Anytime Motion Replanning in State Lattices for Wheeled Robots

Adrián González-Sieira, Manuel Mucientes and Alberto Bugarín

***Abstract*—Autonomous robots require robust and fast motion planning algorithms to operate in complex real environments. In the last years, motion planning in state lattices has emerged as a powerful paradigm to real time path planning taking into account the kinematic restrictions of the vehicle. The approach requires the definition of the state lattice and the off-line calculation of the motion primitives. Therefore, motion planning is transformed into a search problem over a directed graph. In this paper, we apply the state lattice approach for motion planning of wheeled robots usign the AD* algorithm. Thus, the planning algorithm is anytime and dynamic, i.e., the path is improved incrementally and, also, the algorithm can replan. Results of different tests on a Pioneer P3-DX show the validity of the proposal.**

***Index Terms*—motion planning, state lattices, motion primitives, replanning, wheeled robots.**

I. INTRODUCTION

Motion planners are essential components of autonomous vehicles. Their purpose is to determine a set of control actions that allow the vehicle to reach one or more objectives from the starting position.

Desirable properties for a motion planner are: i) react in real time to changes in the environment (specially in dynamic or dangerous situations); ii) avoid unnecessary risks when traversing areas with obstacles; iii) generate optimal paths fulfilling certain criteria (time spent in reaching the objective, energy cost, minimum distance, etc.); iv) consider changes in the environment and adapt the solution to guarantee the successful completion of the task.

Many motion planners have been proposed over the years. However, most of them present drawbacks related with the computational burden necessary to calculate the solution, the weakness of the planner from preventing local minimums or the lack of mechanisms to allow an efficient replanning in dynamic environments, or not fulfilling the kinematic restrictions of the motion model of the robot.

Motion planning solutions can be classified in two sets of approaches: those that consider the motion model of the vehicle at planning time, and those that obtain the path without considering those kinematic restrictions. The second approach is the most usual, and it makes the control of the vehicle an independent problem to be solved separately. This simplifies a lot the path finding problem, but it may provoke fails when the solution paths are not truly feasible because of the vehicle motion model.

Adrián González-Sieira, Manuel Mucientes and Alberto Bugarín are with the Centro de Investigación en Tecnoloxías da Información (CITIUS), Universidade de Santiago de Compostela (Spain).
Their e-mails are: adrian.gonzalez@usc.es, manuel.mucientes@usc.es and alberto.bugarin.diz@usc.es

On the other hand, planning with the motion model involves a greater computational effort, as the algorithms must consider the information about the complete vehicle state (not only its pose), manage the kinematic restrictions of the motion model, assign a cost for each action depending on several criteria (obstacles, time, energy, etc.).

The discretization of the robot state space is a usual way to reduce the computational complexity of motion planning. A special discretization, called *state lattice* [1], can be used to express the planning problem as a directed graph, where the nodes are all the reachable states of the vehicle, and the arcs the feasible motions that connect them. This type of discretization has proved to have two important properties: its construction observes the motion model kinematic constraints, and its regularity allows to perform many optimizations to reduce the computational effort of calculating a path between states.

In this paper, a global planning method for a wheeled robot is presented. It uses the *state lattice* discretization to efficiently represent the search space. The connectivity is built from a motion primitives set that fulfills the vehicle kinematic constraints. The planner uses an anytime and incremental search algorithm (AD* [2]) guarantee a real time response, and refine the solution while there is available computation time. The algorithm is also able to replan: it takes into account the recent changes in the environment and modifies the proposed trajectory in real time.

The paper is structured as follows: Sec. II summarizes recent work related with this proposal. Sec. III presents the proposed *state lattice* discretization, while Sec. IV describes the search algorithm and its heuristics and Sec. V the construction of the feasible motions. Sec. VI presents the obtained results and, finally, Sec. VII points out the conclusions.

II. RELATED WORK

There are many different approaches to solve motion planning problems [3]. The simplest proposals are those based on local planning, which are focused on giving the next feasible action for the vehicle that seems closer to the objective considering the surrounding environment. These techniques include the potential fields [4], [3], [5], the curvature velocity [6], dynamic window [7], etc.

This set of algorithms is very efficient and its performance in simple problems is reasonably good. However, these algorithms have demonstrated their lack of utility in complex configuration spaces. As local search methods, the decision of the next control action is greedy, which makes them to fail when trapped in local minima.

To get better results than the previous approximations, the local control was improved by selecting the most promising choice according to the estimated distance to the goal [8], [9] using the global environment information. Since they still follow a greedy strategy, they cannot guarantee the vehicle to reach the objective. This strategy was modified to use the composition of several simple actions [10], [11], but the problem of falling in local minimums is still present.

Solutions using the global environment information can be grouped in [3]: geometric methods, and sample based approximations. The first one uses the robot and obstacle representation to search all feasible paths in the environment. The second ones use a discretization scheme to avoid taking into account the full environment representation in order to minimize the computational burden.

Geometric methods try to connect every two points of the free space. They require the fulfillment of two conditions: all points in the free space are connected at least for a path, and the possibility of representing the free space points as a graph. Although the approach is very interesting due to its completeness property, these methods are very dependent of the configuration space and they need very high computational resources in the presence of a complex environment with many obstacles. An important consideration about them is the impossibility of working with dynamic environments: each time the environment changes, it is necessary to recalculate the set of all paths that traverse the free space, and there is no way to represent moving obstacles.

