Planning using Situation Calculus, Prolog and a Mobile Robot

Pilar Pozos[2], Edgardo Yescas[1], Jacob Vásquez[1],

[1]División de Estudios de Postgrado, Universidad Tecnológica de la Mixteca, Carretera a Acatlima Km. 2.5, Huajuapan de León, Oaxaca.

[2]División Académica de Informática y Sistemas, Universidad Juárez Autónoma de Tabasco, Carretera Cunduacán Km. 1, Cunduacán, Tabasco.

maripozos@gmail.com, yescas@mixteco.utm.mx, jvasquez@mixteco.utm.mx,

Abstract. This paper presents a system that controls the behavior of a mobile robot. The system is based on situation calculus, the initial state is described and a goal is given, Prolog produces an answer to this goal and we use an interface in Visual Basic to interpret the answer given by it. After of interpreting the given actions in response we send commands to the mobile robot through a serial port from a computer. This is a tool aimed at researchers and instructors in cognitive robotic.

1. Introduction

Generally a course in Artificial Intelligence (AI) can be considered interesting from a theoretical point of view. However, a practical side of this area is not commonly presented. This work tries to show a way of bringing together theory and practice. This paper presents a system which controls a mobile robot. The central part of the system uses a planner that is based on situation calculus, which has been implemented in Prolog. The resulting plan is used by a program written in Visual Basic (VB) that interprets the plan and sends the commands via the serial port to a mobile robot so that it carries out the actions needed to achieve the goal. This paper is organized in the following way: the second section begins with a brief revision of situation calculus and its use in the representation of the evolution of the world. The third section shows the design of the system. The fourth section describes the implementation, followed by the tests and results in the fifth section. Finally we conclude in the sixth section.

2. The Situation Calculus

The situation calculus is a dialect of first order logic in which situation and actions are explicitly taken to be objects in the domain. In particular, two types of first-order terms are distinguished: *actions* and *situations*. *Actions*: every change requires actions. The constant and function symbols for *actions* are completely dependent on the application. *Situations*: denote possible world histories. A distinguished constant

s0 and function symbol *do* are used; *s0* denotes the initial situation before any action has been performed; *do(a,s)* denotes the new situation that results from the performing action *a* in situation *s*.
Predicates and functions whose values may vary from situation to situation are called fluents. The last argument of a fluent is a situation. Actions typically have preconditions; that is, conditions that need to be true for the action to be executed. Actions typically have effects, that is, fluents that are changed as a result of performing the action. Effect axioms are called positive if they describe when a fluent becomes true and negative otherwise [1]. For any fluent *p* and situation *s*, *p(s)* denotes the true value of *p* in *s*. It is assumed that every change in the world is caused by an action. The evolution of the fluents is represented by "successor state axioms"; these axioms were introduced to solve the infamous frame problem [6], that is, the problem of specifying exactly what features of a scenario are not affected by an action. For a fluent *p*, the successor state axiom is

$$(S_p) \qquad p(do(a,s)) \leftrightarrow \gamma_p^+(a,s) \vee (p(s) \wedge \gamma_p^-(a,s)). \tag{1}$$

where $\gamma_p^+(a,s)$ captures exactly the conditions under which *p* turns from false to true when *a* is performed in *s*, and similarly $\gamma_p^-(a,s)$ captures exactly the conditions under which *p* turns from true to false when *a* is performed in *s*. Furthermore, in order to solve the problem of specifying precisely the conditions under which an action is executable, the "action precondition axioms" were introduced. These are used with a special fluent, *Poss(a,s)*, meaning it is possible to execute the action *A* in situation *s*. The axioms are of the form:

$$(P_A) \qquad poss(A,s) \leftrightarrow \Pi_A(s). \tag{2}$$

Where *A* is a symbol of an action and $\Pi_A(s)$ is a formula that defines the preconditions for the possibility of executing action *A i*n *s*.

