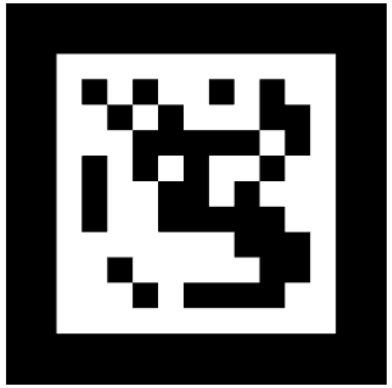
Robotic Software
Lezione 10

ISAAC ROS
ISAAC SIM

AR Markers

- AR marker stands for:
  - Augmented reality markers
- Any object that can be placed in a scene to provide a fixed point of reference of position and/or scale
- With one camera we can not estimate the distance between the objects and the camera
  - We can solve this problem with triangulation procedures
- Defined patterns can be used to get multiple information
  - Object type – an id defines the marker
  - Position (3D)
  - Orientation (3D)



AR Markers

- Information
  - Where the robot is: which floor? Which room? (like QRCodes)
  - Where to push the button to open the elevator
- Elaboration is based on a binary classification of the squares of the marker
- Common problems:
  - Precision
  - Low-speed image elaboration
  - Marker detection
- Different software
  - Each software has its marker library

Isaac_ros_apriltag

- Install the apriltag NVIDIA packages
  - cd ~/workspaces/isaac_ros-dev/src
  - git clone https://github.com/NVIDIA-ISAAC-ROS/isaac_ros_apriltag
  - git clone https://github.com/NVIDIA-ISAAC-ROS/isaac_ros_nitros
  - git clone https://github.com/NVIDIA-ISAAC-ROS/isaac_ros_image_pipeline
- Get the bagfile
  - cd ~/workspaces/isaac_ros-dev/src/isaac_ros_apriltag
  - git lfs pull -X "" -I "resources/rosbags/quickstart.bag“
- Compile the package
  - cd ~/workspaces/isaac_ros-dev/src/isaac_ros_common
  - ./scripts/run_dev.sh
  - cd /workspaces/isaac_ros-dev
  - colcon build
  - source install/setup.bash

Isaac_ros_apriltag

- Test the apriltag package
- Terminal 1: launch the apriltag package
  - $ cd ~/workspaces/isaac_ros-dev/src/isaac_ros_common
  - $ ./scripts/run_dev.sh
  - $ ros2 launch isaac_ros_apriltag isaac_ros_apriltag.launch.py
- Terminal 2: start the bagfile
  - $ cd ~/workspaces/isaac_ros-dev/src/isaac_ros_common
  - $./scripts/run_dev.sh
  - $ ros2 bag play --loop src/isaac_ros_apriltag/resources/rosbags/quickstart.bag
- Terminal 3: see the output
  - $ cd ~/workspaces/isaac_ros-dev/src/isaac_ros_common
  - $./scripts/run_dev.sh
  - $ ros2 topic echo /tag_detections
  - $ rviz2 -> check the tfs
- Terminal 4:
  - Use rqt:
    - rqt -> visualization -> image

Isaac_ros_apriltag

- Apriltag input

ROS Parameter	Type	Default	Description
`size`	`double`	`0.22`	The tag edge size in meters, assuming square markers. E.g. `0.22`
`max_tags`	`int`	`64`	The maximum number of tags to be detected. E.g. `64`

ROS Topic	Interface	Description
`image`	sensor_msgs/Image	The input camera stream.
`camera_info`	sensor_msgs/CameraInfo	The input camera intrinsics stream.

ROS Topic	Type	Description
`tag_detections`	isaac_ros_apriltag_interfaces/AprilTagDetectionArray	The detection message array.
`tf`	tf2_msgs/TFMessage	Pose of all detected apriltags(`TagFamily:ID`) wrt to the camera topic frame_id.

Isaac_ros_apriltag

- Launch apriltag
  - remappings=[('/image', '/image_rect')]

Isaac_ros_visual_slam

- SLAM:
  - Estimate the robot’s pose starting from odometry information and sensor information
- GMAPPING is a SLAM ROS1 package performing 2D slam from encoder odometry and laser scanner

Isaac_ros_visual_slam

- This repository provides a ROS2 package that performs stereo visual simultaneous localization and mapping (VSLAM)
- Estimates stereo visual inertial odometry using the Isaac Elbrus GPU-accelerated library
  - It takes in a time-synced pair of stereo images (grayscale) along with respective camera intrinsics to publish the current pose of the camera relative to its start pose
- Elbrus is based on two core technologies: Visual Odometry (VO) and Simultaneous Localization and Mapping (SLAM).
- Visual SLAM is a method for estimating a camera position relative to its start position
- This method has an iterative nature
  - At each iteration, it considers two consequential input frames (stereo pairs)
  - On both the frames, it finds a set of keypoints
  - Matching keypoints in these two sets gives the ability to estimate the transition and relative rotation of the camera between frames

Isaac_ros_visual_slam

- Simultaneous Localization and Mapping is a method built on top of the VO predictions
- It aims to improve the quality of VO estimations by leveraging the knowledge of previously seen parts of a trajectory
- It detects if the current scene was seen in the past and runs an additional optimization procedure to tune previously obtained poses.
- Along with visual data, Elbrus can optionally use Inertial Measurement Unit (IMU) measurements
- It automatically switches to IMU when VO is unable to estimate a pose; for example, when there is dark lighting or long solid featureless surfaces in front of a camera
- Elbrus allows for robust tracking in various environments and with different use cases: indoor, outdoor, aerial, HMD, automotive, and robotics.

Isaac_ros_visual_slam

- Download the packages
  - cd ~/workspaces/isaac_ros-dev/src
  - git clone https://github.com/NVIDIA-ISAAC-ROS/isaac_ros_visual_slam
- Get the bagfile
  - cd ~/workspaces/isaac_ros-dev/src/isaac_ros_visual_slam
  - git lfs pull -X "" -I isaac_ros_visual_slam/test/test_cases/rosbags/
- Compile the workspace
  - cd ~/workspaces/isaac_ros-dev/src/isaac_ros_common
  - ./scripts/run_dev.sh
  - cd /workspaces/isaac_ros-dev
  - colcon build
  - source install/setup.bash

Isaac_ros_visual_slam

- Start the nodes
- Terminal 1:
  - $ ros2 launch isaac_ros_visual_slam isaac_ros_visual_slam.launch.py
- Terminal 2:
  - $ source /workspaces/isaac_ros-dev/install/setup.bash
  - $ rviz2 -d src/isaac_ros_visual_slam/isaac_ros_visual_slam/rviz/default.cfg.rviz
- Terminal 3:
  - $ source /workspaces/isaac_ros-dev/install/setup.bash
  - $ ros2 bag play src/isaac_ros_visual_slam/isaac_ros_visual_slam/test/test_cases/rosbags/small_pol_test/
- Terminal 4:
  - $ ros2 topic echo visual_slam/tracking/vo_pose_covariance

Isaac_ros_pose_estimation

- This packages performs object pose estimation
- Starting from the model of an object, estimate its pose
  - Detect the object in the scene
  - Understand how it is placed
  - Estimate its pose in the 3D world

- This repository provides NVIDIA GPU-accelerated packages for 3D object pose estimation
- Using a deep learned pose estimation model and a monocular camera it can estimate the 6DOF pose of a target object
- We need also the DNN interface

- Download the packages
  - $ cd ~/workspaces/isaac_ros-dev/src
  - $ git clone https://github.com/NVIDIA-ISAAC-ROS/isaac_ros_pose_estimation
  - $ git clone https://github.com/NVIDIA-ISAAC-ROS/isaac_ros_dnn_inference

Isaac_ros_pose_estimation

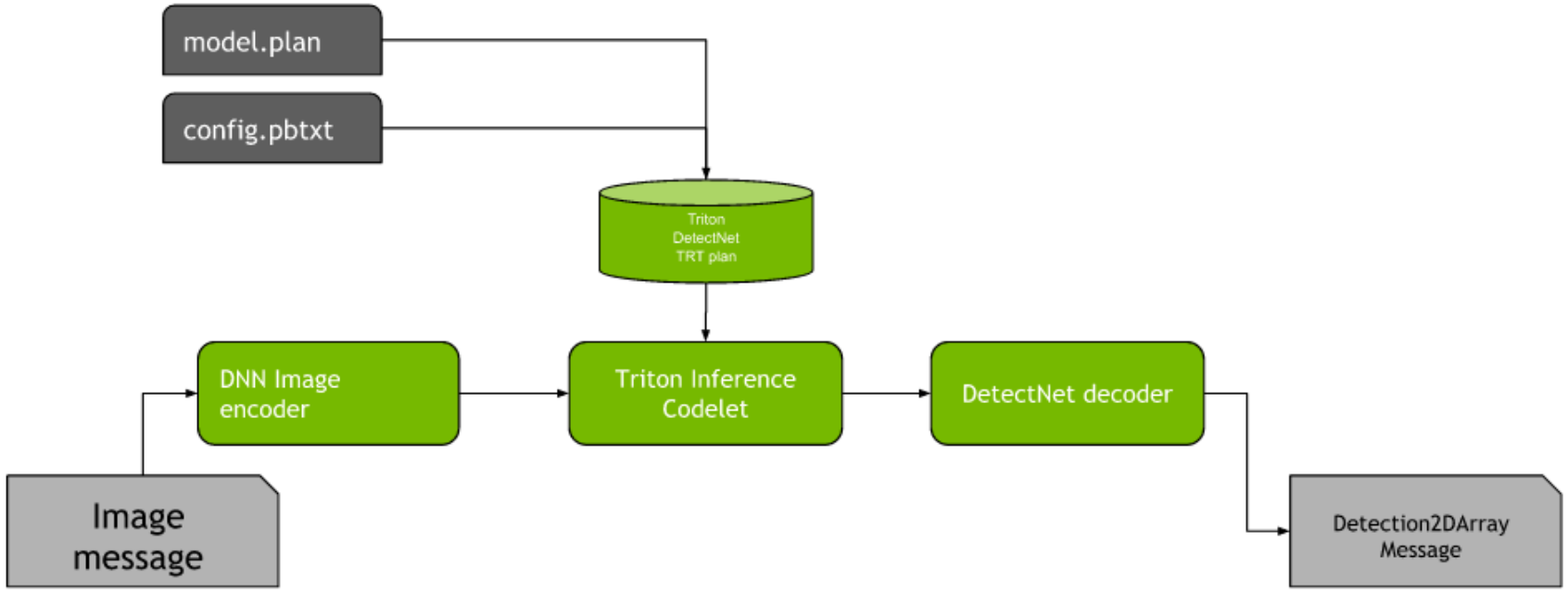
- Get the bagfile
  - $ cd ~/workspaces/isaac_ros-dev/src/isaac_ros_pose_estimation
  - $ git lfs pull -X "" -I "resources/rosbags/“
- Compile the workspace
  - $ cd ~/workspaces/isaac_ros-dev/src/isaac_ros_common
  - $ ./scripts/run_dev.sh
- Training the mode
  - mkdir -p /tmp/models/
  - cd ~/Downloads
  - Use the Ketchup.pth from the DOPE object repo (https://drive.google.com/drive/folders/1DfoA3m_Bm0fW8tOWXGVxi4ETlLEAgmcg)
- Copy the model into the docker container
  - docker cp Ketchup.pth isaac_ros_dev-x86_64-container:/tmp/models
- Convert the model
  - python3 /workspaces/isaac_ros-dev/src/isaac_ros_pose_estimation/isaac_ros_dope/scripts/dope_converter.py --format onnx --input /tmp/models/Ketchup.pth
- Compile the workspace
  - colcon build
  - source install/setup.bash

Isaac_ros_pose_estimation

- Run the package
- Terminal 1:
  - $ ros2 launch isaac_ros_dope isaac_ros_dope_tensor_rt.launch.py model_file_path:=/tmp/models/Ketchup.onnx engine_file_path:=/tmp/models/Ketchup.plan
- Terminal 2:
  - $ ros2 bag play -l src/isaac_ros_pose_estimation/resources/rosbags/dope_rosbag/
- Terminal 3:
  - $ ros2 topic echo /poses
  - $ rviz2
  - Then click on the Add button, select By topic and choose PoseArray under /poses
  - Finally, change the display to show an axes by updating Shape to be Axes, as shown in the screenshot below
  - Make sure to update the Fixed Frame to camera.

