ROAR: an agent oriented architecture for the autonomy of robots

Arnaud Degroote Simon Lacroix
{adegroot,simon}@laas.fr

Laboratoire d'Analyse et d'Architecture des Systèmes, Toulouse, FRANCE

10 mai 2012

LAAS-CNRS

Problematic

A coordination problem for autonomous robots

Considering a population of functional components, how to arrange them

- to have a globally coherent behaviour (and to fill the mission)
- to react accordingly the environment

Problematic

A coordination problem for autonomous robots

Considering a population of functional components, how to arrange them

- to have a globally coherent behaviour (and to fill the mission)
- to react accordingly the environment

Requirements

- Deliberation time more or less important
- Handling concurrency correctly
- Robustness
- Modular, composable, open
- Expressive enough

Presentation Plan

How to split the robot system in a composable system ?

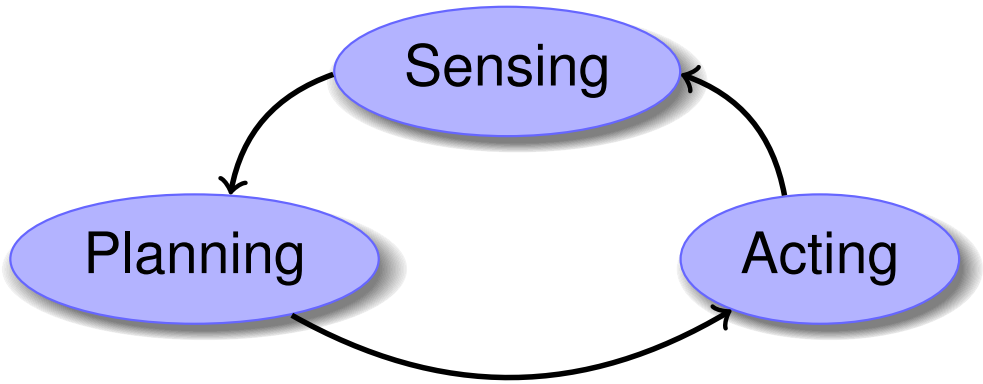
Why splitting the robot coordination layer ?

- to keep sub-system reactive enough (following IDEA and T-REX)
- for modularity purpose

How to split the robot system in a composable system ?

Why splitting the robot coordination layer ?

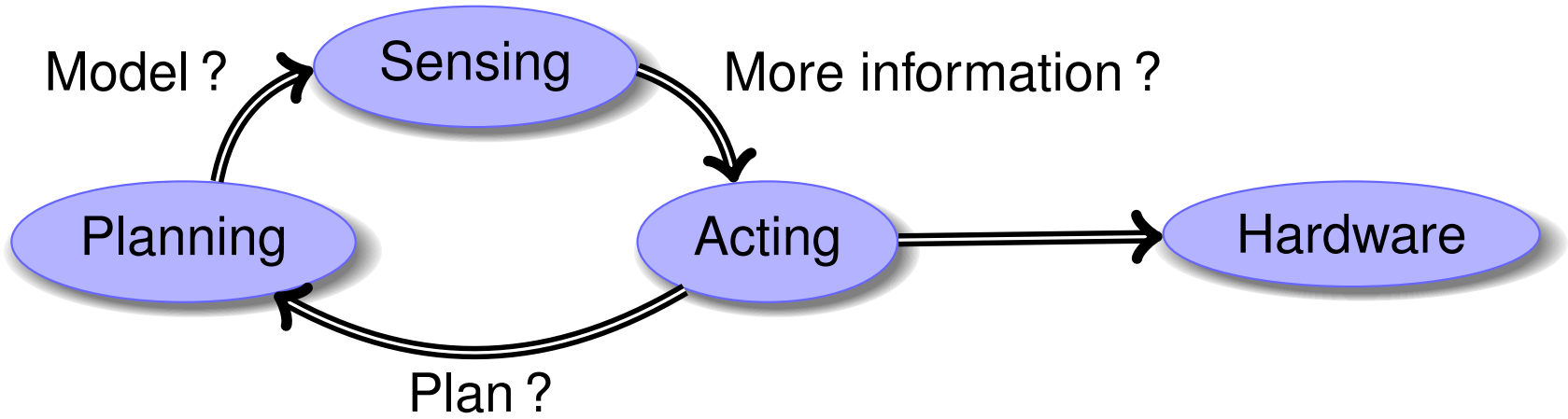
- to keep sub-system reactive enough (following IDEA and T-REX)
- for modularity purpose



How to split the robot system in a composable system ?

Why splitting the robot coordination layer ?

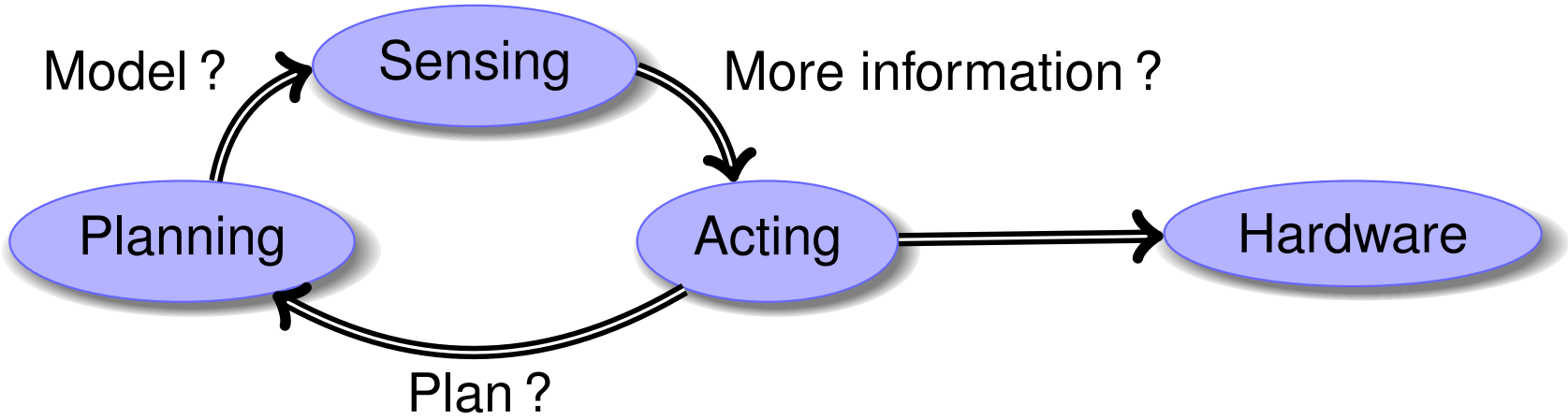
- to keep sub-system reactive enough (following IDEA and T-REX)
- for modularity purpose



How to split the robot system in a composable system ?

Why splitting the robot coordination layer ?

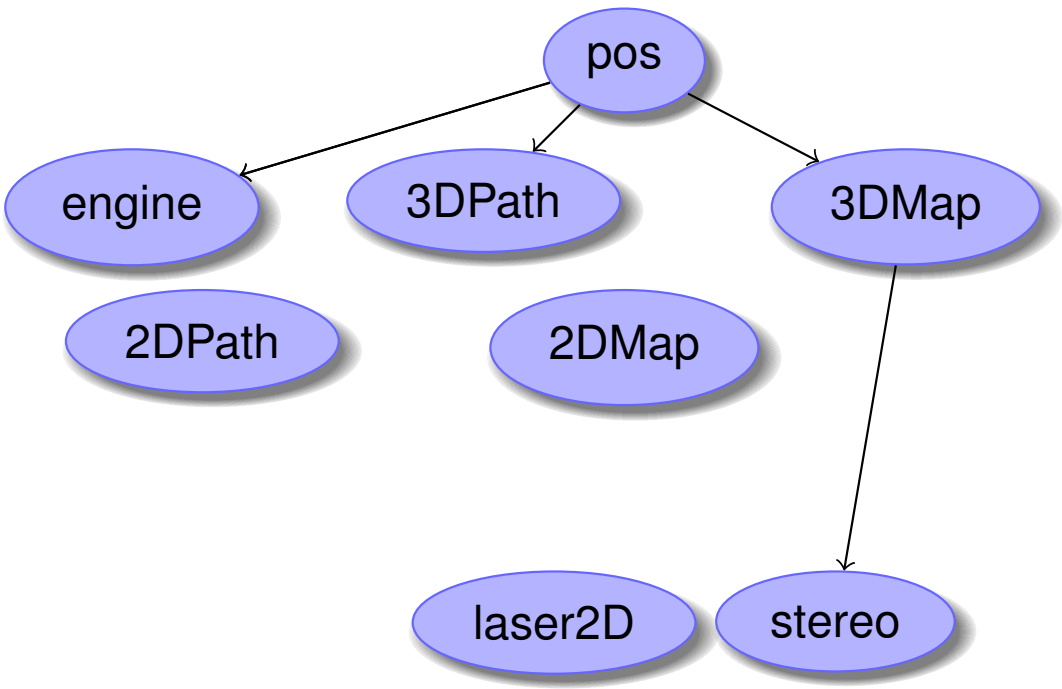
- to keep sub-system reactive enough (following IDEA and T-REX)
- for modularity purpose



Why partitioning on top of resource ?

