Robot Platform Motion Planning using Answer Set Programming

Julian J. Portillo, Carmen L. Garcia-Mata, Pedro R. Márquez-Gutiérrez, and Rogelio Baray-Arana A.

Instituto Tecnológico de Chihuahua, Chihuahua, Chih., Mex.,
{jjportillo,clgarcia,pmarquez,rbaray}@itchihuahua.edu.mx

Abstract. We discuss how a path planning problem for a robotic platform was solved modeling it with the axioms of the action language $\mathcal{AL}$ and described in Answer Set Programming, ASP. The environment of the autonomous vehicle is static. A set of dynamic, static and inertial laws are used to describe the vehicle domain. The optimum solution is chosen using two criteria: minimum distance and minimum obstacles dodged required to arrive to the goal. One advantage in using ASP for planning problems is related with its complexity. In case of classical planning problems it is known they are PSPACE-complete for finite domains and undecidable in the general case. By fixing the plan length, the planning problem is reduced to NP-complete. Since Answer Set Programming is a totally declarative language, the problem solution relies in the ASP's solver, which is NP-space, as it has been already proved elsewhere. Consequently, for solvable problems, the solutions are always found.
Keywords: Planning, Answer Set Programming, Action Languages.

1 Introduction

There is a wide spectrum of applications where domestic and service robots are required, like medical and manufacturing applications [14], [5]. In contrast to popular idea of humanoid robots, promoted by media, real-world applications happen in closed spaces and generally use fixed robots or robots mounted on a platform.

A fundamental problem of a mobile robot is to find a collision-free path to its target, while minimizing the total cost. Approaches to robot motion planning can be roughly divided into two categories: the classical motion planning or model-based motion planning, and sensor-based planning. These categories also reflect whether the path planning is global (model-based motion planning) or local (sensor-based planning). The first approach assumes that the robot system has an explicit representation of the environment. On the other hand, in the second approach, the environment is unknown and the robot is driven directly by the sensors input without building an internal representation of the environment. Examples of global path planning methods are genetic algorithms and neural networks based methods, [4] and [5]. However, most of popular local path planning approaches are potential field and bug algorithms [6].

Typically, a path planning problem is modeled taking into account the configuration space (C) formed from the free configuration space (F) and the Obstacle Space (O). F is the subset of C in which the robot does not intersect any obstacle. The dimension of C depends on the degrees of freedom (dof) of the robot. A robot working in 3D environments has at least 6 degrees of freedom. Three are used to represent the robot's position and the other 3 are used for the robot's orientation. For robots with high dofs there are no algorithms capable of compute an exact solution to the problem and approximate solutions are used instead. For this reason, a large family of model-based planners has been developed through the years, [8].

Main issues to be considered for robot path planning are related to the computational complexity, local optimum and low adaptability to the environment. Besides, there are other aspects that must be taken into account such as the robot physical characteristics, the application type and the robot's environment in order to choose the best methodology to model and solve the planning problem.

In spite of our path planning problem is situated in a static environment, our next research stage will deal with a dynamic environment and our robot will have sensing capability to adjust the plan every time will be needed according to new sensing information arrives. Based on this considerations, it was evident we should select an appropriate formalism capable of reasoning about action effects and world changing.

The axiomatization of domains through action theories add expressiveness and elegance to the problem description. More importantly, building the planner on formal validation methods let us be confident that the encoding of any domain will yield correct results. In the last years, several formalisms oriented to dynamic worlds have been designed [13]. Action Languages [7] are considered one of the more promising formalisms and, in particular, the action language $\mathcal{AL}$ shows important advantages over other action languages, basically because it axiomatizes the frame and qualification problems [15].

At the initial research stage of our work, we have only considered the model-based motion planning problem. The focus of this research is to solve the path planning problem for a robotic platform working in a static environment. The problem is modeled using the formalism of the action language $\mathcal{AL}$ and coding in Answer Set Programming. The answer sets or solutions are found by Clingo, which is the most efficient solver nowadays as the results of the last ASP solvers' competition reported [16].

2 Issues on the Robot Platform Motion Planning

Robotic systems of this type normally include a hierarchy of behaviors. A generic two-layer hybrid architecture was chosen for our robot platform. Top layer deals with highest cognitive processes for world modeling and planning. The bottom layer integrates a set of device drivers to control the robot sensors and actuators. Low level device drivers were developed in the C language.

