Intra-Logistics with Integrated Automatic Deployment:
Safe and Scalable Fleets in Shared Spaces

H2020-ICT-2016-2017
Grant agreement no: 732737

DELIVERABLE 5.3

Final system for accurate AGV motion planning

Due date: month 42 (June 2020)
Deliverable type: R
Lead beneficiary: Bosch

Dissemination Level: PUBLIC

Main author: Luigi Palmieri (Bosch)

Contents

Executive Summary

This deliverable details the final motion planning unit for autonomous ground vehicles (AGV) developed in the ILIAD project. Its goal is the generation of accurate and safe human-aware motion. The architecture is composed of many components, developed from several consortium partners (Robert Bosch GmbH, ORU, UoL, TUM), namely: a hierarchical flow-aware global path planner (Bosch, ORU, UoL), post-smoothing and a model predictive controller to track the global path (ORU), a vehicle safe motion unit (TUM) that filters out velocities that may harm the persons in the scene.

As part of this deliverable, as detailed hereinafter in the report, we have achieved the following objectives:

- *developed a safe and (human) flow-aware planning architecture*, by exploiting the synergies between and the background of different partners. The architecture has been evaluated in several real-word experiments, also during the review and stakeholder meetings of MS3;
- *benchmarked state-of-the-art motion planning techniques* with a large variety of environments and robotic systems. Based on the results, we provide solid guidelines on planners' beneficial properties and their limitations;
- presented several improvement directions for the current architecture, namely: an efficient *numerical model predictive control* technique for real-time safe collision avoidance and *deterministic sampling* approaches to further enhance computational performance of sampling-based planners.

1 Introduction

This document describes the final motion planning architecture developed in the ILIAD project as result of task T5.2, "Quantitative motion planning" as part of **Work Package 5 – Fleet Management**. The system aims to generate smooth, accurate and real-time robot motion among humans. The initial prototype has been tested during the latest milestones MS2 and MS3 of the project (see Deliverables D7.3 and D7.4). The results hereinafter contribute directly to three ILIAD objectives: **O3 – On-line, self-optimising fleet management**, **O4 – Human-aware AGV fleets in shared environments** and **O6 – Human safety**.

The report is structured as follows: we initially describe the overall architecture of the system and the hierarchical global planning respectively in Section 2 and Section 3, detail the efforts of benchmarking different global planners and the introduction of deterministic sampling in Sections 4 and 5. In Section 6 we briefly describe also the findings obtained in terms of collision avoidance in very cluttered environments. Conclusions are reported in Section 7.

2 Architecture

Motion planning for the wheeled mobile platforms is a key component of the overall ILIAD safety architecture described in Deliverable D3.3, see Table 1 and Figure 1. The planning architecture, more specifically, is composed of the following layers:

- a *coordinator* that provides tasks (i.e. goals to reach) for each robot (Tasks T5.3, T5.6);
- a *qualitative situation-aware planning* module, for including local interactions into planning as additional cost term (T3.4);

#	Layer	Role	WP
5	Hierarchical (Flow-aware) global motion planning	Create paths consistent with human behaviours	WP5
4	Navigation Intent Communication	Make robot motion legible by humans	WP3
3	Situation-aware planning	Include local interactions into planning	WP3
2	Extended VSMU: Speed Constraints	Adjust robot speed to human interactions	WP3, WP4
1	Safety stops	Avoid collisions	

Table 1: Safety layers in ILIAD (from D3.3).

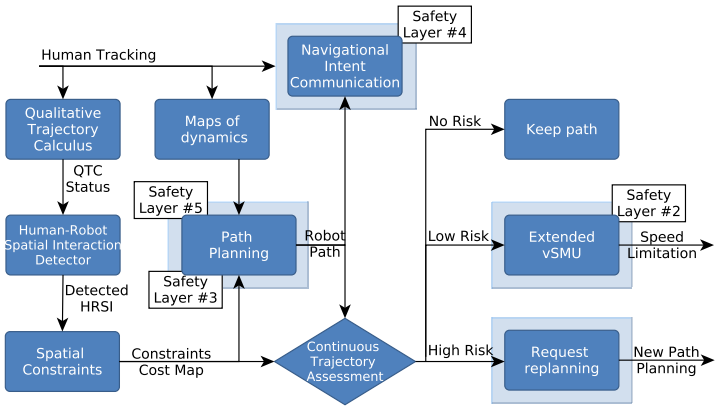

Figure 1: Interaction between ILIAD safety layers (from D3.3).

- a hierarchical global motion planner that includes a state-lattice planner and a *flow-aware quantitative sampling-based motion planner*. The latter quickly generates paths among humans by adhering to kinematic constraints and learned statistical human behaviors (T5.2);
- a path smoother (T5.2);
- a vehicle execution node that steers the AGV on the planned path based on a *model predictive control technique* (T5.2);
- a *safety unit* (named vSMU) that filters out velocities that could harm humans moving around the AGV (T4.3).

In the overall software architecture these components are interconnected and interfaced with the perception and mapping units, as shown in Figure 2: in the overall structure they are part of the *ability* level. Next we describe the hierarchical global motion planner algorithm developed in the first prototype of the ILIAD motion planning system.

3 Hierarchical Global Motion Planning

The global motion planning developed for the ILIAD system is a hierarchical planner that combines a state-lattice planning technique with a stochastic flow-aware sampling based planner, see Figure 3.

The system queries both state-lattice and sampling-based planners to generate a path. Once the two paths have been generated, a path selector will choose the path to be provided to the robot units. State lattice planning together with path smoothing belong to background knowledge of the project and it will not be discussed hereinafter [2].

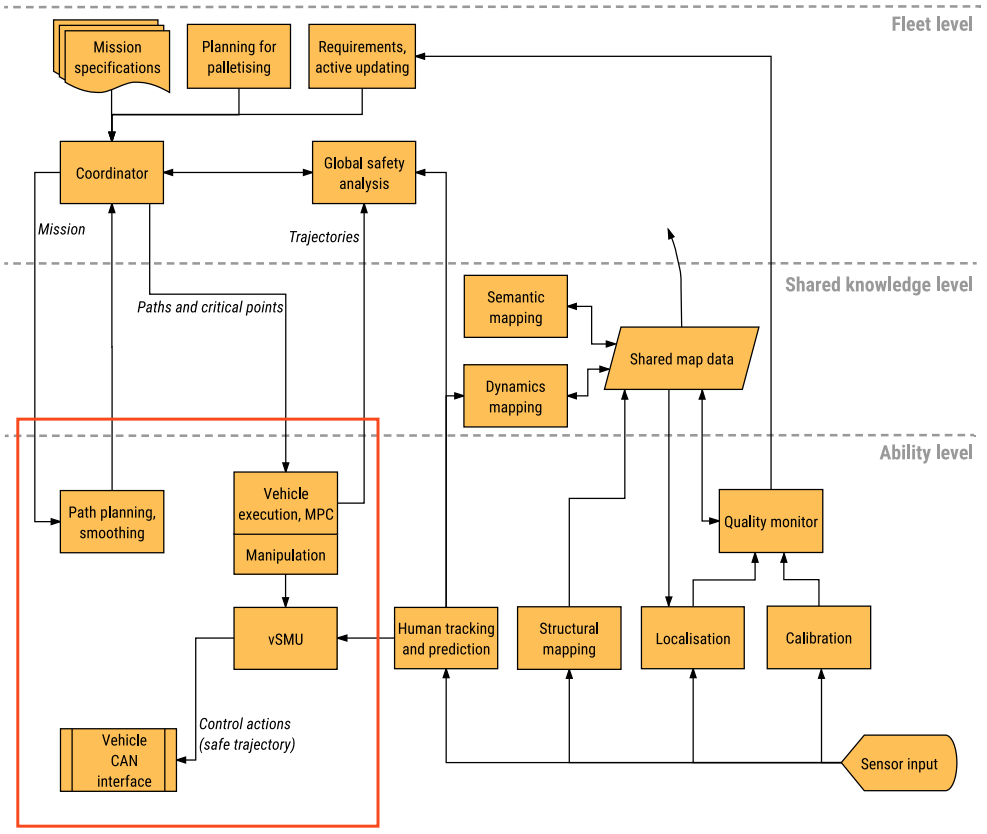

Figure 2: Motion planning and control components are a sub-system of the entire ILIAD software architecture. The components that generate robot motions are grouped into a red rectangle.

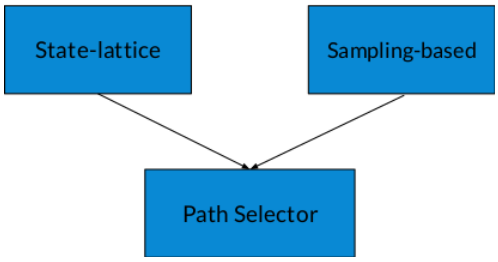

Figure 3: ILIAD hierarchical planning system. Both state-lattice and sampling-based planners are asked to generate a path. The path selector will then choose the path to be provided to the robots.

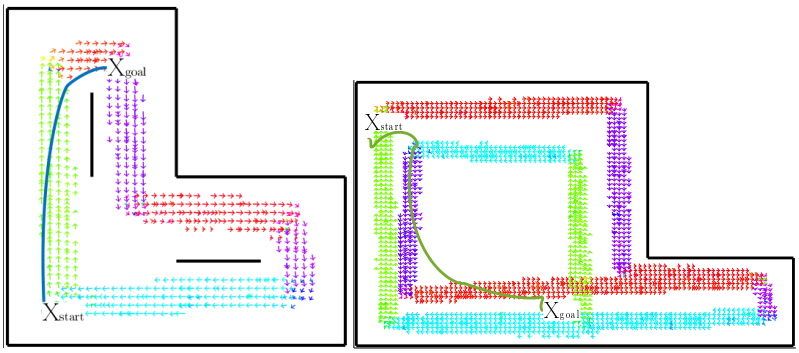

Figure 4: Left: an example CLiFF-RRT* path (in blue) generated in the *L* scenario. Right: an example CLiFF-RRT* path (in green) generated in the *P* scenario. The arrows represent the learned mixtures. In these environments just a few obstacles are present. The algorithm finds the best solution that optimizes path length and the upstream criterion: the solutions follow the learned flows.

The path selectors forward the path that has minimum cost. Different cost functions can be used in this context. In ILIAD settings, we often use path length.

In the following, we will detail the flow-aware sampling-based planning used in the hierarchical architecture.

Algorithms In ILIAD we have introduced two flow-aware sampling-based planning techniques, CLiFF-RRT* [32] and DTC-RRT* [41], which generate robot motion considering also the learned maps of dynamics (i .e., CLiFF maps [19]) provided by WP2 (long-term operation).

Both approaches [32, 41] provide novel cost-functions and algorithms for exploiting the flow information included in the CLiFF-map, see Figures 4 and 5. Both trade off classical path quality metrics with the compliance to the environment dynamics. This results in planning algorithms aware of the usual flow patterns in a given environment. Those algorithms can therefore generate paths that can obey two different behaviors, namely *follow or not follow* a flow of humans walking.

In CLiFF-RRT*, we solve a motion planning problem hierarchically by first generating a discrete path that selects mixtures at relevant locations from the map of dynamics, and then use those mixtures to bias the sampling and rewiring procedures in RRT* [17]. The first step makes sure that an initially feasible path is found quickly given a CLiFF map while the second step, generates and incrementally improves a trajectory that satisfies

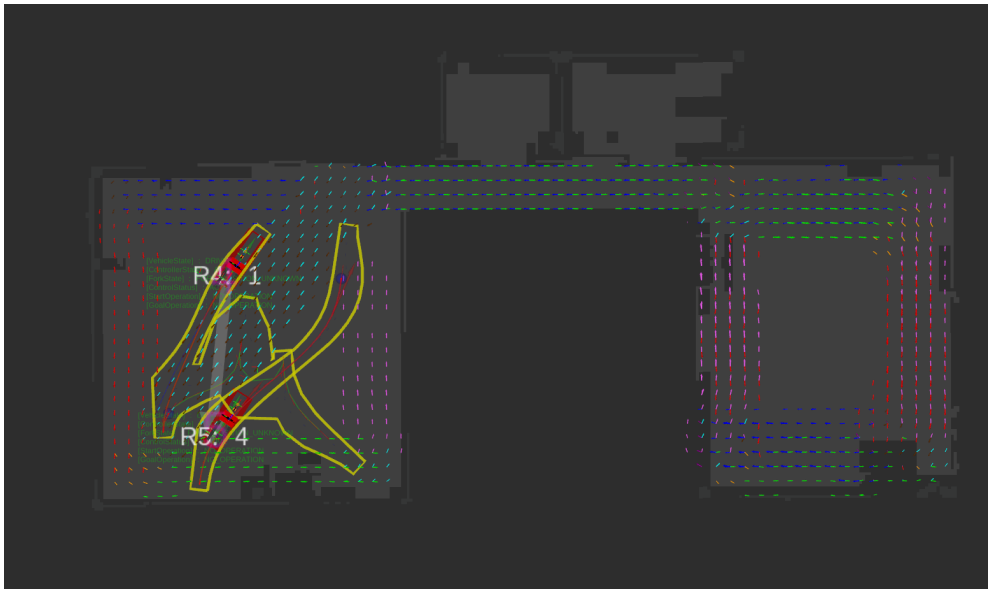

Figure 5: Example paths generated by a flow-aware motion planner in the ILIAD planning system. The paths are generated respecting the CLiFF map and they are provided to the coordination unit.

the kinodynamic vehicle constraints, see Figure 6 for an example of resulting paths in simulated dynamic environments.

DTC-RRT* further extends CLiFF-RRT* by also considering the mean speed encoded in the distribution and the information about uncertainty. In DTC-RRT* [41], we use rejection sampling from a biased distribution in order to guide the exploration of the configuration space. In particular, we use the motion and observation ratios in the sampling procedure so as to prefer an *exploratory and congestion-avoiding* behaviour that guides exploration towards regions with fewer observations or less motion, while also trying to closely match the modes of flow that have been learnt. The biasing strategy is as follows: With a 5% probability, a state is chosen from the goal region. Otherwise, a state is sampled uniformly from the entire state space. With a 20% probability, this sample is unconditionally accepted. This number was arbitrarily picked to ensure probabilistic completeness. Otherwise, we do the following: if the motion ratio is low, we accept the sample, since either the cost is low (if the observation ratio is also low) or we are confident that there is little motion in this region. If the motion ratio is high, but the observation ratio is low, we try to follow the flow. If both the observation and motion ratios are high, then we reject the sample (cost is likely to be higher).

