Enhancing Lattice-based Motion Planning with Introspective Learning and Reasoning

Mattias Tiger[1] and David Bergström[1] and Andreas Norrstig[1] and Fredrik Heintz[1]

Abstract—Lattice-based motion planning is a hybrid planning method where a plan is made up of discrete actions, while simultaneously also being a physically feasible trajectory. The planning takes both discrete and continuous aspects into account, for example action pre-conditions and collision-free action-duration in the configuration space. Safe motion planning rely on well-calibrated safety-margins for collision checking. The trajectory tracking controller must further be able to reliably execute the motions within this safety margin for the execution to be safe. In this work we are concerned with introspective learning and reasoning about controller performance over time. Normal controller execution of the different actions is learned using machine learning techniques with explicit uncertainty quantification, for safe usage in safety-critical applications. By increasing the model accuracy the safety margins can be reduced while maintaining the same safety as before. Reasoning takes place to both verify that the learned models stays safe and to improve collision checking effectiveness in the motion planner using more accurate execution predictions with a smaller safety margin. The presented approach allows for explicit awareness of controller performance under normal circumstances, and detection of incorrect performance in abnormal circumstances. Evaluation is made on the nonlinear dynamics of a quadcopter in 3D using simulation. Video: https://youtu.be/F0p0FFRZwM4

Index Terms—Motion and Path Planning; Collision Avoidance

I. INTRODUCTION

SAFE motion planning is a necessity for robots navigating in dynamic, unstructured, and human-tailored environments such as indoors or in urban settings. Operating in real dynamic environments make introspective capabilities important since situations can easily change beyond reasonable design assumptions: hardware degradation, modeling errors, software bugs as well as rare external disturbances such as extreme weather or unexpected adversarial attacks from other agents. It is important to know what normal motion execution
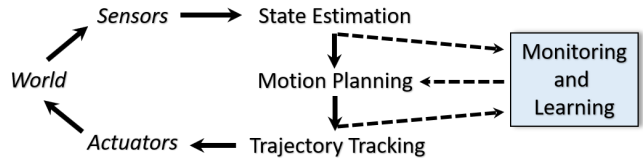
Fig. 1

When motion execution is safety-critical, for an autonomous agent such as a robot, it is important for the robot itself to know what normal execution looks like (learning) and if the current motions are normal or abnormal (monitoring).

looks like from the robot's perspective and to detect when the executed behavior become abnormal in a timely manner.

Lattice-based motion planning is one of the most frequently used motion planning technique in real implementations for automated vehicles [1]. It works by restricting motion to a finite number of pre-computed *motion primitives* which moves the robot between points on a state-lattice, and a physically feasible trajectory is found as a sequence of compatible motion primitives using graph-search. It is a technique appropriate for dynamic environments [2] and a fast search for the (resolution) optimal trajectory can be performed in real-time, taking comfort, safety and vehicle constraints into consideration [1]. Lattice-based motion planning has successfully been implemented on various robots [3], [4], [5], with for example recent advancement on the challenge[1] to re-plan new collision free trajectories with multiple dynamic obstacles in real-time [3].

It can be incredibly challenging to specify what normal behavior looks like using formal languages such as Metric Temporal Logic (MTL) [6] or Probabilistic Signal Temporal Logic (ProbSTL) [7], for use in modern runtime verification frameworks to perform execution monitoring of the motion plans. Machine learning can be leveraged to complement formal safety requirements with learned models of normal action execution of for example robot manipulation tasks [8].

Lattice-based motion planning provides an opportunity to effective learning and monitoring of motion action execution due to the already discrete and well defined motion primitives (the actions). *In this paper we present a general approach for enhancing lattice-based motion planning methods with (1) learning models of normal motion primitive execution, (2) using the learned models to improve collision checking effectiveness during planning and to (3) efficiently monitor the motion primitive execution for abnormalities.* The approach makes few assumptions and acts as a flexible plug-in to any modern AI-robotic software stack (Figure 1).

Since both collision checking and abnormality detection

are safety-critical the learning is performed using machine learning techniques with explicit uncertainty quantification from Bayesian machine learning [9]. The monitoring for abnormalities also verifies at runtime that the learned models remain valid for collision checking use in the motion planner.

Modern lattice-based motion planning and control given dynamic obstacles is described in Section II. Related work in III. Our approach for learning, improving collision checking and monitoring motion primitive executions is presented in Section IV, with results in V and conclusions in Section VI.

II. PROBLEM FORMULATION

A. Motion Planning

Consider a robot that is modeled as a time-invariant non-linear system

$$\dot{x}(t) = f(x(t), u(t)), \tag{1}$$

where $x(t) \in \mathbb{R}^n$ denotes the robot's states[1] and $u(t) \in \mathbb{R}^m$ its control signals. The robot has physically imposed constraints on its states $x(t) \in \mathcal{X}$ and its control signals $u(t) \in \mathcal{U}$. The robot operates in a 2D or 3D world $\mathcal{W}(t)$, i.e. $\mathcal{W}(t) \subset \mathbb{R}^2$ or $\mathcal{W}(t) \subset \mathbb{R}^3$. There are regions $\mathcal{O}_{\text{obs}}(t) \subset \mathcal{W}(t)$ which are occupied by static and dynamic obstacles. Free space $\mathcal{X}_{\text{free}}(t) = \{x(t) \in \mathcal{X} \mid \mathcal{O}_{x(t)} \cap \mathcal{O}_{\text{obs}}(t) = \emptyset\}$ is where the region occupied by the robot $\mathcal{O}_{x(t)}$, transformed by the state $x(t)$, is not in collision with any obstacle at time t.

The objective of the motion planner is to produce a feasible reference trajectory $(x_0(t), u_0(t))$, $t \in [t_S, t_G]$ that moves the robot from a starting state x_S to a goal state x_G while optimizing a given performance measure J, for example a compromise between minimum time and smoothness as in [3]. Taking the obstacles into account, the reference trajectory also must be collision free. This problem is called the dynamic motion planning problem (DMPP) [3]

$$
\begin{aligned}
\underset{u_0(\cdot),\, t_G}{\text{minimize}} \quad & J = \int_{t_S}^{t_G} L(x_0(t), u_0(t), t)\, dt \\
\text{subject to} \quad & \dot{x}_0(t) = f(x_0(t), u_0(t)), \\
& x_0(t_S) = x_S, \quad x_0(t_G) = x_G, \\
& x_0(t) \in \mathcal{X}_{\text{free}}(t), \quad \forall t \in [t_S, t_G], \\
& u_0(t) \in \mathcal{U}, \quad \forall t \in [t_S, t_G],
\end{aligned}
\tag{2}
$$

and it is typically intractable to even find a feasible solution.

