MINIMUM COST PATH SEARCH IN AN UNCERTAIN-CONFIGURATION SPACE

Eric Pierre and Romain Pepy
Institut d'Électronique Fondamentale
Université Paris XI
Orsay, France

Keywords: Path planning, uncertainty, A* algorithm.

Abstract: The object of this paper is to propose a minimum cost path search algorithm in an uncertain-configuration space and to give its proofs of optimality and completeness. In order to achieve this goal, we focus on one of the simpler and efficient path search algorithm in the configuration space : the A* algorithm. We then add uncertainties and deal with them as if they were simple dof (degree of freedom). Next, we introduce towers of uncertainties in order to improve the efficiency of the search. Finally we prove the optimality and completeness of the resulting algorithm.

1 INTRODUCTION

Today's researches on mobile robots focalize on a specific aspect: the autonomous navigation of a robot. Navigating a robot generaly involves the use of a path planner, which determines a path in a graph built from the environment. However, where path planner taking into account geometric models of the environment work well in simulation, the implementation of such algorithms on real robots leads in most case to failure to follow the path planned (Latombe, 1991). Thus, searchers have decided to introduce the concept of uncertainty, in order to generate a path that would work even with uncertain data. This field of research has leaded to various works (Fraichard and Mermond, 1998; Lazanas and Latombe, 1995; Pepy and Lambert, 2006).

The present paper is the following of the work in (Lambert and Gruyer, 2003), which proposes a path planner based on the A* algorithm ((Hart et al., 1968; Nilsson, 1988; Russel and Norvig, 1995)) that takes uncertainties into account. Using a simulator of exteroceptive and proprioceptive sensors and an Extended Kalman Filter to process the sensors information, the algorithm finds a *safe* path for a mobile robot in a known-but-imperfect indoor environment despite the uncertainties associated with the lack of precision of the sensors. Unfortunately, the authors do not prove the optimality and completeness of the algorithm they propose.

The goal of the present paper is firstly to generalize the algorithm given in (Lambert and Gruyer, 2003) to any problem based on planning a path in a graph with taking into account uncertainties on the graph's parameters. Secondly, it is to prove the optimality and completeness of this generalized algorithm.

In section 2, we build the mathematical structure needed to explain the A* algorithm and to prepare the proofs. Using the mathematical bases proposed in the previous section, section 3 presents the A* while taking the uncertainties into account, which we will call A* in an Uncertain Configuration Space (AUCS*). Finally, section 4 proposes a second and optimized version of this algorithm (called Safe A* with Towers of Uncertainties, or SATU*) using a new concept: the towers of uncertainties (Lambert and Gruyer, 2003).

2 THE A* ALGORITHM

2.1 Overview

The famous A* algorithm (algorithm 1) is one of the simpler and efficient path planner in a graph. This algorithm has first been proposed in (Hart et al.,

Pierre E. and Pepy R. (2007).
MINIMUM COST PATH SEARCH IN AN UNCERTAIN-CONFIGURATION SPACE.
In *Proceedings of the Fourth International Conference on Informatics in Control, Automation and Robotics*, pages 103-109
DOI: 10.5220/0001645301030109

1968), and is based on the Dijkstra algorithm (Dijsktra, 1959) on which a heuristic function has been added.
The algorithm searches the graph in order to find a path that goes from a start node to a goal node. When we deal with a path planner, it is important to know its behaviour, especially about two particular properties :

- Optimality: When the algorithm chooses a path to go from the start node to the goal node, this path must be the optimal one (given some criteria of optimality).
- Completeness: Given that the goal is reachable, the algorithm must reach it in a finite time.

If the path planner we deal with verifies those properties, we know for sure that a path will eventually be found (given that the goal node is reachable) and that this path will be the best one, according to a given criteria.

2.2 Spaces Used

We want to find a path in a **n**-dimension configuration space $\mathcal{X}$. The **n** dimensions represent the degree of freedom (dof) of the system. We have to deal with two subspaces of $\mathcal{X}$, one representing the *free*-configuration space (called $\mathcal{X}_{free}$), where we can move, and the other representing the *collision*-configuration space (called $\mathcal{X}_{collision}$), where the movements are impossible.

