>

```
1
2   <actuator>
3     <motor name="thrust1" site="chassis_frame" gear="0 0 1 0 0
      .1" ctrlrange="-1000 1000" />
4     <!-- <position name="pservo1" joint="pin" kp="100" /> -->
5     <!-- <velocity name="vservo1" joint="pin" kv="10"  /> -->
6   </actuator>
7
```

MuJoCo XML: (5) <sensor> ... </sensor>

- En <sensor> ... </sensor> se definen los sensores.
- Ver más en: Sensorshttps://mujoco.readthedocs.io/en/stable/XMLreference.html#sensor

<sensor>

```
1
2   <sensor>
3     <framepos  objtype="site" objname="chassis_frame" />
4     <framequat objtype="site" objname="chassis_frame" />
5     <velocimeter name="sensor_vel" site="chassis_frame" />
6     <accelerometer name="sensor_accel" site="chassis_frame" />
7     <gyro name="sensor_gyro" site="chassis_frame" />
8     <magnetometer name="compass" site="chassis_frame" />
9     <rangefinder site="rangefinder_frame" noise="0" />
10  </sensor>
11
```

Simulación: Loop

Los pasos principales son:

1. Leer el input del usuario (teclado, mouse, joystick — PyGame, GLFW)
2. Actualizar el estado del modelo (integrar ODE — NumPy, MuJoCo)
3. Actualizar la pantalla (renderizar — PyGame, GLFW)
4. (opcional: Guardar los datos)

Simulación: Loop

Loop de simulación

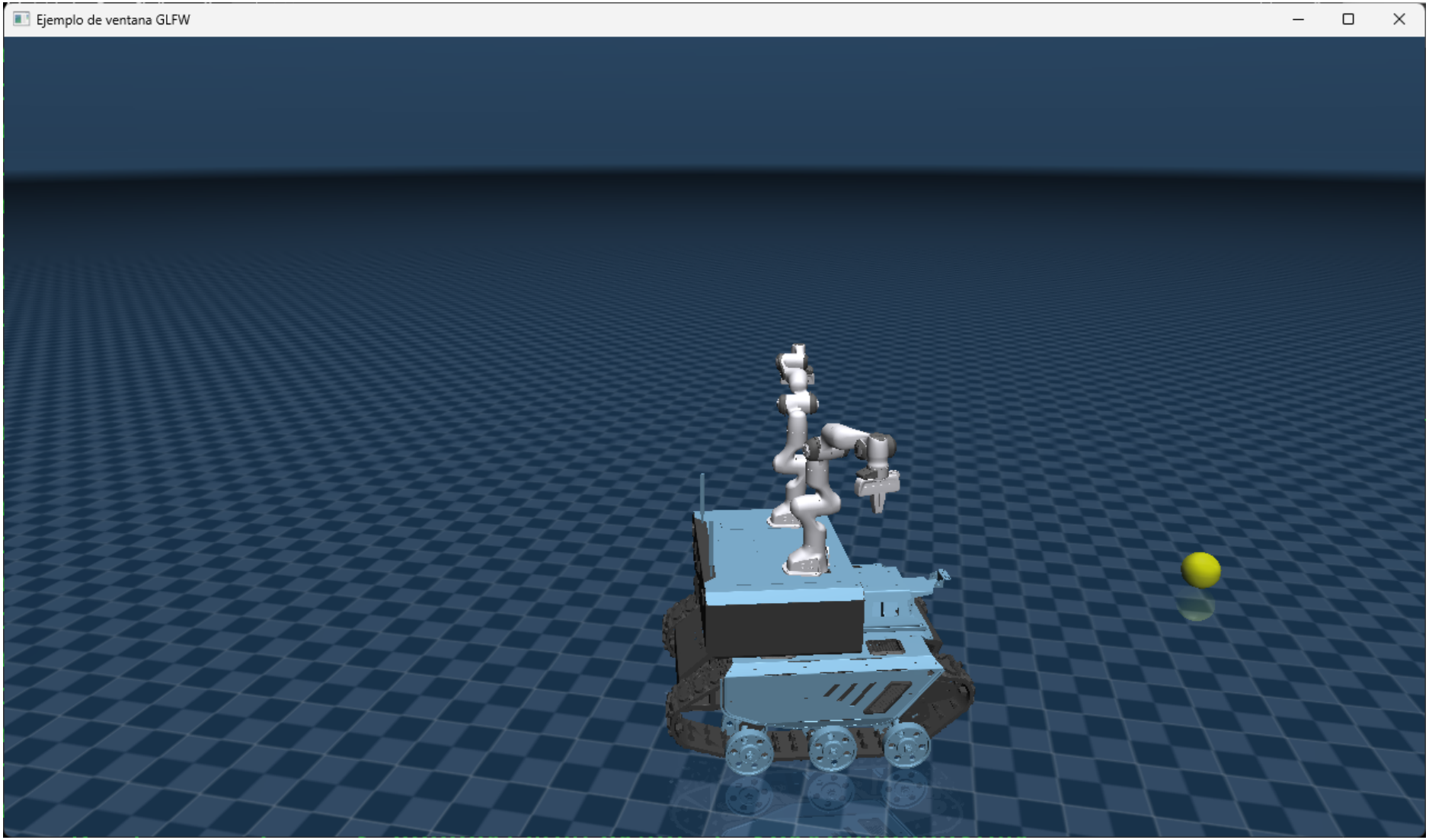
```
while not glfw.window_should_close(window):
    # Obtener y ejecutar eventos
    glfw.poll_events()

    # Actualizar la simulación en un paso
    mujoco.mj_step(model, data)

    # Obtenemos algunos datos de sensores
    print('Position  -qpos: ', data.qpos[0], data.qpos[1], data.qpos[2])

    # Get framebuffer viewport
    viewport_width, viewport_height = glfw.get_framebuffer_size(window)
    viewport = mujoco.MjrRect(0, 0, viewport_width, viewport_height)

    # Update scene and render
    mujoco.mjv_updateScene(model, data, opt, None, cam,
                           mujoco.mjtCatBit.mjCAT_ALL.value, scene)
    mujoco.mjr_render(viewport, scene, context)

    glfw.swap_buffers(window)

glfw.terminate()
```

