ANPL

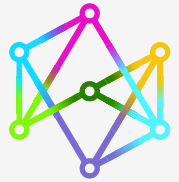
Autonomous Navigation and Perception Lab

Software Infrastructure Workshop

Asaf Feniger

29/11/2017

Software Infrastructure Goals

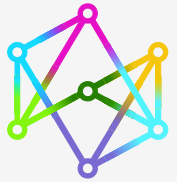


- Create infrastructure to evaluate ANPL research on single/multi robot SLAM & BSP in realistic simulation and real-world experiments.
- Allow researchers to implement only what they need (use parts other created and fit their needs).
- Currently implemented for multi robot centralized case with ground robots using laser scans and odometry measurements.

Workshop Topics

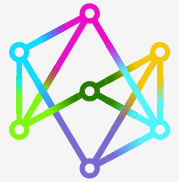
- **End to end full demo – MRBSP with 2 UGV in gazebo simulation.**
- Installation – How to install ANPL software
- ROS notations
- High level code structure – passive and active
- Launch files and software configuration
- how to run the code
- Code outputs
- Matalb analysis tools
- Matlab interface with the code
- Code handling policy
- Hands on exercises

MRBSP Demo



- 2 UGVs (Unmanned Ground Vehicle) in a simulated environment (Gazebo simulator).
- UGVs are equipped with 2D range sensor (LIDAR) and odometry sensor. The code also supports 3D range sensors for the creation of the map.
- Each robot listens to its own sensor and passes on only keyframes. Each robot has a state machine and a controller.
- All other calculations (DA, belief, map, planning) are centralized.
- Candidate actions are manually generated and the planner chooses the best set of actions according to a cost function.

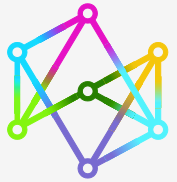
Workshop Topics



- **End to end full demo – MRBSP with 2 UGV in gazebo simulation.** ☑
- **Installation – How to install ANPL software**
- ROS notations
- High level code structure – passive and active
- Launch files and software configuration
- how to run the code
- Code outputs
- Matalb analysis tools
- Matlab interface with the code
- Code handling policy
- Hands on exercises

ANPL | Autonomous Navigation and Perception Lab

Installation



- This installation assumes Ubuntu 16.04 with ros-kinetic installed. It also assumes the ANPL environment is installed.
- Create new catkin workspace (named mrbsp_ws)

  $ mkdir –p ~/ANPL/code/mrbsp_ws/src

  $ cd ~/ANPL/code/mrbsp_ws
- Initialize catkin workspace and build it

  $ catkin init

  $ catkin build
- Source the workspace (and save it in .bashrc file)

  $ echo “source $HOME/ANPL/code/mrbsp_ws/devel/setup.bash” >> ~/.bashrc
- Clone mrbsp_ros package to src folder

  $ git clone https://bitbucket.org/ANPL/mrbsp_ros src/mrbsp_ros
- Clone anpl software workshop repository

  $ git clone https://bitbucket.org/ANPL/anpl_software_workshop src/anpl_software_workshop
- Build workspace

  $ catkin build

Code Updating

- If the code is already installed on the computer, make sure you have the latest version.

  ```
  $ cd ~/ANPL/code/mrbsp/src/mrbsp_ros
  $ git pull
  ```

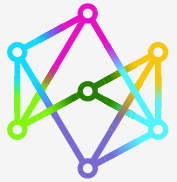
- Re-compile the code

  ```
  $ cd ../..
  $ catkin build
  ```

Workshop Topics

- **End to end full demo – MRBSP with 2 UGV in gazebo simulation.** ☑
- **Installation – How to install ANPL software** ☑
- **ROS notations**
- High level code structure – passive and active
- Launch files and software configuration
- how to run the code
- Code outputs
- Matalb analysis tools
- Matlab interface with the code
- Code handling policy
- Hands on exercises

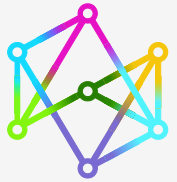
ROS Notations



The ANPL software is using ROS. Some key concepts:

- **ROSNODE** – A node is a process that performs computation. Nodes are combined together into a graph and communicate with one another using streaming topics, RPC services, and the parameter server. (link)
- **ROS master** – Provide naming and registration service to the rest of the nodes in the ROS system. (link)
- **ROS parameter server** – A shared, multi-variate dictionary that is accessible via network APIs. (link)
- **ROSCORE** – A collection of ROS nodes and programs that are pre-required of a ROS based system. ROSCORE enables communication between ROS nodes, and it start up ROS master and ROS parameter server. (link)

ROS Notations



- **ROS package** – Software in ROS is organized in packages. A package might contain ROS nodes, a ROS independent library, a dataset, configurations files, a third party piece of software or anything else that logically constitutes a useful module. (link)
- **ROS (catkin) workspace** – A folder where you modify, build, and install catkin (ROS) packages. (link)
- **ROSLAUNCH** – A tool for easily launching multiple ROS nodes locally and remotely via SSH, as well as setting parameters on the parameters server. (link)
- **RVIZ** – A tool to visualize ROS topics and other information. RVIZ also allow user interface with ROS.

Workshop Topics

- **End to end full demo – MRBSP with 2 UGV in gazebo simulation.** ☑
- **Installation – How to install ANPL software** ☑
- **ROS notations** ☑
- **High level code structure – passive and active**
- Launch files and software configuration
- how to run the code
- Code outputs
- Matalb analysis tools
- Matlab interface with the code
- Code handling policy
- Hands on exercises

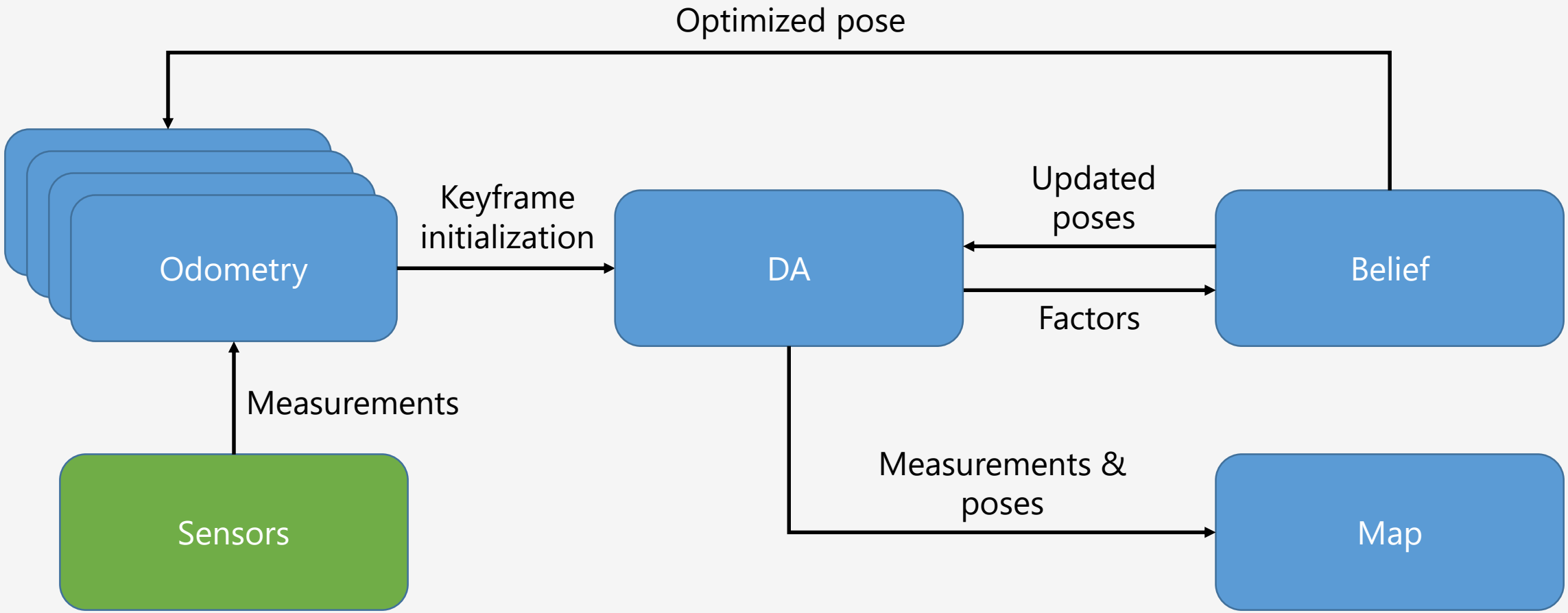
High level code structure

- The code can be divided to 4 operating modes:

	Passive	Active
SR	Single robot passive slam	Single robot active slam
MR	Multi robot passive slam	Multi robot active slam

- In addition, the code can be implemented as centralized / decentralized system.

