FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

Meta-Heuristic Extended Local Planning

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Computational Intelligence

eingereicht von

Markus Suchi
Matrikelnummer 9103533

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung: o.Univ.-Prof. Dipl.-Ing. Dr. techn. Markus Vincze
Mitwirkung: Dr. Markus Bader

Wien, TT.MM.JJJJ

(Unterschrift Markus Suchi)

(Unterschrift Betreuung)

Erklärung zur Verfassung der Arbeit

Markus Suchi
Karl Heinz Straße 67/11/51, 1230 Wien

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit - einschließlich Tabellen, Karten und Abbildungen -, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

(Ort, Datum) (Unterschrift Markus Suchi)

Acknowledgements

Thanks Thanks Thanks Thanks Thanks Thanks Thanks Thanks Thanks Thanks Thanks Thanks Thanks Thanks. Thanks Thanks Thanks Thanks Thanks Thanks Thanks Thanks Thanks Thanks. Thanks Thanks Thanks. Thanks Thanks.

META HEURISTIC EXTENDED LOCAL PLANNING

MARKUS SUCHI

August 2012 – version 4.1

ABSTRACT

Short summary of the contents in English...

ZUSAMMENFASSUNG

Kurze Zusammenfassung des Inhaltes in deutscher Sprache...

CONTENTS

LIST OF FIGURES

LIST OF ALGORITHMS

LIST OF TABLES

ACRONYMS

Part I

INTRODUCTION

1 INTRODUCTION

General Introduction to Thesis

1.1 MOTIVATION AND APPLICATIONS

1.2 GOAL OF THESIS

1.3 OUTLINE

2 INTRODUCTION TO PATH PLANNING

General Introduction to Thesis

2.1 BASIC PATH PLANNING PROBLEM

2.2 REPRESENTATION OF THE ENVIRONMENT

2.2.1 *Decomposition*

2.3 ROBOTIC MODELS

3 PLANNING ALGORITHMS

Overview and classification of path planning algorithms

3.1 GLOBAL PLANNING

3.2 LOCAL PLANNING

Part II

META HEURISTIC EXTENDED LOCAL PLANNER

4

META HEURISTIC EXTENDED LOCAL PLANNING

4.1 META HEURISTIC EXTENDED LOCAL PLANNING

4.2 INTRODUCTION

Planning and navigation are essential for mobile robots to act in out- and indoor environments. Uncountable articles are describing approaches and applied solutions of autonomous system driving safely in different domains, e.g. Stanley [17] a self driving car that won the DARPA Grand Challenge by driving 132 miles through the Mojave desert, or the mobile robots of Kiva Systems [7] handling goods in distribution centers and warehouses like Amazon. Articles like [17] and [7] present the relationship between the environmental complexity and the computational on-board power needed to deal with it. Stanley has a six processor computing platform sponsored by Intel whilst Kiva robots are using low cost DSP's for navigation and vision processing[1] to drive within a known environment. The robot shown in Figure 1 is used on our institute. The robot is able to drive autonomous within our lab/office environment and is equipped with one Intel processor (i5@3,4GHz) running Robotic Operation System (ROS)[2]. While the application domain and computational power varies strongly between the robotic systems, all of them have to move safely and efficient from one location to another.

A common strategy to deal with the complex planning problem is the division into a global and a local planning problem [12]. Global path-planning requires a simplified representation, e.g. static map, of the search problem to efficiently compute an optimal shortest path using variants of Dijkstra's [2] or A^* [9] algorithm, ignoring kinematic and acceleration constraints of the robot. In succession the retrieved global path is used by a local planner for guiding the robot through the environment. The local planner takes sensor readings of the robot into account and is reactive to changes within the sensor range. It chooses the best values of available motor controls in respect to the kinematic and dynamic constraints of the robot. The main task is to avoid collision with obstacles, by generating feasible velocity commands to produce a trajectory for the robot near the global path. The heuristic strategy presented optimizes the local planner.

1 Kiva Systems Uses "Smart" Blackfin-powered Robots for Warehouse Navigation | Analog Devices: http://www.analog.com/en/content/kiva_systems_bf548/fca.html

2 Robot Operating System (ROS): http://ww.ros.org/

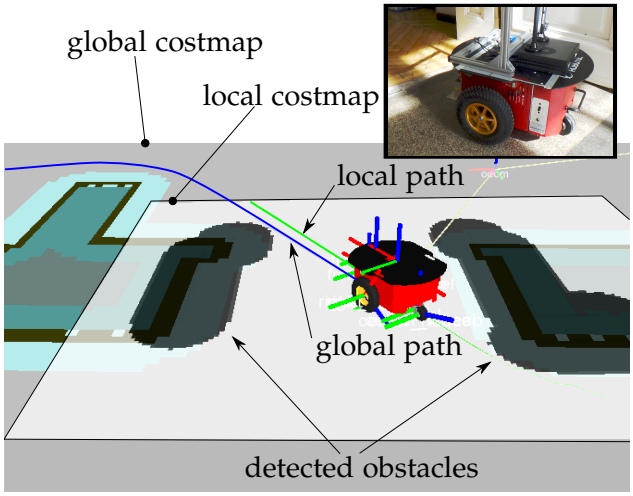


Figure 1: This figure shows a Pionner3DX and its view of the office environment while passing through a door. The blue line shows the global path and the green line the selected trajectory of the local planner.

