International Olympiad in Artificial Intelligence

Team Challenge

See you in 2025.08.04

UPDATES

Date	Changes Made
0602	Original release

Date	Changes Made

"Factory of the Future" Grand Challenge

Have you ever visited a modern factory? Rows of robotic arms precisely sort and assemble parts, and large equipment, cars, electronics, and everyday products are standardized and produced on assembly lines in a matter of minutes.

However, there are still certain tasks that are difficult to standardize or require flexible adjustments, which currently rely on human cooperation. But in the near future, these tasks will be handled by general-purpose humanoid robots. They will help free humans from repetitive manual labor, giving people more time to engage in creative and passionate work.

1. Introduction

You and your team are general-purpose humanoid robot engineers at the Factory of the Future. You have designed and built an impressive humanoid robot—**Galbot**. Please write a program to help Galbot complete its challenges in the Factory of the Future.

Galbot has been assigned to organize the warehouse. There are scattered, unsorted items on the tables in the warehouse. Please help Galbot move around, locate these items, and organize them onto the shelves.

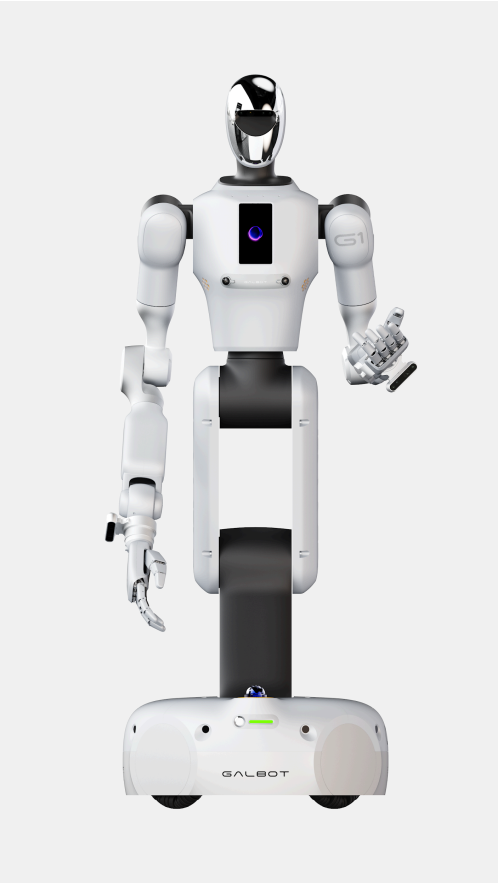
Be careful—there are obstacles on the warehouse floor. Don’t let Galbot bump into them! You and your team should help Galbot complete these tasks as quickly as possible!

2. About Galbot

2.1. Galbot performance parameters introduction.

		Galbot G1
Specification	Max. Height	1730mm
	Arm Length (Shoulder to Wrist Center)	710mm
	Leg Lift Height Difference	650mm
	Workspace (Wrist Center)	Vertical: 0-2100mm, Horizontal: 1900mm
	Payload Capacity (Wrist Center)	Single Arm: 5kg, Combined: 10kg
	Body Weight	80Kg
	Operating Environment	0-40°C, Humidity 90% (non-condensing)

System	Computing Power	275TOPS
	Network	WIFI 2.4/5 GHz, BT
	Display	5.5" Touchscreen, 1920*1080 resolution
	Microphone	Linear 4-Mic Array ×2
	Speaker	5W X 2
Perception	Head	Depth Camera (Realsense D436)×1
	Wrists (Left+Right)	Depth Camera (Realsense D415)×2
		6-Axis Force Sensor×2
	Chassis	3D LiDAR (Mid360)×1
		Ultrasonic Sensor×8
		IMU x 1
Degrees of Freedom (excluding chassis and end effector)	Neck DOF	2
	Arm DOF (per arm)	7 x 2
	Waist DOF	3
	Leg DOF	2
Chassis	Chassis Type	4-Wheel Omnidirectional Mobile Base
	Chassis Dimensions (Length/Projection Diameter)	600mm/600mm/625mm
	Max Speed	1.5m/s



