POLITECNICO DI MILANO

Cognitive Robotics – Introduction

Matteo Matteucci – matteo.matteucci@polimi.it

About me and my lectures …

Lectures given by Matteo Matteucci

- +39 02 2399 3470
- matteo.matteucci@polimi.it
- http://www.deib.polimi.it/people/matteucci

Research Topics (several Thesis available)

- Robotics and Autonomous Systems
- Computer Vision and Perception
- Pattern Recognition & Machine Learning
- Benchmarking in Robotics

Aims of the lectures: learning how to design and implement the software which makes autonomous an autonomous mobile robot (e.g., symbolic planning, trajectory planning, localization, perception, mapping, etc.)

Lectures Outline

Middleware in robotics
- Motivations and state of the art
- A case study: ROS

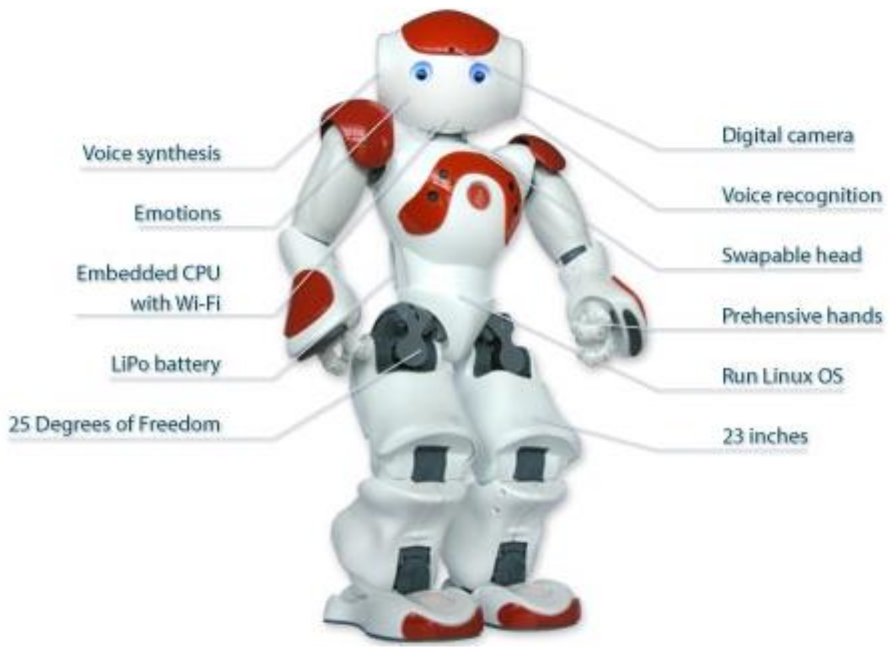
Symbolic Planning
- The PDDL language
- PDDL Extensions

Path planning
- Path planning in ROS
- SPBL vs OMPL

Localization and Mapping
- Localization vs Mapping
- Simultaneous Localization and Mapping
- 2D Navigation and localization in ROS

Object recognition
- 3D object recognition with RGBD cameras (kinect)

The NAO Case Study



POLITECNICO DI MILANO

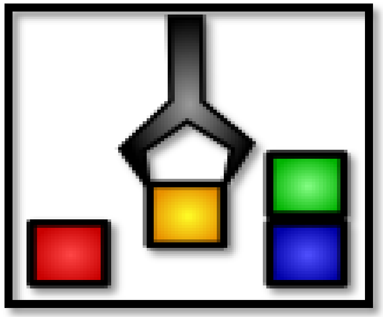
Cognitive Robotics – PDDL

Matteo Matteucci – matteo.matteucci@polimi.it

Planning Problems in Artificial Intelligence

Planning Problem := <P,A,S,G>

- P:= a SET of Predicates
- A:= a SET of Operators (Actions)
- S:= initial State
- G:= Goal(s)



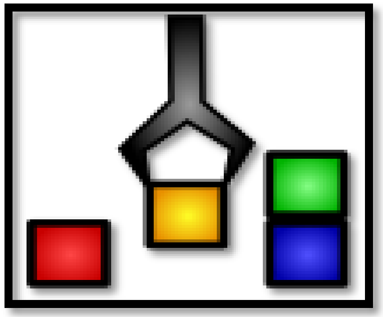
A Plan Domain or Domain Theory is defined as := P + A

A Problem Solution or Plan is := a sequence of Actions that

- *if executed* frm the initial state S
- *will result* in a state satisfying the Goal

