UNIVERSITÄT OSNABRÜCK

German Research Center for Artificial Intelligence

ROSPlan

Wissensbasierte Systeme - Planbasierte Robotersteuerung, April. 26, 2021

Oscar Lima Carrion

German Research Center for Artificial Intelligence (DFKI)

Plan-Based Robot Control research department
Osnabrück, Germany

Course structure

1. ROS
2. ROSPlan
3. Q+A

Course structure

1. **ROS**
2. ROSPlan
3. Q+A

What is ROS?

- ROS = Robot Operating System
- *Not really an operating system, but is easier to explain in the business world this way…*
- ROS = (A) plumbing + (B) tools + (C) capabilities + (D) ecosystem (Brian Gerkey)
- (A) provides publish-subscribe messaging infrastructure designed to support the quick and easy construction of distributed computing systems.

ROS

Open Source Robotics Foundation

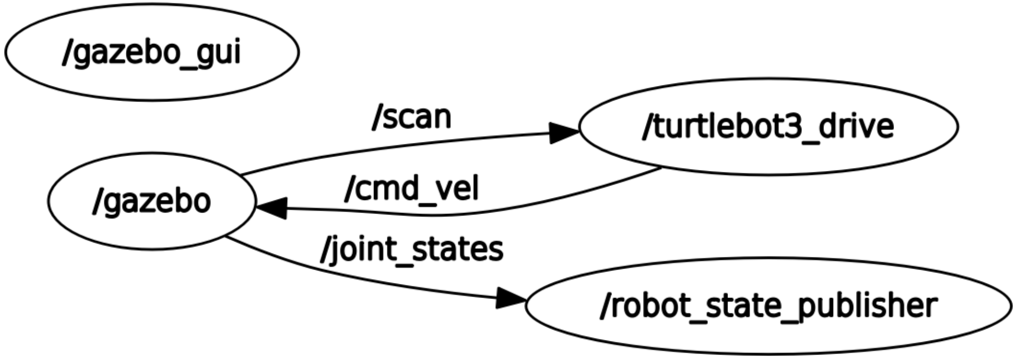
- Source: https://answers.ros.org/question/12230/what-is-ros-exactly-middleware-framework-operating-system/

What is ROS? (A) plumbing + (B) tools + (C) capability + (D) ecosystem

- (B) ROS provides an extensive set of tools for configuring, starting, introspecting, debugging, visualizing, logging, testing, and stopping distributed computing systems.
- (C) ROS provides a broad collection of libraries that implement useful robot functionality, with a focus on mobility, manipulation, and perception.
- (D) ROS is supported and improved by a large community, with a strong focus on integration and documentation. ros.org is a one-stop-shop for finding and learning about the thousands of ROS packages that are available from developers around the world.

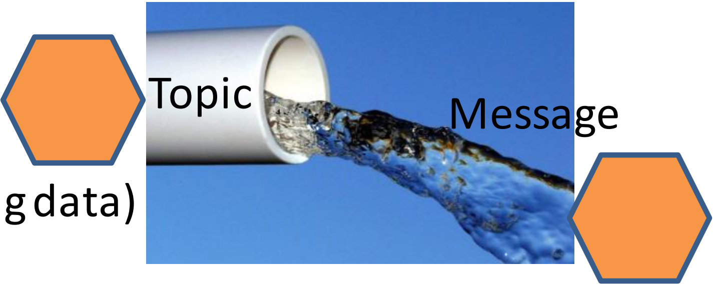
Basic concept 1 : Node

- **Node : a process that performs computation**
- Nodes are combined together into a graph and communicate with one another using topics, services, actionlib and the Parameter Server
- Nodes may reside in different machines transparently



Basic concept 2: Topic

- Topics: Named buses (pipes) over which nodes exchange messages.
- Producer – Consumer design pattern
- In general, nodes are not aware of who they are communicating with.
- nodes that are interested in data ***subscribe*** to the relevant topic.
- nodes that generate data ***publish*** to the relevant topic.



Node
(publishing data)

Topic

Message

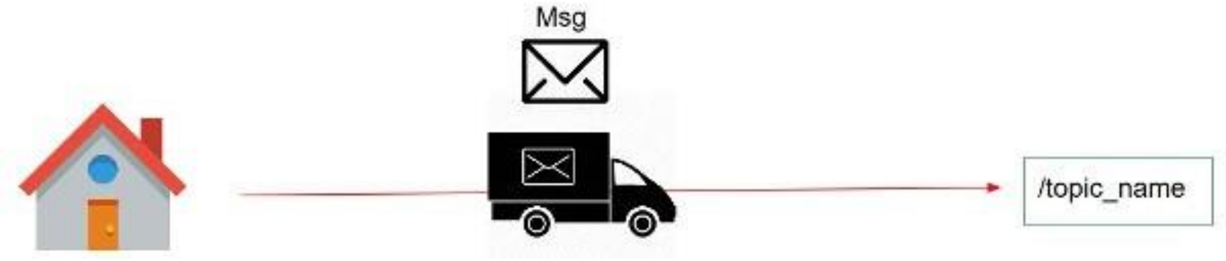
Node
(subscribing to data)

Properties of topics

- There can be multiple publishers and subscribers to a topic.
- Intended for unidirectional, streaming communication.
- Nodes communicate with each other by publishing **messages** to topics.
- An active topic can only have a single **message** type at a time.

Basic concept 3: Messages

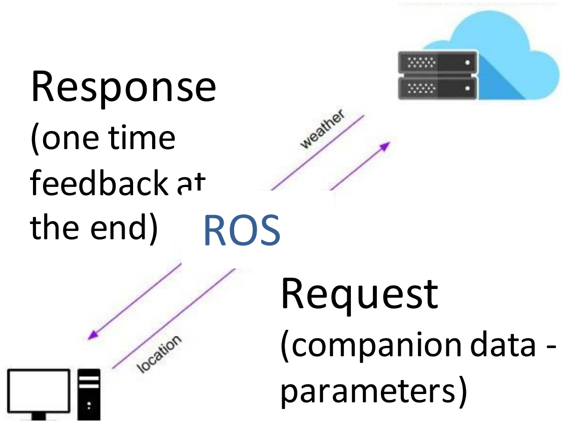
- Earlier we saw that nodes can make connections with each other via topics.
- Informally the message can be seen as the envelope you send via post.
- Messages state what kind of information your nodes need to produce in order to communicate together.



Basic concept 4: Service

- Service: is a client/server communication request system.
- There is no feedback while the operation is being performed but only one time at the end.

server



Performs operation

Response (one time feedback at the end)

ROS

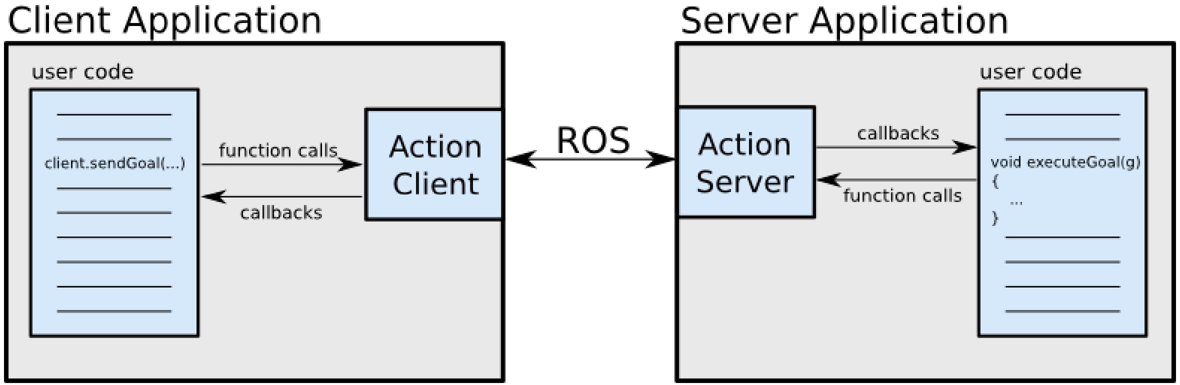
Request (companion data - parameters)

client

Typically is a blocking operation, but you can specify a timeout...

