2023 IEEE International Conference on Robotics and Automation (ICRA 2023)
May 29 - June 2, 2023. London, UK

Obstacle avoidance using Raycasting and Riemannian Motion Policies at kHz rates for MAVs

Michael Pantic*, Isar Meijer*, Rik Bähnemann, Nikhilesh Alatur, Olov Andersson,
Cesar Cadena, Roland Siegwart, and Lionel Ott

***Abstract*— This paper presents a novel method for using Riemannian Motion Policies on volumetric maps, shown in the example of obstacle avoidance for Micro Aerial Vehicles (MAVs). Today, most robotic obstacle avoidance algorithms rely on sampling or optimization-based planners with volumetric maps. However, they are computationally expensive and often have inflexible monolithic architectures. Riemannian Motion Policies are a modular, parallelizable, and efficient navigation alternative but are challenging to use with the widely used voxel-based environment representations. We propose using GPU raycasting and tens of thousands of concurrent policies to provide direct obstacle avoidance using Riemannian Motion Policies in voxelized maps without needing map smoothing or pre-processing. Additionally, we present how the same method can directly plan on LiDAR scans without any intermediate map. We show how this reactive approach compares favorably to traditional planning methods and can evaluate up to 200 million rays per second. We demonstrate the planner successfully on a real MAV for static and dynamic obstacles. The presented planner is made available as an open-source package[1].**

I. Introduction

From the moment mobile robots could move through the world autonomously, obstacle avoidance algorithms have been fundamental. Obstacle avoidance is especially critical and challenging for flying robots, as they are usually not collision tolerant and have limited computational and sensing capabilities. These challenges have sparked a wide variety of research in obstacle avoidance for *micro aerial vehicle*s (MAVs), from sensor-based reactive controllers [1] [2] to onboard map building for sampling-based and optimization-based methods [3] to recent data-driven approaches [4]. Early methods, such as potential fields [5], are computationally efficient and versatile. However, *how* to design such potential functions in real-world applications is not apparent. Map-based approaches, e.g., occupancy maps and *signed distance field*s (SDFs), filter and extract information to obtain obstacle gradients and occupancy probability. While these maps are highly effective for global sampling-based or optimization-based planning, the typical mapping-planning cycle is computationally expensive, relies on accurate state estimates, and introduces aliasing and delays. Additionally, there is the risk that the map is simply wrong or outdated, giving rise to the need for a navigation paradigm that can effortlessly combine map and live sensor data. While end-to-end methods have shown great promise and robustness recently, they often suffer in generalizability, introspectability, and modularity.

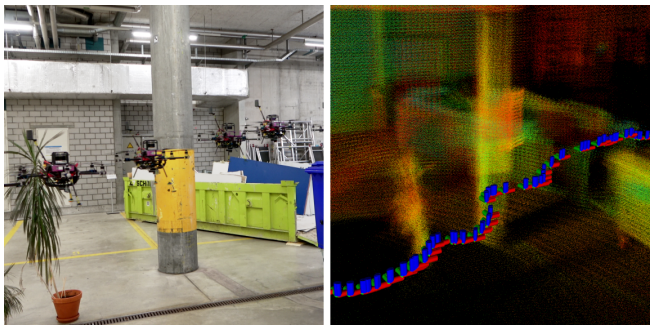

Fig. 1. Left: An MAV avoiding obstacles using the proposed navigation algorithm without a map, but raw LiDAR data instead. We only command the desired goal location. Right: Accumulated point cloud for visualization and the executed trajectory.

A promising class of navigation algorithms is *Riemannian motion policies* (RMPs) [6]. Most works using RMPs focus on settings centered around (self-intersections of) robotic arms or structured environments, where idealized environment representations, such as meshes or primitives, are available. So far, planning research has not shown how to use RMPs efficiently on voxelized maps in a realistic onboard setting. Earlier work considers a single policy to the closest obstacle [7] or needs to convert the voxelized map to a different representation, such as a geodesic field [8]. This paper presents and investigates highly parallelizable Riemannian Motion Policies for obstacle avoidance in unstructured environments on MAVs on voxel-based map representation. To this end, we contribute:

- a novel raycasting-based policy generation method capable of evaluating thousands of policies in parallel on 3D voxel maps;
- an extension to synthesize motion policies directly from LiDAR data;

Additionally, the proposed navigation algorithms are evaluated on actual flight experiments and made available open-source. The modular nature of RMPs facilitates the easy integration of new policies and adjusting to novel scenarios, with the proposed policies providing building blocks for further research and practical applications.

II. Related work

In the following, we present relevant related work focusing on applying RMPs for MAV navigation in unstructured envi-

* Authors contributed equally to this work.
All authors are with the Autonomous Systems Lab, ETH Zürich.
Corresponding Author: M. Pantic, mpantic@ethz.ch
This work was supported in part by the Swiss National Science Foundation through the National Centres of Competence in Research Robotics (NCCR Robotics) and Digital Fabrication (NCCR dfab), and in part by Amazon Research Award #4808.