- Resource is a first-class object of the functional layer
- Competition on resource acquisition
- Resource is robot-agnostic

A dynamic graph of agents



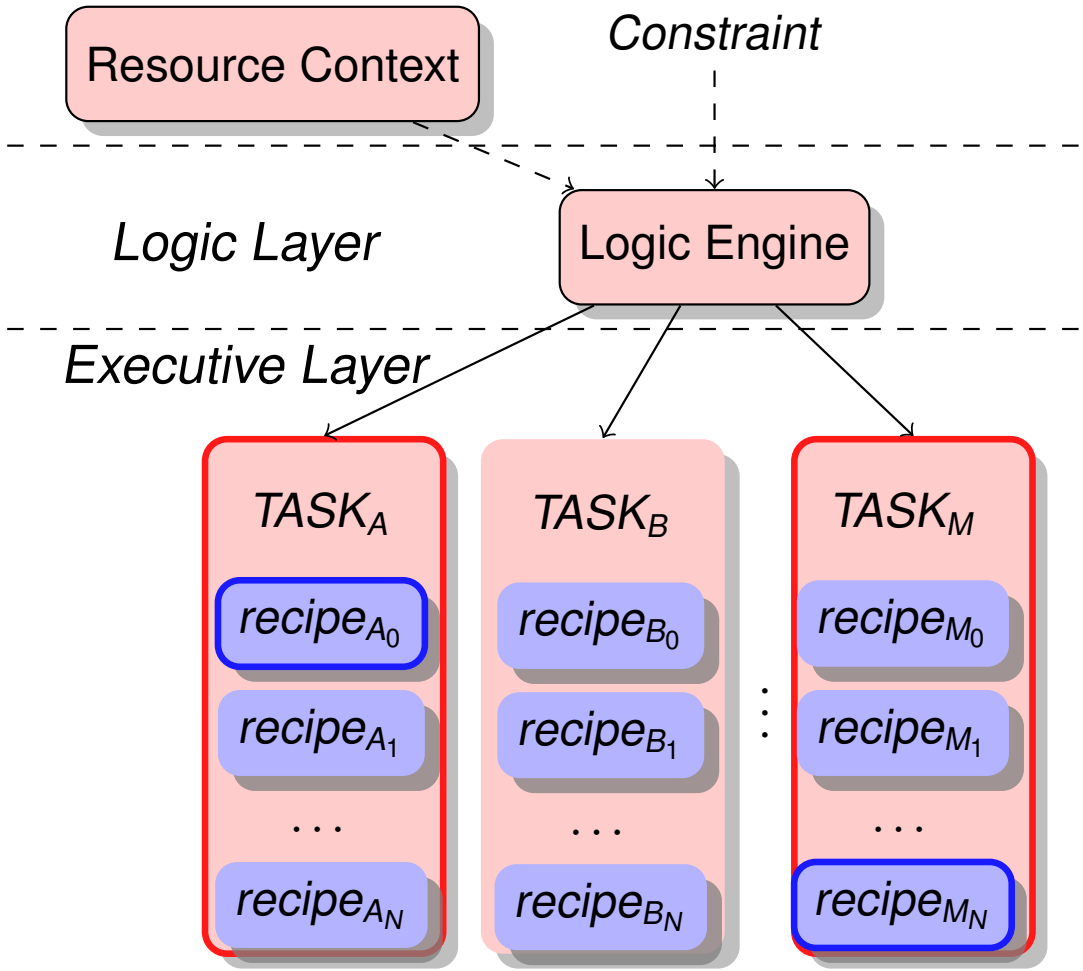
Agent resource

- Robustness
- Composition
- Generic

Relation between agents

- constraints messages as logic proposition
- data messages

Inside a ROAR agent



Logic layer

Task

- Represents transition from one logic state to another logic state
- Defining by the developer as a contract (pre and post conditions)

Logic layer

Task

- Represents transition from one logic state to another logic state
- Defining by the developer as a contract (pre and post conditions)

How to decide if a task can handle a constraint

Solving a first-order logic Horn Clause of kind

$$Post_1 \wedge Post_2 \wedge \cdots \wedge Post_i \wedge R_1 \wedge \cdots \wedge R_i \rightarrow C$$

Logic layer

Task

- Represents transition from one logic state to another logic state
- Defining by the developer as a contract (pre and post conditions)

How to decide if a task can handle a constraint

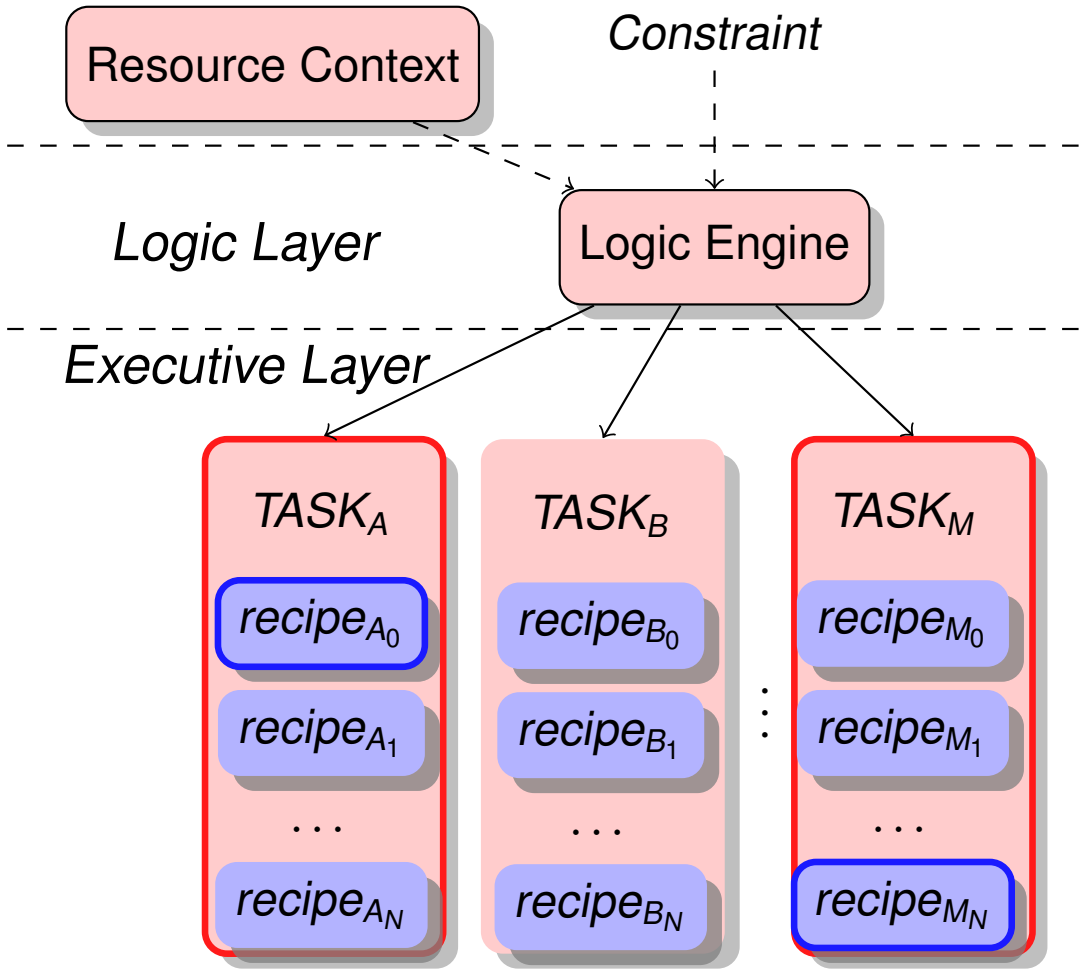
Solving a first-order logic Horn Clause of kind

$$Post_1 \wedge Post_2 \wedge \cdots \wedge Post_i \wedge R_1 \wedge \cdots \wedge R_i \rightarrow C$$