Sample based algorithms are global planning methods that avoid the explicit representation of free and occupied zones. The space representation is minimized using a discretization scheme, and uses a global planning strategy to build a path that can be tracked by the vehicle [12], [13], [2]. This approximation is independent of the collision detection problem and the vehicle geometric representation, making a difference with the rest of methods introduced before. This is the most successful approximation of the state of the art, and it is widely used in motion planning problems. The feasibility of the problem depends on the discretization scheme used. Moreover, these type of algorithms are probabilistic complete: if the number of points to represent the planning problem is high enough, the probability of finding an existing solution tends to 1.

There are two main approximations for sample based algorithms: i) probabilistic roadmap methods [3], that takes random samples from the robot space configuration and tests if they occur in free space, using a local planner to connect the generated configurations to other nearby ones; ii) state lattice based methods [1], that use a regular discretization scheme to optimize the search space representation and set up the problem as a directed graph.

III. Motion planning in state lattices

The proposed planning algorithm is based on a state lattice discretization scheme. The *state lattice* [1] is a regular sampling of the state space that considers by construction the motion differential constraints. Different type of lattices exist, depending on the spatial arrangements of the states (triangular, diamond, rectangular, etc.). In this work a rectangular lattice has been used.

The state space forms a directed graph, where the nodes are the discrete robot states, and the arcs are sequences of control commands that connect them. The arcs are generated through the robot motion model and, therefore, fulfill the kinematic constraints. In this manner, a discrete search algorithm can be executed over the lattice to obtain the optimal path between any pair of states. As the connections between states are feasible motions of the robot, it is guaranteed that the obtained path will be a feasible plan as well.

For this type of algorithms, the robot state is defined as a vector of variables that are sampled with a given resolution. In order to consider the vehicle motion constraints, a state must contain information about the pose, and the linear and angular velocities:

$$s = [x, y, \theta, v, \omega] \tag{1}$$

The regularity of the lattice introduces important benefits such as the translational invariance: a connection between states is replicated for all pairs of states equally arranged. This allows to use a precomputed set of motion primitives, called primitive trajectories (or canonical control set), to replicate the connectivity for any node in the lattice, regardless of its position, as depicted in Fig. 1.

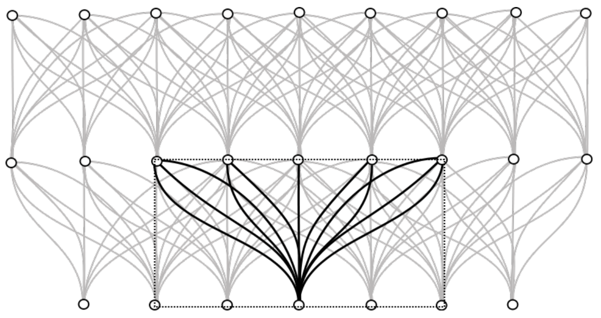


Fig. 1. A typical rectangular lattice. The primitive trajectories (in black) are replicated in all the equivalent nodes.

To improve the effectiveness of the planner and the quality of the solutions found, the construction of the state lattice was designed such as it meets the following properties:

- Optimality of the motions between states, considering only those with a minimal transition time.
- Completeness of the canonical control set to represent all feasible motions of the robot model.
- Minimum complexity of the state lattice connectivity, by removing all redundant transitions.

IV. Search Algorithm

The problem of finding a path in the state lattice was solved using a discrete search algorithm. These type of algorithms executes a informed search over a directed graph to find the optimal solution between a pair of nodes. Its efficiency depends on two aspects: the quality of the heuristic, that estimates the transition cost between every node in the graph and the goal, and the connectivity of the state lattice, that is determined by the canonical control set

There are in the literature search algorithms that have interesting properties to use in motion planning problems: one of them is centred in replanning in dynamic environments, and others in obtaining fast sub-optimal solutions that are improved in several iterations to, finally, get the optimal one.

Anytime Dynamic A* (AD*) [2] combines a replanning mechanism and anytime search. It makes it specially interesting for real time problems located in dynamic environments. It is possible to calculate a sub-optimal solution, adjusting the upper bound of sub-optimality depending on the time available for the calculation; and it is also possible to repair iteratively a previously calculated solution introducing changes in the transition cost of arcs in the graph.

Another benefit of AD* is the backward exploration of the graph. The solution is obtained beginning the search process from the objectives to reach. This allows to change the starting node between iterations, obtaining a solution that varies with the changes in the present position of the vehicle.

The cost function gives a value for each one of the arcs that connects the nodes in the search space. This cost is a combination between the time that takes the robot to execute a motion, and the information about the obstacles in the occupancy grid map.

Depending on the occupancy probability, each cell of the grid map can be in one of these states: "free", "occupied" or "unknown". To calculate the cost of a trajectory, it is necessary to consider the occupancy information of cells passing through the robot. Taking into consideration only the cells traversed by the centre of the vehicle may result in obtaining solutions that could be unfeasible due to the robot shape.