[1]https://github.com/ethz-asl/reactive_avoidance

ronments. Early approaches for reactive obstacle avoidance on MAVs combined simple ultra-sound or infrared distance sensing with attitude control [1] or handcrafted algorithms [9] to steer the MAV away from obstacles. While these early approaches are robust, reactive, and power efficient, they do not scale well regarding the number of measurements. Later, online stereo depth estimation algorithms made reactive collision avoidance at higher resolution possible [2]. Existing reactive, sensor-based approaches often operate at the controller level. Data-driven obstacle avoidance methods have recently shown great robustness and versatility, especially for fast and dynamic flight [4]. However, all previously presented methods are monolithic designs requiring complete re-design or re-training for additional use cases or new requirements.

The second significant part of research on collision avoidance for MAVs revolves around online mapping and planning using a volumetric map representation. Octomap [10] and voxblox [11] are standard volumetric occupancy mapping systems used for planning. Voxblox additionally provides SDF information and obstacle gradients, which can improve planning speeds and quality at the expense of additional processing during mapping. Sampling-based algorithms such as RRT* [12], derived methods, and optimization-based methods such as CHOMP [13] will find efficient and safe paths in voxel-based maps. Combined with a depth sensor and an odometry/slam framework, a closed-loop obstacle avoidance system with global and local planning is possible [3], [14]. The resulting closed-loop planning rates are only a few hertz. The robustness of the solution hinges on the map quality, which is limited by state estimation drift, voxel resolution, and compute resources for sampling, collision checking, and optimization.

Riemannian Motion Policies [6] is a modular and fast navigation and motion synthesis framework. Second-order differential equations encode behaviors on task-specific manifolds that are combined efficiently to generate the overall robot motion. The original work [6] uses collision meshes with applications for robotic manipulators. RMPs have use cases such as navigating cars based on sensor and learned policies [7], steering legged robots via extracting geodesic fields from height maps [8] and navigating MAVs with meshes [15] or NeRFs [16] as world representation. These works focused on using RMPs on map representations or modifications thereof such that path planning can be performed without multiple lookups or sampling. However, to the best of our knowledge, no method currently exists that allows the usage of voxel-based map representations directly with RMPs. As voxel-based maps do not contain higher-level geometrical information such as connectivity, often sampling or optimization methods are needed for planning, impeding fast reactive navigation.

III. Background

Before describing our method, we first provide a brief introduction to RMPs. For more details, please refer to the original work [6]. One of the main ideas behind Riemannian Motion Policies is to formulate a potentially complex behavior as a combination of many small and straightforward behaviors. The designer can formulate these policies in whichever space they are easiest to describe. The essential building block is a policy $\mathcal{P}$ consisting of a function $f(\mathbf{x}, \dot{\mathbf{x}})$ and accompanying Riemannian metric $\mathbf{A}(\mathbf{x}, \dot{\mathbf{x}})$, where $\mathbf{x} \in \mathbb{R}^3$ is the robot's current position in the policy's coordinate frame. The function $\mathbf{f}$ defines the instantaneous acceleration, while the Riemannian metric $\mathbf{A}$ corresponds to the weight of the policy, which can be isotropic (identical in all axes) or directional. A single policy $\mathcal{P}_C$ combines multiple policies $\{\mathcal{P}_0, \ldots, \mathcal{P}_N\}$,

$$\mathcal{P}_c = (\mathbf{f}_c, \mathbf{A}_c) = \left(\left(\sum_i \mathbf{A}_i \right)^+ \sum_i \mathbf{A}_i \mathbf{f}_i, \sum_i \mathbf{A}_i \right). \quad (1)$$

Equation (1) optimizes the geometric-weighted mean of all summed individual policies. Note that there is no explicit optimizer but rather an implicit gradient-descent-like optimization by the physical system evaluating and executing the policy at every timestep. The policy combination enables fast and efficient navigation algorithms. However, as there is no time correlation and the system is purely reactive, the policies must provide stability and smoothness through appropriate design.

A. Base policies

We use the attractor and obstacle avoidance policies defined in [6] as basic building blocks. An attractor policy creates an attractive force and can be used to define goal locations. Let $\mathbf{x}_a$ denote the target towards which the robot should move. The attractor policy $\mathcal{P}_a = (\mathbf{f}_a, \mathbf{A}_a)$ defines as

$$\begin{aligned} \mathbf{f}_a(\mathbf{x}, \dot{\mathbf{x}}) &= \alpha_a \boldsymbol{s}\left(\mathbf{x}_a - \mathbf{x}\right) - \beta_a \dot{\mathbf{x}} \\ \mathbf{A}_a(\mathbf{x}, \dot{\mathbf{x}}) &= \mathbb{I}^{3\times 3} \end{aligned}, \quad (2)$$

where $\alpha_a, \beta_a > 0$ are gain respectively damping scalar parameters. We define the soft normalization function $\boldsymbol{s}$ as:

$$\boldsymbol{s}\left(\boldsymbol{v}\right) = \frac{\boldsymbol{v}}{\|\boldsymbol{v}\| + c \log\left(1 + \exp\left(-2c\boldsymbol{v}\right)\right)}, \; c > 0\,, \quad (3)$$

