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

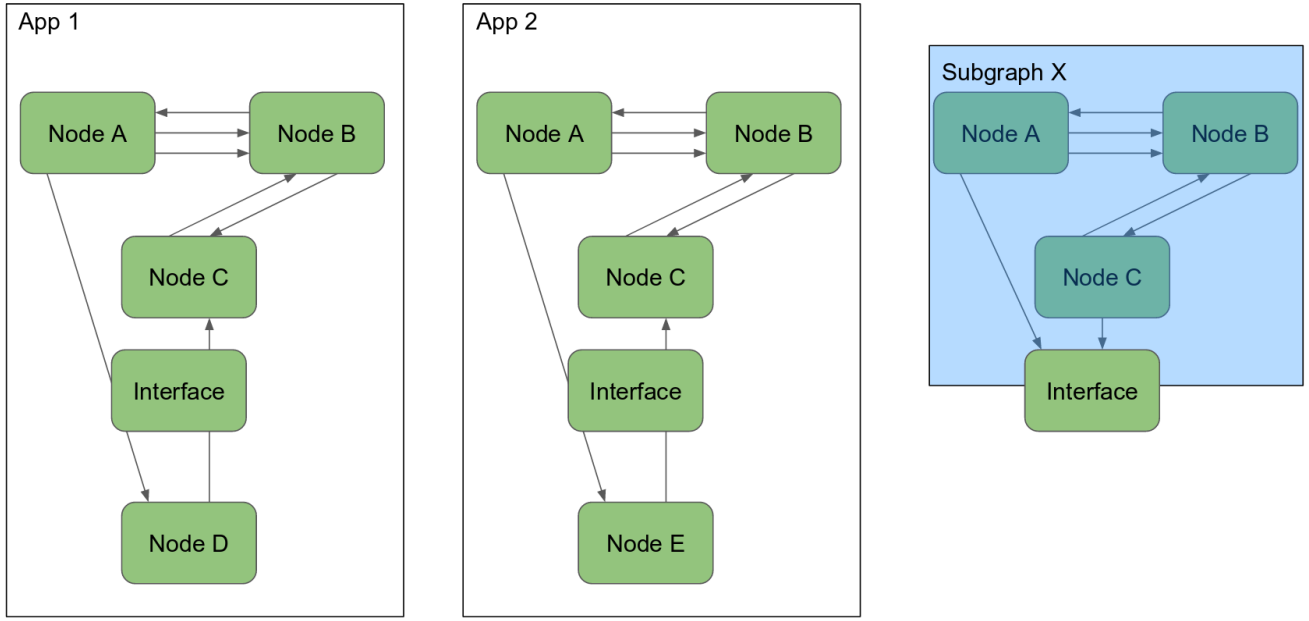
- Complicated algorithms can often be parameterized in various different ways
  - ISAAC_PARAM allows you to define a configuration parameter which can be set via configuration, read in the code, and changed in the frontend
- Maximum acceleration to use (helps with noisy data or wrong data from simulation)
  - 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

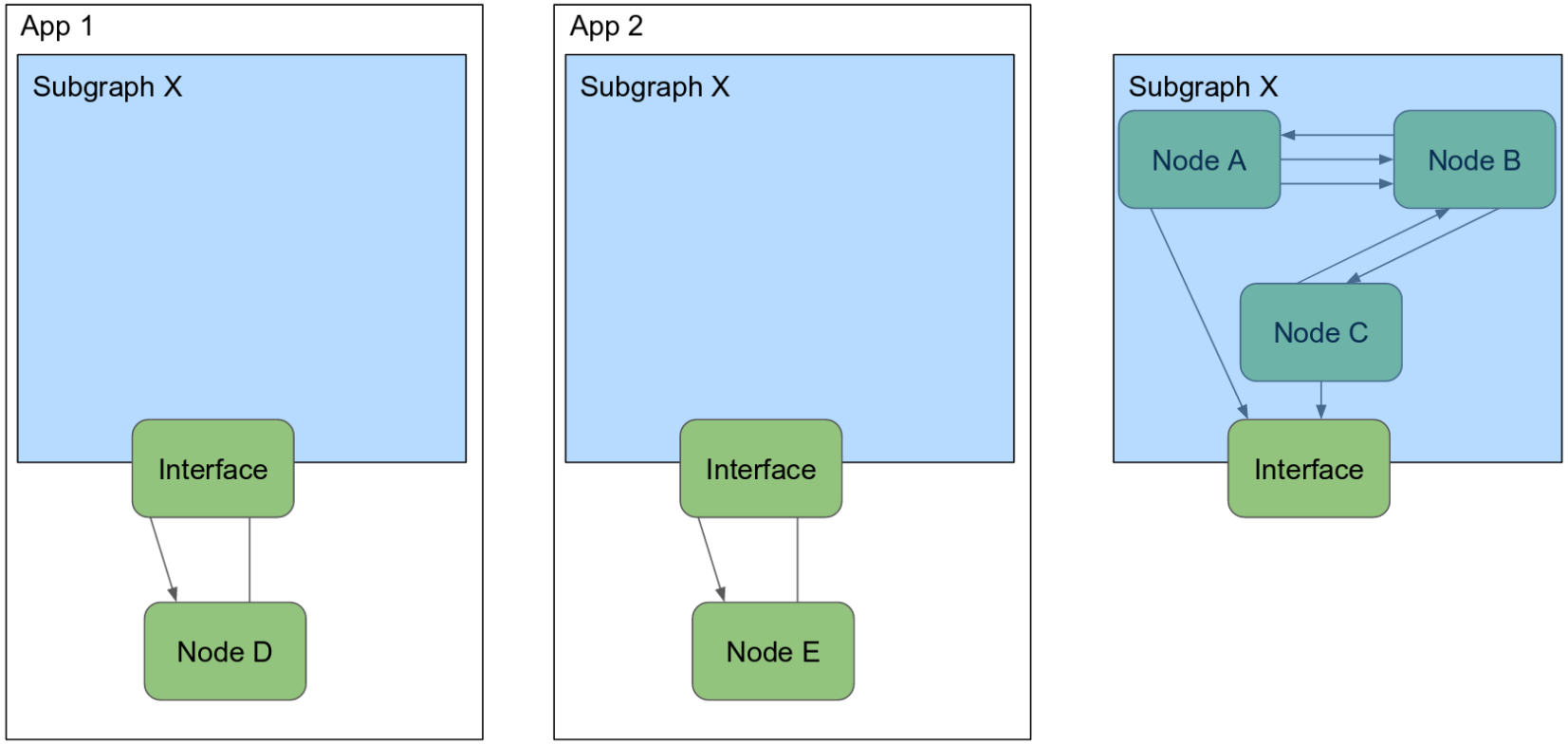
Sub-graphs

- 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