2.2. Joint Motor Limit

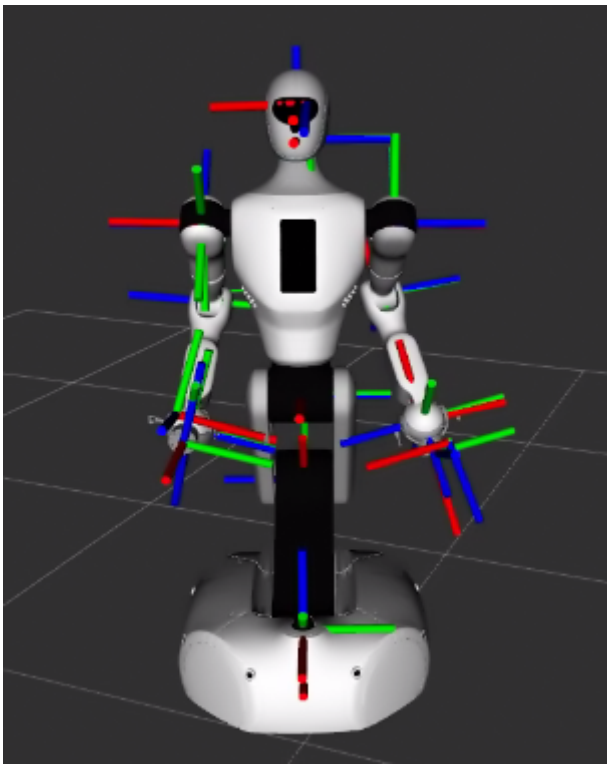
Joint Name	Limit (radians)
leg_joint1	0.0 - 0.9374
leg_joint2	0.0 - 2.5847
leg_joint3	0.0 - 2.3262
leg_joint4	-1.5906 - 1.5906
leg_joint5	-0.1645 - 0.1645
head_joint1	-1.5208 - 1.5208
head_joint2	-0.321147949 - 0.321147949
right_arm_joint1	-3.00432619 - 3.00432619
right_arm_joint2	-1.608062789 - 1.608062789
right_arm_joint3	-2.916972222 - 2.916972222
right_arm_joint4	-1.869862177 - 2.5679938779914944
right_arm_joint5	-2.916972222 - 2.916972222
right_arm_joint6	-0.7353981633974483 - 0.8226646259971648

right_arm_joint7	-1.538202778 - 1.538202778
left_arm_joint1	-3.00432619 - 3.00432619
left_arm_joint2	-1.608062789 - 1.608062789
left_arm_joint3	-2.916972222 - 2.916972222
left_arm_joint4	-2.5679938779914944 - 1.869862177
left_arm_joint5	-2.916972222 - 2.916972222
left_arm_joint6	-0.8226646259971648 - 0.7353981633974483
left_arm_joint7	-1.538202778 - 1.538202778

2.3. Coordinate Frames

The joint coordinate frames are defined as follows (see figure):

- **X-axis**: Red
- **Y-axis**: Green
- **Z-axis**: Blue



3. Installation Guide

~~Issac Sim~~

Please view the docs at SynthNova Physics Simulator Edu | SynthNova Physics Simulator Edu documentation**for the latest update!** Physics Simulator Edu is **a MuJoCo-based simulation platform** for robotics applications with high-level robot control interfaces designed for Galbot G1.

To install the physics_sim_edu pachage, please follow the bash commands below:

```
git clone https://github.com/galbot-ioai/physics_sim_edu.git
cd physics_sim_edu
pip install -e ./src/synthnova_config
pip install -e .
```

To verify the installion, you can run the following command:

```
from physics_simulator import PhysicsSimulator
from synthnova_config import PhysicsSimulatorConfig

# Test basic import
config = PhysicsSimulatorConfig()
sim = PhysicsSimulator(config)
print("Installation successful!")
```

