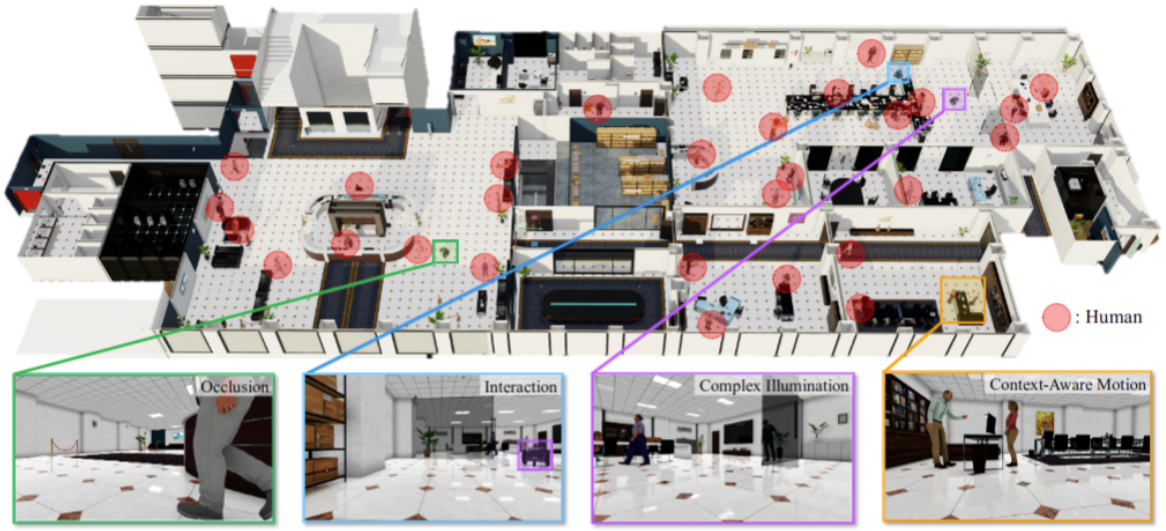
CSE-Dataset Tutorial



A Benchmark Synthetic Dataset for **C**-SLAM in **S**ervice **E**nvironments. (CSE Dataset)

This tutorial page is designed to guide users through the overall process of constructing the CSE dataset, providing all necessary processes and source code. By leveraging this page, users are empowered to generate highly customized sequences tailored to their specific requirements, including the construction of detailed simulation scenes and precise robot trajectories. All procedures are based on the NVIDIA Isaac Sim, which is a robotics simulator powered by the Omniverse platform that provides photo-realistic and physically accurate virtual environments. We hope this tutorial page will significantly contribute to the robotics community.

This tutorial was created by the authors of the CSE dataset on August 27, 2024.

0. Contents

1. NVIDIA Isaac Sim Setup

We follow the page (Workstation setup) to install the NVIDIA Isaac Sim in a workstation. Our workstation has the following system and development environment. It is highly recommended that you read the system requirements and Isaac Sim driver requirements included in the above link before installing NVIDIA Isaac Sim.

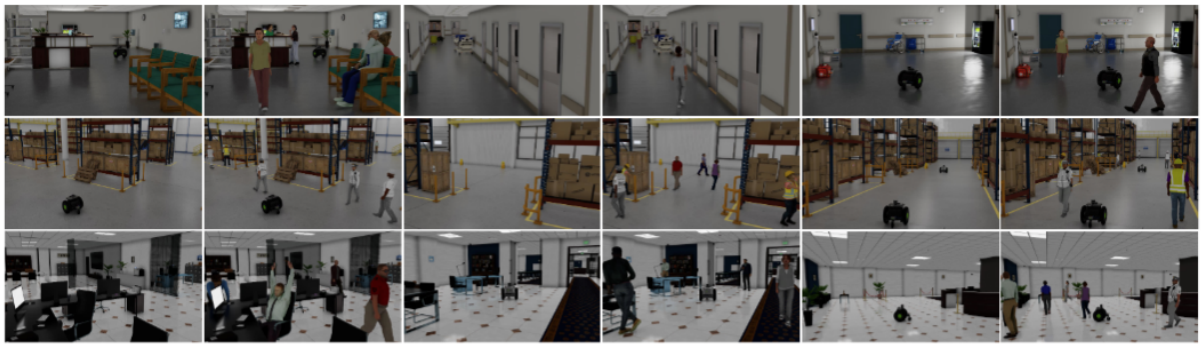
1.1. System

OS	CPU	Cores	RAM	Storage	GPU	VRAM
Ubuntu 20.04	Intel Core i7 (11th Generation)	16	128 GB	2T SSD 8T SSD	GeForce RTX 3090 * 2	24 GB

1.2. Development Environment

Isaac Sim Driver version (Ubuntu)	Isaac Sim release version	ROS Version
525.147.05	2022.2.1	Noetic

2. Simulation Scene Setup



Examples of service environments in the CSE dataset. (First row : Hospital, Second row : Warehouse, Third row : Office)

To construct the CSE dataset, we created three indoor service environments such as *Hospital*, *Office*, and *Warehouse* where real service robots are operated. Our dataset is organized into a total of six environments by categorizing each environment into "**Static**" and "**Dynamic**" types. Note that the "Dynamic" version is the environment where only dynamic objects like humans are added to the "Static" version of each environment. **Therefore, in this section (2. Simulation Scene Setup), we summarize how to create the customized "Static" scenes using Isaac Sim and how to add and control the dynamic objects.** All content is tailored to the version of Isaac Sim 2022.2.1. Please note that there may be some differences when using other versions of Isaac Sim. This section also does not detail the process exactly as we conducted it but rather shares the methods we utilized.

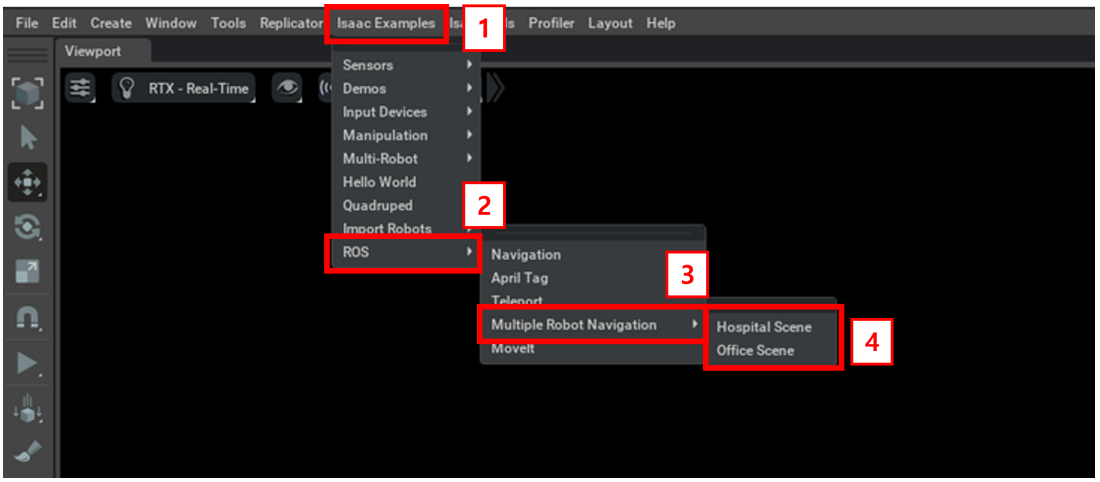
2.1. Static Scene Generation

This section describes the methods created for each environment. Since the *Hospital* and *Office* were constructed through the same process, their explanation is provided together. We recommend that users follow each step according to their purpose!

2.1.1. Hospital / Office

For the *Hospital* and *Office* environments, we utilize the basic scenes provided by NVIDIA and modify them with additional construction for our purposes. We proceed with structural and textural modifications to reflect the challenging cases such as homogeneous texture, visual redundancy, repetitive patterns, etc. In addition, we made an effort to create realistic environments by manually placing various assets suitable for each environment. All utilized assets are provided by Isaac Sim for free. The creation process is like below.

1. Open the Isaac Sim.
2. In a new stage, open the scene.



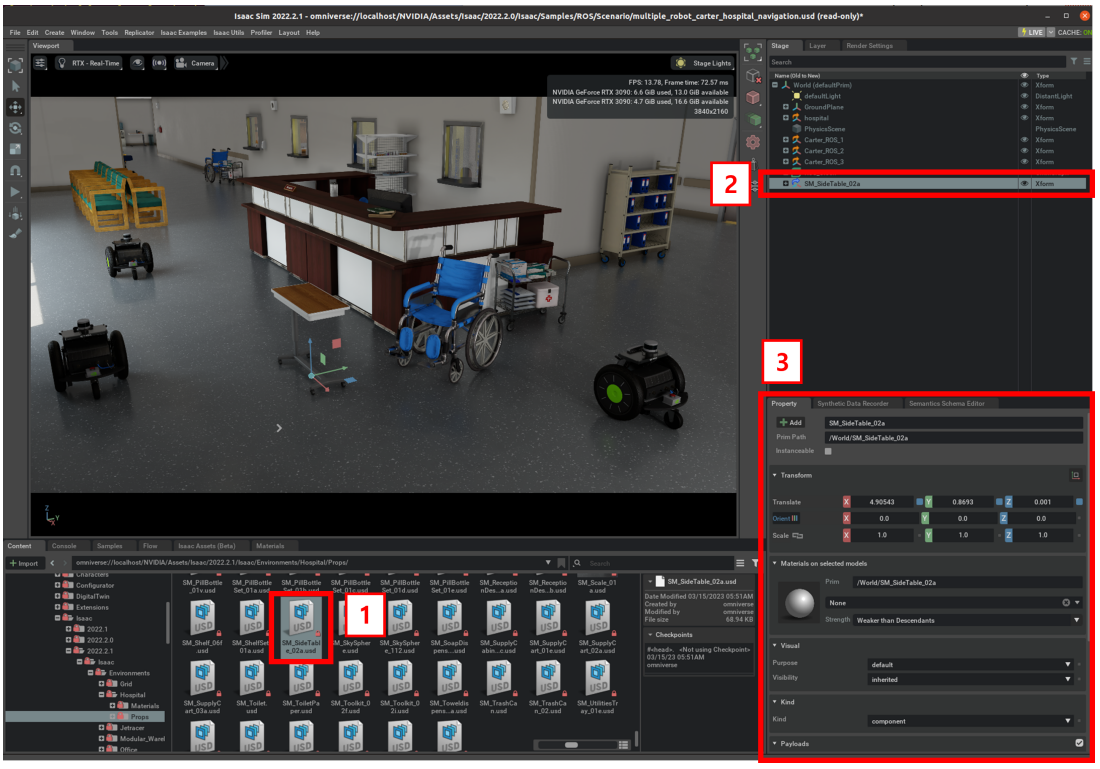
Hospital / Office scene path

- Hospital Path : (Top bar) Isaac Examples > ROS > Multiple Robot Navigation > Hospital Scene
- Office Path : (Top bar) Isaac Examples > ROS > Multiple Robot Navigation > Office Scene

3. Modify the scene to suit your purposes.

💡 **Note :**

If you open those scenes, there will be three robots (Carter 1, 2, 3). We utilized these to construct our dataset. However, details related to the robots and sensors are described in (Sec 3. Robot and Sensor Setup).



