ROSPlan:

AI Planning and Robotics

Gerard Canal, Michael Cashmore, Daniele Magazzeni

AI Planning...

- Path Planning ✗
- Motion Planning ✗
- Task Planning ✓
  - Action Planning
  - AI Planning
  - Symbolic Planning
  - (...)

- The generation of a set of **actions** that bring the system from an **initial state** to a **goal state** (possibly optimizing a **metric function**)
  - A model of the world/environment is needed
- Actions modify the current state
  - If their **preconditions** are met.
  - The **effects** are applied to the state.
  - ...But they can incur in costs or rewards.

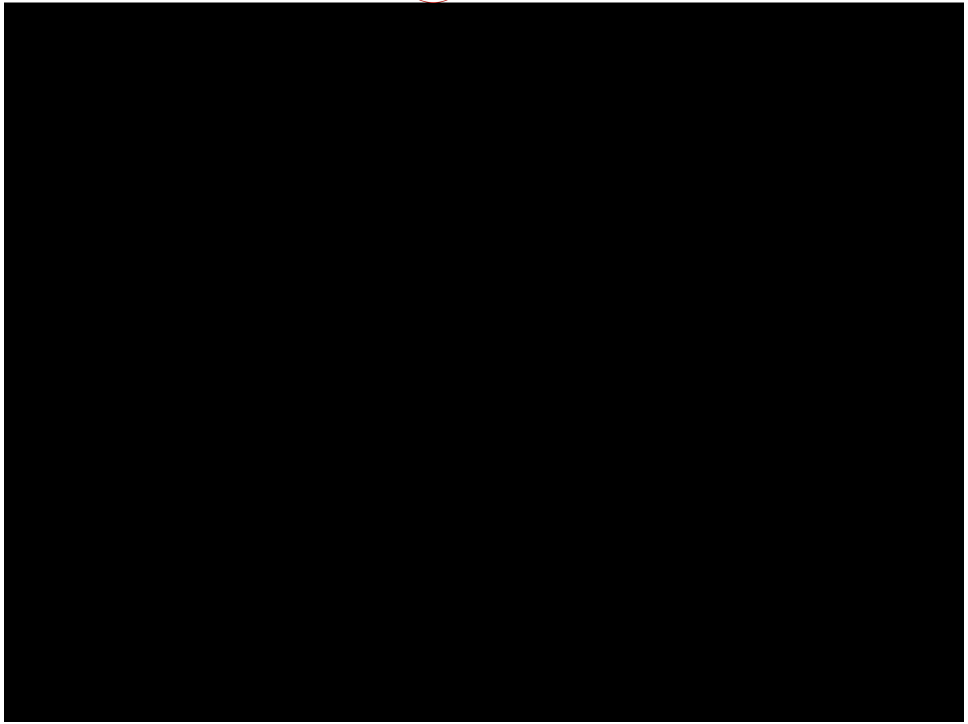
What does Task Planning need?

- Planners have two input files:
  - Domain file → Stores the model, dynamics and rules of the world
    - Predicates, Constraints, Actions (parameters, preconditions, effects)
  - Problem file → Defines the task to be solved using the domain.
    - Objects of the world, Initial State, Goal state
- Written in some description language: PDDL

Motivation: let’s fit a shoe!

Institut de Robòtica
i Informàtica Industrial

- “Simple task”
- Low number of actions
- Very easy to describe in PDDL
- Very difficult to define every possible configuration in a state machine



x4

Extracted from:
G. Canal, E. Pignat, G. Alenyà, S. Calinon and C. Torras, "Joining High-Level Symbolic Planning with Low-Level Motion Primitives in Adaptive HRI: Application to Dressing Assistance," *IEEE International Conference on Robotics and Automation (ICRA)*, 2018.