4. Basic Examples

4.1. Basic Simulation

```
from physics_simulator import PhysicsSimulator
from synthnova_config import PhysicsSimulatorConfig, RobotConfig
from pathlib import Path

# Create sim config
my_config = PhysicsSimulatorConfig()
physics_simulator = PhysicsSimulator(my_config)

# Add default scene
physics_simulator.add_default_scene()

# Initialize the simulator
physics_simulator.initialize()
```

```
# Run the display loop
physics_simulator.loop()

# Close the simulator
physics_simulator.close()
```

4.2. Add Entities

4.2.1. Add Robot

```
from physics_simulator import PhysicsSimulator
from synthnova_config import PhysicsSimulatorConfig, RobotConfig
from pathlib import Path

def main():
   # Create sim config
   my_config = PhysicsSimulatorConfig()
   physics_simulator = PhysicsSimulator(my_config)
   physics_simulator.add_default_scene()

   # Add robot
   robot_config = RobotConfig(
       prim_path="/World/Galbot",
       name="galbot_one_charlie",
       mjcf_path=Path()
           .joinpath(physics_simulator.synthnova_assets_directory)
           .joinpath("synthnova_assets")
           .joinpath("robot")
           .joinpath("galbot_one_charlie_description")
           .joinpath("galbot_one_charlie.xml"),
       position=[0, 0, 0],
       orientation=[0, 0, 0, 1]
   )
   physics_simulator.add_robot(robot_config)
   physics_simulator.initialize()

   # Get robot state
   robot_state =
physics_simulator.get_robot_state(robot_config.prim_path)
   print(robot_state)

   physics_simulator.loop()
   physics_simulator.close()
```

4.2.2. Add Basic Geometry

```
from synthnova_config import PhysicsSimulatorConfig, CuboidConfig
from physics_simulator import PhysicsSimulator
from pathlib import Path

def main():
   # Initialize simulator
   my_config = PhysicsSimulatorConfig()
   physics_simulator = PhysicsSimulator(my_config)
   physics_simulator.add_default_scene()

   # Add cube 1
   cube_1_config = CuboidConfig(

prim_path=Path(physics_simulator.root_prim_path).joinpath("cube_1"),
       position=[2, 2, 2],
       orientation=[0, 0, 0, 1],
       scale=[1, 1, 1],
       color=[1.0, 0.0, 0.0],
   )
   cube_1_path = physics_simulator.add_object(cube_1_config)

   # Add cube 2
   cube_2_config = CuboidConfig(

prim_path=Path(physics_simulator.root_prim_path).joinpath("cube_2"),
       position=[0, 0, 2],
       orientation=[0, 0, 0, 1],
       scale=[1, 1, 1],
       color=[0.0, 1.0, 0.0],
   )
   cube_2_path = physics_simulator.add_object(cube_2_config)

   # Add cube 3
   cube_3_config = CuboidConfig(

prim_path=Path(physics_simulator.root_prim_path).joinpath("cube_3"),
       position=[-2, -2, 2],
       orientation=[0, 0, 0, 1],
       scale=[1, 1, 1],
       color=[0.0, 0.0, 1.0],
   )
   cube_3_path = physics_simulator.add_object(cube_3_config)

   # Initialize and run
```

```
    physics_simulator.initialize()
    physics_simulator.get_object_state(cube_1_path)
    physics_simulator.loop()
    physics_simulator.close()
```

