LEONARDO TECHNICAL TRAINING

Robotic Software Lezione 6

NVIDIA ISAAC SDK

Codelets basic theory

- Codelets are the basic building blocks of a robotics application built with Isaac
- The Isaac SDK includes various codelets which can use in your application
- We can define new codelets
- Codelets implement the alice::Codelet interface
  - Codelets are very common components which enable you to write code which is executed repeatedly
  - The developed class is derived from the alice::Codelet
  - Codelets run in one of the three following ways:
    - Tick periodically: The tick function is executed regularly after a fixed time period
      - A typical example is a controller which ticks 100 times a second to send control commands to hardware
    - Tick on message: The tick function is executed whenever a new message is received
      - A typical example is an image processing algorithm which computes certain information on every new camera image which is captured
    - Tick blocking: The tick function is executed immediately again after it has finished
      - A typical example is a hardware driver which reads on a socket in blocking mode (sync from another event)
  - When a codelet ticks, it prevents other codelets in the same node from ticking at the same time
  - To run codelets in parallel, place them in separate nodes of a graph

Codelets basic theory

- Codelets can receive messages under the publish/subscribe protocol
- Many components receive or transmit data to other components
- Message passing is a powerful way to encapsulate components and ensuring modularity of the codebase
  - ISAAC_PROTO_RX(StateProto, state);
- The ISAAC_PROTO_RX macro is used to define a receiving (RX) channel.
- The macros take two arguments:
  - the type of the message and the name of the channel
- A message can be read on a receiving channel for example
  - const auto& rmp_reader = rx_state().getProto();
  - ...
  - state_.speed() = rmp_reader.getLinearSpeed();
- The function rx_state is automatically generated by the ISAAC_PROTO_RX macro
- A StateProto message containing a DifferentialBaseDynamics is expected
- All message schemas of the Isaac SDK can be found in the message folder or on the documentation

Codelets basic theory

- Codelets can send messages under the publish/subscribe protocol
- At the end of the tick, after all computations are done, a component often wants to send out a new message to whomever is listening
    - ISAAC_PROTO_TX(Odometry2Proto, odometry)
- The ISAAC_PROTO_TX macro is used to define a transmitting (TX) channel
- This is very similar to the way in which ISAAC_PROTO_RX macro works.
- A message can be created and sent:
    - auto odom_builder = tx_odometry().initProto();
    - ToProto(odom_T_robot, odom_builder.initOdomTRobot());
    - odom_builder.setSpeed(state_.speed());
    - ...
    - tx_odometry().publish();
- The tx_odometry function is automatically created by the ISAAC_PROTO_TX macro
- Use initProto to start a new message on this channel
- Functions automatically generated
- When the message is complete it can be sent via the publish() function
- Only one message can be generated at a time.

ToProto/FromProto Functions

- Primary data types such as integers are supported directly by cap’n’proto
    - While Cap'n Proto has code generators in a variety of languages, the immaturity of the implementations and relatively smaller number is frequently cited as a barrier to adoption
- It can be more difficult to process complicated data types
- To handle such cases, Isaac SDK provides convenient ToProto/FromProto functions
    - Write a variable to a Proto
        - void ToProto(const Uuid& uuid, ::UuidProto::Builder builder)
    - Read a UUID from a proto
        - Uuid FromProto(::UuidProto::Reader reader)
- Or parses a tensor from a message
    - bool FromProto(::TensorProto::Reader reader, const std::vector<isaac::SharedBuffer>& buffers, isaac::UniversalTensorConstView<Storage>& universal_view);
- Creates a tensor from a proto. Will print errors and return false if the tensor type is not compatible with the proto.
    - bool FromProto(::TensorProto::Reader reader, const std::vector<isaac::SharedBuffer>& buffers, isaac::TensorBase<K, Dimensions, BufferType>& tensor_view)
- See the messages folder for header files containing ToProto/FromProto functions.

Parameters

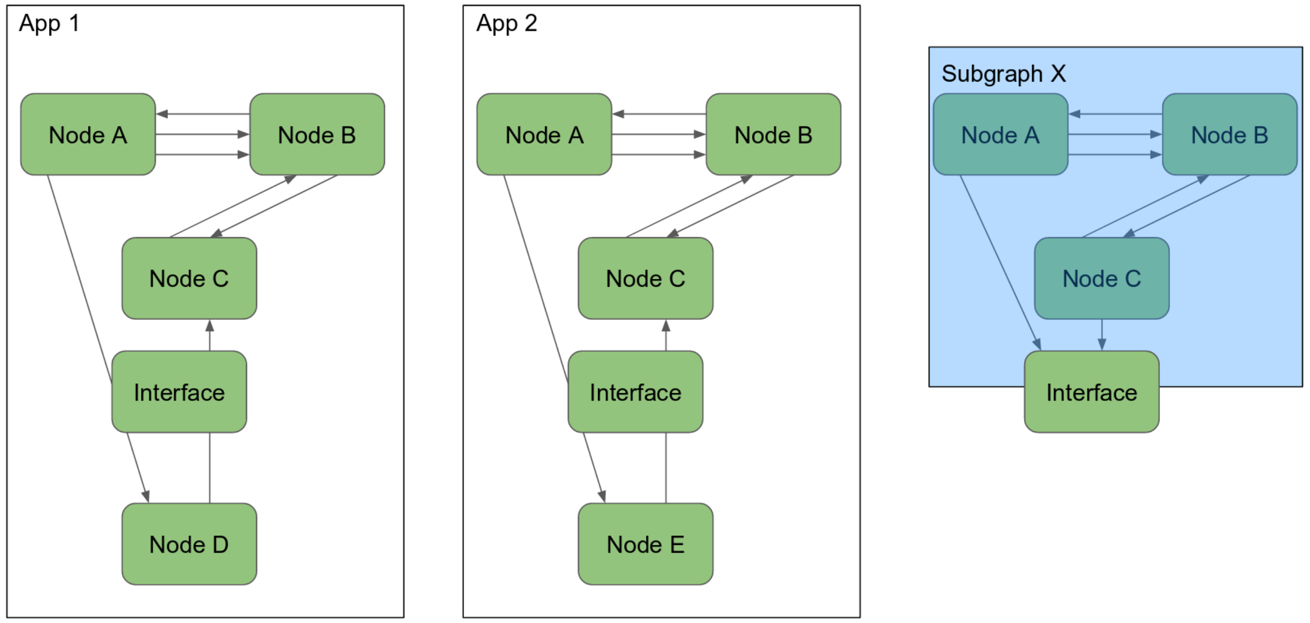
- Complicated algorithms can often be parameterized in various different ways
    - ISAAC_PARAM allows you to define a configuration parameter which can be set via configuration, read in the code, and changed in the frontend
- Maximum acceleration to use (helps with noisy data or wrong data from simulation)
    - ISAAC_PARAM(double, max_acceleration, 5.0)
- There are three parameters to ISAAC_PARAM:
    - type: this is the type of configuration parameter
        - the basic types are int, double, bool and std::string
    - name: the name defines the key under which the parameter is stored in the configuration file and the function name under which it can be accessed in code
    - default value: In case no value is specified in the configuration file this value is used instead
        - The default can also be omitted which forces the user to specify a value in the configuration file.