STRIPS as a Language

STRIPS was a planner developed at SRI in 1971, now it has been used as formal language for Planning Problems



- list of Predicates: *atomic formulae*
- list of Actions:
  - *NAME: string*
  - PRECONDITIONS: *PartiallySpecifiedState*
  - EFFECTS: *ADDlist, DELETElist*
  - + *"STRIPS assumption"*
- Initial State: *State*
- Goal: *PartiallySpecifiedState*

"A State S satisfies a PartiallySpecifiedState G if S contains all the atoms of G"

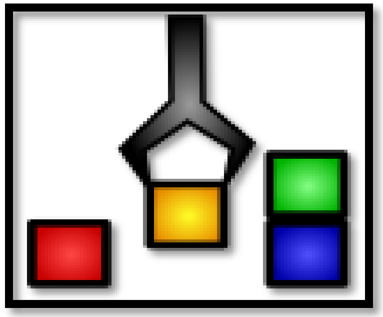
According to the previous

- Atomic formula (atom):= predicate + arguments
- State:= set of **positive** atoms + **CWA!**
- PartiallySpecifiedState:= set of **positive** atoms

The Block World in STRIPS

List of Predicates

- *empty*: the gripper is not holding a block
- *holding(B)*: the gripper is holding block B
- *on(B1,B2)*: block B1 is on top of block B2
- *ontable(B)*: block B is on the table
- *clear(B)*: block B has no blocks on top of it and is not being held by the gripper



List of actions

Action	**Preconditions**	**Add List**	**Delete List**
unstack(B1, B2)	empty & clear(B1) & on(B1, B2)	holding(B1), clear(B2)	empty, on(B1, B2), clear(B1)
pickup(B)	empty & clear(B) & ontable(B)	holding(B)	empty, ontable(B), clear(B)
stack(B1, B2)	holding(B1) & clear(B2)	empty, on(B1, B2), clear(B1)	clear(B2), holding(B1)
putdown(B)	holding(B)	empty, ontable(B), clear(B)	holding(B)

PDDL: Planning Domain Definition Language

PDDL (Planning Domain Definition Language) is a standard encoding language for “classical” planning tasks

- *Objects*: Things in the world that interest us
- *Predicates*: Properties of objects that we are interested in (*true/false*).
- *Initial state*: The state of the world that we start in.
- *Goal specification*: Things that we want to be true.
- *Actions/Operators*: Ways of changing the state of the world.

Planning tasks specified in PDDL are separated into two files

- A *domain file* for predicates and actions
- A *problem file* for objects, initial state and goal specification

PDDL was invented in 1998 for the first IPC and nowadays most common planners read PDDL files …

PDDL: Domain files

```
(define (domain <DOMAIN_NAME>)
    (:requirements :strips )
    (:predicates  (<PREDICATE_1_NAME> ?<arg1> ?<arg2> ...)
                  (<PREDICATE_2_NAME> ...)
                  ...)
    (:action <ACTION_1_NAME>
          :parameters (?<par1> ?<par2> ...)
          :precondition <COND_FORMULA: PartiallySpecifiedState>
          :effect <EFFECT_FORMULA: ADDlist + DELETElist>
    )
    (:action <ACTION_2_NAME>
          ...)
    ...)
```

PDDL: Problem Files

(define (problem <PROBLEM_NAME>)

    (:domain <DOMAIN_NAME>)

    (:objects <obj$_1$> <obj$_2$> ...)

    (:init <ATOM$_1$> <ATOM$_2$> ...)

    (:goal <*COND_FORMULA:* *PartiallySpecifiedState*>)

)

Where we have:

- Init and Goal are *ground*! (not parameterised, i.e., not ?x kind of things)
- *COND_FORMULA:* conjunction of atoms

  (AND atom$_1$ ... atom$_n$)
- *EFFECT_FORMULA:* conjunction of ADDED & DELETED (NOT) atoms

  (AND atom$_1$ ... (NOT atom$_n$))

PDDL 1.2 (IPC 2000)

In successive revisions of the language requirements where added:

• :strips	
• :typing	*in :predicates, :parameters and :objects*
• :equality	**=**
• :negativepreconditions	**not**
• :disjunctivepreconditions	**or**
• :existentialpreconditions	**exists**
• :universalpreconditions	**forall**
• :quantifiedpreconditions	= :existentialpreconditions + :universalpreconditions
• :conditionaleffects	**when**
• :adl = *all the above (Action Description Language)*	

PDDL: Typing in Domain and Problem Files

(define (domain <DOMAIN_NAME>)

(:requirements :strips :typing)

(:types <type$_1$> <type$_2$> ...)