Generate task tree

- Consider failed-precondition as new constraints
- Search depth-first, one each time (same strategy than Prolog)
- If multiples alternatives, choose tasks using the following protocol
  1. minimise number hypothesis of remote agents
  2. minimise number hypothesis of local agents / failed pre-conditions

Inside a ROAR agent



Executing a task

Recipe definition

- a set of pre-conditions
- a body (the implementation), written by a developer

Executing a task

Recipe definition

- a set of pre-conditions
- a body (the implementation), written by a developer

Selecting a recipe

1. evaluate pre-condition for each recipe
2. remove recipe with failed pre-condition
3. choose one possible recipe, depending of a 'by-agent' heuristic
   - **best-satisfaction** : the most appropriated recipe
   - **optimise-cost**
   - **optimise-time**
   - ...

Recipe example

```
letname mapping_ctr distance(3DMap::last_update, pos::current) <= 0.5
letname plan_ctr LocalPathPlanner::goal == LocalPathPlanner::expected_goal
letname exec_ctr locomotion::expected_port == locomotion::port
letname terminaison_cond distance(pos::current, pos::goal) < 2.0
letname monitor_ctr trajectory::expected_port == trajectory::tracked_port

let follow_goal = fun {
  ensure(plan_ctr where LocalPathPlanner::expected_goal == goal)
  ensure(exec_ctr where locomotion::expected_port == LocalPathPlanner::port)
  ensure(monitor_ctr where trajectory::expected_port == LocalPathPlanner::port)
  wait(terminaison_cond)
}

let move_blind_zone speed_ distance = fun {
  let speed make_speed(speed_, 0.0)
  let id ensure(locomotion::_speed == locomotion::speed where locomotion::_speed == speed)
  let cur pos::current
  wait(distance(pos::current, cur) > distance)
  abort id
}

go_to = recipe {
  pre = {{3DMap::empty == true}}
  post = {}
  body = {
    make(delay(3DMap::last_refresh) < 50.0)
    ensure(mapping_ctr)
    move_blind_zone(1.0, 3.0)
    follow_goal()
  }}
```

LAAS-CNRS

Handling errors

An error ?

- Unexpected error from the functional layer
- Rejected constraint from an agent

Handling errors

An error ?

- Unexpected error from the functional layer
- Rejected constraint from an agent

Locally Handling error

1. Select another recipe that handles this specific error
2. Select another recipe that does not use the offending constraint
3. Select another combination of task to handle the constraint

Handling errors

An error ?

- Unexpected error from the functional layer
- Rejected constraint from an agent

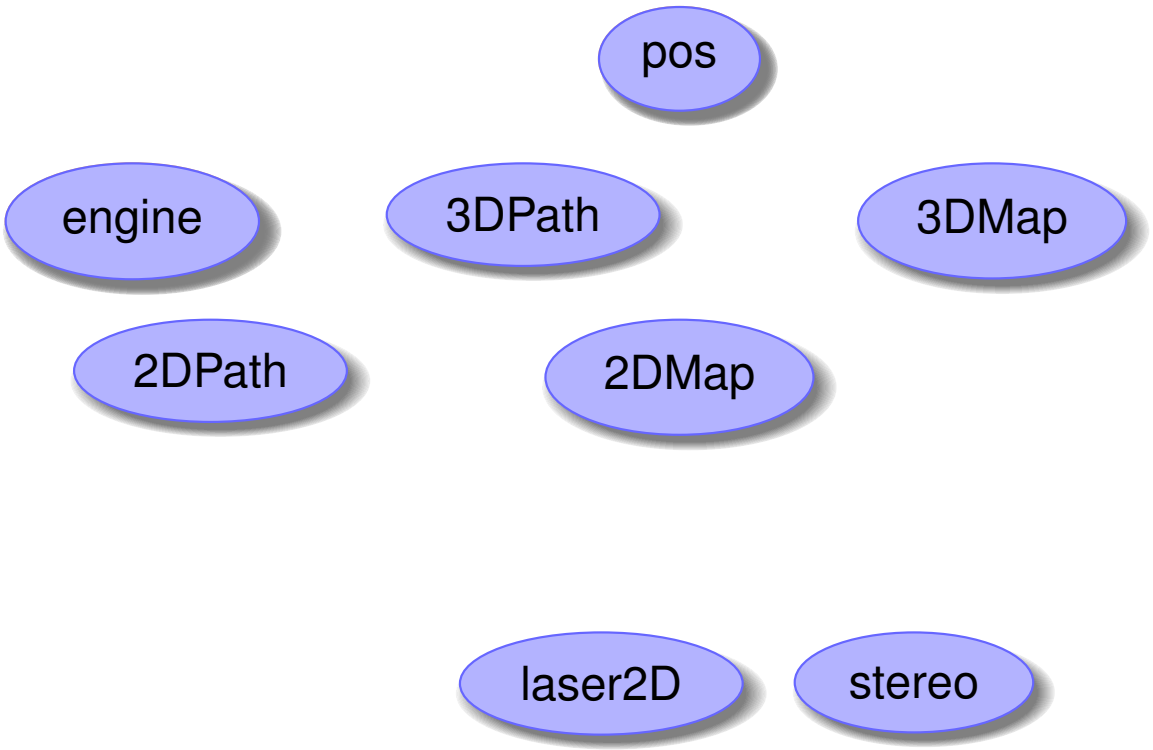
Locally Handling error

1. Select another recipe that handles this specific error
2. Select another recipe that does not use the offending constraint
3. Select another combination of task to handle the constraint

Handling error at the system level : Backtracking in the graph

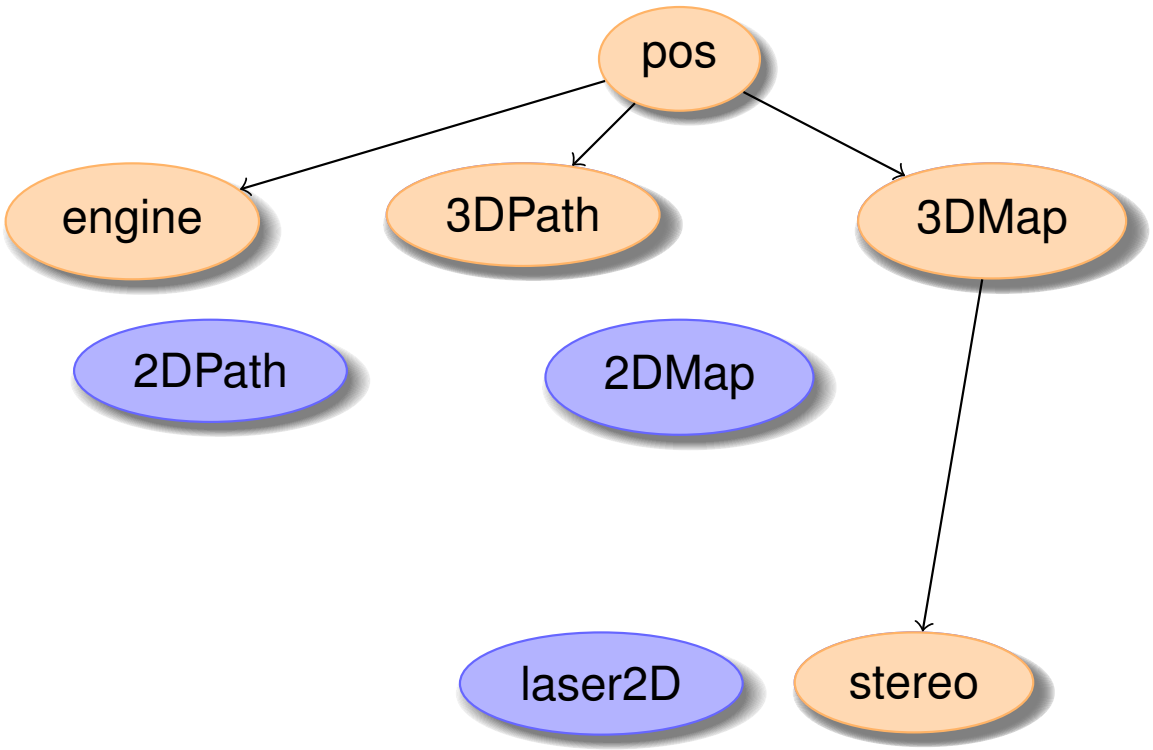
- The failure goes up in the constraint graph, with error context
- Error context is the set of constraints which failed to be enforced
- At each level, use the previously methodology to find a solution