Parameters

- Configuration can be changed in multiple ways:
  - The default configuration parameter can be changed
  - This should be used with caution, because it’ changes the value for all applications which have not overwritten the value in a configuration file
  - The value can be set in a JSON configuration file
  - Most sample applications include a JSON file where various parameters are set
    - "max_acceleration": 2.0

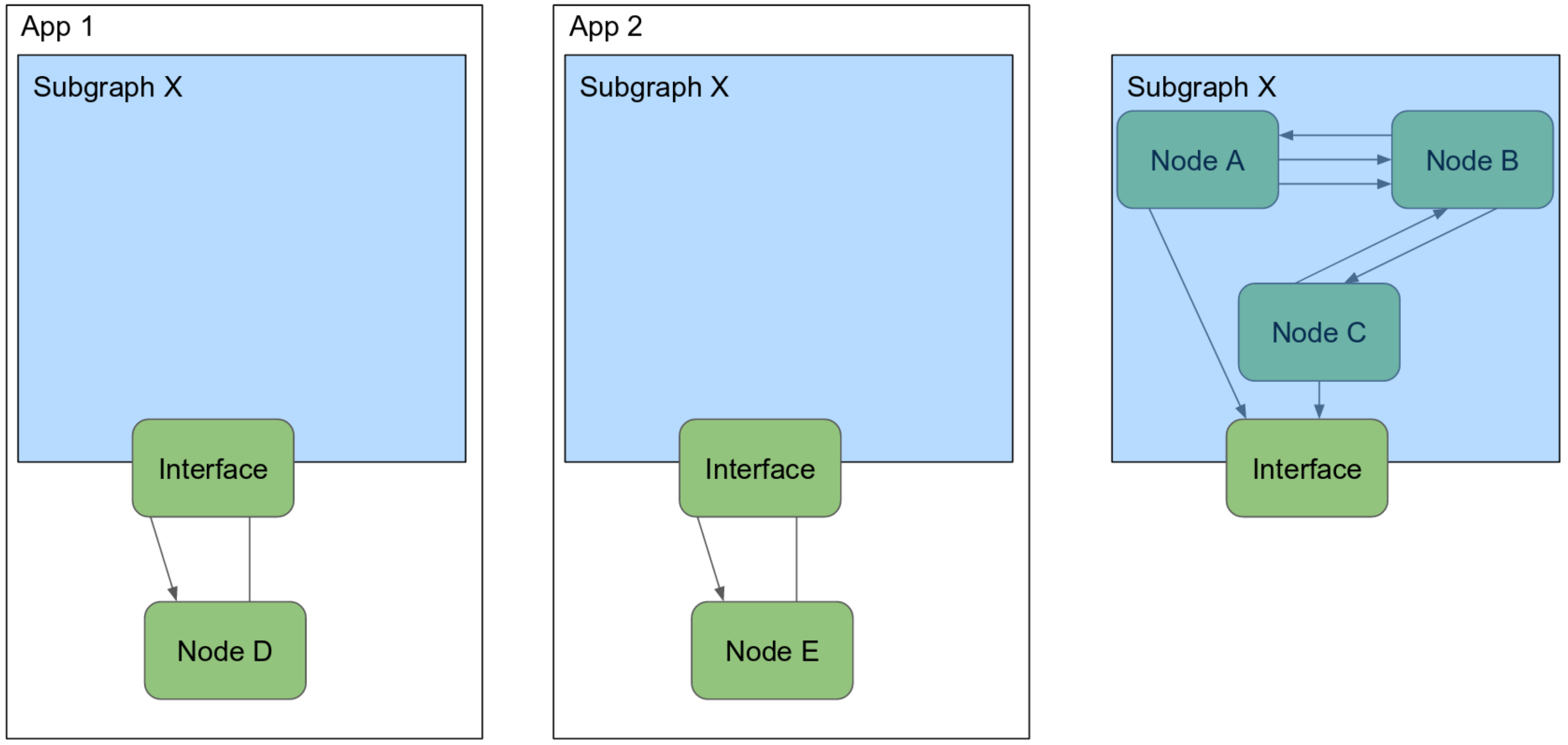
Sub-graphs

- As more components are added to applications, application graphs can get lengthy and repetitive
  - Subgraphs can prove useful in simplifying application graphs
  - When using and re-using nodes with multiple components connected together, a JSON subgraph can include required components, edges, and configurations, so that you can add a relatively high-level group without concern for the lower-level details.
- In this diagram, nodes A, B, and C in App 1 and App 2 are identical
- Instead of duplicating them in the JSON for each application, a subgraph X can be created.



Sub-graphs

- This abstraction simplifies the apps, lowers the maintenance, hides the expertise, and provides a better user experience.