The focus of the current research was on the top layer without interacting with the bottom software layer or hardware. Currently we are working on the integration of both layers. Once this integration be realized, the planner will be capable of solving the planning problem interacting concurrently with the bottom layer doing the plan execution and the planner must be capable of adapting the original plan whenever is needed, based on the perception system readings.

3 Action Theories and Planning Languages for Complex Environments

While studying changing world dynamics, AI research has developed a number of techniques ranging from simple ones like space search, to procedural, probabilistic and deductive ones. Automating reasoning about common sense knowledge implies, among other things, reasoning about actions and environmental changes.

Attempts in solving challenges like the knowledge-action relationship has had a deep influence on current directions taken by the AI community. There exists, nowadays, a number of formalisms to reason about knowledge which allow an agent to adapt its beliefs based on the environment changes he observe. However, these models do not involve reasoning about action properties causing these changes. For dynamic environments it is necessary to have a set of action theories which take into account the environment state and its change, besides a notion of causality of potential action side effects, action preconditions, and relationships between fluents and side effects of actions. Therefore, it is extremely important to find a formal frame gathering together action theories in order to model real world applications requiring knowledge acquisition, its storage in a knowledge base and that could realize actions in real-time [2].

In the reasoning about action and change arena (RAC), there exist significant advances and powerful theories that have been designed to express formally knowledge and changes caused by actions. Among most relevant models axiomatizing the RAC problem we find STRIPS, Situational Calculus, Fluent Calculus, Event Calculus [10] and Action Languages. Extensions of these models have developed several planning languages like Golog derived from Situational Calculus, Flux derived from Fluent Calculus, Language E derived from Event Calculus and several action languages like $\mathcal{A}$, $\mathcal{B}$, $\mathcal{C}$ and more recently, $\mathcal{AL}$ [13].

These models and extensions differ from each other in several aspects, like time management and concurrency, no determinism and uncertainty, action derived knowledge, and others. Since world knowledge is not completely certain and sometimes contradictory to previous knowledge, it is asked these models be able to reason using beliefs instead of knowledge. Both Situational Calculus and Fluent Calculus provide extensions which reasons using beliefs. However, these extensions do not provide solutions to the similar frame problem in the context of beliefs. Instead, the action language $\mathcal{AL}$ axiomatizes both the frame problem and the branching problem.

In dynamic environments we have to consider not only the action effects but also the relationship between fluents. For instance, in a robot path planning

problem we could find the following relations between fluents: a) the robot cannot be in A if it is in B, and b) the robot is close to C if it is in B. Moreover, the static law (b) implies that it is true that the robot is close to C since it is an indirect effect of moving the robot from A to B. This side effect is an instance of the branching problem.

In spite of the solutions proposed for the branching problem and several development frames, it is uncommon in practice to use static causal laws in planning. The Planning Domain Description Language (PDDL) allows to implement static causal laws or axioms, but it is not common to use them in designing a planner since PDDL axioms do not allow recursion, and this limits its use.

4 Answer Set Programming in a Nutshell

A problem can be encoded as a set of facts, rules and constraints that describes the (candidate) solutions. An ASP-program is a collection of rules of the form

$$a_o \Leftarrow a_1, \ldots, a_n \; not \; a_m, \ldots, a_{m+k} \tag{1}$$

where each a_i is an atom. The left hand side and the right hand side of the clause are called head and body, respectively. The head of a rule is a positive literal and its body is composed of literals (a literal is an atom a or its negation, denoted by not a). A rule without body is a fact. A rule without head is a constraint. Also, the rules can be positive ($n > 0$); negative ($m > 0$) or both ($n > 0$ and $m > 0$). The symbol not stands for default negation, also known as negation as failure.

If P is a ground, positive program, a unique answer set is defined as the smallest set of literals constructed from the atoms occurring in program P. The last definition can be extended to any ground program P containing negation by considering the reduct of P w.r.t. a set of atoms X obtained by the Gelfond-Lifshitz's operator [6].

Once a program is described as an ASP-program P, its solutions, if any, are represented by the answer set of P. One important difference between ASP semantics and other semantics is that a logic program may have several answer sets or may have no answer set at all.