2.1 Planning

The planning is one of the most useful ways that an intelligent agent can take advantage of the knowledge it has and its ability to reason about actions and their consequences. The planning produces intelligent behavior and, in particular, the use of what is known to find a course of action that will achieve some goal. Given its properties of representing dynamically changing worlds, the situation calculus is a candidate to support planning [1].
The planning task can be formulated in the language of the situation calculus as follows: Given the knowledge base (KB), and a formula, *Goal(s)*, of the situation calculus with a single free variable, *s*, find a sequence of actions, $\vec{a}_n = \langle a_1,...,a_n \rangle$, such that:

$$KB \models \text{Goal}(\text{do}(\vec{a}, s0)) \wedge \text{Legal}(\text{do}(\vec{a}, s0)). \tag{3}$$

Where $do(\vec{a}_n, s0)$ means $do(a_n, do(a_{n-1}, ..., do(a_1, s0)...))$, and $Legal(do(\vec{a}_n, s0))$ is equal to $\Lambda_{i=1}^{n} Poss(a_i, do(\vec{a}_{i-1}, s0))$, where $do(\vec{a}_0, s0) = s0$.

In others words, given a goal formula, we "hope" to find a sequence of actions, beginning in the known initial situation, so that:

- the goal formula will hold in the situation that results from executing the actions in sequence starting in the initial state, and
- it is possible to execute each action in the appropriate situation, (that is, each action's precondition is satisfied).

3. Design

Take as an example the robot presented in [3]. It considers four possible actions: *advance (avanzar)*, *reverse (retroceder)*, *remove_obstacle (elimina_obstaculo)* and *add_obstacle (agrega_obstaculo)*. If the robot carries out the action of *advance* the result is that the robot is in *x* if it was in *x*-1, if the robot carries out the action of *reverse* the result is that the robot is in *x* if it was in *x*+1. Moreover, if it executes advance or reverse the result is that the robot is not in *x* anymore if it was in *x*, we formally express these facts in the following successor state axiom:

$$\begin{aligned} robot(x,do(a,s)) \leftrightarrow\ & (robot(x-1,s) \wedge a = avanzar) \vee \\ & (robot(x+1,s) \wedge a = retroceder) \vee \\ & (robot(x,s) \wedge \neg(a = avanzar \vee a = retroceder)) \end{aligned} \quad (\mathbf{3})$$

Observe that the translation of this axiom to Prolog language has the same structure.

```
robot(X,do(A,S)):-robot(Y,S),
                  (
                  (A=avanzar, X is Y+1);
                  (A=retroceder, X is Y-1)
                  );
                  (
                  robot(X,S),
                  \+A=avanzar,
                  \+A=retroceder
                  ).
```

4. Implementation

Considering the preconditions of actions, the axioms of successor state, the axioms of legality and finally the necessary conversions for VB, the next program in Prolog works as the planner for the proposal in order to determine plans for the specified scenario. Some comments are given below.

```
/*initial conditions */
robot(3,s0).
obstaculo(5,s0).

/*preconditions for primitive actions */
poss(avanzar,S):-robot(X,S),
                 Y is X+1,
                 \+obstaculo(Y,S).
poss(retroceder,S):-robot(X,S),
                    Y is X-1, \+obstaculo(Y,S).

poss(agregar_obstaculo(X),S):-robot(Y,S),
                              (X is Y-1; X is Y+1),
                             \+obstaculo(X,S).
poss(eliminar_obstaculo(X),S):-robot(Y,S),
                               (X is Y-1; X is Y+1),
                               obstaculo(X,S).

/*successor state axioms for primitive fluents */
robot(X,do(A,S)):-robot(Y,S),
                  (
                  (A=avanzar, X is Y+1);
                  (A=retroceder, X is Y-1)
                  );
                  (
                  robot(X,S),
                  \+A=avanzar,
                  \+A=retroceder
                  ).
obstaculo(X,do(A,S)):-A=agregar_obstaculo(X);
                      obstaculo(X,S),
                      \+A=eliminar_obstaculo(X).

/*legal axioms*/
legal(s0).
legal(do(A,S)):-legal(S),poss(A,S).

cambia_s2n(s0,ACC,ACC).
cambia_s2n(do(A,S1),ACC,X):-cambia_s2n(S1,[A|ACC],X).

/* translates  */
convierte(X,C) :- X=avanzar->C=1;
                  X=eliminar_obstaculo(_)->C=2;
                  X=retroceder ->C=3;
                  X=agregar_obstaculo(_)->C=4.
```

The two first lines specify the initial state of the scenario, in this case the robot is in position three and there is an initial obstacle in position five.