Example 1.6

- The ping example v2
- Try message exchange
- Developing Codelets in C++: ping_v2
  - Develop two codelets in C++
    - The first is a machine that goes "ping"
    - The second (pong) responds to the ping message, only when received
  - Develop
    - Application
    - Bazel build
    - Json
- Compile with bazel build
  - $ bazel build ping_v2
- Run with bazel run
  - $ bazel run ping_v2

Example 2.6

- Let’s do a more complex example
    - Implement GoTo behaviour
    - The GoTo behaviour is implemented in an already given source file
    - Implement an interface for the GoTo behaviour to read the destination point from a parameter
        - Read the state of the navigation function to check if it is arrived or not
    - Clone the GoTo.cpp and GoTo.hpp source and header from the github repository
        - $ cd sdk/apps/examples
        - $ git clone https://github.com/robotic-software/goal_geneator
        - $ cd goal_generator
    - Create the additionally needed files
        - $ touch BUILD
        - $ touch my_goal_generator.app.json
        - $ touch MyGoalGenerator.cpp
        - $ touch MyGoalGenerator.h

Example 3.6

- Python is also supported from the ISAAC SDK codelets
- While in terms of performance, the best language for writing codelets is C++, not all codelets of an application need to be in the same language
- The Isaac SDK also supports Python codelets (pyCodelets)
  - Create Python codelets
  - Also, in this case the BUILD, the JSON and the source files must be created in the codelet directory

Example 3.6

- The main difference is in the BUILD file
  - Here we define a py_binary module and an isaac_pkg
- Create a python node to replicate the ping behaviour
- Call the directory ping_python

Example 4.6

- Develop a proportional controller in python
- Receive the goal value from an external parameter
- Publish the result over the ISAAC network
- Display commanded position on sight

Sight

- Robotics application can become quite complicated, and it is necessary to inspect the inner workings of algorithms and components
- Two of the most important visualization tools of Isaac SDK are sight and websight
  - sight is an API which can be used to create variable plots, or to visualize data in 2D or 3D renderings
  - websight is a web-based frontend which can be used to look at data which is provided via the sight API
    - Applications communicate with websight by running an instance of WebsightServer
    - The WebsightServer instance allows applications to display plots, 2D and 3D drawing, current application status (active nodes and connections), and update configuration.

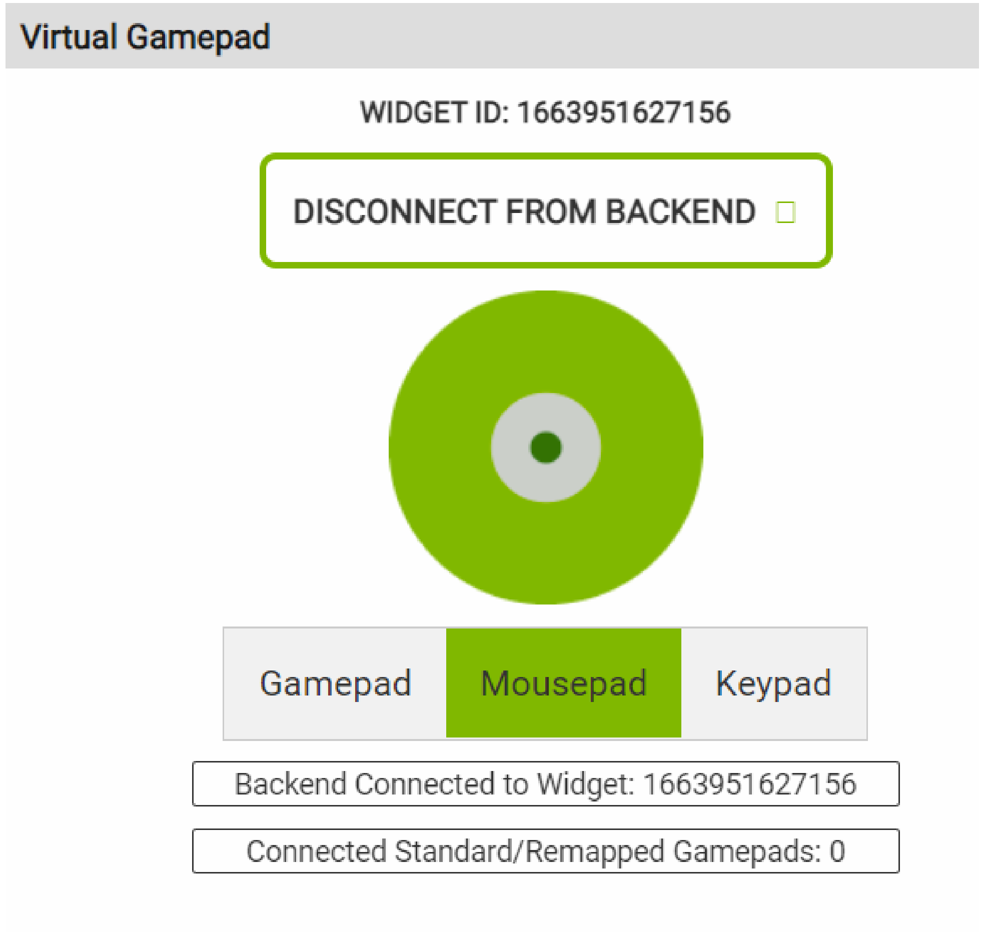
- To get started run one of the sample apps which have visualization setup apps/samples/stereo_dummy
- Open a web browser and navigate to http://IP_ADDR:3000
- You will see a couple of windows showing you visualization data about the application you are running.