Evaluation In our experiments [32, 41], both approaches are compared to several baselines and it is shown that flow-aware planning is significantly *faster* than the baselines and produces solutions that better comply to the flow directions as modeled by the map. The algorithms also achieve shorter and smoother paths and retains the probabilistic completeness and asymptotic optimality properties of RRT* [17].

In particular we have obtained the following results:

- CLiFF-RRT* and DTC-RRT* with its focused search find an initial solution faster than all the baselines. RRT and RRT* do not avoid the time-consuming exploration of the entire state space. For this reason, the latter more often fails to find an initial

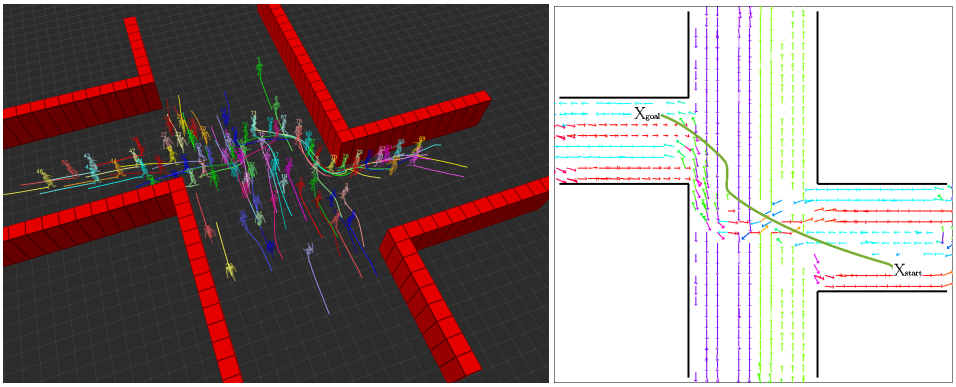

Figure 6: **Left:** The figure shows an example from the studied simulated *intersect* scenario where multiple flows of people encounter each other. **Right:** An example path (in **green**) generated among the Circular Linear Flow Field (CLiFF) map which associates a Gaussian mixture model to each location, whose components encode multiple weighted flow directions.

solution in the given time.

- CLiFF-RRT* and DTC-RRT* finds less costly solutions if compared to all the baselines.
- The CLiFF-RRT* sampling strategy results in smoother trajectories than the baselines. Mainly because the off-line learned mixtures bias the tree towards concatenation of extensions with fewer velocity discontinuities. Uniform sampling generates velocities without prior knowledge about common motions in particular portions of the state space, thus producing less correlated velocities.
- DTC-RRT* with a sampling bias yields solutions with lower upstream cost compared to CLiFF-RRT*. Minimizing the Mahalanobis distance (a strong cost-term in the cost function of the DTC-RRT*) should automatically minimize upstream cost because both vehicle speed and heading at each trajectory point are closer to the mean of underlying flows when the Mahalanobis distance is minimized.

Figure 7 shows example policies (i.e. trajectory distributions) in the form of heat-maps generated by DTC-RRT*.

We refer the reader to the papers [32, 41] for more details about the theoretical properties and performance evaluation of the algorithms. We are planning to extend the approach to also consider time-varying pedestrian flow models [43].

We have tested the flow-aware planning algorithms on different types of robots. During the MS3 experiments, we have deployed our software on two different robotic systems: the BT truck and the citi truck robot, see Figure 8.

4 Benchmarking of Global Motion Planning Algorithms

In ILIAD, to investigate the current state-of-the-art in motion planning for wheeled mobile robots, we establish a benchmarking framework that is tailored toward these kinds of kinodynamic systems and their application in real-world scenarios. We here report preliminary results also available in [12].

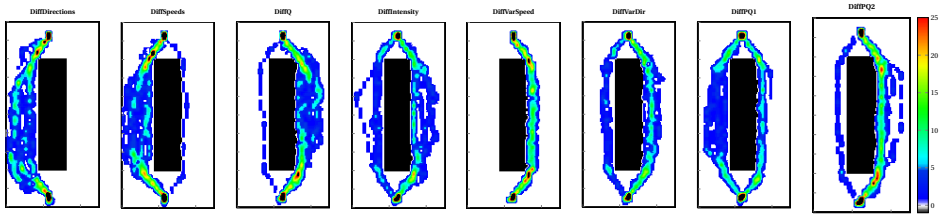

Figure 7: Heat maps for trajectories generated by DTC-RRT*. Colors indicate number of trajectory points at a cell. Black denotes an obstacle, White denotes free space. The eight maps used can be described in words as follows: (1) *DiffDirections* map has two flows with opposite mean directions; (2) *DiffSpeeds* map has two flows with different mean speeds; (3) *DiffQ* map has one flow on the left side but no flow on the right, hence the two regions have different motion ratios; (4 *DiffVarSpeed* has two flows with different variance in speed; (5) *DiffVarDir* has two flows with different variance in orientation; (6) *DiffIntensity* has two flows where the corresponding regions have different motion ratios; (7) *DiffPQ1* and (8) *DiffPQ2* have two flows with different motion and observation ratios.

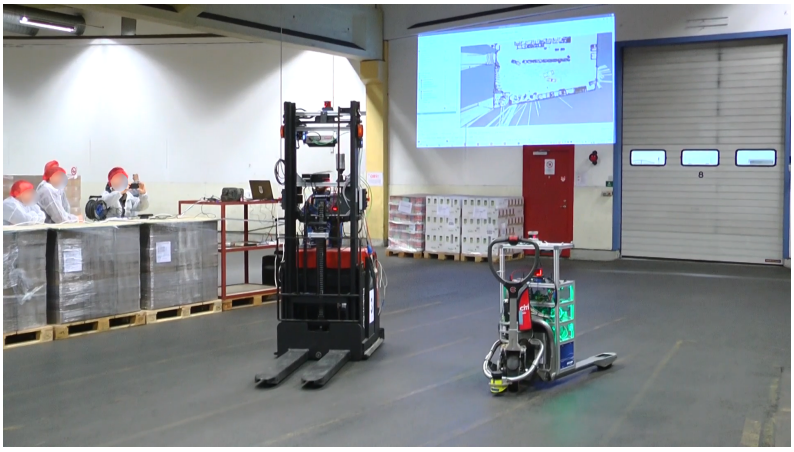

Figure 8: The flow-aware planning has been deployed on two different robotic systems during the review and stake-holder meeting of MS3: the BT truck (bigger robot on the left-hand side) and the CiTi truck (smaller robot on the right-hand side)

As shown in Figure 9, our benchmarking is based on the following ingredients: motion planners, post-smoothing methods, steer functions[1] and collision checkers. The combination of these building blocks is then evaluated in a variety of scenarios (environments with start and goal configurations) along various metrics.

While much of our benchmarking software largely builds on the Open Motion Planning Library (OMPL) [40], we provide interfaces to implementations of planners (such as SBPL planners and Theta*) and steer functions (POSQ and continuous-curvature steering) outside of OMPL. This enables us to get a more comprehensive picture of the current progress in motion planning for wheeled mobile robots, while being more agnostic to particular implementations of the building blocks.

[1] In the literature often denoted also as steering or extend functions.

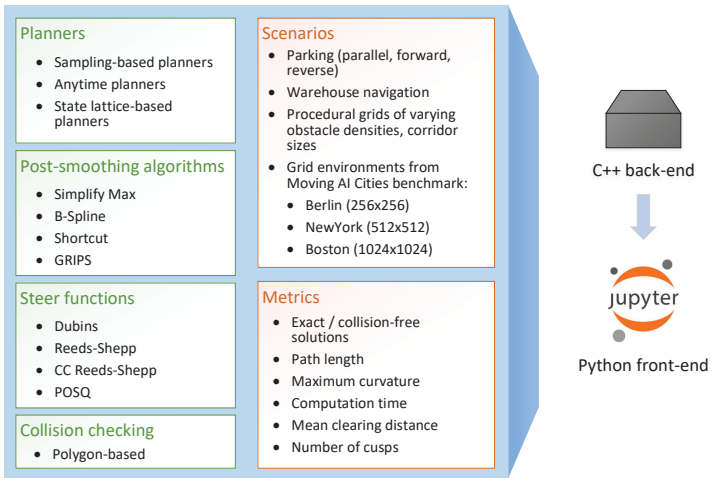

Figure 9: Architecture of the proposed motion-planning benchmarking framework. The components necessary for motion planning are shown in the box on the left (green), and the ingredients used in the evaluation are shown in the box on the right (red). The implementation is split into a C++ back-end for running the resource-intensive motion-planning components, and a Python front-end for providing a flexible interface to the design and evaluation of the benchmarking scenarios through Jupyter notebooks.

4.1 Approach

We benchmark global motion planning algorithms commonly used for wheeled mobile robots, and provide general recommendations on the usage of these methods, considering their combination with post-smoothing methods and various steer functions. Our benchmark is based on two fundamental pillars: the components involved in motion planning and the evaluation procedures (shown in the box on the left and right, respectively, in Figure 9). In particular, evaluating the performance of a motion planning algorithm requires selecting the appropriate testing environments (e.g., considering different types of map representations) and metrics (related to planning efficiency and quality of the results). We carefully selected these components by considering their scientific impact, and their recognition and popularity in the open source community [8, 25, 40].

4.2 Evaluation

In this section, we describe the set of experiments, environments and the metrics used to evaluate the planners and post-smoothing methods in terms of planning efficiency and in returned path quality. The list of all experiments, pointing to the related results' sections, is reported in Table 2. The table collects eleven different types of experiment we run: three of them use Moving-AI grid environments, four use procedurally generated grids, and four use a polygonal representation of the environment and robot. Of the latter, three study the behavior of the planners when the environment complexity change and one properties of post-smoothers and planners' combinations. Throughout all our experiments, we use two-dimensional polygon-based collision models (see Figure 10) where the robot is represented by a convex shape.

Experiment Section	Environment	Collision model	Description
`cross_corridor` Section 4.4.3	100 × 100 grid (procedural)	car	Evaluation on procedurally generated corridor environments with varying corridor diameters (Figure 11 bottom)
`cross_turning` Section 4.4.3	100 × 100 grid (procedural)	car	Evaluation on procedurally generated grid environments with varying turning radii in Reeds Shepp steering
`cross_density` Section 4.4.3	100 × 100 grid (procedural)	car	Evaluation on procedurally generated grid environments with varying obstacle densities (Figure 11 top)
`sam_vs_any` Section 4.4.4	150 × 150 grid (procedural)	car	Comparison of anytime planners vs. a combination of sampling-based planners and post-smoothing methods
`Berlin_0_256` Section 4.4.1	256 × 256 grid (MovingAI)	car	Evaluation of the 50 hardest scenarios from the Berlin_0_256 MovingAI benchmark
`parking_1` Section 4.4.2	polygon	car	Evaluation on the polygon-based environment parking_1 (Figure 12)
`parking_2` Section 4.4.2	polygon	car	Evaluation on the polygon-based environment parking_2 (Figure 12)
`parking_3` Section 4.4.2	polygon	car	Evaluation on the polygon-based environment parking_3 (Figure 12)
`warehouse` Section 4.4.2	polygon	warehouse bot	Evaluation on the polygon-based environment warehouse (Figure 12)

Table 2: Overview of experiments conducted in this benchmark.

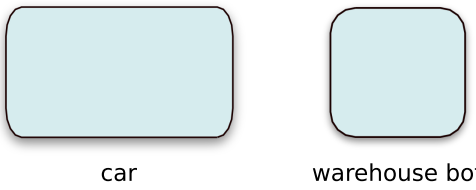

Figure 10: The two different polygon-based collision models used throughout this benchmark.

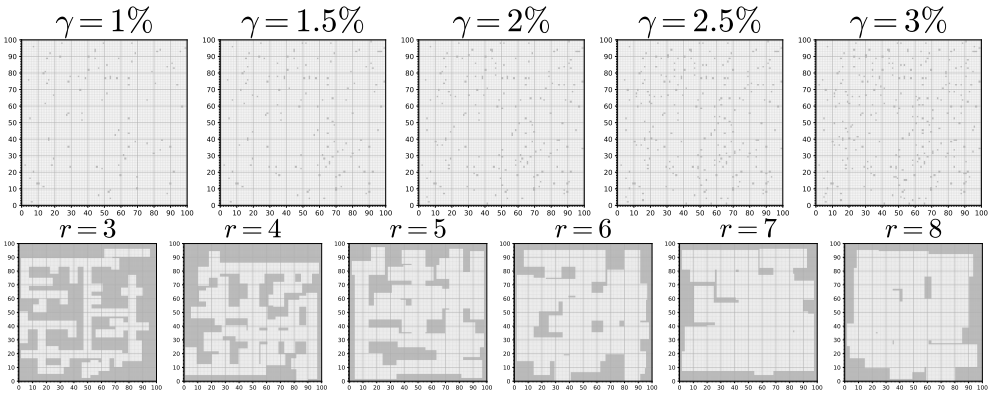

Figure 11: Examples of the procedurally generated grid environments. Top: varying obstacle ratios. Bottom: varying corridor sizes.

4.2.1 Environments

In the following, we describe the two types of environments we consider throughout our benchmarking, as well as the how the *scenarios* are defined; i .e., the start and goal configurations for each environment. We consider the two main classes of environmental representation used today for motion planning: grids and polygon-based maps.

Grid maps is a typical approach used in robotics navigation, in particular when planning in large environments (i .e., cities, airports, train stations or large office-like environments). We design two sets of environments, a sub-selection of the grids form the Moving AI benchmark [39] and a set of grids procedurally generated by varying corridor sizes or obstacle densities, see an example in Figure 11. .

Polygon-based environments are often adopted when planning in tight and small environments (i.e. parking and warehouse like environments), where the planning system should more carefully and precisely consider obstacles' geometry. We show the four types of polygon-based environments we designed in Figure 12 and with example paths in Figure 13. We choose five start-and-goal configurations and validate them by ensuring that the planner BFMT[2] finds exact solutions using the Reeds-Shepp steer function (Figure 12). In the first three cases, we consider the scenario of an autonomous car-like vehicle that needs to park itself among a set of surrounding parked cars and other obstacles. We consider the three common cases of parking: (1) parking forward, (2) parking backward into a parking lot, and (3) parallel-park in a street of parked cars. In the last type of polygon-based environments (4), a complex warehouse-like environment is simulated where the robot has to navigate between shelves of various sizes and irregular orientations.

[2]Through preliminary experiments, we found BFMT to be among the most reliable planning algorithms that gave high-quality solutions in short time on the polygon-based environments.