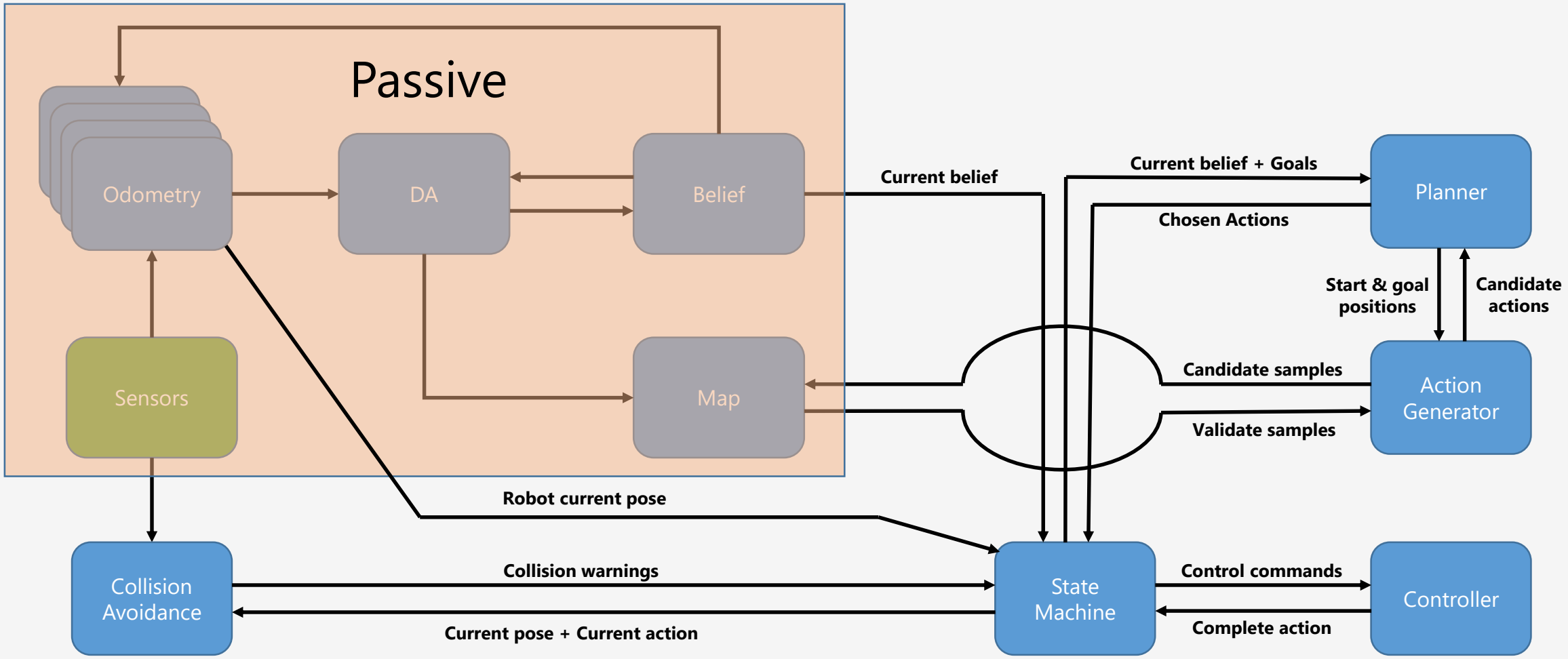
Passive SLAM Data Flow



Passive SLAM

- Packages:
  - Odometry node – Keyframe selections, calculate relative motion between keyframes. Calculate the robot current pose.
  - DA node – Create factors and initialize state for the belief.
  - Belief node – Belief optimization.
  - Map node – Simulate scans from visited places, samples and trajectory validations.
- Passive SLAM runs in a loop until the user decides to stop it.

Active SLAM Data Flow



Active SLAM

- Packages :
  - Passive nodes
  - State machine node – Decide which active node will be call.
  - Action generator node – Generating candidate actions for the planner node.
  - Planner node – Evaluate candidate actions and return to state machine node the chosen actions.
  - Controller node – Translate the actions to the robot actuators.
  - Collision avoidance node – Check for immediate collisions using current scans.
- The active SLAM is managed by the state machine.

Utility Packages

- Packages:
  - Mrbsp ros msgs – Provide custom ROS messages and service to exchange data between nodes.
  - Mrbsp ros utils – Provide headers with utility functions.
  - Config node – Generate global setting for the experiment (e.g. creating a results folder).
  - Mrbsp scenarios – Contains configuration files to run different scenarios. Results are stored in this packages.
  - Rosbag node – Provide a mechanism to run the code by reading sensors data directly from bagfiles.

Code Structure - Summery

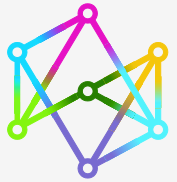
- ANPL's software code has 14 ROS packages.

Utility packages	Passive SLAM	Active SLAM
Mrbsp ros msgs	Odometry node	State machine node
Mrbsp ros utils	Da node	Action generator node
Config node	Belief node	Planner node
Mrbsp scenarios	Map node	Controller node
Rosbag node *		Collision detection node *

(*) - not implemented yet.

- Each package contains executables with the same purpose.
- Each package contains a base class, and all executable in the package inherit from it. The base class contains member and function that are common to the package executables.
- When running the code only one executable from each package should be executed.
- Not all packages need to be involved.
- Implementation is possible with C++, python and Matlab.

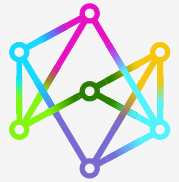
Workshop Topics



- **End to end full demo – MRBSP with 2 UGV in gazebo simulation.** ☑
- **Installation – How to install ANPL software** ☑
- **ROS notations** ☑
- **High level code structure – passive and active** ☑
- **Launch files and software configuration**
- how to run the code
- Code outputs
- Matalb analysis tools
- Matlab interface with the code
- Code handling policy
- Hands on exercises

ANPL | Autonomous Navigation and Perception Lab

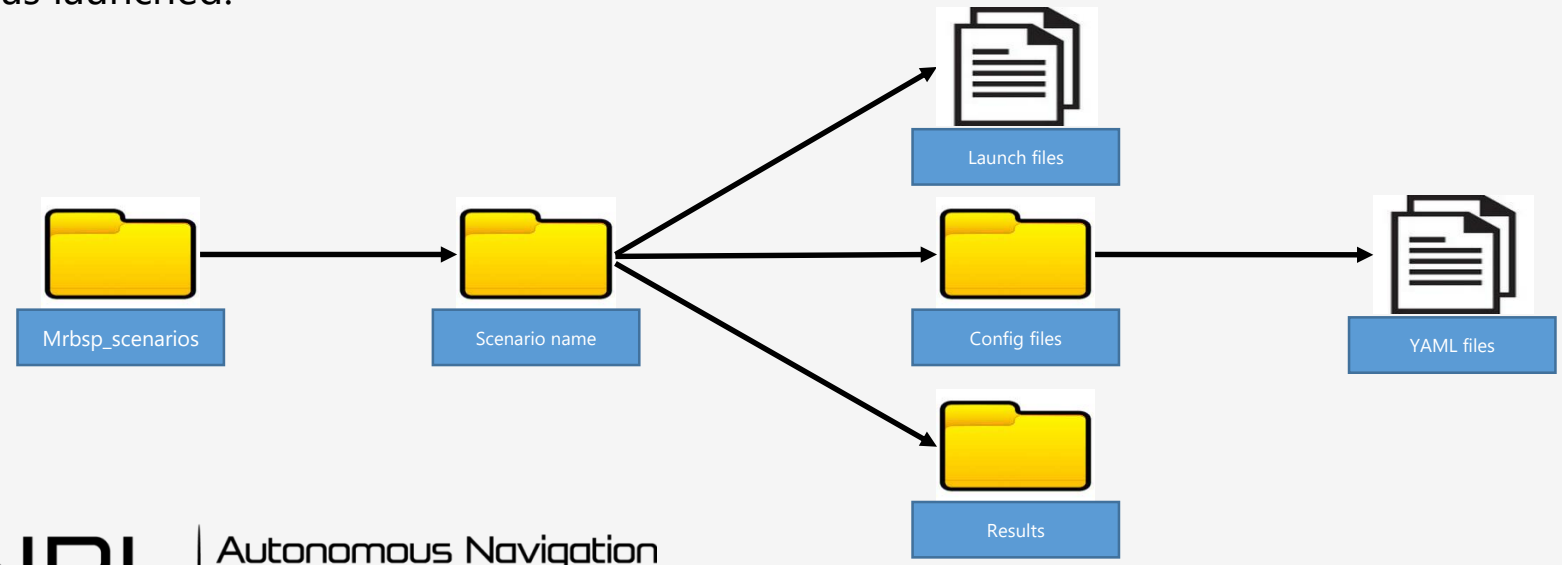
Launch Files and Software Configuration



- ROS provide tools to easily run executables and load parameters into the code without re-compiling.
- For more info, visit ros wiki pages:
  - ros launch files: http://wiki.ros.org/roslaunch
  - ros parameters server: http://wiki.ros.org/Parameter%20Server
  - ros parametes tutorial: http://wiki.ros.org/roscpp_tutorials/Tutorials/Parameters
- Launch files and configuration are created for each scenario.