One of the most popular local planer and reactive collision avoidance method is the Dynamic Window Approach (DWA)[4]. Its based on evaluating a fixed number of trajectory samples in a reduced velocity space. A dynamic window around the current robot fused with the current sensor readings of the robot represents a so called costmap. Trajectories are sampled into that costmap and weighted by a cost function. Recent adoptions of this method can be found in [15][14]

In this paper we propose a method which finds the best velocity commands by using search strategies based on Meta-Heuristics instead of evaluating a fixed number of trajectories in a brute force manner. A combination of Iterated Local Search (ILS) with different neighborhood structures, Variable Neighborhood Search (VNS), and a Tabu list provides a significant documented performance boost.

This enables the local planer to:

- run at a higher frequency
- simulate trajectories for a longer time interval
- moving the robot at higher speed
- investigate a larger amount of trajectories
- use a higher costmap resolution

The next section presents related work, followed by a detailed description of the approach in Section 4.4, and results of the experiments are given in Section 4.5.

4.3 RELATED WORK

Of particular interest are local planning methods which apply sampling and simulation of trajectories. Here the maximization step can easily be substituted by the proposed method. The next two section describe the recent developments of this family of planners and gives an introduction to the used Meta-Heuristic algorithm.

4.3.1 *Local Planning and Obstacle Avoidance*

A well known method for local planning is the Dynamic Window Approach proposed in [4]. The method discretely samples the velocity space (v, w) of the robot, where v is the *linear velocity* and w the *angular velocity* of the robot, to create a set of feasible trajectories. The velocity space is reduced to the reachable minimal and maximal velocity in one control cycle, taken the acceleration limits of the robot into account. For a fixed amount of velocity samples the corresponding trajectories are created using a predefined granularity by performing forward simulation for a short period of time, starting at the current position of the robot. Evaluating all trajectories with respect to a weighted cost function (cf. Equation 1) identifies the best trajectory.

$$f_c(v, w) = \alpha f_a(v, w) + \beta f_d(v, w) + \gamma f_v(v, w) \tag{1}$$

The function $f_a(v, w)$ judges the angle between the robots heading and a given goal position. It is maximal if the heading is a straight line to the goal. The distance to the closest obstacle is calculated in the function $f_d(v, w)$. The function $f_v(v, w)$ takes the forward velocity into account and rewards faster movements of the robot. This method does not use a global plan to guide the robot, so without further changes it is subject to get captured in local minima.

Other applications of this approach in recent planning systems, adapt the corresponding cost function. The excellent `move_base`[3] motion-planning framework introduced in [14] implements within the navigation stack of Robot Operating System (ROS) local planner which incorporate global plans.

One implementation is based on DWA. There is also the option to use Trajectory rollout [5] as a local planner which is very related to the DWA, but in contrast improves in simulating the robots trajectory by accurately applying acceleration limits over the whole simulation time. The cost function maximizes characteristics like proximity to obstacles, proximity to the goal, proximity to the global path, and speed. Furthermore a number of escaping strategies try to avoid the vulnerability to local minima.

3 move_base planning framework: http://wiki.ros.org/move_base

Collision detection and cost calculation is performed by using the footprint of the robot following the calculated trajectory. Hence the discretized footprint, which is usually given as a simple polygon, is projected on the costmap. Bresenham's Line algorithm [1] is used for ray-tracing the contour of a robot in the discrete workspace. Figure 2a shows the global view of the planning task. In Figure 2b the corresponding local view is depicted, including all sampled trajectories which are evaluated using a local costmap.

Concerning the optimization of the used cost functions the most common approach for DWA and related local planner is to evaluate all possible trajectories in a reduced discrete velocity space. Examples of this approach can be found in [11][14][15].

The Curvature Velocity Method in [16] considers approximation techniques like simulated annealing to maximize the cost function using the whole velocity space, but abandoned the idea due to still high computational costs. Instead of using discrete samples of the velocity space, it divides the space in sets of curvature intervals. Evaluation is performed by evaluating over all curvature intervals.

The proposed method extends the DWA approach, by using approximation algorithm to maximize the cost function in a discrete representation of the velocity space.

