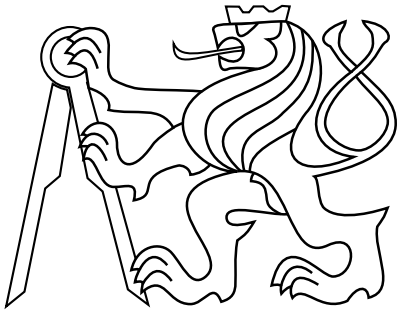
Czech Technical University in Prague

Faculty of Electrical Engineering
Department of Cybernetics
Multi-robot Systems



Model Predictive Path Integral for Agile UAV Control with Online Collision Avoidance

Bachelor's Thesis

Pavel Pochobradský

Prague, May 2025

Study programme: Cybernetics and Robotics

Supervisor: Ing. Robert Pěnička, Ph.D.

Acknowledgments

First, I would like to thank my supervisor Ing. Robert Pěnička, Ph.D. for his invaluable support. In addition, I am thankful to my friends and family for supporting me throughout my studies.

Abstract

Agile flight in unknown environments is an important step towards autonomous Unmanned Aerial Vehicle (UAV) flights. In this thesis, we introduce the Geometric Model Predictive Path Integral (GMPPI) controller, which is capable of tracking agile trajectories and avoiding obstacles online by directly using depth sensor data. It is a sampling-based predictive control method that considers nonlinear system dynamics and uses a proposed UAV-specific cost function. The algorithm structure is based on Model Predictive Path Integral (MPPI). We newly integrate rollout trajectories created with an SE(3) controller into the GMPPI method, set dynamic time steps in rollout simulation, and deterministically control yaw rotation velocity. Together, these features improve the trajectory tracking capability over existing MPPI implementations and match the obstacle avoidance capability of a state-of-the-art real-time obstacle-aware planner. Finally, we develop a method to interface the proposed controller with a depth camera sensor by projecting candidate rollout trajectories onto the image plane of the latest available depth image. A single configuration of the proposed method can track a variety of simulated reference trajectories with precision that exceeds the geometric SE(3) controller and avoid obstacles online at speeds of up to 13 $m\,s^{1}$, exceeding the performance of learning-based methods without requiring any training data.

Keywords Obstacle Avoidance, Agile Flight, Unmanned Aerial Vehicle, Automatic Control, Model Predictive Path Integral

Abstrakt

Agiln let v neznamem prostred je dulezitym krokem k autonomnmu rzen bezpilotnch vzdusnych prostredku. Tato prace predstavuje metodu rzen Geometric Model Predictive Path Integral (GMPPI) schopnou sledovat agiln trajektorie a vyhybat se prekazkam v realnem case na zaklade vstupu z hloubkove kamery. Jedna se o metodu prediktivnho rzen zalozenou na metode Model Predictive Path Integral (MPPI), zohlednujc nelinearn dynamiku rzeneho systemu a vyuzvajc zde vyvinutou ohodnocovac funkci. V teto praci do metody GMPPI integrujeme trajektorie generovane s vyuzitm SE(3) regulatoru, pridavame moznost dynamicky nastavit delku casoveho kroku v simulaci navrzenych trajektori, a pridavame schopnost deterministicky rdit uhlovou rychlost kolem osy z souradne soustavy bezpilotnho prostredku. Tyto funkce zlepsuj presnost pri sledovan referencnch trajektori oproti existujc implementaci MPPI a umoznuj vyhybat se prekazkam s uspesnost srovnatelnou s dedikovanym planovacem trajektori. Vysledna metoda rzen je propojena hloubkovou kamerou promtnutm navrzenych trajektori do aktualnho hloubkoveho snmku. Navrhovana metoda umoznuje bez zmen v konguraci sledovat uzivatelem simulovane referencn trajektorie s presnost presahujc samostatny SE(3) regulator a vyhybat se prekazkam pri rychlostech az 13 m s[1] s uspesnost vets, nez metody zalozene na strojovem ucen, a to bez nutnosti trenovan.

Klcova slova Vyhyban se prekazkam, Agiln let, Bezpilotn vzdusne prostredky, Automaticke rzen, Model Predictive Path Integral

Abbreviations

CTU Czech Technical University

GPU Graphics Processing Unit

CPU Central Processing Unit

CNN Convolutional Neural Network

PI Path Integral

GMPPI Geometric Model Predictive Path Integral

MPPI Model Predictive Path Integral

MPC Model Predictive Control

PID Proportional Integral Derivative

LQR Linear Quadratic Regulator

MRS Multi-robot Systems Group

UAV Unmanned Aerial Vehicle

UAVs Unmanned Aerial Vehicles

APF Arti cial Potential Field

APF Arti cial Potential Field

VFH Vector Field Histogram

RL Reinforcement Learning

IL Imitation Learning

CCS Constrained Covariance Steering

CVaR Conditional Value-at-Risk

Contents

1 Introduction

Autonomous Unmanned Aerial Vehicles (UAVs) are increasingly being deployed in missions requiring navigation in unknown cluttered environments. In various scenarios, such as search and rescue [1], power line inspection [2], bridge inspection [3], aerial deliveries [0] and even digitization of historical monuments [4], UAVs might need to perform complex maneuvers in unknown environments while avoiding obstacles. Meanwhile, speed is an important criterion across use cases, allowing for better e ciency in the case of infrastructure inspection and even potentially helping save lives in the case of search and rescue operations. Control and local trajectory planning for UAVs in such conditions is a challenging open problem, further exacerbated by the limited sensory and computational capabilities a small Unmanned Aerial Vehicle (UAV) can carry on board.