B. Lattice-based Motion Planning

Lattice-based motion planning [2] is a tractable approximation to DMPP [3] where the state space is discretized into a state lattice and a finite number of translation-invariant motion primitives (actions) are constructed to allow motion between states (nodes) on the lattice. Graph search techniques such as A* with an admissible heuristic can be used to find a valid sequence of motion primitive actions from x_S to x_G. A motion primitive action $a_i \in \mathcal{A}$ is a trajectory

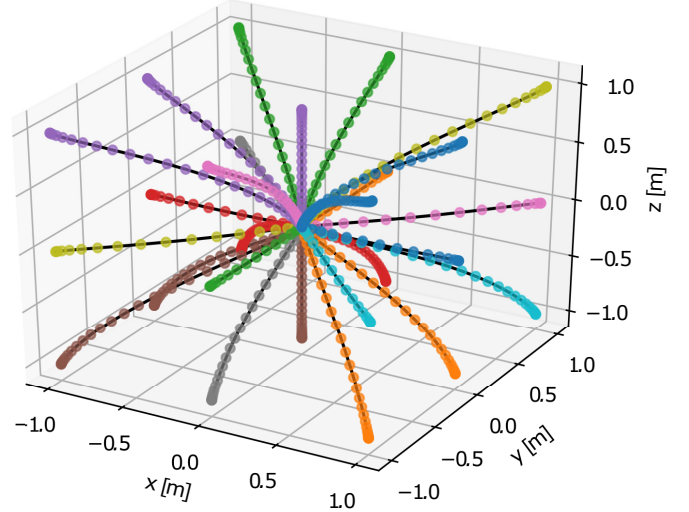[1]For example position, velocity, orientation and angular velocity in 3D.

Fig. 2
The 26 motion primitives with zero initial and final velocity, out of a total of 104 primitives. All start at position $[0,0,0]^T$.

$(x_0^i(t), u_0^i(t))$, $t \in [0, t_F^i]$ with initial state x_I^i and final state x_F^i on the lattice grid $\mathcal{X}_d \subset \mathcal{X}$ which satisfies the following

$$
\begin{aligned}
& \dot{x}_0^i(t) = f(x_0^i(t), u_0^i(t)) && \text{(3a)} \\
& x_0^i(0) = x_I^i \in \mathcal{X}_d, \quad x_0^i(t_F^i) = x_F^i \in \mathcal{X}_d && \text{(3b)} \\
& x_0^i(t) \in \mathcal{X}, \quad u_0^i(t) \in \mathcal{U}, \quad \forall t \in [0, t_F^i] && \text{(3c)}
\end{aligned}
$$

The motion primitives are generated offline leveraging numerical optimal control using the same loss function $L(x_0(t), u_0(t), t)$ and are assigned the resulting objective function value J_i. Figure 2 show some motion primitives.

The Lattice approximation to the DMPP is

$$
\underset{a_1,\ldots,a_N,\, N}{\text{minimize}} \quad J = \sum_{n=1}^{N} J_n \tag{4}
$$

$$
\begin{aligned}
\text{subject to} \quad & (x_0^n(t), u_0^n(t)) = a_n \in \mathcal{A}, \quad \forall n, \\
& \left\| \mathbf{N} x_0^n(0) - \mathbf{N} x_0^{n-1}(t_F^n) \right\|_2 = 0, \quad \forall n > 0, \\
& \mathbf{T}(x_I)\, x_0^0(0) = x_I, \\
& \mathbf{T}_{N-1}\, x_0^N(t_F^N) = x_G, \\
& \mathbf{T}_{n-1}\, x_0^n(t) \in \mathcal{X}_{\text{free}}(\mathcal{T}_{n-1} + t), \,\, \forall t \in [0, t_F^n], \,\, \forall n,
\end{aligned}
$$

where $\mathbf{T}(x)$ is a translation matrix defined by the position in state x and $\mathbf{N}$ is a diagonal matrix with zeros at the position dimensions (projecting a state's position to zero under multiplication). $\mathbf{T}_K = \mathbf{T}(x_I) \prod_{k=0}^{K} \mathbf{T}(x_0^k(t_F^k))$ is the resulting translation of the first K motion primitives in the plan and $\mathcal{T}_K = t_S + \sum_{k=0}^{K} t_F^k$ is the start time of the K:th motion primitive in the plan.

The resulting plan $(t_S, x_I, a_0, \ldots, a_N)$ consists of a sequence of N motion primitive actions, $a_0, \ldots, a_N$, $\forall a_n \in \mathcal{A}$. The end time of the plan is $t_G = \mathcal{T}_N$. The reference trajectory $(x_0(t), u_0(t)), t \in [t_S, t_G]$ is constructed from the plan by sequential spatio-temporal concatenation of the sequence of motion primitives in the plan. Lattice-based motion planning is resolution complete and is equivalent with the DMPP in the resolution-limit.

The reference trajectory found by the motion planner is executed by a trajectory tracking controller, for example using a nonlinear Model Predictive Controller (MPC) [3]. The objective of the trajectory tracking controller is to have the robot follow the desired reference trajectory with a small tracking error $\tilde{x}(t) = x(t) - x_0(t)$ while keeping close to the feedforward control signal $\tilde{u} = u(t) - u_0(t)$. The continuous-time nonlinear MPC problem, which is solved with respect to the current time point t_0, is formulated as [3]

$$
\begin{aligned}
\underset{u(\cdot)}{\text{minimize}} \quad & ||\tilde{x}(t_0 + T)||^2_{\mathbf{P}_N} + \int_{t_0}^{t_0+T} ||\tilde{x}(t)||^2_{\mathbf{R}_1} + ||\tilde{u}(t)||^2_{\mathbf{R}_2} \, dt \\
\text{subject to} \quad & \dot{x}(t) = f(x(t), u(t)), \\
& u(t) \in \mathcal{U},
\end{aligned}
\tag{5}
$$

where the design parameters $\mathbf{P}_N, \mathbf{R}_1, \mathbf{R}_2$ are positive-definite weight matrices and T is the prediction horizon.