Shoe fitting example: PDDL domain

```
(:predicates (shoeInHand ?s - shoe)
             (footMoving ?f - foot)
             (reachableFoot ?f - foot)
             (footInCorrectPose ?f - foot)
             (correctShoe ?s - shoe ?f - foot)
             (footInShoe ?s - shoe ?f - foot)
)

(:action approachFoot
    :parameters (?f - foot ?s - shoe)
    :precondition (and (not (reachableFoot ?f))
                       (shoeInHand ?s)
                       (not (footMoving ?f)) )
    :effect (reachableFoot ?f)
)
```

```
(:action insertFootInShoe
    :parameters (?s - shoe ?f - foot)
    :precondition (and (reachableFoot ?f) (shoeInHand ?s)
                       (correctShoe ?s ?f) (not (footMoving ?f))
                       (not (footInShoe ?s ?f)))
    :effect (and (footInShoe ?s ?f) )
  )

  (:action releaseShoe
    :parameters (?s - shoe ?f - foot)
    :precondition (and (shoeInHand ?s) (shoeInFoot ?s ?f)
                       (not (footMoving ?f)) (informedUser) )
    :effect (not (shoeInHand ?s))
  )
```

Example: PDDL problem

```
(define (problem roscon_shoefit)
(:domain shoe_fitting)
(:objects
   rightf leftf - foot
   rights lefts - shoe
)
(:init
   (not (shoeInHand rights))
   (not (shoeInHand lefts))
   (footInCorrectPose rightf)
   (footInCorrectPose leftf)
)
(:goal (and
   (not (shoeInHand rights))
   (not (shoeInHand lefts))
   (footInShoe rights rightf)
   (footInShoe lefts leftf)
))
```

Plan

```
0.000:   (approachShoe lefts)              [10.000]
10.001:  (graspShoe lefts)                 [5.000]
15.002:  (goto_startpose)                  [15.000]
30.003:  (approachFoot leftf lefts)        [20.000]
50.004:  (insertFootInShoe lefts leftf)    [25.000]
75.005:  (releaseShoe lefts leftf)         [5.000]
80.000:  (graspShoe lefts)                 [5.000]
105.007: (insertFootInShoe lefts leftf)    [25.000]
130.008: (releaseShoe lefts leftf)         [5.000]
```

**More info on AI Planning can be found at https://planning.wiki*

Why Planning with Robots?

- Planning provides:
  - Robustness
  - Adaptivity
  - Simplification
- Where reactive behaviors may not suffice:
  - HRI
  - Search & rescue
  - Dangerous missions
  - Time-constrained tasks
  - Task reasoning

- Many challenges arise:
  - Failures
  - Handle risk and uncertainties
  - Unexpected or unplanned events
  - Integration with control (task + motion)
  - **Humans around!**

Task Planning with Robots (in general)

- Difficult to integrate:
  - Usually without an API available
  - Planners are usually research oriented/non-commercial software
  - Same input language, with subtleties, but output may differ
- To use a planner requires a lot of work and some coupling with the planner.
- Try another planner… Repeat the most of the work!

Tedious!

Process for task Planning in Robotics

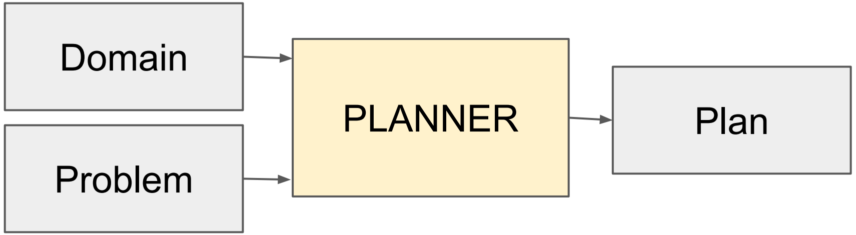
1. Store the **state** representation
2. Produce the **Domain** and **Problem**
3. Generate the **plan**
4. **Execute** the plan
    - Executing the actions
    - Making observations