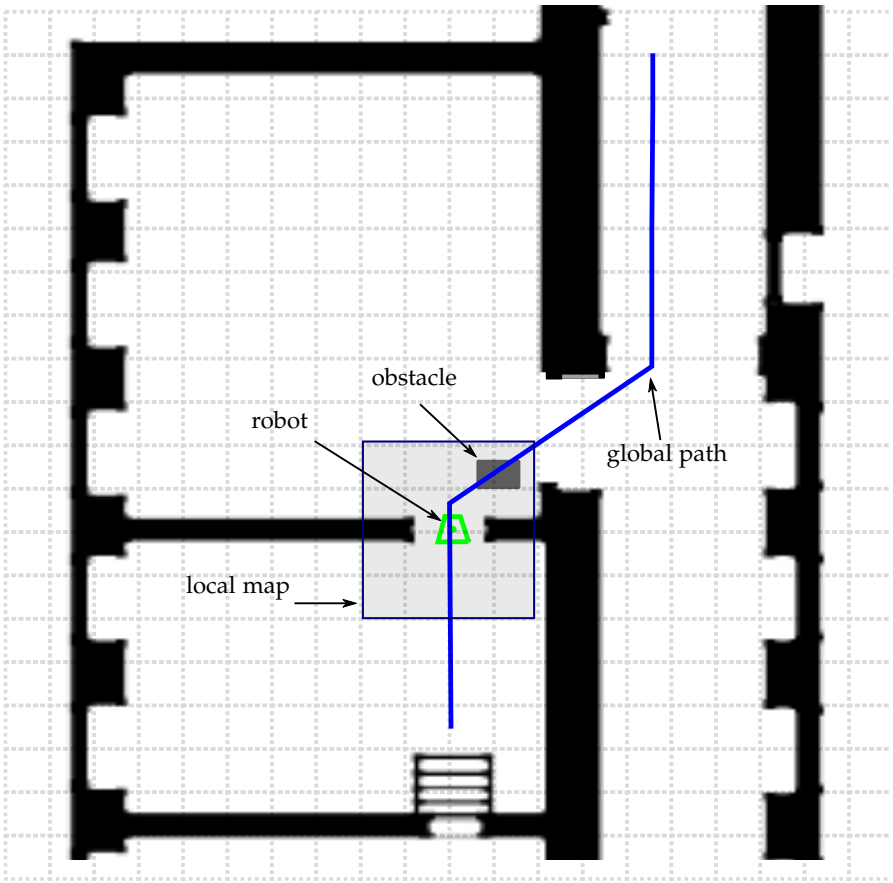
4.3.2 *Meta-Heuristic Search*

The family of algorithm using Meta-Heuristics is extremely successful in solving combinatorial problems (e.g. Traveling Salesman problem, MAX-Sat problem, scheduling problems). These algorithm build on the idea of executing multiple searches to find local optimal solutions in different parts of the search space using randomization, which together yield a global approximate optimal solution.

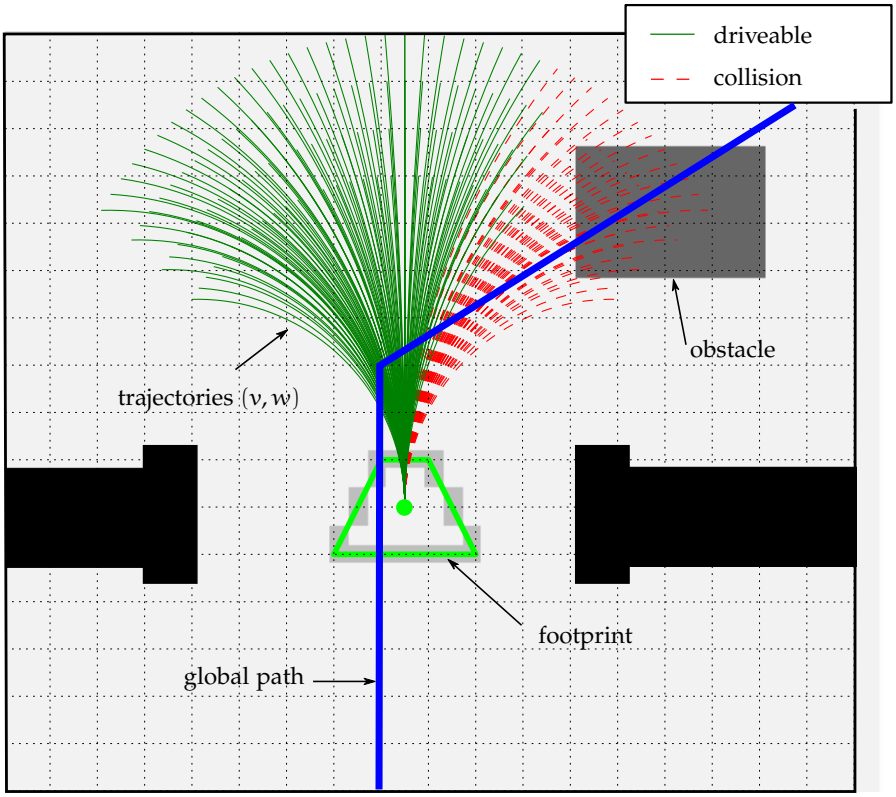
The basic Local Search (LS) (cf. Algorithm 1) finds a local optimum according to a cost function $\mathrm{cost}(x)$ and a fixed sized region around an initial solution in the solution space. The set of solutions in this region is denoted as the neighborhood of x. The necessary steps to get from the initial solution to a solution in the neighborhood is called a move.

algochapter 1 Local Search (LS)

$x \leftarrow$ initial solution
repeat
  select a $x' \in$ NEIGHBORHOOD(x)
  if $\mathrm{cost}(x') \leqslant \mathrm{cost}(x)$ **then**
    $x \leftarrow x'$
  end if
until stopping criteria satisfied



(a) Path planning in global costmap.