2.3 Structure of the Graph

We discretize $\mathcal{X}_{free}$ into an infinite number of nodes. Let $\mathcal{N}$ be the set of those nodes. They are characterised by **n** parameters. We can build a directed graph (the nodes are linked together by one-way edges) in $\mathcal{X}_{free}$, which follows those properties :

Property 1 *There is a minimum positive edge cost between any two nodes (i.e., there is a given positive* ε *so that the cost between any two nodes is greater than* ε*)*

Property 2 *The number of edges starting from a node is finite.*

Property 3 *The edge cost between any two nodes in a path is finite.*

When a graph possesses the first two properties, it is called a *locally finite graph* (in a given and finite area of the space, there is a finite number of paths between any two nodes) (Nilsson, 1988).

Algorithm 1 A*

```
 1: CLOSE ← ∅
 2: OPEN ← NodeStart
 3: while OPEN≠ ∅ do
 4:   Node ← Shortest_f*_Path(OPEN)
 5:   CLOSE ← CLOSE + Node
 6:   OPEN ← OPEN - Node
 7:   if Node = NodeGoal then
 8:     return (Success,Node)
 9:   end if
10:   NEWNODES ← Successors(Node) ∉ CLOSE
11:   for all NewNode of NEWNODES do
12:     if NewNode ∉ OPEN or g(NewNode) >
        g(Node)+cost(Node,NewNode) then
13:       g(NewNode) ← g(Node)+cost(Node,NewNode)
14:       f*(NewNode) ←
          g(NewNode)+h(NewNode,NodeGoal)
15:       parent(NewNode) ← Node
16:       if NewNode ∉ OPEN then
17:         OPEN ← OPEN+NewNode
18:       end if
19:     end if
20:   end for
21: end while
22: return NoSolution
```

2.4 Paths of the Graph

- let $\mathcal{P}$ be the set of all the paths beginning at the start node and exploring the configuration space. An important characteristic is that no path in $\mathcal{P}$ can loop (a path cannot go through a node twice).
- let $\mathcal{P}_{goal}$ be the set of all the paths beginning at the start node and ending on the goal node.
- let $\mathcal{P}_{\overline{goal}}$ be the set of all the paths beginning at the start node and that do not reach the goal node. We have

$$\mathcal{P}_{\overline{goal}} = \mathcal{P} \setminus \mathcal{P}_{goal}. \quad (1)$$

An important thing to keep in mind is that the A* works incrementally with $\mathcal{P}$. It tests paths of $\mathcal{P}$ node by node and does not know if those paths belong to $\mathcal{P}_{goal}$ or $\mathcal{P}_{\overline{goal}}$ until it reaches their final node.

2.5 Parameters of the Algorithm

2.5.1 Criteria of Optimality

Each path has an associated cost, which is the cumulated costs of every edge in the path. Let $f(c_i)$ be this cost, for all $c_i \in \mathcal{P}$.

Let us introduce, $g(c_i, n)$, which represents the length of the part of the path c_i going from the start node to the node n, $n \in c_i$. If c_i is finite, we can observe that

$$f(c_i) = g(c_i, \text{finalnode}(c_i)). \quad (2)$$

We want to find a shortest path, i.e. a path $c \in \mathcal{P}_{goal}$ that verify

$$f(c) \leq f(c_i) \quad \forall c_i \in \mathcal{P}_{goal}. \tag{3}$$

Finding such a path is our *criteria of optimality*.

2.5.2 Rule to Expand the Graph

As it is well known, the A* works with a path-independant heuristic function $h(n_j, n_k)$ that estimates the unknown length between two nodes n_j and n_k. We can then work with an estimated total length $f^*(c_i, n)$ rather than the real (unknown) $f(c_i)$, such that

$$f^*(c_i, n) = g(c_i, n) + h(n, \text{goalnode}). \tag{4}$$

2.6 Proofs of the Algorithm

Both the proof of optimality and of completeness have been given in (Nilsson, 1988). We won't detail them further here.