C. Collision Checking

The robot occupy a region $\mathcal{O}_{x(t)}$ that depends on its state (e.g. its position and orientation) at time point t. We want to find a reference trajectory $x_0(t)$ which at any time-point $t \in [t_S, t_G]$ is collision free $\mathcal{O}_{x_0(t)} \cap \mathcal{O}_{\text{obs}}(t) = \emptyset$.

There exists many possible sources of uncertainty in applied motion planning. These can be due to e.g. modeling error, sensor noise, noisy control or the unpredictability of other agents. The uncertainty can be categorized into sensing of state and predictability of future state, due to for example robot actions, of both the robot and its environment [10]. A common way to alleviate the uncertainty is to use a safety margin to mitigate uncertainty induced collisions [11]. The kind of safety margin $\mathcal{O}_{\text{safety}}(t) \subset \mathcal{W}(t)$ considered in this work is a region relative to the robot's coordinate frame that extends the spatial occupancy of the robot,

$$
\mathcal{O}_{x_0(t)} = \mathcal{O}^*_{x_0(t)} * \mathcal{O}_{\text{safety}}(t),
$$

where $\mathcal{O}^*_{x_0(t)}$ is the actual occupied region of the robot and the binary operator $*$ over regions is defined as

$$
\mathcal{O}_A * \mathcal{O}_B = \{p_A + p_B \,|\, p_A \in \mathcal{O}_A, \, p_B \in \mathcal{O}_B\}.
$$

Another approach is to extend the obstacle regions [12].

The safety margin can be divided into three main parts,

$$
\mathcal{O}_{\text{safety}}(t) = \mathcal{O}_{\text{perception}}(t) * \mathcal{O}_{\text{control}}(t) * \mathcal{O}_{\text{others}}(t),
$$

reflecting the uncertainties from the robot's perspective of the state of the world, the control result and in the behavior of others. In safety-critical applications where uncertainties are present, it is necessary to quantify the uncertainties explicitly as probability distributions in order to guarantee probabilistic safety [9] and probabilistic feasibility [13]. A safety margin can be constructed from such a representation to bound the probability of collision, e.g. to at most 0.01% per second, frequently called *chance constraint* in the literature [13].

The third safety margin component $\mathcal{O}_{\text{control}}$ is important to lattice-based approaches but has thus far been largely overlooked. Due to modeling errors, noise sources and hardware limitations it is in practice unreasonable to expect the trajectory tracking controller will follow the reference trajectory

perfectly. Even using exact models and having no noise, the accuracy depends on the motion primitive sequence, together with the choice of the design parameters $T, \mathbf{P}_N, \mathbf{R}_1, \mathbf{R}_2$. Such tracking errors tend to have both a bias and a covariance, and directly affect $\mathcal{O}_{\text{control}}$. In this paper we present a method for systematically deciding $\mathcal{O}_{\text{control}}$ for lattice-based motion planning, and also to runtime verify that $\mathcal{O}_{\text{control}}$ is indeed correctly probabilistically grounded. The latter to make sure that the safety guarantees are maintained in changing situations.

III. Related Work

The problem of real-time motion planning in cluttered, complex and dynamic environments has seen an increasing attention in the literature [2]. Advances such as multi-resolution lattices [14] have made lattice-based approaches suitable for cluttered and complex environments. Lattice-based motion planning achieve state-of-the-art (SOTA) performance in for example trucks with multiple trailers [5], and for quadcopters [3] for planning and re-planning, with both dynamics and time, in real-time with dynamic obstacles in complex environments.

We outline the general problem setting of [3] in section II-A and II-B. We show how to integrate our proposed introspective capabilities to lattice-based motion planning by providing a detailed context, and our work is fully compatible with the multi-resolution lattice approach in [3]. The multi-resolution lattice approach in [3] is in turn based on [14] and the soft constraints in [14] can be included in the loss function J_n.

Common practice is to relax the composition of $\mathcal{O}_{\text{safety}}(t)$ and to pick an arbitrary large region as $\mathcal{O}_{\text{safety}}(t)$ likely to satisfy a desired safety constraint. The most common simplification of this kind is to use a spherical safety margin like [15], defined by a single radius parameter. This practice, although computationally efficient, generally require a much larger $\mathcal{O}_{x_0(t)}$ than what is actually necessary and consequently reduce collision checking effectiveness. This can be caused by different position uncertainty in different directions and position bias, both of which may change over time in strength and direction (e.g. strong wind). Previous work has investigated different parts of $\mathcal{O}_{\text{safety}}(t)$ and provided methodology to probabilistically ground both $\mathcal{O}_{\text{perception}}(t)$ and $\mathcal{O}_{\text{others}}(t)$.

In [16], using lattice-based *motion planning under uncertainty*, the state uncertainty is taken into consideration explicitly in the reference trajectory. A consequence is that $\mathcal{O}_{\text{perception}}(t)$ can be probabilistically grounded for every time point t during planning. It can for example be as a 99% probability region of the state for time point t. On its own this corresponds to a safety margin for which it is at least 99% probable that the robot occupied region is within $\mathcal{O}_{x_0(t)}$.

If a realistic simulator of the target domain is available it can be used to simulate realistic behaviors of other agents, and the interaction between such agents and the robot. Such a simulator can be used to ground the uncertainty of $\mathcal{O}_{\text{others}}(t)$ and the margin can be tuned to satisfy for example a 99.9% probability of no collision [17]. Since $\mathcal{O}_{\text{others}}(t)$ is grounded in predictive models of other agents, it is applicable to various motion planning approaches including lattice-based methods.

The safety margin part $\mathcal{O}_{\text{control}}$ has been considered in other motion planning approaches, such as those based on

Rapidly-expanding Random Trees (RRT), where process noise is used to sample possible motion plan executions [13]. In [16] the process noise affects the covariance around the reference trajectory, but the uncertainty is unbiased.

Our contributions, apart from the presented integration into lattice-based motion planning, are useful outside of the lattice-based paradigm. Learning models and abnormality detection of motion primitives assume that there is a fixed set of motion primitives and that we can observe their execution, such as in SOTA for quadcopters motion planning [18], [19]. The improved effectiveness of collision checking further assumes that the models are a precise description of execution variability, which may not always hold with post-search trajectory optimization [4], [18]. This would cause our proposed method to also include the variation in different optimization results.