4.2.3. Add Mesh Object

```
from synthnova_config import PhysicsSimulatorConfig, MeshConfig,
RobotConfig
from physics_simulator import PhysicsSimulator
from pathlib import Path

def main():
    # Initialize simulator
    my_config = PhysicsSimulatorConfig()
    physics_simulator = PhysicsSimulator(my_config)
    physics_simulator.add_default_scene()

    # Add robot
    robot_config = RobotConfig(
        prim_path="/World/Galbot",
        name="galbot_one_charlie",
        mjcf_path=Path()
            .joinpath(physics_simulator.synthnova_assets_directory)
            .joinpath("synthnova_assets")
            .joinpath("robot")
            .joinpath("galbot_one_charlie_description")
            .joinpath("galbot_one_charlie.xml"),
        position=[0, 0, 0],
        orientation=[0, 0, 0, 1]
    )
    physics_simulator.add_robot(robot_config)

    # Add shelf mesh
    shelf_config = MeshConfig(
        prim_path="/World/Shelf",
        name="shelf",
        mjcf_path=Path()
            .joinpath(physics_simulator.synthnova_assets_directory)
            .joinpath("synthnova_assets")
            .joinpath("default")
            .joinpath("shelves")
            .joinpath("1")
            .joinpath("model")
```

```
            .joinpath("mjcf")
            .joinpath("convex_decomposition.xml"),
        position=[0.55, 0, 0],
        orientation=[0, 0, 0, 1]
    )
    physics_simulator.add_object(shelf_config)

    # Add bottle
    bottle_config = MeshConfig(
        prim_path="/World/Bottle",
        name="bottle",
        mjcf_path=Path()
            .joinpath(physics_simulator.synthnova_assets_directory)
            .joinpath("synthnova_assets")
            .joinpath("default")
            .joinpath("skus")
            .joinpath("1")
            .joinpath("model")
            .joinpath("mjcf")
            .joinpath("convex_decomposition.xml"),
        position=[0.55, 0, 0.1],
        orientation=[0, 0, 0, 1],
        scale=[0.122, 0.122, 0.122]
    )
    physics_simulator.add_object(bottle_config)

    # Initialize and run
    physics_simulator.initialize()
    physics_simulator.loop()
    physics_simulator.close()
```

4.3. Control parts of Galbot using galbot_interface

4.3.1. Chassis Control

```
from physics_simulator import PhysicsSimulator
from physics_simulator.galbot_interface import GalbotInterface,
GalbotInterfaceConfig
from physics_simulator.utils.data_types import JointTrajectory
from synthnova_config import PhysicsSimulatorConfig, RobotConfig
import numpy as np
from pathlib import Path
```

```
def interpolate_joint_positions(start_positions, end_positions, steps):
    return np.linspace(start_positions, end_positions, steps)

def main():
    # Create sim config
    my_config = PhysicsSimulatorConfig()
    physics_simulator = PhysicsSimulator(my_config)
    physics_simulator.add_default_scene()

    # Add robot
    robot_config = RobotConfig(
        prim_path="/World/Galbot",
        name="galbot_one_charlie",
        mjcf_path=Path()
            .joinpath(physics_simulator.synthnova_assets_directory)
            .joinpath("synthnova_assets")
            .joinpath("robot")
            .joinpath("galbot_one_charlie_description")
            .joinpath("galbot_one_charlie_olympic.xml"),
        position=[0, 0, 0],
        orientation=[0, 0, 0, 1]
    )
    robot_path = physics_simulator.add_robot(robot_config)
    physics_simulator.initialize()

    # Initialize the galbot interface
    galbot_interface_config = GalbotInterfaceConfig()

galbot_interface_config.modules_manager.enabled_modules.append("chassis")
    galbot_interface_config.chassis.joint_names = [
        f"{robot_config.name}/mobile_forward_joint",
        f"{robot_config.name}/mobile_side_joint",
        f"{robot_config.name}/mobile_yaw_joint",
    ]
    galbot_interface_config.robot.prim_path = robot_path

    galbot_interface = GalbotInterface(
        galbot_interface_config=galbot_interface_config,
        simulator=physics_simulator
    )
    galbot_interface.initialize()

    # Start the simulation
    physics_simulator.step()

    # Get current joint positions
```

```
    current_joint_positions =
galbot_interface.chassis.get_joint_positions()
    target_joint_positions = [1, 6, 1.5]

    # Interpolate joint positions
    positions = interpolate_joint_positions(
        current_joint_positions, target_joint_positions, 5000
    )
    # Create a joint trajectory
    joint_trajectory = JointTrajectory(positions=positions)

    # Follow the trajectory
    galbot_interface.chassis.follow_trajectory(joint_trajectory)

    # Run the display loop
    physics_simulator.loop()
    physics_simulator.close()
```