Basic concept 5: actionlib

- Similar to services but additionally you can:
  - Get periodic feedback about the progress of the request
  - Temporarily interrupt a task being carried out
  - Cancel the request



Development in ROS

- Mainly C++ and Python are supported.
- Experimental support for multiple other languages:
  - e.g. Java, lisp, nodejs, lua, ruby, R, Go, etc.
- Code is organized in ROS **packages** that live inside "catkin" workspaces (development folder).
- A catkin workspace is a folder where you modify, build, and install ROS **packages**.

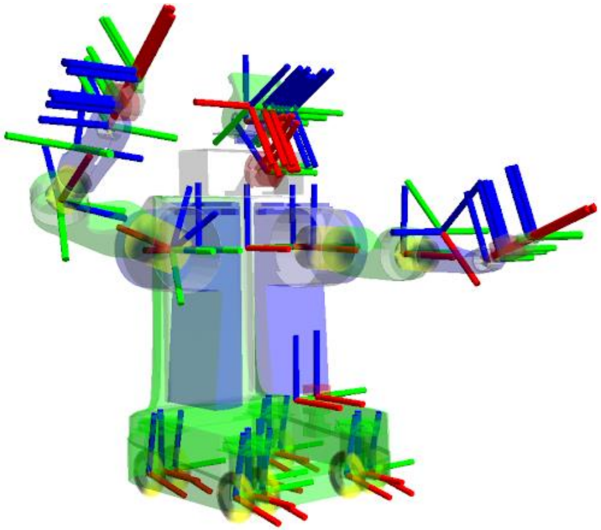
© Owen Scrimshaw

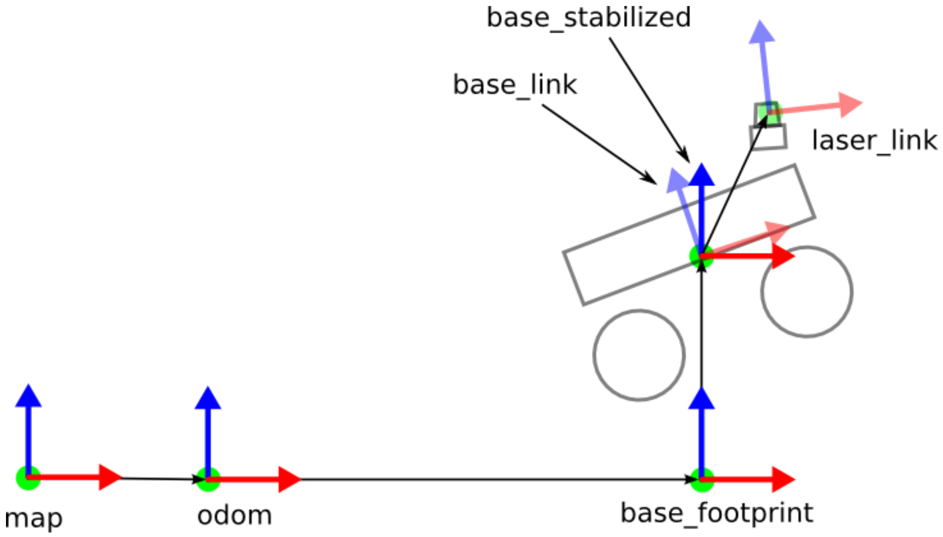
ROS bash

- Offers a set of shell ROS commands
- rosbash enables tab completion on: roslaunch, rosparam, rosnode, rostopic,rosservice, rosmsg, rossrv, rosbag.
- Most popular include:
- roscd pkgname (cd to pkgname easily)
- rosed pkgname filename (quickly edit a file)
- roscat pkgname filename (quickly visualize a file in terminal)
- rosrun pkgname executable (run executable from anywhere without having to give its full path)

Some important libraries 1/4

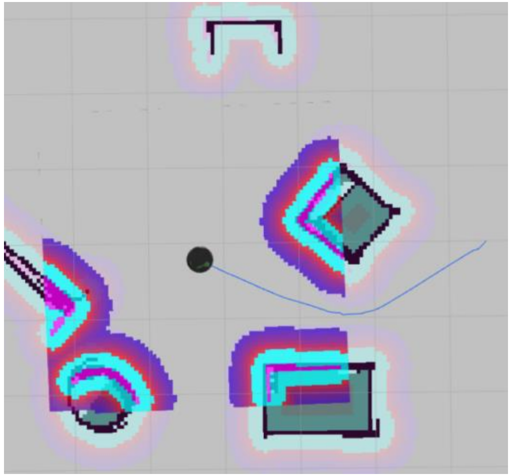
- **tf** : Library to keep track of multiple coordinate frames over time.
- Users can transform points, vectors, etc between any two coordinate frames at any desired point in time.