where c is a tuning parameter. The second building block, the obstacle avoidance policy $\mathcal{P}_{obs} = (\mathbf{f}_{obs}, \mathbf{A}_{obs})$, is the summation of a repulsive term $\mathbf{f}_{rep}$ and a damping term $\mathbf{f}_{damp}$:

$$\mathbf{f}_{obs}\left(\mathbf{x},\, \dot{\mathbf{x}},\, \mathbf{r},\, d\right) = \mathbf{f}_{rep}\left(\mathbf{x},\, \mathbf{r},\, d\right) + \mathbf{f}_{damp}\left(\dot{\mathbf{x}},\, \mathbf{r},\, d\right)\,, \quad (4)$$

where $\mathbf{r}$ is the unit vector pointing away from the obstacle, and d is the distance to the obstacle, both functions of the robot's position. The repulsive term is:

$$\mathbf{f}_{rep}\left(\mathbf{x},\, \mathbf{r},\, d\right) = \eta_{rep} \exp\left(-\frac{d}{\upsilon_{rep}}\right) \mathbf{r}\,, \quad (5)$$

where $\eta_{rep}, \upsilon_{rep} > 0$ are a gain and a length scaling factor, respectively. Next, we define the vector $\mathbf{g}_{obs}$ that points away from the obstacle, scaled according to the velocity projected onto $-\mathbf{d}_{\mathbf{q}}$, and vanishing to 0 if the velocity vector points away from the obstacle.

$$\mathbf{g}_{obs}\left(\dot{\mathbf{x}},\, \mathbf{r}\right) = \max\left\{0, -\dot{\mathbf{x}}^T \mathbf{r}\right\}^2 \mathbf{r}\,, \quad (6)$$

Using (6) the damping term is:

$$\mathbf{f}_{damp}\left(\dot{\mathbf{x}},\,\mathbf{r},\,d\right)=\eta_{damp}\Big/\left(\frac{d}{\upsilon_{damp}}+\epsilon\right)\cdot\mathbf{g}_{obs}\left(\dot{\mathbf{x}},\,\mathbf{r}\right),\quad(7)$$

where η_{damp} and υ_{damp} are gain and length scaling factors, and $0<\epsilon\ll 1$ ensures numerical stability. The resulting metric $\mathbf{A}_{obs}$ is:

$$\mathbf{A}_{obs}\left(\mathbf{x},\,\dot{\mathbf{x}},\,\mathbf{r},\,d\right)=\\ w_r\left(d\right)\cdot\boldsymbol{s}\left(\mathbf{f}_{damp}\left(\dot{\mathbf{x}},\,\mathbf{r},\,d\right)\right)\boldsymbol{s}\left(\mathbf{f}_{damp}\left(\dot{\mathbf{x}},\,\mathbf{r},\,d\right)\right)^{T},\quad(8)$$

where the soft-normalization function $\boldsymbol{s}\left(\cdot\right)$ is the same as (3), and $w_r\left(\cdot\right)$ is a weight function that defines the radius r in which the policy is active. The weight is a cubic spline $w_r\left(d\right)=\frac{1}{r^2}d^2-\frac{2}{r}d+1$ in the interval from $w_r\left(0\right)=1$ to $w_r\left(r\right)=0$, with derivatives equal to 0 at the endpoints. For distances greater than r, the weighing function returns 0. This metric is directionally stretched towards the obstacle, indicating that the policy acts stronger in this direction.

IV. Method

The challenge in using voxel-based map representations with RMPs is that, unlike with meshes or geometric primitives, one can not derive a single efficient policy that induces a collision-avoiding behavior. We overcome this by leveraging modern GPUs and developing a massively parallel policy evaluation system. We combine raycasting in a voxel-based map with per-ray obstacle avoidance policies directly on the GPU, enabling the efficient evaluation of up to 200 million individual ray policies per second. Compared to approaches using single location or SDF lookups, superposing numerous policies enables the navigation algorithm to factor in more varied information about the local geometry. The following shows how such massive parallelism can be obtained and used for navigation.

A. Problem description and notation

Our goal is to define an acceleration policy

$$\mathcal{P}^k=(\mathbf{f}^k(\mathbf{x}^k,\dot{\mathbf{x}}^k),\mathbf{A}^k(\mathbf{x}^k,\dot{\mathbf{x}}^k))\quad(9)$$

that steers a robot from a start to a goal state while staying clear of obstacles. At every time step k, our method evaluates the policy as a function of the robot's state consisting of its position $\mathbf{x}^k$ and velocity $\dot{\mathbf{x}}^k$. It then applies the resulting acceleration $\ddot{\mathbf{x}}^k=\mathbf{f}(\mathbf{x}^k,\dot{\mathbf{x}}^k)$ to the robot. Without loss of generality, we assume a 3D world and 3D policies in the following, i.e., $\mathbf{x},\dot{\mathbf{x}},\mathbf{f}\in\mathbb{R}^3$ and $\mathbf{A}\in\mathbb{R}^{3\times3}$. Note that both $\mathbf{f}^k$ and $\mathbf{A}^k$ can absorb arbitrary information about the current map, sensor data, or environment – which we here express by the explicit time dependency of both functions but omit in the following for brevity.