4.3.2. Left Arm Control

```
from physics_simulator import PhysicsSimulator
from physics_simulator.galbot_interface import GalbotInterface,
GalbotInterfaceConfig
from physics_simulator.utils.data_types import JointTrajectory
from synthnova_config import PhysicsSimulatorConfig, RobotConfig
import numpy as np
from pathlib import Path

def main():
    # Create sim config and add robot
    my_config = PhysicsSimulatorConfig()
    physics_simulator = PhysicsSimulator(my_config)
    physics_simulator.add_default_scene()

    robot_config = RobotConfig(
        prim_path="/World/Galbot",
        name="galbot_one_charlie",
        mjcf_path=Path()
            .joinpath(physics_simulator.synthnova_assets_directory)
            .joinpath("synthnova_assets")
            .joinpath("robot")
            .joinpath("galbot_one_charlie_description")
            .joinpath("galbot_one_charlie.xml"),
```

```
        position=[0, 0, 0],
        orientation=[0, 0, 0, 1]
    )
    robot_path = physics_simulator.add_robot(robot_config)
    physics_simulator.initialize()

    # Initialize the galbot interface
    galbot_interface_config = GalbotInterfaceConfig()

galbot_interface_config.modules_manager.enabled_modules.append("left_arm"
)
    galbot_interface_config.left_arm.joint_names = [
        f"{robot_config.name}/left_arm_joint1",
        f"{robot_config.name}/left_arm_joint2",
        f"{robot_config.name}/left_arm_joint3",
        f"{robot_config.name}/left_arm_joint4",
        f"{robot_config.name}/left_arm_joint5",
        f"{robot_config.name}/left_arm_joint6",
        f"{robot_config.name}/left_arm_joint7",
    ]
    galbot_interface_config.robot.prim_path = robot_path

    galbot_interface = GalbotInterface(
        galbot_interface_config=galbot_interface_config,
        simulator=physics_simulator
    )
    galbot_interface.initialize()

    # Control arm
    physics_simulator.step()
    current_joint_positions =
galbot_interface.left_arm.get_joint_positions()
    target_joint_positions = [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7]

    # Create and follow trajectory
    positions = np.linspace(current_joint_positions,
target_joint_positions, 500)
    joint_trajectory = JointTrajectory(positions=positions)
    galbot_interface.left_arm.follow_trajectory(joint_trajectory)

    physics_simulator.loop()
    physics_simulator.close()
```

4.3.3. Left Gripper Control

```
from physics_simulator import PhysicsSimulator
from physics_simulator.galbot_interface import GalbotInterface,
GalbotInterfaceConfig
from physics_simulator.utils.data_types import JointTrajectory
from synthnova_config import PhysicsSimulatorConfig, RobotConfig
import numpy as np

from pathlib import Path

def interpolate_joint_positions(start_positions, end_positions, steps):
    return np.linspace(start_positions, end_positions, steps)


def main():
    # Create sim config
    my_config = PhysicsSimulatorConfig()

    # Instantiate the simulator
    synthnova_physics_simulator = PhysicsSimulator(my_config)

    # Add default ground plane if you need
    synthnova_physics_simulator.add_default_scene()

    # Add robot
    robot_config = RobotConfig(
        prim_path="/World/Galbot",
        name="galbot_one_charlie",
        mjcf_path=Path()
        .joinpath(synthnova_physics_simulator.synthnova_assets_directory)
        .joinpath("synthnova_assets")
        .joinpath("robot")
        .joinpath("galbot_one_charlie_description")
        .joinpath("galbot_one_charlie.xml"),
        position=[0, 0, 0],
        orientation=[0, 0, 0, 1]
    )
    robot_path = synthnova_physics_simulator.add_robot(robot_config)

    # Initialize the simulator
    synthnova_physics_simulator.initialize()

    # Initialize the galbot interface
    galbot_interface_config = GalbotInterfaceConfig()
    # Enable the modules
```

```
galbot_interface_config.modules_manager.enabled_modules.append("left_grip
per")
   galbot_interface_config.left_gripper.joint_names = [
       f"{robot_config.name}/left_gripper_robotiq_85_right_knuckle_joint"
   ]
   # Bind the simulation entity prim path to the interface config
   galbot_interface_config.robot.prim_path = robot_path
   galbot_interface = GalbotInterface(
       galbot_interface_config=galbot_interface_config,
       simulator=synthnova_physics_simulator,
   )
   galbot_interface.initialize()

   # Set the gripper to close
   galbot_interface.left_gripper.set_gripper_close()

   # Run the display loop
   synthnova_physics_simulator.loop()

   # Close the simulator
   synthnova_physics_simulator.close()


if __name__ == "__main__":
   main()
```