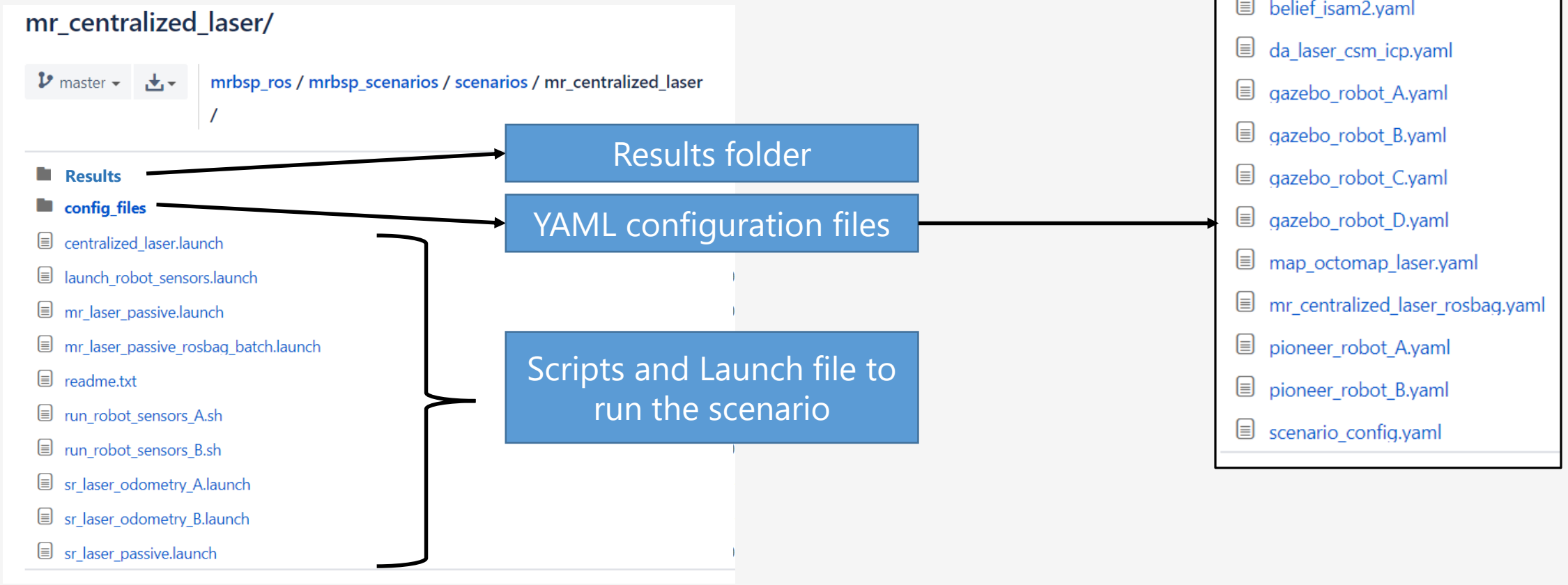
Scenario Definition

- Scenarios are defined in mrbsp_scenarios package.
- Each scenario is a set of executables executed from launch files. One from each package.
- Each scenario contains launch files and configurations files relevant to its implementation.
- Each time a scenario is launched the results are stored in a results folder under the researcher name and the time the code was launched.



Scenario Definition

- mr_laser_centralized scenario folder example:



Launch Files

```
<launch>

  <!-- Parameters to change each time-->
  <arg name="current_scenario"  default="mr_centralized_laser" />

  <arg name="scenario_dir"      default="$(find mrbsp_scenarios)/scenarios" />
  <arg name="config_files_dir"  default="$(arg scenario_dir)/$(arg current_scenario)/config_files" />
  <arg name="robot_name" default="Robot_A" />

  <!-- Putting the time back to real time-->
  <rosparam> /use_sim_time : true </rosparam>
  <param name="scenario_folder" value="$(arg scenario_dir)/$(arg current_scenario)"/>

  <node pkg="odometry_node" type="odometry_laser_node" name="$(arg robot_name)" clear_params="true" output="screen">
    <rosparam command="load" file="$(arg config_files_dir)/gazebo_robot_A.yaml" />
  </node>

  <node pkg="controller_node" type="controller_pioneer_node" name="$(arg robot_name)_controller_pioneer" clear_params="true" output="screen">
    <rosparam command="load" file="$(arg config_files_dir)/controller_pioneer.yaml" />
  </node>

  <node pkg="state_machine_node" type="state_machine_default_node" name="$(arg robot_name)_state_machine_default" clear_params="true" output="screen">
    <rosparam command="load" file="$(arg config_files_dir)/state_machine_default.yaml" />
    <!-- Load robot goals -->
    <rosparam command="load" file="$(arg config_files_dir)/gazebo_Robot_A_goals.yaml" />
  </node>

  <!-- Create static transformation between the laser frame id to the robot pose for laser drawing in rviz -->
  <node pkg="tf" type="static_transform_publisher" name="$(arg robot_name)_laser_link" args="0.27 -0.03 0.13 0 0 0 1 pioneer1/base_link pioneer1/hokuyo_link 100" />

  <node pkg="tf" type="static_transform_publisher" name="$(arg robot_name)_base_link" args="0.0 0.0 0 0 0 0 1 $(arg robot_name) pioneer1/odom 100" />

  </launch>
```

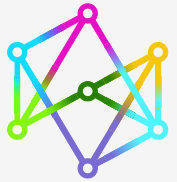
create arguments in the current launch file

create ros parameters without namespace

load ros parameters from yaml file. Prams get node name as namespace

run executables

YAML Configuration Files



```
# parameters for robots odometry
robot_name: pioneer1
robot_id: A
init_pose: {
  position:    {X: -09.22, Y: -19.36, Z: 0.0},
  orientation: {Yaw: 1.570796327, Pitch: 0.0, Roll: 0.0}
}
topics: {pulse: /odom, odom: /odom, laser: /laser/scan, ground_truth: /ground_truth}
sensor_pose: {
  odom: {
    position:    {X: 0.0, Y: 0.0, Z: 0.0},
    orientation: {qX: 0.0, qY: 0.0, qZ: 0.0, qW: 1.0}
  },
  laser: {
    position:    {X: 0.0, Y: 0.0, Z: 0.0},
    orientation: {qX: 0.0, qY: 0.0, qZ: 0.0, qW: 0.0}
  },
  pointcloud: {
    position:    {X: 0.0, Y: 0.0, Z: 0.0},
    orientation: {qX: 0.0, qY: 0.0, qZ: 0.0, qW: 0.0}
  }
}

informative_condition: {distance: 2.0, yaw: 0.2618}

visualize_laser: true
is_3D_vis: false
index_timeout_threshold: 50
is_noised_odom: false
error_dynamic_percentage: 0.1
```

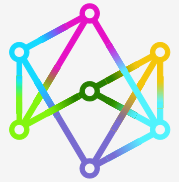
/robot_name

/sensor_pose/odom/orientation/qW

/informative_conditions/yaw

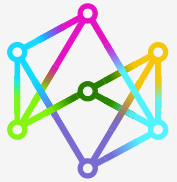
ANPL | Autonomous Navigation and Perception Lab

Reading ROS parameters in C++ File



- ROS global node handler - ros::NodeHandle nh
  - ➢ Parameter name does not have a namespace.
- ROS private node handler – ros::NodeHandler pnh(~)
  - ➢ Parameter name will start with the node name

Reading ROS parameters in C++ File



- Load param from launch file (not within a node):

```
<param name="scenario_folder" value="$(arg scenario_dir)/$(arg current_scenario)"/>
```

- Read parameter in C++ code:

```
std::string scenario_folder;
pnh.getParam("/scenario_folder", scenario_folder);
```

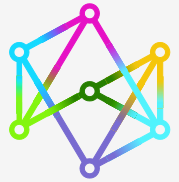
- Load param from yaml file (within a node):

```
robot_name: pioneer1
```

- Read parameter in C++ code:

```
std::string robot_name;
pnh.getParam("robot_name", robot_name);
```

Reading ROS parameters in C++ File



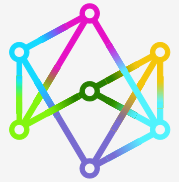
- Load param from yaml file (within a node):

```
sensor_pose: {
  odom: {
    position:    {X: 0.2, Y: 0.1, Z: 0.2},
    orientation: {qX: 0.0, qY: 0.0, qZ: 0.0, qW: 1.0}
  }
}
```

- Read parameter in C++ code (not from the node that load the parameter):

```
double X_sensor, Y_sensor, Z_sensor, qX, qY, qZ, qW;
std::string robot_ns = std::string("/robot_" + current_robot_id);
nh.param(std::string(robot_ns + "/sensor_pose/laser/position/X"), X_sensor, double(0.0));
nh.param(std::string(robot_ns + "/sensor_pose/laser/position/Y"), Y_sensor, double(0.0));
nh.param(std::string(robot_ns + "/sensor_pose/laser/position/Z"), Z_sensor, double(0.0));
nh.param(std::string(robot_ns + "/sensor_pose/laser/orientation/qX"), qX, double(0.0));
nh.param(std::string(robot_ns + "/sensor_pose/laser/orientation/qY"), qY, double(0.0));
nh.param(std::string(robot_ns + "/sensor_pose/laser/orientation/qZ"), qZ, double(0.0));
nh.param(std::string(robot_ns + "/sensor_pose/laser/orientation/qW"), qW, double(0.0));
```

Workshop Topics



- **End to end full demo – MRBSP with 2 UGV in gazebo simulation.** ☑
- **Installation – How to install ANPL software** ☑
- **ROS notations** ☑
- **High level code structure – passive and active** ☑
- **Launch files and software configuration** ☑
- **how to run the code**
- Code outputs
- Matalb analysis tools
- Matlab interface with the code
- Code handling policy
- Hands on exercises

ANPL | Autonomous Navigation and Perception Lab

Current Implementation

- Currently one scenario is implemented – MR passive/active with UGV's
- This scenario is using odometry sensor for odometry calculations, 2D planar ICP for data association, ISAM2 for belief generation, octomap occupancy grid for mapping, and rviz user interface to generate candidate actions.
- This implementation is a centralized passive/active SLAM.