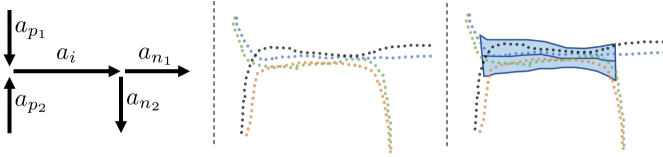## IV. INTROSPECTIVE LATTICE-BASED MOTION PLANNING AND CONTROL

Fig. 3
Illustration of learning a model of the *normal* execution of primitive a_i. **Left:** All valid plans of triplets with a_i in the middle (a_{p_*} previous and a_{n_*} next). **Center:** Observed execution of each triplets plan. **Right:** Mean predictive and 95% probability region of unimodal distribution (11) capturing the expected variability of normal a_i executions.

A. Learning Normal Primitive Execution

The execution of a motion primitive $a_i \in \mathcal{A}$, as perceived by the robot, is a discrete trajectory $\boldsymbol{x} = \hat{x}_{t_0}, \hat{x}_{t_1}, \ldots$ with time points $\boldsymbol{t} = t_0, t_1, \ldots$. Let $a_p \in \mathcal{A}$ and $a_n \in \mathcal{A}$ denote a valid previous and next motion primitive with respect to a_i such that $[a_p, a_i, a_n]$ is a valid plan. For every $a_i \in \mathcal{A}$, let $\boldsymbol{X}^i = \boldsymbol{x}^1, \ldots, \boldsymbol{x}^{J_i}$ denote the J_i observed executions of primitive a_i, one for every valid triplet $[a_p, a_i, a_n]$. All executions of primitive a_i have the same time duration. Note that the time-points will not be aligned, and the number of data-points might vary slightly between executions. An illustration of the model learning steps are shown in Figure 3.

To recover $x(t)$ for the execution of primitive a_i from $\boldsymbol{x}^j$ we assume a nonlinear additive Gaussian noise regression model with diagonal covariance $\Sigma = \mathrm{diagonal}(\sigma_0^2, \sigma_1^2, \ldots)$,

$$\hat{x} = x(t) + \epsilon, \qquad \epsilon \sim \mathcal{N}(0, \Sigma),$$

and place a Gaussian process prior on the function x

$$x \sim \mathcal{GP}(m(\cdot), k(\cdot, \cdot)).$$

A Gaussian process [20] is a distribution over functions which has been highly successful in many statistical analysis and regression tasks, for example for motion pattern recognition [21]. It is a Bayesian non-parametric model which is very

suitable for modeling trajectories and trajectory-based motion patterns [22]. The Gaussian process is defined by mean function $m(t)$ and covariance function $k(t_1, t_2)$. By conditioning the GP on the observed trajectory $\boldsymbol{x}^j$ we get a predictive distribution where, for every time point t,

$$p(x|t, \boldsymbol{x}^j) = \mathcal{N}\big(x \,|\, \mu_{\boldsymbol{x}^j}(t),\, \Sigma_{\boldsymbol{x}^j}(t)\big), \tag{6}$$

where, over all state dimensions $d = 1, \ldots, n$,

$$\mu_{\boldsymbol{x}^j}(t) = [\ldots,\ \mu_{\boldsymbol{x}_d^j}(t), \ldots]^T, \tag{7}$$

$$\Sigma_{\boldsymbol{x}^j}(t) = \mathrm{diagonal}\big(\ldots,\ \sigma_{\boldsymbol{x}_d^j}^2(t), \ldots\big), \tag{8}$$

where, using a zero mean function $m(t) = 0$ since we can readily subtract the mean from the data,

$$\mu_{\boldsymbol{x}_d^j}(t) = \mathrm{K}(t, \boldsymbol{t}^j)^T \mathrm{V}_d^{-1}(\boldsymbol{x}_d^j), \tag{9}$$

$$\sigma_{\boldsymbol{x}_d^j}^2(t) = \mathrm{K}(t, t) + \sigma_d^2 - \mathrm{K}(t, \boldsymbol{t}^j)^T \mathrm{V}_d^{-1} \mathrm{K}(t, \boldsymbol{t}^j), \tag{10}$$

where $[\mathrm{K}]_{ab} = k(t_a, t_b)$, $\mathrm{V}_d = \mathrm{K}(\boldsymbol{t}^j, \boldsymbol{t}^j) + \sigma_d^2 \mathrm{I}_{\boldsymbol{t}^j}$, $\mathrm{I}_{\boldsymbol{t}^j}$ is an identity matrix. Since t is a scalar then $\mathrm{K}(t, t)$ is a scalar, $\mathrm{K}(t, \boldsymbol{t}^j)$ is a column vector and V_d a square matrix. Each output dimension is for simplicity treated as being independent which is equivalent to them being modeled by a separate function each with a separate Gaussian process prior.

Using (6) it is now possible to align all J_i triplet executions over the same time interval $[0, t_F^i]$. This set of executions spans the variety of primitive a_i and we expect any future *normal* execution of a_i to be similar to this set.

A motion primitive is a 1D curve through nD state space parameterized by time. We therefore consider a unimodal distribution over functions (like a single GP) a suitable model to represent the execution variability of a_i [22].

We formulate the *motion primitive execution model* as a unimodal distribution over functions

$$p(x|t, \boldsymbol{X}^i) = \mathcal{N}\big(x \,|\, \mu_i(t),\, \Sigma_i(t)\big) \tag{11}$$

where $\mu_i(t)$ and $\Sigma_i(t)$ are given, per dimension, by [22]

$$\mu_i(t) = \frac{1}{J_i} \sum_{m=1}^{J_i} \mu_{\boldsymbol{x}^j}(t), \tag{12}$$

$$\sigma_i^2(t) = \frac{1}{J_i} \sum_{m=1}^{J_i} \big(\sigma_{\boldsymbol{x}^j}^2(t) + \mu_{\boldsymbol{x}^j}(t)\big) - \mu_i^2(t),$$