4.4. Get Sensor infos

4.4.1. Front Head Camera

```
from physics_simulator import PhysicsSimulator
from synthnova_config import (
   PhysicsSimulatorConfig,
   RobotConfig,
   RgbCameraConfig,
   RealsenseD435RgbSensorConfig,
   DepthCameraConfig,
   RealsenseD435DepthSensorConfig,
)
from physics_simulator.galbot_interface import GalbotInterface,
GalbotInterfaceConfig
from physics_simulator.utils import preprocess_depth
import os
```

```python
import numpy as np
import cv2
from pathlib import Path

def main():
    """Main function to set up and run the front head camera example."""
    # Instantiate the simulator
    my_config = PhysicsSimulatorConfig()
    physics_simulator = PhysicsSimulator(my_config)

    # Add default scene
    physics_simulator.add_default_scene()

    # Add robot
    robot_config = RobotConfig(
        prim_path="/World/Galbot",
        name="galbot_one_charlie",
        mjcf_path=Path()
        .joinpath(physics_simulator.synthnova_assets_directory)
        .joinpath("synthnova_assets")
        .joinpath("robot")
        .joinpath("galbot_one_charlie_description")
        .joinpath("galbot_one_charlie.xml"),
        position=[0, 0, 0],
        orientation=[0, 0, 0, 1]
    )
    robot_path = physics_simulator.add_robot(robot_config)

    # Add front head RGB camera (RealSense D435)
    front_head_rgb_camera_config = RgbCameraConfig(
        name="front_head_rgb_camera",
        prim_path=os.path.join(
            robot_path,
            "head_link2",
            "head_end_effector_mount_link",
            "front_head_rgb_camera",
        ),
        translation=[0.09321, -0.06166, 0.033],
        rotation=[
            0.683012701855461,
            0.1830127020294028,
            0.18301270202940284,
            0.6830127018554611,
        ],
        sensor_config=RealsenseD435RgbSensorConfig(),
```

```
parent_entity_name="galbot_one_charlie/head_end_effector_mount_link"
    )
    front_head_rgb_camera_path =
physics_simulator.add_sensor(front_head_rgb_camera_config)

    # Add front head depth camera (RealSense D435)
    front_head_depth_camera_config = DepthCameraConfig(
        name="front_head_depth_camera",
        prim_path=os.path.join(
            robot_path,
            "head_link2",
            "head_end_effector_mount_link",
            "front_head_depth_camera",
        ),
        translation=[0.09321, -0.06166, 0.033],
        rotation=[
            0.683012701855461,
            0.1830127020294028,
            0.18301270202940284,
            0.6830127018554611,
        ],
        sensor_config=RealsenseD435DepthSensorConfig(),

parent_entity_name="galbot_one_charlie/head_end_effector_mount_link"
    )
    front_head_depth_camera_path = physics_simulator.add_sensor(
        front_head_depth_camera_config
    )

    # Initialize the galbot interface
    galbot_interface_config = GalbotInterfaceConfig()
    # Enable the modules

galbot_interface_config.modules_manager.enabled_modules.append("front_hea
d_camera")
    # Bind the simulation entity prim path to the interface config
    galbot_interface_config.robot.prim_path = robot_path
    galbot_interface_config.front_head_camera.prim_path_rgb =
front_head_rgb_camera_path
    galbot_interface_config.front_head_camera.prim_path_depth = (
        front_head_depth_camera_path
    )
    galbot_interface = GalbotInterface(
        galbot_interface_config=galbot_interface_config,
simulator=physics_simulator
    )
```

```
    galbot_interface.initialize()

    # Start the simulation
    physics_simulator.play()

    # Initial steps to stabilize the simulation
    physics_simulator.step(10)

    while True:
        physics_simulator.step(7)

        # Get rgb data
        rgb_data = galbot_interface.front_head_camera.get_rgb()

        # Get depth data
        depth_data = galbot_interface.front_head_camera.get_depth()

        # Preprocess depth data for visualization
        # You can also use this function to preprocess the depth data for
other purposes
        depth_data = preprocess_depth(
            depth_data,
            scale=1000,  # m to mm
            min_value=0.0,
            max_value=3 * 1000,  # 3m to mm
            data_type=np.uint16,
        )

        # Display images in non-blocking way
        cv2.imshow("RGB Camera", cv2.cvtColor(rgb_data,
cv2.COLOR_RGB2BGR))
        cv2.imshow("Depth Camera", depth_data)

        # Wait for 1ms and check for 'q' key to quit
        if cv2.waitKey(1) & 0xFF == ord("q"):
            cv2.destroyAllWindows()
            break

    # Get camera parameters
    params = galbot_interface.front_head_camera.get_parameters()
    intrinsic_matrix = params["rgb"]["intrinsic_matrix"]
    print(params)
    print("intrinsic_matrix: ", intrinsic_matrix)

    # Close the simulator
    physics_simulator.close()
```

```
if __name__ == "__main__":
   main()
```