To avoid this, the cost function considers the information given by all the cells traversed by the shape of the robot. The value assigned to each motion is calculated as the convolution cost of all cells affected by it, and its execution time:

$$c(s, s') = \left(\frac{\sum_{\alpha \in cells} t(\alpha) \cdot w(\alpha)}{\sum_{\alpha \in cells} t(\alpha)} \right) \cdot t(s, s') \qquad (2)$$

where $w(\alpha)$ is the weight of cell α in the occupancy map (proportional to the occupancy probability of α), $t(\alpha)$ is the time spent in a cell during the execution of a trajectory, and $t(s, s')$ is the transition time between states. Given this transition cost function, the planner minimizes the time spent by the vehicle reaching the objective state from the initial one, while avoiding the obstacles in the map.

Depending on the size of the cells in the map, calculating the convolutions for all transitions can become expensive. In many cases convolutions calculation can be avoided: when the complete path traverses only free space, or when it passes through an obstacle, the value of the convolution is the minimum and maximum weight of the cells, respectively.

This optimization can be performed generating two additional maps of the environment: the optimistic and pessimistic maps [14].

- The optimistic map expands all obstacles and unknown cells in the environment to a distance equal to the inner radius of the robot shape (Fig. 2). Assuming a point robot, when the trajectory goes through at least one occupied cell in the optimistic map, the robot is guaranteed to collide when performing the same action in the real map.
- In the same way, the pessimistic map expands all obstacles and unknown zones to the outer radius of the robot (Fig. 2). When all the cells crossed by the punctual robot when performing an action in the pessimistic configuration space are free, the path is guaranteed to be collision free in the real map.

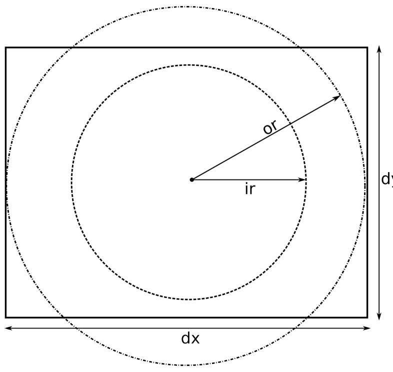


Fig. 2. The robot shape is represented by a rectangle. Its inner and outer radius (ir and or) are used to define a pair of circles that represent optimistic and pessimistic shapes of the vehicle.

Only in those cases when this first analysis does not produce a clear conclusion, it is neccesary to calculate the convolution in the occupancy map to get the transition cost of an action. As this situation is rarely produced, it performs an important saving on computation time that allows having a better real time response of the planner.

A. Heuristic

AD* is an informed discrete search algorithm. It assigns an exploration priority to each node depending on its heuristic value. A good heuristic allows finding the optimal solution of a query very efficiently, minimizing the number of nodes expanded and the computation time required.

A valid heuristic function must be always optimistic: the estimation of the traverse cost between a node and the goal must be lower than (or equal to) the real cost itself. A typical heuristic value in 2D problems is based on the euclidean distance between positions. When introducing the motion differential constraints in the search process, this value loses much of its information, as it does not consider the heading or the robot speed.

To estimate the traverse cost from a robot state to the goal, a heuristic value that considers both the differential constraints and the information about obstacles positions was used. This heuristic is the maximum of two values already introduced by Likhachev et al. in [15]: the cost from traversing between the states with the kinematic constraints and considering a free obstacle environment (FSH), and the cost to execute the same path regardless the vehicle motion model, but with the obstacles information (H2D):

$$h(s, s') = max(h_{H2D}(s, s'), h_{FSH}(s, s')) \qquad (3)$$

1) H2D: This value is calculated as the cost of the path between two positions, taking into consideration the environment cost map, but not the kinematic constraints of the vehicle [15]. The most efficient form to calculate H2D is executing an A* search over a 8-connected 2D grid where the positions match with the robot states in the lattice. As only (x, y) positions are considered, it is necessary to define the traverse cost between them, and a heuristic value to solve the A* queries faster.

The cost of a motion between two positions for H2D is the time required to move along the straight line that connects them at the maximum linear velocity of the vehicle, multiplied by the convolution of the cost of all cells affected by the trajectory, approximating the shape of the robot with the inner circle.

As the efficiency of the planner depends largely on the efficient calculation of the heuristic values for the candidate nodes to expand, it is important to minimize the time spent in solving this A* queries. To do this, a heuristic value for H2D is defined, as the time spent traversing the straight line that connects two positions at the maximum linear velocity.

2) FSH: This value is calculated as the cost of the path fulfilling the vehicle kinematic constraints, and regardless the environment cost map i.e., assuming free space [15]. In this case, the transition cost between states is the time of executing a motion between them, given the connectivity of the canonical control set.

Calculating the FSH heuristic value has a high computational cost, because it is a problem of the same dimensionality of the motion planning. As it only depends on the construction of the state lattice and the connectivity of the canonical control set, it can be precomputed offline and stored in a Heuristic Look-Up Table [16] (HLUT). In this manner the search algorithm performance is not penalized.