(b) Trajectory generation for linear, and angular velocities (v, w) in local costmap guided by global path.

Figure 2

A nice survey of Iterative Local Search including the basic algorithm (cf. Algorithm 2) can be found in [13]. The main idea is to call a local search procedure iteratively, until a certain stopping criteria is satisfied. In each iteration the current solution might be perturbed by changing parts of the solution. The following local search takes this altered solution as a starting point and returns a new solution. If it satisfies an acceptance condition (e.g. the best solution so far), the process restarts with the new solution. In addition, a history of already found solutions may be used to steer perturbation and the acceptance test.

algochapter 2 Iterative Local Search (ILS)

$x_0 \leftarrow$ initial solution
$x^* \leftarrow$ LocalSearch(x_0)
repeat
  $x' \leftarrow$ Perturbation(x^*,history)
  $x^{*\prime} \leftarrow$ LocalSearch(x')
  $x^* \leftarrow$ AcceptanceCriterion(x^*,$x^{*\prime}$,history)
until stopping criteria satisfied

Instead of using a fixed neighborhood, the Basic Variable Neighborhood Search (cf. Algorithm 3) as presented in [8] uses a neighborhood structure of possibly nested neighborhoods $N_k(x) = N_1(x), N_2(x), \ldots, N_{k_{max}}(x)$ which together are guaranteed to explore the whole solution space. In the shaking phase the algorithm chooses a random solution of the current neighborhood to avoid getting captured in local minima. If the solution found by the Local Search does not improve the next neighborhood will be considered (cf. Algorithm 4).

algochapter 3 Variable Neighborhood Search (VNS)

function VNS(x, k_{max})
  repeat
    $k \leftarrow 1$
    repeat
      $x' \leftarrow$ Shake(x, k)
      $x'' \leftarrow$ LocalSearch(x')
      NeighborhoodChange(x, x'', k)
    until $k = k_{max}$
  until stopping criteria satisfied
end function

Another successful Meta-Heuristic strategy is Tabu Search (cf. Algorithm 5) proposed in [6]. To avoid local minima a Tabu list keeps track of moves which are not allowed during the exploration of the current neighborhood. In its simplest form the Tabu list includes all visited solutions. This might be too restrictive, or the list might grow too large, hence one can restrict the list to a certain length, and delete

algochapter 4 Neighborhood Change

```
function NeighborhoodChange(x, x′, k)
    if cost(x′) < cost(x) then
        x ← x′
        k ← 1
    else
        k ← k + 1
    end if
end function
```

e.g. the oldest item in the list in each iteration. One advantage of this method is, that it can be easily combined with other Meta-Heuristic algorithm.

algochapter 5 Tabu Search

```
Tabulist ← 0
x ← initial solution
repeat
    X′ ← Neighborhood(x) ∉ Tabulist
    x′ ← best solution in X′
    Tabulist = Tabulist ∪ {x′}
    x ← x′
    if x is overall best solution then
        store x as best solution
    end if
until stopping criteria satisfied
```

4.4 APPROACH

The aforementioned trajectory selection for forward movements, evaluation and collision test are costly operations. The focus lies on improving this part of the DWA algorithm. The trajectory sampling and selection of the DWA are implemented in python minimizing a simpler cost function $f_c(v, w)$ (cf. Equation 2), where $f_g(v, w)$ is the distance of the center of the robot in the end position to a predefined goal position, and $f_o(v, w)$ is the maximal distance to an obstacle on the trajectory path.

$$f_c(v, w) = \alpha f_g(v, w) - \beta f_o(v, w) \tag{2}$$

Instead of performing an exhaustive *Brute Force* search on all velocity samples, Meta-Heuristic algorithm are here used to boost the search performance.

The reduced search space are all tuples of forward and angular velocities (v, w) within given limits $v_{min} \geqslant v \leqslant v_{max}$ and $w_{min} \geqslant$

$w \leqslant w_{max}$ and a step size for discretization by fixing the number of samples.

4.4.1 *Neighborhood and Local Search (LS)*

The neighborhood of a solution is simply defined by making a number of discretization steps to reachable regions from the current solution velocity tuple. The 4-neighborhood makes a step by either increasing or decreasing the current linear and angular velocity by one discretization step (Manhattan distance $= 1$). The 8-neighborhood takes all neighbors into account which are reachable in one discretization step (Moore neighborhood). 16-neighborhood are all neighbors reachable in two steps. This process continues until the whole search space is the neighborhood. Figure 3 visualizes the neighborhood structure.