Definiendo un controlador

Definiendo un controlador

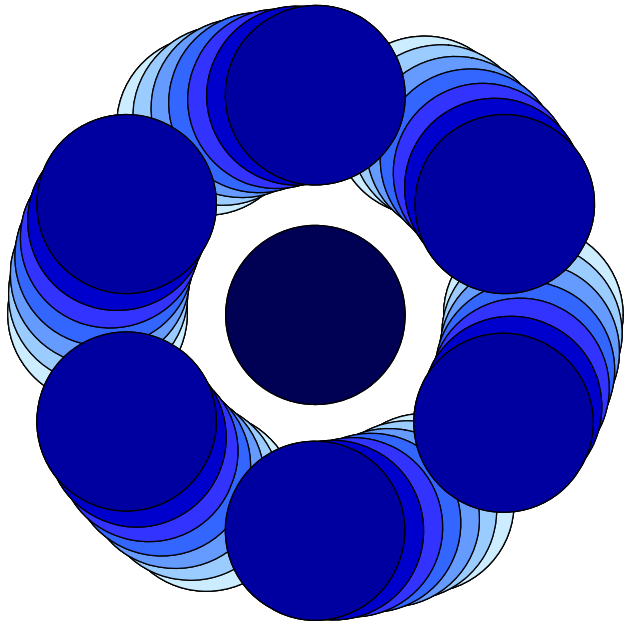
```
1
2  # --- Asignación del manejador del controlador ---
3  mujoco.set_mjcb_control(controller)
4
5  def controller(model, data):
6      global Z_des, Thrust, MV, error1, error2
7
8      if automatic:
9          # """
10         # Place you controller here.
11         # """
12         Z = data.qpos[2] # Altitud actual del UAV
13         error = Z_des-Z
14         MV = MV + Kp*(error-error1) + Ki*Ts*error + (Kd/Ts)*(
       error-2*error1+error2)
15         data.ctrl[0] = MV
16         print('Altitud deseada: ', Z_des, 'Altitud actual: ',
       data.qpos[2], 'Thrust: ', MV)
17         error2 = error1
18         error1 = error
19
20     else: # Manual
21         data.ctrl[0] = Thrust
22
23     return None
```

Dual-arm mobile manipulator en Python con controlador



Ejemplo: Simone_Tilleria/papers/simulación/Github
ROBOT/simulation_franka.py

¡Gracias!



ral.ing.puc.cl

Robotics & Automation Laboratory
School of Engineering
Pontificia Universidad Católica de Chile