Sight

- Websight can also be used to control your robot
- Markers can be used to generate goals for mobile robots
- Parameters can be changed from the webpage
- Complex web objects can be used to generate velocity commands

Example 5.6

- Develop a joypad to stream velocity commands
  - We can use something of already implemented
  - Part of Navigation GEM
    - isaac::navigation::VirtualGamepadBridge

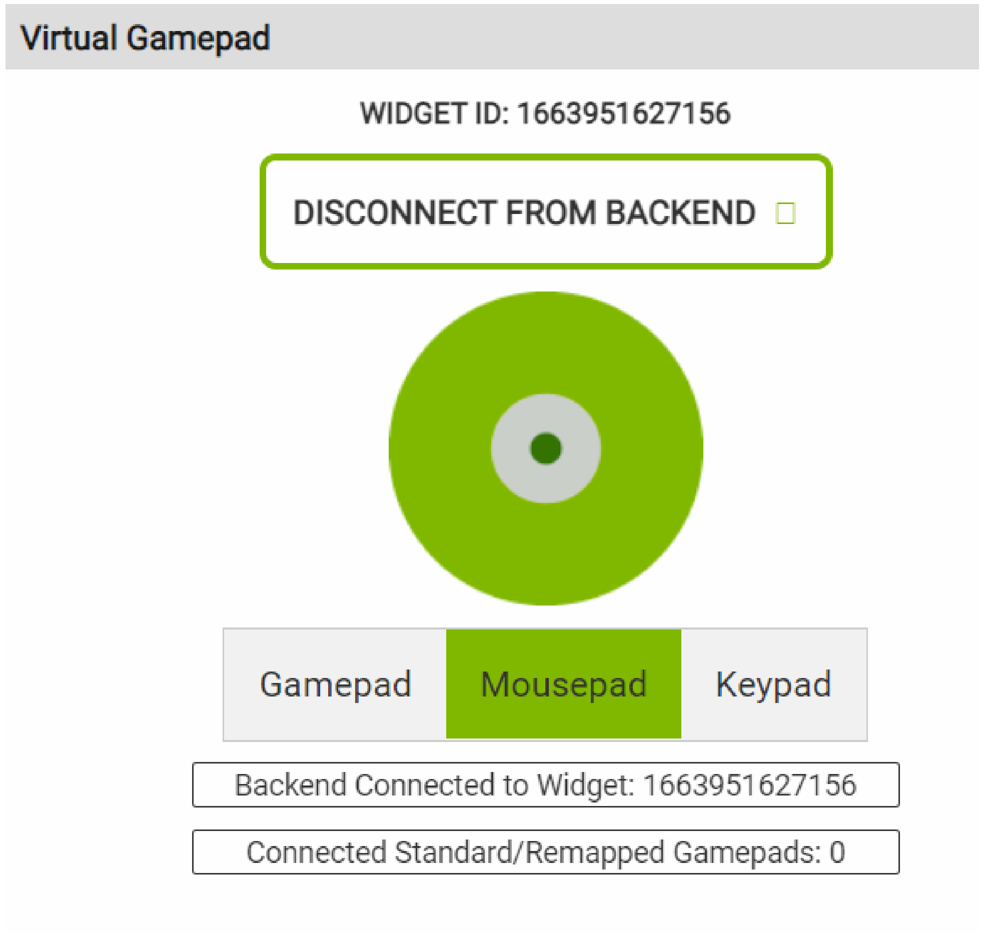


Backend Connected to Widget: 1663951627156

Connected Standard/Remapped Gamepads: 0

Example 6.6

- Develop a joypad to stream velocity commands
- Catch the input of the webpage in an Isaac node



Backend Connected to Widget: 1663951627156

Connected Standard/Remapped Gamepads: 0

Example 7.6

- Application file can also be written directly into the cpp code
- This semplifies the generation of multiple files
- Try to transform our ping example writing the json file directly into the .cpp
- Create a ping_no_json.cpp codelet

Example 8.6

- Test visualization capabilities of sight
  - Display an Image
  - Display a signal
- Create a visualization app
  - Develop 2 codelets in the app
  - One to visualize an image
  - One to visualize a sinusoidal signal

Record & Replay

- An Isaac application is represented by a graph where the components inside their respective nodes can receive and send messages
- Recording is storing all the messages emitted by certain components in a log
- In the same way Replay means to replay all the recorded messages from a log.
- The Isaac SDK provides two special components to achieve this purpose:
  - Recorder and Replay
  - Typically, in an app, two nodes are created that contain recorder and replay components respectively
- The log in an Isaac application is a directory
- The path to the log folder is made up of three parts:
  - The base directory, an application UUID directory and a tag
  - The base directory needs to be a directory where the user running the application has write privileges
  - The application UUID is a string representing a unique ID per execution of the application
  - This is a new unique ID every time the application runs
  - It cannot be changed or predetermined
  - Finally, the tag is an optional folder that lets the user distinguish between different logs captured during a single execution of the application
    - 543dba0a-4926-11ed-a6e0-c533c34f6eba/test/6b66156a-4926-11ed-891e-9586411b00dc