3 THE A* IN AN UNCERTAIN CONFIGURATION-SPACE

3.1 The Concept of Uncertainties

The uncertainties represent the fact that in the real world, one cannot precisely know the real model of the systems. Thus, we can determine the errors induced by all the models used and we can calculate the uncertainties associated to each node (in $\mathcal{N}$) of each path (in $\mathcal{P}$). Let us consider that the uncertainties are fully described in a $\mathbf{m}$-dimension space. Consequently, a full uncertain configuration is now given by $\mathbf{n}+\mathbf{m}$ parameters: $\mathbf{n}$ for the coordinates in the space $\mathcal{X}$ and $\mathbf{m}$ for the coordinates in the uncertainties-space. Such an uncertain configuration will be called an *extended node*.
In this section, we are going to present a way to run a simple A* on a new graph based on extended nodes, in considering that the uncertainties can be managed as additional degree of freedom.

3.2 Spaces Used

Let $\mathcal{X}^e$ be the uncertain configuration space, which is $\{\mathbf{n}+\mathbf{m}\}$-dimension space and fully include $\mathcal{X}$ (we can then consider that we deal with $\{\mathbf{n}+\mathbf{m}\}$ dof, even if some of them are uncertainties). Knowing the uncertainties associated to a given node, we can build a new $\mathbf{n}+\mathbf{m}$ dimensions free-configuration space $\mathcal{X}^e_{free}$ taking into account those incertainties.
Let $\mathcal{X}^e_{collision}$ be characterised by

$$\mathcal{X}^e_{collision} = \mathcal{X}^e \backslash \mathcal{X}^e_{free}. \tag{5}$$

Our goal is now to find a safe path, which is a path that never go in $\mathcal{X}^e_{collision}$ (considering the uncertainties of every extended node).

3.3 Structure of the Graph

In the previous case (see section 2.3 above) we already introduced a graph based on $\mathcal{N}$. However, we do not want to work with the set of nodes $\mathcal{N}$, as it does not take the uncertainties into account. Let us introduce a new set of nodes, called $\mathcal{U}$. The nodes in this set will then be *extended* nodes. The properties applying on $\mathcal{N}$ apply also on $\mathcal{U}$ (the three properties presented in section 2.3 applie on this new graph). Thus, our new graph is locally finite.
Another property may be pointed out:

Property 4 *An infinite number of extended node can be based on the same node.*

3.4 Paths of the Graph

We can now introduce the paths generated through the *extended nodes* of $\mathcal{U}$.

- let $\mathcal{G}$ be the set of all the paths beginning at the start *extended node* and exploring the $\{\mathbf{n}+\mathbf{m}\}$-dimension space configuration $\mathcal{X}^e_{free}$. The criteria of optimality presented in section 2.5 applies fully here.
- let $\mathcal{G}_{goal}$ be the set of all the paths beginning at the start *extended node* and reaching the goal *extended node*. A direct consequence is that $\mathcal{G}_{goal} \subset \mathcal{G}$.
- let $\mathcal{G}_{\overline{goal}}$ be the set of all the paths beginning at the start *extended node* and that do not reach the goal *extended node*. We have

$$\mathcal{G}_{\overline{goal}} = \mathcal{G} \setminus \mathcal{G}_{goal}. \tag{6}$$

- g, f, f^* and h represent respectively the length from the start node to a fixed node in a path, the total length of a path, the estimated length of a path and the heuristic of a path in the $\{\mathbf{n}+\mathbf{m}\}$-dimension space.

What we finally search is the *shortest path*, which is the shortest path c in $\mathcal{G}$. This path must verify

$$f(c) \leq f(c_i) \quad \forall c_i \in \mathcal{G}. \tag{7}$$

3.5 The Algorithm

It is now possible to implement the A*, which we will call A* in an Uncertain Configuration-Space (AUCS*), directly on the newly described $\{\mathbf{n}+\mathbf{m}\}$-dimension graph. The AUCS* is exactly the same algorithm than the A*, the only difference is the graph on which it performs. The goal node is then defined by $\mathbf{n}+\mathbf{m}$ parameters, the first $\mathbf{n}$ corresponding to the location we want to reach and the $\mathbf{m}$ other parameters being the uncertainties we want to have at this location.
As the algorithm is exactly the same (despite it performs on a different graph), we do not give it again. See algorithm 1.