Method to add/adjust assets or modify scenes

1. Click the assets.
   - Asset path : (Content) > omniverse://localhost/NVIDIA/Assets/isaac/2022.2.1/Isaac/Environments/{Scene_name}/Props/
2. Drag and drop it onto the Stage or Viewport.

3. Modify the scene while adjusting the Transform of the assets.
   Clicking on the added asset will activate the Property panel where you can adjust the translation, rotation, and scale of the asset as shown below. Use these to adjust. We used this simple method to rearrange the asset and make structural and textural modifications.
4. Save you own scene.

2.1.2. Warehouse

For *Warehouse* environment, we use the SceneBlox tool of NVIDIA Omniverse Replicator to generate a basic scene. SceneBlox is a tool for creating large and consistent simulation scenes. Based on this tool, we generate the basic scene for *Warehouse*, and do the same post-processing (e.g., placing object assets, modification of structure and texture) as *Hospital* and *Office*. The process is like below.

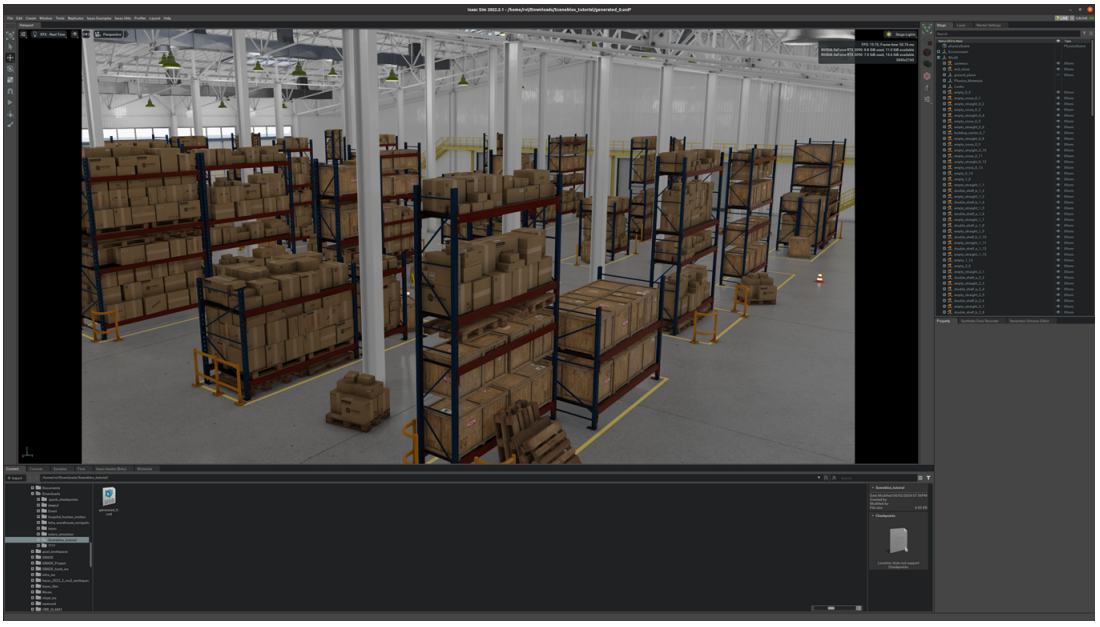
1. Generate a basic scene by referring to the page (10.8.3 Warehouse generation example).

   ** We utilized the default parameters and built a basic scene by executing the code below.*

   ```
   cd /home/{user}/.local/share/ov/pkg/isaac_sim-2022.2.1

   ./python.sh tools/scene_blox/src/scene_blox/generate_scene.py {/put/your/path/to/save/} --grid_config tools/scene_blox/parameters/warehouse/tile_config.yaml --generation_config tools/scene_blox/parameters/warehouse/tile_generation.yaml --cols 15 --rows 11 --constraints_config tools/scene_blox/parameters/warehouse/constraints.yaml --variants 1 --units_in_meters 1.0 --collisions
   ```

2. After a while, *Warehouse* scene file (e.g., generated_0.usd) will be created in the path you set.
3. Open the Isaac Sim and load the scene.



The basic scene of Warehouse using SceneBlox tool

We use this scene as our basic warehouse scene. Using this, we build the final *Warehouse* scene with the post-processing (e.g., Add/adjust the assets or modify the structure / texture) we did in Hospital / Office.

- Asset path : (Content) > omniverse://localhost/NVIDIA/Assets/isaac/2022.2.1/Isaac/Environments/Simple_Warehouse/Props/

2.2. Dynamic Objects Handling

We generate the “Dynamic” scene by adding a number of dynamic objects, such as humans, to the “Static” scene. We use two main methods (Sequencer & Omni.Anim.People) to add dynamic objects.

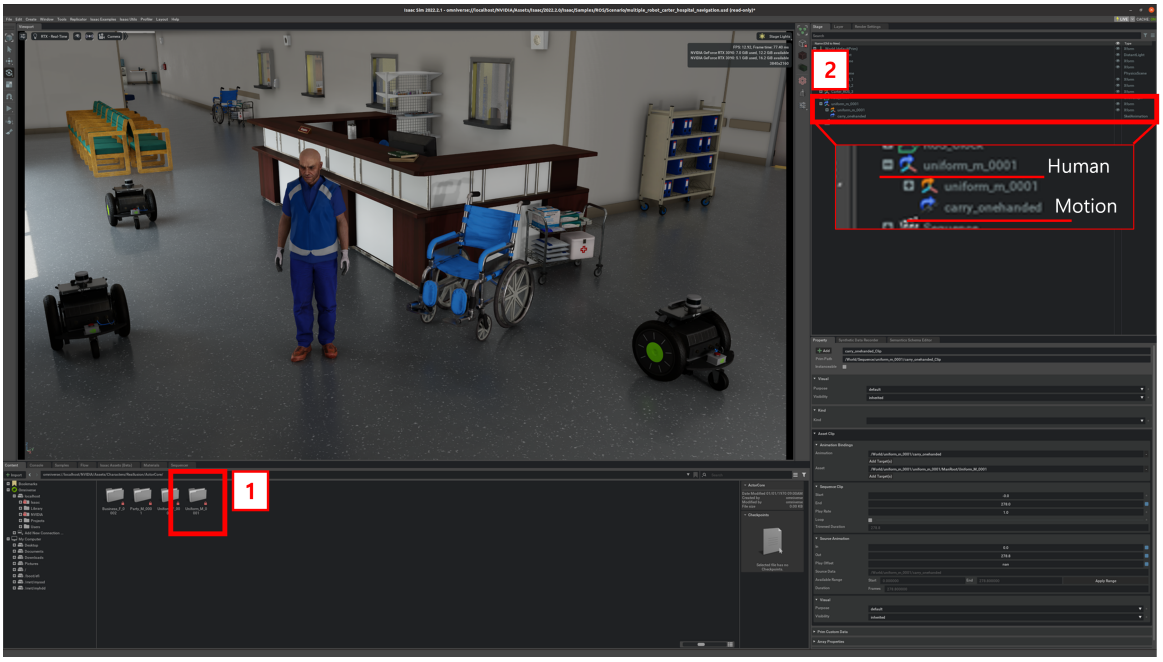
1. **Sequencer**
    a. An extension to animate using only Human and Motion assets.
    b. Characteristics :
        i. Advantage : The operation is simple.
        ii. Disadvantage : In the simulation's timeline, when reaching the end time code and looping back to the start time code, the motion is reset to the beginning. (e.g., People walking around will be moved back into place.)
2. **Omni.Anim.People**
    a. An extension that allows for simulating humans in the simulation.
    b. Characteristics :
        i. *Advantage* : We can create Humans that continue to move over long distances regardless of the simulation's Timeline (start / end time code).
        ii. *Disadvantage* : It is complicated because we need to manually write the file for the plan regarding human movement.

Considering the pros and cons of each method described above, we utilize "**Sequencer**" for generating dynamic humans performing repetitive movements in place, and "**Omni.Anim.People**" for creating dynamic humans that continuously walk around the simulation environment.

In our CSE Dataset, to add an environmentally suitable variety of dynamic objects, we not only used the Human / Motion assets provided by NVIDIA but also directly purchased additional ones from ActorCore for utilization. However, Note that users that they can still perform the following processes using the Human/Motion assets freely provided by NVIDIA. The tutorials for each method are as follows.