Any solution to the challenge of real-time navigation in cluttered environments must consider a variety of factors, including modeling obstacles and considering dynamics constraints of the UAV, all while computing nal control commands at a high enough frequency for online operation. The classical solution to this is a modular approach, where an environment map is constructed from sensor input, a global planner nds an obstacle-free trajectory, followed by a controller executing the trajectory. Although this conceptual split of tasks is clear and transparent, latency can accumulate throughout the system, preventing ight at higher speeds [5]. Segmentation of responsibility also prevents the planner from considering dynamic constraints of the UAV and the controller from accounting for obstacles, further reducing the agile ight capability. A more recently used approach aims to bypass the modular design by learning end-to-end mappings from sensor input to control commands [5], [6]. Although such methods avoid the segmentation challenges mentioned above, they require large amounts of training data and cannot be easily reused across di erent UAV models [7].

This thesis proposes a UAV controller called Geometric Model Predictive Path Integral (GMPPI). Based on the Model Predictive Path Integral (MPPI) method [8], GMPPI is a sampling-based controller capable of avoiding obstacles based on stereo camera images, as well as agile control of a UAV. The proposed approach retains the conceptual simplicity of learning-based methods and their latency advantages, without requiring training. It also considers UAV dynamics and obstacle avoidance requirements at the same time. To interface the proposed method with a depth camera, we project each of the GMPPI candidate rollout trajectories onto the depth image and assess if a collision occurs based on this projection. To utilize the full range of the depth sensor without adding computational complexity, the time step length of the rollouts is varied throughout each rollout. Incorporation of candidate trajectories generated with the geometric SE(3) control method [9] allows the generation of a small number of precise trajectories and a UAV-tailored cost function enables smooth and aggressive ight depending on the shape of the reference trajectory.

We verify the trajectory tracking performance of the proposed controller against geometric SE(3) control [9] and existing MPPI controller implementation [8]. An ablation study is also included to highlight the contribution of each of the key controller features to overall performance. Obstacle avoidance capability is veri ed against learning-based controllers and a state-of-the-art obstacle-aware motion planner. In simulated runs, GMPPI performs on par with the state-of-the-art motion planner in obstacle avoidance and exceeds the performance of all other controllers tested in trajectory tracking.

2 Related work

This thesis mainly contributes by applying MPPI control to the challenge of sensor-based real-time obstacle avoidance while tracking reference trajectories. To contextualize this contribution, the following section reviews relevant reference tracking methods, prior approaches to obstacle avoidance, and key developments in the application of the MPPI method.

2.1 Trajectory Tracking

Various methods can be applied to quadrotor UAV trajectory tracking, from linear Proportional Integral Derivative (PID) or Linear Quadratic Regulator (LQR) control to nonlinear Model Predictive Control (MPC) and other nonlinear methods [10]. Among these, the Geometric SE(3) control method stands out due to its almost global exponential stability, which enables aggressive ight at the limit of UAV dynamics, and relative algorithmic simplicity, which allows fast computation [9]. The ability to track aggressive trajectories makes it a good benchmark to compare the proposed controller against, while its low computational overhead allows integration of parts of the method presented in [9] into the proposed controller.

2.2 Obstacle Avoidance

Traditionally, obstacle avoidance on UAV systems has been handled as part of a navigation pipeline consisting of localization, mapping, trajectory planning and control, where trajectory planning includes global path search and local trajectory optimization. The latter is relevant to this work.

Traditional Methods

One of the rst use cases for real-time replanning and optimization of collision-free trajectories was manipulator obstacle avoidance [11], speci cally using the concept of \Arti cial Potential Field (APF)", where an arti cial force de ned as the gradient of an APF acts on the trajectory, the goal position has the lowest potential and states where a collision would occur have a high potential. Alternatively, Vector Field Histogram (VFH) [12] and VFH+ [13] discretize the possible trajectories into a polar histogram and balance path smoothness, distance to goal and obstacle avoidance. Both APF and VFH+ have been applied to UAV obstacle avoidance in [14], [15] and [16] respectively, with APF applied to more speci c and practical use cases where obstacle avoidance is required, such as obstacle avoidance with payload suspended under the UAV [17].

Although methods such as APF and VFH+ are applicable to obstacle avoidance, they cannot take full advantage of the agility of some quadrotor UAV systems. Directly nding an optimal trajectory utilizing the UAV agility while avoiding obstacles is not computationally feasible in real time. However, instead of directly optimizing the trajectory, precomputed motion primitives [18] can be used to build a feasible trajectory. These can be state-based, control-based, or motion-based [19]. Improvements to motion primitive-based search algorithms for local obstacle-aware trajectory planning were made in [20]. To further decrease trajectory computation time and therefore latency, in addition to using precomputed motion primitives, the mapping step can be bypassed. Methods such as [19] or [21] use most recent image data to generate local trajectories. Finally, multiple levels of motion primitive delity can be used

to smooth out the resulting trajectory [21]. These methods have been successfully integrated into complete navigational pipelines and extensively tested [22].