4.4.2. Left Wrist Camera

```
from physics_simulator import PhysicsSimulator
from synthnova_config import (
PhysicsSimulatorConfig,
RobotConfig,
RgbCameraConfig,
RealsenseD415RgbSensorConfig,
DepthCameraConfig,
RealsenseD415DepthSensorConfig,
)
from physics_simulator.galbot_interface import GalbotInterface,
GalbotInterfaceConfig
from physics_simulator.utils import preprocess_depth
import os
import numpy as np
import cv2
from pathlib import Path
def main():
   """Main function to set up and run the left wrist camera example."""
   # Instantiate the simulator
my_config = PhysicsSimulatorConfig()
physics_simulator = PhysicsSimulator(my_config)
   # Add default scene
physics_simulator.add_default_scene()
   # Add robot
robot_config = RobotConfig(
prim_path="/World/Galbot",
name="galbot_one_charlie",
mjcf_path=Path()
       .joinpath(physics_simulator.synthnova_assets_directory)
       .joinpath("synthnova_assets")
       .joinpath("robot")
       .joinpath("galbot_one_charlie_description")
       .joinpath("galbot_one_charlie.xml"),
position=[0, 0, 0],
orientation=[0, 0, 0, 1]
   )
```

```
robot_path = physics_simulator.add_robot(robot_config)
   # Add left wrist RGB camera (RealSense D415)
left_wrist_rgb_camera_config = RgbCameraConfig(
name="left_wrist_rgb_camera",
prim_path=os.path.join(
robot_path,
           "left_arm_link7",
           "left_arm_end_effector_mount_link",
           "left_wrist_rgb_camera",
       ),
translation = [0.005124659527755139, 0.06720377942456242,
-0.0056533333810578162],
rotation = [-0.0353557, 0.7194, -0.020291, -0.6934],
camera_axes="usd",
sensor_config=RealsenseD415RgbSensorConfig(),
parent_entity_name="galbot_one_charlie/left_arm_end_effector_mount_link"
   )
left_wrist_rgb_camera_path =
physics_simulator.add_sensor(left_wrist_rgb_camera_config)
   # Add left wrist depth camera (RealSense D415)
left_wrist_depth_camera_config = DepthCameraConfig(
name="left_wrist_depth_camera",
prim_path=os.path.join(
robot_path,
           "left_arm_link7",
           "left_arm_end_effector_mount_link",
           "left_wrist_depth_camera",
       ),
translation = [0.005124659527755139, 0.06720377942456242,
-0.0056533333810578162],
rotation = [-0.0353557, 0.7194, -0.020291, -0.6934],
camera_axes="usd",
sensor_config=RealsenseD415DepthSensorConfig(),
parent_entity_name="galbot_one_charlie/left_arm_end_effector_mount_link"
   )
left_wrist_depth_camera_path = physics_simulator.add_sensor(
left_wrist_depth_camera_config
   )
   # Initialize the galbot interface
galbot_interface_config = GalbotInterfaceConfig()
   # Enable the modules
galbot_interface_config.modules_manager.enabled_modules.append("left_wris
t_camera")
   # Bind the simulation entity prim path to the interface config
galbot_interface_config.robot.prim_path = robot_path
galbot_interface_config.left_wrist_camera.prim_path_rgb =
```

```
left_wrist_rgb_camera_path
galbot_interface_config.left_wrist_camera.prim_path_depth = (
left_wrist_depth_camera_path
    )
galbot_interface = GalbotInterface(
galbot_interface_config=galbot_interface_config,
simulator=physics_simulator
    )
galbot_interface.initialize()
    # Start the simulation
physics_simulator.play()
    # Initial steps to stabilize the simulation
physics_simulator.step(10)
    while True:
physics_simulator.step(7)
        # Get rgb data
rgb_data = galbot_interface.left_wrist_camera.get_rgb()
        # Get depth data
depth_data = galbot_interface.left_wrist_camera.get_depth()
        # Preprocess depth data for visualization
        # You can also use this function to preprocess the depth data for
other purposes
depth_data = preprocess_depth(
depth_data,
scale=1000,  # m to mm
min_value=0.0,
max_value=5 * 1000,  # 5m to mm
data_type=np.uint16,
        )
        # Display images in non-blocking way
cv2.imshow("RGB Camera", cv2.cvtColor(rgb_data, cv2.COLOR_RGB2BGR))
cv2.imshow("Depth Camera", depth_data)
        # Wait for 1ms and check for 'q' key to quit
        if cv2.waitKey(1) & 0xFF == ord("q"):
cv2.destroyAllWindows()
            break
    # Get camera parameters
params = galbot_interface.left_wrist_camera.get_parameters()
intrinsic_matrix = params["rgb"]["intrinsic_matrix"]
    print(params)
    print("intrinsic_matrix: ", intrinsic_matrix)
    # Close the simulator
physics_simulator.close()
if __name__ == "__main__":
main()
```