Isaac_ros_object_detection

- Object detection
  - This package provides a GPU-accelerated package for object detection based on DetectNet
  - Using a trained deep-learning model and a monocular camera, the isaac_ros_detectnet package can detect objects of interest in an image and provide bounding boxes.
  - DetectNet is similar to other popular object detection models such as YOLOV3



Isaac_ros_object_detection

- Get the package
  - $ cd ~/workspaces/isaac_ros-dev/src
  - $ git clone https://github.com/NVIDIA-ISAAC-ROS/isaac_ros_object_detection
- Get the bagfile
  - $ cd ~/workspaces/isaac_ros-dev/src/isaac_ros_object_detection/isaac_ros_detectnet
  - $ git lfs pull -X "" -I "resources/rosbags"
- Compile the workspace
  - $ cd ~/workspaces/isaac_ros-dev/src/isaac_ros_common
  - $ ./scripts/run_dev.sh
  - $ cd /workspaces/isaac_ros-dev
  - $ colcon build
  - $ source install/setup.bash

Isaac_ros_object_detection

- Train the model
  - $ cd /workspaces/isaac_ros-dev/src/isaac_ros_object_detection/isaac_ros_detectnet
  - $ ./scripts/setup_model.sh --height 632 --width 1200 --config-file resources/quickstart_config.pbtxt
- Launch the detector
  - $ ros2 launch isaac_ros_detectnet isaac_ros_detectnet_quickstart.launch.py
- Visualize and validate the output of the package in the rqt_image_view window. After about a minute, your output should look like this:

Isaac_ros_object_detection

- Input:
  - Params define the object to detect

`label_list`	`string[]`	`{"person", "bag", "face"}`	The list of labels. These are loaded from labels.txt(downloaded with the model)

ROS Topics Subscribed

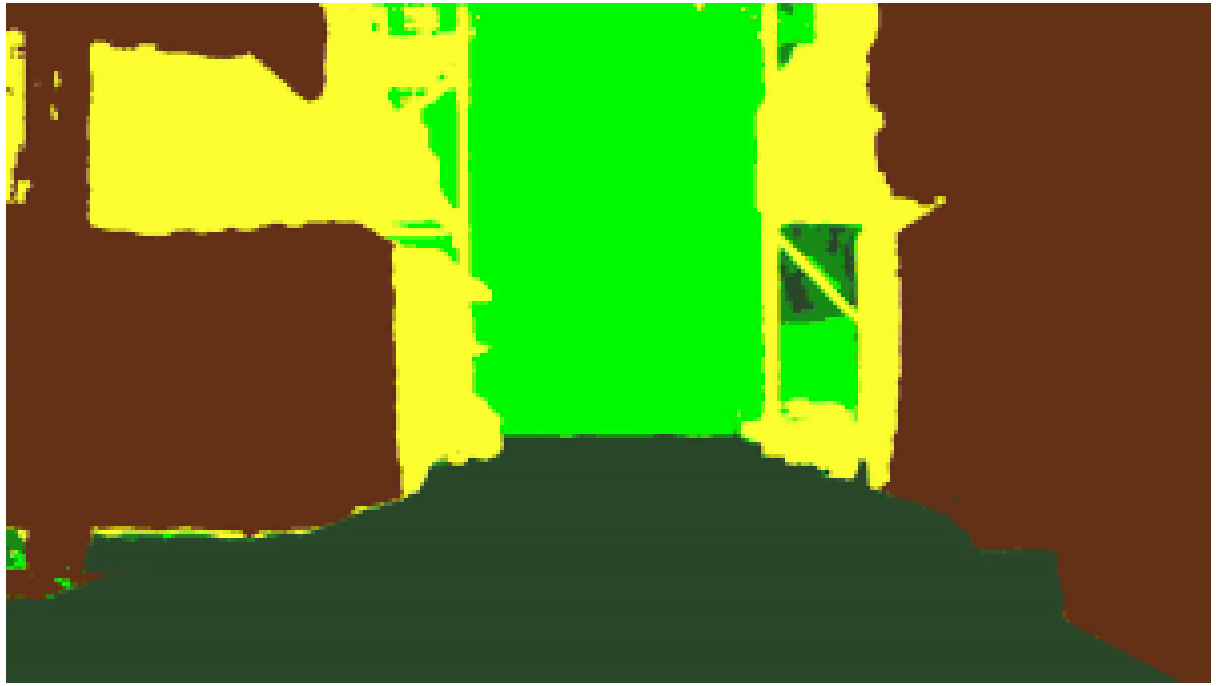
ROS Topic	Interface	Description
`tensor_sub`	isaac_ros_tensor_list_interfaces/TensorList	The tensor that represents the inferred aligned bounding boxes.

ROS Topics Published

ROS Topic	Interface	Description
`detectnet/detections`	vision_msgs/Detection2DArray	Aligned image bounding boxes with detection class.

Isaac_ros_proximity_segmentation

- Proximity segmentation predicts freespace from the ground plane
- It leverages this functionality to produce an occupancy grid that indicates freespace in the neighborhood of the robot
- This camera-based solution offers a number of appealing advantages over traditional 360 lidar occupancy scanning including better detection of low-profile obstacles



Isaac_ros_proximity_segmentation

- Get the package
  - $ cd ~/workspaces/isaac_ros-dev/src
  - $ git clone https://github.com/NVIDIA-ISAAC-ROS/isaac_ros_proximity_segmentation
- Get the bagfile
  - cd ~/workspaces/isaac_ros-dev/src/isaac_ros_proximity_segmentation
  - git lfs pull -X "" -I "resources/rosbags/bi3dnode_rosbag"
- Compile the workspace
  - cd ~/workspaces/isaac_ros-dev/src/isaac_ros_common
  - ./scripts/run_dev.sh
- Train the model
  - mkdir -p /tmp/models/bi3d
  - cd /tmp/models/bi3d
  - wget 'https://api.ngc.nvidia.com/v2/models/nvidia/isaac/bi3d_proximity_segmentation/versions/2.0.0/files/featnet.onnx’
  - wget 'https://api.ngc.nvidia.com/v2/models/nvidia/isaac/bi3d_proximity_segmentation/versions/2.0.0/files/segnet.onnx'

Isaac_ros_proximity_segmentation

  - $ /usr/src/tensorrt/bin/trtexec --saveEngine=/tmp/models/bi3d/bi3dnet_featnet.plan --onnx=/tmp/models/bi3d/featnet.onnx --int8
  - $ /usr/src/tensorrt/bin/trtexec --saveEngine=/tmp/models/bi3d/bi3dnet_segnet.plan --onnx=/tmp/models/bi3d/segnet.onnx --int8
- Compile the workspace
  - $ cd /workspaces/isaac_ros-dev
  - $ colcon build
  - $ source install/setup.bash
- Run the example
  - Terminal 1:
    - $ ros2 launch isaac_ros_bi3d_freespace isaac_ros_bi3d_freespace.launch.py featnet_engine_file_path:=/tmp/models/bi3d/bi3dnet_featnet.plan segnet_engine_file_path:=/tmp/models/bi3d/bi3dnet_segnet.plan max_disparity_values:=10
  - Terminal 2:
    - $ ros2 bag play --loop src/isaac_ros_proximity_segmentation/resources/rosbags/bi3dnode_rosbag
  - Terminal 3:
  - $ rviz2
    - In the left pane, click the Add button, then select By topic followed by Map to add the occupancy grid.

Isaac_ros_image_segmentation

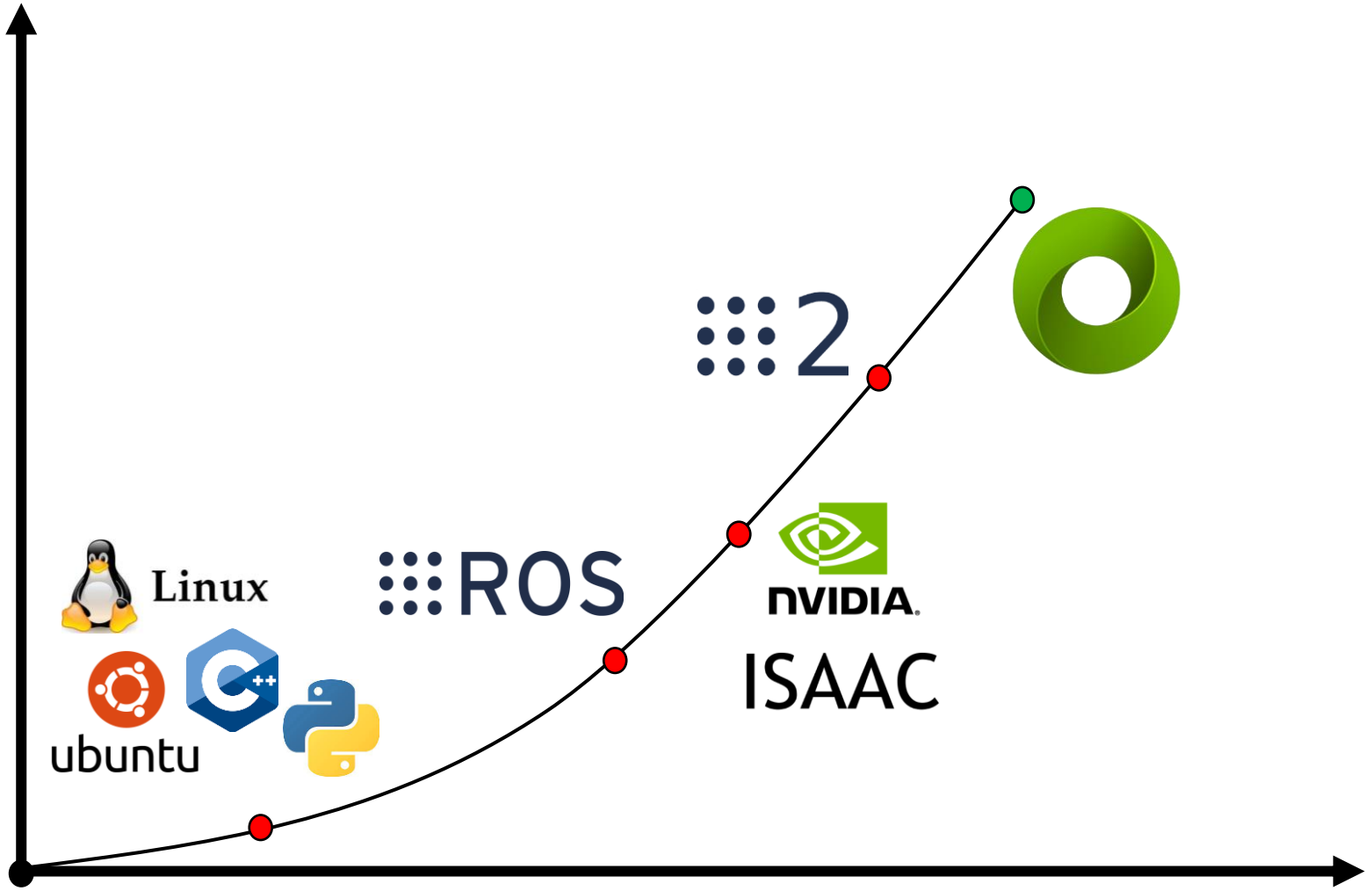
- This repository provides NVIDIA GPU-accelerated packages for semantic image segmentation
- Using a deep learned U-Net model, such as PeopleSemSegnet, and a monocular camera, it can generate an image mask segmenting out objects of interest
- Get the package
  - $ cd ~/workspaces/isaac_ros-dev/src
  - $ git clone https://github.com/NVIDIA-ISAAC-ROS/isaac_ros_image_segmentation
- Get the bagfile:
  - $ cd ~/workspaces/isaac_ros-dev/src/isaac_ros_image_segmentation
  - git lfs pull -X "" -I "resources/rosbags/"
- Get and train the model:
  - cd ~/workspaces/isaac_ros-dev/src/isaac_ros_common
  - ./scripts/run_dev.sh
  - mkdir -p /tmp/models/peoplesemsegnet_shuffleseg/1
  - cd /tmp/models/peoplesemsegnet_shuffleseg
  - wget https://api.ngc.nvidia.com/v2/models/nvidia/tao/peoplesemsegnet/versions/deployable_shuffleseg_unet_v1.0/files/peoplesemsegnet_shuffleseg_etlt.etlt
  - wget https://api.ngc.nvidia.com/v2/models/nvidia/tao/peoplesemsegnet/versions/deployable_shuffleseg_unet_v1.0/files/peoplesemsegnet_shuffleseg_cache.txt

Isaac_ros_image_segmentation

- $ /opt/nvidia/tao/tao-converter -k tlt_encode -d 3,544,960 -p input_2:0,1x3x544x960,1x3x544x960,1x3x544x960 -t int8 -c peoplesemsegnet_shuffleseg_cache.txt -e /tmp/models/peoplesemsegnet_shuffleseg/1/model.plan -o argmax_1 peoplesemsegnet_shuffleseg_etlt.etlt
- $ cp /workspaces/isaac_ros-dev/src/isaac_ros_image_segmentation/resources/peoplesemsegnet_shuffleseg_config.pbtxt /tmp/models/peoplesemsegnet_shuffleseg/config.pbtxt

- Compile the workspace
  - $ cd /workspaces/isaac_ros-dev
  - $ colcon build
  - $ source install/setup.bash
- Launch the package
  - Terminal 1:
    - $ ros2 launch isaac_ros_unet isaac_ros_unet_triton.launch.py model_name:=peoplesemsegnet_shuffleseg model_repository_paths:=['/tmp/models'] input_binding_names:=['input_2:0'] output_binding_names:=['argmax_1'] network_output_type:='argmax’
  - Terminal 2:
    - $ ros2 bag play -l src/isaac_ros_image_segmentation/resources/rosbags/unet_sample_data/
  - Terminal 3:
  - $ ros2 run rqt_image_view rqt_image_view
  - Then inside the rqt_image_view GUI, change the topic to /unet/colored_segmentation_mask to view a colorized segmentation mask.