(:predicates (<PREDICATE_1_NAME> ?<arg$_1$> - <type$_1$> ...)

(<PREDICATE_2_NAME> ...))

(:action <ACTION_1_NAME>

:parameters (?<par$_1$> - <type$_1$> ?<par$_2$> - <type$_2$> ...)

:precondition <*COND_FORMULA:* *PartiallySpecifiedState*>

:effect <*EFFECT_FORMULA:* *ADDlist* + *DELETElist*>)

…)

(define (problem <PROBLEM_NAME>)

(:domain <DOMAIN_NAME>)

(:objects <obj$_1$> - <type$_1$> <obj$_2$> - <type$_2$> ...)

(:init <ATOM$_1$> <ATOM$_2$> ...)

(:goal <*COND_FORMULA:* *PartiallySpecifiedState* >)

)

STRIPS vs ADL Conditional Formulas

The *:requirement* clause defines the power of the language that should be understood by the planner

- :strips
  - Conjunction of atoms (AND atom_1 ... atom_n)
  - If :equality added atoms my be in the form (= arg_1 arg_2)
  - Only positive
- :adl
  - equality (=) (= arg_1 arg_2)
  - negation (NOT) (NOT atom_1)
  - conjunction (AND) (AND atom_1 ... atom_n)
  - disjunction (OR) (OR atom_1 ... atom_n)
  - quantifier (FORALL, EXISTS)

(FORALL (?v - t) (*PREDICATE* ?v))

(EXISTS (?v - t) (*PREDICATE* ?v))

STRIPS vs ADL Effect Formulas

The *:requirement* clause defines the power of the language that should be understood by the planner

- :strips
  - Conjunction of added and deleted atoms (AND atom$_1$ ... (NOT atom$_n$))
- :adl
  - Conditional effect:

    (WHEN *PRECOND_FORMULA* EFFECT_FORMULA)
  - Universal quantified formula:

    (FORALL (?<v$_1$> - <t$_1$> ?<v$_2$> - <t$_2$>) *EFFECT_FORMULA*)

Basic PDDL Example: Gripper Domain

Gripper task with four balls:

There is a robot that can move between two rooms and pick up or drop balls with either of his two arms. Initially, all balls and the robot are in the first room. We want the balls to be in the second room.

- Objects: The two rooms, four balls and two robot arms.
- Predicates: Is x a room? Is x a ball? Is ball x inside room y? Is robot arm x empty? [...]
- Initial state: All balls and the robot are in the first room. All robot arms are empty. [...]
- Goal specification All balls must be in the second room.
- Actions/Operators: The robot can move between rooms, pick up a ball or drop a ball.

Gripper Domain: Objects

Objects in the gripper domain

- Rooms: rooma, roomb
- Balls: ball1, ball2, ball3, ball4
- Robot arms: left, right

In PDDL without typing

- (:objects rooma roomb ball1 ball2 ball3 ball4 left right)

In PDDL with typing

- (:types room ball robot-arm)
- (:objects rooma – room roomb – room
  ball1 – ball ball2 – ball ball3 – ball ball4 – ball
  left – robot-arm right – robot-arm)

Gripper Domain: Predicates (without typing)

Predicates in the gripper domain without typing

- ROOM(x) – true iff x is a room
- BALL(x) – true iff x is a ball
- GRIPPER(x) – true iff x is a gripper (robot arm)
- at-robby(x) – true iff x is a room and the robot is in x
- at-ball(x, y) – true iff x is a ball, y is a room, and x is in y
- free(x) – true iff x is a gripper and x does not hold a ball
- carry(x, y) – true iff x is a gripper, y is a ball, and x holds y

In PDDL this translates into:

```
(:predicates
        (ROOM ?x) (BALL ?x) (GRIPPER ?x)
        (at-robby ?x) (at-ball ?x ?y)
        (free ?x) (carry ?x ?y)
)
```

Gripper Domain: Predicates (with typing)

Predicates in the gripper domain with typing

- at-robby(x) – true iff x is a room and the robot is in x
- at-ball(x, y) – true iff x is a ball, y is a room, and x is in y
- free(x) – true iff x is a gripper and x does not hold a ball
- carry(x, y) – true iff x is a gripper, y is a ball, and x holds y