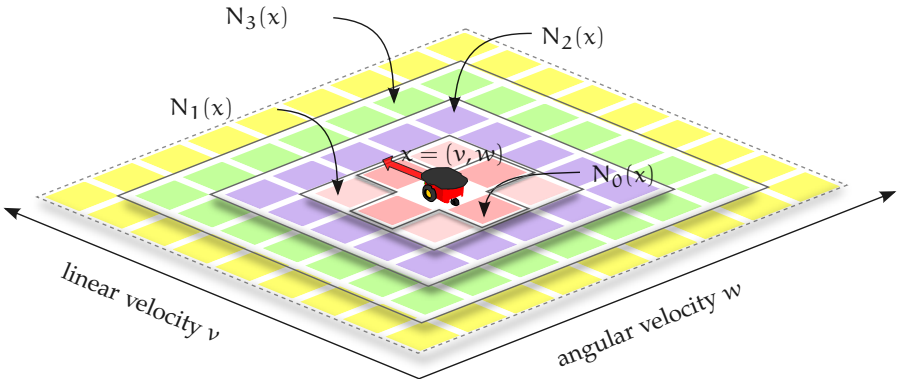


Figure 3: Neighborhood structure in the 2-dim velocity space.

Using Local Search the neighborhood is either exhaustively searched for the best solution (Best-Improvement heuristic) or stopped after finding the first improving solution (First-Improvement heuristic).

4.4.2 *Tabu List*

Instead of recording all steps made in a Local Search run, all visited states are marked as tabu and will not be considered as valid solution in future steps of the algorithm. The Tabu list is used by the other Meta-Heuristic algorithm during the Local Search.

4.4.3 *Iterated Local Search (ILS)*

The perturbation step is very simplified and just finds the next random valid velocity tuple. Instead of altering the current solution in each step, we make use of the history and apply it after a fixed amount of iterations. The algorithm can use a 4,8, and 16-neighborhood with Best Improvement heuristic for the Local Search.

4.4.4 *Variable Neighborhood Search (VNS)*

For the VNS algorithm local search is performed in one neighborhood until no improvement occurs. The shaking chooses a random solution in the current neighborhood. If the shaking does not yield any solution, because the whole neighborhood is tabu, or does not include a collision free trajectory, the next neighborhood is chosen. An ordering of neighborhoods according to their size yields the neighborhood structure $N_0(x)$ = 4-connected, $N_1(x)$ = 8-neighborhood, ..., $N_k(x)$ = k-steps reachable neighborhood. Larger neighborhoods are too costly to evaluate. Therefore the neighborhood structure is bounded above by $k_{max} = 8$. If no improvement is made up to the $N_{k_{max}}$ neighborhood, a new initial solution is generated at random. The VNS algorithm can be used with Best-, or First-Improvement heuristic for the Local Search.

4.4.5 *Experiments*

To select a benchmark cost a brute force search is performed on random generated test instances, evaluating a fixed number of trajectories. The time the algorithm needs to find this benchmark solution is used to compare their performance.

All algorithm are tested using different minimal, and maximal velocities to account for different acceleration limits. The weighting coefficients of the cost function are fixed to $\alpha = 0.01$ and $\beta = 1$. The local goal is also at a fixed location in the map. The step size of the collision test is fixed to 0.015 meter. Forward simulation time is fixed to one second.

The following 60 test instances include different obstacle counts and random placement of quadratic obstacles:

- 15 instances with 1 obstacle and side length 1 meter.
- 15 instances with 3 obstacles and side length 1 meter.
- 15 instances with 5 obstacles and side length 0.5 meter.
- 15 instances with 25 obstacles and side length 0.1 meter.

Figure 4 illustrates three random instances. A generated costmap together with a visualization of consecutive local path planning steps in a simulation is shown in Figure 5.

The following list shows the tested algorithm:

- **Random Search with Tabu List:** A repeated random guess of a velocity tuple (v, w) (RST).
- **Iterated Local Search:** Performing Iterated Local Search with 4, 8 ,and 16 neighbors and Tabu List (ILS4, ILS8, ILS16).

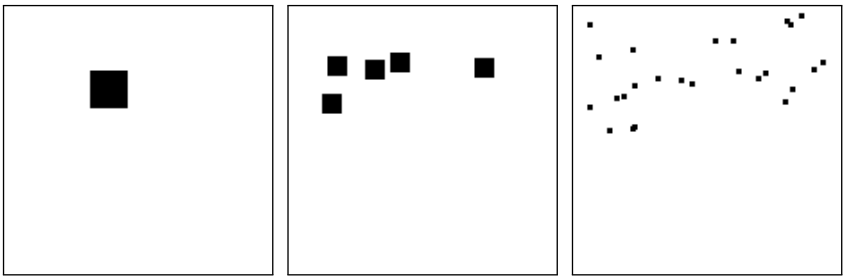


Figure 4: Figures (a)-(c) show 3 out of 60 random instances for experiments. The instances differ in number and size of obstacles and are used for local costmap creation.