2.2.1. Sequencer

1. Open the Isaac Sim.
2. Open Sequencer.
    - Path : (Top bar) > Windows > animation > sequencer
3. Drag the human and motion asset to Stage



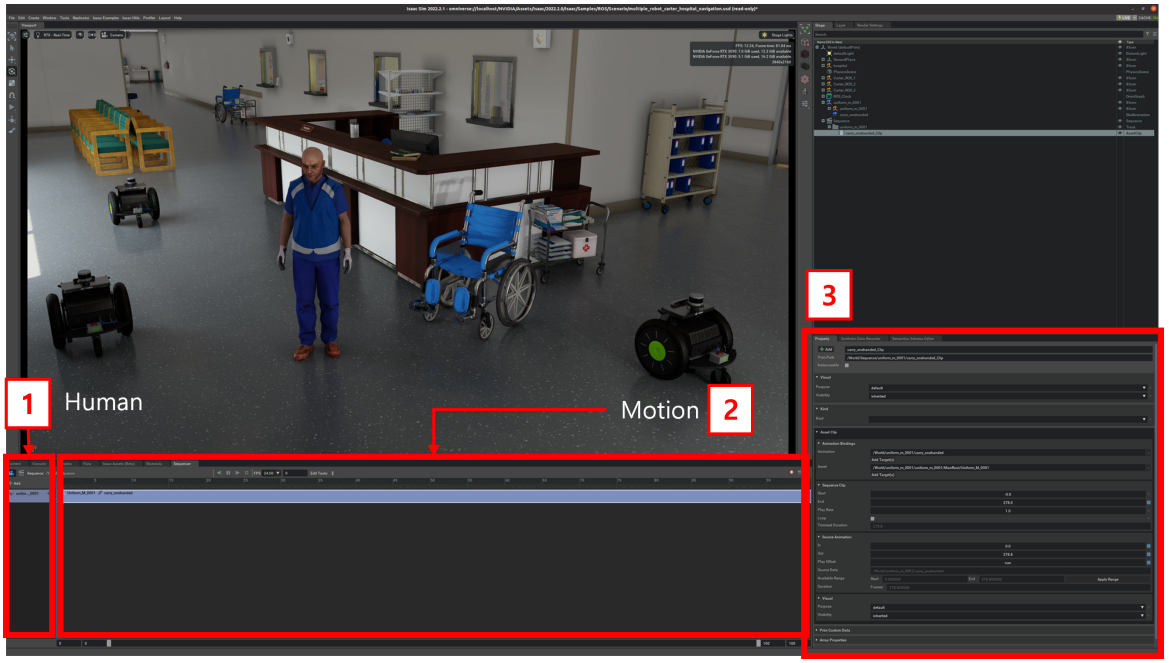
Method to add the human and motion assets to the Stage

- Human / Motion Asset path : (Content) > omniverse://localhost/NVIDIA/Assets/Characters/Reallusion/ActorCore

💡 **Note :**

When you first load a human, it may not be scaled correctly. This can be corrected by adjusting the scale in the Property panel.

4. In Sequencer :



Method to use the Sequencer

1. Drag and drop the (Stage > Human asset) into the left box in the Sequencer
2. Drag and drop the (Stage > Motion asset) into the right box in the Sequencer

3. When you click the Motion asset in Sequencer, you can adjust it in the Property panel. You can clip it or adjust its speed to suit your purposes.

2.2.2. Omni.Anim.People

💡 **Note** :

We referred to the "Omni.Anim.People Tutorial". We highly recommend reading through that page. Moreover, all files used in this process are available on the CSE Dataset project page.

1. Open the Isaac Sim.
2. Enable the extension.
   - Path : (Top bar) > Window > Extensions

     omni.anim.people → enable

     NAVIGATION BUNDLE (BETA) → enable

     NAVIGATION CORE (BETA) → enable

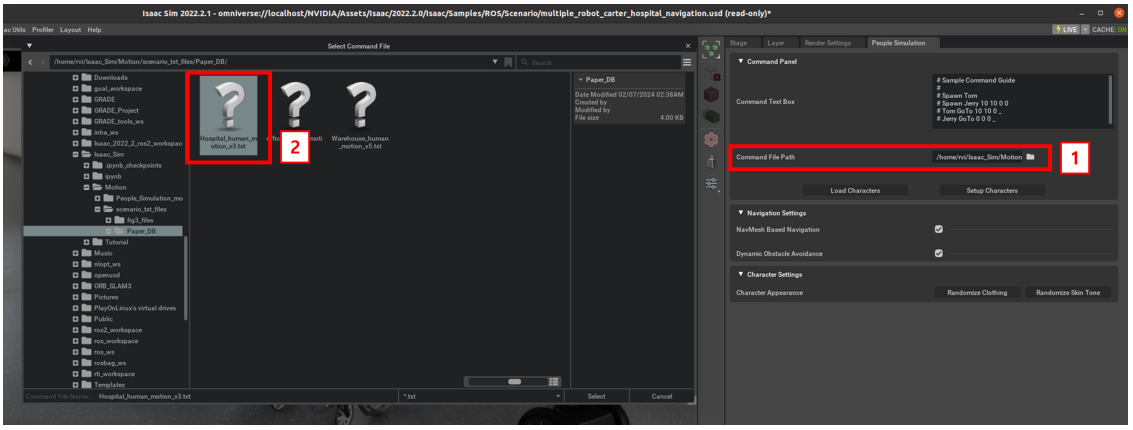
     NAVIGATION UI (BETA) → enable

   💡 **Note :**

   When selecting each extension, choose AUTOLOAD on the right side. By clicking this, the extension will automatically be enabled every time Isaac Sim is launched.
3. Open the People Simulation.

Method to open the “People Simulation (Omni.Anim.People)”

   - Path : (Top bar) > Window > People Simulation
4. In People Simulation :
   a. Load the human motion txt file.

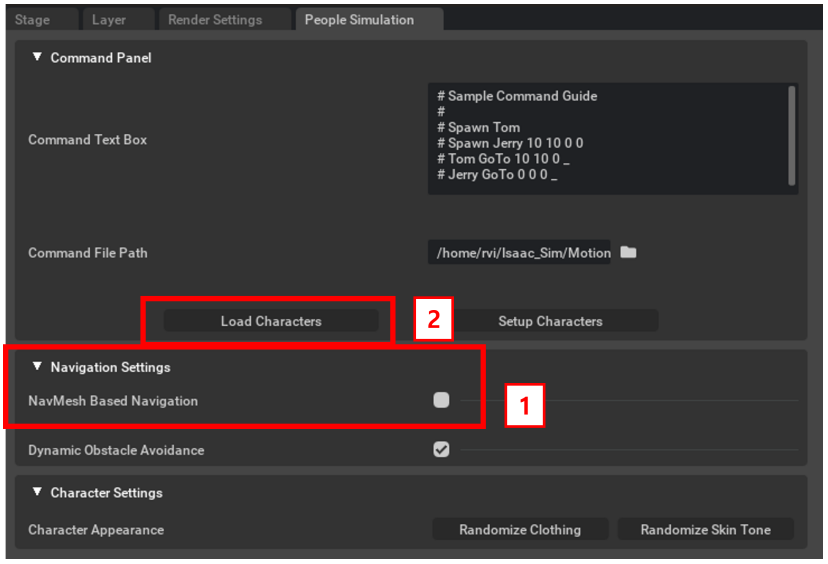


Method to load the human motion txt file

1. Click Command panel > Command File Path
2. Select your txt file.

   We recommend reading (Sec 6.1. Omni.Anim.People Tips) for guidance on generating human motion txt files.

b. Load characters



Method to load characters

1. Uncheck Navigation settings > NavMesh Based Navigation
2. Click Command Panel > Load Characters

   If you want to control the speed of characters, please refer to (Sec 6.1. Omni.Anim.People Tips).

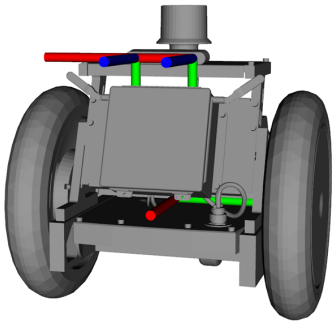
c. Click Command Panel > Setup Characters

d. Play the simulation

3. Robot and Sensor Setup

In this section, we describe all the information related to the robots and sensors used in our CSE dataset. Specifically, we intend to focus on explaining how to publish topics at the desired Hz for the various sensors mounted on each robot using ROS (Robot Operating System).

3.1. Robot and Sensor



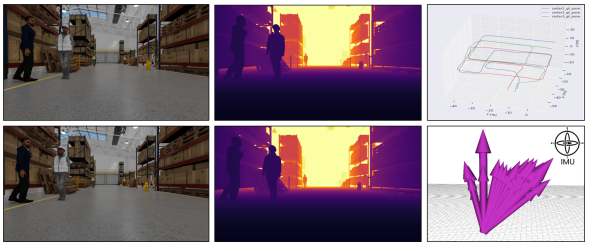
Robot 'Carter'

As a robot platform, we utilize the NVIDIA Carter URDF* provided by Isaac Sim. Carter is a differentiable drive robot with two wheels on each side that is designed for verifying the capabilities of the Isaac SDK. Isaac SDK is intended to develop applications for complicated use cases, such as delivery robots, and the Carter is developed as a delivery robot. Accordingly, we chose Carter as our robot platform for building the C-SLAM dataset tailored for service robots and used a total of three robots.

Therefore, this tutorial will be based on Carter, and the Carter asset is available for free in NVIDIA Isaac Sim.

*** URDF?**

URDF (Unified Robot Description Format) is a standard data format for describing a robot's structure in XML that defines its components, size, shape, position, and joint information.



Examples of acquired sensor data (RGB stereo, Depth stereo, GT poses, and IMU)

For each robot, we attach several types of sensors for perception in service environments. Specifically, the sensor system of each robot is equipped with RGB and depth stereo cameras and an IMU sensor. The configurations for each sensor are as follows in the table below.