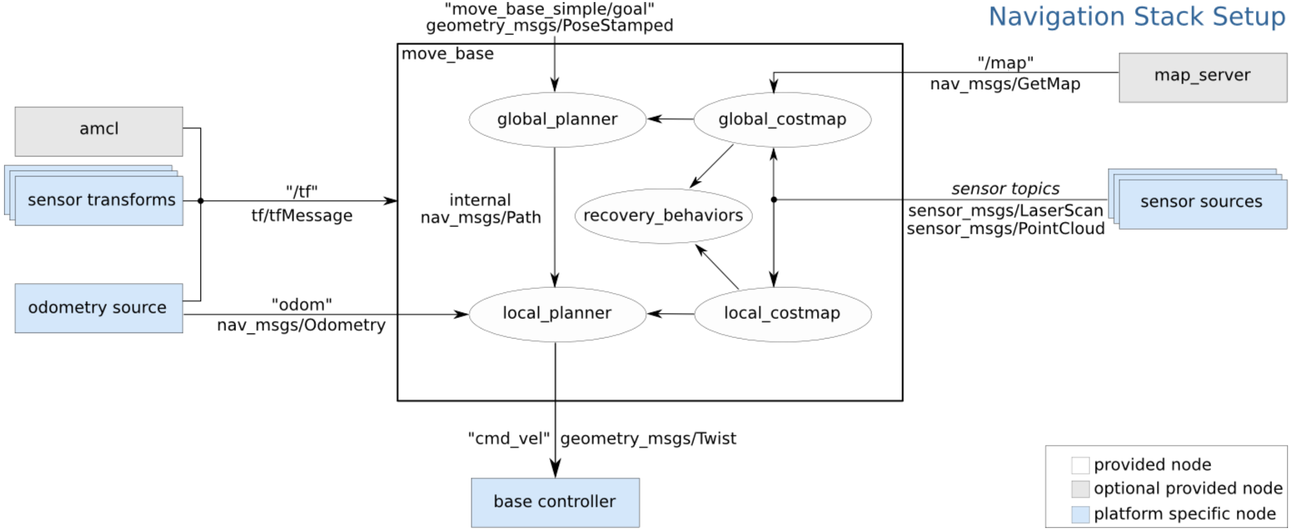




Some important libraries 2/4

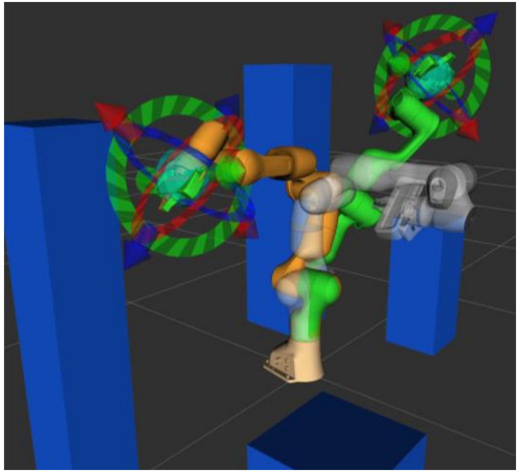
- **move_base**

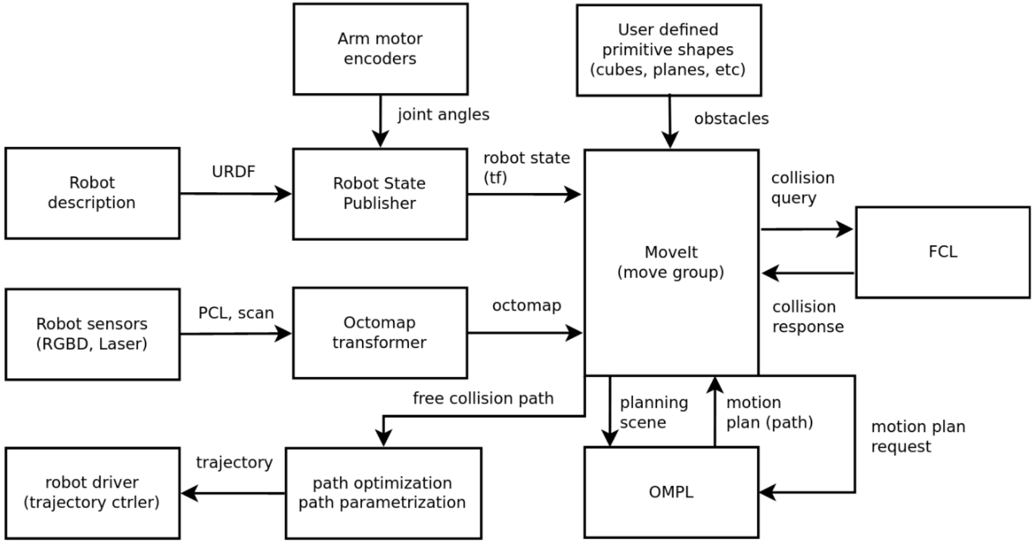




Some important libraries 3/4

MoveIt!

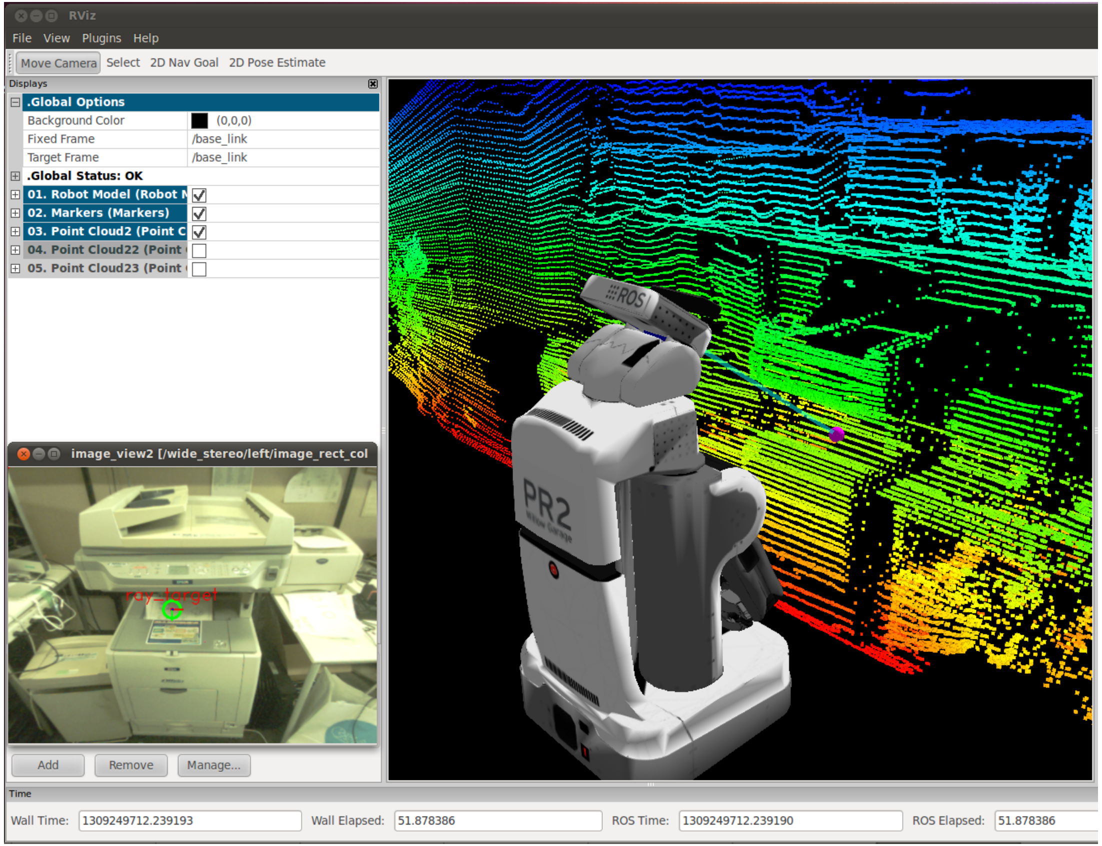




Some important libraries 4/4

- **Rviz**

3D visualisation tool for ROS



.Global Options

Background Color	(0,0,0)
Fixed Frame	/base_link
Target Frame	/base_link

.Global Status: OK

- 01. Robot Model (Robot M ☑
- 02. Markers (Markers) ☑
- 03. Point Cloud2 (Point C ☑
- 04. Point Cloud22 (Point ☐
- 05. Point Cloud23 (Point ☐

image_view2 [/wide_stereo/left/image_rect_col

ray_target

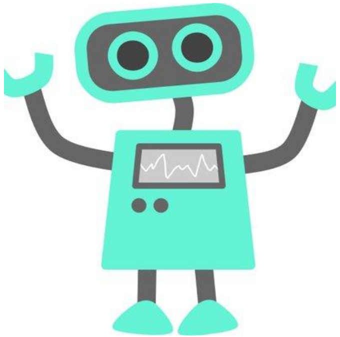
Wall Time: 1309249712.239193 Wall Elapsed: 51.878386 ROS Time: 1309249712.239190 ROS Elapsed: 51.878386