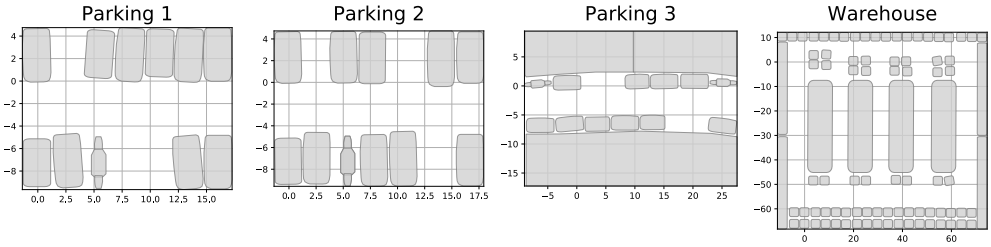

Figure 12: The four polygon-based environments where obstacles are represented by convex shapes.

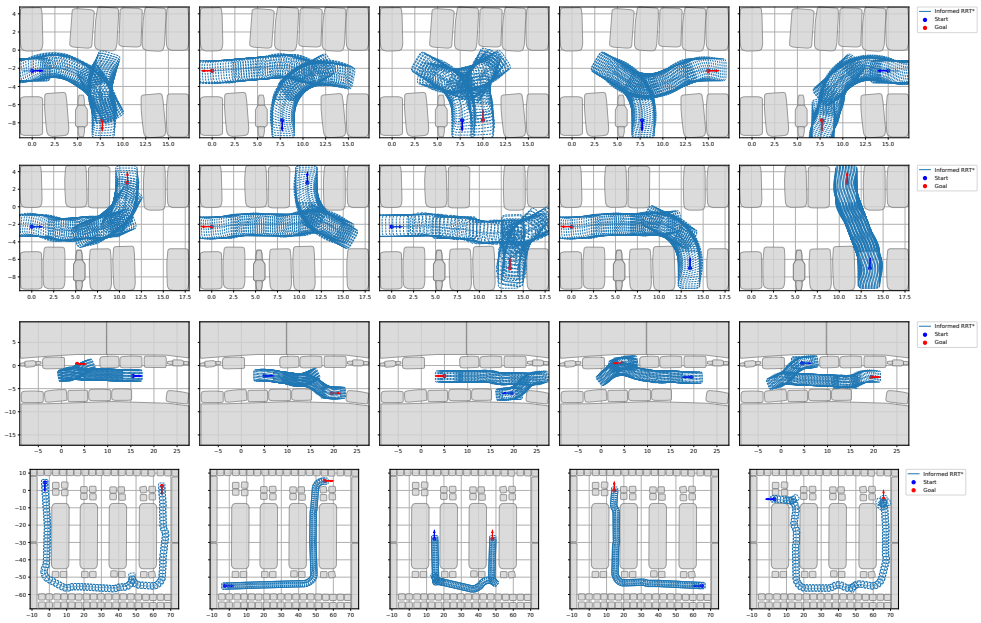

Figure 13: Example results for the polygon-based environments *parking1*, *parking2*, *parking3*, and *warehouse* (from top to bottom) with all five different start/goal configurations. Each subplot shows the computed trajectories from the Informed RRT* planner using the CC Reeds-Shepp steer function.

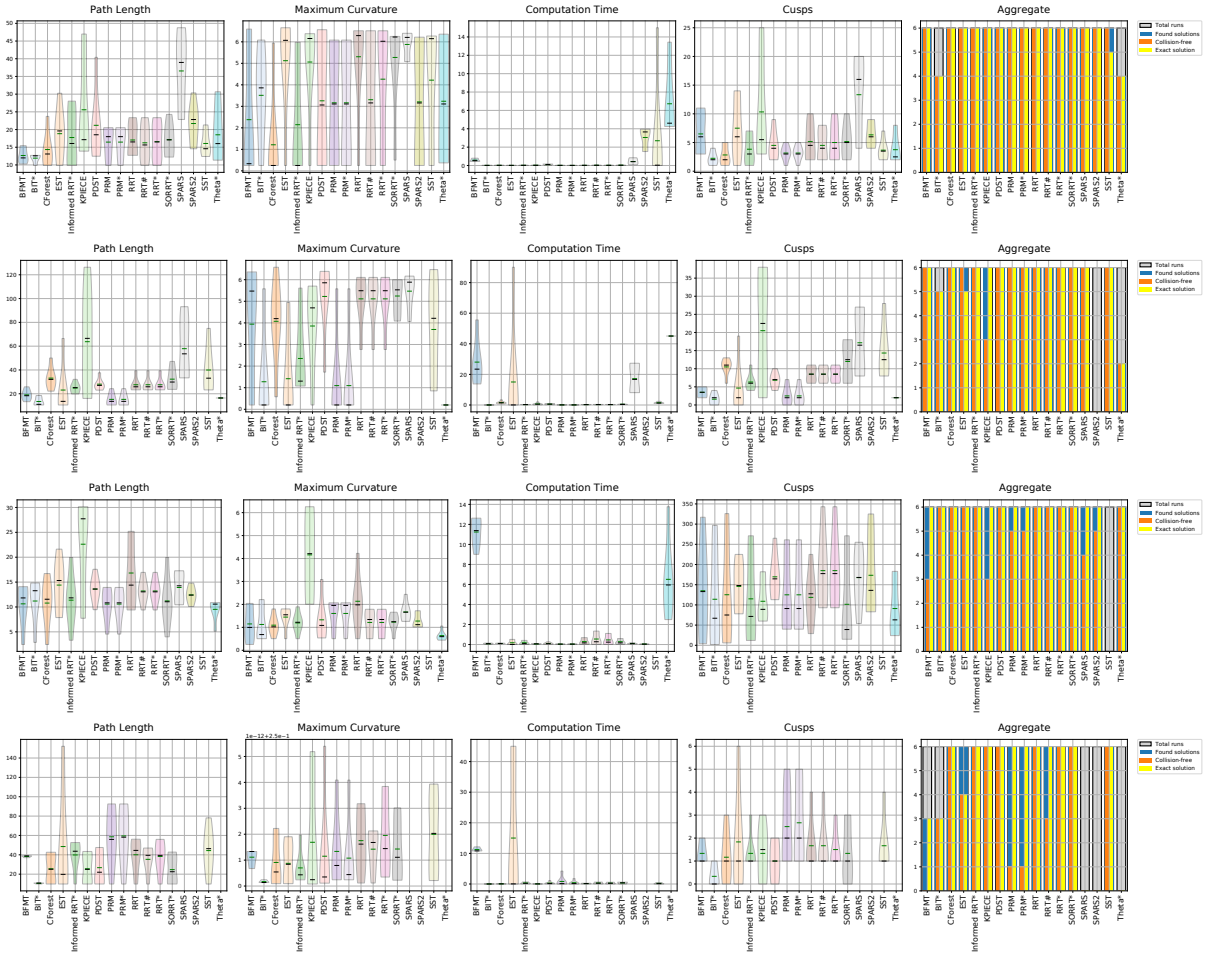

Figure 14: Statistics for the *parking1* scenarios. First row: Reeds Shepp steering, second row: CC Reeds Shepp steering, third row: POSQ steering, fourth row: Dubins steering.

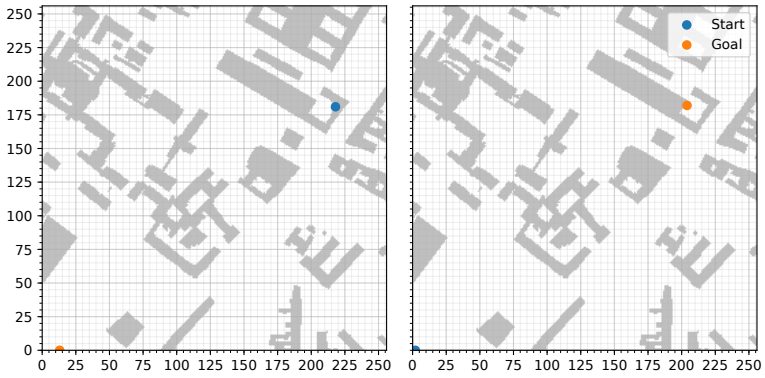

Figure 15: Many of the challenging Moving AI Cities scenarios define start and goal locations that are too close to obstacles to be solvable by a polygon-based collision model of the robot. Shown here are two scenarios from the *Berlin_0_256* map with highlighted start and goal positions.

4.2.2 Metrics

We compare the planners based on a selection of metrics relevant to wheeled mobile robotics applications, such as autonomous driving, service and intralogistic robotics. In particular, we evaluate the planners in terms of quality of the returned solutions and in planning efficiency by considering the following metrics:

- *Success statistics* that measure the ratio of found, collision-free, and exact [3] solutions.
- *Path length* of the obtained solution in the workspace $\mathcal{W}$. All the asymptotically optimal planners are configured to minimize path length, thus we measure how well the planners performs based on their main objective.
- *Curvature* (κ) and *Maximum curvature* ($\kappa_{\max}$): as a way to measure the induced comfort and smoothness of the obtained paths. Keeping the maximum curvature at a low level corresponds to smoother maneuvers, therefore less control effort and energy to steer the robot.
- *Computation time* to find the first solution.
- *Mean clearing distance* ($\overline{\delta}_{\text{dist}}(\gamma)$): with lower values indicating that the solutions are closer to the obstacles.
- *Number of cusps* following Banzhaf et al. [3]: maneuvering in difficult environments may require the robots to stop and turn the wheels in the opposite direction, thus yielding a cusp in the trajectory. Having more cusps correspond to less smooth and more difficult to drive paths.

4.3 Benchmark Implementation

We develop our benchmarking system in C++ and provide a high-level front-end in Python[4]. The experiments are implemented in Jupyter notebooks that leverage our Python front-end and enable the user to monitor the status of the execution through rich progress

[3] A trajectory is *exact* if it connects the start and goal nodes.
[4] Our code will be made open-source at https://github.com/robot-motion/mpb.

reports and plotting capabilities. We are collecting the experimental results and derived plots on our website at https://robot-motion.github.io/mpb/ where the complete data can be analyzed.

We run the benchmark on a server featuring 256Gb RAM, two Intel Xeon Gold 6154 @ 3.00GHz CPUs offering 72 threads in total, running on Ubuntu 18.04 (kernel version 4.15.0). Each experiment is run using 20 parallel processes that correspond to different environment seeds, in the case of the procedurally generated environments. Each process runs a sequence of planners and post-smoothing methods on its predefined environment. We limit the parallelism to 20 out of 72 available CPU cores due to the fact that planners such as CForest spawn multiple threads on their own to find a solution. By further randomizing the order in which each of our benchmark processes executes the planners, we can keep the number of parallel threads in check (e .g., avoid running 20 parallel CForest instances). Each process is automatically canceled if twice of its time limit has been exceeded (time out), or if its memory consumption has exceeded 18 GB.

4.4 Results

This section summarizes the results obtained in our experiments, while focusing on the main findings. The complete statistical analysis will be published on https://robot-motion.github.io/mpb/.

4.4.1 Moving AI Scenarios

This section reports the results obtained of the grids selected from the Moving AI benchmark, see Table 3. The solution column contains two numbers separated by a '/': the second number indicates the number of solutions found (highlighted by the orange bar in the background), the first number indices how many of these solutions are collision-free. Each planner is run on a total of 51 scenarios. The following columns indicate the planning statistics in the format mean ± standard deviation across the metrics (planning time, path length, maximum curvature and average curvature along the paths). The last column shows the total number of cusps in all solutions combined. We group these statistics by the steer functions, for which we selected different time limits, as shown in the tables next to the group labels. These time limits have been determined empirically to ensure that many solutions could be found. The SBPL planners are treated separately since they did not use any of the provided steer functions but their particular unicycle motion primitive.

Path Length and Smoothness Results are detailed in Table 3. In terms of path length and smoothness, anytime path planners achieve better performance within the given maximum planning time for all the steer functions. Feasible planners generate often longer and less smooth paths (higher curvature and number of cusps). Specifically for the scenario *Berlin_0_256*, BFMT achieves the shortest path lengths within the fastest time with average curvature, except with POSQ steering where it has poor runtime and path length. KPIECE throughout all experiments finishes among the fastest but consistently has the longest paths and among the worst maximum curvature. For the Dubins curves, it was considerably more difficult for the planners to find feasible solutions – sampling-based planners, such as EST, SST, PDST and KPIECE were the most successful in finding exact, collision-free paths. We present more results on other environments from the Moving AI Cities dataset in the extended version of our paper [12].

Post-Smoothing Results In Figure 23 we summarize the post-smoothing results across all planners in the *Berlin_0_256* scenarios, which is representative for the other Moving AI

benchmark environments. GRIPS often outperforms the other methods in maximum curvature while achieving similar path length as SimplifyMax. In computation times, B-Spline, Shortcut and SimplifyMax perform similarly, except with POSQ steering where the latter is significantly slower with a median computation time almost twice as high as the other methods. SimplifyMax yields solutions which often have very small clearing distance. B-Spline solutions have considerably more cusps than the results obtained with the other methods.

Theta* and SBPL Issues On the larger-scale environments considered throughout this benchmark (particularly the Moving AI scenarios), we noticed that our current implementation of Theta* makes heavy use of the collision checker that significantly deteriorates its computation time. As can be seen in Table 3, only in the case of a 6 minutes time limit for a fast-to-evaluate steer function, such as Dubins, does this algorithm find a competitive number of collision-free, exact solutions. In other cases, our implementation does not yield a solution before the time limit is up. Similarly, the planners AD*, ARA* and MHA* from SBPL were often unable to find feasible solutions within the time limit.

4.4.2 Polygon-based Environments

The following scenarios are particularly tailored toward autonomous driving. Instead of navigating grid world, the environments use arbitrary convex shapes to represent obstacles.

Parking scenarios Similarly to the grid-based environments, in these scenarios, anytime planners achieve better performance in terms of path length and smoothness than feasible planners, although at the price of being slower. In the scenarios for the first parking environments, we notice that RRT, Informed RRT*, RRT* and SORRT* always find solutions, across all tested steer functions, as shown in Figure 14. SST, Theta*, SPARS and SPARS2, however, do often not find any solutions. Particularly SPARS2 is the only planner that cannot find any solutions for CC Reeds-Shepp steering, SST is the only algorithm that is unable to solve any scenarios with POSQ steering. The Dubins steer function appears to be particularly challenging, as SPARS, SPARS2 and Theta* cannot find any paths, while various other planners, such as PRM, PRM*, BFMT and BIT* only solve a small fraction of the scenarios exactly.

Warehouse scenarios We visualize example solutions obtained from all planners on the fourth scenario from the warehouse environment with Reeds-Shepp steering in Figure 16. BFMT, CForest, Informed RRT* and SORRT* find the shortest solutions which all lie in the same homotopy class.