To do an efficient construction of the HLUT, the number of nodes included must be minimized, at the same time that the relevance of the information in the table is kept. The information level of a FSH heuristic value for a node can be measured as the ratio between this value and a simpler heuristic, called *backup heuristic*, that is the same used to guide the H2D search process. This quotient is called *trim ratio*.

Another important optimization to keep the HLUT size as small as possible is to use the regularity of the state lattice to identify the equivalences between nodes. This provides a double benefit: not to explore the equivalent areas, saving computation time, and to reduce the number of entries in the table, saving memory.

To construct the HLUT table, all states in the lattice that match with the initial position are considered as initial state. After identifying the equivalences, for each one an exploration process in two steps is performed: Dijkstra search and HORIZON search.

First, a Dijkstra search for every starting node is performed in order to populate the HLUT in a very fast way. This step leaves gaps in the table due to high cost non explored zones surrounded by previously explored ones. The FSH value for a node not included in the table is calculated by the backup heuristic, and it underestimates the real cost of nodes in the HLUT gaps. This may affect the performance of the planner due to false leads in the search algorithm, that spends unnecessary search time dismissing them and resulting in a less predictable search time.

The second step to populate the HLUT is the HORIZON method. This algorithm executes A* queries to obtain the path cost between the initial node and the one for which the heuristic value is needed. The list of queries is sorted by *trim level*, similarly as the A* OPEN list. The initial seed of this algorithm is the list of HORIZON neighbours for all nodes explored in the previous step that do not reach the threshold *trim level*. The HORIZON neighbourhood of a state is formed by the adjacent states in a 8-connected (x, y) grid, with the same heading, linear and angular velocity.

In each iteration, the lowest priority query is popped and it is solved by A*; if its *trim ratio* is lower that the threshold trim, its neighbourhood is added to the query list. The process stops when there are not more queries below the threshold *trim level* in the HORIZON list.

V. Motion primitives

The primitive trajectories that determine the lattice connectivity are calculated exploring the reachability of all the states in the lattice into a given levels of neighbourhood [1].

Efficient representation of the trajectories allows to improve the flexibility while minimizing the number of parameters. A reduced number of parameters makes simpler the generation of the primitives. A trajectory can be defined with its linear and angular velocity profiles.

1) Linear Profiles: for most cases, defining the linear velocity with a trapezoidal shape is enough to ensure a good connectivity between states. Each profile is defined by a set of parameters:

$$p_v = [v_0, a_0, v_t, v_f, a_f, t] \tag{4}$$

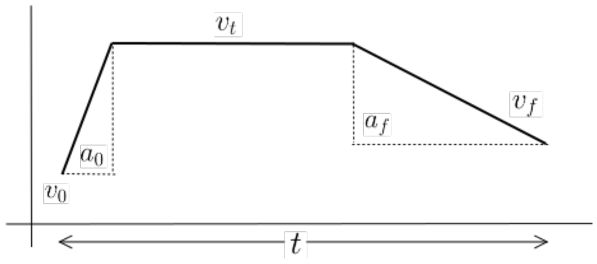


Fig. 3. Trapezoidal control profile.

where v_0 and a_0 are the initial speed and acceleration, v_t the traverse velocity, v_f and a_f the final speed and acceleration, and t the duration. As the initial and final accelerations are free parameters, sharper or smoother controls can be represented (Fig. 3).

2) Angular Profiles: The shape of the motion primitives depends largely on the flexibility allowed on angular velocity. The more the flexibility in the profiles, the greater the connectivity between states. As the number of parameters influences in the complexity of the construction of the canonical control set, it is desirable to minimize the number of parameters, keeping the flexibility in the shape of the motion primitives. A highly efficient representation of the angular velocity control is a polynomial function (figure 4).

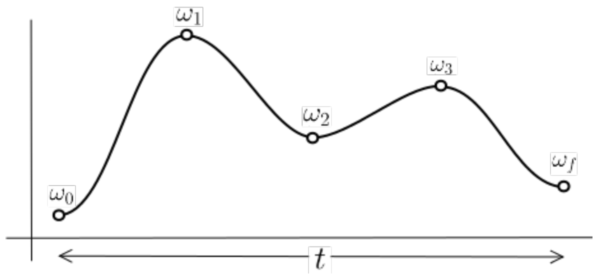


Fig. 4. Example of angular profile

This function can also be substituted by a spline, where the known parameters are the knot points (ω_1, ω_2, ω_3, ...), and the unknown values are obtained by spline interpolation. One of the benefits of using a spline function instead polynomials is that the knot points have roughly equal scale, unlike the polynomial coefficients, which is more desirable to execute numerical optimization methods over the parameters that represents the profile. The knot points are equally spaced, so the only additional parameter is the duration of the profile (t):

$$p_\omega = [\omega_0, \omega_1, ..., \omega_f, t] \tag{5}$$

A. Motion primitives generation

The motion primitives generator was implemented with an architecture that allows to separate the trajectory construction, the motion prediction and the simulation of the robot motion model (figure 5). Given the initial and final robot states to be connected, and the parametrization of the control profiles, the algorithm finds the set of optimal control orders that connects them, if it exists.