Course structure

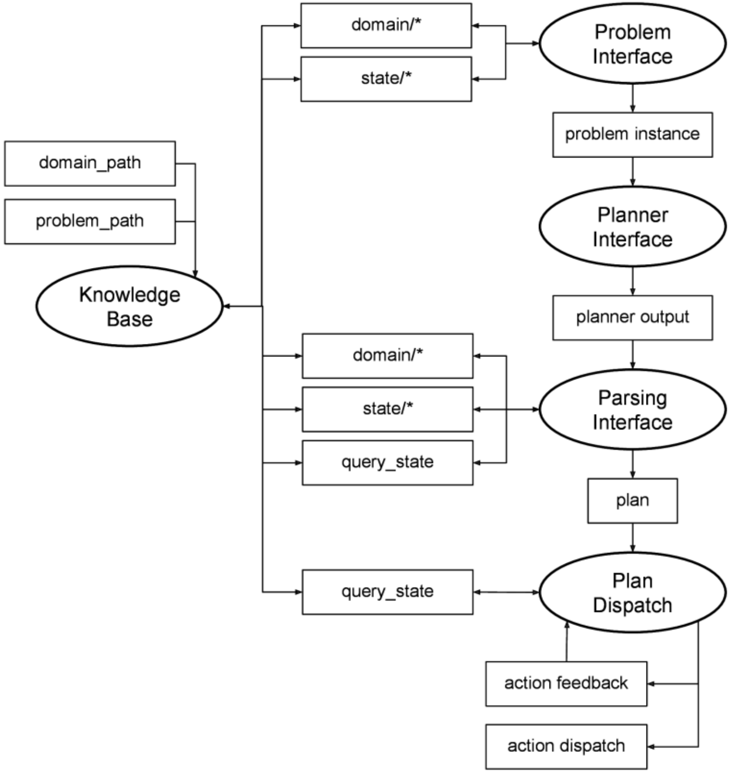
1. ROS
2. **ROSPlan**
3. Q+A

ROSPlan

- *"The ROSPlan framework provides a generic method for task planning in a ROS system".*
- First step towards integration of AI planning and robotics.
- Uses different technologies to provide with high level robot control.
- Main support: PDDL 2.1
- Experimental support: PPDDL, RDDL, HDDL (**HTN** new!)
- Plan execution and monitoring via:
  - Simple plan dispatch or Esterel plan dispatch



Architecture

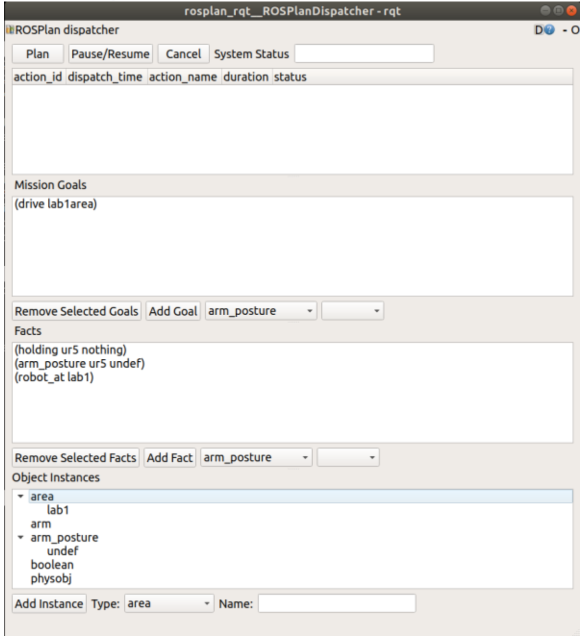


Components

- Knowledge Base (KB) : stores the (symbolic) planning model (domain and state); Communication via services
- Problem interface : query state from KB and create a problem instance
- Planner interface : wrapper around the AI planner, write its output to a topic
- Parsing interface : Convert planner output into a representation suitable for execution
- Plan dispatch : Execution and monitoring layer

Debugging tools

- KB rqt gui : KB visualisation/manipulation
- Esterel rqt gui : Plan visualisation and realtime execution progress



Mission Goals

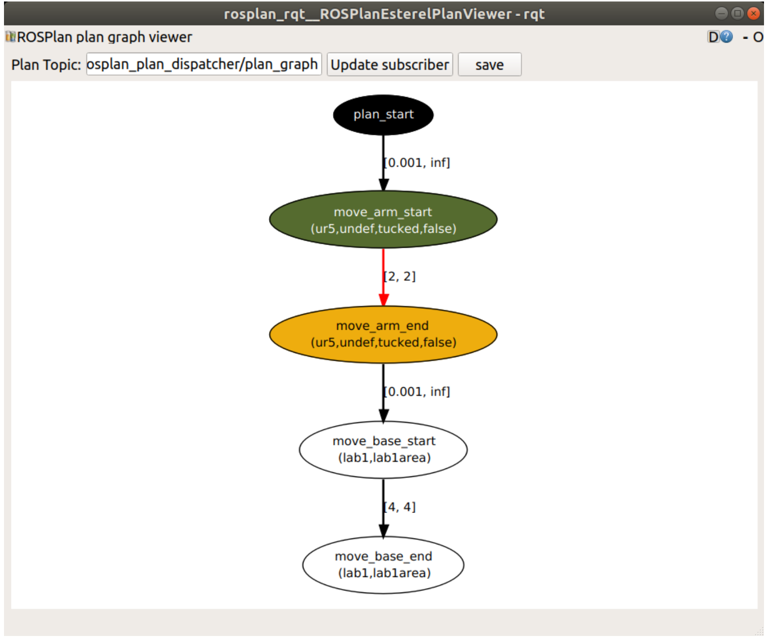
(drive lab1area)

Facts

(holding ur5 nothing)
(arm_posture ur5 undef)
(robot_at lab1)

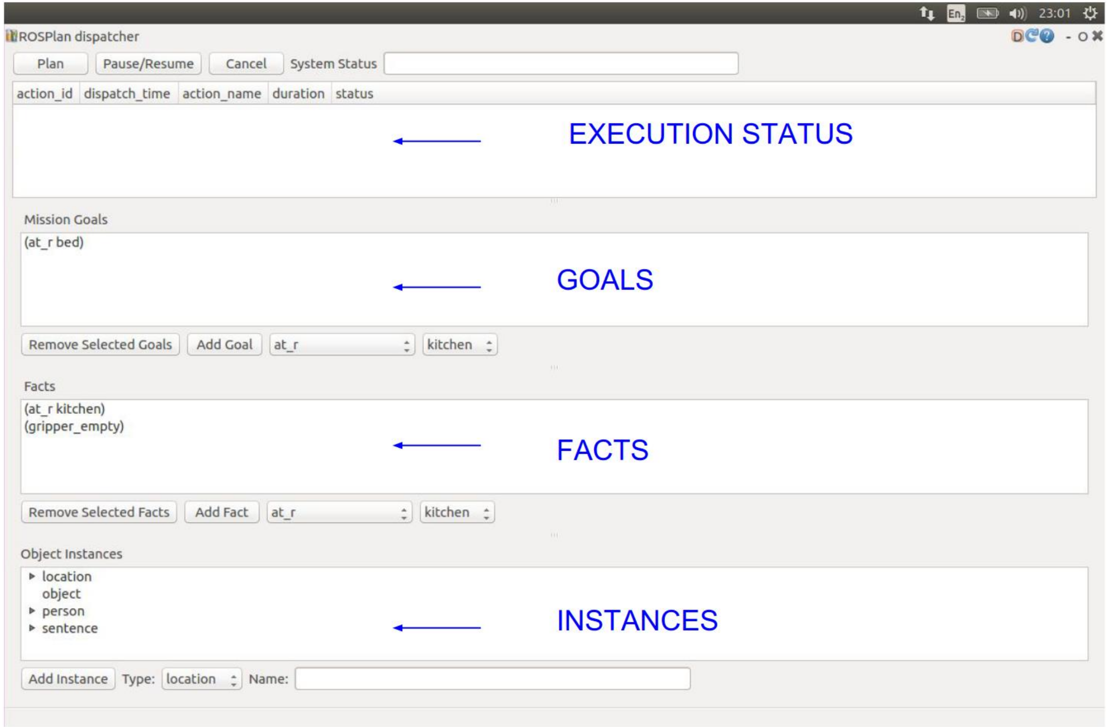
Object Instances

```
▾ area
    lab1
  arm
▾ arm_posture
    undef
  boolean
  physobj
```