Example 8.6

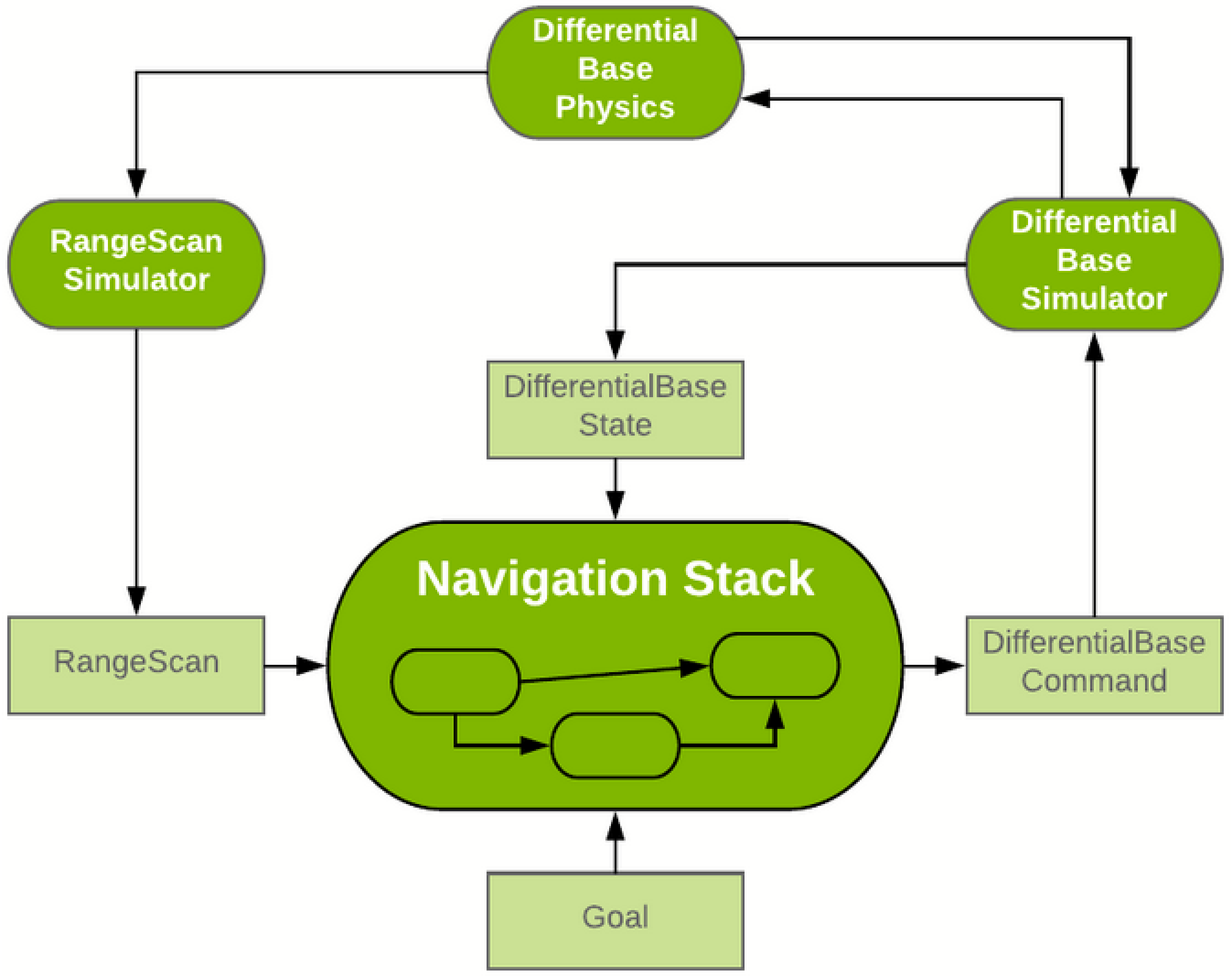
- Recorder Component
  - Record the joypad data from the sight
  - Applications that must record a component’s channel must define a node containing a recorder component, and then connect all the components to be recorded to that recorder component.
- Create two json application file
  - One to record
  - One to replay

Flatsim

- flatsim stands for flatworld simulation
- Small simulation application which allows you to run almost the full Isaac navigation stack
- The flatsim application simulates a laser range scanner by casting rays in a given occupancy grid map
- This kind of simulation is extremely fast compared to a more costly 3D ray scan in a full 3D environment
- It is thus a quick and highly performant way to test the navigation stack.
- flatsim uses the set of nodes from the navigation stack and adds two new nodes:
  - DifferentialBaseSimulator: Runs a basic differential base drive model to move the robot around based on the commands sent from the navigation stack
    - This provides the ground truth pose of the robot for range scan simulation. It also provides wheel odometry which is used as another input by the navigation stack
  - RangeScanSimulator: Computes a simulated flat range scan based on the current position of the robot and the desired map

Flatsim

- flatsim stands for flatworld simulation



Flatsim

- Flatsim is a package of ISAAC
- Start flatsim from inside the Isaac source tree with the following command:
  - $ cd sdk/packages/flatsim/apps
  - $ bazel run flatsim
  - Load the Sight web interface at http://localhost:3000 in a web browser
- The flatsim application starts by default in random walk mode
- In this mode, a dot, representing the robot, navigates to a random target on the map
- When the robot reaches the target, a new target is determined, and the robot moves towards that target

Example 9.6

- Flatsim is a package of ISAAC
- Start flatsim from inside the Isaac source tree with the following command:
    - $ cd sdk/packages/flatsim/apps
    - $ bazel run flatsim
    - Load the Sight web interface at http://localhost:3000 in a web browser
- The flatsim application starts by default in random walk mode
- In this mode, a dot, representing the robot, navigates to a random target on the map
- When the robot reaches the target, a new target is determined, and the robot moves towards that target
- As a first experiment with flatsim, perform the following steps to add an interactive marker to the map
- You can click and drag this marker to different locations on the map
- When you do, the robot changes course to move to the new location of the marker

Example 9.6

- Move the robot with a marker
- Load the Sight web interface at http://localhost:3000 in a web browser.
    - The robot dot begins to navigate to the first random target.
- In the configuration section on the right side of the Sight web interface, in “goals.goal_behavior”, change the desired behavior from “random” to “pose”.
- The robot stops moving because you have changed its behavior but you have not yet added the interactive marker that you will configure as a goal for the new behavior.
- Right-click in the “flatsim - Map View” window and choose Settings.
- In Settings, click the Select marker dropdown menu and choose “pose_as_goal”.
- Click Add marker.
- Click Update. The marker is added to the map, separate from the random walk target, which is still displayed. You may need to zoom in on the map to see the new marker. The robot does not immediately begin navigating to the marker.
- Click and drag the marker to a new location on the map. The robot will begin to navigate to the marker location.

Example 9.6

- Modify the Robot Spawn Position
    - Open the //packages/flatsim/apps/2d_differential_base_simulation.subgraph.json
    - locate the config.robot_spawn_pose.PoseInitializer.pose section
    - modify the “rotation” and “translation” values as needed

Integration with ROS (ROS bridge)

- ISAAC SDK can be easily integrated with ROS
- The main source of integration is the communication pipeline
- A set of functions are already available to convert Proto data to ROS data and vice-versa
    - Communicating between Isaac and ROS requires creating a message translation layer between the two systems
- The ros_bridge package provides a library of message converters and makes it easy to create new ones
- Users can create various bridges with different connections and configurations of converters, and easily create new converters if needed.
- An Isaac-ROS bridge consists of:
    1. One and only one RosNode codelet,
    2. A TimeSynchronizer codelet in the same node as RosNode codelet,
    3. As many message and pose converter codelets as needed,
    4. A behavior tree that starts converters once RosNode establishes connection with the roscore.