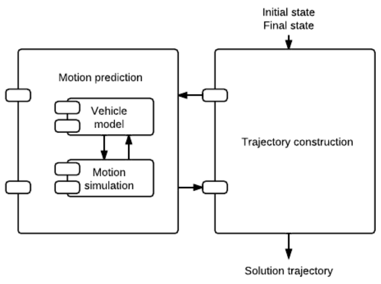


Fig. 5. The method to generate the motion primitives separates the numerical method used to optimize the trajectories, the motion equations of the vehicle and the motion model. This allows to use the same algorithm to generate motions for different vehicles.

The generation of the trajectories is based on the Newton-Raphson numerical optimization method [17]. The algorithm modifies iteratively a vector with the parameters of the control profiles, $p = [p_v, p_\omega]$, until the final state of the simulated trajectory, $s(t_f)$, satisfies the constraint equations, $C(s)$.

A pair of vectors is formed with the information of the initial and final states of the trajectory to find:

$$s_0 = [x_0, y_0, \theta_0, v_0, \omega_0] \tag{6}$$

$$s_f = [x_f, y_f, \theta_f, v_v, \omega_f] \tag{7}$$

As the stop condition of the algorithm is that the final state of the simulated trajectory and the desired final state are equal, the following constraint equations are used:

$$C(s(t_f)) = \begin{bmatrix} x_f - x(t_f) \\ y_f - y(t_f) \\ \theta_f - \theta(t_f) \\ v_f - v(t_f) \\ \omega_f - \omega(t_f) \end{bmatrix} = 0 \tag{8}$$

To apply the Newton-Raphson method it is necessary to define the Jacobian matrix that contains the information of how the variables included in the robot state change with respect to every parameter used as input.

In many cases, as this, it is not possible to obtain the Jacobian matrix (J) in analytic form from the equations of motion, and it is necessary to estimate it numerically. The partial derivatives were obtained using the central difference linearization.

$$J_{i,j} = \frac{\partial \Delta s_i(p)}{\partial p_j} = \frac{s_i(p_j + e, p) - s_i(p_j - e, p)}{2e} \tag{9}$$

Following the Newton-Raphson method, the Jacobian is inverted in order to find a correction in the parameter vector, p, that minimizes the constraint error, $\Delta s_f(p)$.

$$\Delta p = -\left[\frac{\partial \Delta s_f(p)}{\partial p}\right]^{-1} \Delta s_f(p) \tag{10}$$

The correction magnitude of the parameter vector is linearly dependent with the error magnitude. It assures a fast convergence of the method in the first iterations, and a finer correction of the parameters when they are closer to the solution.

B. Motion model

The equations that define the motion model are not completely accurate, since they do not take into consideration physical events as the inertia. When estimating future states of the vehicle in a real problem, inertia plays an important role in the difference between the theoretical path and the real one.

Improving the ability to predict future states produces important benefits in motion planning. Using the acceleration model of the vehicle [18] gets a more realistic state transition cost when the vehicle must operate with control orders formed by different velocity profiles.

The acceleration in consecutive time steps can be expressed as:

$$a_{v,t} = \beta_1 v_t + \beta_2 u_{v,t} + \beta_3 \tag{11}$$

$$a_{\omega,t} = \gamma_1 \omega_t + \gamma_2 u_{\omega,t} + \gamma_3 \tag{12}$$

where β and γ are parameters of the vehicle acceleration model and $u_{v,t}$ and $u_{\omega,t}$ are the control order at time t. Finding the parameters that minimize the squared difference between the two sides of Eqs. 11 and 12 minimizes the one step prediction error.

Extending the expressions in Eqs. 11 and 12 to the h previous time steps, the acceleration model considers the information about the robot state between $t-h$ and t to calculate its prediction. As the intermediate states used by the model to have a prediction depends on the model parameter themselves, the update equations become non linear. An approximation is done in this point obviating this dependence to have a linear model.

The algorithm used to find the parameters of the motion model is an iterative process that uses the one step simulation to generate the intermediate accelerations between t and $t+h$, followed by a least squares solving step to update the parameters values. The complete process is detailed in Algorithm 1.

Algorithm 1 Acceleration model learning

```
1: obtain initial $B$ and $\Gamma$ minimizing with least squares the one-step acceleration prediction error
2: **while** $|B^{i+1} - B^i| > \epsilon$ or $|\Gamma^{i+1} - \Gamma^i| > \epsilon$ **do**
3:     for all actual states in $t = 1...T$ obtain with the current model the simulated ones $\hat{v}_{t+h|t}$ in $h = 1...H$
4:     minimize by least squares the $h$-step acceleration prediction error to obtain $\bar{B}$ and $\bar{\Gamma}$:
       $arg\ min_B \sum_{t=1}^{T-H} \sum_{h=1}^{H} \| \sum_{\tau=0}^{h-1} (a_{v,t+\tau} - (\beta_1 \hat{v}_{t+\tau|t} + \beta_2 u_{v,t+\tau}) + \beta_3) \|^2$
       $arg\ min_\Gamma \sum_{t=1}^{T-H} \sum_{h=1}^{H} \| \sum_{\tau=0}^{h-1} (a_{\omega,t+\tau} - (\gamma_1 \hat{v}_{t+\tau|t} + \gamma_2 u_{\omega,t+\tau}) + \gamma_3) \|^2$
5:     update the model parameters:
       $B^{i+1} = (1-\alpha)B^i + \alpha\bar{B}$
       $\Gamma^{i+1} = (1-\alpha)\Gamma^i + \alpha\bar{\Gamma}$
6: **end while**
```

B and Γ vectors of the parameters β and γ, while α defines the update rate of the parameters between iterations.