ROSPlan KB gui

- Command: rqt --standalone rosplan_rqt.dispatcher.ROSPlanDispatcher



Mission Goals

(at_r bed)

Add Goal: at_r, kitchen

Facts

(at_r kitchen)
(gripper_empty)

Add Fact: at_r, kitchen

Object Instances

- location
- object
- person
- sentence

Type: location

Action interface (RPActionInterface)

- Provides a base class implementation to ease the process of robot action creation
- Available in C++ and Python
- Steps:
- 1) Subscribe to the plan
- 2) If action is not relevant, exit
- 3) Send actionlib feedback telling the action is enabled
- 4) Provide virtual function for concrete implementation
- 5) Upon action success, update KB according to the model
- 4) Send actionlib result (action achieved or failed)

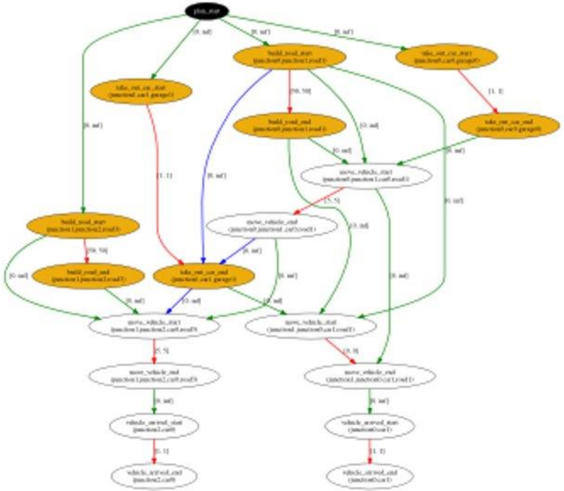
Simulated Actions (RPSimulatedActionInterface)

- Syntethic simulator of actions
- Made as a replacement of a physics-based robot simulator
- Make multiple mockup actions, useful for testing/debugging
- Parameters:
  - action_duration
  - action_duration_stddev
  - action_probability

Esterel plan dispatch

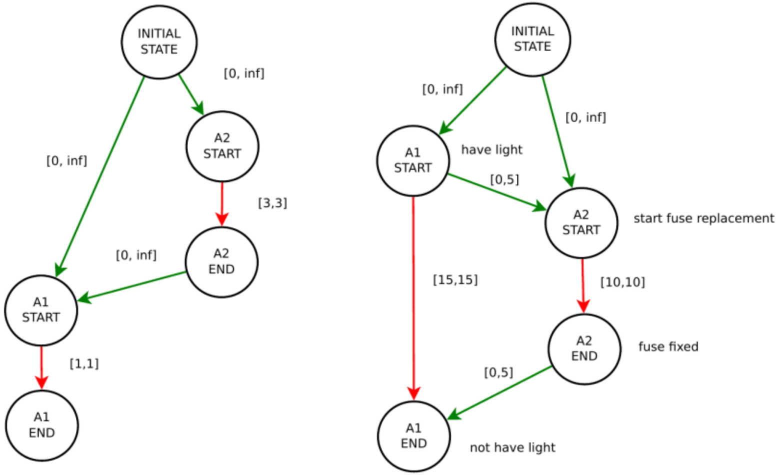
- Realtime graph-based algorithm for plan execution - monitoring
- Support for concurrent actions
- Preconditions are checked before sending action for execution

- In a nutshell: Converts a temporal PDDL planner output into a graph, which edges represent ordering constraints.



Semantics behind edges

- Conditional edge encapsulates 1 or more casual links
- All edges specify ordering constraints: source node effects need to
- happen before sink node gets signal
- Node cannot fire unless it has received all incoming edges / signals
- Examples:

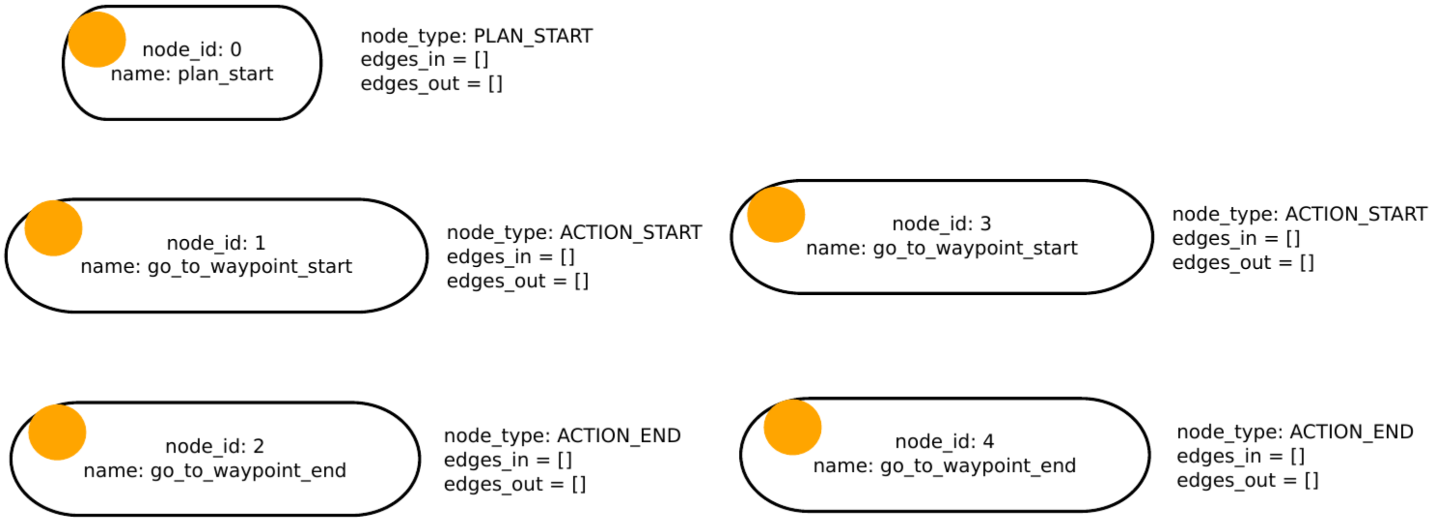


Esterel plan dispatch explanation 1/4

step 1 : PreparePlan() output

```
planner output example:
===
10.001: (goto_waypoint kenny wp0 wp1)  [10.000]
20.002: (goto_waypoint kenny wp1 wp2)  [10.000]
```

● Node associated action (not displayed)



node_id: 0
name: plan_start

node_type: PLAN_START
edges_in = []
edges_out = []

node_id: 1
name: go_to_waypoint_start

node_type: ACTION_START
edges_in = []
edges_out = []

node_id: 3
name: go_to_waypoint_start

node_type: ACTION_START
edges_in = []
edges_out = []

node_id: 2
name: go_to_waypoint_end

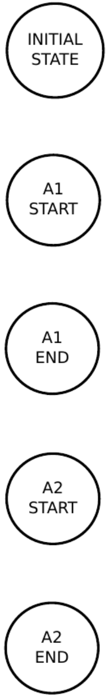
node_type: ACTION_END
edges_in = []
edges_out = []

node_id: 4
name: go_to_waypoint_end

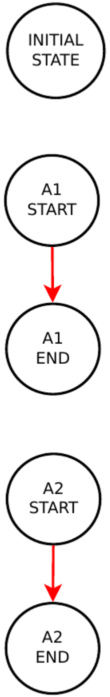
node_type: ACTION_END
edges_in = []
edges_out = []