Compared to most parking scenarios, the warehouse environment typically requires longer computation times for the planners (especially anytime planners) to find solutions. It offers considerably more opportunity for the planners to find solutions of varying homotopy classes (see Figure 16), resulting in a larger variance of path length. CForest, Informed RRT*, and SORRT* consistently find among the shortest paths, although Informed RRT* has among the longest computation times (see Figure 17).

4.4.3 Procedurally-generated grid environments

As described in Section 4.2.1, we procedurally generate environments to have full control over the shape of the free space within the planners need to find solutions. This allows

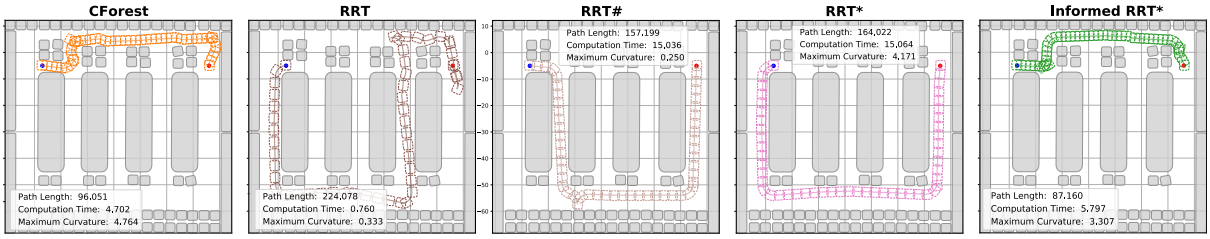

Figure 16: Example trajectories for the different planners in one of the five *warehouse* scenarios with Reeds-Shepp steering.

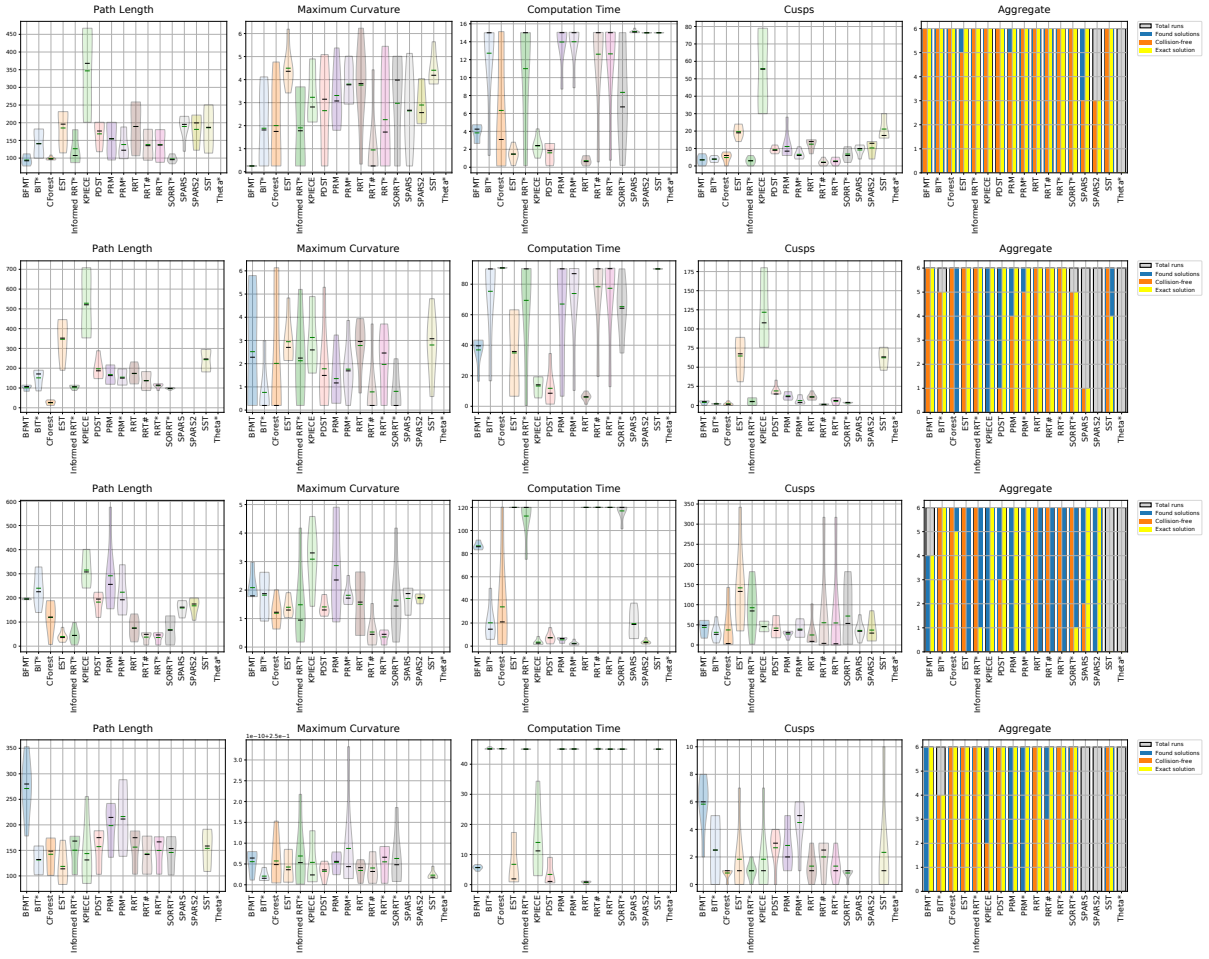

Figure 17: Statistics for the *warehouse* scenarios. First row: Reeds-Shepp steering, second row: CC Reeds-Shepp steering, third row: POSQ steering, fourth row: Dubins steering.

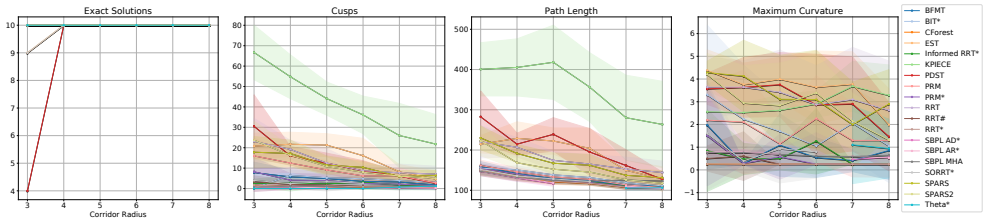

Figure 18: Various planning statistics for the Reeds-Shepp steer function in the procedurally grid environments with varying corridor sizes.

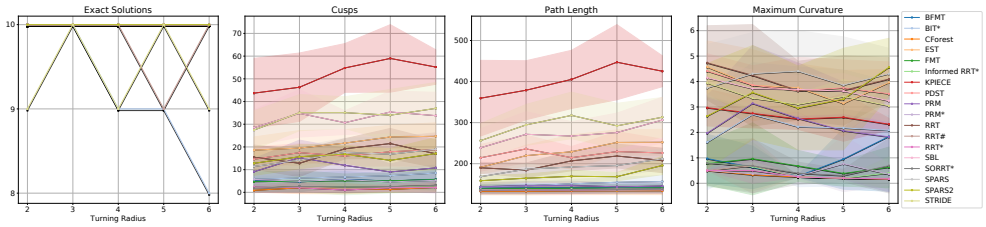

Figure 19: Various planning statistics for different turning radii (in meters) of the Reeds-Shepp steer function in the procedurally grid environments (size: 100×100).

us to precisely analyze how varying features of the environments influence the planning results.

Varying corridor sizes As shown on the abscissa in Figure 18, the corridor sizes are expressed in the number of grid cells. We sample five 100×100 grid environments for each corridor radius (Figure 11 bottom row), sampled from the same starting seed over radii between three and eight grid cells. As we increase the corridor size, the path lengths of all planners decrease, as well as the number of cusps. The curvature metric remains mostly unaffected, except for PDST, PRM and SPARS2 where it considerable decreases. Theta*, the SBPL planners, Informed RRT* and RRT# constantly have a low number of cusps and achieve very low path lengths across all conditions. KPIECE performs the worst in number of cusps and path length. EST, KPIECE, SPARS, SPARS2 and PRM have poor curvature, but PDST improves by a factor of two toward the maximum corridor size. While PDST and RRT initially find four and nine out of ten possible solutions, only at a corridor radius of four cells do all planners find exact solutions in every case.

Varying turning radii We vary the turning radius used by the Reeds-Shepp steer function and evaluate the planners on a 100×100 indoor-like grid environment with a corridor radius of five grid cells (see Figure 11 bottom row). The change in turning radius has a surprisingly little effect on the path quality, see Figure 19. Slight developments can be observed where the path lengths tend to increase as the turning radius becomes larger. Especially PRM has a pronounced inclination in the number of cusps. The curvature is generally not tending in any direction significantly. The number of exact solutions is at zero for Theta*, PRM constantly finds two out of ten solutions, while the other planners find all of the solutions exactly.

Varying obstacle densities As described in Section 4.2.1, in this experiment, we randomly set cells of a 100×100 grid environment to be occupied until a selected density (i .e., ratio between occupied and free cells) has been achieved (see Figure 11 top row). Through

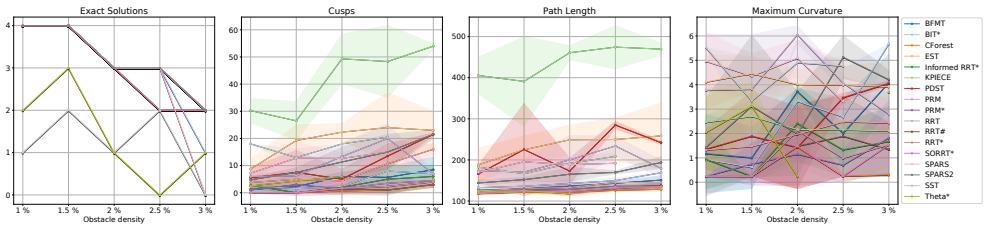

Figure 20: Planning statistics of the Reeds-Shepp steer function in the procedurally generated grid environments (size: 100×100) with varying occupancy ratios.

various experiments, we determined the ranges between 1% and 3% to yield meaningful results. We successively increase the obstacle density in steps of 0.5%, and yet the influence on the quality of the found solutions is significant. From Figure 20, we can see that none of the planners are able to find exact solutions in all ten cases. Most start at four solutions which drops to one and zero as the maximum obstacle density is approached. Meanwhile, the number of cusps increases dramatically, especially for KPIECE, PDST; and even BFMT, SPARS and PRM* have a relatively strong increase. The path lengths are not as much affected, although increasing in many cases, such as EST, SPARS2, SPARS, KPIECE. The curvature is increasing for many planners, such as PDST, PRM, PRM*.

4.4.4 Planning and Post-Smoothing

Based on our experiments with varying time limits over a range of time limits between zero and 30 seconds, we are investigating how post-smoothing methods can benefit the motion planning pipeline. In combination with sampling-based planners, which quickly find feasible solutions, can these improvement techniques yield results that are qualitatively competitive with the solutions obtained by anytime planners within shorter computation times?

To answer this question, we run a set of sampling-based planners (EST, RRT, SBL, STRIDE) with all post-smoothing methods considered in this benchmark, and compare it against anytime planners run at time limits ranging between five and 60 seconds.

As shown in Figure 21, we observe that the algorithms GRIPS and Simplify Max yield significant improvements in path length and maximum curvature. They both reduce the path length typically by a factor of two and similarly smooth the path in a way that the maximum curvature drops by close to a factor of two. In most cases, Simplify Max is considerably faster than GRIPS to obtain these results. The B-spline algorithm does not always improve the path quality, which may be explained by the problem that B-splines do not translate well to curves that can be followed by Reeds Shepp steering, leading to slight turns that increase the curvature.

Overall, there exist several couplings between sampling-based planners and post-smoothers that outperform anytime planners in speed and solution quality. For example, within three seconds RRT combined with Simplify Max smoothing achieves a maximum curvature at the same level as an anytime planner such as Informed RRT* after 60 seconds, while yielding a shorter path length (Figure 22).

4.5 General Observations

Based on the results detailed in Section 4.4, in this section we provide a general analysis across the experiments and give specific recommendations.

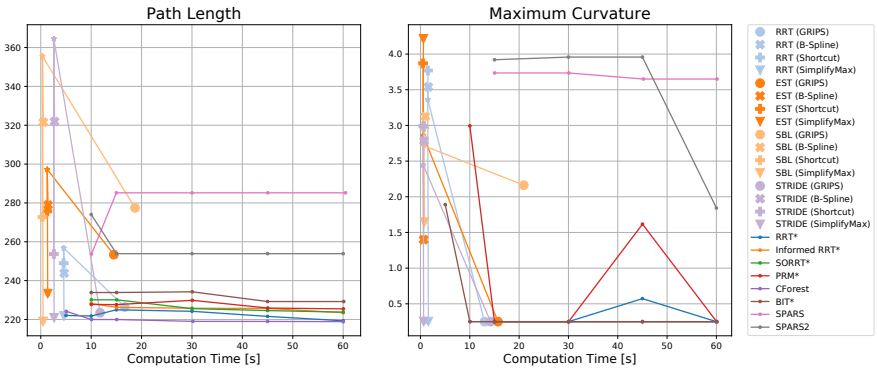

Figure 21: Comparison of sampling-based planners in combination with post-smoothing methods and anytime planners evaluated over maximum time limits 5, 10, 15, 30, 45, 60 seconds on a 150×150 grid environment. The initial solution found by the sampling-based planners is indicated by a ★ symbol, the post-smoothers GRIPS (●), B-Spline (✖), Shortcut (✚) and SimplifyMax (▼) are marked according to the legend. The anytime planners are shown as solid lines with · markers. *Left:* path length of the respective solutions. *Right:* maximum curvature.