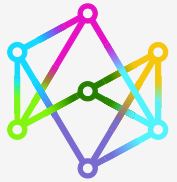
How to Run the Code

- In Ubuntu, open terminal window and run the following commands in separated terminals.

  use (ctrl + shift + T) to open a new terminal:

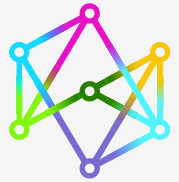
  1. Launch roscore: $ roscore
  2. Load simulated environment: $ roslaunch anpl_software_workshop pioneer3at_world.launch
  3. Run centralized nodes: $ roslaunch mrbsp_scenarios centralized_laser.launch
  4. Run robot A nodes: $ roslaunch mrbsp_scenarios robot_setup_gazebo_A.launch
  5. Run robot B nodes: $ roslaunch mrbsp_scenarios robot_setup_gazebo_B.launch

Launch ROSCORE



- Terminal command: $ roscore
- This command will:
  - ➢ Allow the ROS nodes to communicate with each other.
  - ➢ Start up ROS parameter server.

Load Simulated Environment



- Terminal command: $ roslaunch anpl_software_workshop pioneer3at_world.launch
- This command will:
  - ➢ Load Gazebo simulator with a chosen environment (world).
  - ➢ Spawn (add) robots into the simulator and simulate their sensors.

Load Simulated Environment

```
<launch>
  <arg name="world_name" default="willowgarage.world"/>
  <arg name="robot1_name" default="pioneer1"/>
  <arg name="robot2_name" default="pioneer2"/>

  <!-- <arg name="world" default="worlds/empty.world" /> -->
  <include file="$(find gazebo_ros)/launch/empty_world.launch">
        <arg name="world_name" value="$(find anpl_inf)/gazebo_worlds/$(arg world_name)" />
  </include>

    <!-- spawn the two robots -->
    <group ns="$(arg robot1_name)">
        <param name="tf_prefix" value="$(arg robot1_name)"/>
        <include file="$(find anpl_inf)/launch/spawn_pioneer3at.launch">
          <arg name="robot_name" value="$(arg robot1_name)" />
          <arg name="init_pose" value="-x -9.22 -y -19.36 -z 0 -Y 1.5708 -R 0.0 -P 0.0"/>
        </include>
    </group>

    <!-- spawn the two robots-->
    <group ns="$(arg robot2_name)">
        <param name="tf_prefix" value="$(arg robot2_name)"/>
        <include file="$(find anpl_inf)/launch/spawn_pioneer3at.launch">
          <arg name="robot_name" default="$(arg robot2_name)" />
          <arg name="init_pose" value="-x -3.3 -y 24.5 -z 0 -Y -1.5708 -R 0.0 -P 0.0"/>
        </include>
    </group>

</launch>
```

Launch file arguments

Load gazebo world

Spawn robot 1

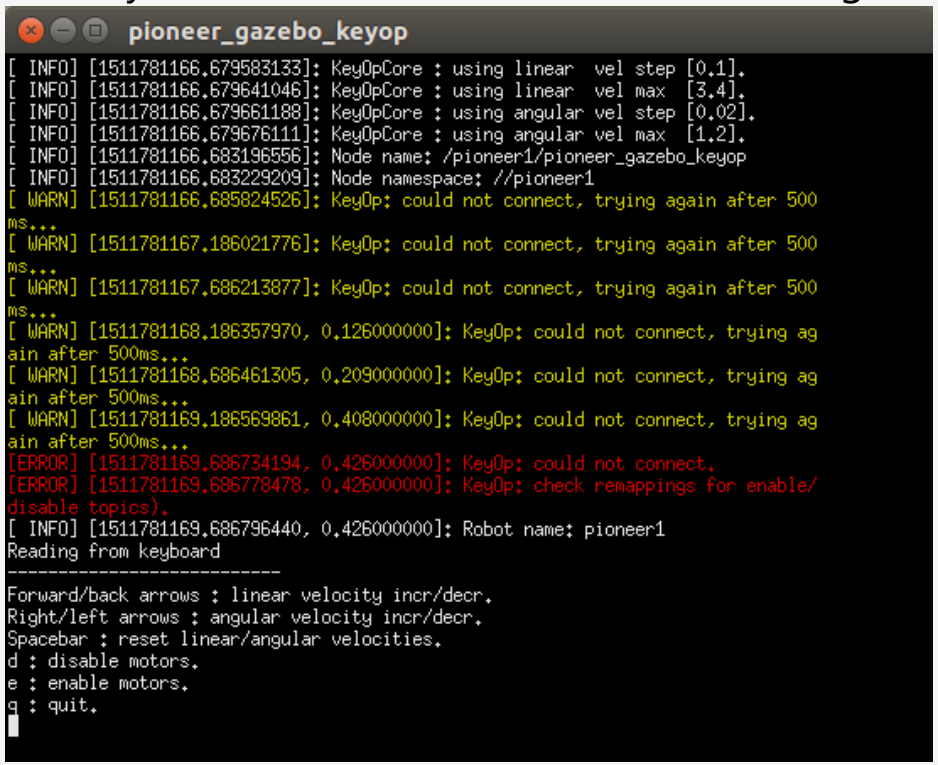
Spawn robot 2

Robot 2 arguments
Robot name & robot initial pose

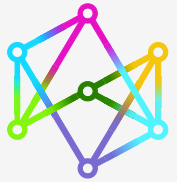
Keyboard Controller

In order to use the Keyboard controller:

1. Enable motors (press e)
2. Increase/decrease velocity with the keyboard arrows (the controller is moving the robot at constant velocity)

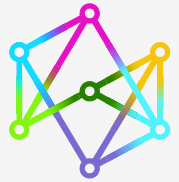


Run Centralized Nodes



- Terminal command: $ roslaunch mrbsp_scenarios centralized_laser.launch
- This command will:
  - ➢ Open all nodes in the centralized machine (config, da, belief, map, planner, actions generator).
  - ➢ Create folder for the log files and code outputs.
  - ➢ Open RVIZ interface to view live results and use the user interface planner.

Run Centralized Nodes – Utility Nodes



```
<launch>

  <!-- Parameters to change each time-->
  <arg name="researcher_name"   default="ANPL_Reasearcher" />
  <arg name="current_scenario"  default="mr_centralized_laser" />

  <arg name="scenario_dir"      default="$(find mrbsp_scenarios)/scenarios" />
  <arg name="config_files_dir"  default="$(arg scenario_dir)/$(arg current_scenario)/config_files" />

  <!-- Putting the time back to real time-->
  <rosparam> /use_sim_time : false </rosparam>

  <!-- Load robot params -->
  <group ns="robot_A">
    <rosparam command="load" file="$(arg config_files_dir)/pioneer_robot_A.yaml" />
  </group>
  <group ns="robot_B">
    <rosparam command="load" file="$(arg config_files_dir)/pioneer_robot_B.yaml" />
  </group>

  <param name="scenario_folder" value="$(arg scenario_dir)/$(arg current_scenario)"/>

  <node pkg="config_node" type="config_setup_node" name="config_setup" clear_params="true" output="screen">
    <rosparam command="load" file="$(arg config_files_dir)/scenario_config.yaml" />
    <param name="researcher_name" value="$(arg researcher_name)" />
  </node>
```

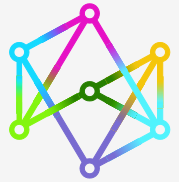
Launch file arguments

Global ROS parameters

Run config node

ANPL | Autonomous Navigation and Perception Lab

Run Centralized Nodes – DA Node



- **Launch file command:**

```
<node pkg="da_node" type="da_laser_node" name="da_laser" clear_params="true" output="screen">
  <rosparam command="load" file="$(arg config_files_dir)/da_laser_csm_icp.yaml" />
</node>
```

- **Implementation:**

  This implementation uses planar ICP method to determine data association between pose states. This algorithm uses laser scan measurements from the robot sensors. The algorithm generates prior and between factors for the belief.

- **Parameters file:**

```
# parameters for da_laser_icp node
loop_closure: { min_index_range: 5, spatial_search_range: 5}
# default: 10, 5
multi_robot: {spatial_search_range: 2}
# default: 2
```

ANPL | Autonomous Navigation and Perception Lab

Run Centralized Nodes – Belief Node

- **Launch file command:**

```
<node pkg="belief_node" type="belief_laser_node" name="belief_laser" clear_params="true" output="screen">
  <rosparam command="load" file="$(arg config_files_dir)/belief_isam2.yaml" />
</node>
```

- **Implementation:**

  This implementation uses ISAM 2 to calculate the belief. This node calculates the current belief (inference).

- **Parameters file:**

```
# parameters for belief_isam2 node
save: {matlab_results: true, serialized_files: true}
```

Run Centralized Nodes – Map Node

- **Launch file command:**

```
<node pkg="map_node" type="map_laser_node" name="map_octomap_laser" clear_params="true" output="screen">
  <rosparam command="load" file="$(arg config_files_dir)/map_octomap_laser.yaml" />
</node>
```

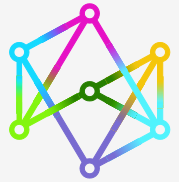
- **Implementation:**

  This implementation uses Octomap to generate map form the laser scans.