The next four clauses specify the preconditions associated with the four actions that the robot can execute: *avanzar*, *retroceder*, *elimina_obstaculo(X)* and *agrega_obstaculo(X)*. For example, the first clause specifies that the robot can advance in situation *S*, only if the robot is at the position *X* and there is not an obstacle

at the position *X+1* in the same situation *S*. The rest of the preconditions have similar interpretations.

The next two clauses consider the successor state axioms regarding the position of the robot (fluent robot) and the positions of the obstacles (fluent obstaculo). For example, there is an obstacle in position *X* in situation *do(A,S)* only if action *A* considers the introduction of an obstacle in position *X* (*A=agrega_obstacle(X)*) or if in the previous situation, i.e. in situation *S*, there is already an obstacle in position *X* and the action does not consider the elimination of this obstacle (*not (A=remove_obstacle(X))*.
The Prolog program considers the clause legal with only one parameter of type situation; this clause allows the calculation of a situation that satisfies all the preconditions of actions appearing in the situation, i.e. the legal clause verifies that the preconditions of the sequence of actions appearing in a situation are satisfied.

The function *cambia_s2n* is used to convert the form in which the answer appears so as to simplify its use in VB by removing the term *do* and ordering the actions to be executed by the robot.

We will say that the length of an optimal plan is less or equal to any other plan. Notice that the program obtains an optimal plan as its first answer. Given that in the search of the solution the planner builds, using the *legal* clause, situations which have the possibility of being executed, and given that *legal* provides as the first situation the initial situation, if it does not satisfy the goal, then *legal* provides the situations with one action; if these kinds of situations do not satisfy the goal, then *legal* provides the situations with two actions and so on. We could be confident that the plan will contain the minimum number of actions possible.

The objective of this work is to use these results to control the movements of a mobile Khepera® II robot, its arm and its gripper. The robot can be controlled through a serial port by issuing a sequence of instructions [7]. Thanks to the research-oriented nature of the robot's design its central processing unit (CPU or microcontroller) does not required any task specific programming and it responds to a set of commands. To achieve this transmission of a series of commands to the robot it was necessary to use a program that served as a connection between Prolog and the robot. This program should be developed in a language capable of sending data via the computer's serial port. After some research, it was determined that the SICStus-Prolog program v3.12.7 provides a simple interface for unidirectional use, which allows the loading and to calling of Prolog programs from VB. As SICStus is executed under Windows XP, in order to make a query in SICStus from VB, the following considerations must be taken into account: VB must explicitly be initialized, calling the *PrologInit()* function before calling others. In order to execute a query, it must be explicitly opened, via the *PrologOpenQuery* function, which returns a query identifier that can be used for recovery of the next solutions [5]. The *PrologCloseQuery* will close the consultation represented by a query identifier. The Prolog queries can then be executed with the help of the *PrologNextSolution* function: this obtains a solution of the open query represented by the identifier of the query given as a parameter, returning 1, when there is success, 0 when there is a fault and -1 in the case of an error.