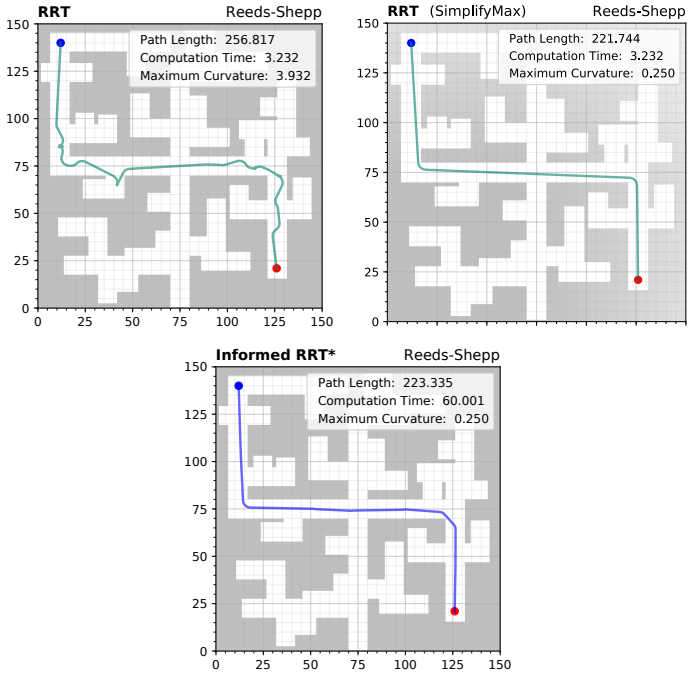

Figure 22: Trajectories resulting from the comparison of sampling-based planners in combination with post-smoothing methods against anytime planners. The solution on the left is obtained from the sampling-based planner RRT after 3.232 seconds. Using the SimplifyMax algorithm, this solution is smoothed (center), within a total time (including RRT planning) of 3.232 seconds. On the right, the solution from Informed RRT* is shown, which is computed after 60.001 seconds.

Planner	Solutions	Time [s]	Path Length	Curvature	Clearance	Cusps
Scenario: `Berlin_0_256` (SBPL, 12 minutes time limit)						
SBPL AD*	9 / 9	120.05()	361.92(623)	1.36(174)	10.40(188)	34
SBPL ARA*	4 / 4	120.05(1)	360.43(394)	0.77(7)	10.99(135)	11
SBPL MHA*	10 / 10	4.52(453)	397.16(558)	2.45(262)	12.81(290)	37
Scenario: `Berlin_0_256` (Reeds-Shepp steering, 1.5 minutes time limit)						
BFMT	50 / 51	1.39(389)	369.51(872)	1.13(96)	8.82(287)	204
BIT*	13 / 50	90.05(9)	362.10(634)	1.27(101)	7.94(211)	159
CForest	5 / 51	90.04(7)	347.10(554)	0.56(73)	7.05(183)	70
EST	46 / 51	2.19(1247)	566.35(11692)	1.70(43)	11.16(157)	769
Informed RRT*	13 / 51	90.01(1)	350.08(589)	0.35(33)	7.68(202)	67
KPIECE	45 / 51	1.03(695)	1077.02(34455)	1.17(51)	11.22(138)	1791
PDST	43 / 51	3.40(1355)	580.61(14346)	1.42(61)	11.32(174)	555
PRM	24 / 51	90.08(7)	364.16(1069)	1.21(112)	8.80(240)	329
PRM*	33 / 51	90.07(6)	357.09(846)	0.67(83)	8.99(251)	156
RRT	48 / 51	4.08(1743)	475.59(7071)	1.93(53)	11.00(144)	636
RRT#	26 / 51	90.09(47)	346.96(1478)	0.58(75)	7.36(197)	90
RRT*	18 / 51	90.01(1)	347.58(1250)	0.44(55)	7.31(194)	80
SORRT*	21 / 51	90.01(1)	350.00(564)	0.44(59)	7.44(189)	75
SPARS	46 / 51	90.33(43)	471.09(10080)	1.68(71)	11.05(95)	586
SPARS2	44 / 50	90.01(1)	410.60(4150)	2.07(56)	10.26(246)	490
SST	48 / 51	90.01(1)	506.46(7741)	2.03(52)	9.43(161)	1348
Theta*	0	N/A	N/A	N/A	N/A	N/A
Scenario: `Berlin_0_256` (CC Reeds-Shepp steering, 18 minutes time limit)						
BFMT	49 / 50	35.67(426)	366.93(1215)	0.59(77)	8.78(214)	126
BIT*	26 / 28	1080.46(85)	372.78(939)	0.75(70)	7.83(283)	101
CForest	0 / 51	5.66(2036)	334.50(4635)	0.74(86)	7.95(260)	122
EST	49 / 51	48.41(14831)	670.16(11472)	1.41(32)	11.46(189)	1793
Informed RRT*	39 / 51	1080.19(18)	353.80(1332)	0.39(56)	7.61(200)	73
KPIECE	26 / 51	77.82(23560)	1022.48(24238)	1.04(35)	11.23(125)	3221
PDST	40 / 51	132.05(24465)	523.04(12990)	1.18(61)	11.63(169)	540
PRM	38 / 51	1062.67(12451)	389.49(3272)	0.84(80)	9.06(189)	411
PRM*	38 / 51	1080.34(19)	379.15(2899)	0.86(89)	9.12(194)	260
RRT	49 / 51	35.39(15367)	500.27(7597)	1.26(66)	11.42(159)	527
RRT#	51 / 51	1080.54(62)	351.14(1994)	0.29(39)	7.97(196)	69
RRT*	47 / 51	1080.15(8)	348.93(1716)	0.36(48)	7.71(195)	55
SORRT*	9 / 15	1080.23(28)	352.77(2192)	0.20()	6.19(262)	48
SPARS	48 / 49	1083.04(292)	468.54(7590)	1.39(73)	11.28(130)	429
SPARS2	0	N/A	N/A	N/A	N/A	N/A
SST	49 / 51	1080.03(2)	605.81(7993)	1.45(110)	9.81(157)	6950
Theta*	0	N/A	N/A	N/A	N/A	N/A
Scenario: `Berlin_0_256` (POSQ steering, 12 minutes time limit)						
BFMT	4 / 10	582.35(6916)	1010.48(28789)	0.98(33)	13.03(139)	137
BIT*	35 / 41	141.23(9690)	790.39(36817)	0.98(36)	12.78(223)	396
CForest	28 / 51	149.11(18951)	341.97(14455)	0.92(41)	13.75(526)	171
EST	50 / 51	720.03(3)	138.47(5790)	1.42(37)	15.91(665)	1563
Informed RRT*	49 / 51	663.75(19344)	177.58(11116)	0.99(46)	14.64(563)	636
KPIECE	21 / 51	24.20(10024)	1236.73(34943)	1.00(32)	10.72(157)	1662
PDST	7 / 51	65.20(12695)	579.09(13792)	1.44(48)	10.86(225)	1487
PRM	4 / 51	88.85(22391)	643.63(23700)	1.25(67)	7.94(184)	1150
PRM*	4 / 51	66.34(19056)	607.59(17927)	1.18(61)	8.09(187)	817
RRT	49 / 50	688.95(14108)	496.80(18433)	1.09(76)	13.67(312)	378
RRT#	44 / 51	692.14(13887)	145.06(9717)	1.00(47)	16.19(663)	1087
RRT*	46 / 51	692.09(13898)	152.48(10535)	1.01(43)	15.57(615)	1240
SORRT*	45 / 50	669.23(17653)	162.12(10961)	1.06(48)	16.22(713)	563
SPARS	0 / 47	165.72(17192)	662.60(16832)	0.98(32)	10.01(144)	317
SPARS2	7 / 51	28.64(6118)	549.87(12671)	1.19(43)	10.41(182)	517
SST	0	N/A	N/A	N/A	N/A	N/A
Theta*	9 / 9	504.14(14032)	364.74(1022)	0.79(19)	3.08(53)	36
Scenario: `Berlin_0_256` (Dubins steering, 6 minutes time limit)						
BFMT	3 / 39	39.60(6914)	517.98(6621)	0.25()	9.64(173)	343
BIT*	21 / 36	360.06(8)	363.68(1681)	0.25()	7.95(215)	18
CForest	6 / 51	360.07(3)	352.27(2064)	0.25(2)	7.52(218)	43
EST	39 / 51	59.46(13088)	675.37(16761)	0.25()	12.22(154)	372
Informed RRT*	19 / 50	360.04(3)	346.49(5074)	0.25(1)	7.82(219)	52
KPIECE	37 / 51	44.94(11576)	1210.09(39320)	0.25(2)	12.87(141)	885
PDST	31 / 49	48.08(11929)	524.89(11419)	0.25()	11.59(212)	124
PRM	0 / 51	360.05(3)	580.32(9338)	0.25()	8.20(258)	554
PRM*	0 / 51	360.07(5)	545.61(5986)	0.25()	7.31(197)	532
RRT	37 / 51	59.14(12673)	518.19(15543)	0.25(1)	12.21(231)	126
RRT#	2 / 51	360.03(2)	338.41(7186)	0.25(1)	7.42(209)	50
RRT*	23 / 51	360.04(3)	337.69(7068)	0.25()	7.50(215)	43
SORRT*	18 / 51	360.04(4)	347.88(5229)	0.25(1)	7.83(224)	59
SPARS	0 / 29	360.80(77)	569.63(8317)	0.25()	9.43(160)	104
SPARS2	0 / 30	360.01(1)	504.95(8048)	0.25()	9.39(151)	155
SST	39 / 49	360.02(2)	593.49(63750)	0.27(11)	10.75(338)	1375
Theta*	14 / 14	177.62(10184)	454.41(4319)	0.25()	7.66(155)	70

Table 3: Planning statistics using different steer functions from the `Berlin_0_256` scenario from the Moving AI benchmark.

4.5.1 Planning Time

Feasible planners are much faster and reliable in finding a single solution. RRT consistently ranked among the fastest of the planners we evaluated. While anytime (i.e., asymptotically optimal) planners require more time to find solutions, these are of higher quality than the paths found by feasible planners. The complexity of the steer function also severely impacts the performance of the planners. Dubins curves, for example, are computationally challenging systems for planning in very cluttered environments. An added burden on the runtime complexity stems from the polygon-based collision model, that, in contrast to typical point-based collision checkers, further penalizes algorithms that are not implemented in a way to make as few state validity checks as possible, such as our non-optimized Theta* implementation. Collision checking often consumed most of the allotted planning time such that this planner, in many cases, did not find any solution.

4.5.2 Quality of Anytime Solutions

Overall the results confirm what we know from the theory: on average, anytime planners obtain better solutions in terms of path length, number of cusps and maximum curvature. Informed anytime approaches (e.g., Informed-RRT*, SORRT*, BIT*) can achieve sometimes shorter paths throughout all tested steer functions. However, this is not always the case. These approaches are still impacted by larger complexity in the environment, and do not perform faster in highly constrained environments.

4.5.3 Variability of the Results

The main concern regarding sampling-based planners (feasible and anytime ones) is the high variance of the obtained results, which occasionally may lead to low performance. In particular, we believe that the stochasticity of the sampling phase is a major drawback that should be addressed from the community. Deterministic sampling [15, 29, 45] is an approach that mitigates this issue – see Section 5. State-lattice planners are an example of deterministic techniques, which, at the price of the solution quality, offer deterministic performance.

4.5.4 Post-smoothing Synergies

Post-smoothing combined with feasible planners is a good strategy in terms of planning efficiency and final path quality (sub-optimal and may not completely fulfill kinodynamic requirements). The results show that there exist several couplings of feasible sampling-based planners and post-smoothers that outperform anytime planners both in computation time and solution quality.

4.5.5 Environment Complexity

Our benchmarking confirms that the environments significantly influence the performance of the planners. Environments such as the polygon-based warehouse scenarios revealed vastly different solutions between the planners (see Figure 16).

As pointed out in Section 4.4.3, the planning performance is further impacted by different environment characteristics, such as narrow corridors and spaces cluttered with small obstacles. Plain state-of-the-art approaches that do not implement additional sampling heuristics, such as goal biasing, often fail to return solutions in very difficult environments where the corridors are small or the obstacle density is high.

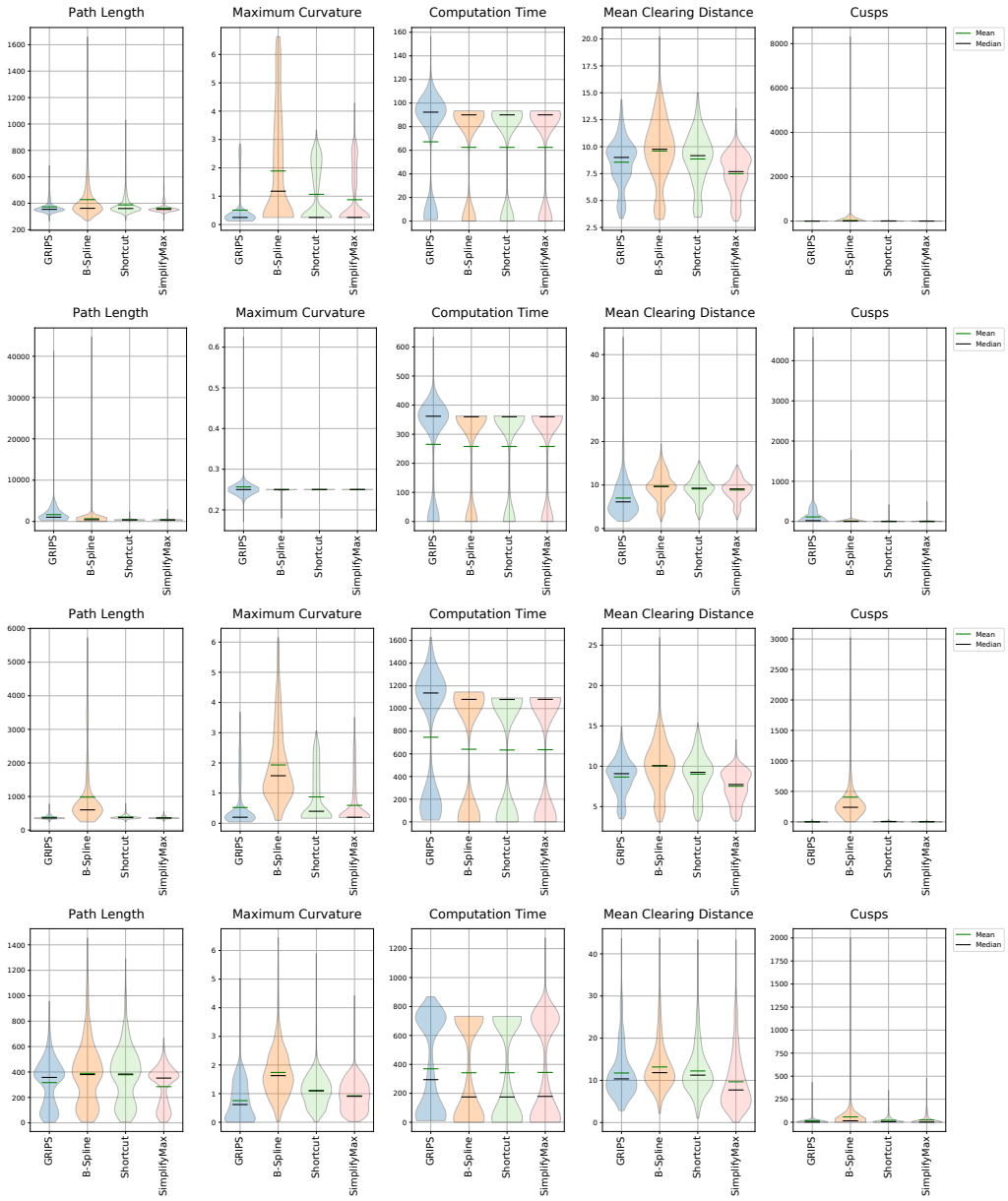

Figure 23: Planning statistics for the post-smoothing algorithms GRIPS, B-Spline, Shortcut, SimplifyMax (left to right per subplot) using different steer functions from the *Berlin_0_256* scenario from the Moving AI benchmark. These are the 50 most difficult start-goal configurations from the benchmark. First row: Reeds-Shepp steering, second row: Dubins steering, third row: CC Reeds-Shepp steering, fourth row: POSQ steering.