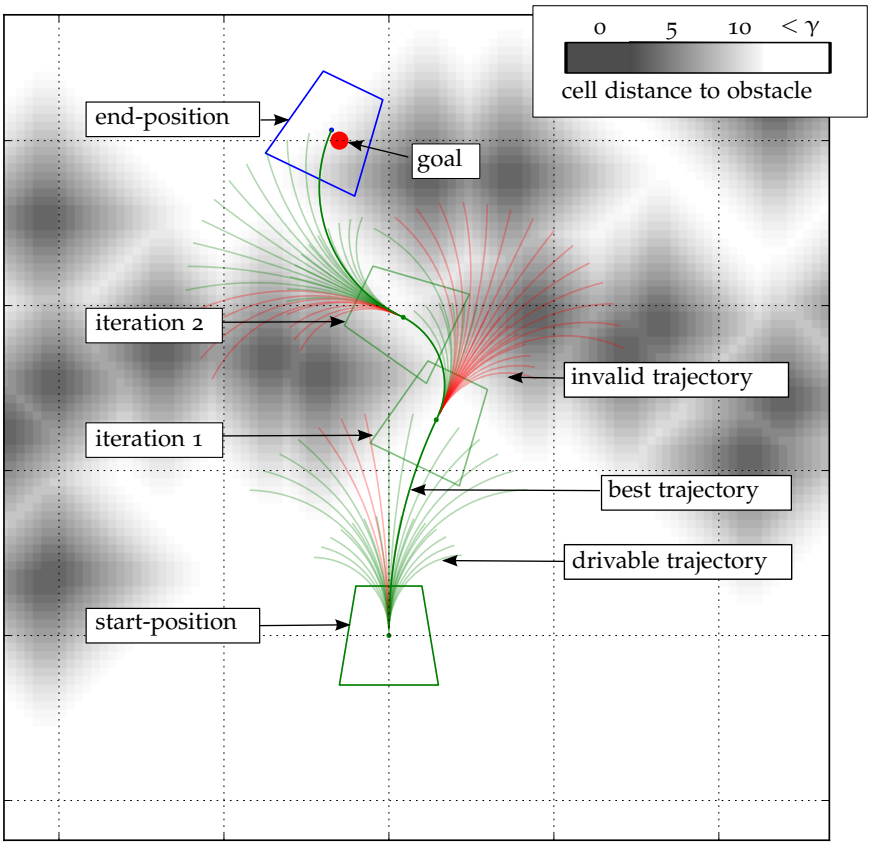


Figure 5: This figure shows three applications of the local planner in a simulated experiment. The greyscale background image visualizes obstacle costs of a generated costmap. Trajectories in green are driveable, whereas red trajectories collide with obstacles. In each local planning step possible trajectories are weighted with respect to obstacle closeness and progression towards goal destination. For example in the second step a longer valid trajectory is rejected, while a shorter trajectory which stays farther away from obstacles is selected for execution. After three local planning applications the robot safely reaches the goal destination.

- **Variable Neighborhood Search:** Variable Neighborhood search with Best-,and First-Improvement heuristic, and Tabu List (VNSB, VNSF).

4.5 TEST RESULTS

All tests were performed on a 2.4 GHz, Intel Core 2 Duo processor using 4 GB RAM.

In the first experiment the algorithm were applied to all 60 instances to evaluate a broad spectrum of possible environments. Figure 6 illustrates the results using 240 trajectories, and using 2400 trajectory samples.

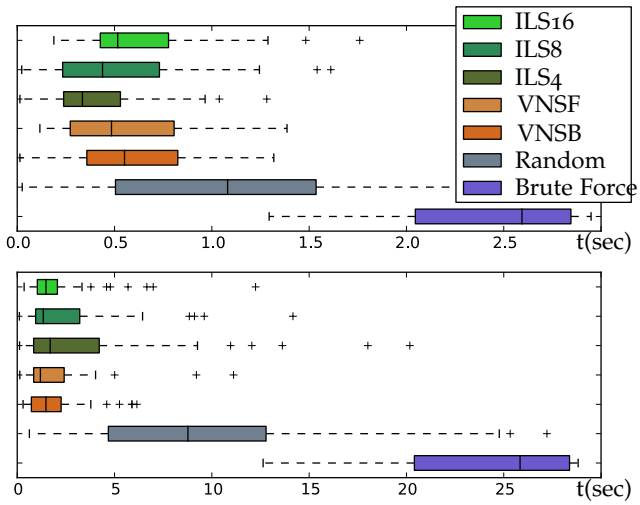


Figure 6: This figure shows the results of testing all 60 randomly generated instances. The top figure shows the run time performance for 240 trajectories, and the bottom figure for 2400 trajectories. Compared to brute force search, the Meta-Heuristic algorithm show a significant improvement.

The results show that all algorithm, including RST, outperform the Brute Force generate-and-test method significantly. As expected increasing the number of trajectories greatly favors the Meta-Heuristic algorithm, since they benefit from larger search spaces. Notice that ILS and VNS algorithm differ apparently from the RST by exhibiting much smaller variance in their test results, indicating that randomization alone is not enough to achieve very good and stable performance. Furthermore the VNS exhibit a more stable performance than the ILS methods. Comparing the ILS algorithm reveals the connection of the search space size to the size of the neighborhood. A small number of trajectories benefits smaller sized neighborhoods, whereas increasing the number of trajectories benefits larger neighborhoods.