ISAAC SIM



ISAAC SIM

- NVIDIA Omniverse Isaac Sim is a robotics simulation toolkit for the NVIDIA Omniverse platform
- Isaac Sim has essential features for building virtual robotic worlds and experiments
- It provides researchers and practitioners with the tools and workflows they need to create robust, physically accurate simulations and synthetic datasets
- Isaac Sim supports navigation and manipulation applications through ROS/ROS2
- It simulates sensor data from sensors such as RGB-D, Lidar, and IMU for various computer vision techniques such as domain randomization, ground-truth labeling, segmentation, and bounding boxes

ISAAC SIM

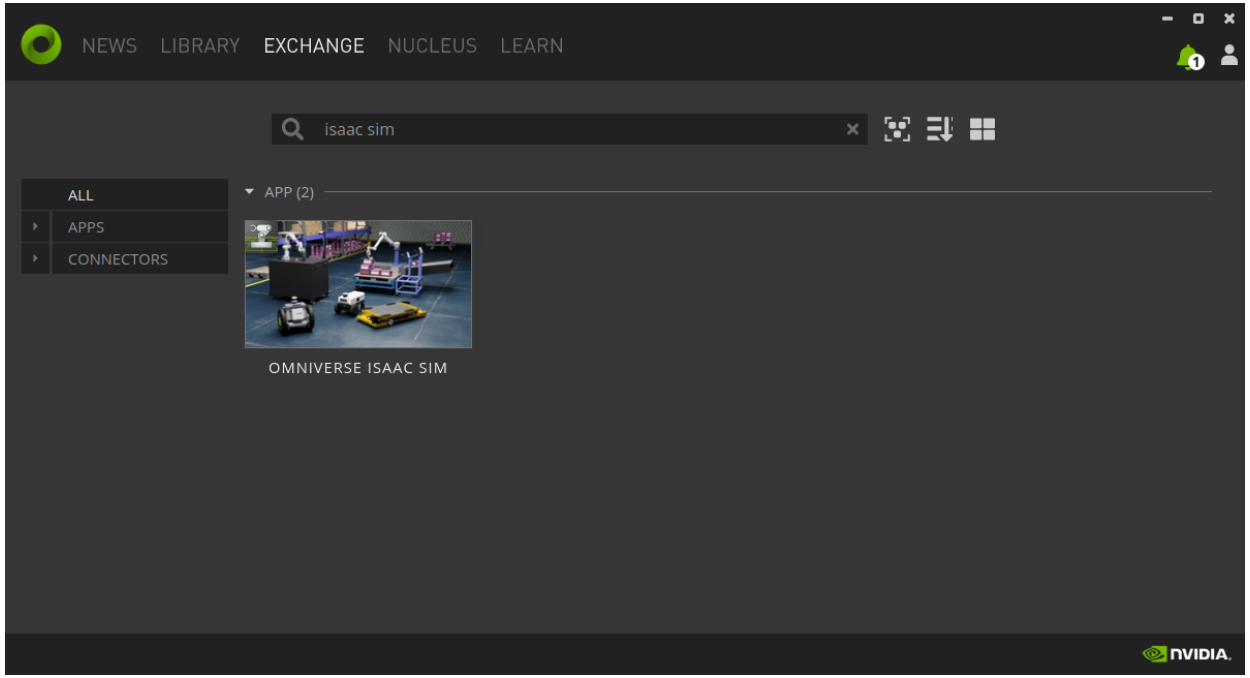
- Omniverse Isaac Sim uses the NVIDIA Omniverse Kit SDK, a toolkit for building native Omniverse applications and microservices
- Omniverse Kit provides a wide variety of functionality through a set of light-weight plugins
- Plugins are authored with C interfaces for persistent API compatibility; however, a Python interpreter is also provided for accessible scripting and customization
- Omniverse Isaac Sim uses the USD interchange file format to represent scenes
- Universal Scene Description (USD) is an easily extensible, open-source 3D scene description and file format developed by Pixar for content creation and interchange among different tools
- Because of its power and versatility, USD is being adopted widely, not only in the visual effects community, but also in architecture, design, robotics, manufacturing, and other disciplines

ISAAC SIM

- Omniverse Isaac Sim uses NVIDIA Omniverse Nucleus to access content such as USD files for environments and robots
- Omniverse Nucleus services allow a variety of client applications, renderers, and microservices to share and modify representations of virtual worlds in Omniverse Isaac Sim
  - Nucleus operates under a publish/subscribe model
  - Subject to access controls, Omniverse clients can publish modifications to digital assets and virtual worlds to the Nucleus Database or subscribe to their changes
  - Changes are transmitted in real-time between connected applications
  - Digital assets can include geometry, lights, materials, textures and other data that describe virtual worlds and their evolution through time
- This allows a variety of Omniverse-enabled client applications (Apps, Connectors, and others) to share and modify authoritative representations of virtual worlds

Installation

- ISAAC Sim can be installed in different ways
- Omniverse launcher
  - Omniverse Isaac Sim can be found and installed on the Exchange tab in the Omniverse Launcher. To simplify the process, enter “isaac sim” in the search bar
  - The launcher is supported by different operating systems
    - Windows
    - Linux
  - With the launcher you can install the streaming client to run the Isaac sim remotely (on a server, on an amazon web service, …)



Installation

- ISAAC Sim can be installed in different ways
- ISAAC Sim can be installed on a docker
  - To keep Isaac Sim configuration and data persistent when running in a container, use the flags below when running the docker container
- sudo docker run --name isaac-sim --entrypoint bash -it --gpus all -e "ACCEPT_EULA=Y" --network=host \
  -v ~/docker/isaac-sim/cache/kit:/isaac-sim/kit/cache/Kit:rw \
  -v ~/docker/isaac-sim/cache/ov:/root/.cache/ov:rw \
  -v ~/docker/isaac-sim/cache/pip:/root/.cache/pip:rw \
  -v ~/docker/isaac-sim/cache/glcache:/root/.cache/nvidia/GLCache:rw \
  -v ~/docker/isaac-sim/cache/computecache:/root/.nv/ComputeCache:rw \
  -v ~/docker/isaac-sim/logs:/root/.nvidia-omniverse/logs:rw \
  -v ~/docker/isaac-sim/config:/root/.nvidia-omniverse/config:rw \
  -v ~/docker/isaac-sim/data:/root/.local/share/ov/data:rw \
  -v ~/docker/isaac-sim/documents:/root/Documents:rw \
  nvcr.io/nvidia/isaac-sim:2022.1.1

Installation

- ISAAC Sim can be installed in different ways
- ISAAC Sim can be installed on a docker
  - To keep Isaac Sim configuration and data persistent when running in a container, use the flags below when running the docker container
  - Name: the name of the container
  - Network, host: use the same network of the host computer
  - nvcr.io/nvidia/isaac-sim:2022.1.1: the remote image to download
  - The other flags will use the use Home folder to save the Isaac Sim cache, logs, config and data

Dockers

- For ISAAC ROS, the run_dev.sh script handles the image and container for us
- After run the previous command, a new contained is instantized from the nvcr.io/nvidia/isaac-sim:2022.1.1 image
- See the current dockers:
  - *$ docker ps -a*
    - This command is used to list the containers
    - The first element of the result is the CONTAINER ID
  - *$ docker start CONTAINER ID*
    - This command is used to start the container
    - Starting the container is mandatory to connect to it
  - *$ docker rm CONTAINER ID*
    - This command is used to remove the container
      - Be careful, you will lose all the modification made to the image
  - $ docker exec –it CONTAINER ID bash
    - The exec command runs a new command in a running container
    - Using bash command as argument we open a new terminal in the container

Install ISAAC SIM

- Install ISACC SIM
  - You can access to Isaac Sim with the provided docker
  - $ docker run [OPTIONS] IMAGE [COMMAND] [ARG...]
  - The image must be pulled
    - docker pull nvcr.io/nvidia/isaac-sim:2022.1.1
  - Before to pull the image, we must authenticate into the system generating an NGC (NVIDIA Cloud Computing) API Key
    - Login at https://ngc.nvidia.com/signin/email
    - Username -> Setup -> Get API Key
    - The API Key is used in the authentication
  - Docker login is used to authenticate
    - $ docker login nvcr.io

    Username: $oauthtoken

    Password:

    WARNING! Your password will be stored unencrypted in /home/username/.docker/config.json.

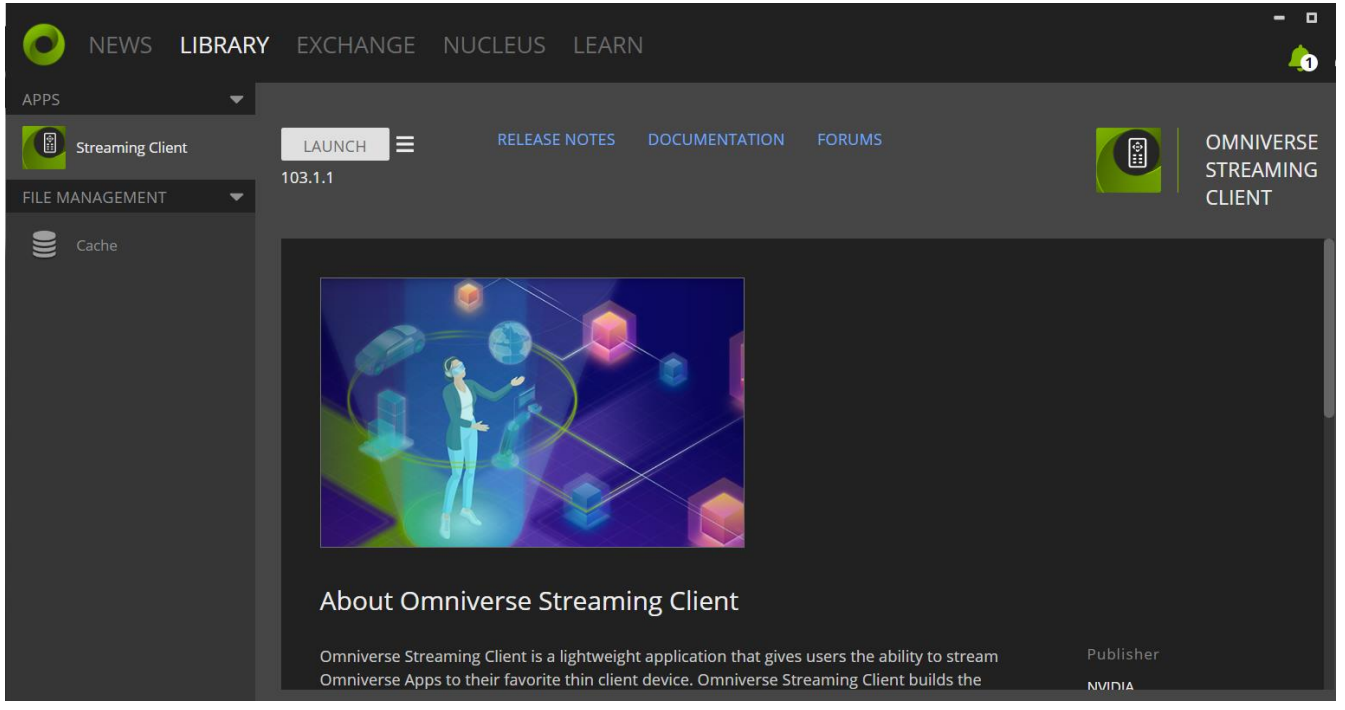
    Configure a credential helper to remove this warning. See

    https://docs.docker.com/engine/reference/commandline/login/#credentials-store

    Login Succeeded
  - Now, the docker can be pulled

Start ISAAC SIM

- After the run of the docker, you are into the container
- To start the simulator, use the following command:
  - *$ ./runheadless.native.sh*
- This will download and launch the Isaac Sim container in headless mode
- Note
  - By using the *-e "ACCEPT_EULA=Y"* flag, you are accepting the NVIDIA Omniverse License Agreement of the image
- See Livestream Clients in our User Guide to connect to Isaac Sim using your desktop. The default livestream client is Omniverse Streaming Client