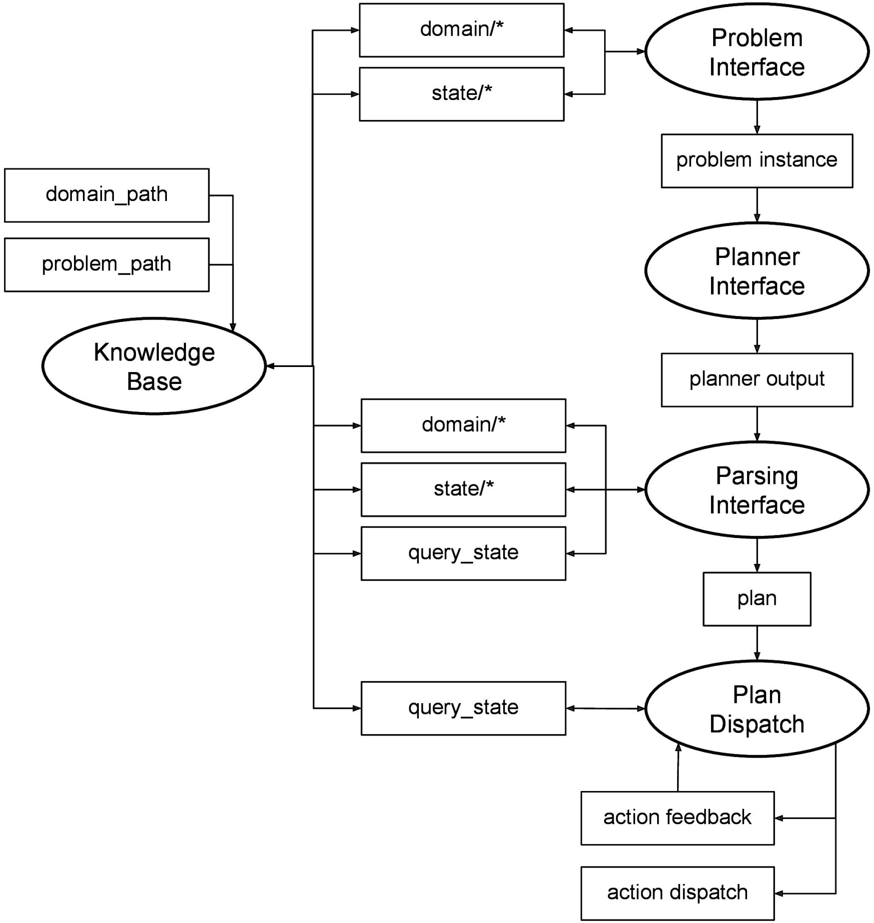
Meet ROSPlan *(I wish I had done before…)*

- The ROSPlan framework provides a collection of **tools** for **AI Planning** in a **ROS** system.
- ROSPlan has a variety of nodes which encapsulate planning, problem generation, and plan execution.
  - Plus many state-of-the-art planners available to use!
  - Extensible and modular → Easy to add new planners and new architectures.
  - And it's Open Source!
- In summary, makes task planning for robots simpler...