The following tests only include the VNSF, VNSB and ILS4 algorithm. The algorithm are executed with specific world instances, and

repeated 50 times. The results in Figure 7 show again that the VNS algorithm significantly outperform the Brute Force method.

Analyzing the results of the ILS4 algorithm shows that a too small environment will quickly degrade to random search. Here the use of a neighborhood structure pays off and the VNS approaches perform evidently better than ILS. In addition, the results show that the algorithm perform good independent of number and size of obstacles.

As for nearly all optimization problems, the No Free Lunch theorems [18] apply to the local planning domain. Looking at all the results, there is no clear winner among the algorithm. Nevertheless using Variable Neighborhood search with Tabu List and Best Improvement heuristic seem to yields the best and most stable overall performance.

In general the run time of the python implementation is not very efficient compared to tuned C++ implementations. Therefore the absolute numbers of the run time evaluations should be handled with care.

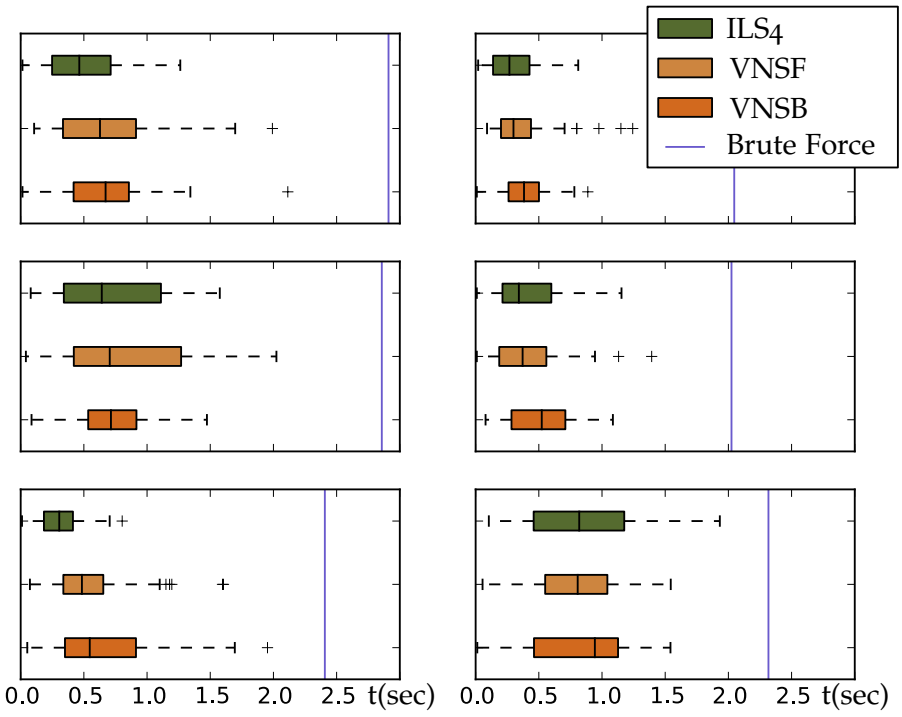
4.6 CONCLUSIONS

Applying Meta-Heuristic search to trajectory selection of local planners like DWA is a first step in using the power of these search procedures in the context of local planning. The results of our experiments in Section 4.5 show, that already the small selection using Iterated Local Search, and Variable Neighborhood Search provide significant performance improvement. Therefore it would be of interest to investigate related algorithm like GRASP [3], reduced VNS, or Simulated Annealing [10]. In addition, developing more sophisticated neighborhood structures, and extending the tabu search method would also be valuable.

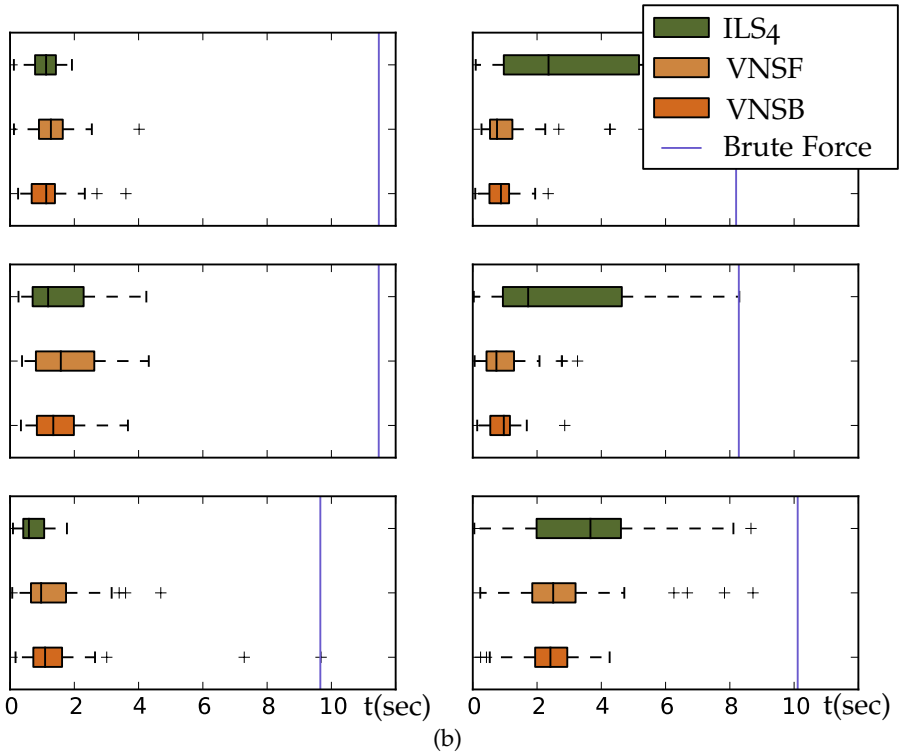
The proposed method is also applicable for robot models of higher degree of freedom, since dealing with large trajectory samples is a particular strength of Meta-Heuristic search.