Install ISAAC SIM

- Docker structure
  - Convenient scripts to launch the simulation
  - Ros1_workspace
  - Ros2_workspace
  - /opt with the common structure for the installation dir with ros ros2 frameworks
- Install ROS1 (we can not use sudo)
  - The docker is set for ubuntu 18.04
    - ROS Melodic supported
  - Setup your sources.list
    - $ sh -c 'echo "deb http://packages.ros.org/ros/ubuntu bionic main" > /etc/apt/sources.list.d/ros-latest.list'
  - Set up your keys
    - $ apt install curl
  - curl -s https://raw.githubusercontent.com/ros/rosdistro/master/ros.asc | apt-key add -
  - Installation
    - $ apt update
    - $ apt install ros-melodic-desktop-full
  - Test the installation
    - $ source /opt/ros/melodic/setup.bash
    - $ roscore

Install ISAAC SIM

- Compile the ROS1 Workspace
  - $ cd /isaac-sim/ros_workspace
  - This workspace already contains different Isaac sim ROS package
  - $ catkin_make

Install ISAAC SIM

- Docker structure
  - Convenient scripts to launch the simulation
  - Ros1_workspace
  - Ros2_workspace
  - /opt with the common structure for the installation dir with ros ros2 frameworks
- Install ROS2 (we can not use sudo)
  - The docker is set for ubuntu 18.04
    - ROS Dashing supported
  - apt update && apt install curl gnupg2
  - curl -sSL https://raw.githubusercontent.com/ros/rosdistro/master/ros.key -o /usr/share/keyrings/ros-archive-keyring.gpg
  - echo "deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/keyrings/ros-archive-keyring.gpg] http://packages.ros.org/ros2/ubuntu bionic main" | sudo tee /etc/apt/sources.list.d/ros2.list > /dev/null
  - apt update
  - apt install ros-dashing-desktop
  - apt install -y python3-pip
  - pip3 install -U argcomplete
  - apt install python3-colcon-common-extensions

Install ISAAC SIM

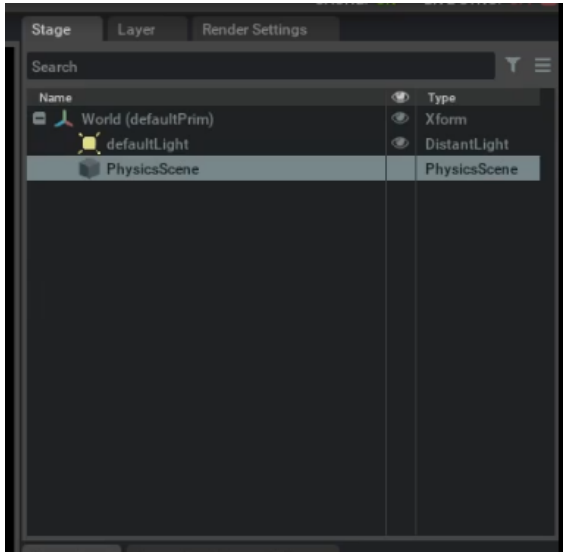
- Test ROS2 installation
  - $ source /opt/ros/dashing/setup.bash
  - $ cd /isaac-sim/ros2_workspace
  - $ colcon build
  - $ ros2 topic list

First steps: add an object in the scene

- Begin by adding a cube to the scene
  - Go to the top Menu Bar and Click **Create > Shapes > Cube**.
  - There should now be a Cube in the center of the Viewport. The Cube is Selected (highlighted in orange), and the Move (W) command is enabled by default.
  - To use any simulation features in Omniverse Isaac Sim, like joints, scripting, or collision meshes, the simulation must be running.
    - Press the Play button in the Toolbar to begin the simulation. At the bottom of the screen, the timeline marker will begin to move, and will loop over the timeline.
    - After pressing it, the Play button turns into a Pause button, which can be used to stop the simulation temporarily but allow it to continue from its current state.
    - While the simulation is running or paused, the Stop button is also visible below the Play/Pause button. The Stop button ends the simulation and resets it, allowing it to be played again from its starting configuration.
    - The simulation’s start/stop progress can also be viewed on the Timeline at the bottom of the default layout.

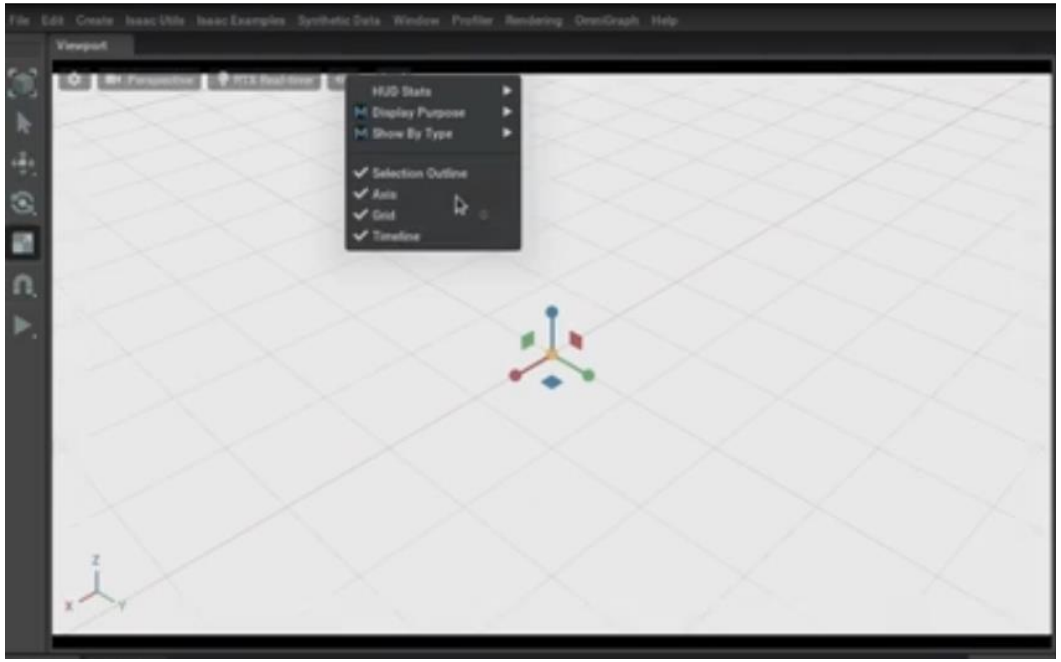
First steps: add physics to the scene

- Physics Scene
  - Physics Scene provides the general physics simulation properties to the world, such as gravity and physics time steps
  - Go to the Menu Bar and click **Create > Physics > Physics Scene**
  - A PhysicsScene should be added to the stage tree
  - Click on it to examine its properties
    - You can see that gravity is set to point at the -Z direction with a magnitude of 9.8, a reminder that the default unit of length is meters
    - Unless you are simulating hundreds of rigid bodies and robots, it is more efficient to use a CPU solver instead of GPU, so for the purpose of this tutorial, disable GPU dynamics and use MBP Broadphase inside the Physics Scene's Property tab



First steps: add a ground plane

- Add ground plane
  - The ground plane will prevent any physics-enabled objects from falling below it
  - Use the top Menu Bar and Click Create > Physics > Ground Plane
  - The ground plane's collision property extends indefinitely even though the plane is only visible up to 25 meters in each direction
  - Turn on the grid to make the ground plane easier to see.



First steps: add a new light

- Add the light
  - Every new Stage is pre-populated with a defaultLight
  - Create an additional spotlight as an exercise
    - Go to Create > Light > Sphere Light
    - Move it up and turn it face down by moving it 7 units up and void the rotation in X and Y axis
    - Inside the Property Tab, change its color in Main > Color by simply clicking on the color bar and pick a color of your choice
    - Change its intensity
      - Main > Intensity to 1e6
      - Main > Radius to 0.05
      - Limit the scope of the spot in Shaping -> cone:angle to 45 degrees, and soften the edge of the spotlight in Shaping -> cone:softness to 0.05

First steps: create a complex object

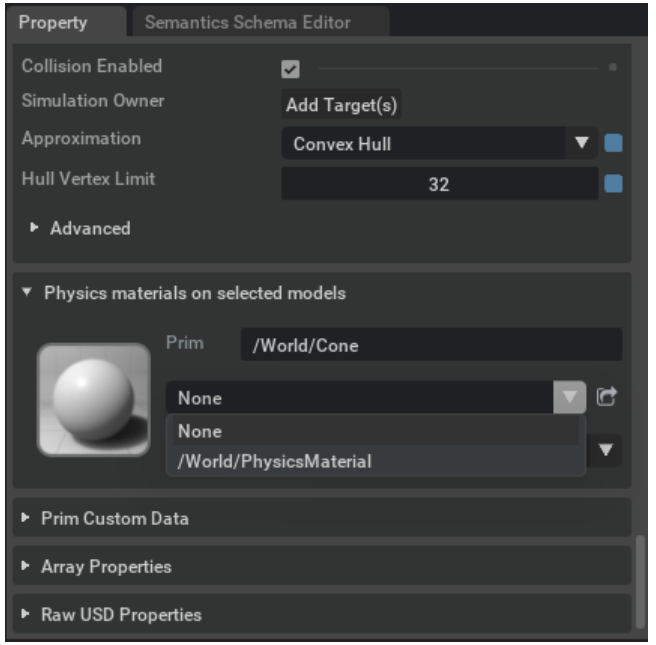
- Add a Cube to the stage as the mobile robot's body
  - Go to the Menu Bar and click Create > Shapes > Cube.
- Change the size of the cube by first select the Cube on the Stage tree, then go to the Property tab, find Geometry > Size, and type in 1.0
- Move it above the ground and change its shape by using the mouse
- You can directly type into the property window to make your changes
  - Let's fix the z-position of the cuboid at 1, the x-scale at 2, and the z-scale at 0.5
  - Bring its altitude to 1 meter
- In the same way, add a cylinder to the stage, change its Geometry > Radius to 0.5, and Geometry > Height to 1.0
- Then place it at y = 1.5 and z = 1.0
- Rotate it around the x-axis by 90 degrees
- Duplicate the cylinder we just made by right click it on the stage tree and Duplicate
- Move it to y = -1.5 while keeping all other parameters the same.

First steps: add physics to an element

- When the objects are first added, they are only visual objects, with no physics or collision properties attached to them
  - If you start the simulation by pressing Play and gravity is applied, these objects do not move because they are unaffected by physics
  - Let's turn them into rigid bodies with collision properties, as a robot should be.
- Select the Cube and both Cylinders on the stage tree
- In the Property tab, click on the + Add button, and select Physics > Rigid Body with Colliders Preset
- Press Play, and all three objects should now fall to the ground
  - The Rigid Body with Colliders Preset automatically add both Rigid Body API and Collision API to the objects
- Visualize the collision meshes
  - To see the outlines of the collision meshes, find the eye icon on top of the viewport, and click Show By Type > Physics > Colliders > All
  - Purple outlines should show up surrounding any objects that have collision APIs applied

First steps: add friction to an object

- The friction of an object can be modified as well
- Before to add a friction model, a material should be added to the objects
- We need to first create a different physics material and then assign it to the desired object
  - Go to the Menu Bar and click Create > Physics > Physics Material, select Rigid Body Material in the popup box
  - A new PhysicsMaterial will appear on the stage tree
  - Parameters such as friction coefficients and restitution can be tuned in its property tab
- To apply the assigned physics material to an object, select the object in the stage tree, find the menu item Physics Materials on Selected Model in the Property tab, and select the desired material in the drop-down menu
- Note that this is only available for objects with collision (physics) API applied
- Save the model
- To save the current USD stage, go to the Menu Bar and click Files > Save or Files > Save As .. to save as a new file.
- The file format is USD



Assemble a robot: add joints

- Open the USD model saved so far
  - *File> Open* the MODEL.USD
- To add a joint between two bodies, first click on the parent body and then the child body
- Select body, then while holding Ctrl + Shift, select one cylinder (let's say the left one)
- With both body highlighted, right-click and then select Create > Physics > Joints > Revolute Joint
- A RevoluteJoint will appear under wheel_left on the stage tree
- Rename it to wheel_joint_left.
  - Double check in the Property tab that body0 is the cube, and body1 is the cylinder.
- Change the Axis of the joint to Y
- If you look closely, the joint's two local frames are misaligned
  - To correct it, go to Local Rotation 0 and make x = 0 degrees
  - Next, go to Local Rotation 1 and make x = -90 degrees
  - The example here only shows an orientation misalignment, you may see translational misalignment depending on your robot.
- Do it the same for the right wheel joint
- Now that there are joints attached

Assemble a robot: add drives

- Adding the joint is only adding the mechanical connection
- To be able to control and drive the joints, we need to add a joint drive API
- Select both joints and click the + Add button in the Property tab
  - select Physics > Angular Drive to add drive to both joints simultaneously
    - Position Control: for position-controlled joints, set a high stiffness and relatively low or zero damping
    - Velocity Control: for velocity-controlled joints, set a high damping and zero stiffness
- For joints on a wheel -> velocity controlled
- Set both wheels' Damping to 1000 and Target Velocity to 200
- Press Play to see our mock mobile robot drive off.