In PDDL this translates into:

- (:predicates

```
            (at-robby ?x – room)
            (at-ball ?x – balll ?y – room)
            (free ?x – robot-arm)
            (carry ?x – robot-arm ?y – ball)
    )
```

Gripper Domain: Initial State

The Initial state (according to the example text):

- ROOM(rooma) and ROOM(roomb) are true.
- BALL(ball1), ..., BALL(ball4) are true.
- GRIPPER(left), GRIPPER(right), free(left) and free(right) are true.
- at-robby(rooma), at-ball(ball1, rooma), ..., at-ball(ball4, rooma) are true.
- Everything else is false.

In PDDL this translate into:

-
  ```
  (:init
              (ROOM rooma) (ROOM roomb)
              (BALL ball1) (BALL ball2) (BALL ball3) (BALL ball4)
              (GRIPPER left) (GRIPPER right) (free left) (free right)
              (at-robby rooma) (at-ball ball1 rooma) (at-ball ball2 rooma)
              (at-ball ball3 rooma) (at-ball ball4 rooma)
  )
  ```

Gripper Domain: Goal State

The Goal state (according to the example text):

- at-ball(ball1, roomb), ..., at-ball(ball4, roomb) must be true.
- Everything else we don't care about.

In PDDL this translates into:

-
  ```
  (:goal
              (and (at-ball ball1 roomb)
                        (at-ball ball2 roomb)
                        (at-ball ball3 roomb)
                        (at-ball ball4 roomb)
              )
  )
  ```

Gripper Domain: Movement Operator

The robot can move from x to y:

- Precondition: ROOM(x), ROOM(y) and at-robby(x) are true.
- Effect: at-robby(y) becomes true and at-robby(x) becomes false.
- Everything else doesn't change.

In PDDL this translates into:

-
  ```
  (:action move
           :parameters (?x ?y)
           :precondition (and (ROOM ?x) (ROOM ?y) (at-robby ?x))
           :effect (and (at-robby ?y) (not (at-robby ?x)))
  )
  ```

Gripper Domain: Pick-up Operator

The robot can pick up x in y with z.

- Precondition: BALL(x), ROOM(y), GRIPPER(z), at-ball(x, y), at-robby(y) and free(z) are true.
- Effect: carry(z, x) becomes true while at-ball(x, y) and free(z) become false.
- Everything else doesn’t change.

In PDDL this translates into:

```
• (:action pick-up :parameters (?x ?y ?z)
           :precondition (and (BALL ?x) (ROOM ?y) (GRIPPER ?z)
                              (at-ball ?x ?y) (at-robby ?y) (free ?z))
           :effect (and (carry ?z ?x) (not (at-ball ?x ?y)) (not (free ?z)))
  )
```

Gripper Domain: Drop Operator

The robot can drop x in y from z

- Precondition: BALL(x), ROOM(y), GRIPPER(z), carry(z,x), at-robby(y) are true.
- Effect: at-ball(x, y) and free(z) become true while carry(z, x) becomes false.
- Everything else doesn't change.

In PDDL this translates into:

-
  ```
  (:action drop :parameters (?x ?y ?z)
               :precondition (and (BALL ?x) (ROOM ?y) (GRIPPER ?z)
                                  (carry ?z ?x) (at-robby ?y))
               :effect (and (at-ball ?x ?y) (free ?z) (not (carry ?z ?x)))
  )
  ```

Let’s solve it!

Using satplan to solve the gripper problem

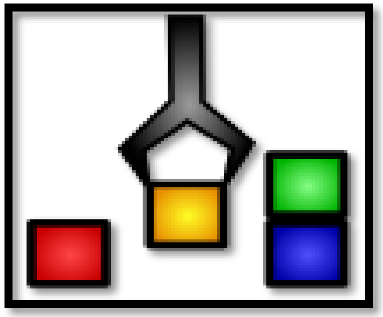
- Download satplan (2006 version, winner of IPC)
  - http://www.cs.rochester.edu/users/faculty/kautz/satplan/index.htm
  - tar -zxvf SatPlan2006.tgz

- Compile satplan by issuing
  - cd SatPlan2006
  - make
- Run vanilla satplan (i.e., default options)
  - cd include/bin/
  - ./satplan -path ../../gripper/ -domain gripper_domain.pddl -problem gripper_problem.pddl
- Observe the plan
  - less gripper_problem.pddl.soln

Exercise: The Block World in PDDL!

