Receding-Horizon Lattice-based Motion Planning with Dynamic Obstacle Avoidance

Olov Andersson[1*], Oskar Ljungqvist[2*], Mattias Tiger[1*], Daniel Axehill[2], Fredrik Heintz[1]

***Abstract*— A key requirement of autonomous vehicles is the capability to safely navigate in their environment. However, outside of controlled environments, safe navigation is a very difficult problem. In particular, the real-world often contains both complex 3D structure, and dynamic obstacles such as people or other vehicles. Dynamic obstacles are particularly challenging, as a principled solution requires planning trajectories with regard to both vehicle dynamics, and the motion of the obstacles. Additionally, the real-time requirements imposed by obstacle motion, coupled with real-world computational limitations, make classical optimality and completeness guarantees difficult to satisfy. We present a unified optimization-based motion planning and control solution, that can navigate in the presence of both static and dynamic obstacles. By combining optimal and receding-horizon control, with temporal multi-resolution lattices, we can precompute optimal motion primitives, and allow real-time planning of physically-feasible trajectories in complex environments with dynamic obstacles. We demonstrate the framework by solving difficult indoor 3D quadcopter navigation scenarios, where it is necessary to plan in time. Including waiting on, and taking detours around, the motions of other people and quadcopters.**

I. Introduction

Safe navigation for autonomous vehicles is an area under intense research. As automotive companies are making strides towards full autonomy in structured street environments, unmanned aerial vehicles (UAVs) such as quadcopters are also increasingly being looked towards for autonomous inspection, monitoring, search, and even delivery tasks. To efficiently solve such tasks in unstructured environments, often requires the capability to both safely navigate in static environments, while at the same time taking into account other moving agents in the area. Such dynamic obstacles may include, e.g., ground vehicles, UAVs and even people.

This is a difficult motion planning problem, where a principled solution requires planning over time, with regard to both vehicle and obstacle dynamics. Additionally, moving obstacles will impose real-time constraints on planning, while real-world autonomous vehicles have computational limitations.

We propose a principled solution to this problem by using a unified optimization-based motion planning and control architecture, where both layers use the system dynamics to generate and execute feasible trajectories in real-time. In particular, we present a novel receding-horizon motion planner which has similarities to multi-resolution state lattices [1], [2]. Motion primitives, a set of dynamically feasible trajectories, are generated offline by the use of numerical optimal control. These are then used in an online graph-search to find a dynamically-feasible and cost efficient solution to the dynamic motion planning problem. Chosen motion primitives are efficiently tracked by a receding-horizon controller.

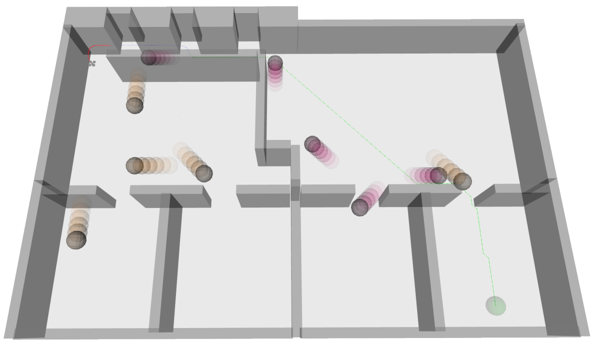
Fig. 1: Example indoor 3D warehouse scenario with both static and dynamic obstacles, including humans (red) on the ground, and other quadcopters (orange) flying at varying altitudes. The faded spheres are predictions of future motion.

As benchmark domain we use a challenging class of indoor quadcopter 3D navigation scenarios seen in Fig. 1, populated by both people and other vehicles. To the authors' knowledge, this remains largely an unsolved problem.

Planning with respect to time is essential to be able to wait for, or move out of the way of, moving obstacles. However, by discretizing the state space and time, and generating optimal motion primitives with boundary constraints on both states, controls and time will interfere with minimum cost objectives in the motion planner. To circumvent this issue we instead propose an augmented multi-resolution state and time lattice where a wait-time state takes the place of time. The wait-time state allows the vehicle to remain in equilibrium points for short time durations, e.g. hovering for a quadcopter or zero longitudinal velocity for a ground vehicle. To manage the complexity of graph search in high-dimensional lattices, we formalize a framework for multi-resolution lattice planning by drawing on similarities to receding-horizon control. In this context, the cost-to-go is approximated by a finalizing search in a graph with low-resolution and moving obstacles can be neglected.