Assemble a robot: add sensors

- Omniverse Isaac Sim provides a variety of sensors that can be used to sense the environment and robot's state
- Attach a camera sensor to our mock robot, a process that can be generalized to other sensors
- Go to the Menu Bar and select Create > Camera
- A camera will appear on the stage tree, and a grey wireframe representing the camera's view will appear on the stage
- You can move and rotate the camera's transform just like any other objects on the stage
  - Let's first rename the newly added camera to car_camera so we can keep track of it.
  - It would be easier to place the camera if we could see both the desired camera input stream, as well as where it is relative to the robot from an outside camera
  - Open up a second viewport window by going to the Menu Bar and click Window > New Viewport Window. A new viewport appears; dock it wherever you'd like.
  - Keep one of the viewport in Perspective camera view, and change the other one to car_camera view
  - Find the Cameras menu on the top edge of the viewport, and switch to Camera > car_camera
  - Attach the camera to the robot's body by dragging the prim under body. Now the camera will move together with the body. You may need to switch the camera view for the viewport again.
  - Press Play. The camera onboard the robot should now move with the robot.

Scripting (Isaac Core API)

- Every action in the GUI has its corresponding Python APIs
- Therefore, everything that's done in the GUI can be done with a Python command
  - Not all Python APIs have corresponding GUI
- Script Editor inside the GUI environment
- Isaac Python REPL extension as the two interactive scripting options for running python while the stage is open
- They are convenient tools for debugging and experimenting
- Script Editor is a python editing environment internal to Omniverse Kit
- It can be used to run snippets of python code to interact with the stage
- To open the script editor window
  - go to the Menu Bar and click Window > Script Editor
  - A window will pop up, and you can dock somewhere you find convenient
- You can open multiple tabs by going to the Tab Menu under Script Editor
- All the tabs share the same environment, so libraries that are imported or variables defined in one environment can be accessed and used in other environments.

Scripting (Isaac Core API)

- Use Isaac Sim Core API
- Use this script in a new EMPTY scene
- This script creates a new environment
- Core APIs simplifies some of the frequently used actions for robotics simulators and abstracts away default parameter settings

```
import numpy as np
from omni.isaac.core.objects import DynamicCuboid
from omni.isaac.core.objects.ground_plane import GroundPlane
from omni.isaac.core.physics_context import PhysicsContext
PhysicsContext()
GroundPlane(prim_path="/World/groundPlane", size=10, color=np.array([0.5, 0.5, 0.5]))
DynamicCuboid(prim_path="/World/cube",
 position=np.array([-.5, -.2, 1.0]),
 scale=np.array([.5, .5, .5]),
 color=np.array([.2,.3,0.]))
```

Scripting (USD APIs)

- The underlying format in NVIDIA Omniverse is USD
- Obtain the same result of the ISAAC Core script with the USD APIs
- To test USD APIs, we can use REPL
  - Read–Evaluate–Print loop is a programming shell that can read and evaluate small snippets of code and allow users to interactively query the state of the variables inside the environment
  - Make sure there is an instance of Isaac Sim already running.
  - Install Telnet:
    - $ apt-get install telnet
  - Enable the REPL Extension
    - Go to Windows > Extensions, search for Isaac Sim REPL, and enable it if it's not already
    - Open a new shell in your docker, and on the command line run telnet localhost 8223
    - A python shell will start in the terminal, and you are all set to start interacting with the stage opened in Isaac Sim via Python
  - Exit the shell environment by Ctrl+D.

Scripting (USD APIs)

- The underlying format in NVIDIA Omniverse is USD
- Obtain the same result of the ISAAC Core script with the USD APIs
- To test USD APIs, we can use REPL
  - Read–Evaluate–Print loop is a programming shell that can read and evaluate small snippets of code and allow users to interactively query the state of the variables inside the environment
  - Make sure there is an instance of Isaac Sim already running.
  - Install Telnet:
    - $ apt-get install telnet
  - Enable the REPL Extension
    - Go to Windows > Extensions, search for Isaac Sim REPL, and enable it if it's not already
    - Open a new shell in your docker, and on the command line run telnet localhost 8223
    - A python shell will start in the terminal, and you are all set to start interacting with the stage opened in Isaac Sim via Python
  - Exit the shell environment by Ctrl+D.

Scripting (USD APIs)

- from pxr import UsdPhysics, PhysxSchema, Gf, PhysicsSchemaTools, UsdGeom
- import omni
- stage = omni.usd.get_context().get_stage()
- # Setting up Physics Scene
- gravity = 9.8
- scene = UsdPhysics.Scene.Define(stage, "/World/physics")
- scene.CreateGravityDirectionAttr().Set(Gf.Vec3f(0.0, 0.0, -1.0))
- scene.CreateGravityMagnitudeAttr().Set(gravity)
- PhysxSchema.PhysxSceneAPI.Apply(stage.GetPrimAtPath("/World/physics"))
- physxSceneAPI = PhysxSchema.PhysxSceneAPI.Get(stage, "/World/physics")
- physxSceneAPI.CreateEnableCCDAttr(True)
- physxSceneAPI.CreateEnableStabilizationAttr(True)
- physxSceneAPI.CreateEnableGPUDynamicsAttr(False)
- physxSceneAPI.CreateBroadphaseTypeAttr("MBP")
- physxSceneAPI.CreateSolverTypeAttr("TGS")

Scripting (USD APIs)

- # Setting up Ground Plane
- PhysicsSchemaTools.addGroundPlane(stage, "/World/groundPlane", "Z", 15, Gf.Vec3f(0,0,0), Gf.Vec3f(0.7))

- # Adding a Cube
- path = "/World/Cube"
- cubeGeom = UsdGeom.Cube.Define(stage, path)
- cubePrim = stage.GetPrimAtPath(path)
- size = 0.5
- offset = Gf.Vec3f(0.5,0.2,1.0)
- cubeGeom.CreateSizeAttr(size)
- cubeGeom.AddTranslateOp().Set(offset)

- # Attach Rigid Body and Collision Preset
- rigid_api = UsdPhysics.RigidBodyAPI.Apply(cubePrim)
- rigid_api.CreateRigidBodyEnabledAttr(True)
- UsdPhysics.CollisionAPI.Apply(cubePrim)

OmniGraph

- OmniGraph is Omniverse's visual programming framework
- It provides a graph framework that can easily connect functions from multiple systems inside Omniverse
- It is also a compute framework that allows for highly customized nodes so that users can integrate their own functionality into Omniverse and automatically harness the efficient computation backend
- OmniGraph is the main engine behind the Replicators, ROS and ROS2 bridges
  - access to many sensors,
  - Controllers
  - external input/output devices
  - UI
  - Etc…

OmniGraph

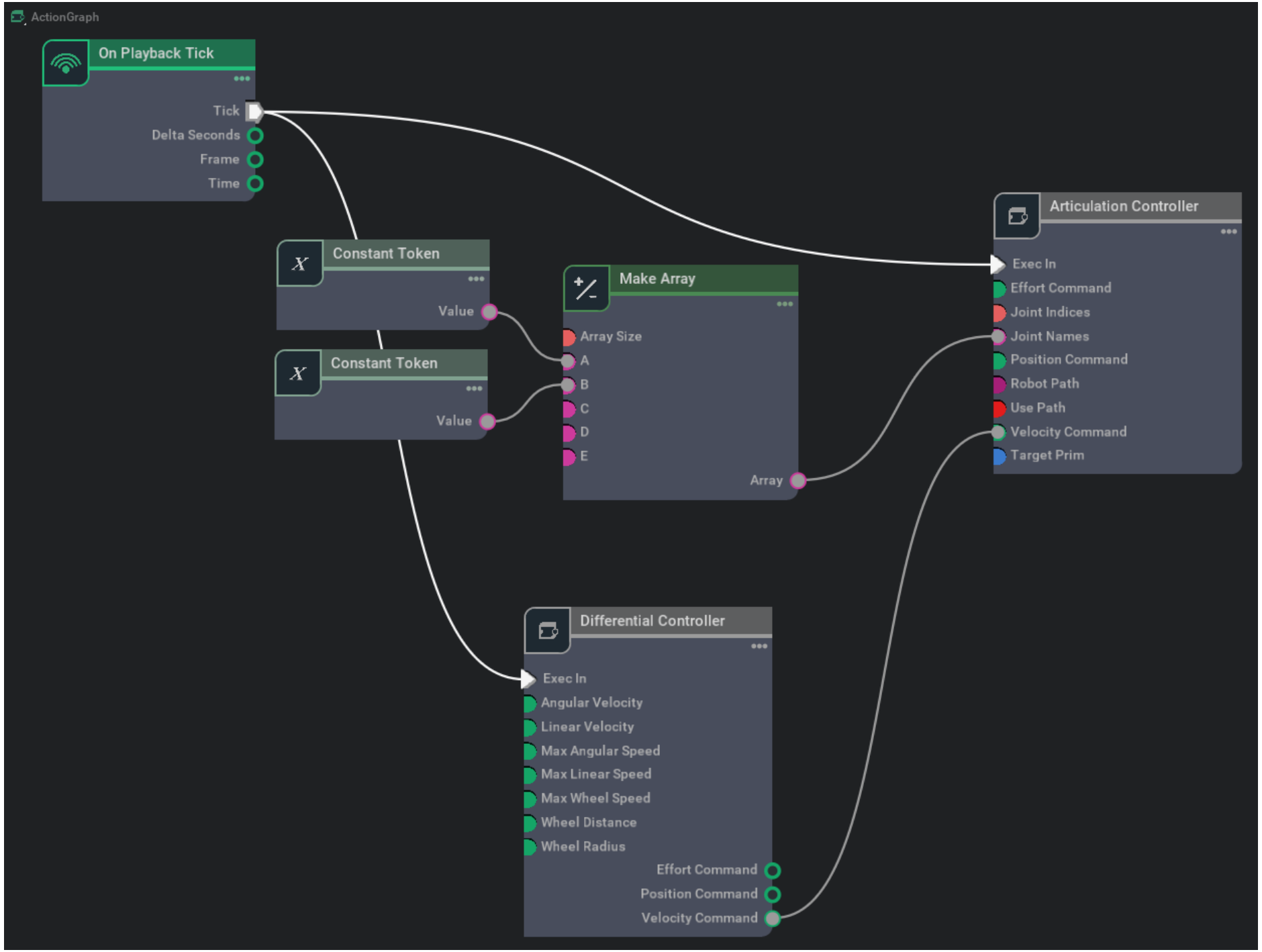
- Add a differential drive robot
  - On a new stage, start by right clicking and selecting Create -> Physics -> Ground Plane anywhere in the viewport
  - Use the content browser to navigate to Isaac/Robots/JetBot and click and drag jetbot.usd onto the stage
  - The jetbot should be under /World/jetbot in the context tree
- Create the graph
  - Select Window -> Visual Scripting -> Action Graph from the dropdown menu at the top of the editor
  - The Graph Editor will appear in the same pane as the Content browser
  - Click New Action Graph to open an empty graph
  - Type controller in the search bar of the graph editor
    - Drag an Articulation Controller and a Differential Controller onto the graph

OmniGraph

- The Articulation Controller applies driver commands (force, position, or velocity) to a joint
- To tell the controller which robot it's going to control, select the articulation controller node in the graph and open up the property pane
  - Keep usePath checked and type in the path to the robot /World/jetbot in the robotPath
- The Differential Controller computes drive commands for a two wheeled robot given some target linear and angular velocity
- Configuration:
  - select the Differential Controller node in the graph, and then in the properties pane, set the wheelDistance to 0.1125 and the and the wheelRadius to 0.03
- The Articulation Controller also needs to know which joints to articulate
  - It expects this information in the form of a list of tokens or index values
  - Each joint in a robot has a name and the jetbot has exactly two
  - Verify this by examining the jetbot in the stage context tree
  - Within /World/jetbot/chassis are two revolute physics joints named left_wheel_joint and right_wheel_joint.