TABLE I: **Sensor Configurations.**

Data	Resolution	Rate	Topic Name	Message Type
Stereo left RGB	1280×720	30Hz	`/carter/rgb_left/compressed`	`sensor_msgs/CompressedImage`
Stereo right RGB	1280×720	30Hz	`/carter/rgb_right/compressed`	`sensor_msgs/CompressedImage`
Stereo left Depth	1280×720	30Hz	`/carter/depth_left`	`sensor_msgs/Image`
Stereo right Depth	1280×720	30Hz	`/carter/depth_right`	`sensor_msgs/Image`
IMU	-	120Hz	`/carter/imu`	`sensor_msgs/Imu`
Ground Truth Pose	-	120Hz	`/carter/gt_pose`	`nav_msgs/Odometry`

We generate the CSE dataset using Carter robot equipped with various sensors. Accordingly, this section aims to explain how to acquire the aformentioned diverse sensor data, Ground Truth (GT) pose, and camera intrinsics as ROS topics. These methods are detailed in (Sec 3.2. Sensor Setup).

3.2. Sensor Setup

💡 **Note :**

We utilize the basic scenes (*Hospital* / *Office*) provided by NVIDIA Isaac Sim. In this case, each *Hospital* / *Office* scenes already contains three Carter robots. Additionally, each robot is set up to acquire stereo RGB and stereo depth. Based on this setup, we acquire data by attaching additional a IMU sensor to each robot and modifying the Omnigraph. For the *Warehouse* scene created using the SceneBlox tool, robots are not included in the basic scene. Therefore, we copy the Carter assets created in the *Hospital* / *Office* and utilize them in the Warehouse as well.

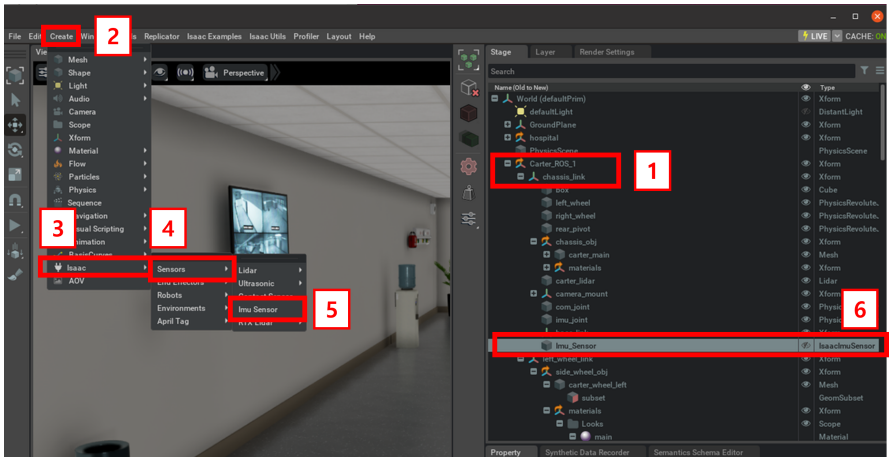
*** Omnigraph ?**

> "OmniGraph is the visual scripting language of Omniverse. It allows the worlds in Omniverse to come alive with behavior and interactivity."
>
> from NVIDIA Omniverse docs

3.2.1. Add the IMU Sensor

The Carter asset provided by NVIDIA Isaac Sim is already set up with stereo RGB and stereo depth cameras. Therefore, we aim to acquire additional sensor data by attaching an IMU Sensor.

1. Open the Isaac Sim.
2. Open the your scene containing the Carter.
3. Attach an IMU sensor to each robot.



1. Click Stage > Carter_ROS_1 > chassis_link

   💡 **Note :**

   You do not need to follow the exact path above. Click on the path where you want to attach the IMU Sensor.

2. Select the IMU Sensor on the top bar

3.2.2. Publishing Data as ROS Topics [Omnigraph]

We aim to save the data written below as a ROS bag file ultimately. Therefore, this section intends to explain how to publish the all data as ROS topics using Omnigraph. In addition, we will also describe how to publish each data at the desired hz. All source code used in this process are available on the CSE dataset project page.

- **Total data List :**

- Stereo RGB (Left / Right)
- Stereo Depth (Left / Right)
- IMU
- Stereo camera intrinsics *(RGB and Depth cameras are same)*
- Ground Truth Pose

1. **IMU**

   💡 **Note :**

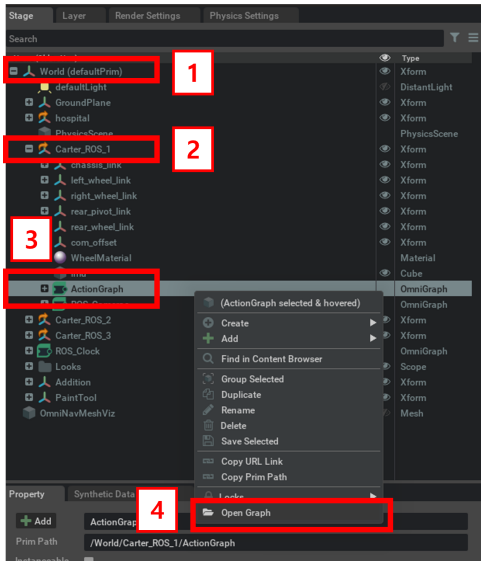
   Before following this process, you must first complete (Sec 3.2.1. Add the IMU Sensor).

   1. IMU hz settings

      💡 **Note :**

      In the CSE Dataset, we aim to acquire IMU sensor data at 120 hz. To achieve this, please follow the (Sec 6.2. IMU hz settings).

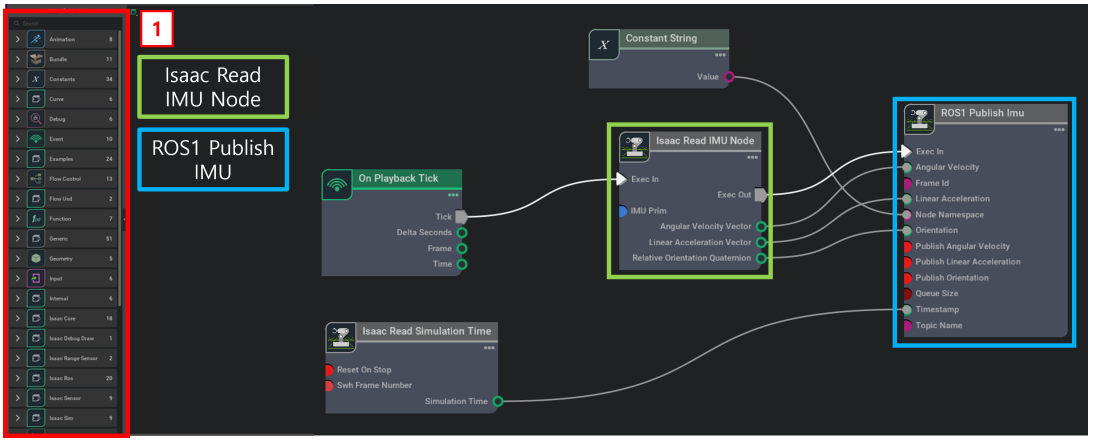
   2. Open the Omnigraph



      1. Click Stage > World > Carter_ROS_1 > ActionGraph > Open Graph

   3. In Omnigraph :

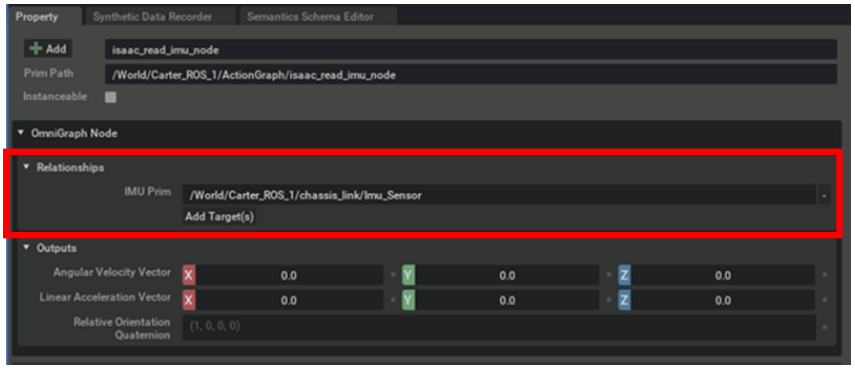
      a. Generate the graph.



1. Search the 2 nodes (Isaac Read IMU Node & ROS1 Publish IMU)
2. Drag and drop the each node into the ActionGraph window.
3. Connect the each node like above figure.

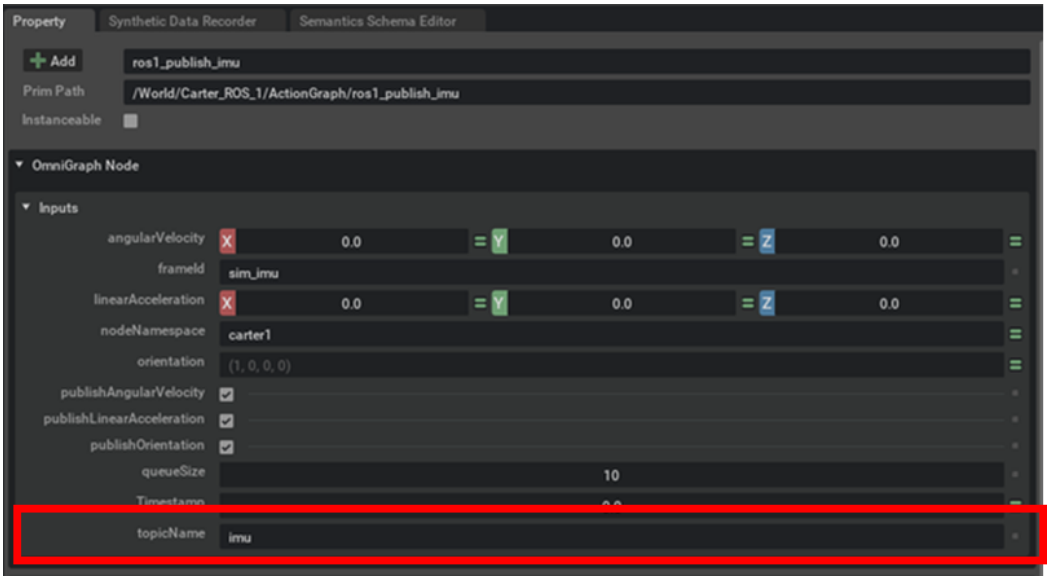
b. Fill in each node.

i. Isaac Read IMU Node



1. Click Add Target
2. Select the IMU sensor for each robot existing on the Stage

ii. ROS1 Publish IMU



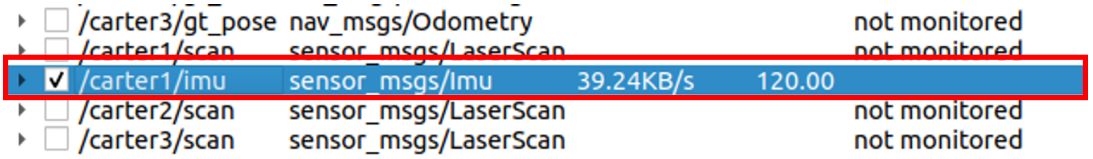
1. Type the topic name

4. Final check

a. Play the simulation.

b. Open the rqt.

c. You can check the imu topic (120hz).