Answer Set Programming is a totally declarative language. ASP programs are not algorithms describing how to solve the problem; the program is just a formal description of the problem. The solution is completely found by the solver. An ASP solver requires grounded programs as input, and that is why before searching the answer set or solutions, the program is grounded by a preprocessor. Actually there are many ASP's solvers.

5 The Path Planning Problem

The implementation of a planner for some dynamic domain starts with encoding of fluents and actions about the domain as an action theory X of some action

language. In the last years, numerous action languages have been developed. Our robot's domain was modeled like axioms in the action language $\mathcal{AL}$, which is basically an extension of the action language A. The $\mathcal{AL}$ language allows reasoning about action and change, through axiomatizing the qualification and ramification problems. The ramification problem basically consists in determine the effects of indirect actions. For instance, if the robot is moved from node 2 to node 8, at the end the robot is in node 8 [12].

The path planning problem was modeled based on the formalisms of the action language $\mathcal{AL}$ through a domain description containing the initial state and a set of action laws, equivalently, it can be said that the domain is represented by a transition diagram whose nodes are the possible states of the domain and whose arcs are actions that take the domain from one state to another. Paths in the diagram correspond directly to possible trajectories of the robot.

In this first approximation to the solution of the path planning problem, the problem was simplified assuming a) that the robot will be situated in a static environment and, b) the only goal of the planner is to take the robot from an initial point to a goal point throughout a certain number of obstacles.

The work area is represented as a 2D space. Each obstacle is a closed polygon. The polygon's vertexes are numbered. This way, an obstacle is defined in a map by its vertexes coordinates and the relations with the other obstacles.

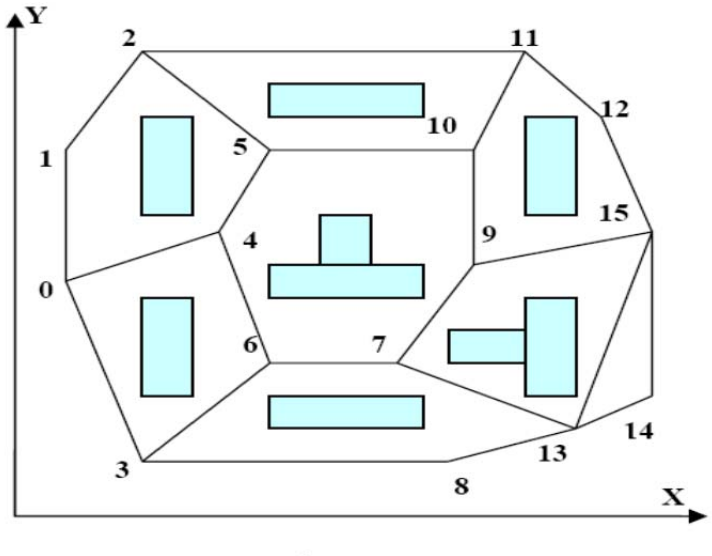

Fig. 1. *Workspace map used in the example*

In calculating a path there are certain physical aspects that could be considered as a constraint problem. Two important restrictions that must be taken

in account are the actual robot position and the possible move that the robot can execute at the next step. These valid movements are directly related to the current robot position (robot state or node) since at each step the robot can only move to nodes directly linked to the one it currently is on.

Figure 1 shows the workspace a map used as an example.

Information about obstacles consists of their positions and connections between obstacles. The following code fragment shows how the initial state is represented in $\mathcal{AL}$.

```
pt(0;1;2;3;4;5;6;7;8;9;10;11;12;13;14;15;str;gol).
link(0,1;3;4).

%%  node coordinates.
loc(node, x, y)
```

Fluents represent properties of objects that could change during the solution evolution. In our example, fluents required are the robot position and coordinates to which the robot can move.

```
fluent(on(r,X)) :- pt(X).
holds(on(r,str),0).
holds(neg(F),0):-fluent(F), not holds(F, 0).
```

In predicate on(r,X), r represents the robot and X represents the node where the robot is currently positioned. The initial robot position is given by holds(on(r, str), 0). The second holds rule says that if it is not possible to know whether a fluent holds for an initial time 0, then the fluent must be negated, ergo, there are not reasons to believe that fluent F holds at time zero.