B. Raycasting based obstacle avoidance

In order to efficiently use as much spatial information as possible at every time step, we use many parallel policies combined with efficient map queries through raycasting. The core idea is to generate a set of rays $\{\mathbf{r}_0,\ldots,\mathbf{r}_N\}$, along which we perform raycasting to determine the distance to an obstacle. To generate a sampling of $N+1$ quasi-random, equally spaced rays radiating from the robot's center, we use a 2D Halton sequence $\mathcal{H}(\cdot)$ [17] with base 2 for sampling the azimuth φ_i, and base 3 for elevation θ_i:

$$\begin{aligned}\varphi_i&=\cos^{-1}(1-2\,\mathcal{H}(i,2))\\ \theta_i&=2\,\pi\,\mathcal{H}(i,3)\end{aligned}\qquad\forall i=0,\ldots,N\quad(10)$$

The resulting ray is then transformed into a unit direction vector $\mathbf{r}_i=(\sin(\varphi_i)\cos(\theta_i),\sin(\varphi_i)\sin(\theta_i),\cos(\varphi_i))$. Figure 2 shows a visualization of the resulting rays. The Halton sequence provides a deterministic and parallelizable sampling of orientations without the need for information exchange between threads. For each ray $\mathbf{r}_i$, we calculate the distance to the nearest obstacle from the current robot position $\mathbf{x}$ along the ray using the raycasting function $d_i=\mathcal{R}(\mathbf{x},\mathbf{r}_i)\in\mathbb{R}^+$. Combining raycasting with Equation (4) and Equation (1), we obtain the obstacle avoidance policy $\mathcal{P}_{ray}$ as

$$\begin{aligned}\mathbf{A}_{ray}(\mathbf{x},\dot{\mathbf{x}})&=\sum_{i=0}^{N}\mathbf{A}_{obs}(\mathbf{x},\dot{\mathbf{x}},\mathbf{r}_i,d_i)=\sum_{i=0}^{N}A^i_{obs}\\ \mathbf{f}_{ray}(\mathbf{x},\dot{\mathbf{x}})&=\mathbf{A}^+_{ray}\sum_{i=0}^{N}\mathbf{A}^i_{obs}\Big(\mathbf{f}_{rep}(\mathbf{x},\mathbf{r}_i,d_i)+\\ &\qquad\mathbf{f}_{damp}(\dot{\mathbf{x}},\mathbf{r}_i,d_i)\Big).\end{aligned}\quad(11)$$

The complete raycasting policy $\mathcal{P}_{ray}$ provides obstacle avoidance that is direction, distance, and speed dependent. It has a zero metric for all rays pointing sufficiently away from the robot's velocity, effectively disabling the repulsive component for these rays. All close rays in the direction of the robot's velocity, i.e., posing a risk for collision, generate a large metric and acceleration and therefore repel the robot.

The summation of the raycasting policy $\mathcal{P}_{ray}$ with a suitable goal attractor policy $\mathcal{P}_a$ produces a local planning policy that drives the robot to the desired goal and keeps it clear of obstacles. Policy execution and reiteration implicitly optimize the joint objective of both policies.

While this approach is conceptually simple, it requires the evaluation of thousands of raycasting policies, which is computationally expensive. However, as described in the next section, the inherently parallel nature of RMPs combined with a modern GPU implementation of a voxel-based map [18] allow offloading of the planning process to the GPU.

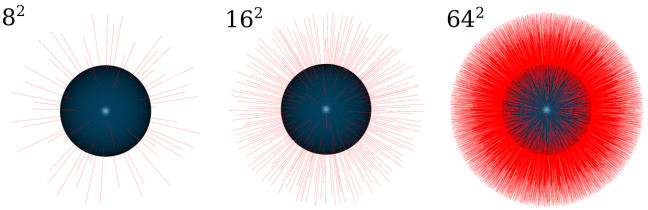

Fig. 2. Visualization of the Halton sequence sampling mapped to spherical coordinates for different numbers of samples $(8^2, 16^2, 64^2)$.

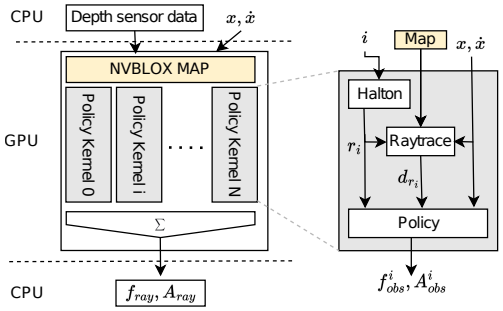

Fig. 3. Left: Overview of the overall execution architecture. The map is kept on the GPU, for each ray an individual policy kernel is started. The summation Σ (Equation (1)) is implemented using an efficient block-reduce paradigm. Right: Data flow diagram of an individual policy kernel. As the policy kernels have no interdependency they parallelize efficiently.