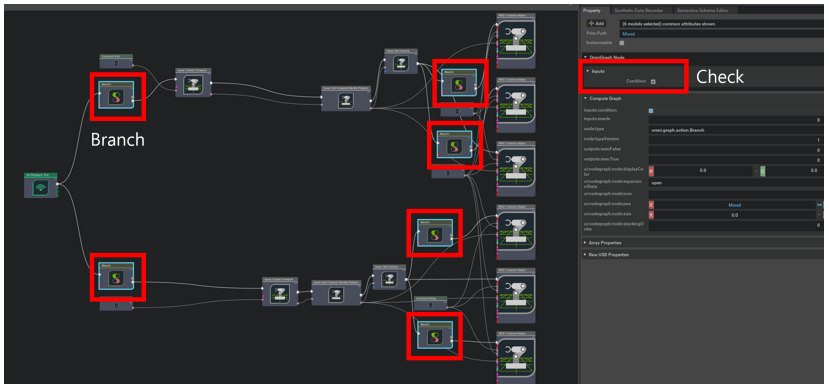
2. **Camera**

   💡 **Note :**

   This process encompasses all aspects related to the stereo RGB camera, stereo depth camera, and camera intrinsics. In the *Hospital* / *Office* basic scenes, stereo RGB/depth cameras and the Omnigraph for them are already set up. Therefore, please note that we use the sensors and Omnigraph provided by default as they are.

   1. Open the Omnigraph.
      a. Click Stage > World > Carter_ROS_1 > ROS_Cameras > Open Graph
   2. In Omnigraph :

      We use the provided Omnigraph as is. However, the branch node that allows turning the camera on/off by default might be unchecked. By checking this, you can publish the image data as ROS topics.



   3. (Option) Image topic compression and hz settings.

      💡 **Note :**

      Due to storage limitations, we compress the image topics and filter it to 30hz. To accomplish this, we implement and utilize the additional ROS nodes. This is not mandatory and if you want to use it, please follow (Sec 6.3 Image topic compression and hz settings).

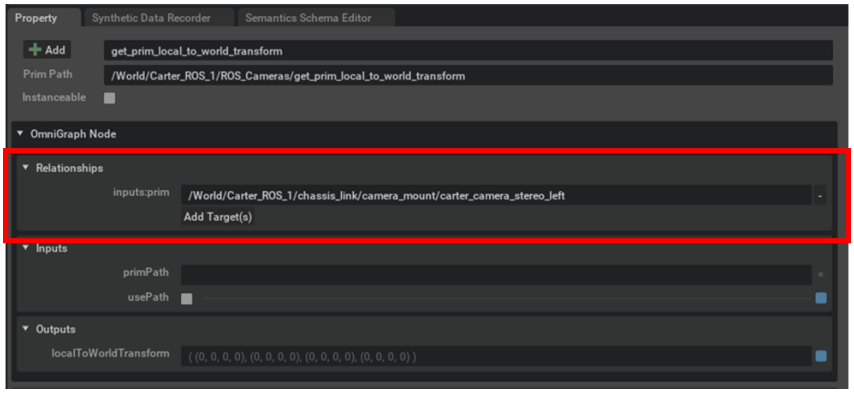
   4. Final check
      a. (Option) Execute the ROS Nodes in terminal.
         i. image_compression
         ii. image_filtering
      b. Play the simulation.
      c. Open the rqt.
      d. You can check the camera topics (RGB/Depth - 30Hz).

3. **GT Poses**
   a. Click Stage > World > Carter_ROS_1 > ROS_Cameras > Open Graph
   b. In Omnigraph :
      i. Generate the graph.

1. Search the nodes and connect them like above figure.

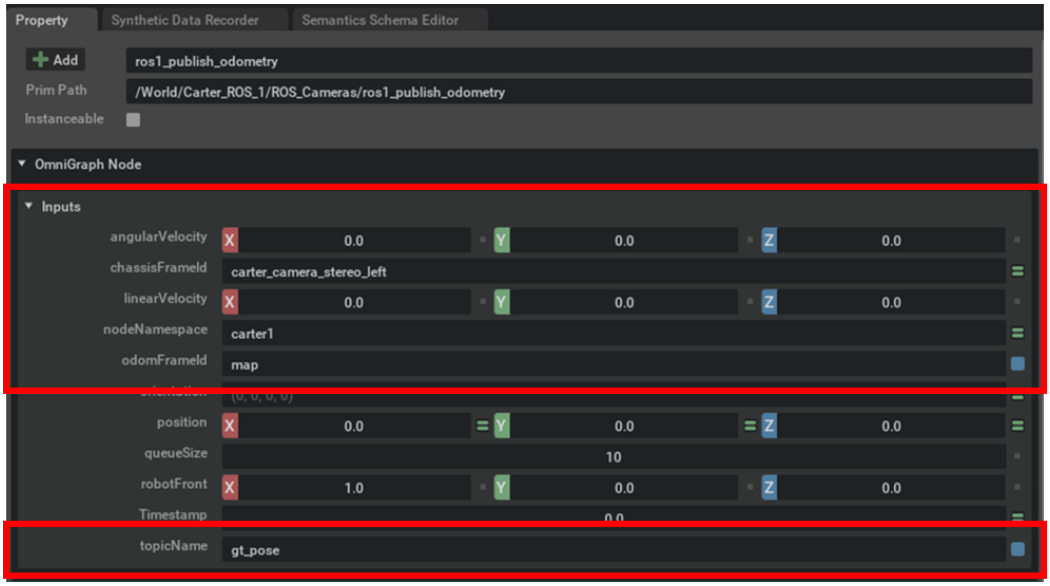
ii. Fill in each node.

1. Get Prim Local to World Transform



We use "carter_camera_stereo_left" as the input to extract the GT Pose based on the left camera of each robot.

2. ROS1 Publish Odometry

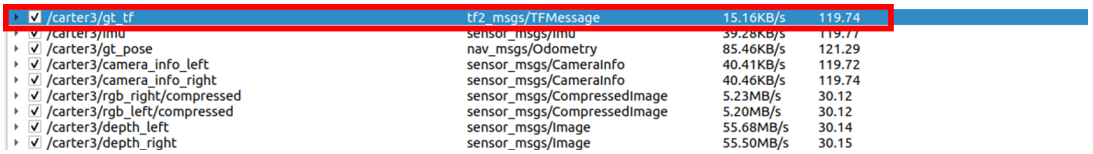


c. Final check

i. Play the simulation.

ii. Open the rqt.

iii. You can check the GT pose topic (120 hz).



4. Robot Navigation

This section describes contents related to the navigation of the robots. We refer to the NVIDIA Omniverse docs (8. Multiple Robot ROS Navigation) and the source code we utilized are available on the CSE dataset project page. Additionally, this process requires some ROS packages that are provided when you download Isaac Sim, so we also explain in detail how to use them.

1. Install the ROS Navigation stack

```
# In our case,
sudo apt-get install ros-noetic-navigation
```

2. Build ROS Workspace

💡 **Note :**

This is explained based on the method and path we used.

1. Copy the ros_workspace.
   - Workspace path : /home/{user}/.local/share/ov/pkg/isaac_sim-2022.2.1/ros_workspace

   We made a copy of the folder to avoid modifying the original.
2. Build the workspace.

```
cd ~/ros_workspace/
catkin_make
source ~/ros_workspace/devel/setup.bash
```

3. Generate the Occupancy map of your own scene.

a. Open the Isaac Sim.

b. Load your own scene.

c. Generate the Occupancy map.

Follow the NVIDIA Omniverse docs (8.2.1. Office Environment Setup).

1. Follow the docs to generate an Occupancy map.
2. Place the generated occupancy map image and yaml file (e.g., carter_hospital2_navigation.png & carter_hospital2_navigation.yaml) inside the map directory located in the carter_2dnav ROS package.

4. Multiple Robot ROS Navigation Setup.

💡 **Note :**

To operate multiple robots in the same environment, please follow the steps below.

1. In `carter_2dnav` ROS package :

   Modify the “multiple_robot_carter_navigation.launch” file.

   - Path : ~/ros_workspace/src/navigation/carter_2dnav/launch/multiple_robot_carter_navigation.launch

```
<!-- Edited Hospital Scenario -->
<group if="$(eval arg('env_name') == 'hospital2')" >
  <include file="$(find carter_2dnav)/launch/amcl_robot_individual.launch">
    <arg name="robot_name"      value="$(arg robot1)" />
    <arg name="initial_pose_x"  value="19.0"/>
    <arg name="initial_pose_y"  value="-2.0"/>
    <arg name="initial_pose_a"  value="1.57"/>
  </include>

  <include file="$(find carter_2dnav)/launch/amcl_robot_individual.launch">
    <arg name="robot_name"      value="$(arg robot2)" />
    <arg name="initial_pose_x"  value="-29.0"/>
    <arg name="initial_pose_y"  value="20.0"/>
    <arg name="initial_pose_a"  value="0.0"/>
  </include>

  <include file="$(find carter_2dnav)/launch/amcl_robot_individual.launch">
    <arg name="robot_name"      value="$(arg robot3)" />
    <arg name="initial_pose_x"  value="11.0"/>
    <arg name="initial_pose_y"  value="36.0"/>
    <arg name="initial_pose_a"  value="-1.57"/>
  </include>
</group>
```

   1. Write your “env_name”.

      In line 45 (<group if="$(eval arg('env_name') == 'hospital2')" >), ‘hospital2’ is the env_name. Please ensure that the occupancy map image and yaml file names match the specified env_name exactly.