We refer to our motion planning approach as receding-horizon lattice planning to stress that the motion planner is operating in a receding-horizon fashion, where the cost-to-go is estimated in a finalizing search in a low-resolution state lattice. The efficacy of the proposed approach is illustrated

[1]Division of Artificial Intelligence and Integrated Computer Systems, Linköping University, Sweden, (e-mail: (olov.a.andersson, mattias.tiger, fredrik.heintz)@liu.se)

[2]Division of Automatic Control, Linköping University, Sweden, (e-mail: (oskar.ljungqvist, daniel.axehill)@liu.se)

[*]These authors contributed equally to this work.

on difficult quadcopter navigation tasks in dynamic 3D indoor environments, using simulations of a commonly used quadcopter research platform.

A. Related Work

Motion planning with moving obstacles is a difficult problem as the robot needs to plan trajectories with regard to both its own motion, and that of the obstacles. While simpler avoidance behaviors exist, e.g. velocity obstacles [3], to efficiently navigate a cluttered corridor with moving obstacles actually requires planning around obstacles. This would suggest an optimization-based approach, where at each point in time the predicted positions of obstacle geometry are included as constraints on the feasible trajectory. Such obstacle geometry make the feasible set non-convex, which poses a significant challenge under the real-time requirements of autonomous robots.

By using receding-horizon control formulations, e.g., Model Predictive Control (MPC), it has previously been demonstrated that finding good local solutions is possible in open environments with a small number of obstacles [4]. Unfortunately, including geometric constraints for complex indoor environments, with both static and dynamic obstacles, makes the optimization problem prohibitively expensive to solve in real-time. This stems both from the number of added obstacle constraints, increasing the iteration cost of MPC solvers, and the extra non-convexity making it hard to find good local solutions. Our proposed approach avoids geometric constraints in the control layer by leaving collision checking and obstacle avoidance to the motion planner.

On the quadcopter control side, with the surging interest in quadcopters, fast trajectory planning methods tailored to quadcopter dynamics have also enjoyed increased attention [5], [6], [7]. A popular trick is to exploit differential flatness to be able to perform the trajectory generation in the flat outputs, i.e. the position and yaw angle of the quadcopter [6]. The trajectory generation problem then boils down to finding piece-wise higher-order polynomial spline in the flat outputs, while satisfying boundary conditions and minimizing a performance measure, e.g. minimal jerk or snap [5]. These methods are faster and scale better in planning horizon compared to general-purpose MPC solvers, but including obstacles still requires an outer optimization procedure and does not address the obstacle avoidance problem, which is typically handled by geometric motion planning [6], [7]. Furthermore, including time and collision constraints to handle moving obstacles is an open problem.

Motion planning using state lattices originates from the ground vehicle domain [8], [9]. The lattice framework relies on precomputation of a large set of feasible motions connecting the nodes. The motion planning problem can then be solved online by applying efficient graph search methods on the motion primitives. They have also been used with randomized tree-search approaches [10], however these generally require a closed-form controller to efficiently connect the nodes.

Lattice-based motion planning framework also been extended to plan in both structured [11] and unstructured dynamic environments [1]. It has also been used off-road in [12], which employs a multi-resolution state lattice with a lattice graph of high resolution in the vicinity of the vehicle, but time and dynamical obstacles are not considered.

In [1], time is included in the discretization of the lattice graph, however, rather than using an optimization-based approach for motion primitive generation, the motion primitives are generated via constant control signals and a massive number of simulations of the system. This method for generating motion primitives is not suitable for systems with complex dynamics, especially not unstable systems. As in [13] we use an optimization-based approach to generate optimal motion primitives. Numerical optimal control [14] is utilized in order to satisfy the system dynamics, physically imposed constraints, ensure smooth control signals, and minimize a desired performance measure.