C. Implementation

An efficient raycasting policy implementation is crucial to reach the required refresh rates for fast and reactive navigation. We implement a single CUDA kernel that combines ray computation based on thread id and the Halton sequence, raycasting, and policy calculation. The raycasting is adopted from nvblox [18] and uses the SDF and voxel information to step through free space efficiently. Each ray policy executes as an instance of this kernel on a CUDA thread. The map of the environment, implemented using nvblox [18], is kept in GPU memory and shared between all kernels, meaning that the kernels do not require additional GPU transfers during execution.

The ray and policy operations do not have any cross-kernel interdependencies except the access to global shared GPU memory, leading to almost linear scaling w.r.t. the number of cores available. A block-reduce scheme combines all the individual policies into a single policy as per Equation (1). Finally, we only transfer the resulting policy and its metric, 12 float values, respectively 48 bytes, back to the host computer for execution.

D. LiDAR scan policies

The raycasting paradigm for obstacle avoidance is a natural fit for raw LiDAR data, where each LiDAR beam represents the result of a physical raycasting operation. Instead of using the Halton sequence $\mathcal{H}$ and raycasting function $\mathcal{R}$ inside a map, we directly use the LiDAR beam direction and sensed range for each LiDAR beam in a laser scan. Due to the highly efficient parallelization, we can execute a sufficient amount of policies to address every LiDAR beam in every scan produced by the LiDAR. For example, evaluating an Ouster OS-0 128 beam LiDAR scan with 2048 rotational increments corresponds to evaluating 262 144 individual obstacle avoidance policies every 0.1 sec, which processes all available information about the local geometry.

E. Simple obstacle avoidance in ESDF maps

Voxel mapping frameworks such as voxblox [11] that carry a *Euclidian signed distance field* (ESDF) representation support closest obstacle distance and gradient queries. We define the function that returns the distance to the closest obstacle given a position as $\mathcal{D}(\mathbf{x}) \in \mathbb{R}^+$, with the corresponding obstacle gradient $\nabla\mathcal{D}(\mathbf{x}) \in \mathbb{R}^3$. A simple single-query obstacle avoidance policy would use $\mathcal{D}(\mathbf{x})$ as d and $\nabla\mathcal{D}(\mathbf{x})$ as $\mathbf{r}$ for the obstacle avoidance policy defined in Section III-A. Together with a goal attractor policy, this constitutes a simple local planner. Such a simple approach, however, in practice suffers from poor stability. We use it as a comparison policy to illustrate how beneficial the use of broader geometric information for planning is, compared to only knowing the local obstacle query.

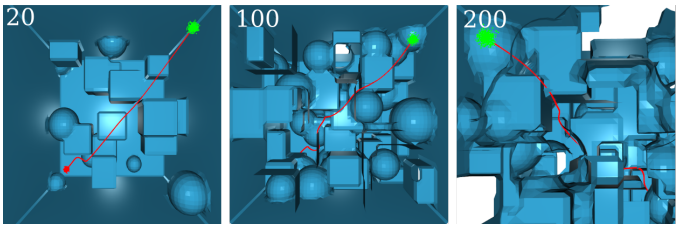

Fig. 4. Rendering of exemplary random maps with different number of obstacles within the same $10 \times 10 \times 10\,[m]$ perimeter. The green and red points indicate desired start and end-points for trajectories. The red trajectories are generated using the RMP raycasting policy with 64^2 rays. Note that the 200 map appears less dense than it actually is, as not all obstacles are shown for visualization (backface culling).

V. Experimental Setup

We evaluate our proposed system quantitatively in simulation to demonstrate the ability to navigate successfully in complex 3D environments. We vary the number of local policies used and compare the results to an ESDF-based policy described in Section III-A, similar to [7] and to CHOMP [13]. Next, we provide run-time information on different hardware configurations. Finally, we present qualitative results of the method running on a real platform. The parameters used throughout the experiments are listed in Table I. Important is the ratio between r and υ_{damp}, respectively υ_{rep}, as they define the length scales at which the policies and repulsors activate. The gains α_g, η_{rep}, and η_{damp} determine the strength of the individual terms. The tuning is relatively insensitive, as the attractor and obstacle policies are tuned independently. We advise setting the length scales first and then adjusting the gains accordingly.