   2. Write your robot’s initial pose.

      Write the initial pose (x, y, angle) of the robots the user have positioned in “initial_pose_x/y/a”.

2. In `isaac_ros_navigation_goal` ROS package :

   💡 **Note :**

   There are primarily two methods to navigate robots in Isaac Sim (RandomGoalGenerator & GoalReader). We use the “GoalReader” method, which allows users to create the robot’s goal points directly, driving the robot along the desired trajectory.

   1. In **assets** folder :

      a. Place the occupancy map png & yaml files you created.

      b. Place your own goal point txt files that you have created.

         To create goal points txt files for each robot, follow (Sec 6.4 Generate the robot goal points txt file).

   2. In **launch** folder :

      We create a new launch file for each environment like below.

      1. Write environment-specific launch files (e.g., isaac_ros_navigation_goal_hospital.launch)


```
<launch>
    <group ns="carter1">
        <param name="map_yaml_path" value="$(find isaac_ros_navigation_goal)/assets/carter_hospital2_navigation.yaml" />
        <param name="iteration_count" type="int" value="14" />
        <param name="goal_generator_type" type="str" value="GoalReader" />
        <param name="action_server_name" type="str" value="move_base" />
        <param name="obstacle_search_distance_in_meters" type="double" value="0.1" />
        <param name="goal_text_file_path" value="$(find isaac_ros_navigation_goal)/assets/goals_red.txt" />
        <rosparam param="initial_pose">[19.0, -2.0, 0.25, 0, 0, 0.70711, 0.70711] </rosparam>
        <node name="set_navigation_goal" pkg="isaac_ros_navigation_goal" type="set_goal.py" output="screen"/>
    </group>
    <group ns="carter2">
        <param name="map_yaml_path" value="$(find isaac_ros_navigation_goal)/assets/carter_hospital2_navigation.yaml" />
        <param name="iteration_count" type="int" value="22" />
        <param name="goal_generator_type" type="str" value="GoalReader" />
        <param name="action_server_name" type="str" value="move_base" />
        <param name="obstacle_search_distance_in_meters" type="double" value="0.1" />
        <param name="goal_text_file_path" value="$(find isaac_ros_navigation_goal)/assets/goals_green.txt" />
        <rosparam param="initial_pose">[-29.0, 20.0, 0.25, 0, 0, 0.0, 1.0] </rosparam>
        <node name="set_navigation_goal" pkg="isaac_ros_navigation_goal" type="set_goal.py" output="screen"/>
    </group>
    <group ns="carter3">
        <param name="map_yaml_path" value="$(find isaac_ros_navigation_goal)/assets/carter_hospital2_navigation.yaml" />
        <param name="iteration_count" type="int" value="19" />
        <param name="goal_generator_type" type="str" value="GoalReader" />
        <param name="action_server_name" type="str" value="move_base" />
        <param name="obstacle_search_distance_in_meters" type="double" value="0.1" />
        <param name="goal_text_file_path" value="$(find isaac_ros_navigation_goal)/assets/goals_blue.txt" />
        <rosparam param="initial_pose">[11.0, 36.0, 0.25, 0, 0, -0.70711, 0.70711] </rosparam>
        <node name="set_navigation_goal" pkg="isaac_ros_navigation_goal" type="set_goal.py" output="screen"/>
    </group>
</launch>
```

Refer to the image above and modify the launch file to suit user's needs.

If you've followed the steps above, you're all set to drive your robots.

5. Robot Navigation
    a. In terminal,

```
# In terminal 1,
roscore

# In terminal 2,
rosparam set use_sim_time true
```

    b. Open the Isaac Sim.
    c. Load your own scene.
    d. Play the simulation
    e. In a new terminal,

```
# In terminal 1,
source ~/ros_workspace/devel/setup.bash
roslaunch carter_2dnav multiple_robot_carter_navigation.launch env_name:=hospit
al2

# In terminal 2.
source ~/ros_workspace/devel/setup.bash
roslaunch isaac_ros_navigation_goal isaac_ros_navigation_goal_hospital.launch
```

    After a few moments, you can see that the robot is driving to the goal points you specified on Rviz.

5. Data Acquisition Process

This section aims to summarize the entire process of constructing your own data on Isaac Sim. Additionally, it explains the process of finally saving the data as a rosbag file. Please note that all the steps mentioned below are performed using ROS1(Noetic).

1. ROS setup
    a. Execute the roscore
    b. Set the Robot Operating System (ROS) node to use simulation time instead of real time.

```
# Terminal 1
roscore

# Terminal 2
rosparam set use_sim_time true
```

2. Open the Isaac Sim.
3. Open your own scene where both the robot and sensors are already set up.

    ⇒ Follow the section "2.1. Static Scene Generation" to generate the static scene.

    ⇒ Follow the section "3. Robot and Sensor Setup" to set up the robots and sensors.
4. (Option) Set up dynamic objects.

    ⇒ Follow the section "2.2. Dynamic Objects Handling".

    If you are using "Omni.Anim.People" method, you need to perform section "2.2.2. Omni.Anim.People" every time you load a scene.
5. Play the simulation.
6. Running ROS Navigation.

    ⇒ Follow the section "4. ROS Navigation".
7. Save the data.
    a. Save all the topics you want in a rosbag.

```
# In Terminal (Example),
cd /path/to/save/

# rosbag record -O {file_name.bag} {topic names}
rosbag record -O name.bag /carter1/rgb_left/compressed /carter2/rgb_left/compresse
d /carter3/rgb_left/compressed
```

8. Finish !

6. ⭐ Tips ⭐

This section summarizes our troubleshooting of the issues we encountered when building the cse dataset. These are the methods that we finally found empirically or through the NVIDIA Forums. Please note that there may be other ways to do this.

6.1. Omni.Anim.People Tips

In this section, tips for utilizing Omni.Anim.People are provided. The method for writing the human motion txt file was referenced from the tutorial page of official Isaac Sim. However, the method for controlling the human speed is an empirically found way. There may be other methods to adjust the speed, but we have used the

following approach to control the speed of human motion. All the files we used in these courses are available on the CSE-dataset project page. The exact path of files on our project page is ./tutorial/human_motion_txt/.

1. **How to generate the human motion txt file ❓**

```
Spawn F_Medical_01
F_Medical_01 GoTo 10 0 0 _

Spawn F_Business_02 10 0 0 0
F_Business_02 GoTo 0 0 0 _
```

💡 **Note :**

All Human names (e.g., F_Medical_01, F_Business_02) must be designated as the folder names in the path below for the corresponding human asset to be loaded.

- Path : (Content) > omniverse://localhost/NVIDIA/Assets/Isaac/2022.2.0/Isaac/People/Characters/

1. Firstly, you need to write "Spawn {character name} {translation} {rotation}" like below.

```
Spawn {character name : "F_Business_02"} {translation : 0 0 0} {rotation : 0}
Spawn F_Business_02 0 0 0 0
```

Through this, the specified human assets are loaded according to the desired location/rotation.

2. After that, you can freely wirte the desired actions (GoTo / Idle / LookAround) according to the desired translation / rotation.

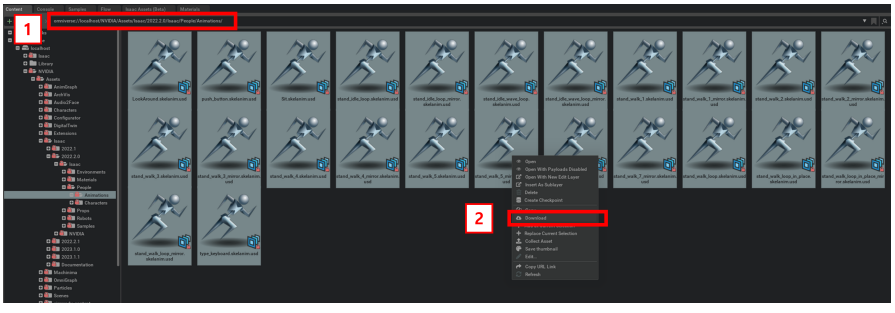
We higly recommend referring to the official Isaac Sim tutorial page for writing Actions.

2. **How to control the human speed ❓**

💡 **Note :**

This process should be done after clicking (Load Characters) and before clicking (Setup Characters).

1. Generate the new motion usd files with adjusted human motion speed.
   a. Download the original motion usd files (24 files) in your local.
      - Motion usd file path : (Content) > omniverse://localhost/NVIDIA/Assets/Isaac/2022.2.0/Isaac/People/Animations/



   b. Convert the format of the downloaded USD file from .usd to .usda.
      i. Install usdcat
      ii. Convert .usd to .usda file (human-readable UTF-8 text) using usdcat tool.

```
usdcat -o example.usda example.usd
```

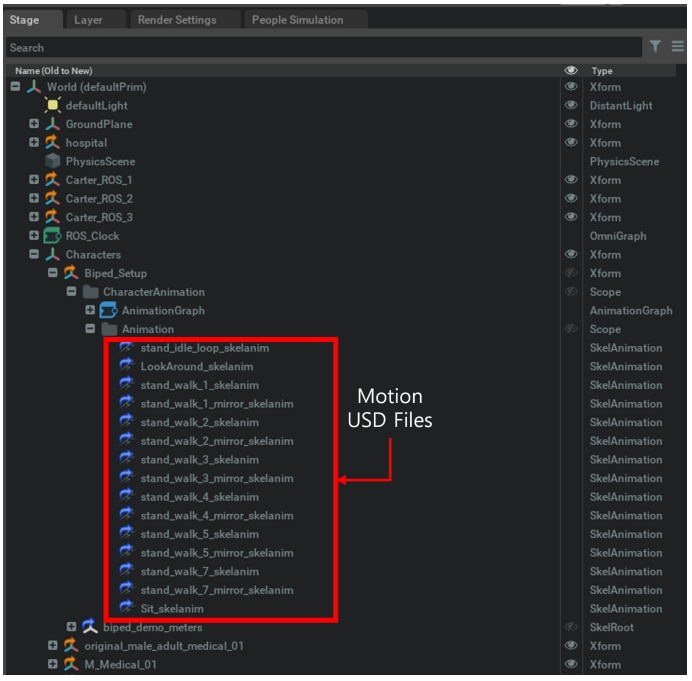
c. Open the motion file (.usda).

d. Modify the variables (framesPerSecond, timeCodesPerSecond) in your motion file.