The applicability to similar path-planning methods using trajectory samples, like Curvature Velocity Method [16], or the Trajectory Rollout method implemented in `move_base`, are also subject of further investigations.

One of the next goals is to integrate this trajectory selection in existing planning systems for robots using ROS and analyze the performance of the algorithm using state of the art methods for costmap calculations and collision testing. With the gained data a fine tuned local planner using Meta-Heuristic based search methods can be tested under real and simulated environments.



(a)



(b)

Figure 7: The results of 50 consecutively executions with (a) 240 and (b) 960 trajectories, on particular instances which differ in number and size of obstacles. The blue line marks the run time for brute force search, which is used as a benchmark.

BIBLIOGRAPHY

[1] Jack E Bresenham. Algorithm for computer control of a digital plotter. *IBM Systems journal*, 4(1):25–30, 1965.

[2] Edsger W Dijkstra. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271, 1959.

[3] Thomas A Feo and Mauricio GC Resende. Greedy randomized adaptive search procedures. *Journal of global optimization*, 6(2): 109–133, 1995.

[4] D. Fox, W. Burgard, and S. Thrun. The dynamic window approach to collision avoidance. *Robotics Automation Magazine, IEEE*, 4(1):23–33, Mar 1997. ISSN 1070-9932. doi: 10.1109/100.580977.

[5] Brian P. Gerkey and Kurt Konolige. Planning and control in unstructured terrain. In *In Workshop on Path Planning on Costmaps, Proceedings of the IEEE International Conference on Robotics and Automation (ICRA*, 2008.

[6] Fred Glover and Manuel Laguna. *Tabu search*. Springer, 1999.

[7] Erico Guizzo. Three engineers, hundreds of robots, one warehouse. *Spectrum, IEEE*, 45(7):26–34, 2008.

[8] Pierre Hansen, Nenad Mladenović, and José A Moreno Pérez. Variable neighbourhood search: methods and applications. *Annals of Operations Research*, 175(1):367–407, 2010.

[9] Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Trans. Systems Science and Cybernetics*, 4(2):100–107, 1968.

[10] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220:671–680, 1983.

[11] Domokos Kiss and Gábor Tevesz. Advanced dynamic window based navigation approach using model predictive control. In *Methods and Models in Automation and Robotics (MMAR), 2012 17th International Conference on*, pages 148–153. IEEE, 2012.

[12] S. M. LaValle. *Planning Algorithms*. Cambridge University Press, Cambridge, U.K., 2006. ISBN 0521862051. Available at http://planning.cs.uiuc.edu/.

[13] Helena R Lourenço, Olivier C Martin, and Thomas Stutzle. Iterated local search. *arXiv preprint math/0102188*, 2001.

[14] Eitan Marder-Eppstein, Eric Berger, Tully Foote, Brian P. Gerkey, and Kurt Konolige. The office marathon: Robust navigation in an indoor office environment. In *ICRA*, pages 300–307. IEEE, 2010.

[15] Marija Seder and Ivan Petrovic. Dynamic window based approach to mobile robot motion control in the presence of moving obstacles. In *ICRA*, pages 1986–1991. IEEE, 2007. URL http://dblp.uni-trier.de/db/conf/icra/icra2007.html#SederP07.

[16] Reid Simmons. The curvature-velocity method for local obstacle avoidance. In *Robotics and Automation, 1996. Proceedings., 1996 IEEE International Conference on*, volume 4, pages 3375–3382. IEEE, 1996.

[17] S. Thrun, M. Montemerlo, H. Dahlkamp, D. Stavens, A. Aron, J. Diebel, P. Fong, J. Gale, M. Halpenny, G. Hoffmann, K. Lau, C. Oakley, M. Palatucci, V. Pratt, P. Stang, S. Strohband, C. Dupont, L.-E. Jendrossek, C. Koelen, C. Markey, C. Rummel, J. van Niekerk, E. Jensen, P. Alessandrini, G. Bradski, B. Davies, S. Ettinger, A. Kaehler, A. Nefian, and P. Mahoney. Winning the darpa grand challenge. *Journal of Field Robotics*, 2006. accepted for publication.

[18] David H Wolpert and William G Macready. No free lunch theorems for optimization. *Evolutionary Computation, IEEE Transactions on*, 1(1):67–82, 1997.