A. Map-based evaluations

To provide a balanced and statistically meaningful evaluation of the raycasting policy described in Equation (11), we use a large and diverse amount of randomly generated maps. To generate these maps, we uniformly sample geometric primitives such as spheres and boxes and merge them through boolean operations. Figure 4 shows three instances of dense, cluttered maps with increasing primitive numbers. The resulting geometry has high variability, as intersections and fusions generate small and intricate but also large features. We use a maximum of 200 obstacles corresponding to about 45% occupied voxels on average. The number of rays sampled is a major design parameter of the raycasting policy, greatly influencing the execution

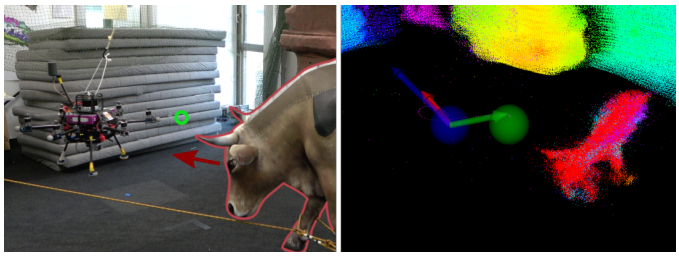

Fig. 5. Left: Moving obstacle experiment. The MAV is commanded to hold the position indicated by the green circle, while an obstacle is pulled through the environment. Right: Visualization of the policy in the same experiment, accumulated over 10 frames for visualization. The color signifies the influence of each point, where red is strongest, purple weakest. The policy's directional metric then minimizes the influence of points facing away from the MAVs motion. The green sphere is the commanded and the blue sphere the current location of the MAV. The green arrow is the instantaneous acceleration of the goal policy, the blue arrow the combined effect of the ray policy and the red arrow the final combination executed by the MAV, after taking into account the directional metrics.

speed and geometry perception. How many should one use generally, and where do diminishing returns begin? To gain insight into this question, we vary this parameter between $4^2 = 16$ and $256^2 = 65\,536$. Another aspect is the comparison against the naive ESDF-based policy described in Section IV-E, which only needs a single geometrical lookup per time step and mimics the information typically available to an optimizer. Another question we aim to answer is how well a purely reactive method can perform against an optimization-based method in terms of planning quality, time, and success rate for obstacle avoidance. Therefore, we compare against different tunings of CHOMP [13].

B. LiDAR policy

Due to its reactive and sensor-based nature, we present the closed-loop LiDAR variant of the proposed navigation method on an actual MAV. The MAV carries an NVidia Jetson Xavier NX and an Ouster OS-1 64 beam LiDAR outputting point clouds of size 512×64 at 20 Hz. All processing is performed onboard and is purely sensor-based. The navigation system consists of just the LiDAR and goal policy executed at a control frequency of 100 Hz. The resulting acceleration is the next setpoint. The reactive nature of this setup allows for avoiding moving obstacles. We perform two experiments with static obstacles in different environments; a large garage and an office-sized room. We command the MAV to reach a position behind the obstacles via the goal attractor policy. Additionally, we run an experiment with a large moving obstacle where we commanded the MAV to stay at a specific location in the path of the moving obstacle.

VI. Results

In the following, we present the results of the map-based statistical experiments and the sensor-based MAV experiments.

Param	Description	Static Map	LiDAR
α_g	Attractor gain	10	0.8
β_g	Attractor damping	15	1.6
c	Softmax parameter	0.2	1.0
η_{rep}	Obstacle repulsive gain	88	1.2
υ_{rep}	Obstacle repulsive scaling	1.4	1.5
η_{damp}	Obstacle damping gain	140	3.0
υ_{damp}	Obstacle damping scaling	1.2	1.0
r	Policy radius	2.4	1.3

TABLE I

Tuning of the obstacle avoidance policy for static map and LiDAR evaluations.

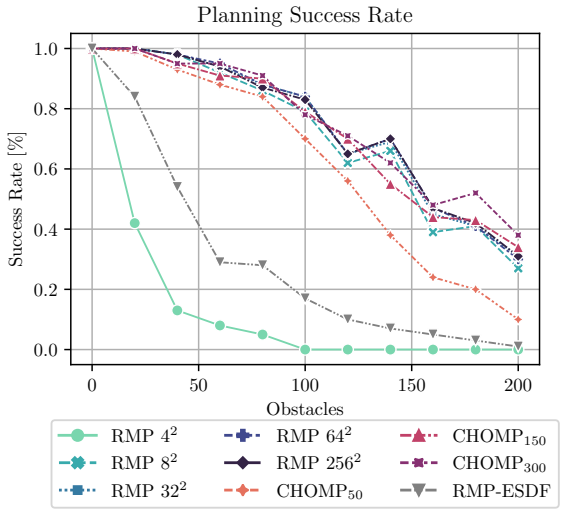

Fig. 6. Comparison of success rates of different local planners. RMP N^2 denotes the proposed planner with N^2 rays generated by Halton sampling, raytraced and evaluated in the policy. For CHOMP variants the number indicates the maximum number of iterations allowed.