OmniGraph

- Use the token to give info about the joints
- Type token in the search bar of the graph editor and add two Constant Token nodes to the graph
- Set their values to left_wheel_joint and right_wheel_joint in the properties pane
- Type make array into the search bar of the graph editor and add a Make Array node to the graph
- Select the Make Array node and set Array Size to 2 in the properties panel
- Finally, connect the constant token nodes to A and B of the Make Array node, and then the output of that node to the Joint Names input of the Articulation Controller node.
- Finally, search for playback in the search bar of the graph editor and add an On Playback Tick node to the graph
- This node will emit an execution event for every frame, but only while the simulation is playing
- Connect the Tick output of the On Playback Tick node to the Exec In input of both controller nodes
- Connect the Velocity Command output of the differential controller to the Velocity Command input of the articulation controller


ActionGraph
On Playback Tick
Tick
Delta Seconds
Frame
Time
Constant Token
Value
Constant Token
Value
Make Array
Array Size
A
B
C
D
E
Array
Articulation Controller
Exec In
Effort Command
Joint Indices
Joint Names
Position Command
Robot Path
Use Path
Velocity Command
Target Prim
Differential Controller
Exec In
Angular Velocity
Linear Velocity
Max Angular Speed
Max Linear Speed
Max Wheel Speed
Wheel Distance
Wheel Radius
Effort Command
Position Command
Velocity Command

OmniGraph

OmniGraph

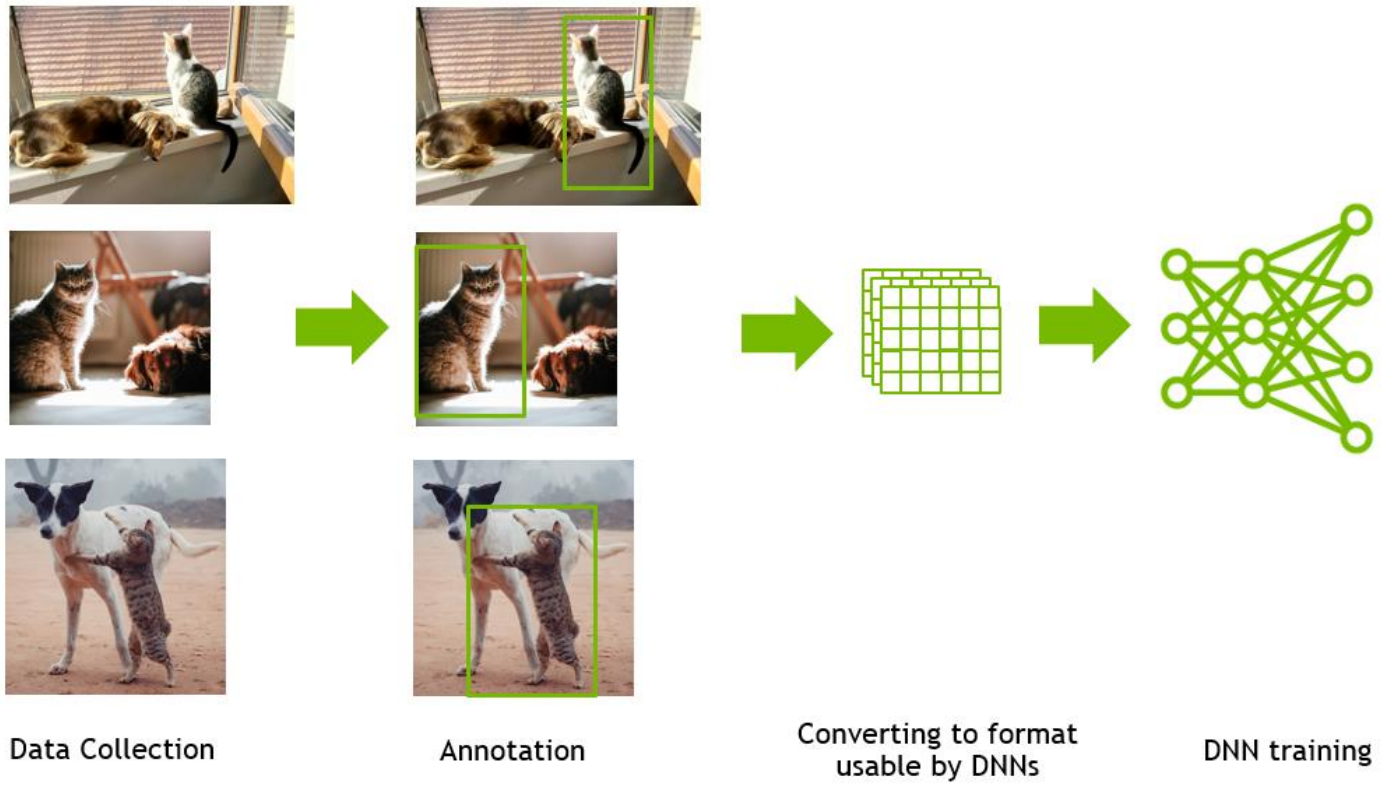
- The same graph can be improved to read input via keyboard

Syntetic data

- Omniverse Replicator is a highly extensible framework built on a scalable Omniverse platform that enables physically accurate 3D synthetic data generation to accelerate training and performance of AI perception networks
  - It provides deep learning engineers and researchers with a set of tools and workflows to bootstrapping model training, improve the performance of existing models or develop a new type of models that were not possible due to the lack of datasets or required annotations
  - It allows users to easily import simulation-ready assets to build contextually aware 3D scenes to unleash a data-centric approach by creating new types of datasets and annotations previously not available
- Built on open-source standards like Universal Scene Description (USD) or similar, Replicator can be easily integrated or connected to existing pipelines via extensible Python APIs
- It is built on the highly extensible OmniGraph architecture that allows users to easily extend the built-in functionalities to create datasets for their own needs
- It provides an extensible registry of annotators and writers to address custom requirements around type of annotations and output formats needed to train AI models
- Extensible randomizers allow the creation of programmable datasets that enable a data-centric approach to training these models
- It is exposed as a set of extensions, content, and examples in Omniverse Code

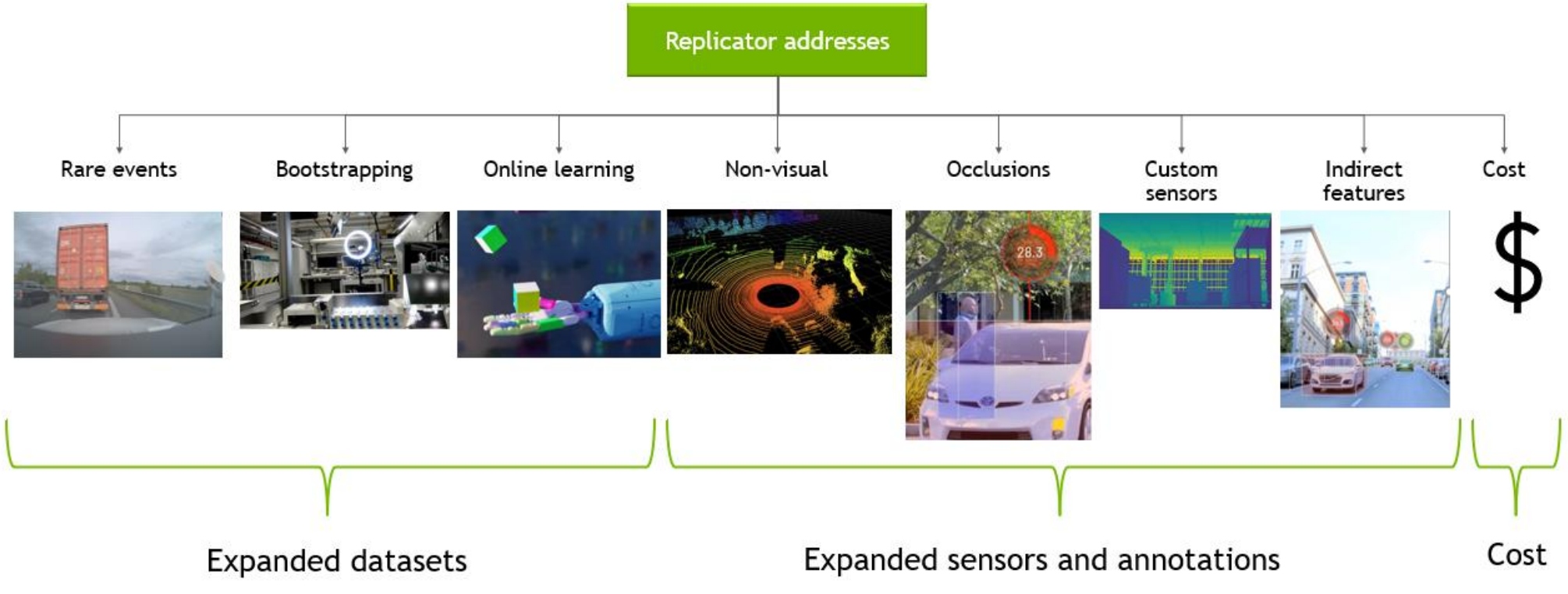
Syntetic data

- A typical process to train a deep neural network for perception tasks involves manual collection of data (images in most cases), followed by manual process of annotating these images and optional augmentations
- These images are then converted into the format usable by the DNNs
- DNN is then trained for the perception tasks
  - Hyperparameter tuning or changes in network architecture are typical steps to optimize network performance
  - Analysis of the model performance may lead to potential changes in the dataset however this may require another cycle of manual data collection and annotation
    - This is an expensive manual process.



Syntetic data

- Synthetic data generation enables large scale training data generation with accurate annotations in a cost-effective manner
- Some more difficult perception tasks require annotations of images that are extremely difficult to do manually (images with occluded objects)
- Programmatically generated synthetic data can address this very effectively since all generated data is perfectly labeled
- The programmatic nature of data generation also allows the creation of non-standard annotations and indirect features that can be beneficial to DNN performance

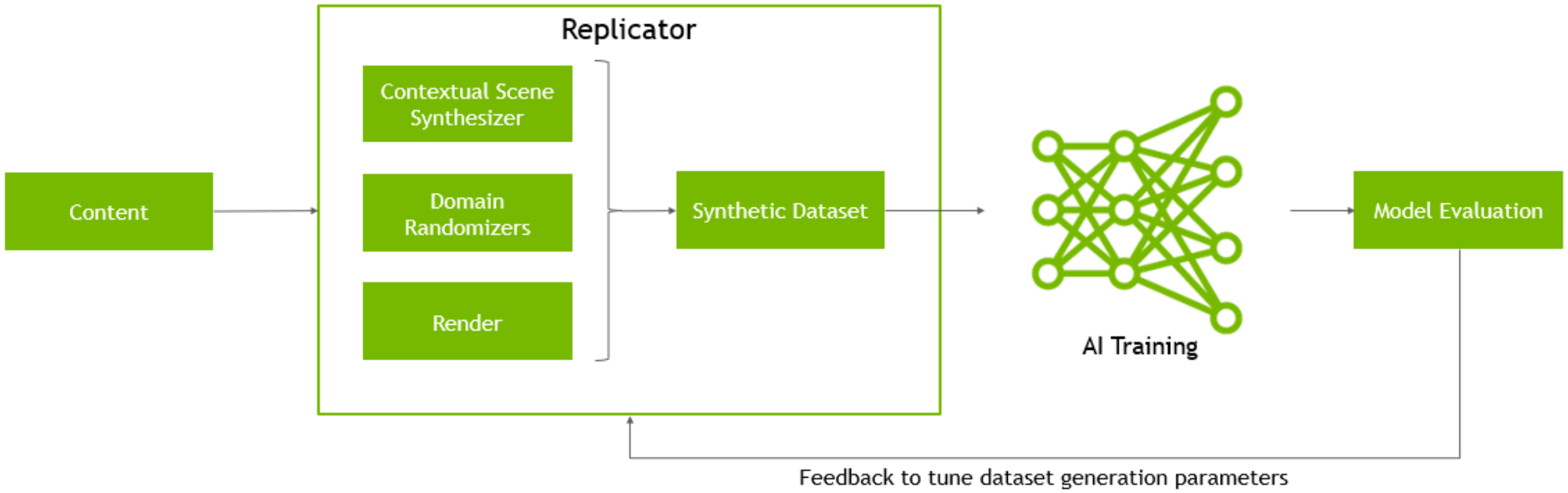


Syntetic data

- Synthetic data sets are generated using simulation
  - Its critical that we close the gap between the simulation and real world
  - This gap is called the domain gap
- The appearance gap is the set of pixel level differences between real and synthetic images
- These differences can be a result of differences in object detail, materials, or in the case of synthetic data, differences in the capabilities of the rendering system used
- The content gap refers to the difference between the domains
- This includes factors like the number of objects in the scene, the diversity in type and placement, and similar contextual information
- A critical tool for overcoming these domain gaps is domain randomization
- Domain randomization increases the size of the domain that we generate for a synthetic dataset to try to ensure that we include the range that best matches reality including long tail anomalies
- By generating a wider distribution of data than we might find in reality, a neural network may be able to learn to better generalize across the full scope of the problem
- The appearance gap can be further addressed with high fidelity 3D assets and ray-tracing or path-tracing based rendering, using physically based materials
- Validated sensor models and domain randomization of their parameters can also help here.