Due to the fact that the quadrotor UAV is di erentially at [23], an alternative to motion primitive-based planning is directly solving the trajectory optimization problem by creating polynomial trajectories [24], [25]. The method introduced in [25] has been applied along with a corridor planner in [26], with performance exceeding some methods based on Reinforcement Learning (RL). Obstacle avoidance can also be bundled with control by using an adapted version of MPC, where obstacle avoidance can be included as a condition of trajectory optimization [27]. Newer methods also consider dynamic obstacles [28]. However, this approach is not as common since trajectory optimization and control are usually considered separate tasks.

- Learning-based Methods

Although recent pipeline-based methods have signi cantly advanced agile ight capabilities [26], there has also been a growing interest in alternatives that mitigate latency, such as learning-based approaches that generate control commands directly from sensor input [5]. Early implementations used linear classi ers trained on manually selected features and simple discretized command structures using Imitation Learning (IL) [29]. Later, a wider adoption of Convolutional Neural Network (CNN) models enabled the use of images directly as input to the planner [30], [31]. These were trained using RL, which produced impressive results, but required a large amount of data and special care to allow transfer from simulation to reality without requiring additional learning. In [5], IL from a privileged sampling-based expert policy was used to train a policy, which converts depth images directly to candidate trajectories to be evaluated and applied. Notably, this controller can follow a trajectory while avoiding obstacles in previously unseen environments and without requiring any training in the real world. Additional research in this area has focused on using better models [32].

Although these methods produce impressive results, they embed very little prior information about the systems they are controlling, treating them instead as black boxes. This creates two notable drawbacks. Firstly, it makes them potentially unsafe, as their performance cannot be fully guaranteed [33] and additional processing is required to guarantee safety. Secondly, RL and IL-based methods have very low sample e ciency [34].

Recent research has suggested mitigating the low sample e ciency by using di erentiable simulations and optimizing physics-driven loss functions instead of a reward for interacting with the environment [6]. According to the authors of [6], their method uses only 10% of the samples that an RL-based method would require, while reaching better results in comparable experiments. In [34], the training of RL and di erentiable physics-based models is compared to the latter, learning faster and reaching better results. A method using di erentiable simulation is also shown to stabilize a quadrotor from a manual throw without explicitly estimating the UAV state at any point [34].

2.3 Model Predictive Path Integral

MPPI is a variant of MPC utilizing principles of Path Integral (PI) [35] control. Where traditional MPC solves the local trajectory optimization problem using iterative optimization methods, MPPI uses a Monte Carlo sampling-based approach [36]. This allows MPPI to work with gradient-free and even non-smooth cost functions. If a lower-level controller is present, control commands can be extracted directly from the generated local trajectory.

Pure MPPI control has a number of drawbacks. First, a good initialization of the nominal commands is required [37]. For UAV control, this can be solved by reusing the results of previous MPPI iterations. Second, disturbances that are not taken into account by the model used for path integration can cause issues with convergence. This can be addressed by generating rollouts from the desired state and implementing an ancillary controller to follow this trajectory (effectively using the MPPI only as a local planner and delegating the controller functionality) [38], [39], or by ensuring the used model is as accurate as possible. Further improvements address the inability of MPPI to directly enforce constraints by augmenting it with Constrained Covariance Steering (CCS) [40] or Conditional Value-at-Risk (CVaR) [41].

MPPI has successfully been applied to quadrotor control in both simulated experiments [42] and real-world flight and separately with obstacle avoidance capabilities [8].

3 Methodology

This section begins with a brief overview of the MPPI framework. Then it details the architecture of the proposed GMPPI controller, highlighting the integration of depth sensing, the adaptations made to the core MPPI algorithm, and the development of a custom designed cost function.

MPPI is a controller capable of precisely tracking trajectories while avoiding obstacles [8]. It is rst initialized with a nominal control sequence u_j^{nom} of length N. In each iteration, K disturbance sequences u_j^k, each of length N, are sampled from a normal distribution with zero mean. Rollout commands u_j^k and states x_{j+1}^k are then computed as

$$\begin{aligned} u_j^k &\in \mathcal{N}(0; \Sigma); \\ u_j^k &= u_j^{nom} + u_j^k; \\ x_{j+1}^k &= x_j^k + f_{RK4}(x_j^k; u_j^k; t); \end{aligned} \qquad \begin{aligned} &\\ &k = 1; \dots; K; \\ &j = 0; \dots; N-1: \end{aligned} \tag{3.1}$$