A. Map-based raycasting policy

Figure 6 shows the success rates of the different planners in increasingly cluttered maps. We calculate the success rate by running 100 randomized repetitions for each planner-map combination, resulting in $\sim 9k$ total runs. We label a planning instance successful if it generates a collision-free path that connects the start and goal coordinates. The proposed raycasting approach performs similarly to a well-tuned CHOMP implementation, while the naive ESDF policy often gets stuck in local minima or collides with geometry. The policy can not react to more distant geometry as it only perceives the local obstacle gradient and distance. The number of rays cast also influences the success rate. A sampling of $4^2 = 16$ rays is too sparse to obtain good coverage in all directions. The success rate approaches the maximum observed at around $32^2 = 1024$ rays. However, collision avoidance in filigree environments or robots with intricate geometry will benefit from even denser raycasting. Figure 4 shows successful paths obtained with the raycasting policy. While a single raycasting policy is comparably simple, complex behavior emerges from the combination of a large number of simple policies. Even in dense and complicated maps, the generated trajectories are smooth and

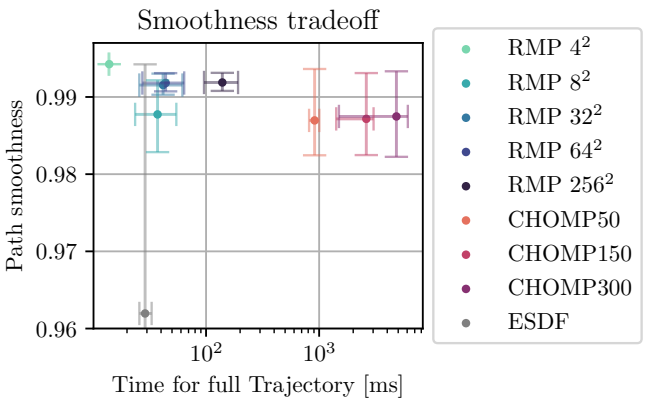

Fig. 7. Visualization of smoothness and trajectory planning time trade-off, from 5159 successful runs. The top left corner is the optimum. Error-bars visualize the 10^{th} and 90^{th} percentile. Path smoothness is the averaged angular similarity between subsequent trajectory segments $(1 - \frac{1}{\pi}\ cos^{-1}\left[A \cdot B/(|A||B|)\right])$, where 1.0 signifies no angular change and values below 0.95 become visibly jagged.

free of collisions. Figure 7 visualizes the trade-off between path smoothness and the time needed to plan a complete trajectory. We observe that the proposed raycasting approach results in smooth trajectories requiring significantly less time than CHOMP. CHOMP uses an explicit smoothness term in its optimization, while RMPs are naturally smoothed through double integration of accelerations. Here, the acceleration is the average of a function of many rays, which implicitly creates a smooth progression of accelerations. The evaluation only considers successful trajectories, which, for methods with low success rates, favors simple environments solvable with straight trajectories. Thus, the high smoothness reported for the 4^2 ray variant is of low practical significance. The ESDF policy's resulting acceleration depends only on two properties read from the map at each time step, causing higher variations in acceleration and less smooth paths.

B. Reactive sensor-based navigation

We demonstrate the qualitative feasibility of the reactive LiDAR collision avoidance on an MAV. The accompanying video shows the experiments and gives additional details. The system performs obstacle avoidance onboard at sensor frequency. The policy steers the MAV smoothly despite imperfections in the raw input point clouds, such as outliers, range noise, and occlusions. We do not perform any point cloud pre-processing or filtering except excluding points on the vehicle with a small range and intensity. Figure 1 shows an example of the trajectory executed by the MAV in the large garage scenario. The system could avoid structured (pillar) and unstructured obstacles (plant). Moving obstacle avoidance is shown in Figure 5. The planner is robust against partial occlusion and partially incomplete point clouds due to absorption. The navigation stack used about 2% of the GPU resources (at maximal 1.1 GHz GPU frequency) and 20% of one ARM CPU core of the Jetson NX. The CPU load is mainly attributed to point cloud I/O.

C. Benchmarks

We benchmarked all policies on three different GPU variants: An NVidia Jetson NX with 384 CUDA cores, an NVidia MX150 with 384 cores, and a Titan XP with 3840 cores. The Jetson represents what a typical an MAV can carry. Figure 8 shows the benchmarks for both policies.

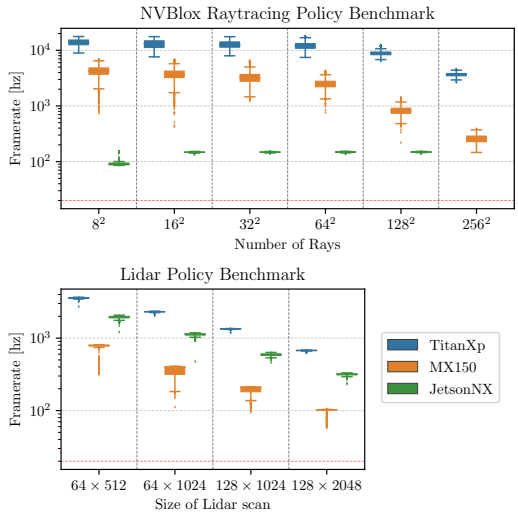

Fig. 8. Benchmark statistics for policy evaluation. The framerate indicates the frequency at which an obstacle policy with the given amount of rays can be loaded, executed, block-reduced and the resulting (f, A) read back for execution. The red line indicates the maximum frequency ($20\,hz$) at which commonly available LiDARs (e.g. Ouster OS-1) can supply laser scans.