Handling errors at the system level : instantiation



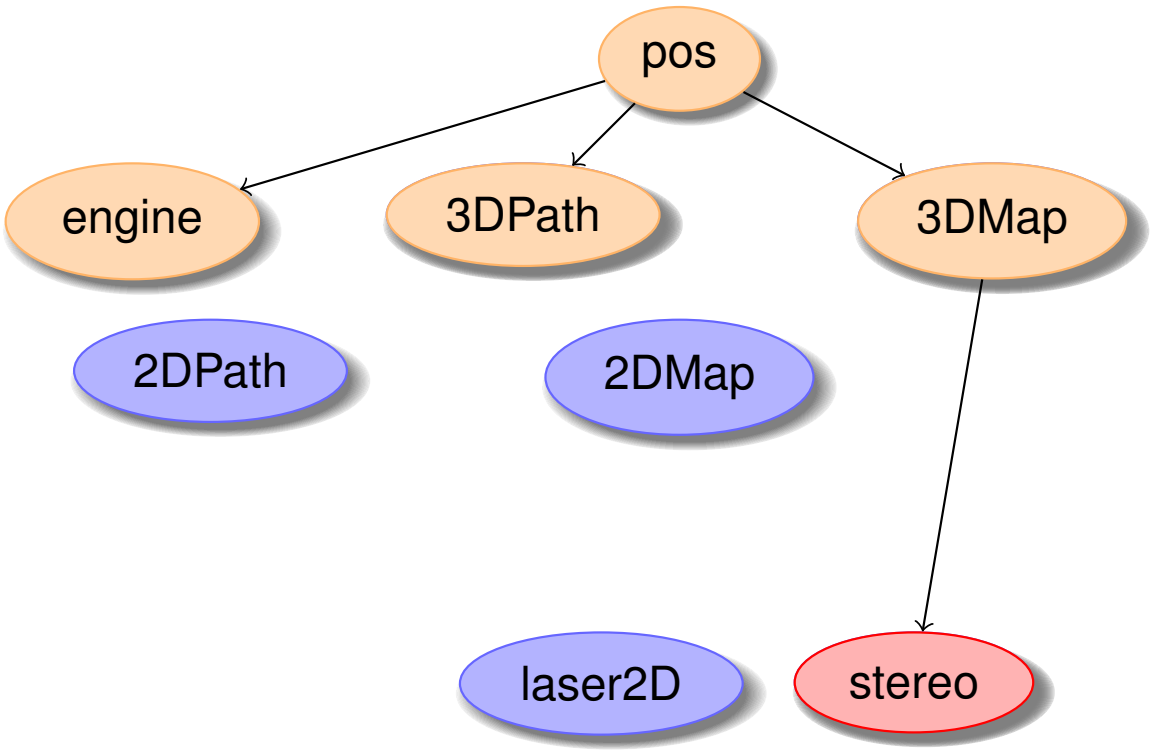
Goto (X, Y)

Handling errors at the system level : instantiation



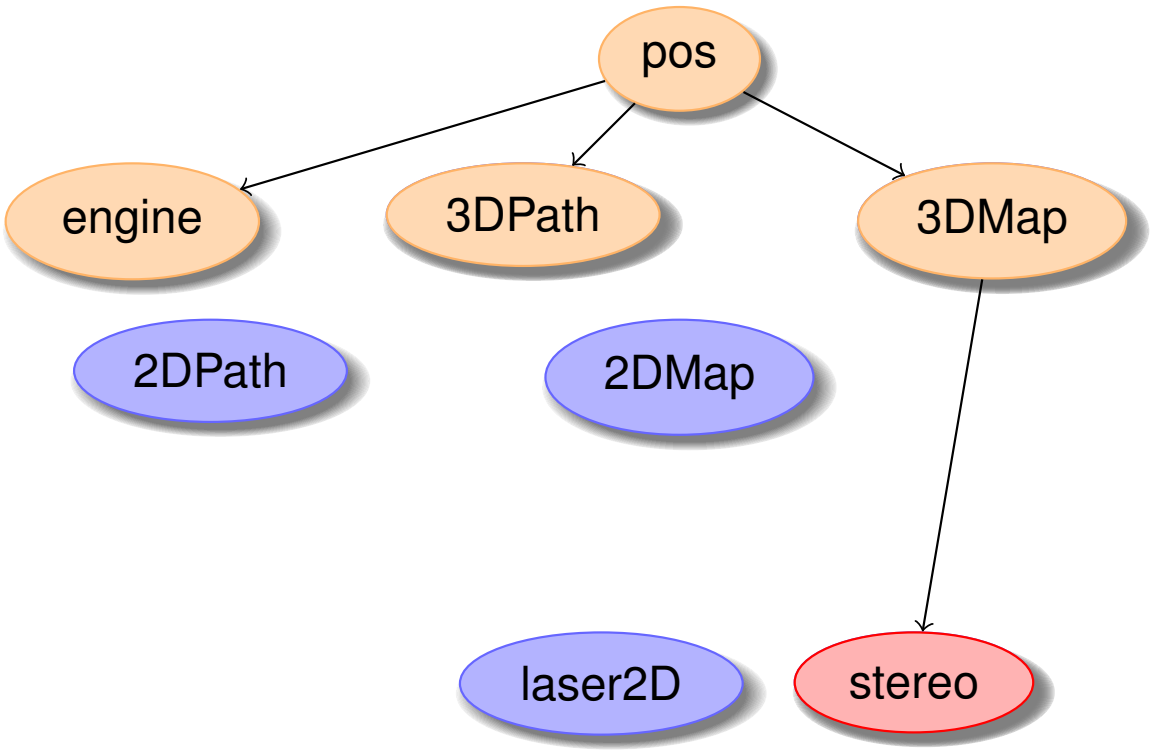
Using 3D vision strategy

Handling errors at the system level : instantiation



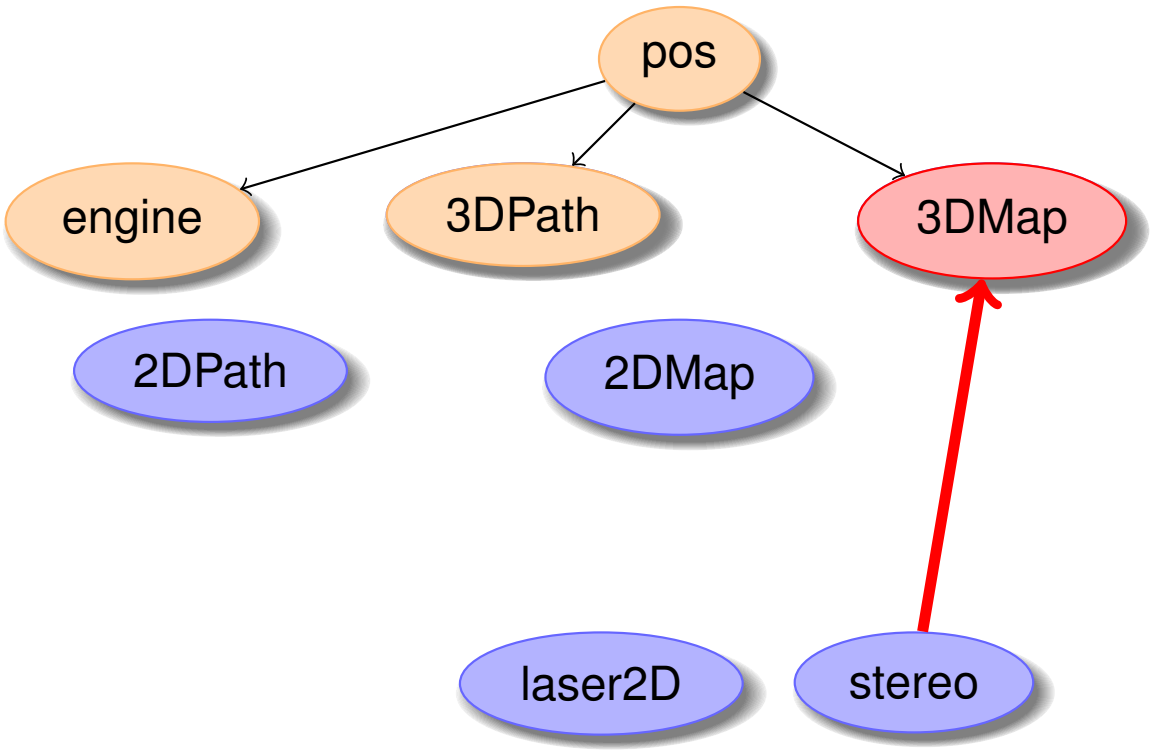
Stereovision fails
⇒ another recipe ?