Integration with ROS (ROS bridge)

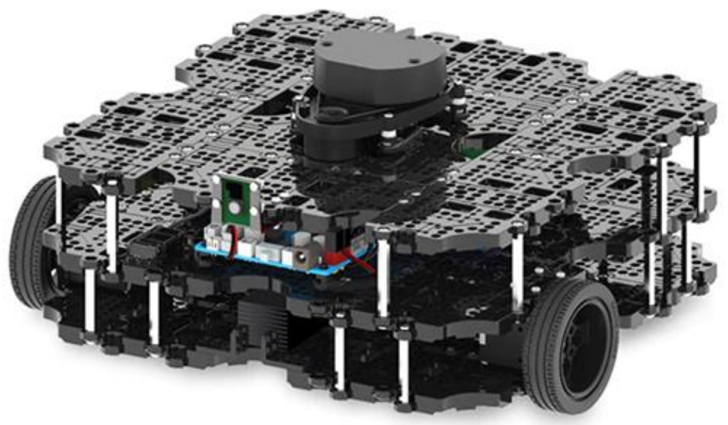
- RosNode
  - RosNode is the Isaac codelet that initializes a ROS node and waits until roscore is up
  - Every Isaac application with ROS bridge needs to have one and only one node with a single component of this type
- TimeSynchronizer
  - Allows converting time stamps between Isaac notation and ROS notation
- Message Converter Bases
  - Isaac provides users with base classes to quickly develop typical converters:
    - ProtoToRosConverter: This is a base class for codelets that convert Isaac proto messages to ROS messages and publish them to ROS
      - Please check OdometryToRos converter to see an example on how to create a new converter using ProtoToRosConverter
      - All we need to do is to define a protoToRos function.
    - RosToProtoConverter: This is a base class for codelets that receives ROS messages and convert them to Isaac proto messages
      - RosToDifferentialBaseCommand converter to see an example on how to create a new converter using RosToProtoConverter: All we need to do is to define a rosToProto function.

Example 10.6

- Convert a video stream taken with a camera node into a sensor_msgs::Image

Navigation stack ISAAC -> ROS

- ROS navigation stack can be integrated with ISAAC SDK
- From ROS
  - Localization
  - Map server
  - Move base (path planning)
- Use flatsim as simulator
  - LASER data
  - Odometry calculation
  - Interface
- ISAAC and SDK share the map



Example 11.6

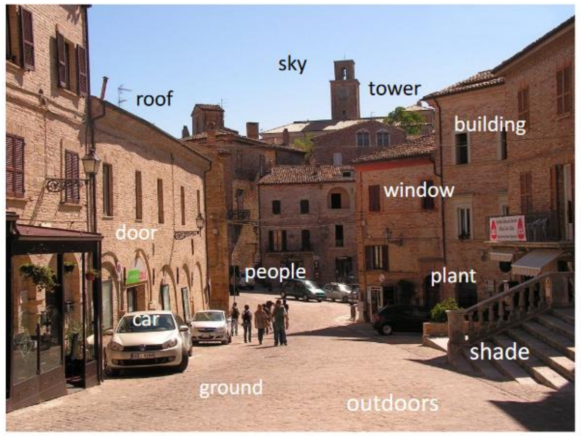
- To use the ROS Navigation stack with ISAAC SDK
  - Install turtlebot3 files
    - $ sudo apt-get install ros-melodic-turtlebot3*
  - Move in the Isaac sdk root directory
  - $ TURTLEBOT3_MODEL=waffle_pi roslaunch turtlebot3_navigation turtlebot3_navigation.launch map_file:=$(realpath packages/ros_bridge/maps/small_warehouse.yaml)
    - If you run this command from another directory, the map will not be read correctly
  - $ bazel run packages/ros_bridge/apps:ros_to_navigation_flatsim -- --more apps/assets/maps/virtual_small_warehouse.json –config ros_navigation:packages/ros_bridge/maps/small_warehouse_map_transformation.config.json,ros_navigation:packages/ros_bridge/apps/ros_to_navigation_turtlebot3_waffle_pi.config.json
  - Open http://localhost:3000/ to monitor through Isaac. Watch RViz window to monitor through ROS
  - The robot should now be navigating to the goal, which can be easily modified by dragging the “pose_as_goal” marker of “Map” window on Sight around.

Computer Vision

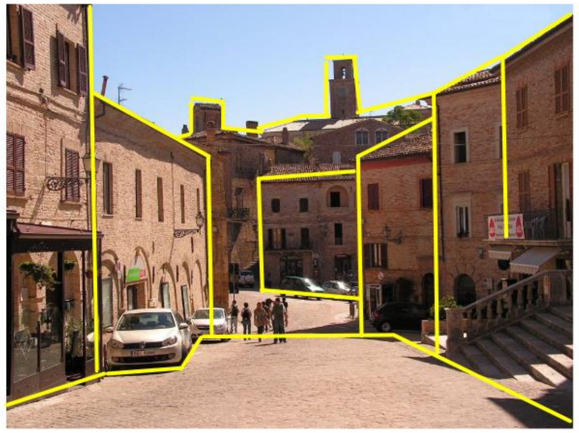
- Computer vision in robot vision is an important aspect for several tasks like manipulation and navigation
- Several vision sensors are available on the market
- To program camera sensors we need to interface them to the onboard computer of the robot
- This can be made mainly in two ways:
  - Using operating system driver
  - Vendor driver
- Standard USB camera (like webcams) are directly accessible using low level routine provided by the operating system
- If the camera is working by default with Ubuntu/Linux systems, generic drivers can be used
- After plugged the camera, check whether a /dev/videoX device file has been created
- You can check with some application such as Cheese, VLC, or similar others
- The video devices present on the system using the following command:
  - `$ ls /dev/ | grep video`

Computer Vision

- OpenCV is one of the most famous library used to elaborate images
- Automatic extraction of “meaningful” information from images and videos
- Computer vision
    - An interdisciplinary scientific field that deals with how computers can gain high-level understanding from digital images
    - From the perspective of engineering, it seeks to understand and automate tasks that the human visual system can do
    - Common functionalities:
        - Thresholding
        - Binarization
        - Features detection
        - Visual odometry