- **Parameters file:**

```
# parameters for map_octomap node
map_filename: octomap.ot
map_resolution: 0.005
```

Run Centralized Nodes – Planner Node



- **Launch file command:**

```
<node pkg="planner_node" type="Planner" name="ui_planner" clear_params="true" output="log">
</node>
```

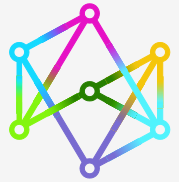
- **Implementation:**

  This implementation waits for all robots to request to plan. It receives candidate actions from the actions generator, simulates measurements (ML assumption, fixed noise model), evaluates objective function either in C++ or Matlab and returns to the state machines the best actions for each robot.

- **Parameters:**

```
<param name="planner_type" value="cpp" /> <!-- or "matlab"-->
<param name="unit_tests" value="false" />
```

Run Centralized Nodes – Action Generator Node



- **Launch file command:**

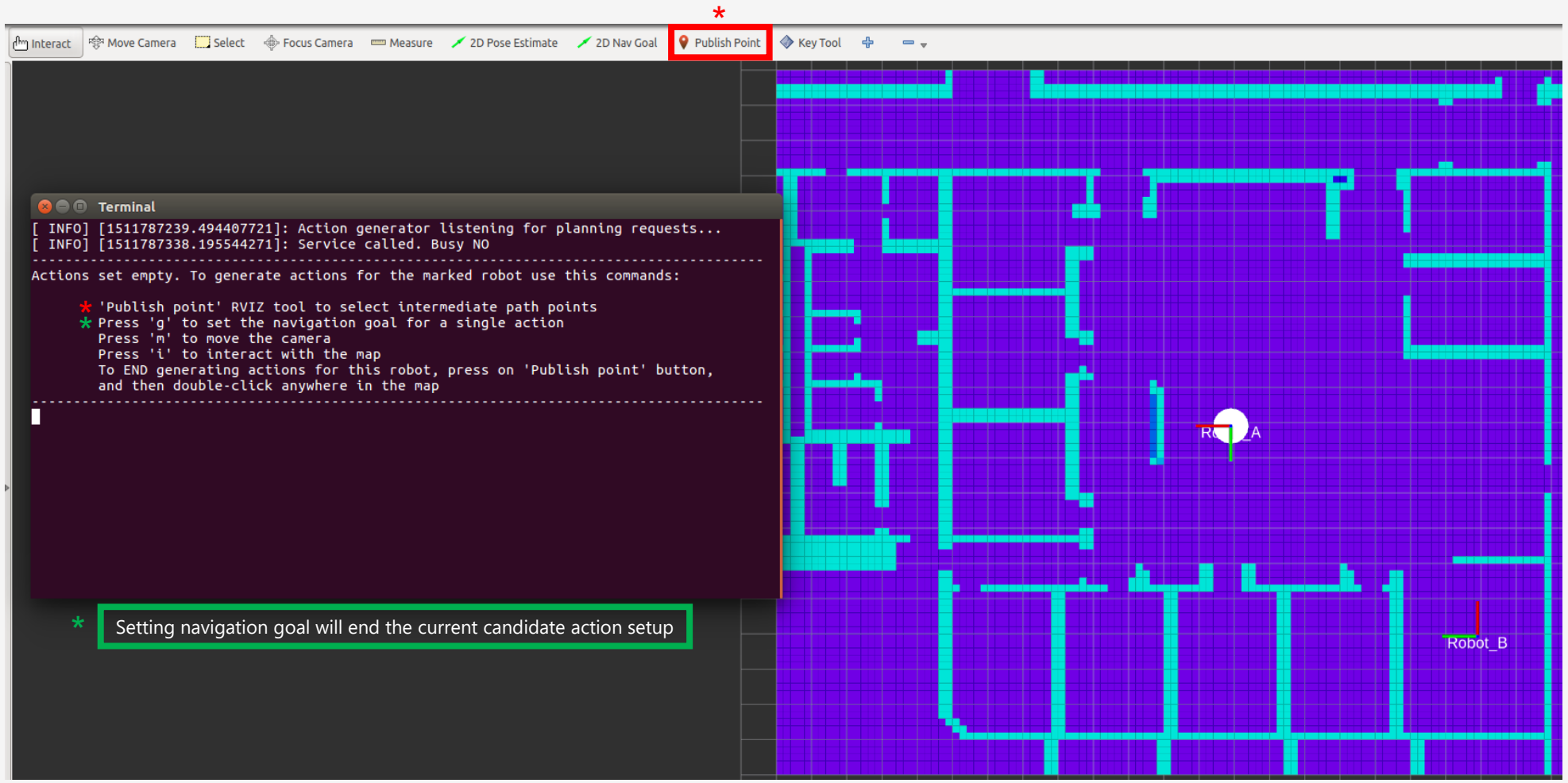
```
  <node pkg="action_generator_node" type="action_generator_node_node" name="ui_rviz_action_generator"
 clear_params="true" output="screen" launch-prefix="gnome-terminal --command">
  </node>

  <include file="$(find action_generator_node)/launch/generate_actions_interactively.launch">
  </include>

</launch>
```

- **Implementation:**

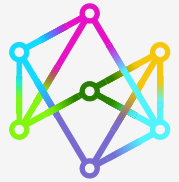
  Current implementation allows the user to manually generate candidate path using RVIZ interface.

  In the near future, this implementation will be **extended** (sampling based action generation, motion primitives)

Run Centralized Nodes – RVIZ User Interface

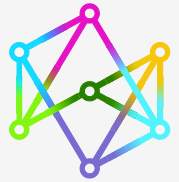


Run Robot "A" Nodes



- Terminal command: $ roslaunch mrbsp_scenarios robot_setup_gazebo_A.launch
- This command will:
  - ➢ Open all nodes on robot A (odometry, state machine, controller).
  - ➢ Use the robot A sensor to provide inputs to the code.
  - ➢ Move robot A according to the chosen trajectory.

Run Robot "A" Nodes – Utility Nodes



```
<launch>

  <!-- Parameters to change each time-->
  <arg name="current_scenario"  default="mr_centralized_laser" />

  <arg name="scenario_dir"      default="$(find mrbsp_scenarios)/scenarios" />
  <arg name="config_files_dir"  default="$(arg scenario_dir)/$(arg current_scenario)/config_files" />
  <arg name="robot_name" default="Robot_A" />

  <!-- Putting the time back to real time-->
  <rosparam> /use_sim_time : true </rosparam>
  <param name="scenario_folder" value="$(arg scenario_dir)/$(arg current_scenario)"/>

  <!-- Create static transformation between the laser frame id to the robot pose for laser drawing in rviz -->
  <node pkg="tf" type="static_transform_publisher" name="$(arg robot_name)_laser_link"
args="0.27 -0.03 0.13 0 0 0 1 pioneer1/base_link pioneer1/hokuyo_link 100" />

  <node pkg="tf" type="static_transform_publisher" name="$(arg robot_name)_base_link"
args="0.0 0.0 0 0 0 0 1 $(arg robot_name) pioneer1/odom 100" />
```

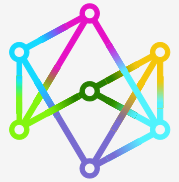
Launch file arguments

Global ROS parameters

Set transfer function between the robot model in gazebo and the code

ANPL | Autonomous Navigation and Perception Lab

Run Robot “A” Nodes – Odometry Node



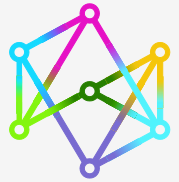
- **Launch file command:**

```
<node pkg="odometry_node" type="odometry_laser_node" name="$(arg robot_name)" clear_params="true" output="log">
  <rosparam command="load" file="$(arg config_files_dir)/gazebo_robot_A.yaml" />
</node>
```

- **Implementation:**

  This implementation uses odometry sensor to calculate relative motion and decides when the robot reaches a new keyframe. It passes relative motion information with laser scans to the da node.

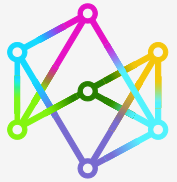
Run Robot "A" Nodes – Odometry Node



- **Parameters file:**

```
# parameters for robots odometry
robot_name: Robot_A
robot_id: A
init_pose: {
  position:    {X: -9.008, Y: 11.646, Z: 0.0},
  orientation: {Yaw: 0.0, Pitch: 0.0, Roll: 0.0}
}
topics: {pulse: /rosaria/pose, odom: /rosaria/pose, laser: /scan, ground_truth: /ground_truth, pointcloud: /camera/depth/points}
sensor_pose: {
  odom: {
    position:    {X: 0.0, Y: 0.0, Z: 0.0},
    orientation: {qX: 0.0, qY: 0.0, qZ: 0.0, qW: 0.0}
  },
  laser: {
    position:    {X: 0.20.5, Y: 0.0, Z: 0.27.5},
    orientation: {qX: 0.0, qY: 0.0, qZ: 0.0, qW: 0.0}
  },
  pointcloud: {
    position:    {X: 0.26.5, Y: 0.0, Z: 0.12},
    orientation: {qX: 0.5, qY: -0.5, qZ: 0.5, qW: -0.5}
  }
}

informative_condition: {distance: 0.5, yaw: 0.2618}

visualize_laser: true
is_3D_vis: true
index_timeout_threshold: 50
is_noised_odom: false
error_dynamic_percentage: 0.1
```

Run Robot "A" Nodes – Controller Node



- **Launch file command:**

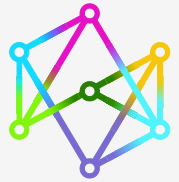
```
 <node pkg="controller_node" type="controller_pioneer_node" name="$(arg robot_name)_controller_pioneer"
clear_params="true" output="log">
   <rosparam command="load" file="$(arg config_files_dir)/controller_pioneer.yaml" />
   <param name="topics/velocity_cmd" value="/pioneer1/cmd_vel"/>
   <param name="robot_name" value="$(arg robot_name)"/>
 </node>
```

This implementation use PI controller to move the robot to a desired position.