4.5.6 Influence of the Steer Function

Regarding the steer functions, we have observed two main phenomena which confirm previous theoretical claims [21]. Computationally complex steer functions, such as CC Reeds-Shepp, severly impact the planning efficiency of all the algorithms. Solving planning queries for systems with complex nonholonomic constraints in very cluttered environments also requires more planning time, particularly for systems which are not small-time locally controllable, such as Dubins curves. On the larger-scale experiments (e .g., Section 4.4.1) we observed a significant variance in the planning time allotment necessary for the planners to find solutions with different steer functions, ranging from 1.5 min (Reeds-Shepp) to more than 18 min (CC Reeds-Shepp).

5 Safety with Deterministic Sampling

5.1 Introduction

For motion planning in safety-critical applications, for instance in ILIAD intralogistic settings where collaborative robots operate amidst and work with humans, safety guarantees, explainability and deterministic performance bounds are of particular interest. In the past, many motion planning approaches have been introduced to improve planning efficiency, path quality and applicability across classes of robotic systems. Probabilistic sampling-based motion planners [**lavalle2001randomized**, 18, 22] and their optimal variants [16, 17] have been shown to outperform combinatorial approaches [27], especially for high-dimensional systems with complex differential constraints in cluttered environments. Sampling-based planners explore the configuration space by sampling states and connecting them to the roadmap, or tree, which keeps track of the state space connectivity. Typically samples are drawn from a uniform distribution over the state space by an independent and identically distributed (i.i.d.) random variable. The randomness of the sample set ensures good exploration of the configuration space, but comes at the expense of stochastic results which may strongly vary for each planning query in terms of planning efficiency and path quality. This stochasticity makes the formal verification and validation of such algorithms, needed for safety-critical applications, difficult to obtain.

To address this issue, several authors [15, 23] propose to use deterministic sets (or sequences). Contrarily to using i.i.d. random variables, this technique allows to achieve deterministic planning behaviors while still getting on par or even better performance. Moreover, as described also in [15, 23], deterministic sampling allows an easier certification process for the planners (e.g., in terms of final cost, clearance from the obstacles). With the goal to further enhance the usage of deterministic sampling to symmetric and optimal driftless systems, one outcome of ILIAD is *Dispertio*, an optimization-based approach to deterministic sampling. The method computes a sampling set which minimizes the actual dispersion of the samples. To compute the dispersion metric, we need access to a steer function [30, 35] that can compute an optimal path connecting two states. We prove that the approach, when combined with PRM*, is deterministically complete and retains asymptotic optimality. Furthermore, we systematically compare our approach to the existing baselines [15, 33]. Our experiments demonstrate that *Dispertio* outperforms the baselines in terms of planning efficiency and overall final path quality.

5.2 Dispertio

In this section, we describe our algorithm and analyze its properties. The approach to solve a motion planning problem by using an optimization-based sampling technique that minimizes the actual dispersion of the sampling set used by batch-processing algorithms

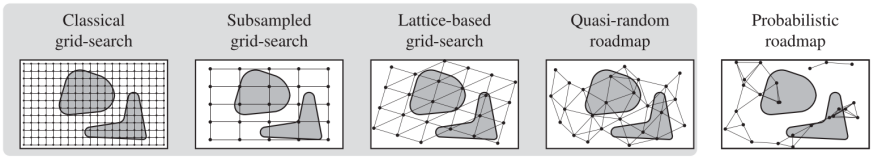

Figure 24: Range of possible sampling and roadmap types as introduced by LaValle [23]. The highlighted ones are deterministic.

Algorithm 1 PRM*. $\mathbf{x}_{\text{start}}$ is the start state, $\mathbf{x}_{\text{goal}}$ the goal state, n the desired number of samples.

```
1: procedure PRM*
2:     S ← SAMPLEFREE(n)
3:     V ← {x_start, x_goal} ∪ S
4:     for v in V do
5:         U ← NEAR(V, v, r_n)
6:         for u in U do
7:             if COLLISIONFREE(v, u) then
8:                 E ← E ∪ {(v, u)}
9:             end if
10:        end for
11:    end for
12:    return SHORTESTPATH(x_start, x_goal, (V, E))
13: end procedure
```

(e.g., PRM*[5], see Algorithm 1). Hereinafter we assume to use a dispersion metric, that is actually returning the path length of the path computed by an optimal steering function between two states [29].

5.2.1 The Dispersion Optimization Algorithm

As discussed by Janson et al. [15] and LaValle and Kuffner [24] multi-query sampling-based planners, such as PRM* or FMT*, generate as the initial step a set $\mathscr{S}$ of collision free samples, see line 2 of Algorithm 1. Instead of using i.i.d. random variables, or an existing deterministic technique to generate $\mathscr{S}$ (e .g., Halton sequence, [15, 23, 33]), we propose to compute the set by minimizing the defined dispersion. Our algorithm named Dispertio is outlined in Algorithm 2. The general idea of the algorithm is to pick in each step the sample (up to $n < N_{CS}$) that maximizes the distance to both the defined border of the configuration space as well as to the next sample. In other words we want to greedily put the sample into the position that currently defines the dispersion.

We propose to make this task computationally feasible by discretizing the configuration space into a fine grid of N_{CS} equidistant (distance could be different per dimension) cells. The dispersion tensor $\boldsymbol{D}$ keeps track of the minimum distance to either the border or closest sample for each grid cell (in Algorithm 2 we denote the dispersion value at the cell or position $\boldsymbol{c}$ as $D_{\boldsymbol{c}}$).

If it is possible to compute the distance to the border quickly (e .g., Euclidean case), we initialize $\boldsymbol{D}$ with the distance to the border for each grid cell, otherwise $\boldsymbol{D}$ is initialized with ∞, line 1 of Algorithm 2. In this case, we check whether the update step to a potential sample would affect any border sample. If this is the case, we will not add the sample to $\mathscr{S}$, but instead run an update step on the border sample without adding it. At each algorithm iteration, we generate a sample $\mathbf{x}_i$ that maximizes the current dispersion tensor $\boldsymbol{D}$ and add

[5] For brevity, we will not detail the algorithm PRM*. A reader interested to the properties of the algorithm can refer to [17].

Algorithm 2 Dispersion Optimization

```
1: procedure DISPERTIO
2:     D ← DISTANCETOBORDER
3:     while |S| < n do
4:         x_i ← argmax_c D_c
5:         UPDATEDISTANCEMATRIX(D, x_i)
6:         S ← S ∪ {x_i}
7:     end while
8: end procedure
```

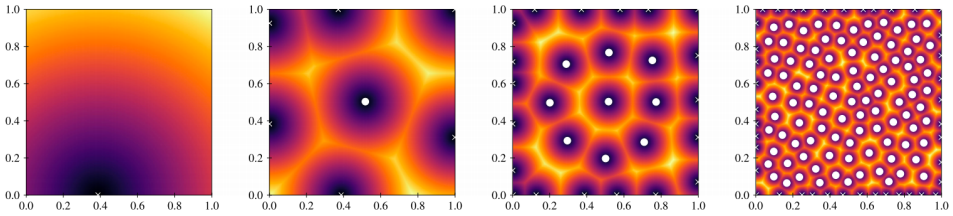

Figure 25: Progression of the algorithm in 2D Euclidean space. The background color indicates the distance to the next sample (i.e., the distance matrix $\boldsymbol{D}$). The white crosses and dots show the processed border points and actual samples respectively.

it to $\mathscr{S}$, see lines 3–7 of Algorithm 2. For a given sample, $\boldsymbol{D}$ is updated (line 5 of Algorithm 2) with a flood-fill algorithm, by only expanding cells for which the dispersion has been updated. In this way we are exploiting the connectedness of time-limited reachable sets. The flood-fill algorithm sequence can be pre-computed to prevent double checking of already tested cells.

Despite having a time complexity exponential in dimensions due to the flood-fill algorithm (i.e., $\mathscr{O}(n\xi^D)$, with the constant $\xi > 0$ being related to discretization and complexity of dist), the algorithm is a feasible pre-computation step for many systems (e .g., Reeds-Shepp space, 6D kinematic chain using Euclidean distance).

Once the set $\mathscr{S}$ has been generated, we can then use it in a motion planning algorithm such as PRM* (Algorithm 1). PRM*-edges are generated with the same steer function used to optimize the set $\mathscr{S}$. As detailed in [29], PRM* [17] when using *Dispertio*, retains the completeness and asymptotic optimality properties as in [15, 23, 33].

A key advantage of our approach in terms of safety is that once you know the allowed dispersion in your environment (i.e. size of the narrowest corridor where the corridor could go into), you can then define the resolution completeness of your planner [29].

5.3 Discussion

In ILIAD, as part of the efforts to improve the final motion planning architecture, we extend deterministic sampling-based motion planning to the class of symmetric and optimal driftless systems, by proposing Dispertio, an algorithm for optimized deterministic sampling set generation. When used in combination with PRM*, we prove that the approach is deterministically complete and retains asymptotic optimality. In the evaluation (see [29] for more details) we show that our sampling technique outperforms state-of-the-art methods in terms of solution cost and planning efficiency, while also converging faster to lower cost solutions. For instance, Figure 27 shows an example planning query for i.i.d., Halton sampling and our approach. It reports the obtained paths, the trend for the success rate and the cost progression. The blue range shows the minimum and maximum cost observed in these runs for the i.i.d. sampler. The cost results are only shown for

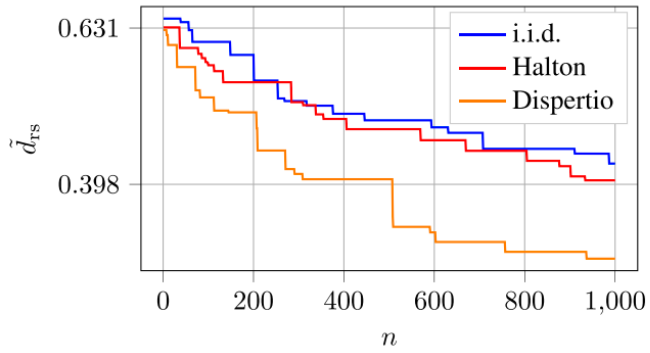

Figure 26: Dispersion trend for the Reeds-Shepp case ($\eta = 1.0$, obstacle free environment). Our approach obtains a better dispersion than the baselines, thus achieving a better coverage of the state space.

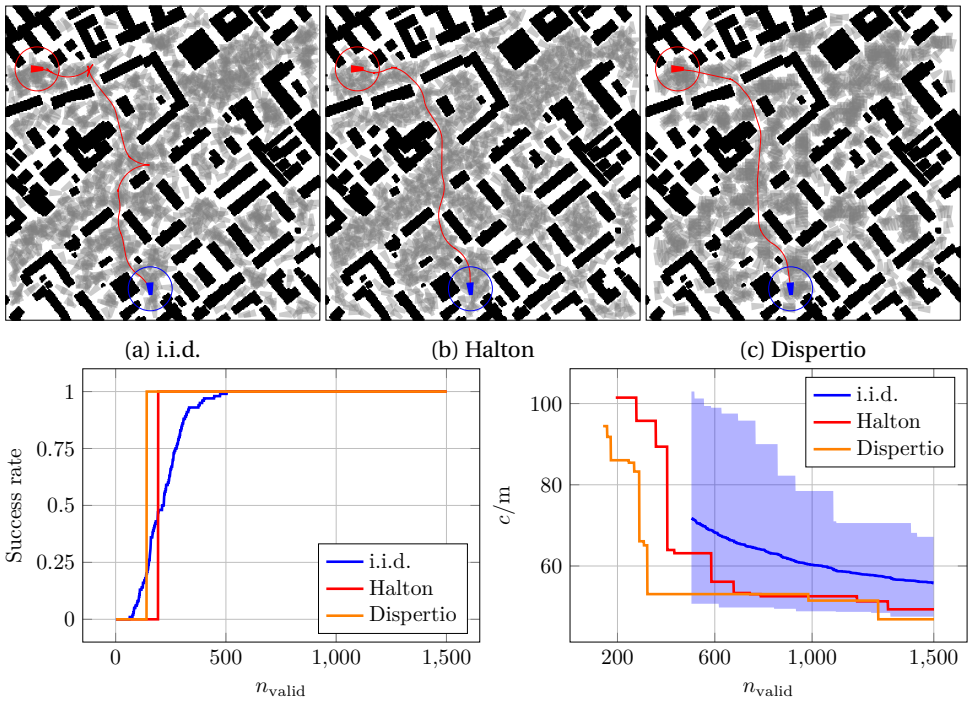

Figure 27: Qualitative comparison of i.i.d., Halton and Dispertio. The top row shows example paths obtained after 1500 valid samples connecting starts (in **red**) with goals (in **blue**). The gray footprints represent the roadmap's vertices. The bottom row shows success rate and cost for this example.

success rates of 100%. Cost and success rate progressions of Figure 27 highlight how our approach is faster in getting an initial good solution, and faster (as the number of samples increases) in converging to lower cost solutions in those cluttered and narrow scenarios. Furthermore, Figure 26 reports a numerical comparison of the dispersion obtained for the Reeds-Shepp case after n samples for i.i.d., Halton and the proposed approach in an

obstacle-free environment. Our approach achieves better dispersion than the baselines.

As future work, we are interested in extending the approach towards non-uniform sampling schemes, for example to exploit learned priors, and to systems with drift.