Handling errors at the system level : instantiation



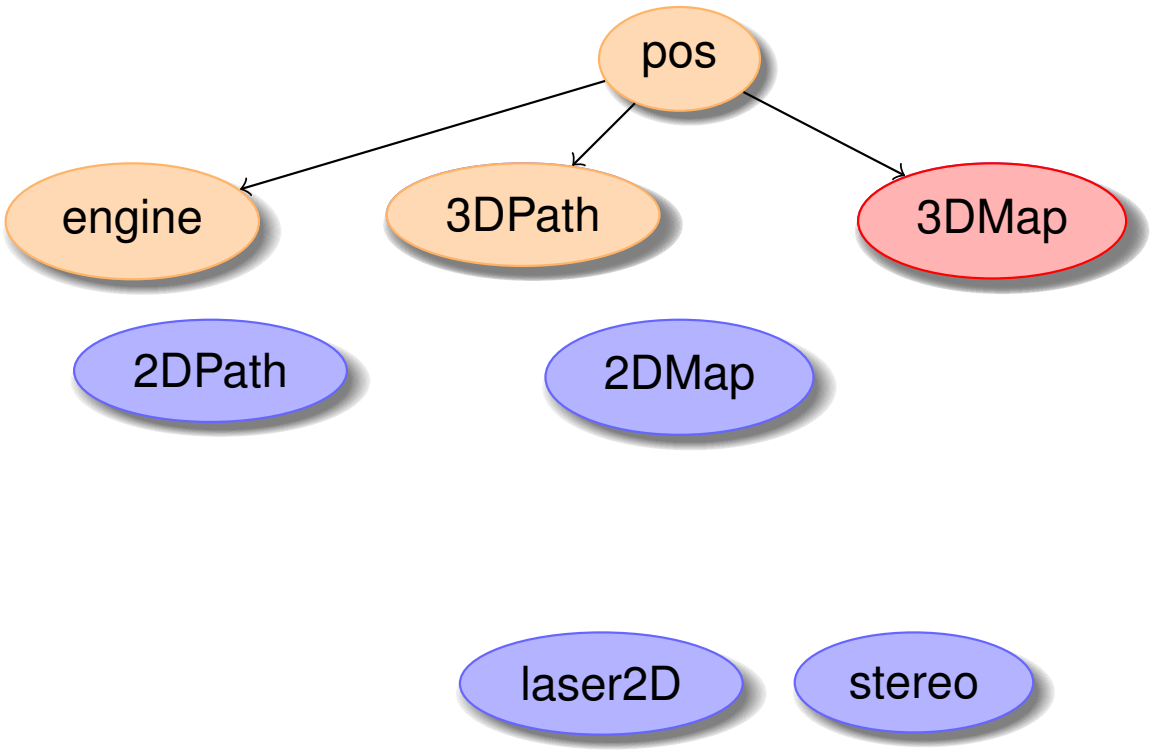
No other recipe
⇒ Another task set ?

Handling errors at the system level : instantiation



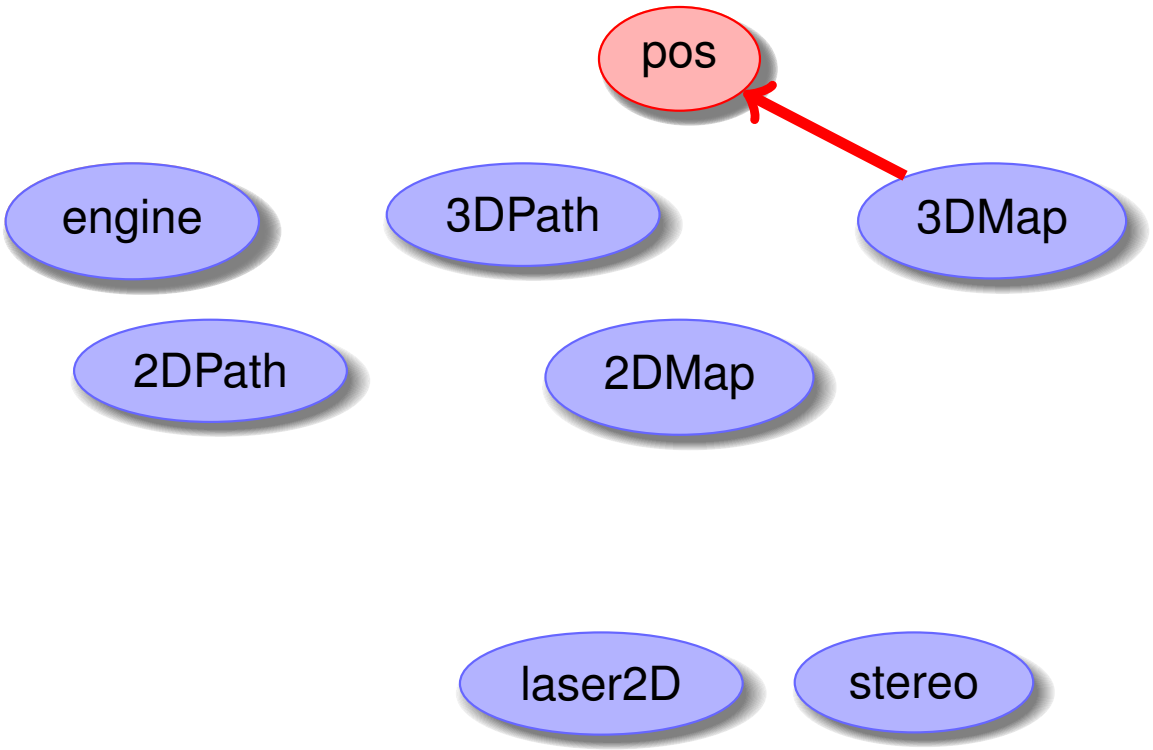
3dMap fails due to stereo failure
⇒ another recipe ?