Esterel plan dispatch explanation 2/4

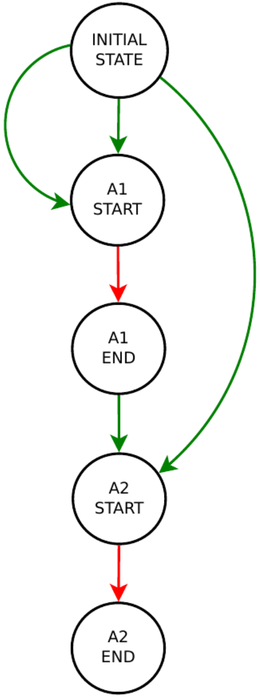
step 2 : order nodes by dispatch time



step 3 : create start-end action edges



step 4 : create conditional support edges



Esterel plan dispatch explanation 3/4

- Interference edge
- a and b interfere if:

$$eff^{+}a \cap eff^{-}b \neq \emptyset$$

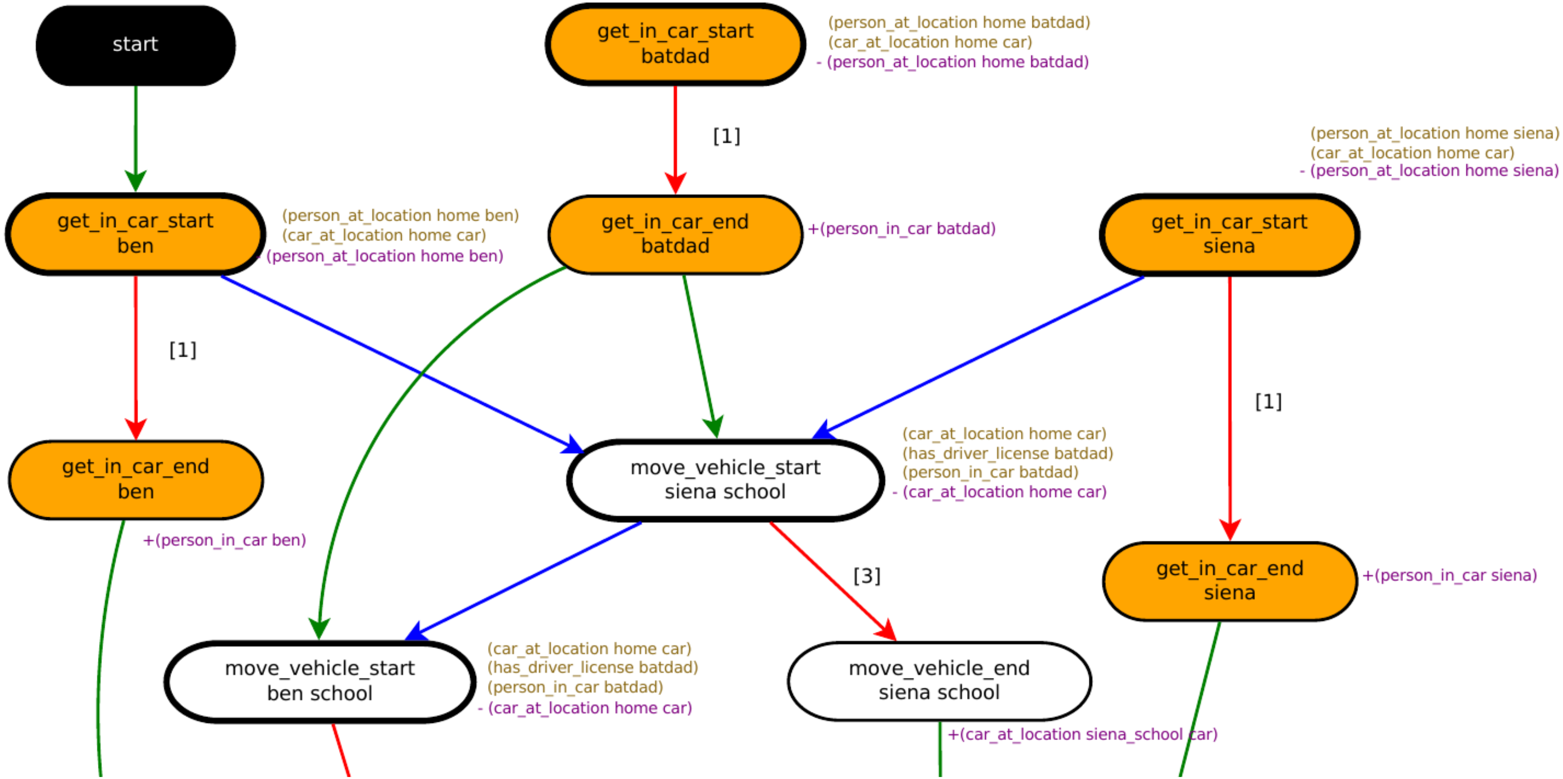
$$pre^{+}a \cap eff^{-}b \neq \emptyset \ *$$

$$pre^{-}a \cap eff^{+}b \neq \emptyset \ *$$

$$eff^{n}a \cap eff^{n}b \neq \emptyset$$

- ROSPlan currently checks the ones marked with *

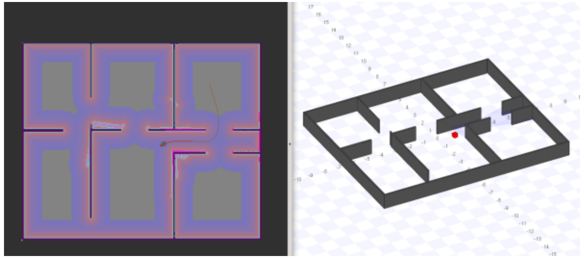
Esterel plan dispatch explanation 4/4



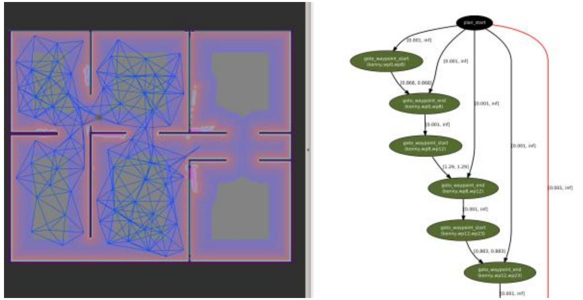
start
get_in_car_start batdad
(person_at_location home batdad)
(car_at_location home car)
- (person_at_location home batdad)
[1]
(person_at_location home siena)
(car_at_location home car)
- (person_at_location home siena)
get_in_car_start ben
(person_at_location home ben)
(car_at_location home car)
- (person_at_location home ben)
get_in_car_end batdad
+(person_in_car batdad)
get_in_car_start siena
[1]
[1]
get_in_car_end ben
move_vehicle_start siena school
(car_at_location home car)
(has_driver_license batdad)
(person_in_car batdad)
- (car_at_location home car)
+(person_in_car ben)
[3]
get_in_car_end siena
+(person_in_car siena)
move_vehicle_start ben school
(car_at_location home car)
(has_driver_license batdad)
(person_in_car batdad)
- (car_at_location home car)
move_vehicle_end siena school
+(car_at_location siena_school car)