3.6 Proofs of the Algorithm

3.6.1 Proofs of Optimality

Despite the algorithm performs on an improved graph, its principal characteritics are unchanged. The heuristic and calculus of length have the same properties (g and h could eventually be identical to those used in the previous section), and regarding this model the AUCS* still find the shortest path. □

3.6.2 Proofs of Completeness

There again, the only cases the AUCS* could not be complete is when there is an infinite number of paths such as their lengths are lower than the found path's or when a path with a length lower than the found path is infinite.
The proof is equivalent to the one determined by (Nilsson, 1988): the properties of the graph built are the same than in the previous case (it is locally finite), this is enough to lead to the same conclusion ; the AUCS* algorithm is complete. □

3.7 Working in a Finite Graph

In the previous sections, we considered the graphs were infinite and built in advance, before the algorithm starts. However, building an infinite graph is impossible for a (necessarily limited) computer. Thus, some choices must be done if we want to be able to run the algorithm on a computer. For example, we could define a sub-space of $\mathcal{X}$ where to run the algorithm. Of course, this implies no path not being entirely contained in $\mathcal{X}$ can be found. This is generally not a problem if the sub-space is cleverly defined (for example, if we want to find a path leading from a room to another in a building, we could restrain our space to the building itself). This is enough to ensure the new graph will be finite:

Property 5 *A graph generated in a finite sub-space of $\mathcal{X}$ and following properties 1 and 2 (section 2.3) is finite.*

Proof: Property 1 ensures that any two nodes must have a given minimum positive edge cost. Consequently, a finite sub-space can only be filled with a finite number of nodes. Property 2 ensures that only a finite numbers of edges can go from a node: the graph is finite. □
Thus, we can work on a pre-built sub-graph of $\mathcal{X}$. However in the case of $\mathcal{X}^e$, the problem is slightly different. Property 4 implies there could be an infinite number of extended nodes on each node. Of course, it is not possible to build such a graph with a computer. Thus, we are going to work on a dynamically built graph: from $\mathcal{P}$, the extended nodes are dynamically added in the new graph $\mathcal{G}$ when they are reached.
Proofs of optimality and completeness ensure that the graph stays finite, as the dynamically building of the extended nodes ends when the algorithm reaches the goal node, which has been proven happens in a finite time.
The only change in algorithm 1 is that line 10, *Successors* function dynamically creates the nodes if they do not already exist. This implies that, given a fully-described extended node and the base of the node where to go, *Successors* must be able to calculate the $\mathbf{m}$ last parameters of the node to reach.
We now have an implementable algorithm in an uncertain configuration space, which does not need infinite memory.

4 THE A* WITH TOWERS OF UNCERTAINTIES

4.1 The Towers of Uncertainties

In section 3.7, we showed that we could work on a finite dynamically built graph. However, finding a way to reduce the need of memory needed would be very helpful. (Lambert and Gruyer, 2003) has determined a method (described below) that reduces the field of search (reducing the memory needed) and reorganizing the way to record the data. This method involves the use of *towers of uncertainties*, and only works on dynamically built graphs (as presented section 3).
However, the graphs must be more constrained than those seen in the sections above.
Constraint 1: This constraint concerns the evolution

of the uncertainties between two nodes: this evolution follows a given function and this function is the same for every uncertainty. If the uncertainties of a given extended node include the uncertainties of another extended node (those extended nodes must be based on the same node), then the pattern ensures that this property will be kept in the following nodes of the paths (when the paths follow the exact same nodes).
Constraint 2: If the function (collision-detection) giving the configuration-space where the path belongs ($\mathcal{X}_{free}$ or $\mathcal{X}_{collision}$) declares the path in $\mathcal{X}_{collision}$ for a given uncertainty, then it will declare a new path following the same (non-extended) nodes with wider uncertainties in $\mathcal{X}_{collision}$ too.