For UAV control, the state is $x = \begin{bmatrix} p^T & v^T & q^T & \omega^T \end{bmatrix}^T$, consisting of the position, velocity, and body rate vectors $p; v; \omega \in \mathbb{R}^3$, respectively, and unit quaternion rotation on the rotation group $q \in SO(3)$. Commands $u = \begin{bmatrix} F_t & \omega_c \end{bmatrix}$ consist of the total desired thrust $F_t \in \mathbb{R}^+$ and angular velocity $\omega_c \in \mathbb{R}^3$. Dynamics of x are de ned by

$$\begin{aligned} \dot{p} &= v; & \dot{v} &= \frac{1}{m} R(q) \begin{bmatrix} 0 \\ 0 \\ F_t \end{bmatrix} - D R(q)^T v + g; \\ \dot{q} &= \frac{1}{2} q \begin{bmatrix} 0 \\ \omega \end{bmatrix}; & \dot{\omega} &= J^{-1}(\tau - \omega_c \times J\omega_c); \end{aligned} \tag{3.2}$$

with Runge-Kutta 4 being used in (3.1) for forward integration of the system. $R(q)$ is a rotation matrix corresponding to the quaternion q and air resistance of the UAV approximated by a linear model with $D = \mathrm{diag}(c_x; c_y; c_z)$ composed of linear drag coe cients.

Each rollout state sequence x^k is evaluated by a cost function. The proposed cost function used in the GMPPI controller is presented in Section 3.5 in (3.20). Rollout costs are transformed into weights w_k using the Softmax transform

$$\begin{aligned} w_k &= \frac{1}{\eta} \exp\left(-\frac{1}{\lambda}(S_k - \rho)\right); \\ &= \sum_{k=1}^{K} \exp\left(-\frac{1}{\lambda}(S_k - \rho)\right); \\ \rho &= \min\{S_1; \dots; S_K\}: \end{aligned} \tag{3.3}$$

Finally, the nominal control actions are set to the weighed average of the rollout commands

$$u_j^{nom} := \sum_{k=1}^{K} w_k \, u_j^k \tag{3.4}$$

and the rst command u_0^{nom} is applied to the controlled UAV.

3.1 Depth Camera Integration

Enabling obstacle avoidance during fast flight requires minimizing the delay between sensor input and control output. Instead of constructing and performing obstacle avoidance based on an environment map or processing the input image with a machine learning model [5], the availability of rollout states $x_j^k = \begin{bmatrix} p_j^k & v_j^k & q_j^k & \omega_j^k \end{bmatrix}$ is used.

The dimensions of the UAV L, W and H are used to define a set

$$H_j^k = \left\{ p_j^k + \frac{1}{2} \begin{bmatrix} L \\ W \\ H \end{bmatrix} \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \end{bmatrix} \middle| \alpha_1, \alpha_2, \alpha_3 \in \{-1, 1\} \right\}, \tag{3.5}$$

of all corner points of the UAV shifted outwards by a safety multiplier $m_o > 1$ and project each of the corner points as well as the UAV center point $p_h \in H_j^k \cup p_j^k$ onto the latest available depth image to determine if a collision occurs, as shown in Fig. 3.1. This approach eliminates nearly all sources of latency, with the exception of that introduced by the camera frame rate.

The sensor used in this work is a depth camera with an intrinsic matrix K that describes the transformation from the camera reference frame C to the image reference frame I. The camera is rigidly mounted to the UAV, with the transformation from the body-fixed reference frame B to C described by a matrix M. The transformation from B to the world frame W at the time of the latest image being captured is described by $R(q)_I$.

The position of each projected point p_h in the camera reference frame C is obtained as

$${}^C p_h = (R(q)_I M)^{-1} p_h. \tag{3.6}$$

This approach allows reusing a single image across multiple control iterations as it automatically compensates for the position and orientation shift of the UAV since the last image was taken. The distance of the rollout state position from the camera ${}^C p_h$ is compared to the distance d_{px} measured by the camera at the pixel coordinates ${}^I p_h = K\,{}^C p_h$, where the rollout position would be projected on the depth image.

The existence of a collision is determined by checking if the distance of the rollout position from the camera is similar to the actual distance measured by the camera at the pixel where the rollout state position would be projected, that is, by checking if the function

$$\mathrm{Col}(p_h) = \mathbb{1}_{\|{}^C p_h\| \in [d_{px} - d_m;\, d_{px} + d_a]} \tag{3.7}$$

evaluates to 1. There is a margin for clearing the obstacle from the front d_m and an assumed depth d_a to allow the controller to optimistically plan a return path to the reference trajectory even if the path is not fully visible yet.

3.2 Dynamic Rollout Time Steps

Small agile quadrotors are limited in terms of the hardware they can carry on board, which in turn limits the available computational power. This restricts both the number of GMPPI rollouts as well as the number of simulation steps in each rollout.

To ensure that the sensor range is fully utilized at any flight speed, a vector n of N elements is defined, where each element n_j acts as a time step multiplier at the j-th step of each rollout.

Figure 3.1: Image captured by a depth camera in simulation with the projection of the front four corners of the UAV collision box as de ned in (3.5) in each of the positions in the nominal trajectory p_j^{nom} $8j \ 2 \ (0; N + 1)$, which is the result of applying the nominal control sequence u_j^{nom} $8j \ 2 \ (0; N)$ to the UAV in its current state x.