which can be interpreted as the sample mean and sample variance from noisy samples with Gaussian-distributed noise. It is a *Gaussian approximation* to a mixture of Gaussian Processes, MoGPs. The MoGPs are based on the set of GPs all with the same weight (every triplet is observed once). The Gaussian approximation allows the MoGP to generalize outside of the individual executions. The motion primitive execution model is illustrated in the third figure to the right in Figure 3 and shown for 3D position in Figure 4 together with the motion primitive state trajectory and observed executions.
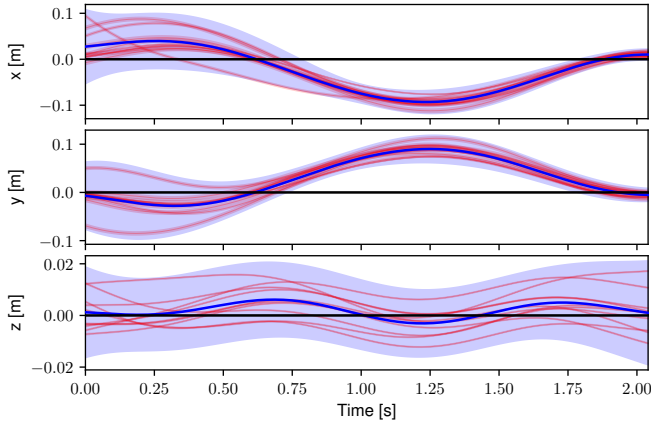
Fig. 4
Learned unimodal distribution for a motion primitive. The plot shows residuals w.r.t. the reference state trajectory of the motion primitive, i.e. $x_0^i(t)$ is subtracted. The red lines are the individual Gaussian processes for the observations. The unimodal distribution is defined by the blue region, displaying its 99% probability region, and the blue line is its mode.

B. Collision Checking

The learned motion primitive models represents the state of the robot when executing the motion primitive. Its mode is equally or more accurate, on average, than the motion primitive reference state trajectory under the assumption of Gaussian process noise. The execution variability is explicitly represented as a probability density over the state, over time.

For a stochastic variable x distributed according to a multivariate Gaussian distribution, $x \sim \mathcal{N}(\mu, \Sigma)$, the centered *probability region* (PR) with probability P is defined as [23]

$$\mathrm{PR}(\mathrm{P}, \mu, \Sigma) = \Big\{ x \mid (x - \mu)^T \Sigma^{-1}(x - \mu) \leq \chi_n^2(\mathrm{P}) \Big\},$$

where $\chi_n^2(\mathrm{P})$ is the chi-square quantile function with n degrees of freedom for probability P. For a diagonal $\Sigma = \mathrm{diagonal}(\dots, \sigma_d^2, \dots)$ it simplifies to a axis-aligned ellipsoid,

$$\mathrm{PR}(\mathrm{P}, \mu, \sigma^2) = \Big\{ x \mid \sum_d (x_d - \mu_d)^2 / \sigma_d^2 \leq \chi_n^2(\mathrm{P}) \Big\},$$

with axis lengths given by $\sqrt{\sigma_d^2 \chi_n^2(\mathrm{P})}$ for each axis d.

Let K be a motion primitive in a motion plan (4), corresponding to motion primitive $a_i \in \mathcal{A}$ and with start time $\mathcal{T}_K$. Let $\tau = t - \mathcal{T}_K$ which limits τ between 0 and t_F^i. The model $p(x|t, \boldsymbol{X}^i)$ of motion primitive a_i is a n-dimensional multivariate distribution with diagonal covariance matrix at each time-point τ. The safety margin $\mathcal{O}_{\mathrm{control}}$ can now be grounded probabilistically given probability P,

$$\mathcal{O}_{\mathrm{control}}(\tau) = \mathrm{PR}\big(\mathrm{P}, \mu_i(\tau), \sigma_i^2(\tau)\big) \tag{13}$$

C. Abnormality Detection

Once we have learned the motion primitive models, we want to use them to to detect abnormal executions of motion primitives. A probability region over the execution variability model of a primitive describes in which states we expect the

robot to be in for a given time point. For example [7] uses probability interval in execution monitoring for robot safety.

There are several possible ways to use the probability region to define what normal behavior is. A naive approach is to check whether the robot ever leaves the region, and if it does, define that execution as abnormal. However, since we pick the region to not contain all probability, we still expect the robot to leave the probability region occasionally.

More precisely, if we pick a 99% PR, the robot is expected to leave the interval in 1% of the time points. If the robot starts leaving the probability region more frequently, it might be an indication of an error. We define the rate of leaving the probability region as the failure rate, and model it as a stochastic variable. We consider the following parameters:

$$\theta - \text{Failure rate}$$
$$W - \text{Time window } t_W \text{ seconds into the past}$$
$$\#\text{normal}_W - \text{The number of normal observations}$$
$$\#\text{abnormal}_W - \text{The number of abnormal observations}$$
$$\mathrm{N}_W - \#\text{normal}_W + \#\text{abnormal}_W$$
$$\alpha - \text{Prior belief about failure rate}$$
$$\beta - \text{Prior belief about success rate}$$

We encode our prior assumptions about θ as a Beta distribution, setting the mode to $1 - \mathrm{P}$, where e.g. $\mathrm{P} = 99\%$, while still allowing for some uncertainty:

$$\theta \sim \mathrm{Beta}(\alpha, \beta),$$

with $\alpha = N\mathrm{P}$ and $\beta = N(1 - \mathrm{P})$, where N is the prior strength.

Given that we know θ, the number of observed failures follows a Binomial distribution:

$$\#\text{abnormal}_W \mid \mathrm{N}_W, \theta \sim \mathrm{Binomial}(\mathrm{N}_W, \theta)$$

As the Beta distribution is a conjugate prior for the Bionomial distribution, the posterior for θ is also Beta:

$$\theta \mid \#\text{abnormal}_W, \#\text{normal}_W$$
$$\sim \mathrm{Beta}(\alpha + \#\text{abnormal}_W, \beta + \#\text{normal}_W)$$

Using the posterior for θ, it is possible to evaluate the probability of θ being larger than expected:

$$p(\theta > (1 - \mathrm{P}) \mid \#\text{abnormal}_W, \#\text{normal}_W)$$
$$= 1 - \mathrm{Beta}_{\mathrm{CDF}}(1 - \mathrm{P}, \alpha + \#\text{abnormal}_W, \beta + \#\text{normal}_W)$$

where $\mathrm{Beta}_{\mathrm{CDF}}(x, a, b)$ is a constant-time standard function in most statistics libraries,

$$\mathrm{Beta}_{\mathrm{CDF}}(x, a, b) = \int_0^x q^{a-1}(1 - q)^{b-1} dq.$$

This probability of the failure rate being higher than $(1 - \mathrm{P})$ can then be monitored and thresholded. In this work we use the threshold of 99.9% probability and the prior strength of $N = 1000$, but these values can be fine-tuned to select a trade-off between precision and recall.