VI. Experimental results

The motion planner has been tested in indoor environments with several space configurations of different complexity on a Pioneer P3-DX robot. The planner begins its exploration from the selected goal poses and generates the optimal solution that consists in the set of control orders to drive it to the goal.

The lattice was built with the following resolution criteria: robot states are arranged in a rectangular grid with resolution of 0.5 m in both x and y coordinates. The vehicle heading is discretized with the orientation values of the neighbours of a 16-connected grid. Three values for linear velocity were selected: 0 m/s, to allow spin motions, 0.2 m/s for complex maneuvers in a complex obstacle configuration, and 0.5 m/s as the maximum traverse velocity.

The map is updated several times per second, introducing new information received by the sensors into the existing one. When the occupancy grid map changes, the new information is used to update the relations affected, and the planner is able to replan the solution iteratively. The backwards exploration minimizes the number of nodes affected by a change in the surrounding environment of the vehicle, avoiding updating the cost for nodes closer to the goal.

The canonical control set was built using the neighbourhood distances 0, 1, 4 and 8. This allows the vehicle to turn on the spot and, also long motions up to 4 m long. To minimize the number of trajectories included in the canonical control set, the turning angle between neighbours was limited to $\pm\pi/2\ rad$ for states with neighbourhood level greater than 0. In all cases the angular speed at the beginning and the ending of the trajectory is limited to be 0 rad/s. A graphical representation of the motion primitives used in the experiments is shown in Fig. 6.

As the size of the primitive motion set directly influences the real-time response of the planner, it is necessary to reduce the number of motions by eliminating the redundant ones. To achieve this, the canonical control set is filtered by removing all the trajectories that can be obtained as a combination of other simpler ones. In complex motion models with a large neighbourhood, the reduction can be quite significant, while keeping all representative motions between states. In the motion primitives in the experiments, this process saves up to the 9.5% of its size, as shown in table I.

TABLE I
Motion primitives set size

Level	Generated	After filtering
0	160	160
1	224	208
4	144	144
8	144	96
Total	672	608

After the obtention of the motion primitives set, the HLUT is generated to store the representative FSH values. The Dijkstra construction step is limited by cost. For this motion model, it is reasonable to initially populate the HLUT with only those states reachable at a maximum cost of 25. The HORIZON step continues filling the HLUT with nodes whose *trim ratio* is lower than 0.8. This ensures a smooth transition between the FSH between HLUT and the *backup heuristic*. The obtained HLUT is populated with 1,136,997 entries. Symmetries of the

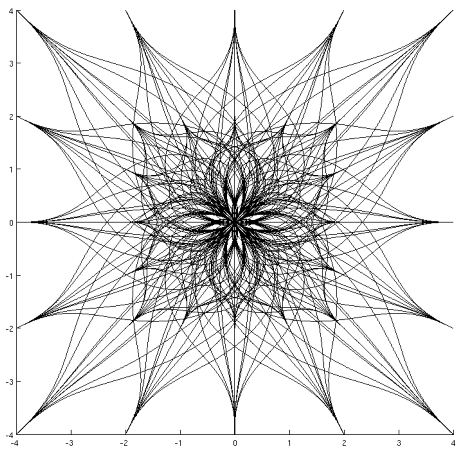


Fig. 6. Lattice control set used in the experiments for the states in the neighbourhood levels: 0, 1, 4 and 8.

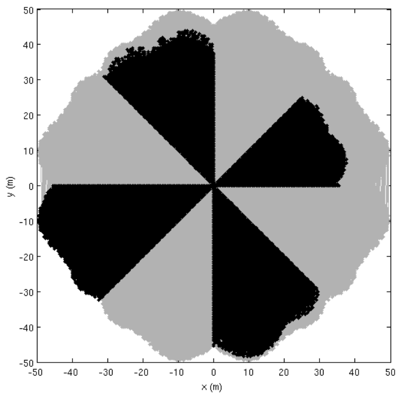


Fig. 7. Spatial representation of the information in the HLUT (gray), and subset really stored (black) for a trim ratio of 0.8.

motion model of the vehicle (Fig. 8) are considered to avoid filling the table with redundant information. This optimization is important to save computation time in the construction of the HLUT, and to reduce the number of entries stored. The information included in the HLUT is visually depicted in Fig. 7. The black dots are the entries stored in HLUT, and the gray ones represent all the states that are really represented in the HLUT through transformations of the HLUT entries (symmetric transformations, Fig. 8).