The corresponding time step is calculated as $t_j = n_j \ t_0$, where $t_0 = 0{:}01s$ is the base controller update period.

The rst M multipliers are xed at smaller values represented by n_{near} to maintain good simulation precision near the current state. The remaining $N \ M$ multipliers are set based on the average speed of the UAV across the nominal state sequence v_{avg}^{nom} and the sensor range s such that

$$n_{far} = \frac{\frac{s}{v_{avg}^{nom} \ t_0} \ \sum_{j=0}^{M} n_j}{N \ M}; \tag{3.8}$$

which ensures that the range of the used sensor is fully utilized. The nal vector is

$$\begin{aligned} n_{0:::M \ 1} &= n_{near}; \\ n_{M:::N} &= n_{far}; \end{aligned} \qquad 0 < M < N; \tag{3.9}$$

and a the layout of a typical rollout in terms of time is illustrated by Fig. 3.2.

3.3 Deterministic Yaw Control

In sampling-based methods such as MPPI, a key factor in uencing performance is the density of the simulated rollout trajectories in the space of all possible rollout trajectories. The e ective volume of this space grows with dimensionality, so a xed number of rollouts becomes increasingly sparse in higher-dimensional space. In machine learning, this phenomenon is called the curse of dimensionality.

It is therefore bene cial to reduce the dimensionality of the space of all possible rollout trajectories. Because rotation around the yaw axis has no e ect on the ability to avoid obstacles, this control variable can be used to follow the reference heading h^{ref} and can be controlled using a proportional controller as

$$! _z = k_{MPPI ;z} \ (h; h^{ref}) + ! _{z;ref}; \tag{3.10}$$

where $k_{MPPI ;z}$ is the proportional feedback gain.

Figure 3.2: Illustration of the layout of the rollout states x_j^k in time. Earlier steps are shorter to ensure precision, while later steps are longer and their length is dynamically adjusted to use the full range of the available depth sensor.

3.4 Geometric Control in Candidate MPPI Trajectories

The quadrotor UAV is a differentially flat system [23], which allows expressing commands required to follow a trajectory described by a flat output in terms of the trajectory and its derivatives.

Adding terms to compensate for any errors in the current state leads to an algorithm controlling the UAV directly on the SE(3) group, similar to those presented in [23] and [9]. This allows generating rollouts that precisely follow the reference trajectory. Introducing at least one of these precise rollouts allows shifting the focus of random MPPI rollouts toward exploration and obstacle avoidance.

A version of the SE(3) controller [9] is incorporated and modified to output total thrust $F_{t;j}^k$ and desired body rates $\omega_{c;j}^k$ for time step j of the rollout at index k. First, the position, velocity, and rotation errors are computed as $e_p = p_j^{ref} - p_j^k$, $e_v = v_j^{ref} - v_j^k$, and $e_R = \frac{1}{2}(R_d^T R(q_j^k) - R(q_j^k)^T R_d)^\vee$. The coefficient matrices K_p^k, K_v^k and K_r^k, constructed as

$$K_p^k = \mathrm{diag}\begin{pmatrix} k_{pxy} \\ k_{pxy} \\ k_{pz} \end{pmatrix}; \quad K_v^k = \mathrm{diag}\begin{pmatrix} k_{vxy} \\ k_{vxy} \\ k_{vz} \end{pmatrix}; \quad K_r^k = \mathrm{diag}\begin{pmatrix} k_{rxy} \\ k_{rxy} \\ k_{rz} \end{pmatrix}; \tag{3.11}$$

are input parameters to the geometric controller. The desired force vector is computed as

$$F = K_p^k e_p + K_v^k e_v + (a^{ref} - g + R(q_j^k) D R(q_j^k)^T v) m \tag{3.12}$$

and the total thrust is $F_{t;j}^k = F \; R(q_j^k) \; 0 \; 0 \; 1^T$. With $F_n = \|F\|$, desired orientation can be calculated from the desired force vector F as

$$R_d = \begin{bmatrix} (F_n \times h^{ref}) \times F_n \\ F_n \times h^{ref} \\ F_n \end{bmatrix}^T : \tag{3.13}$$

Finally, the angular velocity is calculated as $\omega_{c;j}^k = K_R^k e_R + \omega^{ref}$. The command used in the rollout simulation is

$$u_j^k = F_{t;j}^k \; \omega_{c;j}^k : \tag{3.14}$$

Figure 3.3: Combination of geometric and random rollouts. A total of K rollout sequences are simulated, of which K 32 are random and 32 use geometric control. All rollout commands are used to compute the next nominal command sequence $\mathbf{u}_j^{nom}$ as well as the next set of rollout states x_{j+1}^k.

The SE(3) geometric controller is capable of precisely following reference trajectories, but requires tuning and the ideal con guration depends on the speci c UAV and trajectory to be followed. Running multiple rollouts with commands generated using geometric control enables using di erent parameters in each rollout, e ectively testing multiple con gurations of the geometric controller at once. To prevent warp divergence on the Graphics Processing Unit (GPU) used for the testing, the number of rollouts using the geometric controller K_{SE3} must be a multiple of 32. Therefore, $K_{SE3} = 32$ was chosen, with the split between random and geometric rollouts illustrated in Fig. 3.3.