Syntetic data

- On the content side, a large pool of assets relevant to the scene is needed
- Omniverse provides a wide variety of connectors available to other 3D applications
- Developers can also write tools to generate diverse domain scenes applicable to their specific domain.
- These challenges introduce a layer of complexity to training with synthetic data
  - it is not possible to know if the randomizations done in the synthetic dataset were able to encapsulate the real domain
  - To successfully train a network with synthetic data, the network has to be tested on a real dataset



Syntetic data

- Replicator is composed of six components that enable you to generate synthetic data:
  - Semantic Schema Editor: Semantic annotations (data “of interest” pertaining to a given mesh) are required to properly use the synthetic data extension. These annotations inform the extension about what objects in the scene need bounding boxes, pose estimations, etc. The Semantic Schema Editor provides a way to apply these annotations to prims on the stage through a UI
  - Visualizer The Replicator visualizer enables you to visualize the semantic labels for 2D/3D bounding boxes, normals, depth and more
  - Randomizers: Replicator’s randomization tools allow developers to easily create domain randomized scenes, quickly sampling from assets, materials, lighting, and camera positions.
  - Omni.syntheticdata: represents the lowest level component of the Replicator software stack, and it will ship as a built-in extension in all future versions of the Omniverse Kit SDK.
    - The omni.syntheticdata extension provides low level integration with the RTX renderer, and the OmniGraph computation graph system
    - This is the component that powers the computation graphs for Replicator’s Ground Truth extraction Annotators, passing Arbitrary Output Variables or AOVs from the renderer through to the Annotators.
  - Annotators: The annotation system itself ingests the Angle Of Visions and other output from the omni.syntheticdata extension to produce precisely labeled annotations for DNN training
  - Writers: process the images and other annotations from the annotators and produce DNN specific data formats for training

ISAAC SIM + ROS Bridge

- Omniverse Isaac Sim have several tools to facilitate integration with ROS systems
  - We have both ROS and ROS2 bridges, an URDF importer, as well as connection to Gazebo/Ignition
- Let’s start importing a robot in URDF format
  - The ROS bridge comes with a few popular rostopics that are packaged for ease of use
  - To establish a ROS bridge for a specific topic, the steps can be generalized to the following:
    - open an action graph
    - add the OG nodes relevant to the desired rostopics
    - modify any properties as needed
    - connect the data pipeline (with a running roscore)
- By default, the ROS1 is installed in ISAAC SIM

ISAAC SIM + ROS Bridge

- Download the robot model from github
  - git clone -b <distro>-devel https://github.com/ROBOTIS-GIT/turtlebot3.git turtlebot3
- Locate the Xacro file for Turtlebot3 Burger in turtlebot3/turtlebot3_description/urdf/turtlebot3_burger.urdf.xacro. Convert the .xacro file to .urdf file by calling
  - $ rosrun xacro xacro -o <output_name>.urdf <input_file>.urdf.xacro
    - turtlebot3/turtlebot3_description/urdf/turtlebot3_burger.urdf.xacro
- Open the environment by going to the Content tab below the viewport and find Isaac/Environments/Simple_Room/simple_room.usd
- If you do not want to use the provided environment, just make sure there is a GroundPlane and a PhysicsScene to your environment
  - Both can be found in Create -> Physics
- On a new stage, drag the simple_room.usd onto the stage, and place it at the origin by zero out all the Translate components in the Transform Property. You many need to zoom in a bit to see the table inside the room
- Open the URDF importer Isaac Utils > Workflows > URDF Importer
- In the prompt window, inside Import Option section, uncheck clean stage to preserve the existing environment, uncheck Fix Base Link since this is a mobile robot, change Joint Drive Type to Velocity so that wheels can be properly driven later
- Inside Import section, first locate the URDF file you wish to import in the Input File
- The Import button will only enable after you've selected the file

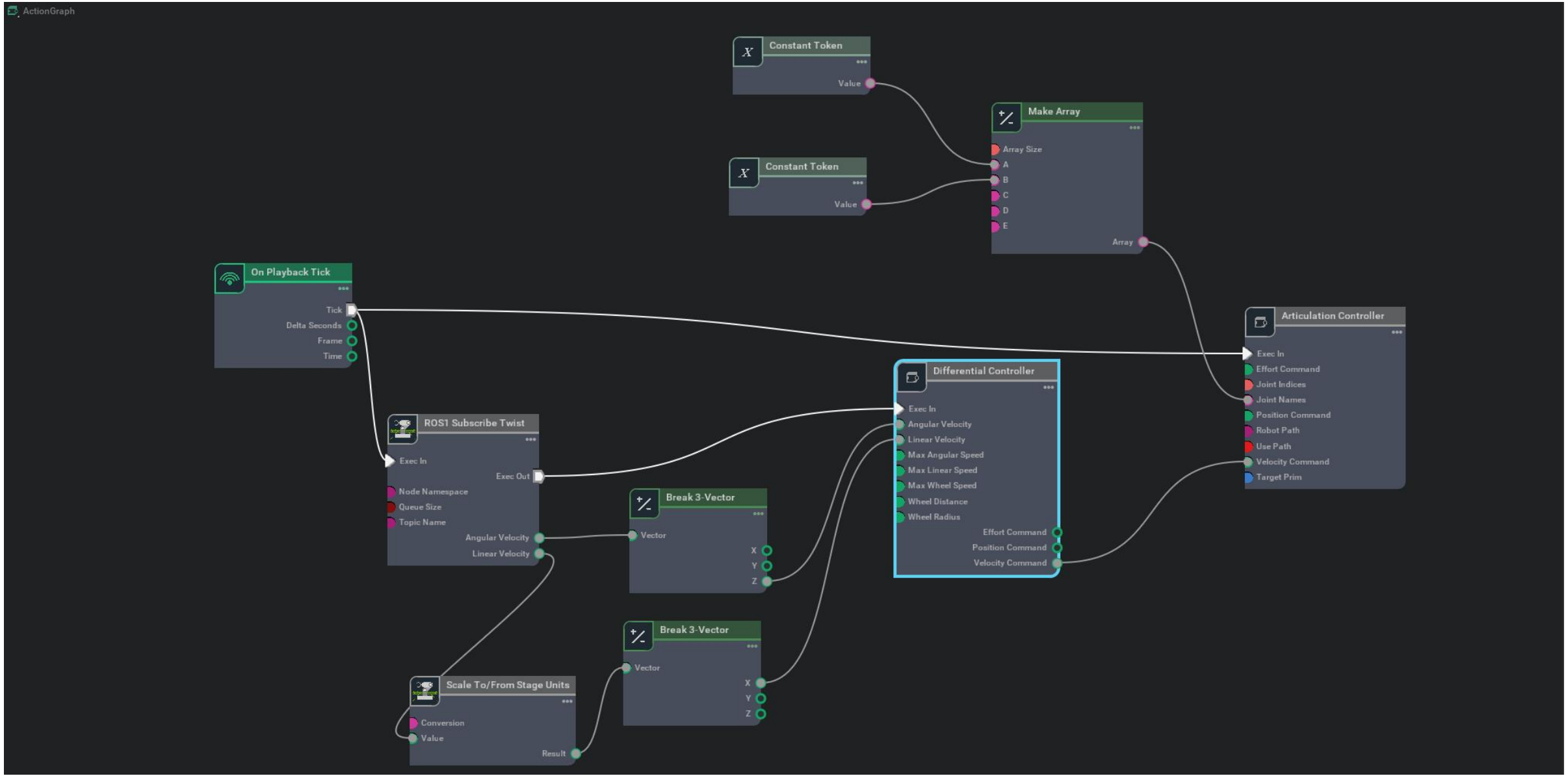
ISAAC SIM + ROS Bridge

- Once the asset is imported into Omniverse Kit, a copy of the .usd version of the asset will be automatically saved
- Specify the folder you wish to save the asset in Output Directory if it’s different than the folder that the .urdf file is located in. A folder name matching the .urdf file will be created in the specified directory, and the .usd file will be inside the newly created folder
- Click Import
- When the Turtlebot is first imported, it will be on the table. Place it just above the floor of the room using the mouse
- Press Play to start the simulation

ISAAC SIM + ROS Bridge

- Drive the robot
  - Subscribe to a ROS twist message and send the data to the controllers that can drive the robot around
  - Open the action graph: Window > Visual Scripting > Action Graph
  - An Action Graph window will appear on the bottom
  - Click on the New Action Graph Icon in middle of the Action Graph Window.
  - Inside the Action Graph window, there is a panel on the left-hand side with all the OmniGraph Nodes (or OG nodes)
  - All ROS related OG nodes are listed under Isaac Ros
  - You can also search for nodes by name. To place node into the graph, simply drag it from the node list into the graph window

ISAAC SIM + ROS Bridge

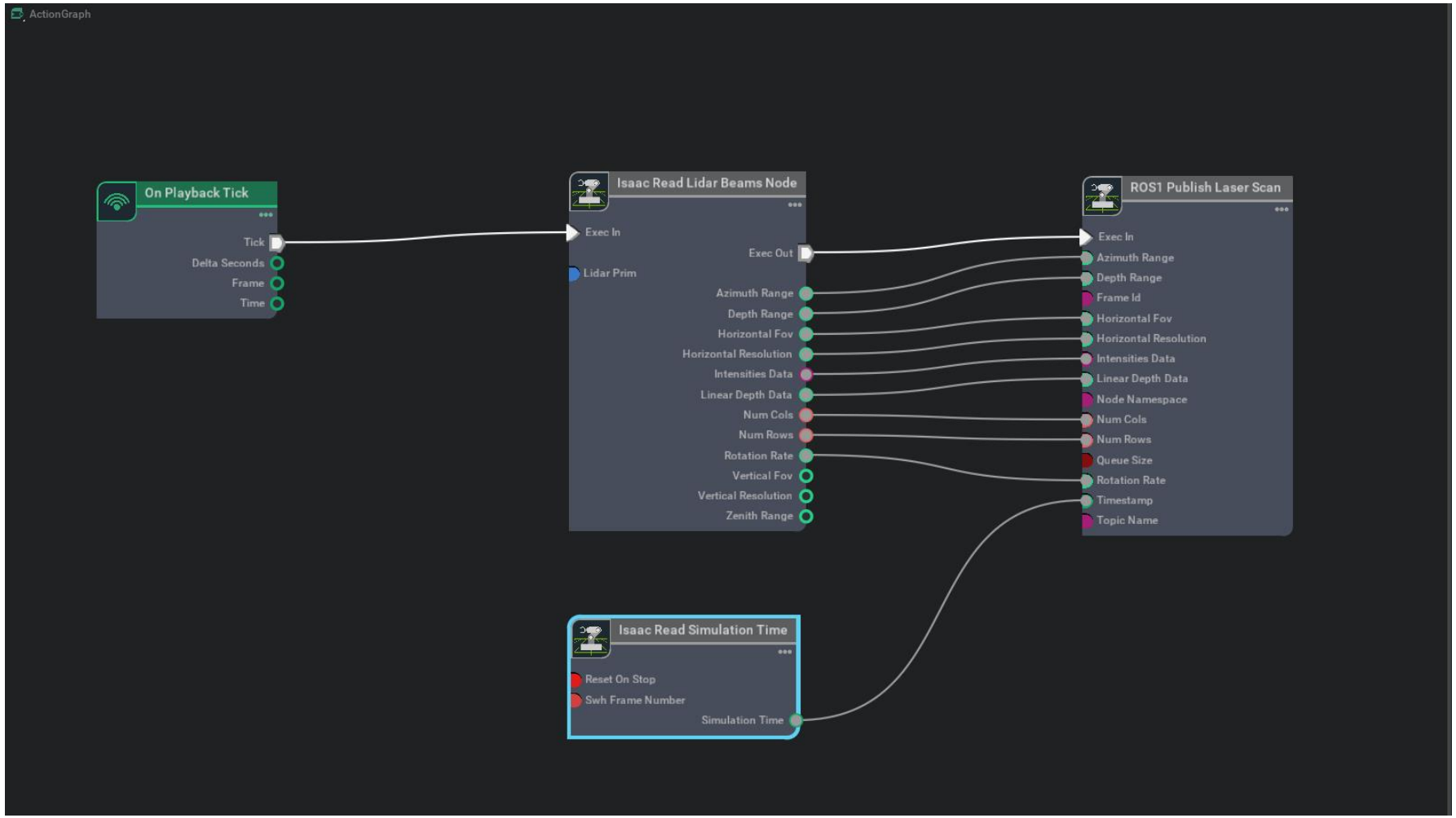


ISAAC SIM + ROS Bridge

- Add sensors: Lidar
- First, we need to add a lidar sensor to the robot
  - Go to Create -> Isaac -> Sensors -> Lidar -> Rotating
- To place the synthetic lidar sensor at the same place as the robot's lidar unit, drag the lidar prim under /World/turtlebot3_burger/base_scan
- Zero out any displacement in the Transform fields inside the Property tab
- The lidar prim should now be overlapping with the scanning unit of the robot
- Inside the RawUSDProperties tab for the lidar prim, set the maxRange to 25.
- This way the lidar will ignore anything that's beyond 25 meters
- This will prevent the lidar reporting a hit everywhere in the room because of the walls.
- We'll check drawLines to visualize the lidar scans.
- Press Play to see the lidar comes to life
- Red lines of the scan means hit, green means no hit, the color spectrum from green to yellow to red is proportional to the distance of the object detected.