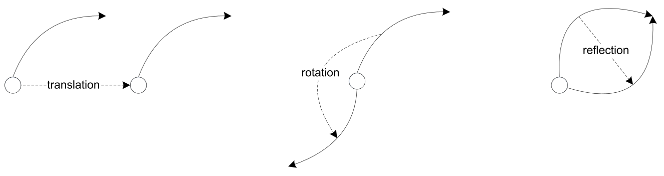
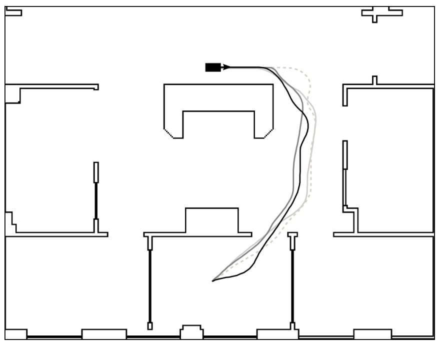


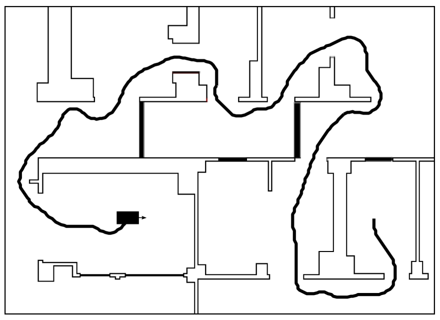
Fig. 8. Symmetries taken into account in the motion model.

The planner is typically able to update the solution fast enough to guarantee that the solution executed by the controller is updated considering the last information of the map. As the vehicle is moving the environment information is updated, and the planner changes the solution previously calculated adapting it to the new situation. The execution time for the calculation of the first solution depends on the suboptimality level chosen, the complexity of the configuration space, and the length of the path. Fig. 9 shows two motion planning examples for different suboptimal bound values (ϵ).

The results of each of them are described in table II. We include the level of suboptimality ($\epsilon = 1$ means optimal path), the time spent for planning, the number of nodes that were expanded, the time to traverse the corresponding path (cost), and the obtained path length. For the first example, the planning time is quite short, and it can be seen that refinement of the trajectory (for decreasing values of ϵ) slightly improves the obtained path. Also, the final path is longer than the previous one, but the cost (time to traverse it) was slightly better. The second example is a really complex motion planning problem. However, the algorithm obtained a first solution in 1.7 s. At that time the robot could start to move

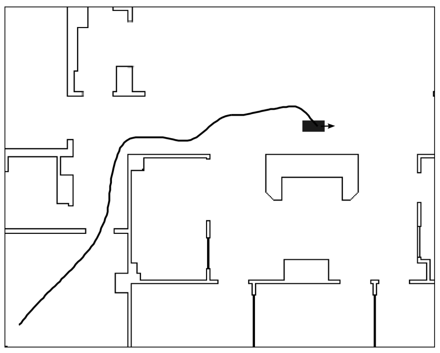


(a) Different suboptimal bounded paths: optimal path (black), 1.25 (grey), 1.5 (light grey) and 2 (dashed)

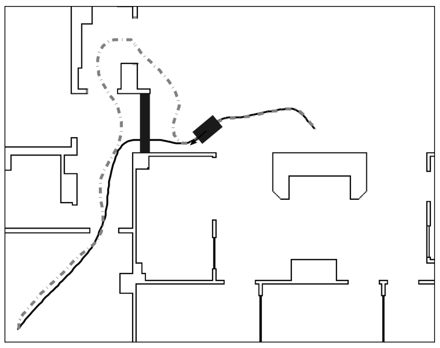


(b) A complex and optimal path (ϵ = 1).

Fig. 9. Examples of initial solutions obtained by the planner.



(a) Initial path.



(b) Path replan (dashed) when a closed door locks the initial path (black).

Fig. 10. Replanning ability when a new obstacle appears on the planned trajectory.

TABLE II
EXECUTION DETAILS OF THE EXPERIMENTS

Problem	Execution details			Solution	
	ϵ	Time (ms)	Expansions	Time (s)	Length (m)
Fig. 9(a)	1.00	1,538	372	29.88	14.96
	1.25	628	110	30.38	14.50
	1.50	480	46	31.23	14.39
	2.00	271	21	32.35	15.67
Fig. 9(b)	1.00	11,071	33,463	126.64	59.59
	1.25	6,185	9,439	132.95	62.85
	1.50	2,714	2,870	140.61	64.41
	2.00	1,705	760	153.43	69.60

and, also, continues to improve the solution. The optimal path required the expansion of many nodes, but it was obtained in a reasonable time. The resulting trajectory has a length of around 60 m and the robot needed around 127 s to traverse it.

The time spent for replanning depends on the number of changes in the environment and, also, if they are important for the current trajectory. Fig. 10 shows how the algorithm replans when a new obstacle suddenly appears in the trajectory. The time for replanning in this example is also shown in Fig. 11. This Fig. represents the relevancy of the changes in the map and the corresponding replanning times. Relevancy is calculated summing the proximity of all changed relations to the solution and normalizing the obtained values. The algorithm needs approximately 0.5 s to get the new optimal path (suboptimal paths are obtained faster). The other replanning actions are due to discrepancies between the previous map and the new sensor readings, and are solved very fast.