To use a di erent geometric controller con guration in each rollout, the noise $k_{SE3}^k \; 2 \; N(0; \; _{SE3})$ is sampled from a normal distribution with zero mean and a covariance matrix $_{SE3}$. This noise is then added to a vector of GMPPI parameters specifying a base geometric controller con guration $k_{SE3} = k_{pxy} \; k_{pz} \; k_{vxy} \; k_{vz} \; k_{rxy} \; k_{rz}$ creating a new con guration speci c to the rollout at index k and de ned by the parameters

$$k_{SE3}^k = k_{SE3} + \; k_{SE3}^k: \tag{3.15}$$

The vector k_{SE3}^k is then reshaped into the coe cient matrices described by (3.11).

3.5 Cost Function

The MPPI method creates a nominal trajectory as a weighted average of rollout trajectories, with higher weights w_k assigned to rollout trajectories with a lower cost S_k using the Softmax transform as shown in (3.3). An appropriate cost function assigns higher costs S_k to trajectories that exhibit unwanted characteristics such as excessive maneuvers or a large trajectory tracking error. Similarly, it assigns lower costs to trajectories that conform to the objectives of the controller. The proposed controller has three goals, which are precise trajectory tracking, smooth and stable ight, and nally obstacle avoidance, which must always take precedence over the remaining objectives.

Tracking Precision

To enable precise trajectory tracking, position, velocity, orientation and angular velocity error magnitudes

$$\begin{aligned} e_p &= \; p_j^k \quad p_j^{ref} \; _2; \quad e_v = \; v_j^k \quad v_j^{ref} \; _2; \\ e_q &= d_q(q_j^k; q_j^{ref}); \quad e_! = \; ! \; _j^k \quad ! \; _j^{ref} \; _2; \end{aligned} \tag{3.16}$$

relative to the reference are calculated and multiplied by the coefficients $c_j^p; c_j^v; c_j^q$ and c_j^l respectively. The orientation difference approximation [8] is defined as

$$d_q(q_1; q_2) = 1 - \langle q_1; q_2 \rangle^2. \tag{3.17}$$

The coefficients can vary with the time index j in the rollout, as shown in Fig. 3.4. This allows assigning lower cost to those rollout trajectories which deviate from the reference trajectory only in later steps of the rollout simulation, which promotes exploration and reallocates precision emphasis toward earlier timesteps of the simulated rollout trajectories. This increases trajectory tracking precision, as well as smoothness. Using the Euclidean norm for position instead of its square requires computing a square root but prevents the creation of an erratic nominal trajectory in cases where a larger departure from the reference trajectory is required to avoid an obstacle.

- Smoothness

To promote smooth flight, a cost component is added for any amount of jerk $\underline{a}^k$, which is larger than a multiple t_j of the jerk present in the reference trajectory $\underline{a}^{ref}$, that is, an excess jerk quantity

$$e_j = \max\left(\underline{a}_j^k - t_j \underline{a}^{ref}; 0\right) \tag{3.18}$$

is introduced and multiplied by a coefficient c_j^j. This punishes any excess control input equally based on how much it disturbs the smoothness of the flight, while still allowing sharp trajectories to be followed precisely. The reference and rollout jerk values are calculated using a finite-difference approximation from acceleration data. Typical noise amplification concerns do not apply, as both the reference and rollout trajectories are noise-free.

- Obstacle Avoidance

Finally, obstacle avoidance capability is introduced by adding cost to any rollout that collides with an obstacle. The set of points $H_j^k \oplus \{p_j^k\}$, defined in Sec. 3.1, is a collision box for the UAV and the function $\mathrm{Col}(p_h)$, defined in (3.7), returns 1 when a collision is detected. The number of points p_h, which collide with an obstacle is then multiplied by $N - j$ to assign a higher cost to collisions occurring earlier in the simulated rollout. The total obstacle cost is

$$e_{obs} = (N - j) \sum_{p_h \in (H_j^k \oplus \{p_j^k\})} \mathrm{Col}(p_h); \tag{3.19}$$

which is multiplied by a coefficient $^{obs}c_j$. This approach provides more granularity than a simple binary switch, which allows for a more informed selection of candidate trajectories.

- Resulting Cost Function

The resulting cost function for a single point in a single rollout is the sum of all partial costs

$$C_j^k = \begin{bmatrix} c_j^p & c_j^v & c_j^q & c_j^l & c_j^j & c_j^{obs} \end{bmatrix} \begin{bmatrix} e_p & e_v & e_q & e_l & e_j & e_{obs} \end{bmatrix}^T \tag{3.20}$$

and the cost of a whole rollout is

$$C^k = \sum_{j=1}^{N} C_j^k. \tag{3.21}$$

Figure 3.4: For each rollout timestep j, the set of cost coefficients $c_j^p; c_j^v; c_j^q$ and $c_j^!$ can differ. Coefficients intended to increase precision are set to higher values for the beginning of each rollout, meaning that generally $c_0 \quad c_N$.