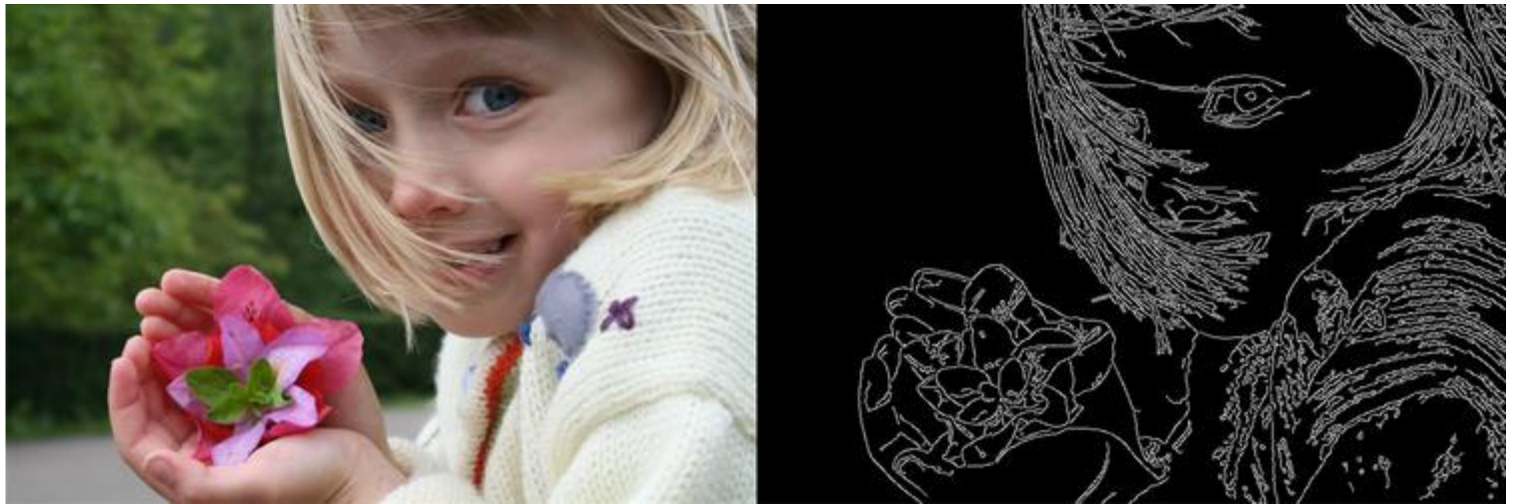


Semantic information


Geometric information

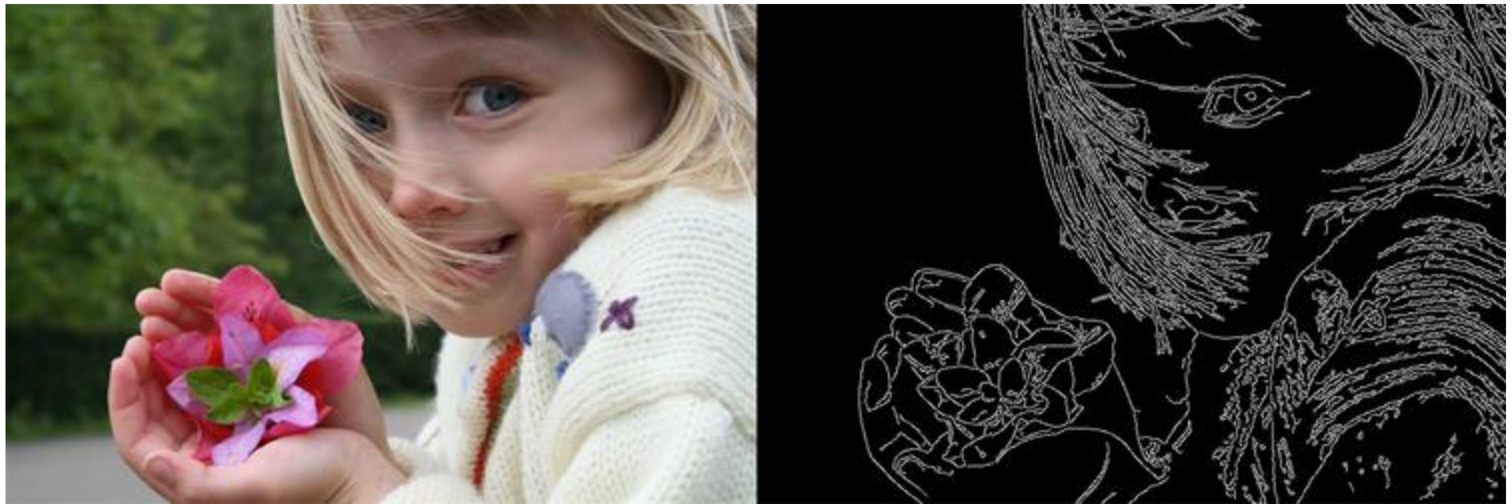
Edge detection

- Edge detection is one of the most importante feature extraction method to recognize elements in an image
- The whole image is elaborated to extract the contours of all the object in the scene
- Example
  - Fingerprint recognition: When recognizing fingerprints, it’s useful to preprocess the image by performing edge detection.
    - In this case, the “edges” are the contours of the fingerprint, as contrasted with the background on which the fingerprint was made. This helps reduce noise so that the system can focus exclusively on the fingerprint’s shape.
- The whole image is elaborated to get the variation of the gradient of the pixles