5. Advanced Examples

5.1. Pick a cube and place it into a box

physics_sim_edu/examples/ioai/ioai_grasp_env.py at ioai/master · galbot-ioai/physics_sim_edu · GitHub

This example demonstrates a basic pick and place task using a simple state machine. The robot performs the following sequence:

1. Move to a pre-grasp position
2. Open the gripper
3. Move to the object
4. Close the gripper to grasp the object
5. Lift the object
6. Move to the target position
7. Open the gripper to release the object

5.2. Navigate in a grid map using A*

physics_sim_edu/examples/ioai/ioai_nav_env.py at ioai/master · galbot-ioai/physics_sim_edu · GitHub

This example showcases autonomous navigation using the A* pathfinding algorithm. The robot:

1. Receives a target position
2. Plans a path using A* algorithm
3. Follows the planned path while avoiding obstacles
4. Reaches the target position

Appendix A: API Documentation

Refer to https://github.com/galbot-ioai/physics_sim_edu.gitfor the detailed api documents

Appendix B: Learning Resources

- MuJoCo Documentation: Overview - MuJoCo Documentation
- PythonRobotics: Python sample codes and textbook for robotics algorithms.GitHub -

AtsushiSakai/PythonRobotics: Python sample codes and textbook for robotics algorithms.(a great reference of python sample codes for different use cases)

- Solve Inverse Kinematics in MuJoCo: Mink
  - Table of Contents — mink documentation
- Object Detection:
  - YOLO Model (YOLOv5, YOLOv7, etc.)
    - GitHub - ultralytics/yolov5: YOLOv5 🚀 in PyTorch > ONNX > CoreML > TFLite
    - GitHub - WongKinYiu/yolov7: Implementation of paper - YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors
  - Object Grasping of Humanoid Robot Based on YOLO
    - https://scispace.com/pdf/object-grasping-of-humanoid-robot-based-on-yolo-29id6r7axh.pdf