List of Predicates

- *empty*: the gripper is not holding a block
- *holding(B)*: the gripper is holding block B
- *on(B1,B2)*: block B1 is on top of block B2
- *ontable(B)*: block B is on the table
- *clear(B)*: block B has no blocks on top of it and is not being held by the gripper



List of actions

Action	**Preconditions**	**Add List**	**Delete List**
unstack(B1, B2)	empty & clear(B1) & on(B1, B2)	holding(B1), clear(B2)	empty, on(B1, B2), clear(B1)
pickup(B)	empty & clear(B) & ontable(B)	holding(B)	empty, ontable(B), clear(B)
stack(B1, B2)	holding(B1) & clear(B2)	empty, on(B1, B2), clear(B1)	clear(B2), holding(B1)
putdown(B)	holding(B)	empty, ontable(B), clear(B)	holding(B)

PDDL 2.1: Time (the idea)

A feasible plan is sometimes not enough, thus a new version of planner was introduced to take into account:

- Durative actions: time
- Fluents: numbers
- Metrics: optimal plan

Time in planning (scheduling)

- actions take time to execute—how long an action takes to execute may depend on the preconditions
- preconditions may need to hold when the action begins, or throughout the execution of the action
- effects may not be true immediately and their effects may persist for only a limited time—an action can have multiple effects on a fluent at different times

PDDL 2.1: Time (the code)

A feasible plan is sometimes not enough, thus a new version of planner was introduced to take into account:

- Durative actions: time
- Fluents: numbers
- Metrics: optimal plan

In the Domain file

- (:durative-action <name>

  :parametes (…)

  :duration (= ?duration <time>)

  :condition (…)

  :effect (…))
- CONDITIONAL_FORMULA: at_start, overall, at_end
- EFFECT_FORMULA: at_start, at_end

PDDL 2.1: Resources (the idea)

A feasible plan is sometimes not enough, thus a new version of planner was introduced to take into account:

- Durative actions: time
- Fluents: numbers
- Metrics: optimal plan

Resources in planning

- A resource is any quantity or (set of) object(s) whose value or availability determines whether an action can be executed
- Resources may be consumable (examples: money, fuel) or reusable (example: a car which becomes available again after a trip)
- In some cases, actions may produce resources (examples: refueling, hiring more staff, etc)
- When planning with resources, a solution is defined as a plan that achieves the goals while allocating resources to actions so that all resource constraints are satisfied

PDDL 2.1: Resources (the code)

A feasible plan is sometimes not enough, thus a new version of planner was introduced to take into account:

- Durative actions: time
- Fluents: numbers ←
- Metrics: optimal plan

In the Domain definition

- (:functions (<name1> ?<obj1> - <type1>)
  (<name2> ?<obj2> - <type2>)
  (…))
- CONDITIONAL FORUMULA: = > < <= => + - * /
- EFFECT FORMULA:
  - assign, increase, decrease, scale-up, scale-down

In the Problem definition

- (:init (= (<ATOM>) <#>))

PDDL 2.1: Metrics (the idea)

A feasible plan is sometimes not enough, thus a new version of planner was introduced to take into account:

- Durative actions: time
- Fluents: numbers
- Metrics: optimal plan

Optimal planning (and scheduling)

- As with search problems, we can distinguish between optimal and satisficing solutions
- A satisficing plan is one that achieves the goal(s) without violating any temporal or resource constraints
- An optimal plan is one that achieves the goal(s) while minimising (or maximising) some metric—metric is often defined in terms of resource usage

PDDL 2.1: Metrics (the code)

A feasible plan is sometimes not enough, thus a new version of planner was introduced to take into account:

- Durative actions: time
- Fluents: numbers
- Metrics: optimal plan

In the problem definition

- (:metric minimize[maximize] <objective_function>)

Built-in function:

- total-time

PDDL 2.1 – IPC-2002

“We have a Four Gallon Jug of Water and a Three Gallon Jug of Water and a Water Pump. The challenge of the problem is to be able to put exactly two gallons of water in the Four Gallon Jug, even though there are no markings on the Jugs.”

Drew McDermott. “The 1998 AI Planning Systems Competition”. AI Magazine (21):2, 2000.

Also know from Die Hard 3 Movie!

POLITECNICO DI MILANO

Cognitive Robotics – PDDL

Matteo Matteucci – matteo.matteucci@polimi.it