HEURISTIC COMPARISON FOR REAL-TIME LOCAL PLANNER IN CONTINUOUS ENVIRONMENT

I. PROBLEM DESCRIPTION

Within the robotics community, there is a great need for fast path planning algorithms, which is justified by societal constraints like road safety and road-space efficient use. Mobile robot applications impose strict time and resource constraints on a motion planner. Due to time constraints, most online planners operate by considering a relatively small set of possible actions over which to find the best choice before the next execution deadline arrives. As failure to generate a feasible path within the given deadline may result in damage to the robot or worse to human life, it is essential that the path planning method returns a path quickly and at the same time with sufficient lookahead.

Our intent is to combine optimal path planner with the idea developed over recent path planning research projects that precomputes a set of trajectories instead of generating them on the fly. By comparing different heuristic using, or not, information about the optimal path we will assess the effectiveness and the utility of such planners.

II. RELATED WORK

There have been several successful approaches to path planning over the past two decades, which can be differentiated into two categories: offline global planners and online local planners.

A. Offline path planning

Popular sampling-based planning algorithms such as Probabilistic Roadmaps (PRM [1]) and Rapidly-exploring Random Trees (RRTs [2]) and their variants ([3], [4]) use random sampling in high-dimensional search spaces to build graphs and trees for path planning. One of the key issues with the basic RRT algorithm is that it does not take into account the holonomic constraints of the vehicle. For example, the global path planning approach may find the optimal path that the robot must follow. But the computed path may include parts that are impossible for the robot to actually perform. Another issue with global offline path planning is that they are based on an a priori knowledge of the entire environment (from the start to the goal), whereas the robot might have limited range sensors that only allow it to know its local environment, making the global path computed useless.

B. Real-time path planning

An interesting approach using RRTs in real-time planning was developed by Frazzoli & Al [5]. At each planning step, it samples states in the space of the controller inputs and feeds them to a model of the vehicle that then predicts the vehicle behaviour. These trajectories are checked for collision avoidance and costs are computed in the 2D environment space to expand the tree of the RRT.

While this approach was shown to be successful, some research has been done using a precomputed set of trajectories during the planning step, avoiding the need for a trajectory predictor on the online computation. One of these approaches is the local planner proposed by Knepper & Al [6] which introduced a new real-time local planner. We implemented this planner and compared several planning heuristics, some using only information about the location of the robot with respect to the goal and some using additional information about the optimal path. The Real-Time Contextual Local Planner algorithm operates as follow. It uses a large set of trajectories which are computed offline and modified based on the state of the robot (position, orientation) at each step. First, the trajectories are tested for collision avoidance i.e. trajectories that collide with an obstacle or are too close to an obstacle are deleted. The remaining trajectories are then sorted in to equivalence classes depending on their location i.e. any two trajectories that are separated by at most the dimensions of robot are deemed equivalent. By connecting equivalent classes one by one in an abstract path space, a graph is created that consists of a set of unconnected sub-graphs. Each sub-graph corresponds to an equivalence class of trajectories. Figure 1 illustrates the correspondence between equivalent trajectories and the graph in the trajectory space. Between two re-plan cycles, the robot moves by only a portion of the path computed during the first planning and it is not assured that the next Best trajectory will be related to the previous one. This induces a loss of efficiency. Therefore the planner includes a memory of previous trajectories and their classes in order to introduce the idea of logical succession of trajectories. This process is known as the multi-stage path selection algorithm. Finally, the paths are classified as wide or narrow (depending on the number of paths in a class) and progressing or not towards the goal. The best path is then computed based on a heuristic function from a set $S$ of trajectories for consideration. The net order of preference for this set is as follow: set containing

1) All wide, progressing, logical successor classes.
2) Any wide, progressing class.
3) All narrow, progressing, logical successor classes.
4) Any narrow, progressing class.