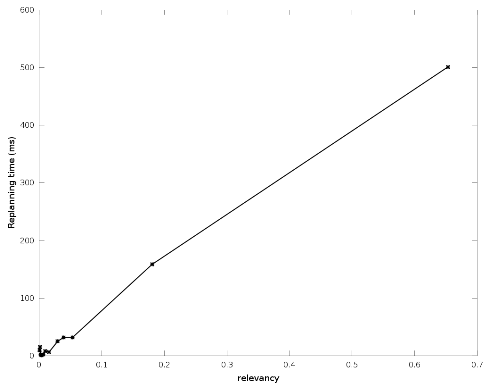


Fig. 11. Replanning times depending on the relevancy of the environment changes for situation in Fig. 10(b).

VII. CONCLUSIONS

An approach to solve the motion planning problem using a discrete and regular representation of the search space (state lattice) has been presented. The AD* algorithm was selected to implement the search, in order to exploit its anytime and replanning characteristics. Moreover, the planning algorithm takes into account the kinematic restrictions of the robot, as the motion primitives are the arcs of the graph. These motion primitives were obtained through a learning process of the motion model and the optimization of the control profiles. Results on a Pioneer P3-DX have shown the ability of the planner to get trajectories in very complex environments, obtaining a first solution very fast and, then, refining the path until the optimal solution is returned. Moreover, the replanning times let the robot to obtain new collision-free trajectories in real time when new obstacles appeared in the original trajectory.

VIII. ACKNOWLEDGEMENTS

This work was supported by the Spanish Ministry of Economy and Competitiveness under grants TIN2011-22935 and TIN2011-29827-C02-02. Adrian González-Sieira is partially supported by the Provincial Council of A Coruña. Manuel Mucientes is supported by the Ramón y Cajal program of the Spanish Ministry of Economy and Competitiveness.

REFERENCES

[1] M. Pivtoraiko, R. A. Knepper, and A. Kelly, "Differentially Constrained Mobile Robot Motion Planning in State Lattices," *Journal of Field Robotics*, vol. 26, no. 3, pp. 308–333, 2009.
[2] M. Likhachev, D. Ferguson, G. Gordon, A. Stentz, and S. Thrun, "Anytime dynamic A*: An anytime, replanning algorithm," in *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, 2005, pp. 262–271.
[3] S. LaValle, *Planning algorithms*. Cambridge Univ Pr, 2006.
[4] O. Khatib, "Real-time obstacle avoidance for manipulators and mobile robots," *The international journal of robotics research*, vol. 5, no. 1, pp. 90–98, 1986.
[5] J. Barraquand, B. Langlois, and J.-C. Latombe, "Numerical potential field techniques for robot path planning," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 22, no. 2, pp. 224 –241, 1992.
[6] R. Simmons, "The curvature-velocity method for local obstacle avoidance," in *IEEE International Conference on Robotics and Automation. Proceedings.*, vol. 4, 1996, pp. 3375 –3382 vol.4.
[7] D. Fox, W. Burgard, and S. Thrun, "The dynamic window approach to collision avoidance," *IEEE Robotics Automation Magazine*, vol. 4, no. 1, pp. 23 –33, 1997.
[8] O. Brock and O. Khatib, "High-speed navigation using the global dynamic window approach," in *IEEE International Conference on Robotics and Automation. Proceedings.*, vol. 1, 1999, pp. 341 –346 vol.1.
[9] R. Philippsen and R. Siegwart, "Smooth and efficient obstacle avoidance for a tour guide robot," in *IEEE international conference on robotics and automation*, vol. 1. Citeseer, 2003, pp. 446–451.
[10] C. Stachniss and W. Burgard, "An integrated approach to goal-directed obstacle avoidance under dynamic constraints for dynamic environments," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, vol. 1, 2002, pp. 508 – 513 vol.1.
[11] C. Urmson, C. Ragusa, D. Ray, J. Anhalt, D. Bartz, T. Galatali, A. Gutierrez, J. Johnston, S. Harbaugh, W. Messner *et al.*, "A robust approach to high-speed navigation for unrehearsed desert terrain," *Journal of Field Robotics*, vol. 23, no. 8, pp. 467–508, 2006.
[12] S. LaValle and J. Kuffner, "Rapidly-exploring random trees: Progress and prospects," 2000.
[13] M. Likhachev, G. Gordon, and S. Thrun, "ARA*: Anytime A* with provable bounds on sub-optimality," *Advances in Neural Information Processing Systems (NIPS)*, vol. 16, 2003.
[14] D. Ferguson and M. Likhachev, "Efficiently using cost maps for planning complex maneuvers," *Lab Papers (GRASP)*, p. 20, 2008.
[15] M. Likhachev and D. Ferguson, "Planning Long Dynamically Feasible Maneuvers for Autonomous Vehicles," *The International Journal of Robotics Research*, vol. 28, no. 8, pp. 933–945, 2009.
[16] R. Knepper and A. Kelly, "High Performance State Lattice Planning Using Heuristic Look-Up Tables," *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 3375–3380, 2006.
[17] T. M. Howard and a. Kelly, "Optimal Rough Terrain Trajectory Generation for Wheeled Mobile Robots," *The International Journal of Robotics Research*, vol. 26, no. 2, pp. 141–166, 2007.
[18] P. Abbeel, "Apprenticeship Learning and Reinforcement Learning with Application to Robotic Control," Ph.D. dissertation, 2008.