ROSPlan demos

- Example launch file on how to launch a turtlebot3 robot in **stage** simulator



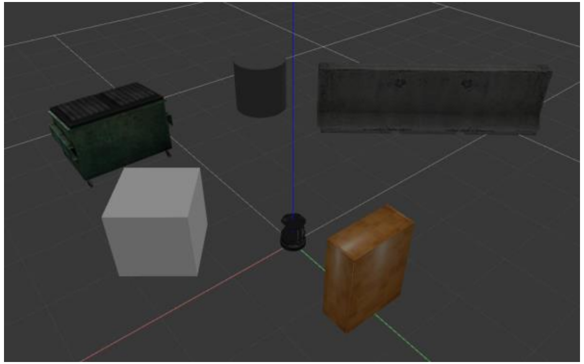
- **Exploration** demo in stage

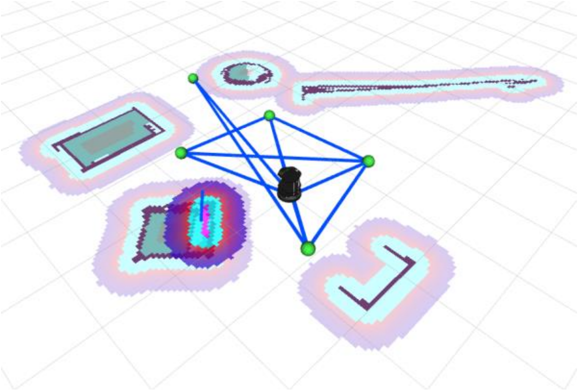


plan_start
[0.001, inf]
goto_waypoint_start (kenny,wp0,wp8)
[0.001, inf]
[0.868, 0.868]
goto_waypoint_end (kenny,wp0,wp8)
[0.001, inf]
[0.001, inf]
goto_waypoint_start (kenny,wp8,wp12)
[0.001, inf]
[1.29, 1.29]
goto_waypoint_end (kenny,wp8,wp12)
[0.001, inf]
[0.001, inf]
goto_waypoint_start (kenny,wp12,wp23)
[0.883, 0.883]
goto_waypoint_end (kenny,wp12,wp23)
[0.001, inf]