- **Parameters file:**

```
# parameters for pioneer robot controller
topics: {velocity_cmd: /pioneer1/cmd_vel}
max_velocity: {linear: 0.5, angular: 0.15} # {[m/sec], [rad/sec]}
max_range: {linear: 5, angular: 0.785} # {[m], [rad]} pre filters for the controller - currently not in use
pid_constants: {
  linear:  {p: 0.100, i: 0.001, d: 0.000},
  angular: {p: 0.100, i: 0.001, d: 0.000}
}
controller_rate: -1 # [Hz] - currently not in use
# default: -1 - no controll of the rate

error_threshold: {linear: 0.1, angular: 0.1} # {[m], [rad]}
# defaults: {0.1, 0.1}
```

Run Robot "A" Nodes – State Machine Node



- **Launch file command:**

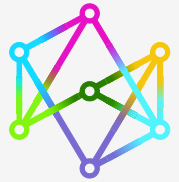
```
 <node pkg="state_machine_node" type="state_machine_default_node" name="$(arg robot_name)_state_machine_default"
clear_params="true" output="screen">
   <rosparam command="load" file="$(arg config_files_dir)/state_machine_default.yaml" />
   <!-- Load robot goals -->
   <rosparam command="load" file="$(arg config_files_dir)/gazebo_Robot_A_goals.yaml" />
 </node>
```

This implementation manages the active part of the code. According to a set of condition it decides when the robot plans, moves or finishes its run.

- **Parameters file:**

```
# parameters for the default state machine
robot_name: pioneer1
robot_id: A
```

Run Robot "B" Nodes

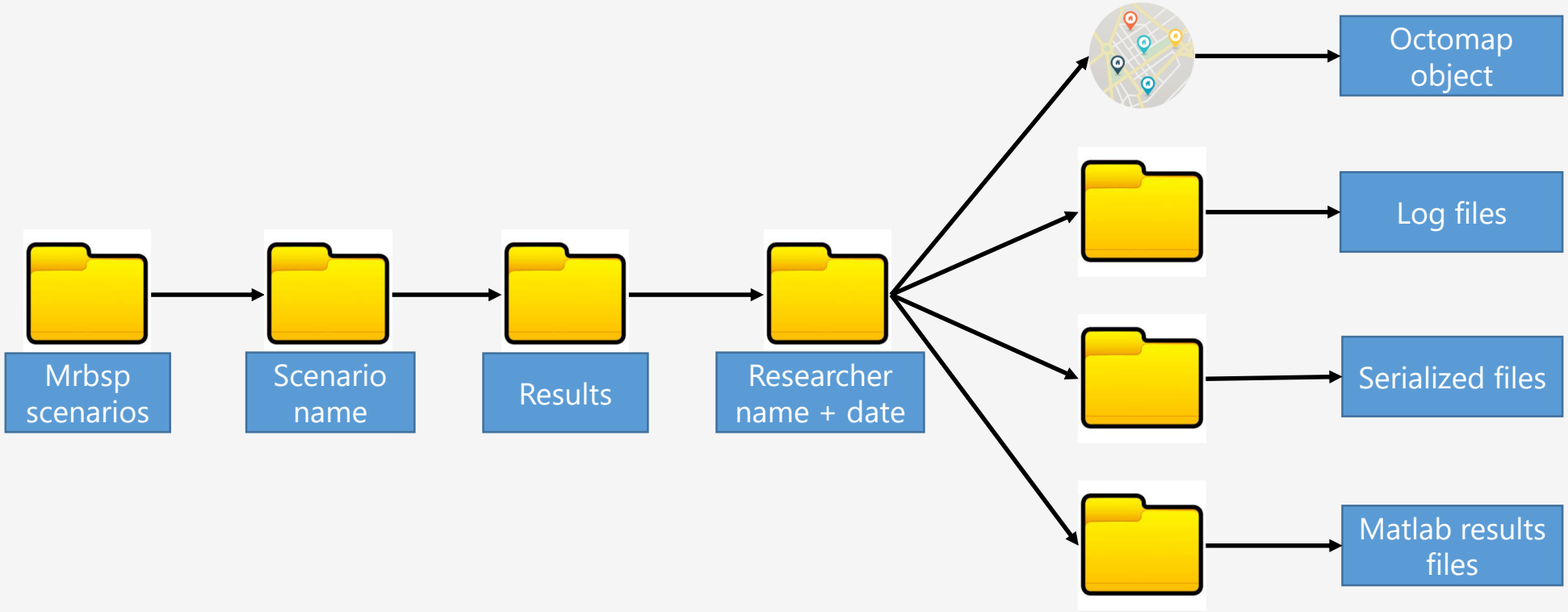


- Terminal command: $ roslaunch mrbsp_scenarios robot_setup_gazebo_B.launch
- This command will:
  - ➢ Open all nodes on robot B (odometry, state machine, controller).
  - ➢ Use the robot B sensor to provide inputs to the code.
  - ➢ Move robot B according to the chosen trajectory.

Workshop Topics

- **End to end full demo – MRBSP with 2 UGV in gazebo simulation.** ☑
- **Installation – How to install ANPL software** ☑
- **ROS notations** ☑
- **High level code structure – passive and active** ☑
- **Launch files and software configuration** ☑
- **how to run the code** ☑
- **Code outputs**
- Matalb analysis tools
- Matlab interface with the code
- Code handling policy
- Hands on exercises

Code Output


Mrbsp scenarios
Scenario name
Results
Researcher name + date
Octomap object
Log files
Serialized files
Matlab results files

Code Output – Octomap Object

- Use octovis to view final map.
- $ octovis octomap.ot

Code Outputs – Log files

- Each node generate a loge file.
- The logs files store information messages that the user defines. It has verbosity levels for filtering.

Open

```
[2017-11-27 09:12:30.469] [da_laser] [info] da node initialized...
[2017-11-27 09:13:05.150] [da_laser] [info] receive new 2D keyframe information...
[2017-11-27 09:13:05.150] [da_laser] [info] DA: Robot A, index 0: Matching in sliding window
[2017-11-27 09:13:05.150] [da_laser] [info] Add new robot, ID A
[2017-11-27 09:13:05.150] [da_laser] [info] DA: Robot A, keyframe0 initialized...
[2017-11-27 09:13:05.150] [da_laser] [info] DA: Publish sliding window data.
[2017-11-27 09:13:05.150] [da_laser] [info] DA: Number of new factors: 1Number of new values: 1
[2017-11-27 09:13:05.151] [da_laser] [info] DA: Publish 2D map data, new scan. number of values: 1
[2017-11-27 09:13:05.151] [da_laser] [info] DA: Robot A, index 0: Searching for loop closure
[2017-11-27 09:13:05.151] [da_laser] [info] DA: Robot A, index 0: Searching for loop closureDA: Robot A, index 0:  Searching for multirobot factors
[2017-11-27 09:13:05.153] [da_laser] [info] DA: receive serialized values.
[2017-11-27 09:13:05.153] [da_laser] [info] DA: Number of optimized values: 1
[2017-11-27 09:13:05.153] [da_laser] [info] --------------------------
Last poses of each robot:
--------------------------
[2017-11-27 09:13:05.153] [da_laser] [info] Robot id: A, index: 0
[2017-11-27 09:13:05.153] [da_laser] [info] Robot position: X = 19.9, Y = -0.1, Z = 0
[2017-11-27 09:13:16.610] [da_laser] [info] receive new 2D keyframe information...
[2017-11-27 09:13:16.610] [da_laser] [info] DA: Robot B, index 0: Matching in sliding window
[2017-11-27 09:13:16.610] [da_laser] [info] Add new robot, ID B
[2017-11-27 09:13:16.610] [da_laser] [info] DA: Robot B, keyframe0 initialized...
[2017-11-27 09:13:16.610] [da_laser] [info] DA: Publish sliding window data.
[2017-11-27 09:13:16.611] [da_laser] [info] DA: Number of new factors: 1Number of new values: 1
[2017-11-27 09:13:16.611] [da_laser] [info] DA: Publish 2D map data, new scan. number of values: 1
[2017-11-27 09:13:16.611] [da_laser] [info] DA: Robot B, index 0: Searching for loop closure
[2017-11-27 09:13:16.611] [da_laser] [info] DA: Robot B, index 0:  Searching for multirobot factors
[2017-11-27 09:13:16.613] [da_laser] [info] DA: receive serialized values.
[2017-11-27 09:13:16.613] [da_laser] [info] DA: Number of optimized values: 2
[2017-11-27 09:13:16.613] [da_laser] [info] --------------------------
Last poses of each robot:
--------------------------
```

Workshop Topics

- **End to end full demo – MRBSP with 2 UGV in gazebo simulation.** ☑
- **Installation – How to install ANPL software** ☑
- **ROS notations** ☑
- **High level code structure – passive and active** ☑
- **Launch files and software configuration** ☑
- **how to run the code** ☑
- **Code outputs** ☑
- **Matalb analysis tools**
- Matlab interface with the code
- Code handling policy
- Hands on exercises