After the successful return of *PrologNextSolution*, the values assigned to the variables of the query can be obtained by the specific functions of the interface. There are separate functions to obtain the value of the variables in a string (*PrologGetString*) or as an integer (*PrologGetLong*). The *PrologGetString* function obtains the value of a variable given within a query as a string where as *PrologGetLong* the value of the given variable is turned into a whole number, returning 1, when there is a success. In this way, the most important lines of the program in VB are the following ones: The string Expr is taken from the text box of the application, which asks the user to write a query.

```
Q = Expr                                    ‘ Form the query,
qid = PrologOpenQuery(Q)                    ‘ Open the query
If qid = -1 Then GoTo Err                   ‘ Verify the identifier of query
    gret = PrologNextSolution(qid)          ‘ Get a solution
   If gret <> 1 Then GoTo Err ' failed or error
    gret = PrologGetString(qid, "SS", result)  ´Take the output in form of string
```

Given the previous scenario, if the robot has as a goal to arrive at position 7, the query would be: *legal(S),robot(7,S)*, and the answer would be *SS* = [avanzar, eliminar_obstaculo(5), avanzar, avanzar, avanzar]. With this solution, VB interprets the code as: *avanzar* = 1, *eliminar_obstaculo(X)* = 2, *retrocede* = 3 y *agrega_obstaculo(X)* = 4, in order to create the string [1, 2, 1, 1, 1]. This corresponds to a code in the VB program which translates orders which are sent to the serial port.
The orders sent to the robot are: G,0,0 initialize the coordinates for the position of the robot; C,600,600 indicate that the robot must be advance a distance of 4.8 cm.; if the command C,0,0 is sent, the robot would be back to the initial position. To control the arm, we use the orders: T,1,E,161 to lift the arm to the required position; T,1,D,100 lowers the arm to the required position in order to remove an obstacle; finally T,1,D,1 and T,1,D,0 is used to open and close the gripper, respectively.

The next figure shows the user interface of the application, in which it is possible to write a query and to get the solution. It also allows the user to set the speed of communication for the serial port.

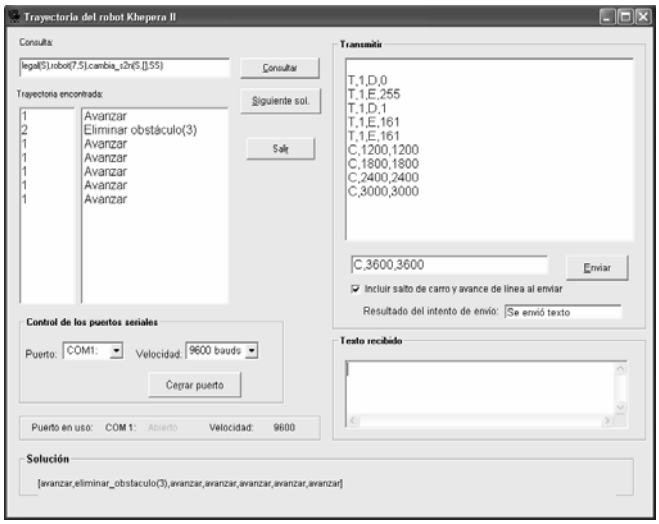

Fig. 1. Interface of user in VB.

To the left it is possible to see the robot's actions appear, and to the right it shows the commands sent to the serial port.

5. Test and results

It is possible to verify that the plan created was executed as a series of action by the robot. The following figure shows the robot and the obstacle.

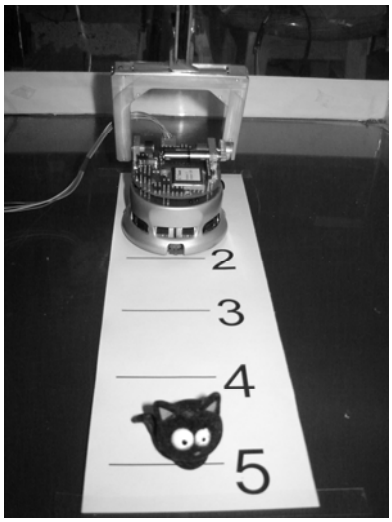

Fig. 2. Mobile robot follows the trajectory to the goal, if an obstacle is in its way, it removes it using the gripper.

The test was applied to satisfy diverse goals with different obstacle's location, and the results were satisfactory. We tested our proposal with the following query: legal(S), robot(X,S), cambia_s2n(S,[],SS), where X is the position that the robot must reach.