When the AUCS* searches the graph, it generates various beginning of paths. Given that numerous extended nodes can be at the same position (see property 4 above), it is perfectly possible to have two or more different paths reaching the same position (same node, but different extended nodes).
Without knowing what comes in the future, could not we already discard some of those paths?
The criteria of optimality we want to minimize is the length of the path. What we search is the smallest path. However, some paths expanded at some step of the algorithm may not reach the goal (for example, every path built from this path does not belong to $\mathcal{X}^e_{free}$). Thus, the current smallest path could be in $\mathcal{X}^e_{collision}$ without the algorithm knowing it (the part being in collision with the environment having not been reached by the algorithm yet) and a longer path could be in $\mathcal{X}^e_{free}$. The immediate consequence is that we should keep both of them during the search.
What about if the longer path has also wider uncertainties? Then it is clear that if the smallest path does belong to $\mathcal{G}_{\overline{goal}}$, the longer path belongs to $\mathcal{G}_{\overline{goal}}$ too (this is ensured by constraint 2 presented above, in section 4.1, as from this node on, the exact same paths will be tried by both of the paths). In this case, we could discard the longer path. On the other hand, if the smallest path is in $\mathcal{G}_{free}$, then we do not need the longer path anymore, even if it is in $\mathcal{G}_{free}$ too, as its length is by definition longer. We could also discard it in this case.
Consequently, we can *always* discard the longer path with the wide uncertainties.
This property considerably reduces the complexity of the algorithm, as it detects earlier useless paths. In order to make the detection of the useless paths the quicker possible, (Lambert and Gruyer, 2003) proposes the use of *towers of uncertainties* in an algorithm we call SATU* (Safe A* with Towers of Uncertainties).
Each extended node is divided in two entities:

- *A base* : Given by the node of the extended node (the **n** first parameters).
- *A level* : The uncertainties associated to the node and function of the path leading up there (the **m** following parameters).

So, we can now dynamically (while the algorithm performs) build a tower of uncertainties, with a common base, and as many levels as the number of extended nodes (with the same common base) opened at this point. The levels are placed following some rules:

- The levels are placed in an increasing order of their length.
- Given a level, if another level under it has uncertainties that completely fits in its own, then the given level is removed (its length is bigger and its uncertainties are wider than the level's under it).

A direct consequence of the description above is that the memory needed for the SATU* is lower than for the AUCS*, even if we consider that the AUCS* build the graph dynamically too: for a given number n of extended nodes which have the same base, the needed memory will be $(\mathbf{n}+\mathbf{m})n$ in the AUCS* and only $\mathbf{n}+n.\mathbf{m}$ in the the SATU*.

4.2 Description of the Algorithm

SATU* is given in algorithm 2. The first part of the algorithm is the same as algorithm 1: two lists are created and initialized, a loop allows to expand extended node after extended node (selecting the extended node with the smallest f), a test verify if the goal extended node has been reached, otherwise the successors of the current extended node are selected and opened.
For each of those successors, a first test (line 12) checks if the base of the successor is already in CLOSE or OPEN. If it is not, then when can create a new tower on this base, add a first level with the uncertainties associated with the successor and add the base of the node in OPEN (without forgetting to calculate f and to store the parent of the node in order to be able to find the path when the algorithm is finished).
If the base of the successor belongs either to CLOSE or OPEN, we need to check if the successor may enter the tower or not.
In order to do that, the algorithm must compare each one of the levels of the tower with the successor's uncertainties and g. Line 20 and 21, the algorithm selects the tower and initialized an index which will represent the value of the current level being checked. Line 22 begins the loop that will compare the current level extracted and the successor.
Line 25, the current level to compare is extracted from

Algorithm 2 (SATU*)

```
 1: CLOSE ← ∅
 2: OPEN ← NodeStart
 3: while OPEN≠ ∅ do
 4:  Node ← Shortest_f*_Path(OPEN)
 5:  CLOSE ← CLOSE + Node
 6:  OPEN ← OPEN - Node
 7:  if Base(Node) = Base(NodeGoal) and uncertainty(Node) ⊆ uncertainty(NodeGoal) then
 8:   return Success
 9:  end if
10:  NEWNODES ← Successors(Node)
11:  for all NewNode of NEWNODES do
12:   if NewNode ∉ OPEN,CLOSE then
13:    g(NewNode) ← g(Node)+cost(Node,NewNode)
14:    f*(NewNode) ← g(NewNode)+h(NewNode,NodeGoal)
15:    build(NEWTOWER,base(NewNode))
16:    AddLevel(NEWTOWER,NewNode)
17:    OPEN ← OPEN+NewNode
18:    parent(NewNode) ← Node
19:   else
20:    TOWER ← ExtractTower(base(NewNode))
21:    level ← -1
22:    do
23:     AddLevel ← false
24:     level ← level+1
25:     LevelNode ← ExtractNode(TOWER,level)
26:     if (g(NewNode) ≥ g(LevelNode) and uncertainty(LevelNode) ⊈ uncertainty(NewNode)) or g(NewNode) < g(LevelNode) then
27:      AddLevel ← true
28:     end if
29:    while level ≠ TopLevel(TOWER) and AddLevel=true
30:    if AddLevel=true then
31:     level ← insert(NewNode,TOWER)
32:     OPEN ← OPEN + NewNode
33:     parent(NewNode) ← Node
34:     UpperNodes ← nodes(UpperLevels(TOWER,level))
35:     for all uppernode of UpperNodes do
36:      if uncertainty(NewNode) ⊆ uncertainty(uppernode) then
37:       remove(TOWER,uppernode)
38:       OPEN ← OPEN - uppernode
39:      end if
40:     end for
41:    end if
42:   end if
43:  end for
44: end while
45: return NoSolution
```