Algorithm 1: Geometric MPPI Iteration

Input: Current state estimation $\hat{x}$, Nominal command sequence u_{in}^{nom}, Reference command sequence u_{in}^{ref}, Depth Image

Params: Number of rollouts K, Number of geometric rollouts K_{SE3}, Rollout length N, Rollout noise , SE3 Parameter noise $_{SE3}$

$n = (n_0; \ldots; n_M; n_{exp}; \ldots; n_{exp})$
for $j = 0, \ldots, N$ do // Set timestep lengths
  $t_j = n_j \quad t_0$
$u^{nom} =$ ResampleNominalCommands(u_{in}^{nom}, n)
$u^{ref} =$ ResampleReferenceCommands(u_{in}^{ref}, n)
// Simulate K rollouts (Parallel on GPU)
for $k = 1, \ldots, K$ do
  $x_0^k = \hat{x}$
  if $k < K_{SE3}$ then // SE(3) Rollout
    $k_{SE3}^k \quad N(0; \ _{SE3})$
    $k_{SE3}^k = k_{SE3} + \ k_{SE3}^k$
    for $j = 0, \ldots, N$ do
      $u_j^k =$ SE3Command(x_j^k, k_{SE3}^k)
      $x_{j+1}^k = x_j^k + f_{RK4}(x_j^k; u_j^k; \ t_j)$
      $C_j^k =$ CalculateCost$(c_j^k, x_{j+1}^k, x_{j+1}^{ref}$, Depth Image)
  else // MPPI Rollout
    for $j = 0, \ldots, N$ do
      $u_j^k \quad N(0; \ )$
      $u_j^k = u^{nom} + \ u_j^k$
      $! \ _z = k_{MPPI \ ;z} \backslash (h; h^{ref}) + ! \ _{z^{ref}}$
      $x_{j+1}^k = x_j^k + f_{RK4}(x_j^k; u_j^k; ! \ _z; \ t_j)$
      $C_j^k =$ CalculateCost$(c_j^k, x_{j+1}^k, x_{j+1}^{ref}$, Depth Image)
    $C^k = \sum_{j=0}^{N} C_j^k$

$u^{nom} =$ AverageWeighedCommands$()$
return u_0^{nom}

The proposed GMPPI controller is summarized in Algorithm 1. It receives a state estimate, a nominal command sequence from the previous run, and a reference command sequence extracted from the reference trajectory. First, the nominal and reference command sequences are resampled to t the timestep sequence that enables the use of the entire range of the depth sensor. All data is then moved to the GPU. A total of K rollouts are simulated, of which K_{SE3} use an implementation of the SE(3) geometric controller to generate commands. The remaining rollouts have their commands generated randomly according to a covariance matrix . Rollouts are generated in parallel, with each timestep of all rollouts being executed before the next. This includes noise and command generation, simulation, and cost calculation. This work ow ensures optimal performance on the GPU. Finally, costs are summed and moved back to the Central Processing Unit (CPU) along with the rollout command sequences, and a weighed average of the rollout commands is saved as the nominal command sequence to be used in the next iteration of the algorithm. The rst element of this new nominal sequence is returned to be applied to the system.

4 Results

This section introduces the performance verification setup, the parameters used for testing, and finally the results of the conducted experiments. The proposed GMPPI controller was integrated into the UAV control and estimation system developed by the Multi-robot Systems Group (MRS) at the Czech Technical University (CTU) [43]. An illustration of the complete system setup is in Fig. 4.1. All simulated experiments were conducted on a laptop with an Intel Core i7-1165G7 CPU and an 869-core NVIDIA GeForce MX450 GPU, which the proposed controller utilizes for parallel rollout simulation. The reference trajectories to be tracked in each simulation were generated by an MPC-based tracker [43].

4.1 UAV & Controller Parameters

The GMPPI controller embeds substantial prior knowledge about the UAV, enabling reuse across different UAV models. In particular, the feasibility of rollout commands is verified by calculating and clipping the desired rotor thrusts at each simulation step [8], requiring knowledge of the UAV mass m, the arm length l and the inertia matrix J. The parameters used in the simulations of the proposed controller are shown in Table 4.1 and the GMPPI parameters along with control input limits for both the minimum and maximum thrust as well as the maximum absolute angular rate values are shown in Table 4.2.

For obstacle avoidance in the simulated runs, a stereo depth camera with the intrinsic matrix

$$K = \begin{bmatrix} 924 & 0 & 360 \\ 0 & 924 & 640 \\ 0 & 0 & 1 \end{bmatrix} \tag{4.1}$$

is mounted to the UAV. The safety margin for passing an obstacle from the front is $d_m = 0.3$ m and the assumed depth of an obstacle is $d_a = 4.0$ m. The sensor range is $s = 13$ m. All experiments with obstacles had the camera tilted up by a fixed angle depending on the maximum intended speed to allow full use of the sensor range.