In order to compare the results obtained, different computers were used:

- Dell Desktop, Pentium IV, 1.99 GHz, 768 MB, XP
- Toshiba Laptop, Celeron 1GHz, 512 MB, XP

Consider as query 1 the following initial situation: the robot is in position 1, and an object is in position 3. Table 1 shows different desired positions, the time required to obtain the answer from Prolog and the inferences necessary for satisfying the goal.

Table 1. Values for the query 1: position to reach, time required to obtain the answer, and number of inferences.

Posi-tion	Time (sec) Dell	Time (sec) Toshiba	Inferences	Plans
1	0	0	51	agregar_obstaculo(0)
2	0	0	22	Avanzar
3	0	0	1,080	avanzar, eliminar_obstaculo(3), avanzar
4	0	0.01	7,361	avanzar, eliminar_obstaculo(3), avanzar, avanzar
5	0.01	0.03	49,645	avanzar, eliminar_obstaculo(3), avanzar, avanzar, avanzar
6	0.13	0.22	330,033	avanzar, eliminar_obstaculo(3), avanzar, avanzar, avanzar, avanzar
7	0.89	1.43	2,179,704	avanzar, eliminar_obstaculo(3), avanzar, avanzar, avanzar, avanzar, avanzar
8	5.74	9.47	14,356,233	avanzar, eliminar_obstaculo(3), avanzar, avanzar, avanzar, avanzar, avanzar, avanzar
9	37.7	66.12	94,532,951	avanzar, eliminar_obstaculo(3), avanzar, avanzar, avanzar, avanzar, avanzar, avanzar, avanzar

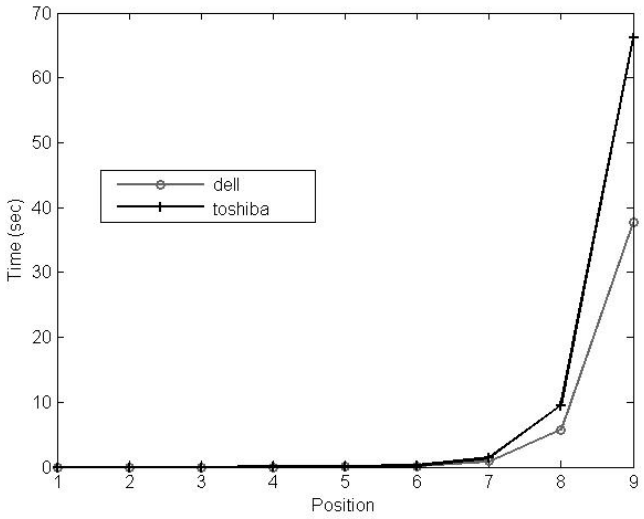

Fig. 3. Graph position to reach vs. time to obtained the answer for query 1.

As query 2 we consider two obstacles and the initial position of the robot is 1. The initial situation of the planner was loaded as follows:

```
robot(1,s0).
obstaculo(3,s0).
obstaculo(4,s0).
```

The obtained results are shown in Table 2.

Table 2. Values for the query 2: position to reach, time for calculating the answer, number of inferences and plans.

Posi-tion	Time (sec) Dell	Time (sec) Toshiba	Inferences	Plans
1	0	0	51	agregar_obstaculo(0)
2	0	0	22	Avanzar
3	0	0	1,080	avanzar, eliminar_obstaculo(3), avanzar
4	0.02	0.03	51,615	avanzar, eliminar_obstaculo(3), avanzar, eliminar_obstaculo(4), avanzar
5	0.13	0.22	338,240	avanzar, eliminar_obstaculo(3), avanzar, eliminar_obstaculo(4), avanzar, avanzar
6	0.89	1.54	2,205,341	avanzar, eliminar_obstaculo(3), avanzar, eliminar_obstaculo(4), avanzar, avanzar, avanzar