Handling errors at the system level : instantiation



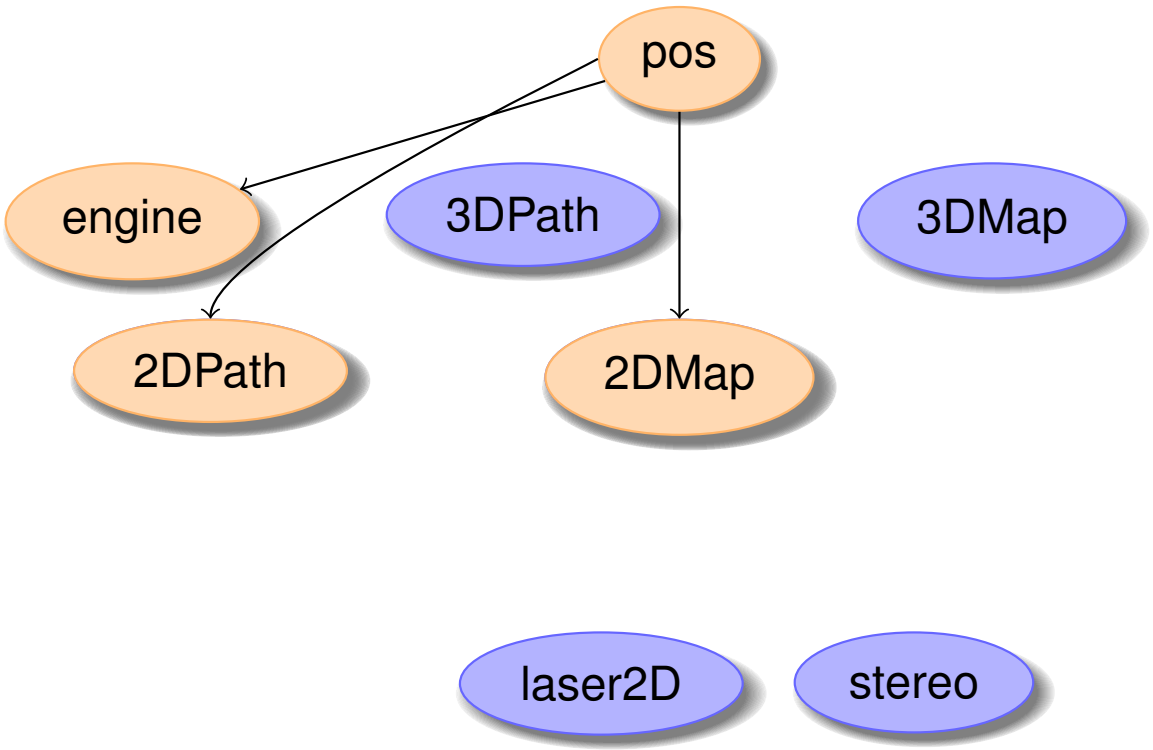
No other recipe
⇒ Another task set ?

Handling errors at the system level : instantiation



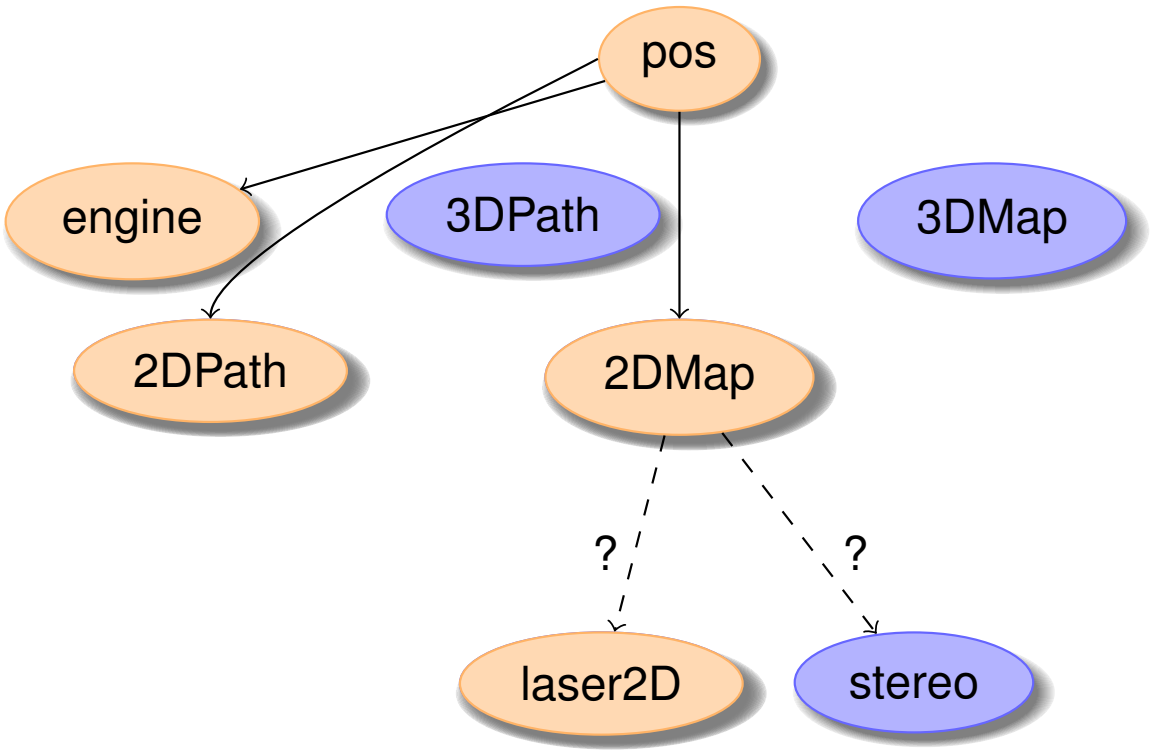
pos fails due to 3DMap failure
⇒ another recipe ?