V. EXPERIMENTS AND RESULTS

In order to evaluate our proposed approach we consider a simulated DJI Matrice 100 quadcopter (Figure 5), a commonly used commercial research platform.
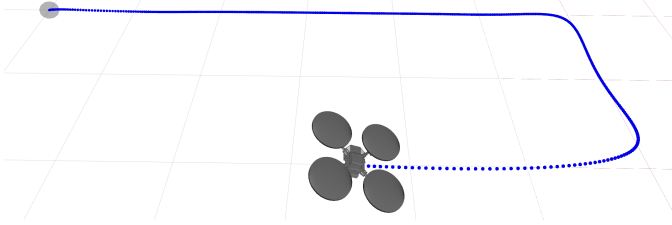

Fig. 5
Simulated DJI Matrice 100 quadcopter used in this work.

A. Motion Planning and Control

The same non-linear model for the DJI Matrice 100 as in [3] is used. We use a nonlinear MPC controller based on the work in [24] and use ACADO [25] to generate an efficient implementation that solves (5). A total of 104 motion primitives are generated using ACADO (Figure 2) using the same objective as the nonlinear MPC controller. Let $\mathcal{Z} = \left\{ (p,v) \,|\, v = p/\|p\|_2, \ p \in \{-1,0,1\}^3 \backslash \{\mathbf{0}\} \right\}$. The motion primitives have the initial and final states:

- From $x_I = (\mathbf{0}, \mathbf{0})$ to $x_F = (p, \mathbf{0})$, $\forall (p,v) \in \mathcal{Z}$
- From $x_I = (\mathbf{0}, \mathbf{0})$ to $x_F = (p, v)$, $\forall (p,v) \in \mathcal{Z}$
- From $x_I = (\mathbf{0}, v)$ to $x_F = (p, \mathbf{0})$, $\forall (p,v) \in \mathcal{Z}$
- From $x_I = (\mathbf{0}, v)$ to $x_F = (p, v)$, $\forall (p,v) \in \mathcal{Z}$

Running a sequence of triplet executions may cause one triplet execution to interfere with the next one. To avoid transient noise propagating across triplets, each triplet plan is augmented with actions before and after the triplet such that the full plan is always started from a state of rest.

B. Learning

For the motion primitive models, a Gaussian process prior with the squared exponential covariance function is used,

$$k(t_1, t_2) = \sigma_f^2 e^{\left(-\frac{1}{2}(t_1-t_2)\Lambda(t_1-t_2)^T\right)}, \quad (14)$$

where Λ is a diagonal matrix with length scales for each input dimension and σ_f^2 is the signal variance. These, together with the diagonal noise covariance Σ, are the hyper parameters $\theta = \{\sigma_f^2, \Sigma, \Lambda\}$ for this hierarchical Bayesian model.

We estimate the hyper-parameters from the data by maximizing the marginal log likelihood (empirical Bayes),

$$\log p(x|t, \theta) = -\frac{1}{2} x^T \mathbf{V}^{-1} x - \frac{1}{2} \log|\mathbf{V}| + C, \quad (15)$$

where $\mathbf{V}$ is defined as in (9-10) and C is a constant.

We investigate how close the predictive mean of the learned motion primitive execution model is to the observed trajectories, and compare it to how close motion primitive reference state trajectory is to the observations. In Table I the first row shows the RMSE between observed executions and the reference state trajectory or the executed motion primitive.

TABLE I

	RMSE (*meters*)
Observed Executions vs Motion Primitive	0.149 ± 0.053
Observed Executions vs $\mu_i(t)$	$\mathbf{0.055} \pm 0.043$
Motion Primitive vs $\mu_i(t)$	0.143 ± 0.052

The table shows several root-mean-square errors (RMSE). **Row 1:** The error between the motion primitive reference state trajectory and the observed executions of the same motion primitive. **Row 2:** The error between the mean prediction of the model of motion primitive execution and the observed executions of the same motion primitive. **Row 3:** The error between the primitive reference trajectory and the mean prediction of the model of executing that motion.
Only positions from the state are considered here. The RMSE mean and standard deviation is over all motion primitives.

The mean predictive is significantly more accurate as a point prediction for the execution trajectory than the reference state trajectory, as can be seen in the row below. The reason for this is that the reference state trajectory has a larger bias to the mean over execution trajectories (across state, not time) for a single primitives in comparison with the learned model. An example can be seen in Figure 4.

The mean predictive of the learned motion primitive model and the primitive's reference state trajectory are compared in the third row in Table I. It is observed that the RMSE between them is almost the same as the RMSE between the reference state trajectory and the observed executions. This provide additional evidence that the learned motion primitive model is a systematically better mean predictor for the execution, compared to the motion primitive itself.

C. Collision Checking

The standard approach in the literature is to use a sphere for $\mathcal{O}_{\text{safety}}(t)$ [15], capturing the aggregation of $\mathcal{O}_{\text{perception}}(t)$, $\mathcal{O}_{\text{control}}(t)$, $\mathcal{O}_{\text{others}}(t)$, with a single radius parameter invariant to time. We use this baseline, R, and construct two more with tighter bounds, R_i (different for each primitive) and $R_i(\tau)$ (also varying over time). For a meaningful comparison with our proposed variability model, we probabilistically ground the baselines by assuming a Gaussian likelihood over the observations centered on the primitive reference trajectory (i.e. relative translation is zero). For R_* in $\{R, R_i, R_i(\tau)\}$,

$$R_* \sim \mathcal{N}\left(0, \sigma_{R_*}^2\right), \quad (16)$$

where $\sigma_{R_*}^2$ for respective baselines are

$$\sigma_{R_i(\tau)}^2 = \frac{1}{J_i - 1} \sum_{j=1}^{J_i} \left(\mu_{\mathbf{X}^i}(\tau) - x_0^i(\tau)\right)^2 \quad (17)$$

$$\sigma_{R_i}^2 = \max_\tau \ \sigma_{R_i(\tau)}^2 \quad (18)$$

$$\sigma_R^2 = \max_i \ \sigma_{R_i}^2 \quad (19)$$

TABLE II

		Average normalized area (m^2)
R	(same for all primitives)	1.000
R_i	(individual for every primitive)	0.646
$R_i(t)$	(individual and time dependent)	0.329
$p(x\|t, \boldsymbol{X}^i),\ \forall i$		**0.109**