ROSPlan unleashed

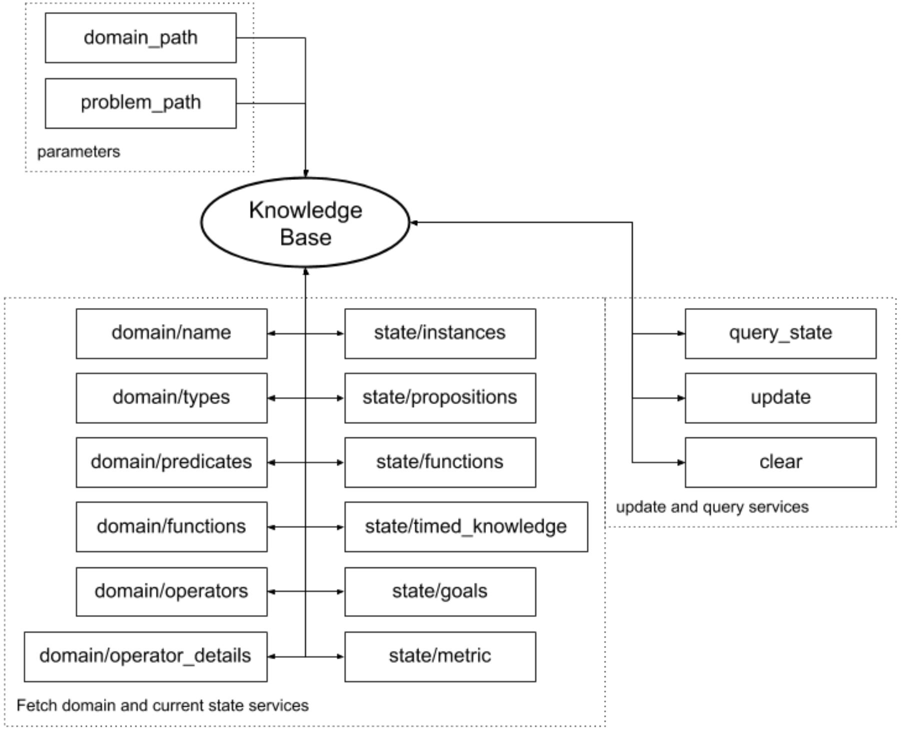
- Not a default system, but a set of tools to suit the developer’s needs.