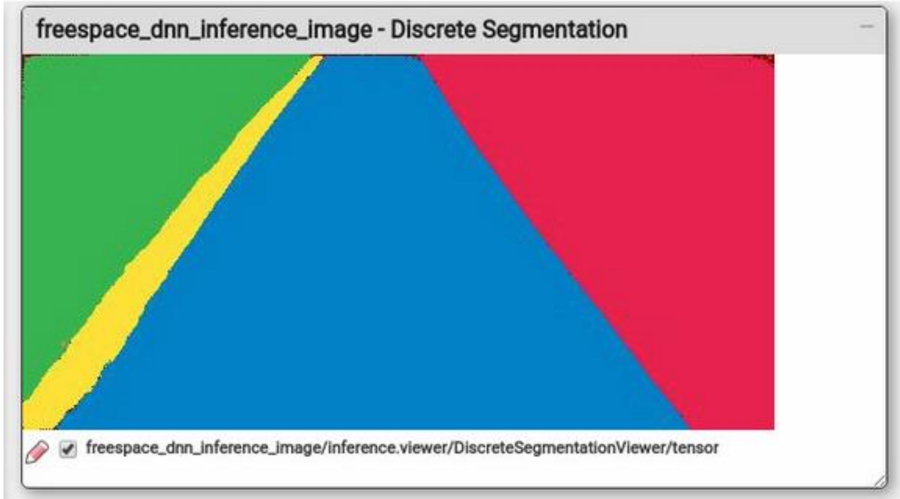
Example 12.6

- Use opnecv to perform the edge detection of a camera stream
  - Use also a recorded image file

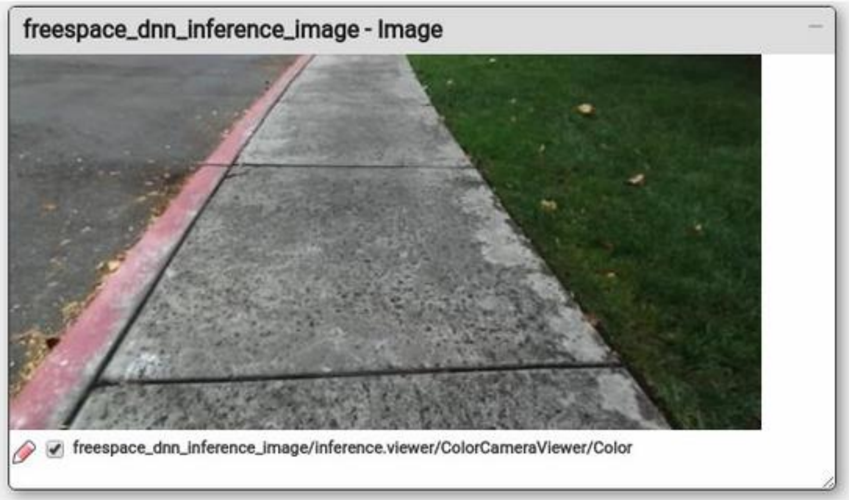


Freespace segmentation

- The goal of the free space Deep Neural Network (DNN) is to segment images into classes of interest like drivable space and obstacles
- The input of the DNN is a monocular image, and the output is pixel-wise segmentation
- This package makes it easy to train a free space DNN in simulation and use it to perform real-world inference
- While this modular package can power various applications, this document illustrates the workflow with free space segmentation for indoors and sidewalk segmentation for outdoors.


freespace_dnn_inference_image - Discrete Segmentation

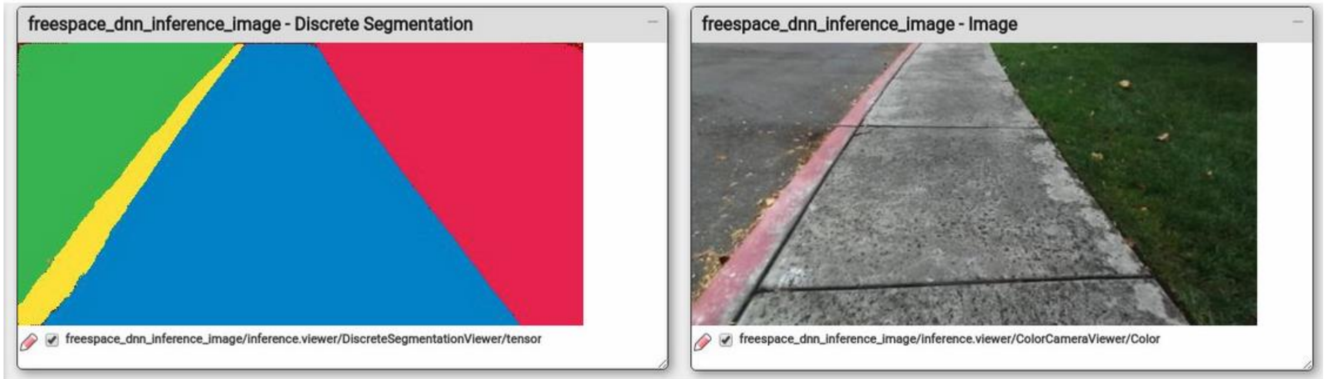
freespace_dnn_inference_image/inference.viewer/DiscreteSegmentationViewer/tensor


freespace_dnn_inference_image - Image

freespace_dnn_inference_image/inference.viewer/ColorCameraViewer/Color

Freespace segmentation

- $ cd sdk/packages/freespace_dnn/apps
- $ bazel run freespace_dnn_interface_image
  - 2022-10-13 07:36:06.404 ERROR external/com_nvidia_isaac_engine/engine/alice/backend/modules.cpp@295: packages/ml/libml_module.so: libnvrtc.so.10.2: cannot open shared object file: No such file or directory
  - 2022-10-13 07:36:06.404 ERROR external/com_nvidia_isaac_engine/engine/alice/backend/modules.cpp@295: packages/perception/libperception_module.so: /home/jcacace/.cache/bazel/_bazel_jcacace/33446a341a1d88054d78d345b8059395/execroot/com_nvidia_isaac_sdk/bazel-out/k8-opt/bin/packages/freespace_dnn/apps/freespace_dnn_inference_image.runfiles/com_nvidia_isaac_sdk//packages/perception/libperception_module.so: undefined symbol: IsaacGatherComponentInfo
  - 2022-10-13 07:36:06.404 PANIC external/com_nvidia_isaac_engine/engine/alice/backend/modules.cpp@297: Could not load all required modules for application


freespace_dnn_inference_image - Discrete Segmentation
freespace_dnn_inference_image/inference.viewer/DiscreteSegmentationViewer/tensor
freespace_dnn_inference_image - Image
freespace_dnn_inference_image/inference.viewer/ColorCameraViewer/Color