7	5.76	9.28	14,351,333	avanzar, eliminar_obstaculo(3), avanzar, eliminar_obstaculo(4), avanzar, avanzar, avanzar, avanzar
8	40.48	60.33	93,454,303	avanzar, eliminar_obstaculo(3), avanzar, eliminar_obstaculo(4), avanzar, avanzar, avanzar, avanzar, avanzar
9	290.22	392.85	608,973,879	avanzar, eliminar_obstaculo(3), avanzar, eliminar_obstaculo(4), avanzar, avanzar, avanzar, avanzar, avanzar, avanzar

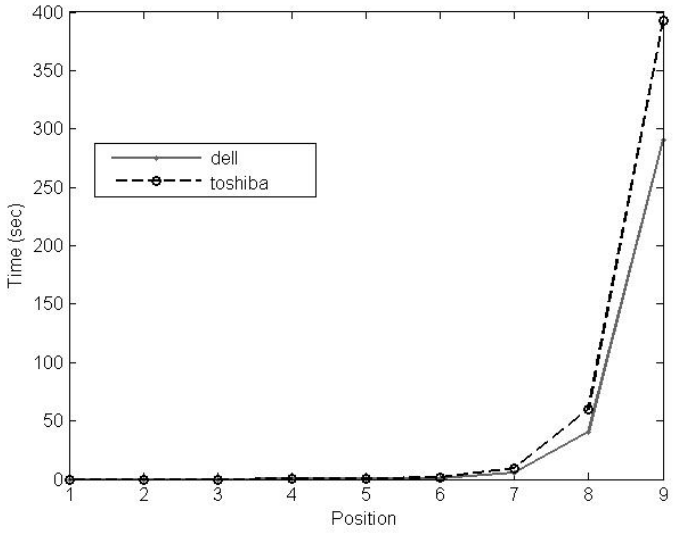

Fig. 4. Position to reach vs. time to obtain the answer for the query 2

Note that the time grows in an exponential fashion as the position increments. On the other hand, once the program in VB, with the help of Prolog finds outs which actions to make, the commands are sent to the robot in a serial form. The time for delay that must exist between two consecutive commands is approximately 1.5 seconds.

6. Conclusions

We present the implementation of an interface that allows interpretation and transfer to a mobile robot of the results provided by a planner implemented in Prolog. The planner was implemented in a declarative language giving the advantage of fast and easy modification of the representation of the initial situations and evolution of the scenes. Handling of dynamical scenes is given through the axioms of successor states. This is one of our first stages towards the creation of a cognitive robot. It makes a compromise between the theory and practice and there are many issues that have been considered on both sides.

Future work will try to provide feedback to the planner based on environmental information. This will be sensed by the robot and transferred via the serial port thus making it possible to revise the plan in the light of any changes to the situation.

In addition the development of two dimensions scenarios will be pursued to increase the degree of complexity of the planning task with the intention of giving the robot more realistic tasks.

References

1. Brachman, R., Levesque, H., Knowledge Representation And Reasoning**,** Morgan Faufmann (2004) 285-297.
2. Demolombe R., Pozos P., Theories Of Intentions In The Framework Of Situation Calculus, Springer-Verlag Berlin Heidelberg 2005.
3. Demolombe R., Pozos P., Implementación de la arquitectura BDI basada en el cálculo de situaciones, In Proc. 2ndo Congreso Internacional de Informática y Computación de la ANIEI. Zacatecas, 2003.
4. Reiter R., Knowledge in Action, Logical Foundations for Specifying and Implementing Dynamical Systems, MIT Press, 2001.
5. Intelligent System Laboratory, SICStus Prolog User´s Manual – release 3.12.7, Swedish Institute of Computer Science, October 2006.
6. Reiter R., The frame Problem in the situation calculus: a simple solution (sometimes) and a completeness result for goal regression. In Lifschitz, V., ed.: Artificial Intelligence and Mathematical Theory of Computation: Papers in Honor of John McCarthy, Academic Press (1991) 359-380.
7. User Manual Khepera II, ftp.k-team.com/khepera/documentation/KheperaUserManual.pdf.