ROSPlan demos

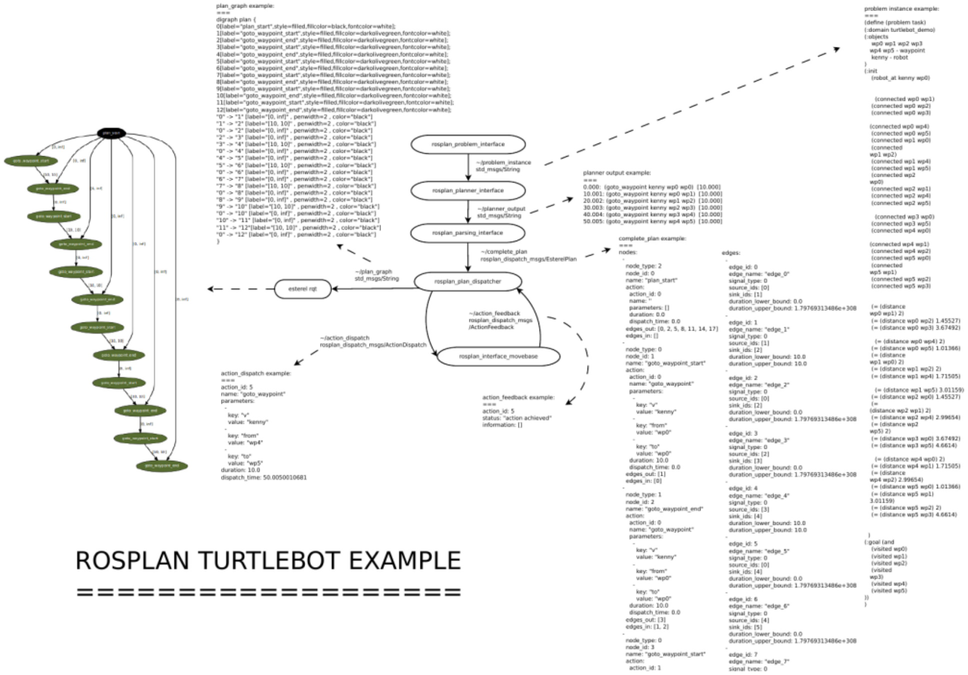
- Turtlebot exploration demo in Gazebo





ROSPlan turtlebot demo

- https://github.com/oscar-lima/rosplan_debug/blob/kinetic/rpd_turtlebot_demo/ros/doc/rosplan_turtlebot_example_detail.pdf



```
plan_graph example:
===
digraph plan {
0[label="plan_start",style=filled,fillcolor=black,fontcolor=white];
1[label="goto_waypoint_start",style=filled,fillcolor=darkolivegreen,fontcolor=white];
2[label="goto_waypoint_end",style=filled,fillcolor=darkolivegreen,fontcolor=white];
3[label="goto_waypoint_start",style=filled,fillcolor=darkolivegreen,fontcolor=white];
4[label="goto_waypoint_end",style=filled,fillcolor=darkolivegreen,fontcolor=white];
5[label="goto_waypoint_start",style=filled,fillcolor=darkolivegreen,fontcolor=white];
6[label="goto_waypoint_end",style=filled,fillcolor=darkolivegreen,fontcolor=white];
7[label="goto_waypoint_start",style=filled,fillcolor=darkolivegreen,fontcolor=white];
8[label="goto_waypoint_end",style=filled,fillcolor=darkolivegreen,fontcolor=white];
9[label="goto_waypoint_start",style=filled,fillcolor=darkolivegreen,fontcolor=white];
10[label="goto_waypoint_end",style=filled,fillcolor=darkolivegreen,fontcolor=white];
11[label="goto_waypoint_start",style=filled,fillcolor=darkolivegreen,fontcolor=white];
12[label="goto_waypoint_end",style=filled,fillcolor=darkolivegreen,fontcolor=white];
"0" -> "1" [label="[0, inf]" , penwidth=2 , color="black"]
"1" -> "2" [label="[10, 10]" , penwidth=2 , color="black"]
"0" -> "2" [label="[0, inf]" , penwidth=2 , color="black"]
"2" -> "3" [label="[0, inf]" , penwidth=2 , color="black"]
"3" -> "4" [label="[10, 10]" , penwidth=2 , color="black"]
"0" -> "4" [label="[0, inf]" , penwidth=2 , color="black"]
"4" -> "5" [label="[0, inf]" , penwidth=2 , color="black"]
"5" -> "6" [label="[10, 10]" , penwidth=2 , color="black"]
"0" -> "6" [label="[0, inf]" , penwidth=2 , color="black"]
"6" -> "7" [label="[0, inf]" , penwidth=2 , color="black"]
"7" -> "8" [label="[10, 10]" , penwidth=2 , color="black"]
"0" -> "8" [label="[0, inf]" , penwidth=2 , color="black"]
"8" -> "9" [label="[0, inf]" , penwidth=2 , color="black"]
"9" -> "10" [label="[10, 10]" , penwidth=2 , color="black"]
"0" -> "10" [label="[0, inf]" , penwidth=2 , color="black"]
"10" -> "11" [label="[0, inf]" , penwidth=2 , color="black"]
"11" -> "12"[label="[10, 10]" , penwidth=2 , color="black"]
"0" -> "12" [label="[0, inf]" , penwidth=2 , color="black"]
}
```

```
planner output example:
===
0.000:  (goto_waypoint kenny wp0 wp0)  [10.000]
10.001: (goto_waypoint kenny wp0 wp1)  [10.000]
20.002: (goto_waypoint kenny wp1 wp2)  [10.000]
30.003: (goto_waypoint kenny wp2 wp3)  [10.000]
40.004: (goto_waypoint kenny wp3 wp4)  [10.000]
50.005: (goto_waypoint kenny wp4 wp5)  [10.000]
```

```
complete_plan example:
===
nodes:
 -
  node_type: 2
  node_id: 0
  name: "plan_start"
  action:
    action_id: 0
    name: ''
    parameters: []
    duration: 0.0
    dispatch_time: 0.0
  edges_out: [0, 2, 5, 8, 11, 14, 17]
  edges_in: []
 -
  node_type: 0
  node_id: 1
  name: "goto_waypoint_start"
  action:
    action_id: 0
    name: "goto_waypoint"
    parameters:
      -
        key: "v"
        value: "kenny"
      -
        key: "from"
        value: "wp0"
      -
        key: "to"
        value: "wp0"
    duration: 10.0
    dispatch_time: 0.0
  edges_out: [1]
  edges_in: [0]
 -
  node_type: 1
  node_id: 2
  name: "goto_waypoint_end"
  action:
    action_id: 0
    name: "goto_waypoint"
    parameters:
      -
        key: "v"
        value: "kenny"
      -
        key: "from"
        value: "wp0"
      -
        key: "to"
        value: "wp0"
    duration: 10.0
    dispatch_time: 0.0
  edges_out: [3]
  edges_in: [1, 2]
 -
  node_type: 0
  node_id: 3
  name: "goto_waypoint_start"
  action:
    action_id: 1
```

```
edges:
 -
  edge_id: 0
  edge_name: "edge_0"
  signal_type: 0
  source_ids: [0]
  sink_ids: [1]
  duration_lower_bound: 0.0
  duration_upper_bound: 1.79769313486e+308
 -
  edge_id: 1
  edge_name: "edge_1"
  signal_type: 0
  source_ids: [1]
  sink_ids: [2]
  duration_lower_bound: 10.0
  duration_upper_bound: 10.0
 -
  edge_id: 2
  edge_name: "edge_2"
  signal_type: 0
  source_ids: [0]
  sink_ids: [2]
  duration_lower_bound: 0.0
  duration_upper_bound: 1.79769313486e+308
 -
  edge_id: 3
  edge_name: "edge_3"
  signal_type: 0
  source_ids: [2]
  sink_ids: [3]
  duration_lower_bound: 0.0
  duration_upper_bound: 1.79769313486e+308
 -
  edge_id: 4
  edge_name: "edge_4"
  signal_type: 0
  source_ids: [3]
  sink_ids: [4]
  duration_lower_bound: 10.0
  duration_upper_bound: 10.0
 -
  edge_id: 5
  edge_name: "edge_5"
  signal_type: 0
  source_ids: [0]
  sink_ids: [4]
  duration_lower_bound: 0.0
  duration_upper_bound: 1.79769313486e+308
 -
  edge_id: 6
  edge_name: "edge_6"
  signal_type: 0
  source_ids: [4]
  sink_ids: [5]
  duration_lower_bound: 0.0
  duration_upper_bound: 1.79769313486e+308
 -
  edge_id: 7
  edge_name: "edge_7"
  signal_type: 0
```

```
problem instance example:
===
(define (problem task)
(:domain turtlebot_demo)
(:objects
   wp0 wp1 wp2 wp3
  wp4 wp5 - waypoint
   kenny - robot
)
(:init
   (robot_at kenny wp0)


     (connected wp0 wp1)
   (connected wp0 wp2)
   (connected wp0 wp3)

  (connected wp0 wp4)
   (connected wp0 wp5)
   (connected wp1 wp0)
   (connected
  wp1 wp2)
   (connected wp1 wp4)
   (connected wp1 wp5)
   (connected wp2
  wp0)
   (connected wp2 wp1)
   (connected wp2 wp4)
   (connected wp2 wp5)

     (connected wp3 wp0)
   (connected wp3 wp5)
   (connected wp4 wp0)

  (connected wp4 wp1)
   (connected wp4 wp2)
   (connected wp5 wp0)
   (connected
  wp5 wp1)
   (connected wp5 wp2)
   (connected wp5 wp3)


   (= (distance
  wp0 wp1) 2)
   (= (distance wp0 wp2) 1.45527)
   (= (distance wp0 wp3) 3.67492)

     (= (distance wp0 wp4) 2)
   (= (distance wp0 wp5) 1.01366)
   (= (distance
  wp1 wp0) 2)
   (= (distance wp1 wp2) 2)
   (= (distance wp1 wp4) 1.71505)

     (= (distance wp1 wp5) 3.01159)
   (= (distance wp2 wp0) 1.45527)
   (=
  (distance wp2 wp1) 2)
   (= (distance wp2 wp4) 2.99654)
   (= (distance wp2
  wp5) 2)
   (= (distance wp3 wp0) 3.67492)
   (= (distance wp3 wp5) 4.6614)

     (= (distance wp4 wp0) 2)
   (= (distance wp4 wp1) 1.71505)
   (= (distance
  wp4 wp2) 2.99654)
   (= (distance wp5 wp0) 1.01366)
   (= (distance wp5 wp1)
  3.01159)
   (= (distance wp5 wp2) 2)
   (= (distance wp5 wp3) 4.6614)


  )
(:goal (and
   (visited wp0)
   (visited wp1)
   (visited wp2)
   (visited
  wp3)
   (visited wp4)
   (visited wp5)
))
)
```

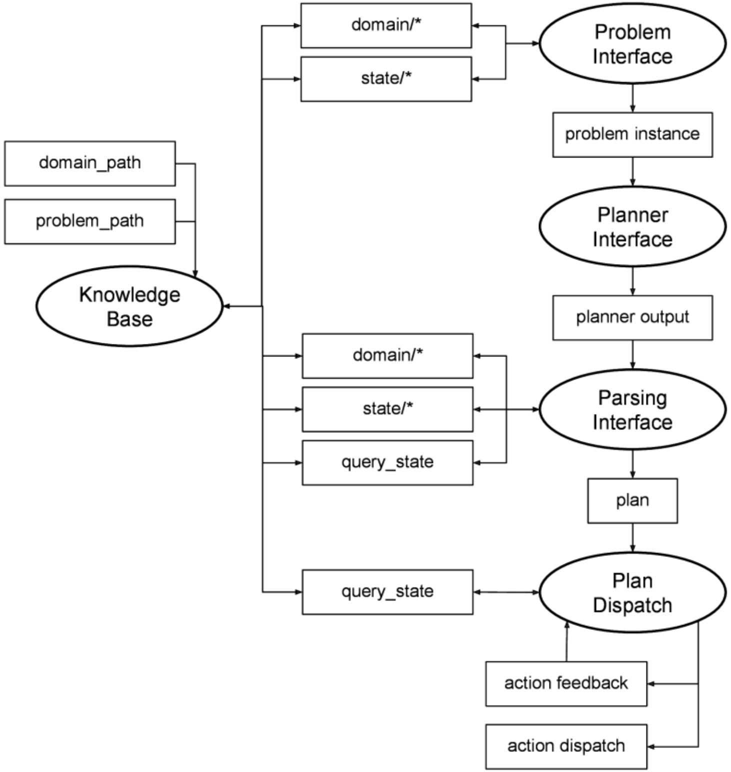
```
action_dispatch example:
===
action_id: 5
name: "goto_waypoint"
parameters:
  -
   key: "v"
   value: "kenny"
  -
   key: "from"
   value: "wp4"
  -
   key: "to"
   value: "wp5"
duration: 10.0
dispatch_time: 50.0050010681
```

```
action_feedback example:
===
action_id: 5
status: "action achieved"
information: []
```

ROSPLAN TURTLEBOT EXAMPLE
===================

Course structure

1. ROS
2. ROSPlan
3. **Q+A**



domain/*

state/*

Problem Interface

problem instance

Planner Interface

planner output

domain_path

problem_path

Knowledge Base

domain/*

state/*

query_state

Parsing Interface

plan

query_state

Plan Dispatch

action feedback

action dispatch