Handling errors at the system level : instantiation



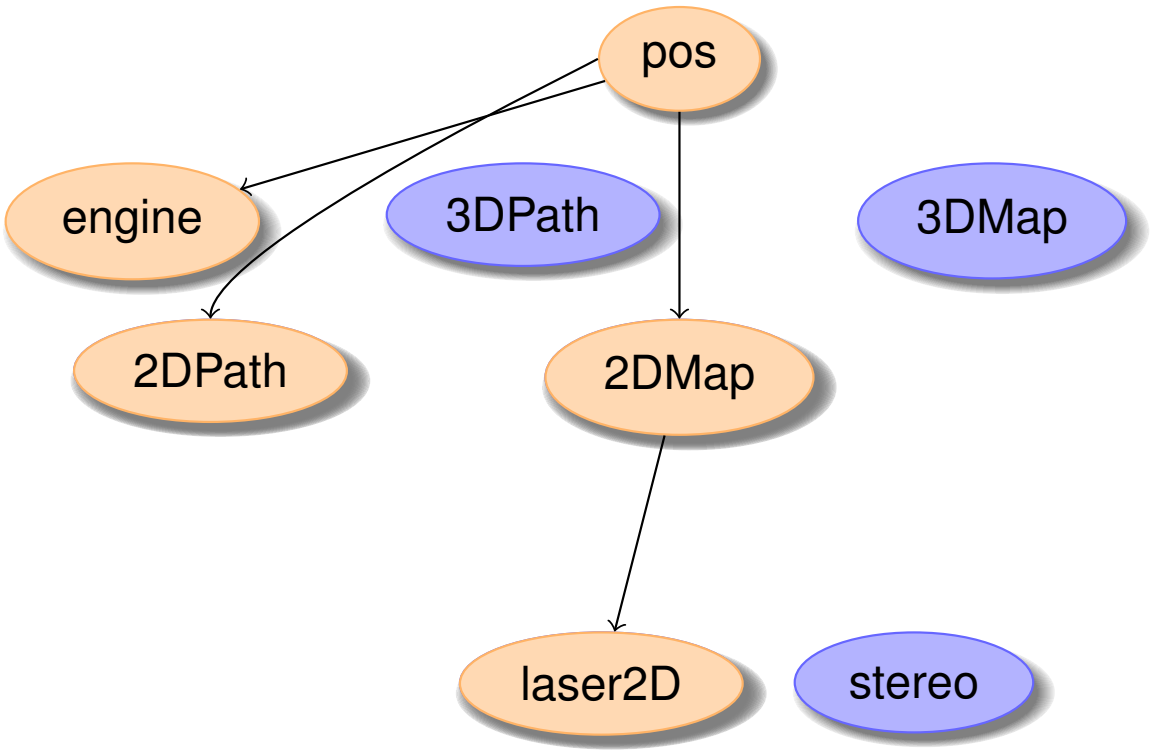
Will use 2D Map strategy

Handling errors at the system level : instantiation



Use stereovision or laser $\Rightarrow$ laser

Handling errors at the system level : instantiation



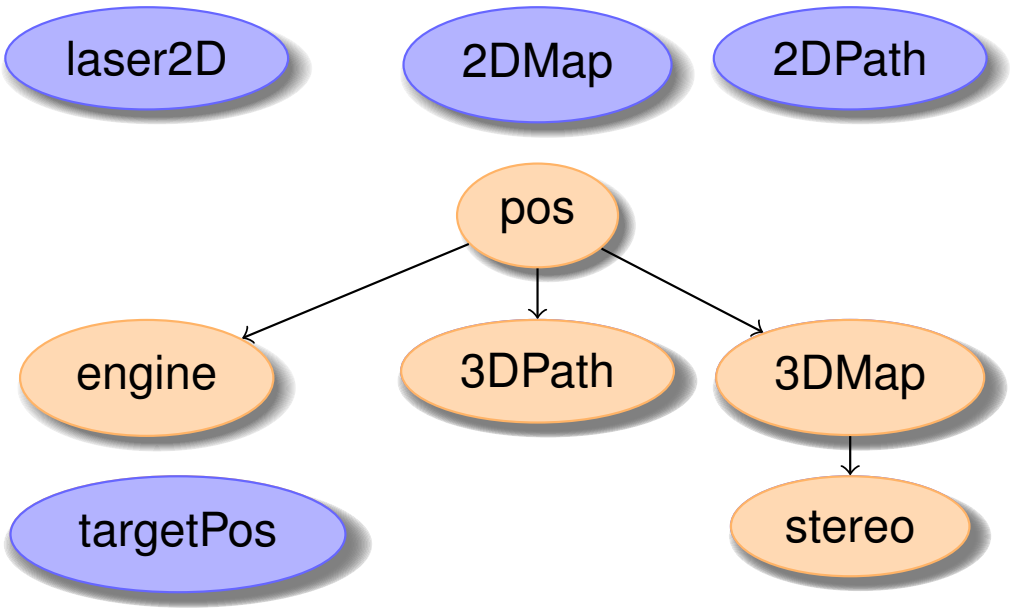
2D Map strategy is running

Handling concurrency issue

At the system level

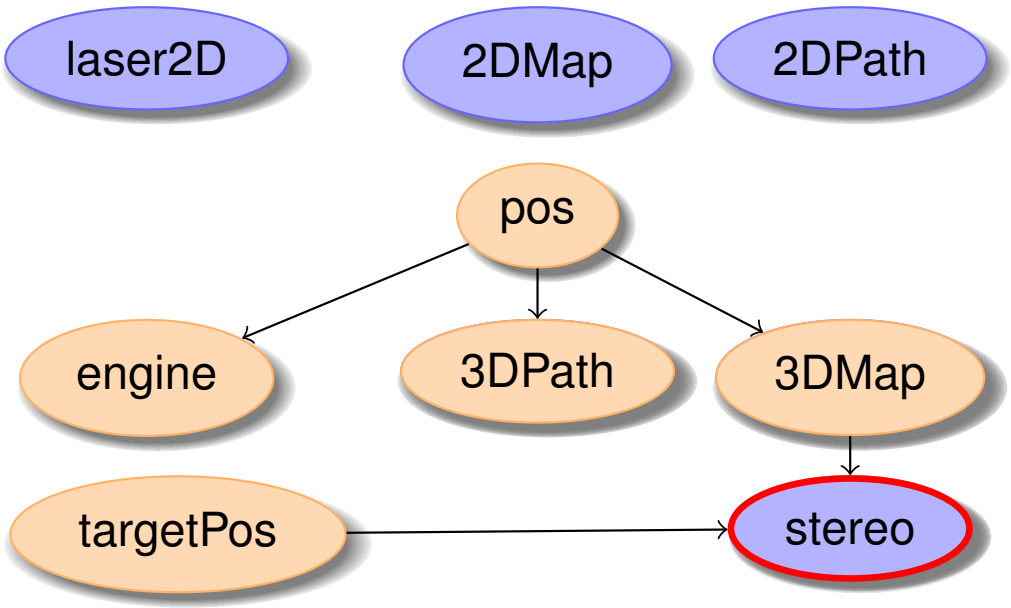
1. Get the constraint context which leads to an inconsistency
2. For each index, try to see if we can resolve C_{i+1} without C_i
3. If the system does not find any solution, treat the error as shown previously

Handling concurrency issue : instantiation



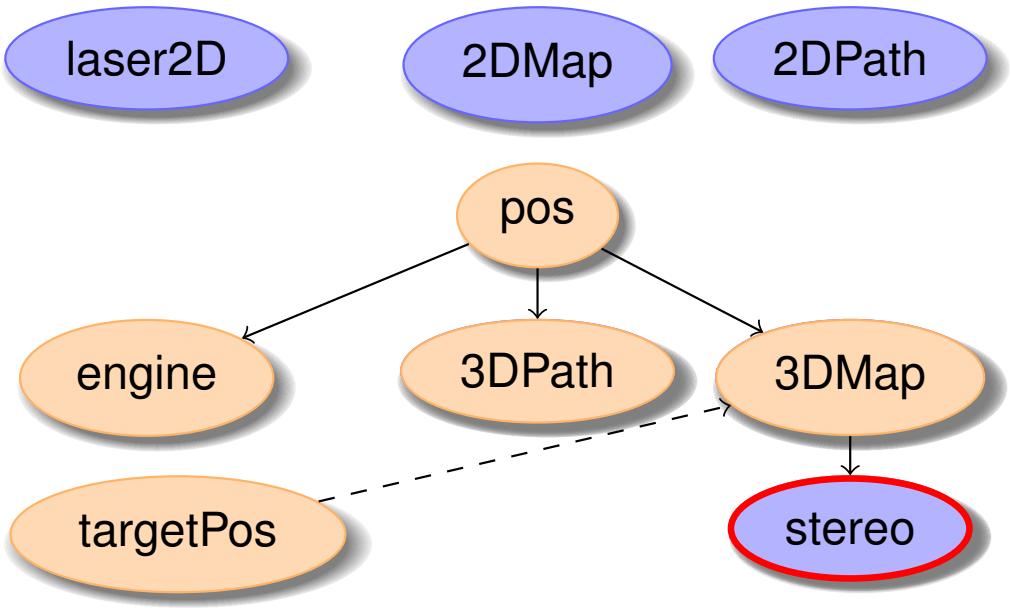
Goto (X, Y), using 3D strategy

Handling concurrency issue : instantiation



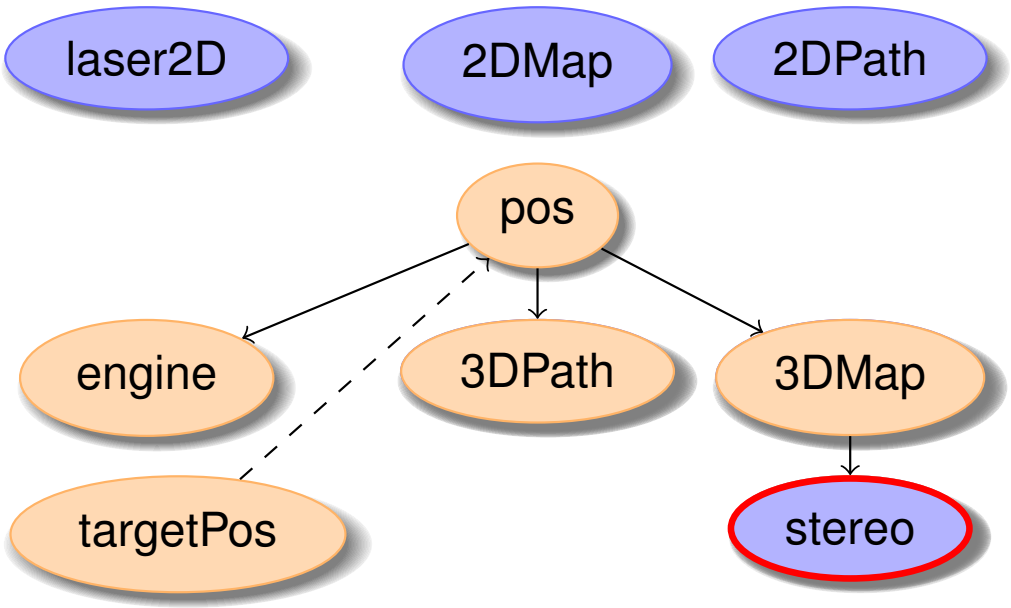
Need to track target ⇒ conflict on stereo