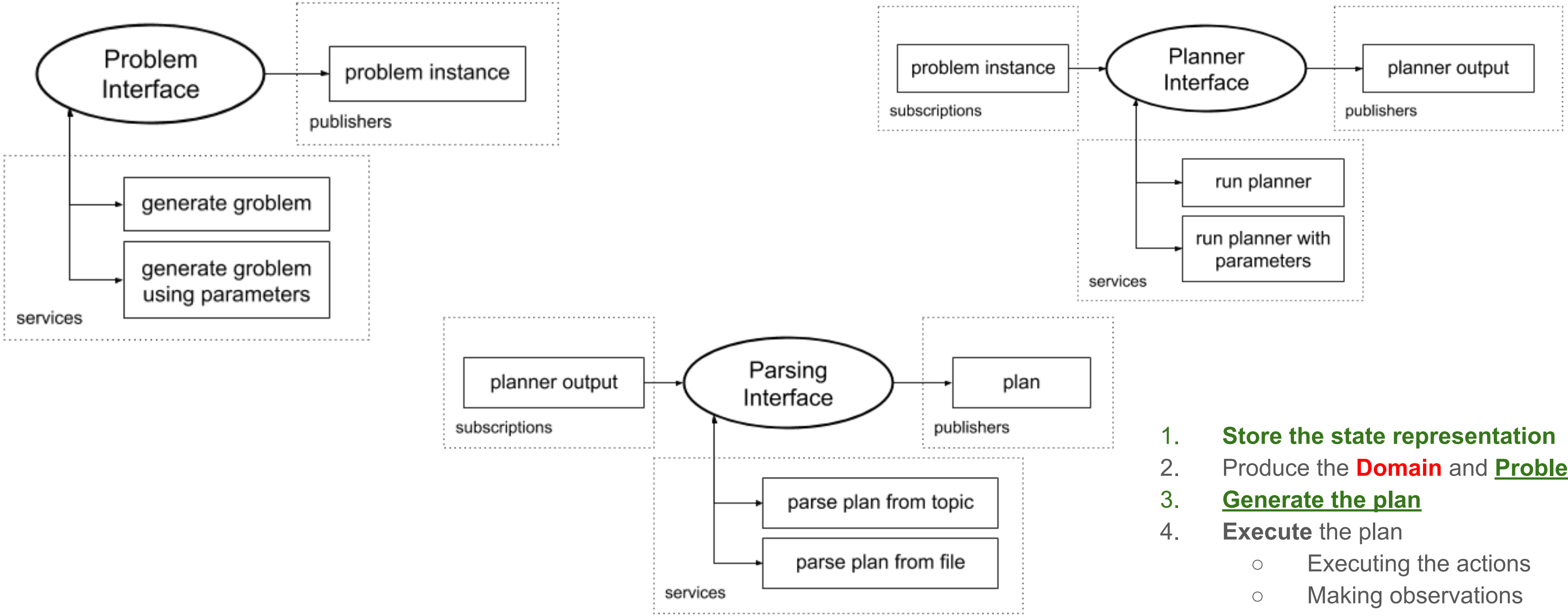
The Knowledge Base

- Stores the domain model and the current state.
- Provides services to fetch domain details, and to query and modify the current state.
- Is updated based on action outcomes, commonly obtained from sensor observation and data processing.



1. **Store the state representation**
2. Produce the **Domain** and **Problem**
3. Generate the **plan**
4. **Execute** the plan
   - Executing the actions
   - Making observations

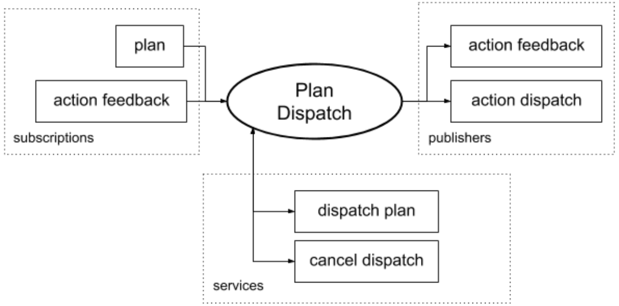
Planning interfaces



1. **Store the state representation**
2. Produce the **Domain** and **Problem**
3. **Generate the plan**
4. **Execute** the plan
    - Executing the actions
    - Making observations

Plan dispatch: taking action

- The dispatcher receives a plan and is responsible of executing its actions in the right times.
- Each action is dispatched by sending its details to a topic
  - Action executor subscribes the topic and provides feedback on the result.
- Different dispatchers available:
  - Sequential, Esterel (graph-based, concurrent actions), Online (*new*)



1. **Store the state representation**
2. Produce the **Domain** and **Problem**
3. **Generate the plan**
4. **Execute the plan**
   - Executing the actions
   - Making observations

Action Interface: The connection with the real world

- We now have seen a complete deliberative system providing plans and launching actions
  - But actions need to be executed, and their outcomes checked!
  - This is robot/action dependent.
- To simplify the task, ROSPlan provides an Action Interface
  - Abstract C++ class to be extended that subscribes to the dispatcher and executes a callback
  - It also updates the Knowledge Base with the fixed outcomes of the task
  - In the callback, the user defines the behavior of the specific action (connection to actionlib actions, calling services… doing the actual work).

1. **Store the state representation**
2. Produce the **Domain** and **Problem**
3. **Generate the plan**
4. **Execute the plan**
   - ***Executing the actions***
   - Making observations

Sensing Interface: seeing the real world

- Assuming each action succeeds and its effects are applied, though useful, is
  - Unrealistic
  - Completely false in robotics
- Updating the Knowledge Base with sensor data is a repetitive process:

1. Get sensor data / state of the world
2. Compute predicate values
3. Update KB

1. **Store the state representation**
2. Produce the **Domain** and **Problem**
3. **Generate the plan**
4. **Execute the plan**
   - ***Executing the actions***
   - ***Making observations***

Automatic KB updating

- The new Sensing Interface provides hassle-free update of the KB
  - With only *5* lines of YAML!
- It automatically subscribes to topics and calls services at a fixed rate.
- Example:

```yaml
topics:
  robot_at:
    params:
        - kenny
        - docking_station
    topic: /amcl_pose
    msg_type: geometry_msgs/PoseWithCovarianceStamped
    operation: "(msg.pose.pose.position.x == 0) and (msg.pose.pose.position.y == 0)"
```

- More complex options can also be considered in an external python file.