The first part of our work was to implement the planner described before: we developed a 2-stage local planner (one level of memory for the previous classes of trajectory) with 111 pre-computed trajectories. This is a small number, but was sufficient for our approach and allowed real-time visualization. We chose a simple bicycle model for the dynamics of the robot, with a constant speed and fixed maximum curvature.

III. APPROACH

Our approach aims at combining the efficiency of a global planner such as RRT, which may not necessarily take into account the dynamic constraints of the vehicle (holonomic constraints, speed, etc.) with the more recent approach developed by Knepper & Al. ([6]) which introduced a new real-time local planner. We implemented this planner and compared several planning heuristics, some using only information about the location of the robot with respect to the goal and some using additional information about the optimal path. The Real-Time Contextual Local Planner algorithm operates as follow. It uses a large set of trajectories which are computed offline and modified based on the state of the robot (position, orientation) at each step. First, the trajectories are tested for collision avoidance i.e. trajectories that collide with an obstacle or are too close to an obstacle are deleted. The remaining trajectories are then sorted in to equivalence classes depending on their location i.e. any two trajectories that are separated by at most the dimensions of robot are deemed equivalent. By connecting equivalent classes one by one in an abstract path space, a graph is created that consists of a set of unconnected sub-graphs. Each sub-graph corresponds to an equivalence class of trajectories. Figure 1 illustrates the correspondence between equivalent trajectories and the graph in the trajectory space. Between two re-plan cycles, the robot moves by only a portion of the path computed during the first planning and it is not assured that the next Best trajectory will be related to the previous one. This induces a loss of efficiency. Therefore the planner includes a memory of previous trajectories and their classes in order to introduce the idea of logical succession of trajectories. This process is known as the multi-stage path selection algorithm. Finally, the paths are classified as wide or narrow (depending on the number of paths in a class) and progressing or not towards the goal. The best path is then computed based on a heuristic function from a set $S$ of trajectories for consideration. The net order of preference for this set is as follow: set containing

1) All wide, progressing, logical successor classes.
2) Any wide, progressing class.
3) All narrow, progressing, logical successor classes.
4) Any narrow, progressing class.

The first part of our work was to implement the planner described before: we developed a 2-stage local planner (one level of memory for the previous classes of trajectory) with 111 pre-computed trajectories. This is a small number, but was sufficient for our approach and allowed real-time visualization. We chose a simple bicycle model for the dynamics of the robot, with a constant speed and fixed maximum curvature.
Approximate trajectory selection algorithm

\[ P_{\text{free}} \leftarrow \text{Test\_All\_Paths}(w,x,P) \]
\[ \text{if } L \neq \emptyset \text{ then} \]
\[ C \leftarrow \text{Get\_Successors}(L) \]
\[ (W_s, N_s) \leftarrow \text{Sort\_Classes}(C) \]
\[ \text{end if} \]
\[ E \leftarrow \text{Equivalence\_classes}(P_{\text{free}}) \]
\[ (W, N) \leftarrow \text{Sort\_Classes}(E) \]
\[ \text{if } W_s \neq \emptyset \text{ then} \]
\[ S \leftarrow W_s \]
\[ \text{else if } W \neq \emptyset \text{ then} \]
\[ S \leftarrow W \]
\[ \text{else if } N_s \neq \emptyset \text{ then} \]
\[ S \leftarrow N_s \]
\[ \text{else} \]
\[ S \leftarrow N \]
\[ \text{end if} \]
\[ p \leftarrow \text{heuristic}(x,S) \]

w: local environment map
x: initial state
P: predefined set of paths
L: equivalence class of path selected in prior call
p: selected path

IV. Evaluation

We tested our planner on several environments which differed by the density of obstacles. The different values chosen for obstacle density were 5%, 10%, 25% and 50% of the total surface of the environment. Several environments were generated for each density by randomly positioning obstacles. We assessed the efficiency of different heuristics using two criteria: the computation time per iteration of the algorithm and the total length of the path travelled. The results are represented in Figure 2 and 3.

Fig. 2. Time per iteration for different density of obstacles.