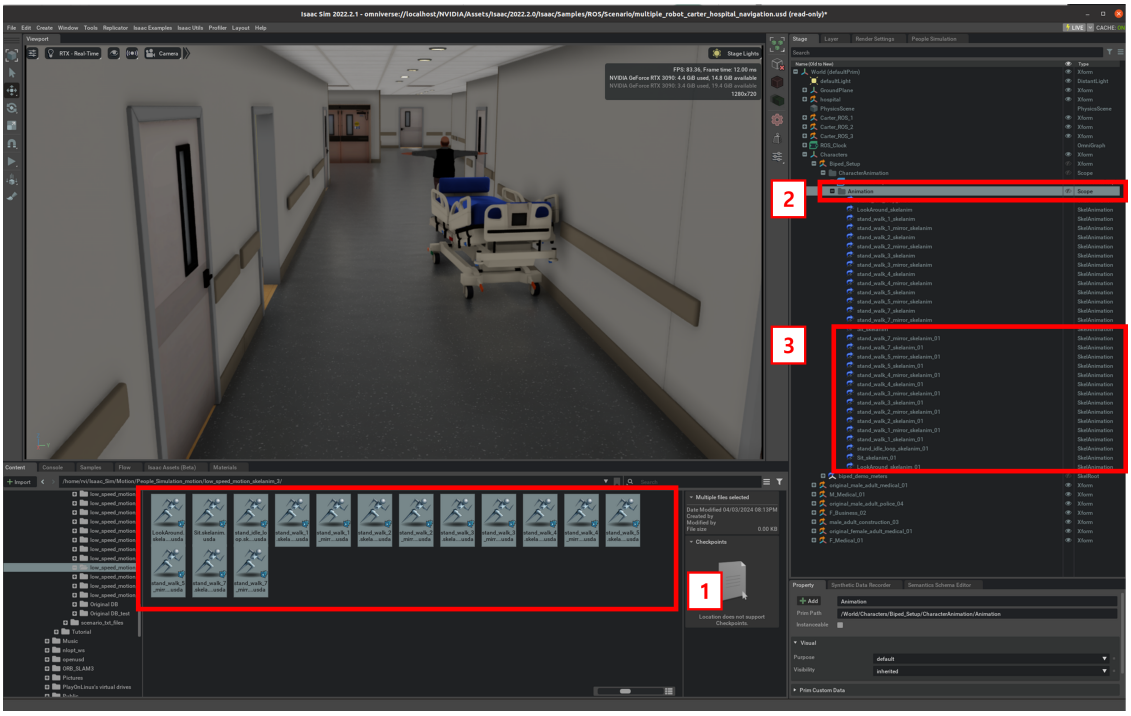
By default, it will be set to "30". Decreasing this number will reduce the speed of human motion.

2. Click Stage > Characters > Biped_Setup > CharacterAnimation > Animation

   The default motion usd files will be contained as shown in the picture below.

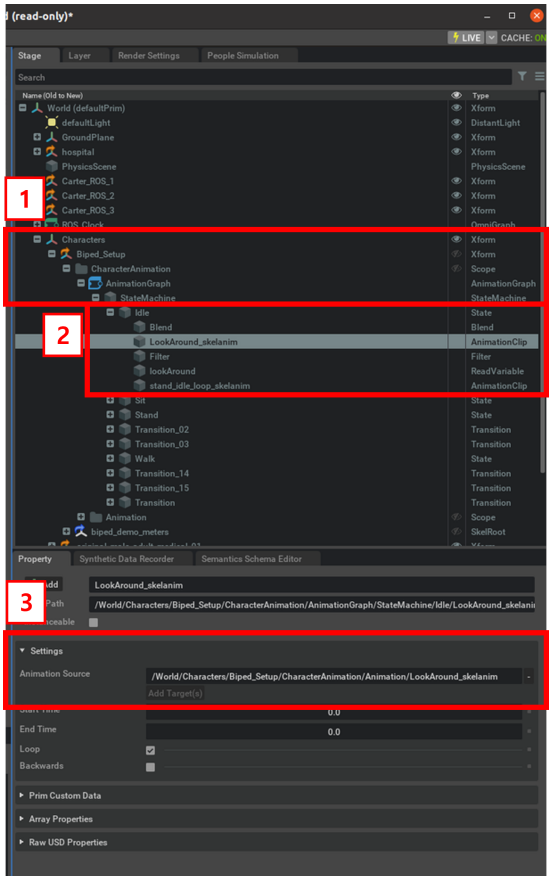


3. Load the new USD files.



1. Drag and drop the new USD files into Stage > Characters > Biped_Setup > CharacterAnimation > Animation.
2. You can see the USD files ending with '_01' under Animation (see Third red box)

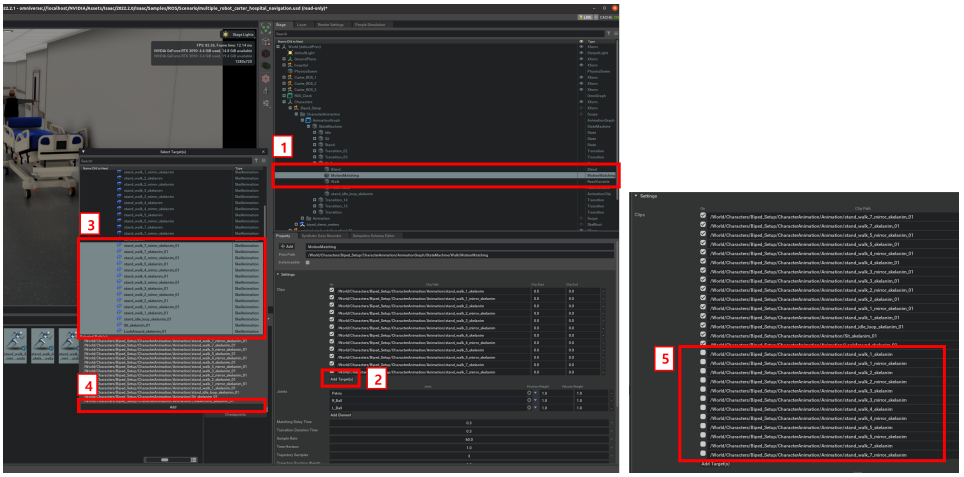
4. Convert from the default motion USD files to new motion USD files.



1. Click Stage > Characters > Biped_Setup > CharacterAnimation > AnimationGraph > StateMachine.
2. Click on the Action state (Idle, Sit, Stand, Walk) in the StateMachine.
3. For each Action state, change the Animation Source in the AnimationClip type that exist in the table below to the new USD file.

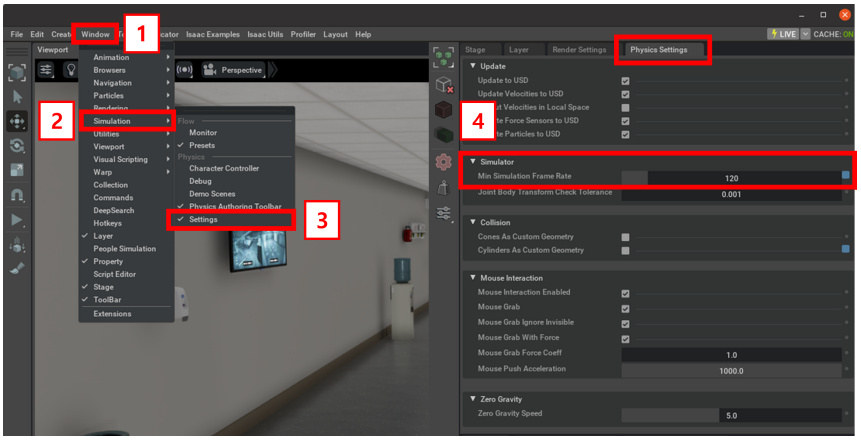
Idle	Sit	Stand	Walk
LookAround_skelanim	sit_skelanim	sit_skelanim	MotionMatching > Clips (All)
stand_idle_skelanim	stand_idle_loop_skelanim		stand_idle_loop_skelanim

However, for Walk > MotionMatching, please modify it through the process below.



6.2. IMU hz settings

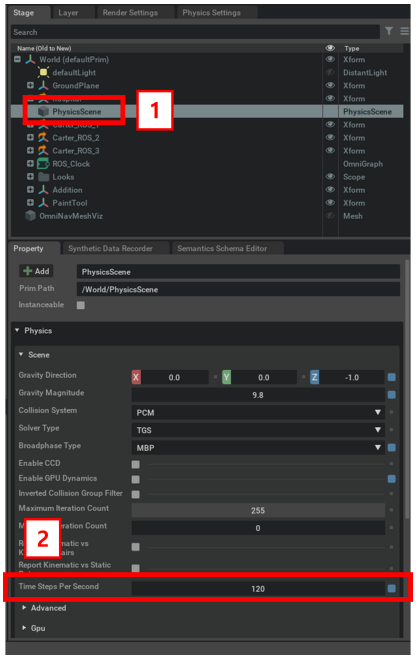
1. Modify the Min Simulation Frame Rate to 120



1. Click (Top bar) > Window > Simulation > Settings
2. In “Physics Settings”, write 120 in the Min Simulation Frame Rate.

   (You can write the Hz you desire.)