Average normalized area (mean $\pm\ 2.6\sigma$, equivalent to P = 99%), normalized with respect to R (using a single safety margin radius R for all dimensions and all motion primitives).

separate for each dimension d, and τ as defined in IV-B. The safety margin $\mathcal{O}_{\text{control}}$, given probability P is

$$\mathcal{O}_{\text{control}}(\tau) = \text{PR}(\text{P}, \boldsymbol{0}, \sigma^2_{R_*}) \qquad (20)$$

In Table II the baselines are compared with the learned motion primitive model. R is the largest 99% probability region over all primitives, per dimension, centered at the reference trajectory. R_i is specific to the individual primitives which allows it to be smaller. $R_i(t)$ is time variant, allowing a further radius reduction where the observations allows it.

Figure 6 show an example comparing R_i, $R_i(t)$ and the learned model $p(x|t, \boldsymbol{X}^i)$ for one motion primitive. Figure 7 show the dissection of Figure 6 at the green vertical line. A symmetric safety margin around a center that has a large bias with respect to the mean can be detrimental. To still ensure safety, e.g. that observed executions should still be within P = 99%, the margin must be larger with larger bias.

A symmetric safety margin is at its smallest when centered at the mean. Table II show that the area reduction is large using the learned variability model compared to the baseline. Even when comparing with a time-varying radius, $R_i(t)$, the area is reduced more than 3 times when using the learned model.
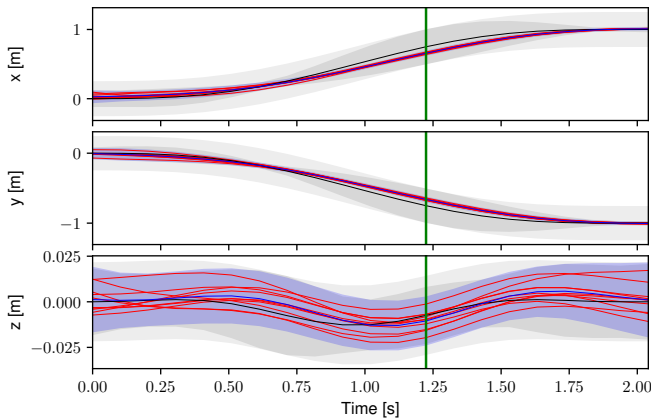

Fig. 6
A comparison of R_i (light gray envelope), $R_i(t)$ (dark gray envelope), $p(x|t, \boldsymbol{X}^i)$ (blue line and blue envelope) and observed executions (red) of a single motion primitive i (c.f. Fig. 4). R_i and $R_i(t)$ share the same black line as their mean (the reference state trajectory). R is not shown in order to make the finer details of the others visible. Envelopes are 2.6σ.
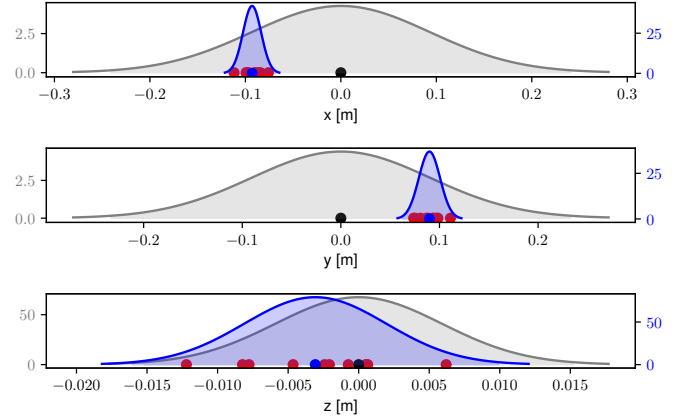

Fig. 7
Residual plots of dissection of Figure 6 at $t = 1.2$ (indicated with a vertical green line). $R_i(t)$ (black dot and dark gray envelope) and $p(x|t, \boldsymbol{X}^i)$ (blue dot and blue envelope) are depicted together with observed executions (red dots) of a single motion primitive i. R and R_i are not shown as to make the finer details of the others visible. Envelopes are 2.6σ.

D. Abnormality detection

For the experiment, a total of 104 motion primitives were used. Apart from the 26 primitives in Fig. 2 the other primitives have non-zero initial and/or final velocity. For each primitive 20 randomized but different valid triplets were executed and recorded. The first 10 triplets for each primitive were used for learning the motion primitive model for each primitive, i.e. a training set. The following 10 were used for testing the abnormality detection, i.e. the test set.

The naive approach of defining abnormal behavior as leaving the probability region is compared with the proposed approach using the Beta posterior. P = 99.9% is used.

For the Beta-method the prior parameters is set to $\alpha = 1$ and $\beta = 999$. The posterior is calculated including on the observations from the last second, meaning we set the time window $W = 1$ second. For each time point, we calculate the probability of the failure rate being larger than $(1 - \text{P}) = 0.01\%$. If the probability at any time point is larger than the threshold of 99.9%, the entire execution is classified as abnormal, else normal.

Both methods use the probability regions given from the learned motion primitive models derived from the training set and then tested on the test set. Figure 8 shows the confusion matrices for each combination of method and data set. The Beta-method performs slightly better than the naive method, while still maintaining perfect precision. This suggests that it might be possible to tweak the 99.9% threshold and the size of the time window to improve recall, without loss to precision.

VI. CONCLUSIONS AND FUTURE WORK

With increased autonomy of cyber-physical systems the need for integrated introspection capabilities is of growing importance. Such capabilities allow a robot to self monitor and

Fig. 8

Confusion matrix of abnormality detection over all primitives using a centered 99.9%-probability volume and (a, c) requiring that all observations fall inside this volume or (b, d) using a Beta posterior requiring the ratio of individual abnormalities less than 99.9% probable to be above 0.01% within a 1s window. **First row:** Performance on the model training set. **Second row:** Performance on previously unseen executions, from the same primitives, but different triplets.

to react to unexpected changes to circumstances in the environment. This is paramount if robots are supposed to operate safely in unstructured, dynamic and complex environments. We present an integrated approach for learning and monitoring the execution of motion actions, motion primitives, within the lattice-based motion planning paradigm.