In this domain just a single action is required: the motion from some point to another. This action is defined by the rule

```
action(move(X,Y)) :- pt(X;Y), link(X,Y).
```

Static laws (or axioms) constitute an important part of every dynamic domain. Unlike an effect of an action, a static causal law represents a relationship between fluents. For instance, the following rule stands that at time T the robot cannot be at Y point if the robot is at Z or equivalently, at time T is false that the robot is at Y if it is true that the robot is at Z.

```
holds(neg(on(r,Y)), T) :- time(T), pt(Y;Z), holds(on(r,Z), T),
Y != Z.
```

The valid moves are established using the enforce law described below

```
possible((move(X,Y)),T):-time(T),pt(X;Y),holds(on(r,X),T), not
hold(on(r,Y), T-1)
```

This rule says that the robot can goes from point X to point Y at time T if the robot is at point X at time T and the time immediately before T, (T-1), the robot has not been at point Y, in other words, it is not allowed the robot chooses a point previously visited. This rule avoids a robot left trapped in a cyclic path.

Solutions with movements no considered by the possibility law are eliminated using the following restriction

```
:- action(A1), time(T), occ(A1, T), not posible(A1,T).
```

Because the robot is situated in a dynamic environment, each time the robot moves from one point to another, the world state changes. In action languages like $\mathcal{AL}$, these changes are defined by dynamic laws. These laws express the effects of actions. Since only one action was defined in this problem, the unique effect of this action is that the robot will be in a different place each time that a movement occurs.

```
holds(on(r,X),T+1):-time(T),pt(X;Y),holds(on(r,Y),T),
                    link(Y,X), occ(move(Y,X),T).
```

The commonsense law of inertia stands that in absence of information, all properties of the world can be assumed to remain as they were in the past. The idea behind commonsense inertia is known as the frame problem. The inertial law is defined by the following rule:

```
holds(on(r,X), T+1):-pt(X), time(T), holds(on(r,X), T),
                     not holds(neg(on(r,X)), T+1).
```

The interpretation of this law is if it is not known that the robot has not moved from the point X at time T therefore, at time T+1 the robot must be at point X.

Only one action is allowed each time and the time line must not be larger than time n, defined by the user. This conditional rule is written as

```
1 {occ(A1,T) : action(A1) }1:- time(T), not holds(on(r,gol),T).
```

The plan is finished once the robot arrives to the goal state in the specified time. And all solutions not satisfying the time criteria must be eliminated.

```
arrive :- holds(on(r,gol),T), time(T).
:- not arrive.
```

Additionally, since the planner's goal is to find a path optimized for some parameters, the most convenient solution is chosen among the set of solutions using some optimization criteria.

In our case the optimization criteria are two: a) minimum distance and, b) minimum steps required to go from the initial state to the goal state. Minimum distance is calculated based on point coordinates. Both, minimum distance and minimum steps are passed as the optimization criteria. Distance is prioritized over minimum steps, but both are considered. The following code fragment details all of this:

```
dis(T,DX,DY,DT):-at(on(r,P1),T), at(on(r,P2),T+1),
loc(P1,X1,Y1),loc(P2,X2,Y2),DX:=#abs(-(X1-X2)),
DY:=#abs(-(Y1-Y2)),DT:=DX+DY.
#minimize [ dis(T,DX,DY,DT) = DT @ 1].
#minimize [ at(on(r,gol),T) = T @ 2 ].
```

6 Conclusions

Results achieved until now have shown that the formalization of action language $\mathcal{AL}$ and its representation with Answer Set Programming represent a good methodology to solve planning problems for mobile robots. Our experimental results found that for any start point and any goal point of the considered workspace, the time required to find the optimal path is in the order of microseconds with a machine running Linux and a CPU of 2.33 Mhz and 1 Gb of RAM. Obviously, different results will be achieved using a different computational platform. We are in the process of testing the planner in more complex settings. Additionally, ASP and the action language $\mathcal{AL}$ showed to be superior to probabilistic and heuristic methodologies used frequently to solve this type of planning problems. Probabilistic methodologies have the disadvantage that their solutions are not optimal, only near optimal. By another side, heuristic methodologies can be lost in local minimum, and some times the solution found is far from the optimal solution. The ASP solver is a complete algorithm and consequently the algorithm finds all the solutions, if any.