ISAAC SIM + ROS Bridge

- Add sensors: Lidar
- Once the lidar sensor is in place, we can add the corresponding OG nodes to stream the detection data to a Rostopi



ISAAC SIM + ROS Bridge

- On Playback Tick Node: Producing a tick when simulation is “Playing”. Nodes that receives ticks from this node will execute their compute functions every simulation step.
- Isaac Read Lidar Beam Node: Retrieve information about the Lidar and data. For inputs:LidarPrim, add target to point to the Lidar sensor we just added at /World/turtlebot3_burger/base_scan/Lidar.
- ROS1 Publish Laser Scan: Publishing laser scan data. Type /laser_scan into the Topic Name field.
- Isaac Read Simulation Time: Use Simulation time to timestamp the /laser_scan messages.

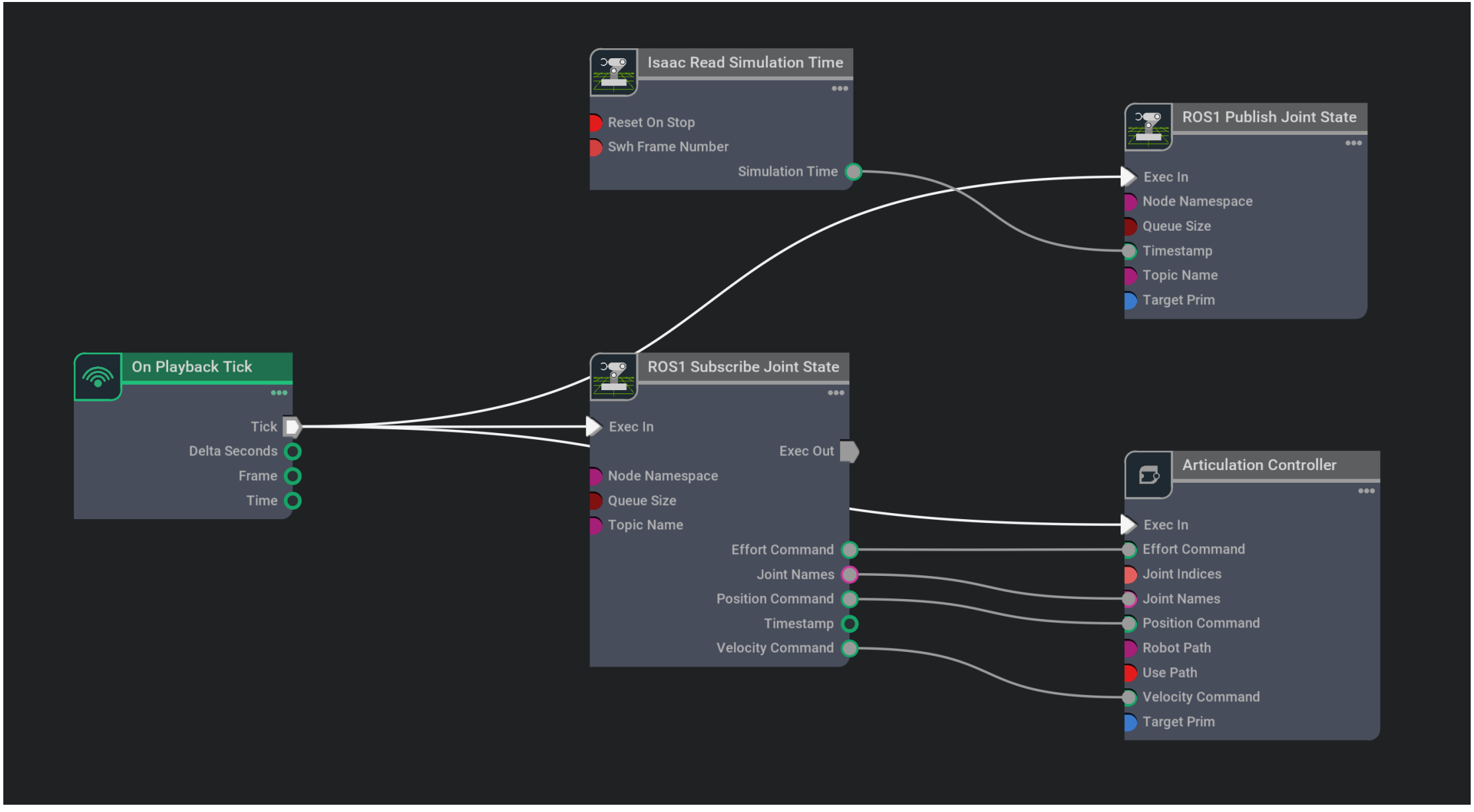
ISAAC SIM + ROS Bridge

- Press Play to start ticking the graph and the physics simulation
- In a separate ROS-sourced terminal , check that the associated rostopics exist with rostopic list. /laser_scan should be listed in addition to /rosout and /rosout_agg
- To visualize the laser scan data, open RViz by typing in rviz on the command line and enter
- Inside rviz, add a Laser Scan type to visualize. Make sure the Topic that the laser scan is listening to matches the topic name inside the ROS1 Publish Laser Scan, and fixed frame matches the frameID inside the ROS1 Publish Laser Scan node

ISAAC SIM + ROS Bridge

- Arm control via joints
- Add Joint States in UI
  - Go to Content tab below the viewport, and open Isaac/Robots/Franka/franka_alt_fingers.usd
  - Go to Create -> Visual Scripting -> Action Graph to create an Action graph
  - Add the following OmniGraph nodes into the Action graph:
    - On Playback Tick node to execute other graph nodes every simulation frame
    - Isaac Read Simulation Time node to retrieve current simulation time
    - ROS1 Publish Joint State node to publish ROS Joint States to the /joint_states topic
    - ROS1 Subscribe Joint State node to subscribe to ROS Joint States from the /joint_command topic
    - Articulation Controller node to move the robot articulation according to commands received from the subscriber node
      - Select ROS1 Publish Joint State node and add the /panda robot articulation to the targetPrim
      - Select Articulation Controller node and add the /panda robot articulation to the targetPrim
        - Additionally make sure to uncheck usePath
      - Connect the Tick output of the On Playback Tick node to the Execution input of the ROS1 Publish Joint State, ROS1 Subscribe JointState and Articulation Controller node
      - Connect the Simulation Time output of the Isaac Read Simulation Time node to the Timestamp input of the ROS1 Publish Joint State node

ISAAC SIM + ROS Bridge

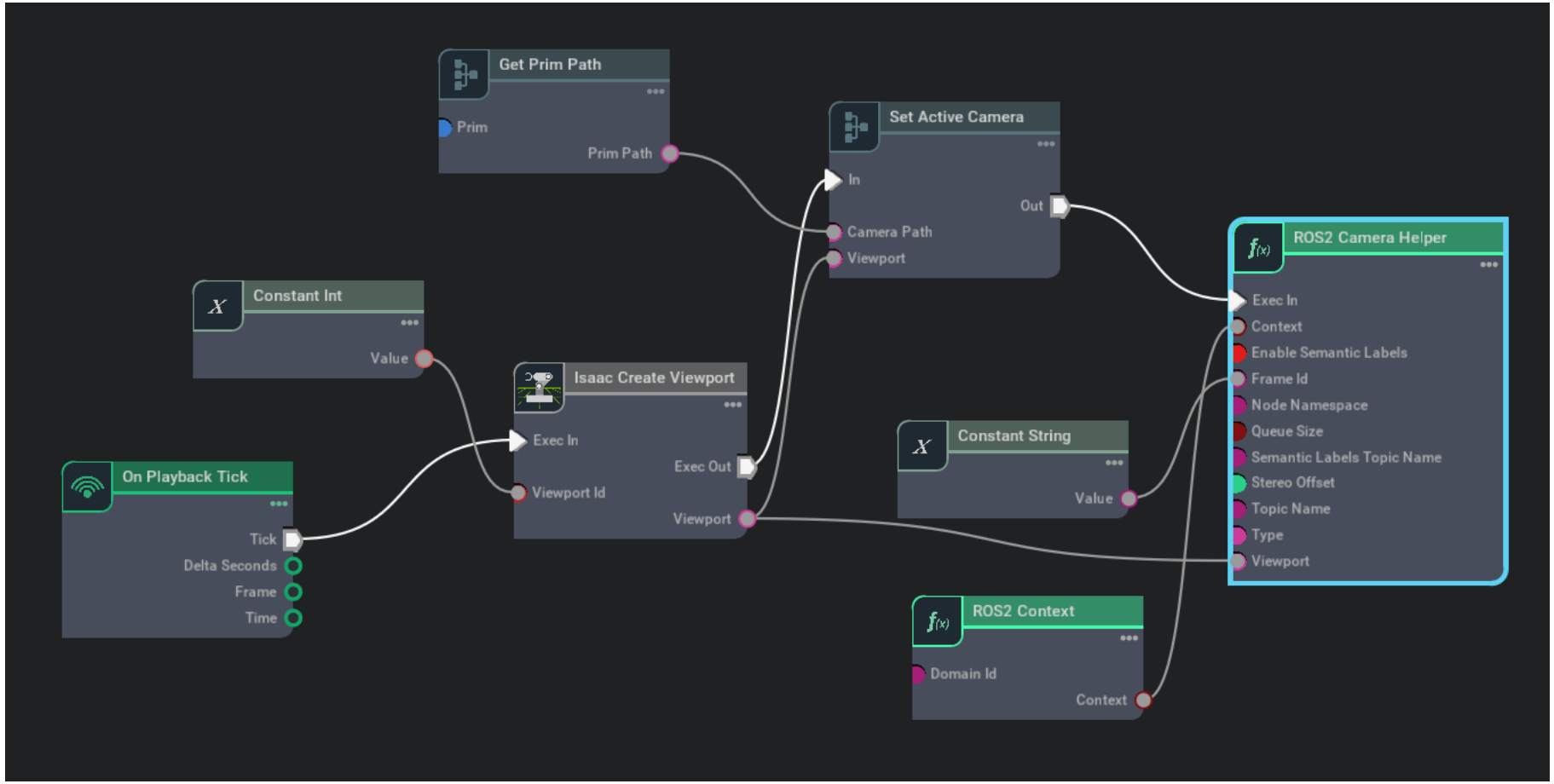


ISAAC SIM + ROS Bridge

- Make sure roscore is running, press Play to start publishing joint states to the /joint_states topic and subscribing commands on the /joint_command topic.
- To test out the ROS bridge, use the provided python script to publish joint commands to the robot. In a ROS-sourced terminal:
  - $ rosrun isaac_tutorials ros_publisher.py
- While the robot is moving, open a ROS-sourced terminal and check the joint state rostopic by rostopic echo /joint_states. You'll see that the position of the joints are changing following the robot.

ISAAC SIM + ROS2 Bridge

- Like for ROS, a ROS2 bridge exist
- ROS and ROS2 bridge can not be activated at the same time
- Go to the extension page and disable the ROS1 bridge to enable the ROS2 bridge
- Add a camera sensor to the AR Marker scene



Conclusions

- Robotic software technologies
- Frameworks
  - ROS
  - ISAAC SDK
  - ROS2

Framework	Supported	Robot-enabled	Ecosystem	OS
ROS	Until 2025	Several	Isaac SDK (bridge) ROS2 (bridge, close to end-of-life)	Linux (ubuntu preferred)
ISAAC	EoL	No	ROS (bridge)	Ubuntu 18.04
ROS2	Not planned	Few	ROS (bridge, close to end-of-life)	Linux, (ubuntu >= 18.04 preferred)

Conclusions

- Simulators
  - Gazebo
    - Used as default simulator for ROS and ROS2
    - Can be used as standalone software
    - Can simulate a multitude of robots (ground, aerial, underwater)
  - ISAAC SIM
    - Can be used as a standalone software with internal programming system
    - Can be used to perform network training and synthetic data creation
    - Simulates ground robot
    - Can communicate with ROS and ROS2 thanks to proper bridge
      - The bridge must be configured