The feasibility is demonstrated empirically using a nonlinear dynamical model of a well known quadcopter platform in simulation. We observe that the motion primitives can be bad predictors of actual motion execution and that our learned models of executions drastically improve both the point predictive performance and the necessary safety margin is much reduced. Both of these aspects are important for effective collision checking during motion planning.

Interesting future works is robust online learning of motion primitive execution models, as well as learning such models for different situations e.g. windy and non-windy conditions.

Consecutive failures are currently modeled as independent of each other in the abnormality detection. Removing this assumption will likely improve the performance on false alarms. The presented approach fills an important role in robot safety and the presented work enhances modern lattice-based motion planning methods with effective collision avoidance and monitoring of model correctness, making robots increasingly reliable and safe in real-world settings.

REFERENCES

[1] D. González, J. Pérez, V. Milanés, and F. Nashashibi, "A review of motion planning techniques for automated vehicles," *IEEE Transactions on Intelligent Transportation Systems*, vol. 17, no. 4, April 2016.

[2] C. Katrakazas, M. Quddus, W.-H. Chen, and L. Deka, "Real-time motion planning methods for autonomous on-road driving: State-of-the-art and future research directions," *Transportation Research Part C: Emerging Technologies*, vol. 60, pp. 416 – 442, 2015.

[3] O. Andersson, O. Ljungqvist, M. Tiger, D. Axehill, and F. Heintz, "Receding-horizon lattice-based motion planning with dynamic obstacle avoidance," in *IEEE Conference on Decision and Control*, 2018.

[4] R. Oliveira, M. Cirillo, J. M. artensson, and B. Wahlberg, "Combining lattice-based planning and path optimization in autonomous heavy duty vehicle applications," in *2018 IEEE Intelligent Vehicles Symposium (IV)*, June 2018, pp. 2090–2097.

[5] O. Ljungqvist, N. Evestedt, D. Axehill, M. Cirillo, and H. Pettersson, "A path planning and path-following control framework for a general 2-trailer with a car-like tractor," *Journal of Field Robotics*, vol. 36, no. 8, pp. 1345–1377, 2019.

[6] P. Doherty, J. Kvarnström, and F. Heintz, "A temporal logic-based planning and execution monitoring framework for unmanned aircraft systems," *Autonomous Agents and Multi-Agent Systems*, 2009.

[7] M. Tiger and F. Heintz, "Incremental reasoning in probabilistic signal temporal logic," *International Journal of Approximate Reasoning*, vol. 119, pp. 325 – 352, 2020.

[8] D. Park, Z. Erickson, T. Bhattacharjee, and C. C. Kemp, "Multimodal execution monitoring for anomaly detection during robot manipulation," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2016, pp. 407–414.

[9] A. Lederer, J. Umlauft, and S. Hirche, "Uniform error bounds for gaussian process regression with application to safe control," in *Advances in Neural Information Processing Systems*, 2019.

[10] S. M. LaValle and R. Sharma, "On motion planning in changing, partially predictable environments," *The International Journal of Robotics Research*, vol. 16, no. 6, pp. 775–805, 1997.

[11] B. Donald, P. Xavier, J. Canny, and J. Reif, "Kinodynamic motion planning," *Journal of the ACM (JACM)*, vol. 40, no. 5, 1993.

[12] K. Bergman and D. Axehill, "Combining homotopy methods and numerical optimal control to solve motion planning problems," in *2018 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 2018, pp. 347–354.

[13] B. Luders, M. Kothari, and J. How, "Chance constrained rrt for probabilistic robustness to environmental uncertainty," in *AIAA guidance, navigation, and control conference*, 2010, p. 8160.

[14] M. Pivtoraiko, R. A. Knepper, and A. Kelly, "Differentially constrained mobile robot motion planning in state lattices," *Journal of Field Robotics*, vol. 26, no. 3, pp. 308–333, 2009.

[15] M. Kamel, J. Alonso-Mora, R. Siegwart, and J. Nieto, "Robust collision avoidance for multiple micro aerial vehicles using nonlinear model predictive control," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Sept 2017, pp. 236–243.

[16] A. González-Sieira, M. Mucientes, and A. Bugarín, "Motion planning under uncertainty in graduated fidelity lattices," *Robotics and Autonomous Systems*, vol. 109, pp. 168–182, 2018.

[17] O. Andersson, M. Wzorek, P. Rudol, and P. Doherty, "Model-predictive control with stochastic collision avoidance using bayesian policy optimization," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2016, pp. 4597–4604.

[18] B. Zhou, F. Gao, L. Wang, C. Liu, and S. Shen, "Robust and efficient quadrotor trajectory generation for fast autonomous flight," *IEEE Robotics and Automation Letters*, vol. 4, no. 4, pp. 3529–3536, 2019.

[19] M. Ryll, J. Ware, J. Carter, and N. Roy, "Efficient trajectory planning for high speed flight in unknown environments," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2019, pp. 732–738.

[20] C. E. Rasmussen, "Gaussian processes for machine learning." MIT Press, 2006.

[21] K. Kim, D. Lee, and I. Essa, "Gaussian process regression flow for analysis of motion trajectories," in *Proc. ICCV*, 2011.

[22] M. Tiger and F. Heintz, "Online sparse gaussian process regression for trajectory modeling," in *Proc. Information Fusion (Fusion)*, 2015.

[23] V. Chew, "Confidence, prediction, and tolerance regions for the multivariate normal distribution," *Journal of the American Statistical Association*, vol. 61, no. 315, pp. 605–617, 1966.

[24] M. Kamel, T. Stastny, K. Alexis, and R. Siegwart, "Model predictive control for trajectory tracking of unmanned aerial vehicles using robot operating system," in *Robot Operating System (ROS) The Complete Reference, Volume 2*, A. Koubaa, Ed. Springer, 2017.

[25] B. Houska, H. Ferreau, and M. Diehl, "ACADO Toolkit – An Open Source Framework for Automatic Control and Dynamic Optimization," *Optimal Control Applications and Methods*, vol. 32, no. 3, 2011.