Besides, because the declarative nature of the paradigm, modularizes a program becomes straightforward. If the initial state description is separated from the domain then whenever the robot is put to work in a different world, it is not necessary to rewrite the program. In fact, the domain description remains exactly the same, and just a new description of the initial state needs to be added.

Finally, our intention is to incorporate robot capabilities to enable it operate in dynamic environments like manufacturing. In fact, choosing the action language $\mathcal{AL}$ was greatly motivated by this ultimate goal. Since the action language $\mathcal{AL}$ has been precisely designed for situations where reasoning about actions and change is needed, as in planning, we hope the $\mathcal{AL}$ language will simplify extending the program so the robot can react in the presence of unexpected events by incorporating all the extra static, inertial and dynamic laws describing the possible events the robot must face.

References

1. Balduccini, M. Gelfond, M. Nogueira and R. Watson.: An A-Prolog decision support system for the Space Shuttle. Lecture Notes in Computer Science-Proceedings of Practical Aspects of Declarative Languages'01, (2001), 1990:169-183.
2. Baral Ch.: Knowledge representation, reasoning and declarative problem solving. ambridge University Press. 2003.

3. Buniyamin N., Wan Ngah W.A.J., Sariff N. M.Z.: A Simple Local Path Planning Algorithm for Autonomous Mobile Robots. International Journal of Systems Applications, Engineering Development. Issue 2, Volume 5, 2011
4. Denecker M., Vennekens J., Bond S., Gebser M., Truszczynski.: The Second Answer Set Programming Competition. In Proceedings of LPNMR. 2009, 637-654.
5. Ereso A., Garcia P., Tseng E., Dua M., Victorino G., Guy T.S.: Usability of Robotic Platforms for Remote Surgical Teleproctoring . Telemedicine journal and ehealth the official journal of the American Telemedicine Association (2009)
6. Gelfond M., Lifschitz V.: The Stable Model Semantics for Logic Programming. Int. Conf.Logic Progr. MIT Press, 1988.
7. Gelfond M., Lifschitz V.: Action Languages. Electron. Trans. Artif. Intell., 193–210, (1998)
8. Goerzen C., Kong Z., Mettler B. A Survey of Motion Planning Algorithms from the Perspective of Autonomous UAV Guidance. Journal of Intelligent and Robotic Systems. Volume 57 Issue 1-4, (2010)
9. Loo C.-K., Rajeswari M., Wong E. K., Rao M.V.C.: Mobile Robot Path Planning Using Hybrid Genetic Algorithm and Traversability Vectors Method. Intelligent Automation and Soft Computing. Vol 10, No. 1, 51–64, (2004)
10. McDermott D. and Hendler J.: Planning: What is, What it could be. An introduction to the Special Issue on Planning and Scheduling. Artificial Intelligence, 76:1-16, (1995)
11. Nagib G., Gharibe W.: Path Planning for a Mobile Robot Using Genetic Algorithms. Proceedings of ICEEC'04, International Conference on Electrical, Electronic and Computer Engineering, ICEEC '04. 2004. pp. 185–189, (2004)
12. Tu, P.H., Son, T.C., Gelfond, M., and Morales A.R.: Approximation of action theories and its application to conformant planning. In Proceedings of Artif. Intell., 79–119, (2011)
13. Patkos T.: A Formal Theory for Reasoning about Action, Knowledge and Time. PhD Dissertation (2010), Department of Computer Science, University of Crete, Greece.
14. Saffiotti A.: Platforms for Rescue Operations. AASS Mobile Robotics Laboratory. Orebro University, Orebro, Swed
15. Son, T.C., Tu, P.H., Gelfond, M., and Morales A.R.: An Approximation of Action Theories of and Its Application to Conformant Planning. In Proceedings of LPNMR. 172–184, 2005
16. Gebser M.,Kaminski R., Kaufmann B., Ostrowski M., Schaub T., Thiele S.: A Users Guide to gringo, clasp, clingo, and iclingo. `http://www.cs.utexas.edu/~vl/teaching/lbai/clingo_guide.pdf`