Other extensions

- In addition, ROSPlan currently supports:
  - Probabilistic planning (planning with uncertainty and non-deterministic effects)
  - The PPDDL and RDDL languages
  - Temporal plan execution with deadlines
  - Robustness through Robust Envelopes

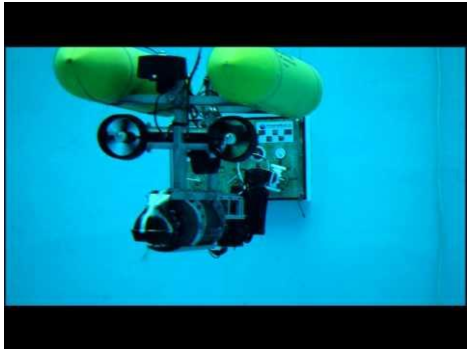
What’s next?

- Improvements on support of planning constructs
- Support of conditional effects
- Integration of Explainable AI Planning to have self-explaining robots
- Goal reasoning
- ...And many more!

Success cases

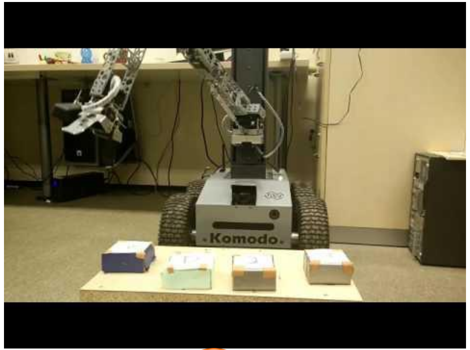
TOYOTA HSR
King's College London

SQUIRREL
Clearing Clutter Bit by Bit

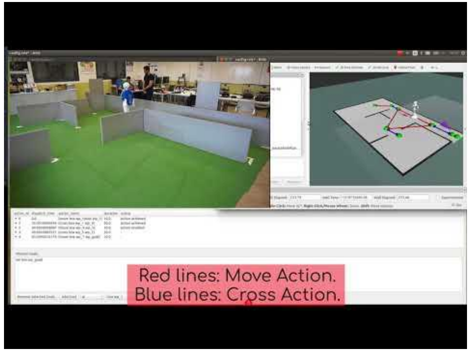


Seventh Framework Programme
Pandora
Persistently Autonomous Robots

King's College London
Institut de Robòtica i Informàtica Industrial



Ben-Gurion University of the Negev



intelligent robotics

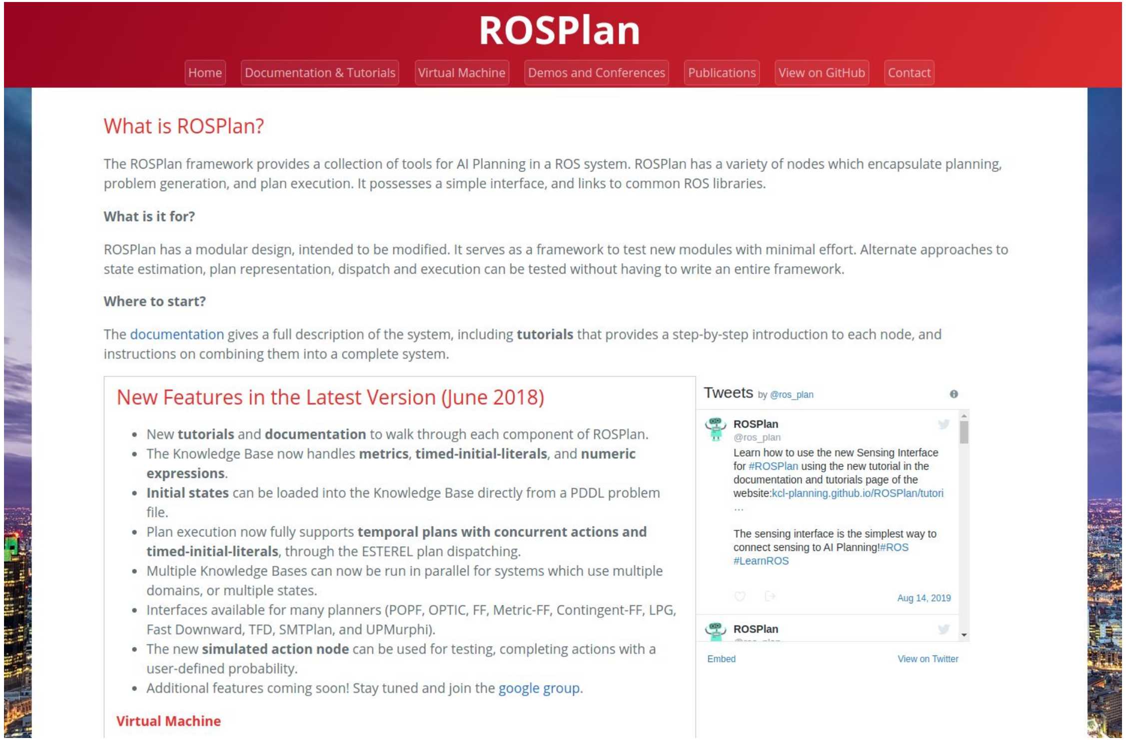
Institut de Robòtica i Informàtica Industrial

TECHNION
Israel Institute of Technology

More info at kcl-planning.github.io/ROSPlan

Available:

- Documentation
- Tutorials
- Demos



ROSPlan

Home | Documentation & Tutorials | Virtual Machine | Demos and Conferences | Publications | View on GitHub | Contact

What is ROSPlan?

The ROSPlan framework provides a collection of tools for AI Planning in a ROS system. ROSPlan has a variety of nodes which encapsulate planning, problem generation, and plan execution. It possesses a simple interface, and links to common ROS libraries.

What is it for?

ROSPlan has a modular design, intended to be modified. It serves as a framework to test new modules with minimal effort. Alternate approaches to state estimation, plan representation, dispatch and execution can be tested without having to write an entire framework.