2. Modify the Time Steps Per Second to 120.



1. Click Stage > PhysicsScene
2. In “PhysicsScene” Property panel, write 120 in the Time Steps Per Second.

   (You can write the Hz you desire.)

Through the above process, you can set all data Topics being published in the simulation to 120.

6.3 Image topic compression and hz settings

1. Make your own workspace and move to it.

```
# Example
mkdir -p ~/cse_workspace
cd ~/cse_workspace
```

2. Git clone src folder to your workspace

   You can find the src folder in CSE dataset project page.

```
# In cse_worksace folder,
git clone https://github.com/vision3d-lab/CSE-Dataset.git
```

3. Build the workspace

```
cd ~/cse_workspace/
catkin_make
source ~/cse_workspace/devel/setup.bash
```

4. Execute the ROS Nodes.

   a. Image compression Node

   In the code, rename the topic to suit your needs.

```
rosrun image_compression image_compressors.py
```

b. Image topic filtering Node

In the code, change it to fit your desired image.

```
rosrun image_filtering filter_topics.py
```

6.4 Generate the robot goal points txt file

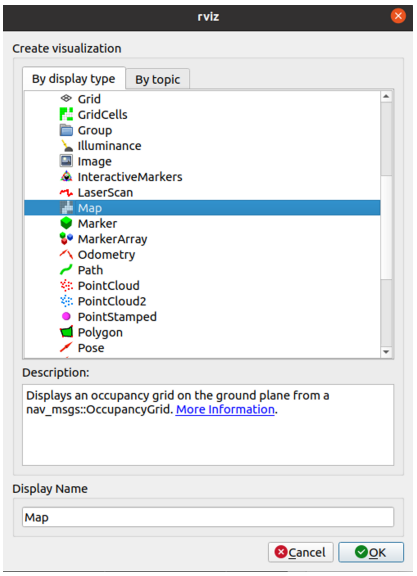
💡 **Note :**

When you follow below methods, just do “roscore” in terminal. Don’t do “rosparam set use_sim_time true” in terminal.

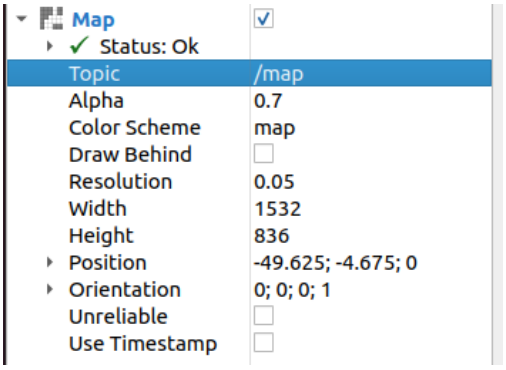
1. Follow the (Sec 6.3. Image topic compression and hz settings) to generate the ROS workspace.
2. Turn on Rviz.
3. Execute map_server (Publish map_server)
   a. Place the occupancy map data (.png & .yaml) under the ‘map’ folder.
      - Map path : ~/goal_workspace/src/goal_to_txt/map/
   b. Execute the following command.

```
rosrun map_server map_server carter_{env_name}_navigation.yaml
```

4. Load “map” in Rviz.
   a. Click “Add” > “By display type” > “Map”



5. Set topic name to “/map”



6. Execute g2t.

```
rosrun goal_to_txt g2t.py
```

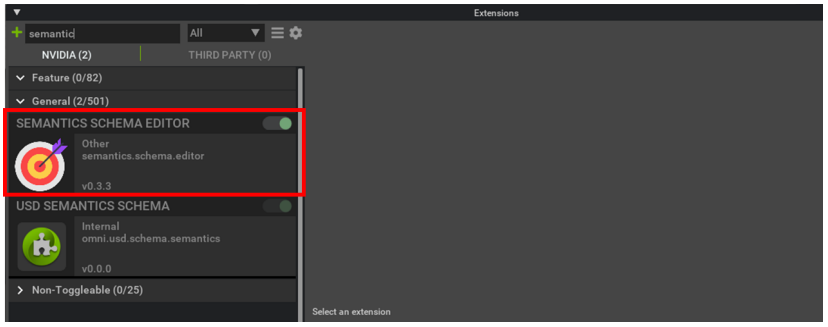
7. Click (Top bar) > Panels > Tool properties
8. Modify 2D Nav Goal > Topic to "/carter_msg_goal".
9. Click "2D Nav Goal" in tool panels.
10. Click goal point where you want.
11. Finish.

    If you clicked on all the goal points, you should have created a "goals.txt" under the path where you ran g2t.py. Place it under the path (~/ros_workspace/src/navigation/isaac_ros_navigation_goal/assets/). However, you need to rename it to the txt file you specified inside the {isaac_ros_navigation_goal_hospital}.launch file.

7. Ground Truth Labels

In this section, we describe the methods for generating the semantic segmentation and bounding box labels in NVIDIA Isaac Sim. Note that we used a method that allows us to extract all GT labels(i.e., semantic, bboxes) at once by manually labeling each asset.

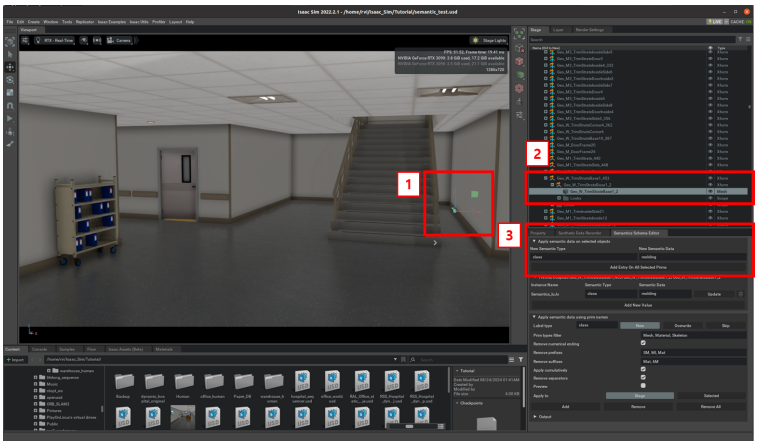
1. Open the Isaac Sim.
2. In a new stage, open your own scene.
3. In extension, turn on "Semantic Schema Editor".



4. Add the class label for each asset.
    a. In the viewport, click the asset you want to add a label to.

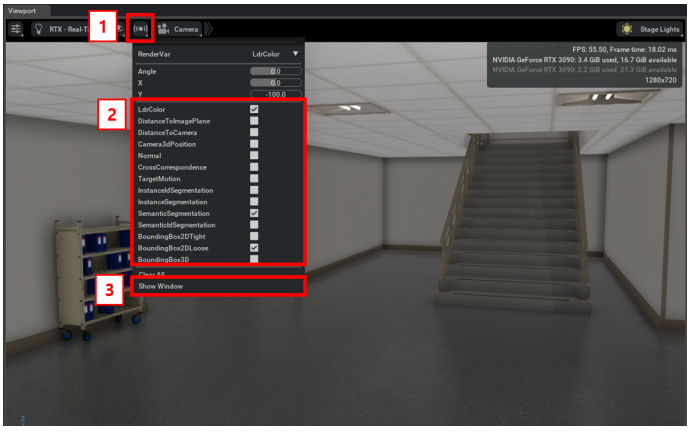
b. Click the ‘mesh’ inside the selected asset in the Stage.

c. In the “Semantic Schema Editor” > “Apply semantic data on selected objects”, write the following :

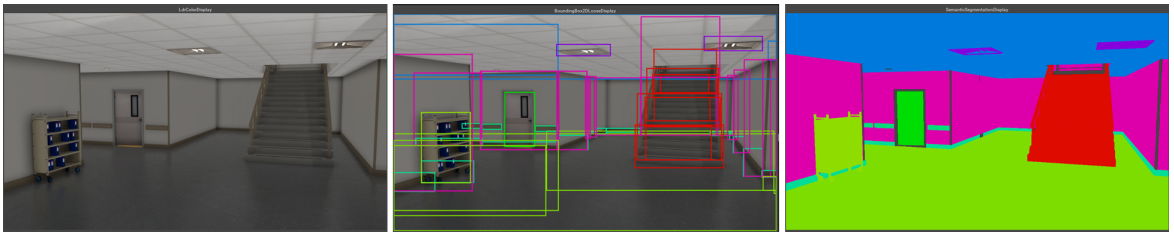


1. New Semantic Type : class
2. New Semantic Data : {class name}
3. Click “Add Entry All Selected Prims”

5. Visualization (Check results)



1. Select the items you want to check the results for, and then click “Show Window”.
2. You can see the results as shown in the image below.



(Left) : RGB Image (Mid) : Bounding boxes (Right) : Semantic segmentation

References

1. NVIDIA Isaac Sim, https://developer.nvidia.com/isaac-sim
2. NVIDIA Omniverse, https://www.nvidia.com/en-us/omniverse/
3. NVIDIA Omniverse docs, https://docs.omniverse.nvidia.com/isaacsim/latest/index.html
4. Isaac Sim installation in workstation, https://docs.omniverse.nvidia.com/isaacsim/latest/installation/install_workstation.html
5. SceneBlox tool, https://docs.omniverse.nvidia.com/isaacsim/latest/manual_replicator_sceneblox.html#isaac-sim-app-manual-replicator-sceneblox
6. Replicator SceneBlox tool tutorial, https://docs.omniverse.nvidia.com/isaacsim/latest/replicator_tutorials/tutorial_replicator_sceneblox.html
7. ActorCore, https://actorcore.reallusion.com/
8. Omni.Anim.People, https://docs.omniverse.nvidia.com/isaacsim/latest/features/warehouse_logistics/ext_omni_anim_people.html
9. NVIDIA Carter, https://docs.nvidia.com/isaac/archive/2020.2/doc/tutorials/carter_hardware.html
10. Multiple Robot ROS Navigation, https://omniverse-content-production.s3-us-west-2.amazonaws.com/Assets/Isaac/Documentation/Isaac-Sim-Docs_2022.2.0/app_isaacsim/app_isaacsim/tutorial_ros_multi_navigation.html