Fig. 3. Cost of the path travelled by the robot for different density of obstacles.

- **Optimal Path oriented**: This does not use the consideration set. The chosen path belongs to the class of trajectories that minimizes the mean distance to the goal. The heuristic chooses the trajectory that ends up closest to the optimal path from the robot to the goal instead of choosing the path that leads it closest to the goal.

- **Optimal Path oriented + Goal oriented class**: This uses the set of consideration trajectories described before and reduces it (narrow/wide, progressing, etc.) before computing the closest to the optimal path.

Regarding the run-time, all heuristics except the Goal oriented heuristic seem to increase with the density of obstacles. These two increase at nearly identical rates. The GOC heuristic seems to compute much faster than the others for medium densities but the GO is also really efficient for small and high densities.

Regarding the cost, we can see in Figure 3 that the total length of the path increases with the density of obstacles, which is obvious because with increase in obstacle density the complexity of the path required to reach the goal increases which thus
means that the length of the path increases. The Goal oriented class heuristic and the Optimal path oriented heuristic have similar costs whereas the Goal oriented heuristic is always more expensive (even if the difference is converging to zero when the density of obstacles increases) and the OPO + GOC heuristic is less expensive for large (≥10%) density of obstacles. As the robot has a constant speed, path length is also proportional to the total time travelled and indirectly to computational time because our planner and our controller are not completely asynchronous. An additional feature to take into account for the evaluation of these heuristics is that some of them can sometimes fail. The proportion of failures seems to increases as the heuristic is differentiating from the GO heuristic and increases as the density of obstacles increases.

V. DISCUSSION

There is a great similarity in the cost function of the GOC and OPO heuristic. We explain this behaviour by the greedy nature of both these approaches: GOC directly by specifying a consideration set among the most greedy trajectories, OPO indirectly because it seeks to follow the greediest path. As GOC+OPO comes from the fusion of these two heuristics, it is normal to find the same asymptotic behaviour but with a lower start because it profits from the advantages of both heuristics.

We also observed that the GOC heuristic was less time consuming. We can explain that by the fact that GOC reduces the consideration set so less trajectories to compare at each iteration. The difference with GO decrease with the density of obstacles because when there are a lot of obstacles, the consideration set decreases because of the increase in the number of equivalence class: the consideration set contains only the best equivalence classes, therefore when there are lots of equivalence classes, these best classes are less populated and so is the consideration set.

When the density of obstacles decreases, the influence of class oriented heuristics decreases because the number of classes becomes very small: for density of 5% or 10% there is at most 3 or 4 classes of equivalences, and often only 1 global class of equivalent paths. This explains why GO/GOC and OPO/OPO+GOC are closer for low densities.

It has come to our attention that OPO heuristic present the drawback that, if the robot finds itself oriented back to the optimal path then the best path regarding this heuristic (the closest to the optimal path) might be one that goes in the opposite direction. However, this case does not seem to appear really often. We might want to correct the implementation in order to avoid this specific case.

Overall, it appears that the OPO+GOC heuristic gives good results regarding the most important criteria, which is the length of the global path travelled by the robot. Compared to the original heuristic (GO) presented by Knepper & Al., we have improved the planner energy efficiency by about 15%, which is a nice result regarding the complexity of this planner.

Among the numerous nice features of the Real-Time Conceptual Local Planner developed by Knepper & Al. is the fact that the planner is proven to be complete regarding the resolution of the path planning problem (if there is a path it will find it). Even if our approach seems to be efficient for several kind of environments, there are still obstacles configurations for which some of our heuristics can fail. However, we believe that this drawback might be due to some implementation problem (and not to deficiencies of the heuristic algorithm) that could be corrected in order to recover the completeness property of the planner.

Among the possible other ameliorations that could be done, an interesting improvement would be to use a global planner that uses a dynamic model of the vehicle to generate the optimal path instead of the A* algorithm that we are currently using. We already implemented an RRT* algorithm for this improvement but did not find time to link it to the rest of the project.

REFERENCES