Where to start?

The documentation gives a full description of the system, including **tutorials** that provides a step-by-step introduction to each node, and instructions on combining them into a complete system.

New Features in the Latest Version (June 2018)

- New **tutorials** and **documentation** to walk through each component of ROSPlan.
- The Knowledge Base now handles **metrics**, **timed-initial-literals**, and **numeric expressions**.
- **Initial states** can be loaded into the Knowledge Base directly from a PDDL problem file.
- Plan execution now fully supports **temporal plans with concurrent actions and timed-initial-literals**, through the ESTEREL plan dispatching.
- Multiple Knowledge Bases can now be run in parallel for systems which use multiple domains, or multiple states.
- Interfaces available for many planners (POPF, OPTIC, FF, Metric-FF, Contingent-FF, LPG, Fast Downward, TFD, SMTPlan, and UPMurphi).
- The new **simulated action node** can be used for testing, completing actions with a user-defined probability.
- Additional features coming soon! Stay tuned and join the google group.

Virtual Machine

Tweets by @ros_plan

ROSPlan
@ros_plan
Learn how to use the new Sensing Interface for #ROSPlan using the new tutorial in the documentation and tutorials page of the website:kcl-planning.github.io/ROSPlan/tutori …

The sensing interface is the simplest way to connect sensing to AI Planning!#ROS #LearnROS

Aug 14, 2019

ROSPlan

Embed
View on Twitter

Thank you!

Questions?

Find us at:

github.com/KCL-Planning/ROSPlan

kcl-planning.github.io/ROSPlan | human-ai-teaming.com

@ros_plan

Any contribution is welcomed!