Handling concurrency issue : instantiation



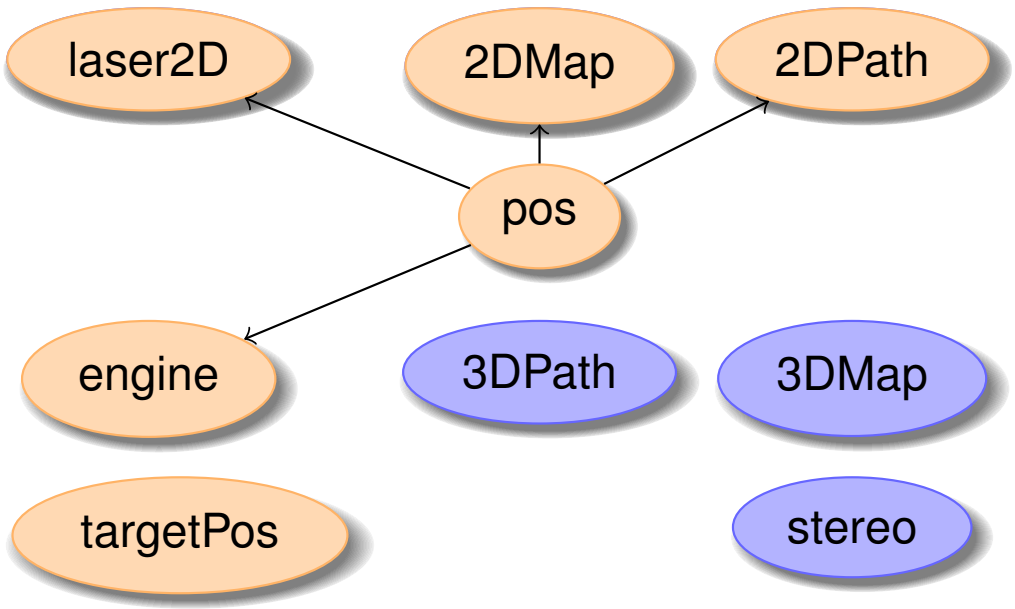
Ask 3DMap if it can handle its constraints without stereo

Handling concurrency issue : instantiation



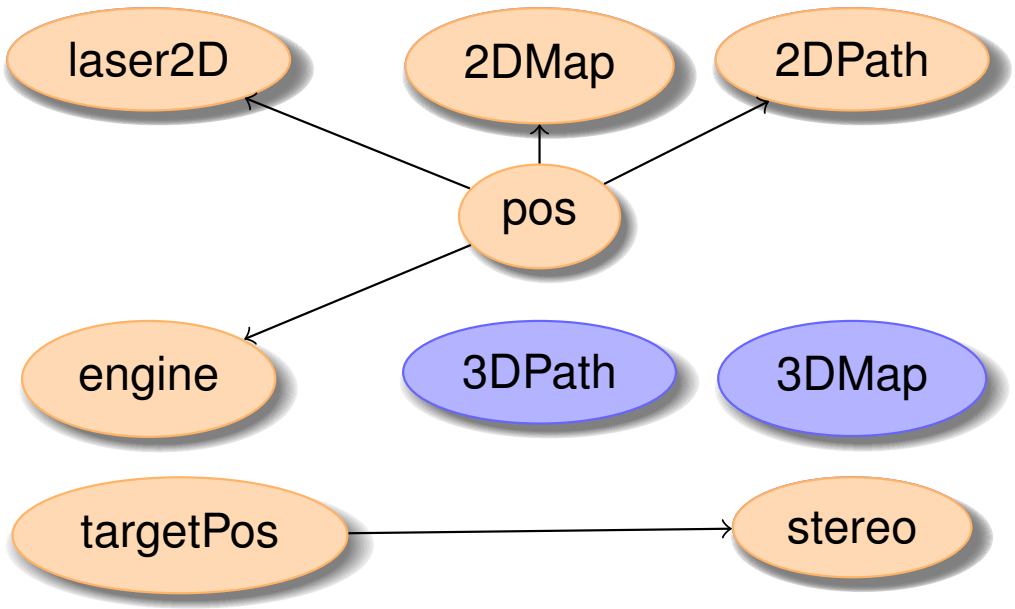
Ask pos if it can handle its constraints without 3DMap

Handling concurrency issue : instantiation



Use 2D strategy for the GoTo constraint

Handling concurrency issue : instantiation



targetPos now can use stereo

Experiments : video

Presentation Plan

Problematic

Why formalize an architecture ?

- To have a formal model to extend the language / model
- To compare in a common language with other architectures
- To integrate symbolic planners, using a common language
- To apply formal method to prove some properties

Problematic

Why formalize an architecture ?

- To have a formal model to extend the language / model
- To compare in a common language with other architectures
- To integrate symbolic planners, using a common language
- To apply formal method to prove some properties

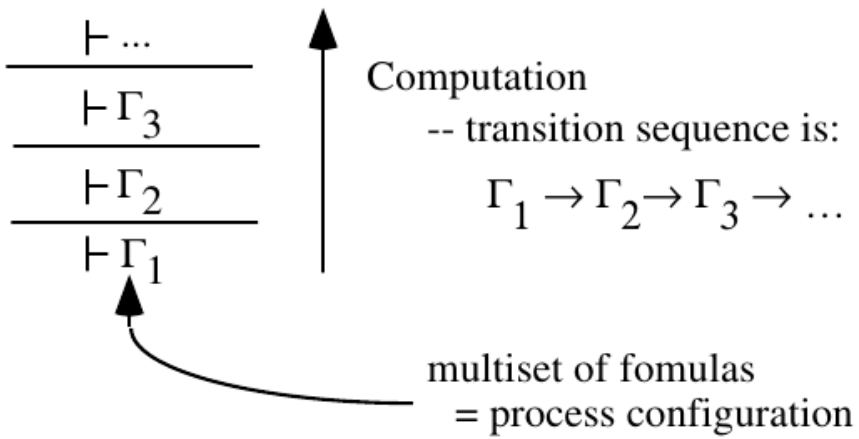
The challenge

Interleaving of

- deliberation (logic)
- concurrent execution (Petri Net, π-calcul)

A logic for Concurrent execution ?

- Curry-Howard correspondence between intuitionistic logic and λ-calcul
- Has been expanded to concurrent system with linear logic using
  - logic formulae as process (or message)
  - sequent as process configuration
  - bottom-up proof corresponds to computation



Linear logic

Classical logic is not enough to model real world computation

- Contraction : $A \Rightarrow A \wedge A$
- Weakening : $A \wedge B \Rightarrow A$

Linear logic

Classical logic is not enough to model real world computation

- Contraction : $A \Rightarrow A \wedge A$
- Weakening : $A \wedge B \Rightarrow A$

Linear logic to the rescue

- Use $\multimap$ to imply and consume

$$H_2 \otimes H_2 \otimes O_2 \multimap H_2O \otimes H_2O$$

- Allows to specify states
- The operator ! to denotes “infinite” resource

Back to our problem (1)

- A, B represents processes or states
- $A \otimes B$ represents parallels processes
- $A \,\&\, B$ represents choices (A or B will be executed)
- $A \multimap B$ represents a transformation of A into B
- $\exists c.A$ creates fresh values

Back to our problem (2)

Using it

- Formalise each primitive of the ROAR language using linear logic
- Formalise recipe selection using the default strategy
- Full logic-model for execution

An example : The make translation

$$
\begin{aligned}
&\mathit{eval\_make} : \mathit{eval}\ ((\mathit{make}\ E_1\ C)\ D) \\
&\qquad \multimap \exists Ch_1 : \mathit{Chan}\ S.\ \exists d_1 :\ \mathit{dest}(\mathit{void}). \\
&\qquad\quad \mathit{eval}\ (\mathit{writeMsg}\ (E1\ (C, Ch_1))\ d_1) \\
&\qquad\qquad \otimes (\Pi V1 : \mathit{void}.\mathit{return}\ (V_1\ d_1) \\
&\qquad\qquad\quad \multimap \mathit{eval}\ ((\mathit{readMsg}\ Ch_1)\ D)).
\end{aligned}
$$

LAAS-CNRS

Presentation Plan

To sum up

ROAR main ideas

- Consider a robot as a set of resources, encapsulated by agents
- A logic engine to deal with concurrency issues
- Backtracking with history to handle failure
- Coordination between agents to improve global efficiency
- A complete model for ROAR based on linear logic

To sum up

ROAR main ideas

- Consider a robot as a set of resources, encapsulated by agents
- A logic engine to deal with concurrency issues
- Backtracking with history to handle failure
- Coordination between agents to improve global efficiency
- A complete model for ROAR based on linear logic

Future work

- Use the proposed formalisation to valid safety properties
- Extend ROAR to multi-robot system
- Integrate at least one symbolic planner within the framework.

Questions ?

Thanks for your attention