6 NMPC for Navigation in Cluttered Environments

ILIAD intralogistic settings require robots to operate in dynamic environments among other agents, such as humans or other autonomous systems. In these scenarios, the reactive avoidance of unforeseen dynamic obstacles is an important requirement. Combined with the objective of reaching optimal robot behavior, this poses a major challenge for motion planning and control and remains the subject of active research. As part of this deliverable we have investigated the usage of novel model predictive control techniques for achieving fast, reliable and safe obstacle avoidance.

Recently, several researchers have tackled the obstacle avoidance problem by formulating and solving optimization problems [4, 5, 7, 9, 13, 26, 28, 34, 36, 38, 42, 46–48]. This approach is well suited for finding locally optimal solutions, but generally gives no guarantee of finding the global optimum. A shortcoming of most common trajectory optimization methods is that they are incapable of respecting kinodynamic constraints, e.g. bounds on the acceleration, and typically lack a notion of time in their predictions, [5, 34, 38, 48]. These approaches are typically limited to the optimization of paths rather than trajectories and impose constraints by introducing penalties.

To counteract these several issues, we have developed in ILIAD a novel model predictive control formulation that allows a fast and safe collision avoidance for highly nonlinear problems.

6.1 The Approach

Our nonlinear model predictive control approach allows us to find a kinodynamically feasible, collision free trajectory by formulating and solving a constrained optimal control problem (OCP). Kinodynamic feasibility is ensured by using a dynamical model to simulate the robot's behavior and collision avoidance is achieved constraining the robot to positions with a *minimum distance* to all obstacles.

As described in Schoels et al. [37], instead of computing the actual true distance to the obstacles, we approximate it via a novel convex inner formulation of the collision avoidance constraint (hereinafter called CIAO), that pushes the optimization to find collision-free robot trajectories into dynamic collision-free balls (centered around a tuple of center points $\mathbf{C}$).

6.1.1 The CIAO-iteration

We will now introduce the CIAO-iteration, as detailed in Algorithm 3. It takes a two step approach. We first find a set of collision-free regions centered around the points $\mathbf{C} = (\mathbf{c}_0, \ldots, \mathbf{c}_N)$, and then we solve the optimization for finding a trajectory that lies withing the collision free areas.

Algorithm 3 the CIAO-iteration

1: **function** CIAO-ITERATION($\mathbf{w}$; $\mathbf{r}$, $\bar{\underline{\mathbf{x}}}_0$, Δt)
2: $\quad\mathbf{C} \leftarrow (\mathbf{c}_k = S_\mathbf{p} \cdot \mathbf{x}_k$ for $k = 0, \ldots, N)$
3: $\quad\mathbf{C}^* \leftarrow (\mathbf{c}^* = \text{MAXIMIZEFB}(\mathbf{c})$ for all $\mathbf{c} \in \mathbf{C})$
4: $\quad\mathbf{w}^* \leftarrow$ SOLVENONLINEAR PROGRAM (NLP)($\mathbf{w}$; $\mathbf{C}^*$, $\mathbf{r}$, $\bar{\underline{\mathbf{x}}}_0$, Δt)
5: **end function return** $\mathbf{w}^*$ ▷ return newly found trajectory

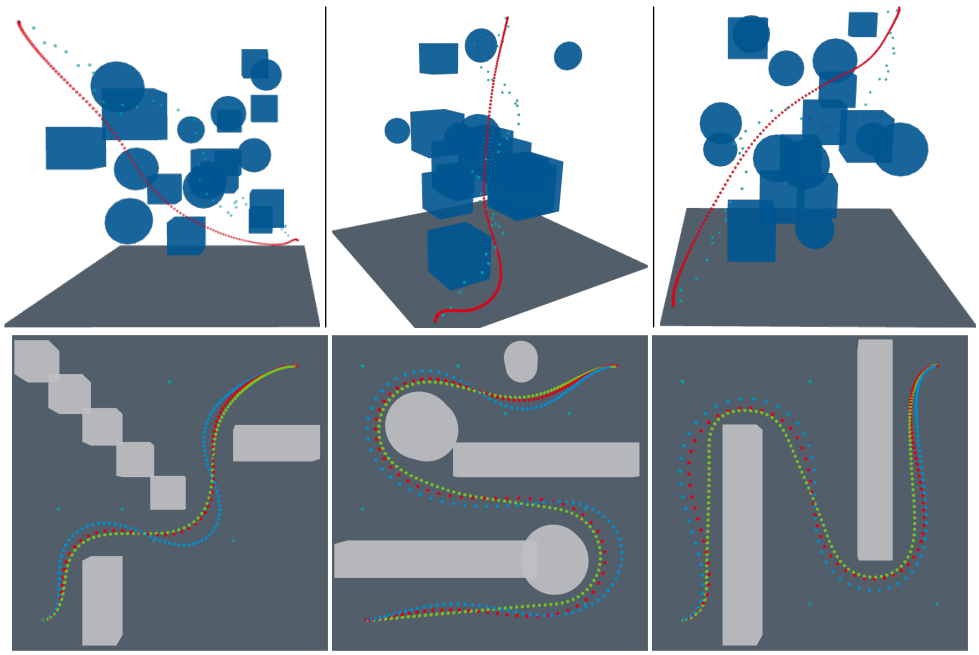

Figure 28: CIAO trajectories for the Astrobee robot (top row) in red and for a unicycle robot (last row) with three different maximum speeds $v_{\max}$ and corresponding minimum distances: green - slow, red - normal, blue - fast. The boxes and spheres represent obstacles, the turquoise dots the reference path. A wider spacing between the dots indicates a higher speed. The start is always located in the bottom and the goal in the top.

In Line 2 we find an initial tuple of center points $\mathbf{C}$. In practice the free balls resulting from these center points are very small and therefore very restrictive, which leaves little room for optimization, especially if the initial guess of the trajectory $\mathbf{w}$ approaches obstacles closely. To overcome this problem we maximize free balls (FBs) (Line 3) by performing a gradient descent on the distance field representing the collision free regions.

6.2 CIAO-based Motion Planning

We now report two ways to use the CIAO iteration.

6.2.1 CIAO for Trajectory Optimization

The first way to use CIAO is for trajectory optimization considering a long horizon (i.e., the entire environment). The proposed trajectory optimization algorithm (see Algorithm 4) starts by computing a feasible initial guess and a reference trajectory (Lines 1–2). In the

Algorithm 4 CIAO for offline trajectory optimization

Require: $\mathbf{x}_S$, $\mathbf{x}_G$, Δt, ε ▷ start and goal state
1: $\mathbf{w}^* \leftarrow$ INITIALGUESS($\mathbf{x}_S$, $\mathbf{x}_G$, Δt) ▷ feasible initialization
2: $\mathbf{r} \leftarrow$ REFERENCETRAJECTORY($\mathbf{x}_S$, $\mathbf{x}_G$, Δt)
3: **do**
4: $\quad \mathbf{w} \leftarrow \mathbf{w}^*$ ▷ set last solution as initial guess
5: $\quad \mathbf{w}^* \leftarrow$ CIAO-ITERATION($\mathbf{w}$; $\mathbf{r}$, $\bar{\underline{\mathbf{x}}}_0$, Δt) ▷ $\bar{\underline{\mathbf{x}}}_0 = \mathbf{x}_S$
6: **while** COST($\mathbf{w}^*$) − COST($\mathbf{w}$) $> \varepsilon$
7: **return** $\mathbf{w}^*$

general case of nonconvex scenarios, such as cluttered environments, feasible initializations can be obtained through a sampling-based motion planner [31, 32]. To monitor the progress the initial guess is copied (Line 4), before using it as initial guess for the CIAO-ITERATION (Line 5).

Lines 4–5 are repeated as long as the COST-function shows an improvement that exceeds a given threshold ε (Line 6). Finally the best known solution $\mathbf{w}^*$ is returned (Line 7). For trajectory optimization the terminal constraint becomes an equality constraint, which enforces that the goal state $\mathbf{x}_G$ is reached at the end of the horizon, i.e. $\mathbb{X}_T = \{\mathbf{x}_G\}$.

6.2.2 CIAO-NMPC

For obstacle avoidance and trajectory tracking we use the CIAO iteration inside an MPC framework. Concretely Algorithm 5 uses a shorter horizon than Algorithm 4 and runs only one CIAO-iteration before shifting it one step forward (Line 7). Therefore the initial guess $\mathbf{w}$ (Line 1) is not required to reach the goal state $\mathbf{x}_G \in \mathbb{X}_G$.

Algorithm 5 CIAO-NMPC

Require: $\bar{\underline{\mathbf{x}}}_0$, $\mathbf{x}_G$, Δt, $\mathbb{X}_G$ ▷ current and goal state
1: $\mathbf{w} \leftarrow$ INITIALGUESS($\bar{\underline{\mathbf{x}}}_0$, $\mathbf{x}_G$, Δt) ▷ feasible initialization
2: **while** $\bar{\underline{\mathbf{x}}}_0 \notin \mathbb{X}_G$ **do**
3: $\quad \bar{\underline{\mathbf{x}}}_0 \leftarrow$ GETCURRENTSTATE()
4: $\quad \mathbf{r} \leftarrow$ REFERENCETRAJECTORY($\bar{\underline{\mathbf{x}}}_0$, $\mathbf{x}_G$, Δt)
5: $\quad \mathbf{w}^* \leftarrow$ CIAO-ITERATION($\mathbf{w}$; $\mathbf{r}$, $\bar{\underline{\mathbf{x}}}_0$, Δt) ▷ Alg. 3
6: $\quad$ APPLYFIRSTCONTROL($\mathbf{w}^*$) ▷ recall $\mathbf{u}_0 \in \mathbf{w}^*$
7: $\quad \mathbf{w} \leftarrow$ SHIFTTRAJECTORY($\mathbf{w}^*$) ▷ recede horizon
8: **end while**

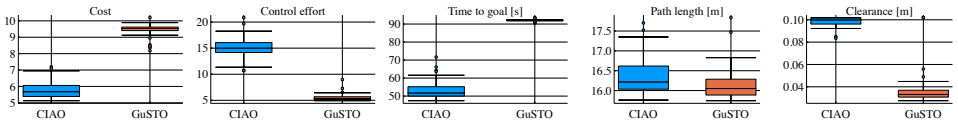

Figure 29: Trajectory Optimization Benchmark Results. CIAO finds faster trajectories with higher clearance than GuSTO.

measure	CIAO	GuSTO
Compute [s]	14.792±11.966	131.367±130.743
Iterations	30.660±15.886	4.520±1.282
Compute / Iteration [s]	0.475±0.230	27.729±22.096
Linearization Error	4.66e-14±3.67e-15	4.10e-06±1.28e-06

Table 4: Numerical Performance: Average ± std values.

While the robot has not reached the goal region $\mathbb{X}_G$ (Line 2), it is iteratively steered to it (Lines 3–8). Each iteration starts by updating the robot's current state $\mathbf{x}_0$. Based on the complexity of the scenario REFERENCETRAJECTORY may return a guiding trajectory to the goal or just the goal state itself (Line 4). We run Algorithm 3 to compute a new trajectory (Line 5), before sending the first control to the robot (Line 6).

6.3 Experiments and Discussion

To evaluate CIAO in terms of planning efficiency and final trajectory quality, we compare it against a set of baselines. We challenge CIAO by using nonlinear dynamics and a nonconvex cost function. Further we use a sampling based motion planner to initialize it with a collision free path that does not satisfy the robot's dynamics. In this case, we use the primal-dual interior point NLP-solver Ipopt [44] with the linear solver MA-27 [14] called through CasADi [1].

6.3.1 Trajectory Optimization Benchmark

In a set of experiments CIAO is compared to GuSTO [4] using the implementation publicly provided by the authors. In these experiments we consider a free-flying Astrobee Robot with 12 states and 6 controls, that has to move from a start position on the bottom front left corner of a $10 \times 10 \times 10$ m cube to a goal in the opposite corner. The room between start and goal point is cluttered with 25 randomly placed static obstacles of varying sizes (between 1 and 2 meters). Figure 28 shows some examples.

The results reported in Table 4 and Figure 29 were performed using JuMP [6] on an Intel Core i7-8559U (2.7GHz) running MacOS. The sequential convex programmings (SCPs) formulated by GuSTO [4] are solved with Gurobi [10]. Both algorithms are provided with the same initial guess, which is computed with RRT [24]. We use a horizon of 250 steps and a sampling time of 0.4 s. Since both GuSTO and CIAO use tailored cost functions we evaluate the computed trajectories using a common cost function J_ρ, which is based on the state distance metric $\rho : R^{n_\mathbf{x}} \times R^{n_\mathbf{x}} \to \mathbb{R}$ proposed by [24]: $J_\rho(\mathbf{w}; \mathbf{x}_G) = \sum_{k=0}^{N} \rho(\mathbf{x}_k, \mathbf{x}_G)$, with goal state $\mathbf{x}_G$ and all weights of the distance metric chosen equal. The controls are evaluated separately and reported as control effort given by $J_\mathbf{u}(\mathbf{w}) = \sum_{k=0}^{N-1} \Delta t \cdot \|\mathbf{u}_k\|_1$. The path quality is evaluated in terms of time to goal, path length, and clearance (minimum

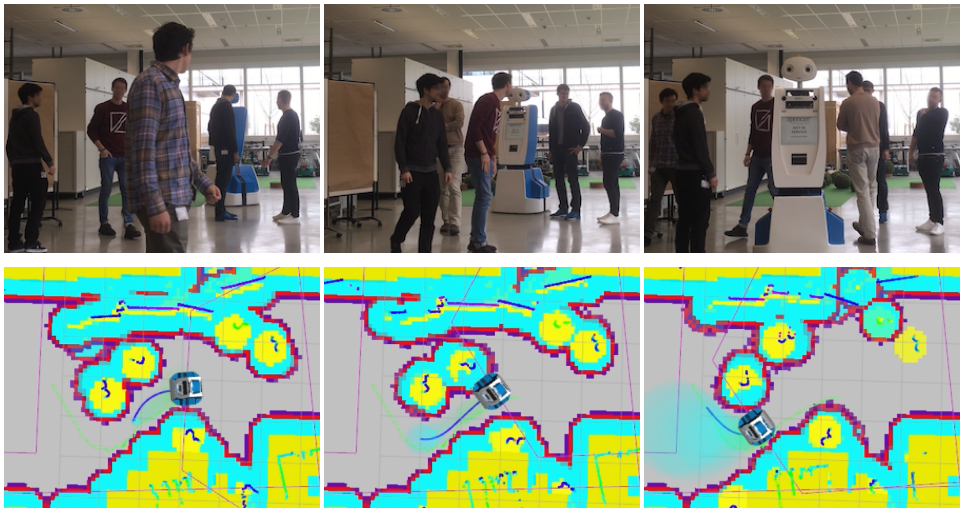

Figure 30: CIAO steers a wheeled mobile robot through a group of people. Real-world (top) and RViz (bottom): Planned trajectory as blue line, free balls as transparent circles, obstacles in yellow, safety margin in light blue.

distance to the closest obstacle along the trajectory). The first two measures take the time and path length until the state distance metric falls below a threshold of 0.5, while the latter is evaluated on the entire trajectory. These three metrics are evaluated on an oversampled trajectory using a sampling time $\Delta t = 0.01\,\mathrm{s}$.