Matlab Analysis Tools

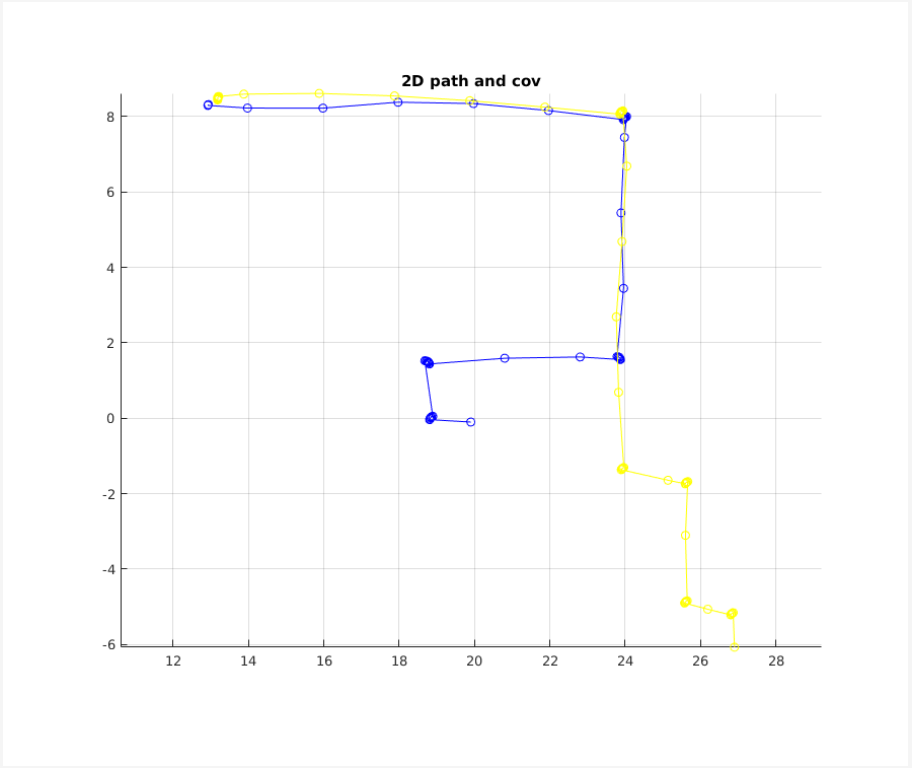
- During its run the produce txt file that can be read in matlab scripts.
- The matlab scripts are located at: …/mrbsp_ros/Matlab
- The main analysis scripts is: “loadResults.m”
- To view the results from a specific run change the scenario name and the results folder to your values.
- Chose the file with the latest index in the matlab folder at the results folder (without the “_values” or “_factors” at the end of the file name).

```
scenario = 'mr_centralized_laser';
rsults_folder = 'ANPL_Reasearcher_2017-11-27_04-17-45';
path_to_results = ['..' filesep 'mrbsp_scenarios' filesep 'scenarios' filesep scenario filesep...
    'results' filesep rsults_folder filesep 'matlab'];
filename = 'belief_B79';
```

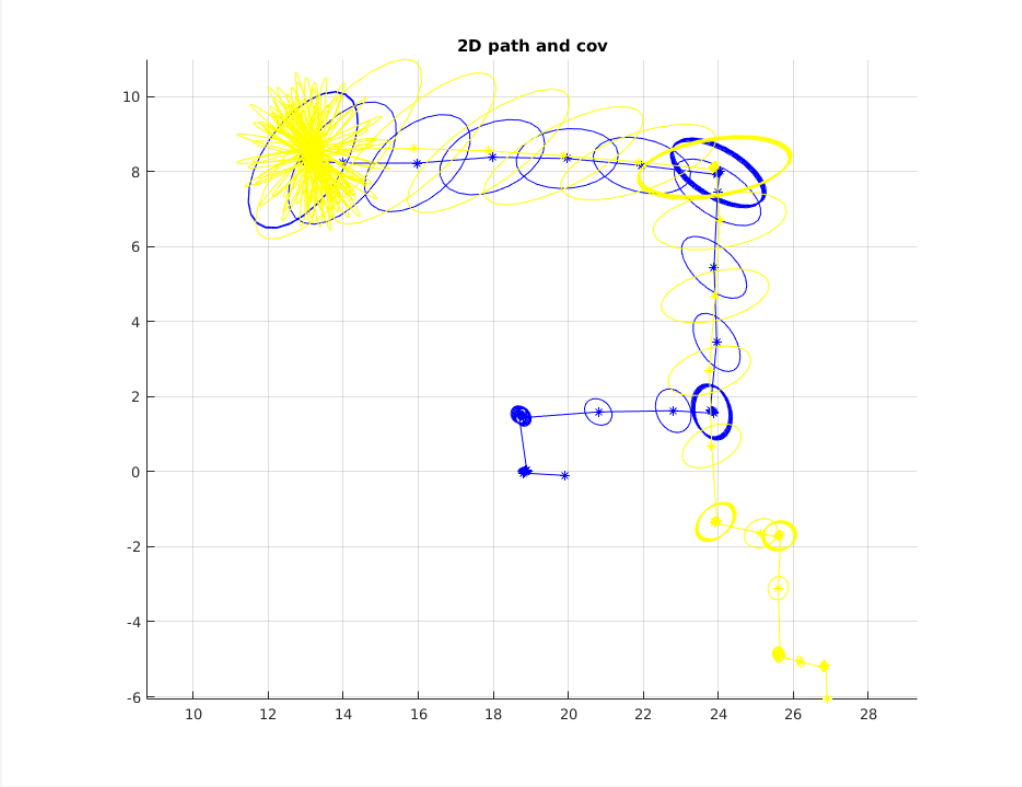
Matlab Analysis Tools

- To plots are available:

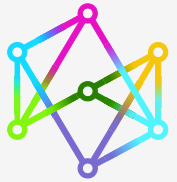
Factor Graph



Pose with covariance



Workshop Topics



- **End to end full demo – MRBSP with 2 UGV in gazebo simulation.** ☑
- **Installation – How to install ANPL software** ☑
- **ROS notations** ☑
- **High level code structure – passive and active** ☑
- **Launch files and software configuration** ☑
- **how to run the code** ☑
- **Code outputs** ☑
- **Matalb analysis tools** ☑
- **Matlab interface with the code**
- Code handling policy
- Hands on exercises

ANPL | Autonomous Navigation and Perception Lab

Matlab Interface With the Code

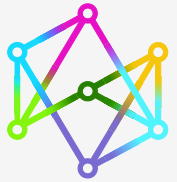
C++ side

```
std::stringstream to_send;
to_send << std::to_string(graph.size()) << m_delimiter;
for (int i = 0; i < graph.size(); i++) {
  gtsam::ISAM2 isam2(parameters);
  isam2.update(graph.at(i), initialEstimate.at(i));
  to_send << gtsam::serialize(isam2.getFactorsUnsafe()) << m_delimiter
        << gtsam::serialize(isam2.calculateBestEstimate()) << m_delimiter;
}
std_msgs::String msg;
msg.data = to_send.str();
m_matlab_pub.publish(msg);

ros::Rate rate(5.); //Hz
// busy wait
while (!m_is_received) {
  ros::spinOnce();
  rate.sleep();
}
if(m_is_correct) {
  ROS_INFO("Matlab answered %d", m_matlab_answer);
  return m_matlab_answer;
} else return 0;
```

Matlab side

```
...
while true
    disp ('planner waiting');
    str  = receive(matlab_sub);

    string_array  = strsplit(str.Data, delimiter);

    size              = str2num(string_array{1});
    disp (['planner received: ' string_array{1} ' candidate beliefs']);

    isam2_graph =gtsam.NonlinearFactorGraph.string_deserialize(string_array{2})
    isam2_values  = gtsam.Values.string_deserialize(string_array{3})

                % run your matlab code here...
                % and send back the result

                index_to_send = 0;
                msg = rosmessage(matlab_pub);
                msg.Data = index_to_send;
                matlab_pub.send(msg);

                disp (['Chosen Path: ' num2str(index_to_send)]);
end
```

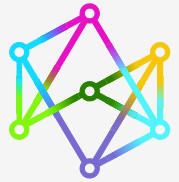
Workshop Topics



- **End to end full demo – MRBSP with 2 UGV in gazebo simulation.** ☑
- **Installation – How to install ANPL software** ☑
- **ROS notations** ☑
- **High level code structure – passive and active** ☑
- **Launch files and software configuration** ☑
- **how to run the code** ☑
- **Code outputs** ☑
- **Matalb analysis tools** ☑
- **Matlab interface with the code** ☑
- **Code handling policy**
- Hands on exercises

ANPL | Autonomous Navigation and Perception Lab

Code Handling Policy

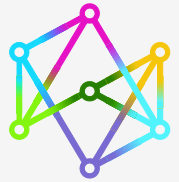


- The code is maintained as a git repository in bitbucket and is available to all ANPL members.
- The main stable version is at **master** branch.
- When starting a new implementation for a specific researcher, create a new branch from the master branch and start its name with the researcher name.
- When the code is ready send a pull request to merge the new branch with devel branch.
- If the integration with devel branch succeeds, the code will be tested more carefully and after it will pass all tests, devel branch will be merged with the master branch.