The cost coefficients from (3.20), which are shown in Fig. 4.2, are dependent on the rollout index, except for the allowed excess jerk multiple $\epsilon_j = 1.4$. This forces all rollout trajectories to remain roughly as smooth as the reference trajectory. The base obstacle cost $q_j^{obs} = 100$. This is applied in (3.19) so that avoiding collisions becomes the highest priority of the controller.

As with the cost parameters, the noise covariance matrix $\Sigma = \mathrm{diag}(F_t, \omega_x, \omega_y)$, which is shown in Fig. 4.3, is dependent on the rollout time index j, allowing larger changes later in the rollout to promote exploration, but limiting the noise earlier in each rollout to promote stability.

Table 4.1: UAV Parameters used for all simulated runs

Model Parameters			Dimensions			Drag Coefficients	
m	[kg]	1.21	L	[m]	0.35	c_x	0.28
l	[m]	0.15	W	[m]	0.35	c_y	0.35
c_{tf}	[m]	0.012	H	[m]	0.215	c_z	0.7
Inertia Matrix			J	[gm^2]		diag([7.06 7.06 13.6])	

Figure 4.1: System diagram of the simulation setup. Blocks outlined in pink and labeled (MRS UAV System), as well as the UAV Plant model are supplied by the MRS UAV system. The GMPPI controller receives the reference trajectory, estimated state and depth image and produces a control command, which is sent to the PX4 low-level flight controller.

Table 4.2: Controller Parameters and Limits

MPPI Parameters		SE3 Parameters		Control Limits		
K	1024	k_p	$[6.0\ 6.0\ 15.0]$	$F_{T,min}$	[N]	0.46
K_{SE3}	32	k_v	$[4.0\ 4.0\ 8.0]$	$F_{T,max}$	[N]	20.6
N	30	k_r	5.0	$\omega_{xy,max}$	[rads^{-1}]	10.0
$k_{MPPI,z}$	2.0			$\omega_{z,max}$	[rads^{-1}]	2.0

Figure 4.2: GMPPI Cost parameters c_j used in (3.20) shown in a heat map. A decrease in c_j^p, c_j^v, c_j^q and c_j^ω after the beginning of each rollout promotes exploration.

Figure 4.3: GMPPI noise parameters for the angular velocities ω_x and ω_y and thrust F_t.

(a) Figure 8 (b) Hypotrochoid

Figure 4.4: Trajectories followed by GMPPI shown in green and the corresponding reference trajectories shown in blue.

4.2 Reference Tracking

To test the ability of the GMPPI controller to follow reference trajectories, three reference trajectories are generated, two of which are shown in Fig. 4.4. Each trajectory is followed in loops for a set amount of time. First, a Hover trajectory asks the UAV to remain still, The Figure 8 trajectory (shown in Fig. 4.4a) is generated as

$$f(20 \sin(t); 5 \sin(t); 6.0) \mid t \in [0; 2\pi]\} \tag{4.2}$$

and the Hypotrochoid trajectory (shown in Fig. 4.4b) is generated with parameters $R = 20$, $r = 12$ and $d = 12$. All trajectories specify the heading of the UAV to be equal to the projection of the velocity vectors to the horizontal XY plane to verify the ability to track the heading reference to be able to see in the direction of the velocity vector to avoid obstacles.

The GMPPI controller set up with parameters from Sec. 4.1 is compared to an existing standalone implementation of MPPI [8] and a standalone implementation of the SE(3) controller [9]. Three additional configurations are included to verify the contribution of each of the key features to the overall performance of the controller. The \no SE(3)" variant has

Table 4.3: Results of reference tracking experiments.

a con guration modi ed with $K_{SE3} = 0$, e ectively disabling the integration of geometric rollouts. The \const t" variant includes $n = (n_{exp}; : : : ; n_{exp})$, which equalizes the timestep lengths in all indexes of all rollouts. Finally, the \const noise" variant has all costs and noise parameters set to be constant for all time steps of all rollouts.

The results are shown in Table 4.3. In the absence of obstacles, GMPPI is able to precisely follow trajectories, improving the precision over standalone MPPI both in agile ight and in hover. In agile ight, the GMPPI controller achieves a lower root mean square position error than even the standalone SE(3) controller [9]. This is achieved by including multiple geometric SE(3) rollouts with varying con gurations, as well as by including a model of aerodynamic drag. Furthermore, the ability of the GMPPI controller to assign low costs to these rollouts, even though they contain sharp maneuvers, allows for the use of the full potential of the geometric SE(3) rollouts. The assignment of low costs to sharp trajectories is made possible by the custom cost function, which does not penalize maneuver sharpness if it appears in the reference trajectory. With SE(3) rollouts disabled, the controller struggles and delivers worse results than the baseline, with signi cant position errors in the Figure 8 trajectory and a crash caused by these position errors in the Hypotrochoid trajectory. This is caused by the tuning of the remaining controller parameters being done under the assumption of the SE(3) rollout presence. Setting constant time steps t causes a slight but consistent decrease in performance due to reduced simulation precision. Finally, setting noise and cost parameters constant with respect to the rollout simulation step index causes issues when hovering, as the controller cannot prioritize short-term precision over long-term precision, which is generally bad for random rollouts when hovering, just like in the baseline MPPI controller.