The results in Figure 29 show that CIAO finds faster trajectories than GuSTO and thereby also achieves significantly lower cost. As depicted in Figure 28 it maintains a larger distance to obstacles for higher speeds. This behavior allows for a higher average speed, at the cost of a higher control activation and slightly longer paths in comparison to GuSTO.

As reported in Table 4, CIAO (Algorithm 4) requires more iterations to converge, but the individual iterations are cheaper. Moreover CIAO obtains a feasible trajectory after the first iteration and therefore could be terminated early, while GuSTO does not have this property and takes several iterations to find a feasible trajectory. Even though the dynamics are mostly linear we observe linearization errors for GuSTO, originating from the linear model they use. In summary CIAO finds trajectories of higher quality than GuSTO at lower computational effort.

6.3.2 Real-World Experiments - Differential Drive Robot

To qualitatively assess the behavior of CIAO-NMPC (Algorithm 5), it was tested in dynamic real-world scenarios with freely moving humans. A representative example is depicted in Figure 30. Note that CIAO has no knowledge of the humans' future movements. It is instead considering all humans as static obstacles in their current position. A differential drive robot is used, this time with a horizon of 5 s and a control frequency of 10 Hz resulting in a total of 405 optimization variables (including slacks). For these experiments CIAO was implemented as a C++ ROS-module, the distance function was realized as distance field based on the code by Lau et al. [20]. Initial guesses and reference paths were computed using an A* algorithm [11].

Since GuSTO is not suitable for receding horizon control (RHC), we used an extended version of the elastic-band (EB) method [34]. To obtain comparable results, we used the

same A* planner and localization method with both algorithms. In summary CIAO and the elastic band (EB) approach show similar behavior. In contrast to EB, CIAO computes kinodynamically feasible and guaranteed continuous time collision free trajectories. Further it has a notion of time for the planned motion, such that predictions for dynamic environments can be incorporated in future work.

7 Conclusions

In this deliverable we have detailed the quantitative motion planning architecture developed in the ILIAD project, a benchmark of the state of the art of planning algorithms and novel techniques to further improve safety and efficiency of the ILIAD planning system. The benchmark and the real-work experiments show the efficiency and the reliability of the developed solution. Overall the techniques allow for a safe and human-aware long-term robot operation in intralogistic settings.

References

[1] Joel A. E. Andersson, Joris Gillis, Greg Horn, James B. Rawlings, and Moritz Diehl. "CasADi: a software framework for nonlinear optimization and optimal control". In: *Mathematical Programming Computation* (2018).

[2] Henrik Andreasson, Jari Saarinen, Marcello Cirillo, Todor Stoyanov, and Achim J Lilienthal. "Fast, continuous state path smoothing to improve navigation accuracy". In: *Int. Conf. on Robotics and Automation (ICRA)*. 2015.

[3] Holger Banzhaf, Luigi Palmieri, Dennis Nienhüser, Thomas Schamm, Steffen Knoop, and J Marius Zöllner. "Hybrid curvature steer: A novel extend function for sampling-based nonholonomic motion planning in tight environments". In: *2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC)*. IEEE. 2017, pp. 1–8.

[4] Riccardo Bonalli, Abhishek Cauligi, Andrew Bylard, and Marco Pavone. "GuSTO: Guaranteed Sequential Trajectory Optimization via Sequential Convex Programming". In: *Int. Conf. on Robotics and Automation (ICRA)*. 2019.

[5] Oliver Brock and Oussama Khatib. "Elastic Strips: A Framework for Motion Generation in Human Environments". In: *Int. Journal of Robotics Research* 21.12 (2002), pp. 1031–1052. DOI: 10.1177/0278364902021012002.

[6] Iain Dunning, Joey Huchette, and Miles Lubin. "JuMP: A Modeling Language for Mathematical Optimization". In: *SIAM Review* 59.2 (2017), pp. 295–320. DOI: 10.1137/15M1020575.

[7] Timm Faulwasser and Rolf Findeisen. "Nonlinear Model Predictive Control for Constrained Output Path Following". In: *TAC* 61.4 (2016), pp. 1026–1039. DOI: 10.1109/TAC.2015.2466911.

[8] Thierry Fraichard and Alexis Scheuer. "From Reeds and Shepp's to continuous-curvature paths". In: *IEEE Transactions on Robotics* 20.6 (2004), pp. 1025–1035.

[9] J. V. Frasch, A. J. Gray, M. Zanon, H. J. Ferreau, S. Sager, F. Borrelli, and M. Diehl. "An Auto-generated Nonlinear MPC Algorithm for Real-Time Obstacle Avoidance of Ground Vehicles". In: *Proc. of the European Control Conf. (ECC)*. 2013, pp. 4136–4141.

[10] Gurobi Optimization, LLC. *Gurobi Optimizer Reference Manual*.

[11] Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. "A Formal Basis for the Heuristic Determination of Minimum Cost Paths". In: *IEEE Transactions on Systems Science and Cybernetics* 4.2 (1968), pp. 100–107. DOI: 10.1109/TSSC.1968.300136.

[12] Eric Heiden, Luigi Palmieri, Kai O. Arras, Gaurav S. Sukhatme, and Sven Koenig. "Experimental Comparison of Global Motion Planning Algorithms for Wheeled Mobile Robots". In: *arXiv preprint arXiv:2003.03543* (2020).

[13] Sylvia L. Herbert, Mo Chen, SooJean Han, Somil Bansal, Jaime F. Fisac, and Claire J. Tomlin. "FaSTrack: a Modular Framework for Fast and Guaranteed Safe Motion Planning". In: *Proc. of the IEEE Int. Conf. on Decision and Control (CDC)*. 2017, pp. 1517–1522. DOI: 10.1109/CDC.2017.8263867.

[14] HSL. *A collection of Fortran codes for large scale scientific computation*. http://www.hsl.rl.ac.uk. 2011.

[15] Lucas Janson, Brian Ichter, and Marco Pavone. "Deterministic sampling-based motion planning: Optimality, complexity, and performance". In: *The International Journal of Robotics Research* 37.1 (2018), pp. 46–61.

[16] Lucas Janson, Edward Schmerling, Ashley Clark, and Marco Pavone. "Fast marching tree: A fast marching sampling-based method for optimal motion planning in many dimensions". In: *Int. Journal of Robotics Research* 34 (2015).

[17] Sertac Karaman and Emilio Frazzoli. "Sampling-based algorithms for optimal motion planning". In: *The international journal of robotics research* 30.7 (2011), pp. 846–894.

[18] LE Kavraki, P Svestka, J-C Latombe, and MH Overmars. "Probabilistic roadmaps for path planning in high-dimensional configuration spaces". In: *IEEE Transactions on Robotics and Automation* 12 (1996).

[19] Tomasz P. Kucner, Martin Magnusson, Erik Schaffernicht, Victor H. Bennetts, and Achim J. Lilienthal. "Enabling Flow Awareness for Mobile Robots in Partially Observable Environments". In: *IEEE Robotics and Automation Letters* 2.2 (Apr. 2017), pp. 1093–1100. ISSN: 2377-3766. DOI: 10.1109/LRA.2017.2660060.

[20] Boris Lau, Christoph Sprunk, and Wolfram Burgard. "Efficient grid-based spatial representations for robot navigation in dynamic environments". In: *Robotics and Autonomous Systems* 61.10 (2013), pp. 1116–1130. DOI: 10.1016/j.robot.2012.08.010.

[21] Jean-Paul Laumond, S Sekhavat, and F Lamiraux. "Guidelines in nonholonomic motion planning for mobile robots". In: *Robot motion planning and control*. Springer, 1998, pp. 1–53.

[22] Steven M. LaValle. *Planning Algorithms*. Cambridge Univ. Press, 2006.

[23] Steven M LaValle, Michael S Branicky, and Stephen R Lindemann. "On the relationship between classical grid search and probabilistic roadmaps". In: *Int. Journal of Robotics Research* 23 (2004).

[24] Steven M. LaValle and James J. Kuffner Jr. "Randomized Kinodynamic Planning". In: *Int. Journal of Robotics Research* 20.5 (2001), pp. 378–400. DOI: 10.1177/02783640122067453.

[25] Maxim Likhachev, David I Ferguson, Geoffrey J Gordon, Anthony Stentz, and Sebastian Thrun. "Anytime Dynamic A*: An Anytime, Replanning Algorithm." In: *ICAPS*. Vol. 5. 2005, pp. 262–271.

[26] A. Liniger, A. Domahidi, and M. Morari. "Optimization-based autonomous racing of 1:43 scale RC cars". In: *Optimal Control Applications and Methods* 36.5 (2015), pp. 628–647.

[27] Tomás Lozano-Pérez and Michael A Wesley. “An algorithm for planning collision-free paths among polyhedral obstacles”. In: *Communications of the ACM* 22 (1979).

[28] Michael Neunert, Cedric de Crousaz, Fadri Furrer, Mina Kamel, Farbod Farshidian, Roland Siegwart, and Jonas Buchli. “Fast Nonlinear Model Predictive Control for Unified Trajectory Optimization and Tracking”. In: *Int. Conf. on Robotics and Automation (ICRA)*. 2016, pp. 1398–1404. DOI: 10.1109/ICRA.2016.7487274.

[29] L. Palmieri, L. Bruns, M. Meurer, and K. O. Arras. “Dispertio: Optimal Sampling for Safe Deterministic Motion Planning”. In: *IEEE Robotics and Automation Letters* 5.2 (2019). ISSN: 2377-3774. DOI: 10.1109/LRA.2019.2958525.

[30] Luigi Palmieri and Kai O Arras. “A novel RRT extend function for efficient and smooth mobile robot motion planning”. In: *Int. Conf. on Intelligent Robots and Systems (IROS)*. Chicago, USA, 2014.

[31] Luigi Palmieri and Kai O Arras. “Distance metric learning for RRT-based motion planning with constant-time inference”. In: *2015 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2015, pp. 637–643.

[32] Luigi Palmieri, Tomasz P Kucner, Martin Magnusson, Achim J Lilienthal, and Kai O Arras. “Kinodynamic motion planning on Gaussian mixture fields”. In: *Int. Conf. on Robotics and Automation (ICRA)*. Singapore, 2017.

[33] Ernesto Poccia. “Deterministic Sampling-Based Algorithms for Motion Planning under Differential Constraints”. MA thesis. Stanford University, University of Pisa, 2017.

[34] Sean Quinlan and Oussama Khatib. “Elastic Bands: Connecting Path Planning and Control”. In: *Int. Conf. on Robotics and Automation (ICRA)*. Vol. 2. 1993, pp. 802–807. DOI: 10.1109/ROBOT.1993.291936.

[35] James Reeds and Lawrence Shepp. “Optimal paths for a car that goes both forwards and backwards”. In: *Pacific Journal of Mathematics* 145 (1990).

[36] Christoph Rösmann, Frank Hoffmann, and Torsten Bertram. “Integrated online trajectory planning and optimization in distinctive topologies”. In: *Robotics and Autonomous Systems* 88 (2017), pp. 142–153. DOI: 10.1016/j.robot.2016.11.007.

[37] Tobias Schoels, Luigi Palmieri, Kai O Arras, and Moritz Diehl. “An NMPC Approach using Convex Inner Approximations for Online Motion Planning with Guaranteed Collision Avoidance”. In: *Int. Conf. on Robotics and Automation (ICRA)*. 2020.

[38] John Schulman et al. “Motion planning with sequential convex optimization and convex collision checking”. In: *Int. Journal of Robotics Research* 33.9 (2014), pp. 1251–1270. DOI: 10.1177/0278364914528132.

[39] N. Sturtevant. “Benchmarks for Grid-Based Pathfinding”. In: *Transactions on Computational Intelligence and AI in Games* 4 (2012).

[40] Ioan A. Şucan, Mark Moll, and Lydia E. Kavraki. “The Open Motion Planning Library”. In: *IEEE Robotics & Automation Magazine* 19.4 (Dec. 2012). http://ompl.kavrakilab.org, pp. 72–82. DOI: 10.1109/MRA.2012.2205651.

[41] Chittaranjan Srinivas Swaminathan, Tomasz Piotr Kucner, Martin Magnusson, Luigi Palmieri, and Achim J Lilienthal. “Down the CLiFF: Flow-aware trajectory planning under motion pattern uncertainty”. In: *Int. Conf. on Intelligent Robots and Systems (IROS)*. 2018.

[42] D. Verscheure, B. Demeulenaere, J. Swevers, J. De Schutter, and M. Diehl. “Time-Optimal Path Tracking for Robots: a Convex Optimization Approach”. In: *TAC* 54 (2009), pp. 2318–2327.

[43] Tomáš Vintr et al. “Time-varying pedestrian flow models for service robots”. In: *Proc. of the European Conf. on Mobile Robots (ECMR)*. 2019.

[44] A. Wächter and L. Biegler. *IPOPT - an Interior Point OPTimizer*. https://projects.coin-or.org/Ipopt. 2009.

[45] Anna Yershova and Steven M LaValle. “Deterministic sampling methods for spheres and SO(3)”. In: *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA'04. 2004*. Vol. 4. IEEE. 2004, pp. 3974–3980.

[46] Xiaojing Zhang, Alexander Liniger, and Francesco Borrelli. “Optimization-Based Collision Avoidance”. In: *arXiv preprint arXiv:1711.03449* (2017).

[47] Zhijie Zhu, Edward Schmerling, and Marco Pavone. “A Convex Optimization Approach to Smooth Trajectories for Motion Planning with Car-Like Robots”. In: *Proc. of the IEEE Int. Conf. on Decision and Control (CDC)*. 2015, pp. 835–842. DOI: 10.1109/CDC.2015.7402333.

[48] Matt Zucker, Nathan Ratliff, Anca D. Dragan, Mihail Pivtoraiko, Matthew Klingensmith, Christopher M. Dellin, J. Andrew Bagnell, and Siddhartha S. Srinivasa. “CHOMP: Covariant Hamiltonian Optimization for Motion Planning”. In: *Int. Journal of Robotics Research* 32.9–10 (2013), pp. 1164–1193. DOI: 10.1177/0278364913488805.