Workshop Topics

- **End to end full demo – MRBSP with 2 UGV in gazebo simulation.** ☑
- **Installation – How to install ANPL software** ☑
- **ROS notations** ☑
- **High level code structure – passive and active** ☑
- **Launch files and software configuration** ☑
- **how to run the code** ☑
- **Code outputs** ☑
- **Matalb analysis tools** ☑
- **Matlab interface with the code** ☑
- **Code handling policy** ☑
- **Hands on exercises**

Hands On Exercises



Exercise #1: Load robot in gazebo, move it manually with the keyboard controller and view its sensors in RVIZ.

Exercise #2: Create a passive slam scenario for one robot. Move the robot with the keyboard controller and examine the results in matlab.

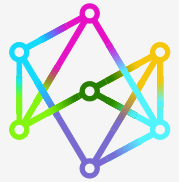
Exercise #3: Create an active scenario with two robots. Use the interactive action generator and choose manually the best trajectory in matlab.

Exercise #4: Create full active scenario. Use the interactive action generator and choose the best path according to cost function in C++.

Exercise #1 – Gazebo Simulation

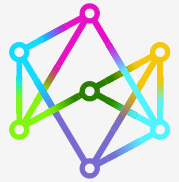
1. Open a new terminal.
2. Go to mrbsp_scenarios package and create a new scenario (name it “workshop”):
   - $ roscd mrbsp_scenarios/scenarios
   - $ mkdir workshop
3. Copy exercise1_gazebo.launch file from the workshop repository to the new scenario folder:
   - $ roscd anpl_software_workshop/launch/exercise1
   - $ cp exercise1_gazebo.launch ~/ANPL/code/mrbsp_ws/src/mrbsp_ros/mrbsp_scenarios/scenarios/workshop
4. Edit the launch file and change the robot initial pose to (x=20.0, y=0.0, z=0.0, Yaw=1.57, Roll=0.0, Pitch=0.0):
   - $ roscd mrbsp_scenarios/scenarios/workshop
   - $ gedit exercise1_gazebo.launch
   - Edit init_pose arguments to the requested values
5. Use the launch file to open gazebo simulator:
   - $ roslaunch mrbsp_scenarios exercise1_gazebo.launch

Exercise #1 – Gazebo Simulation



6. Open a new terminal
7. Open RVIZ to visualize the robot laser scans:
   - $ roslaunch anpl_software_workshop exercise1_rviz.launch
8. Use the keyboard controller (opened in a new terminal) to move the robot around and examine the laser scans in RVIZ:
   - Press ‘e’ to enable the motors
   - Move to robot with the keyboard arrows
   - The red line of the robot origin is directing to the robot forward direction
9. Close gazebo, rviz and the opened terminals (use ctrl C for the terminals)

Exercise #2 – Passive SLAM

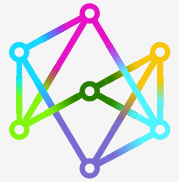


1. Open a new terminal.
2. Copy exercise2 files to the workshop scenario:
   - $ roscd anpl_software_workshop/launch/exercise2
   - $ cp exercise2_centralized.launch exercise2_robot_A.launch ~/ANPL/code/mrbsp_ws/src/mrbsp_ros/mrbsp_scenarios/scenarios/workshop
3. Copy config file folder to the workshop scenario folder:
   - $ roscd anpl_software_workshop
   - $ cp -R config_files ~/ANPL/code/mrbsp_ws/src/mrbsp_ros/mrbsp_scenarios/scenarios/workshop
4. Change researcher name and scenario name in exercise2_centralized.launch:
   - $ roscd mrbsp_scenarios/scenarios/workshop
   - $ gedit exercise2_centralized.launch
   - Change researcher_name argument to your name & current_scenario argument to “workshop”

Exercise #2 – Passive SLAM

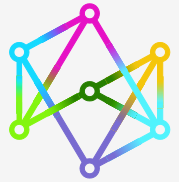
5. Change scenario name in exercise2_robot_A.launch:
   - $ roscd mrbsp_scenarios/scenarios/workshop
   - $ gedit exercise2_robot_A.launch
   - Change current_scenario argument to “workshop”
6. Edit init_pose parameters for robot A according to the initial pose you set in exercise1_gazebo.launch file:
   - $ roscd mrbsp_scenarios/scenarios/workshop/config_files
   - $ gedit gazebo_robot_A.yaml
   - Edit init_pose values
7. Launch gazebo simulator with the file you created in exercise 1:
   - $ roslaunch mrbsp_scenarios exercise1_gazebo.launch
8. Launch robot A and centralized machine nodes:
   - In new terminal: $ roslaunch mrbsp_scenarios exercise2_centralized.launch
   - In new terminal: $ roslaunch mrbsp_scenarios exercise2_robot_A.launch

Exercise #2 – Passive SLAM



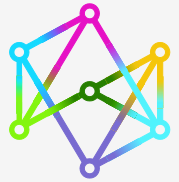
9. Open RVIZ to visualize the estimated trajectory of the robot and the map:
   - $ roslaunch anpl_software_workshop exercise2_rviz.launch
10. Move the robot with the robot controller. Get outside of the big room and enter it form a different entrance in order to detect a loop closure.
11. Close gazebo, RVIZ, and the opened terminal.
12. Go to results folder and examine the code outputs.
    - Go to the current run folder (researcher name + date + time) and type $ octovis octomap.ot
    - Go to log and matlab folder and examine the results.
13. Open matlab and go to ~ANPL/code/mrbsp_ws/srs/mrbsp_ros_matalb.
14. Open loadResutls.m scripts.
15. Change scenario to “workshop”, results folder to the current run and filename to the file with the highest index (without the “_values” at the end).
16. Run the script.

Exercise #3 – Active SLAM with Matlab Planner



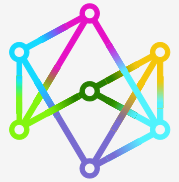
1. Open a new terminal.
2. Add one more robot to gazebo by editing exercise1_gazebo.launch from exercise 1:
   - $ roscd mrbsp_scenarios/scenarios/workshop
   - $ gedit exercise1_gazebo.launch
   - Copy line 3 and paste it at line 4. Change arg name to robot2_name and the defaults value to pioneer2.
   - Copy lines 11-17 (starts with <group and ends with </groupe>) and paste them at line 18 (above the </launch> tag).
   - On the lines you just copy, change arg robot1_name to robot2_name.
   - For robot 2, set init_pose argument to (x=26.9, y=-6.07, z=0.0, Yaw=1.57, Roll=0.0, Pitch=0.0)
3. Launch exercise1_gazebo.launch and check that 2 robot are now existed.
4. Update gazebo_robot_B.yaml file with the initial pose of robot B.
5. Copy exercise3 files from the workshop repository:
   - $ roscd anpl_software_workshop/launch/exercise3
   - $ cp exercise3_centralized.launch exercise3_robot_A.launch exercise3_robot_B.launch ~/ANPL/code/mrbsp_ws/src/mrbsp_ros/mrbsp_scenarios/scenarios/workshop
   - At all files, change researcher_name argument to your name & current_scenario argument to “workshop”.

Exercise #3 – Active SLAM with Matlab Planner



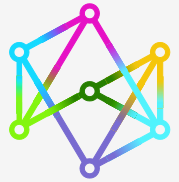
6. Open a new terminal and launch exercise3_centralized.launch:
   - & roslaunch mrbsp_scenarios exercise3_centralized.launch
7. Open a new terminal and launch exercise3_robot_A.launch:
   - $ roslaunch mrbsp_scenarios exercise3_robot_A.launch
8. Open a new terminal and launch exercise3_robot_B.launch:
   - $ roslaunch mrbsp_scenarios exercise3_robot_B.launch
9. Open planner node in Matlab.
   - In a new terminal: $ Matlab
   - Go to ANPL/code/mrbsp_ws/src/mrbsp_ros/planner_node/matlab
   - Open planner_a.m script and place a breakpoint at line 29 (“index_to_send ….”)
   - Run the script.
10. On RVIZ, use the interactive action generator and create a candidate actions for each robot.
11. On matlab, examine the belief you receive from the cpp code.

Exercise #4 – Active SLAM with C++ Planner

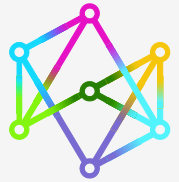


1. Change the planner to be fully C++ implemented
   - $ roscd mrbsp_scenarios/scenarios/workshop
   - $ gedit exercise3_centralized.launch
   - At line 41 change param name “planner_type” to cpp
2. Launch the scenarios:
   - In a new terminal $ roslaunch mrbsp_scenarios exercise1_gazebo.launch
   - In a new terminal $ roslaunch mrbsp_scenarios exercise3_centralized.launch
   - In a new terminal $ roslaunch mrbsp_scenarios exercise3_robot_A.launch
   - In a new terminal $ roslaunch mrbsp_scenarios exercise3_robot_B.launch
3. Use the interactive action generator and create a candidate actions for each robot
4. In RVIZ, before double clicking on the screen to finish the actions generation, on the left panel, uncheck “Loaded Map” and “PoseWithCovariance” boxes and check all the unchecked boxes.
5. Double click on the screen an watch the robots accomplish the chosen actions.

Exercise #4 – Active SLAM with C++ Planner



6. Explore the results of the code in the results folder.
7. Use the Matlab analysis tools to explore the results in Matlab.



The End!!!