The proposed planner achieves frame rates vastly above the maximum sensor frequency on all tested architectures. The time needed for policy integration is nearly constant and, therefore, well predictable, mitigating the risk of bottlenecks during execution. The map-based approach has static overhead caused by the map, which is most evident on the Jetson.

VII. Conclusion

This paper introduced a novel and practical method that enables using RMPs for obstacle avoidance on voxelized maps and raw LiDAR observations. Massive parallel geometrical queries significantly improve planning performance. The resulting reactive planner performs as well as an optimization-based approach at a fraction of the computational cost. Computation is accelerated by leveraging the parallelism of modern GPUs, which allows simultaneous processing of a staggering number of raycasting policies in a GPU-based voxel-grid map. Synthetic and real-world experiments demonstrate the qualities and performance of our approach. The method performs well in success rate, smoothness, and compute utilization. Local minima can occasionally cause the planner to get stuck - which is expected for a *local* collision avoidance system. Thanks to the modular architecture, adding additional policies for *global* planning is straightforward. Furthermore, we open source the implementation of the proposed method to enable future research on RMP-based navigation in challenging and cluttered environments.

REFERENCES

[1] S. Bouabdallah and R. Siegwart, "Full control of a quadrotor," in *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 153–158, 2007.

[2] H. Oleynikova, D. Honegger, and M. Pollefeys, "Reactive avoidance using embedded stereo vision for mav flight," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 50–56, IEEE, 2015.

[3] H. Oleynikova, M. Burri, Z. Taylor, J. Nieto, R. Siegwart, and E. Galceran, "Continuous-time trajectory optimization for online uav replanning," in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 5332–5339, IEEE, 2016.

[4] A. Loquercio, E. Kaufmann, R. Ranftl, M. Müller, V. Koltun, and D. Scaramuzza, "Learning high-speed flight in the wild," *Science Robotics*, vol. 6, no. 59, p. eabg5810, 2021.

[5] O. Khatib, "Real-time obstacle avoidance for manipulators and mobile robots," in *Autonomous robot vehicles*, pp. 396–404, Springer, 1986.

[6] N. D. Ratliff, J. Issac, D. Kappler, S. Birchfield, and D. Fox, "Riemannian motion policies," *arXiv preprint arXiv:1801.02854*, 2018.

[7] X. Meng, N. Ratliff, Y. Xiang, and D. Fox, "Neural autonomous navigation with riemannian motion policy," in *2019 International Conference on Robotics and Automation (ICRA)*, pp. 8860–8866, IEEE, 2019.

[8] M. Mattamala, N. Chebrolu, and M. Fallon, "An efficient locally reactive controller for safe navigation in visual teach and repeat missions," *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 2353–2360, 2022.

[9] N. Gageik, P. Benz, and S. Montenegro, "Obstacle detection and collision avoidance for a uav with complementary low-cost sensors," *IEEE Access*, vol. 3, pp. 599–609, 2015.

[10] A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard, "Octomap: An efficient probabilistic 3d mapping framework based on octrees," *Autonomous robots*, vol. 34, no. 3, pp. 189–206, 2013.

[11] H. Oleynikova, Z. Taylor, M. Fehr, R. Siegwart, and J. Nieto, "Voxblox: Incremental 3d euclidean signed distance fields for on-board mav planning," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 1366–1373, IEEE, 2017.

[12] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *The international journal of robotics research*, vol. 30, no. 7, pp. 846–894, 2011.

[13] M. Zucker, N. Ratliff, A. D. Dragan, M. Pivtoraiko, M. Klingensmith, C. M. Dellin, J. A. Bagnell, and S. S. Srinivasa, "Chomp: Covariant hamiltonian optimization for motion planning," *The International Journal of Robotics Research*, vol. 32, no. 9-10, pp. 1164–1193, 2013.

[14] K. Mohta, M. Watterson, Y. Mulgaonkar, S. Liu, C. Qu, A. Makineni, K. Saulnier, K. Sun, A. Zhu, J. Delmerico, K. Karydis, N. Atanasov, G. Loianno, D. Scaramuzza, K. Daniilidis, C. J. Taylor, and V. Kumar, "Fast, autonomous flight in gps-denied and cluttered environments," *Journal of Field Robotics*, vol. 35, no. 1, pp. 101–120, 2018.

[15] M. Pantic, L. Ott, C. Cadena, R. Siegwart, and J. Nieto, "Mesh manifold based riemannian motion planning for omnidirectional micro aerial vehicles," *IEEE Robotics and Automation Letters*, vol. 6, no. 3, pp. 4790–4797, 2021.

[16] M. Pantic, C. Cadena, R. Siegwart, and L. Ott, "Sampling-free obstacle gradients and reactive planning in neural radiance fields (nerf)," *ICRA 2021 Neural Implicit Representations Workshop / arXiv preprint arXiv:2205.01389*, 2022.

[17] T.-T. Wong, W.-S. Luk, and P.-A. Heng, "Sampling with hammersley and halton points," *Journal of graphics tools*, vol. 2, no. 2, pp. 9–24, 1997.

[18] Nvidia, "nvxblox." https://github.com/nvidia-isaac/nvblox.