the tower.
Line 26, a test is performed. We may insert the successor in the tower if either its g is greater or equal than the level's and its uncertainties are not included in the level's or if its g is lower than the level's. Thus, if those conditions are met, we keep the successor and try to compare it to the next level. If, just once, those conditions are not met, then we can discard the successor.
Line 30, if we may insert the successor in the tower, we do it.
Line 31, we insert the successor in the tower in the increasing order of level's g.
we then add the successor in OPEN, for it to be expanded later.
Line 34, we select the nodes with a greater g than the successor's, and check if they can be kept (if their uncertainties are included in the successor's, then we can discard them). This is done in removing them from the tower and from OPEN lines 37 and 38.

4.3 Proofs of the Algorithm

4.3.1 Proof of Optimality

As shown in section 4, the SATU* works as a AUCS*. The addition of the tower of uncertainties only allows to discard very early a path that is either in $\mathcal{G}_{\overline{goal}}$ or not the smallest path in $\mathcal{G}_{free}$. Thus, regarding the discovery of the smallest path, it has the exact same behaviour than a AUCS*, which proves that it respects the criteria of optimality. The algorithm therefore ensures that its found path c verifies

$$g(c) \leq g(c_i) \quad \forall c_i \in \mathcal{G}_{free}. \tag{8}$$

□

4.3.2 Proof of Completeness

Given that the optimality of the SATU* has been proven above and that the graphs used have the same properties than for the AUCS*, we can directly conclude that the SATU* is complete (as the graphs used are locally finite). □

5 CONCLUSION

In this paper, we firstly presented a simple way to run an A* algorithm in an uncertain-configuration space (AUCS*) by considering the uncertainties as simple degree of freedom of the system. This characteristic allows the add of uncertainties in any path planner algorithm that respects the necessary properties.

Secondly, we introduced a new path planner, the SATU* algorithm, working in an uncertain-configuration space, strongly based on the A* algorithm that limits the memory needed. We then gave the needed proofs of optimality and completeness associated. Some further work will focalize on the calculation and comparison of the AUCS* and SATU*

complexities and on the experimental tests using a real mobile robot.

REFERENCES

Dijsktra, E. (1959). A note on two problems in connexion with graphs. In *Numerische Mathematik*, volume 1, pages 269–271.

Fraichard, T. and Mermond, R. (May 1998). Path planning with uncertainty for car-like robots. In *IEEE Int. Conf. on Robotic and Automation*, pages 27–32.

Hart, P., Nilsson, N., and Raphael, B. (1968). A formal basis for the heuristic determination of minimum cost paths. In *IEEE Transaction on Systems Science and Cybernetics, SSC-4(2)*, pages 100–107.

Lambert, A. and Gruyer, D. (September 14-19, 2003). Safe path planning in an uncertain-configuration space. In *IEEE International Conference on Robotics and Automation*, pages 4185–4190.

Latombe, J. C. (1991). *Robot Motion Planning*. Kluwer Academic Publisher.

Lazanas, A. and Latombe, J. C. (1995). Motion planning with uncertainty : a landmark approach. In *Artificial Intelligence 76(1-2)*, pages 287–317.

Nilsson, N. J. (1988). *Principles of artificial intelligence*. Morgan Kaufman Publisher inc.

Pepy, R. and Lambert, A. (2006). Safe path planning in an uncertain-configuration space using rrt. In *Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5376–5381, Beijing, China.

Russel, S. J. and Norvig, P. (1995). *Artificial Intelligence: A modern Approach*, pages 92–101. Prentice Hall, Inc.