Recently, lattice-based motion planning for quadcopters where motion primitives are generated offline via differentially flatness have also been suggested [15], [16], however, these do not include moving obstacles or time. The former found that their state lattice approach outperformed the popular sampling-based Rapidly-exploring Random Tree (RRT) algorithm [17].

The remainder of the paper is structured as follows. In Section II the receding-horizon motion planning problem is formally explained and in Section III the receding-horizon lattice planner is presented. Finally, in Section IV the framework is applied to a quadcopter with a trajectory tracking MPC controller, together with simulation results on challenging dynamic 3D indoor navigation scenarios.

II. Problem formulation

Consider a vehicle that is modeled as a time-invariant nonlinear system

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

where $x(t) \in \mathbb{R}^n$ denotes the vehicle's states and $u(t) \in \mathbb{R}^m$ its control signals. The vehicle is assumed to have physically imposed constraints on its states $x(t) \in \mathcal{X}$ and control signals $u(t) \in \mathcal{U}$. Furthermore, the vehicle is assumed to operate in a 2D or 3D-world $\mathcal{W}(t)$ where both static and moving obstacles exist. The regions which are occupied with obstacles $\mathcal{O}_{obs}(t)$ are separated into static obstacles $\mathcal{O}_{s,obs}$ and dynamic obstacles $\mathcal{O}_{d,obs}(t)$. The free-space where the vehicle is not in collision with any obstacle at time t is defined as $\mathcal{X}_{\text{free}}(t) = \mathcal{X} \setminus \mathcal{O}_{obs}(t)$. The objective of the motion planner is to generate a feasible and collision-free reference trajectory $(x_0(t), u_0(t))$, $t \in [t_I, t_G]$ that moves the vehicle from its current position x_I to a desired goal position x_G, while optimizing a given performance measure J_D, e.g., minimum time, minimum energy or maximum smoothness. Define the Dynamic Motion Planning Problem (DMPP) as

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

which is a nonlinear optimal control problem that is most often intractable to even find a feasible solution to, and motion planning algorithms are instead utilized in order to achieve real-time performance [17].

An optimal solution to the DMPP in (2) is not only hard to find, there is a high probability that the solution will become infeasible during the trajectory execution due to the fact that the longterm motion of moving obstacles are hard to predict exactly. In order to safely navigate close to dynamic obstacles, the motion planner needs to be reactive to unpredicted motions of other agents and replan at a sufficiently high rate, while also keeping the longterm goal in mind. In order to achieve these two sometimes competing goals we deploy a receding-horizon search-based motion planning framework.

A. *Receding-horizon motion planning*

Similarly to receding horizon control, the DMPP in (2) is relaxed by replacing the fixed final constraint $x_0(t_G) = x_G$ in (2) with a terminal cost $\Phi(x_0(t_H)), x_G, t_H)$, and the motion planning problem is solved in a receding horizon fashion. Define the Receding Horizon Motion Planning Problem (RHMPP) as

$$\begin{aligned}
\underset{u_0(\cdot),\, t_H}{\text{minimize}} \quad & J_H = \Phi(x_0(t_H)), x_G, t_H) + \int_{t_I}^{t_H} L(x_0(t), u_0(t), t)\, dt \\
\text{subject to} \quad & \dot{x}_0(t) = f(x_0(t), u_0(t)), \\
& x_0(t_I) = x_I, \\
& x_0(t) \in \mathscr{X}_{\text{free}}(t), \ \forall t \in [t_I, t_H] \\
& u_0(t) \in \mathcal{U}, \ \forall t \in [t_I, t_H]
\end{aligned} \tag{3}$$

where t_H denotes the end of the planning horizon and is an optimization variable. As in receding horizon control, the terminal cost, i.e. cost-to-go, plays a key role and is intended to estimate the remaining cost to the goal x_G from the state at the end of planning horizon $x_0(t_H)$, i.e.,

$$\Phi(x_0(t_H)), x_G, t_H) \approx \int_{t_H}^{t_G} L(x_0(t), u_0(t), t)\, dt \tag{4}$$

However, estimating the cost-to-go (4) is not a simple task, especially not in maze-like environments where there are plenty of different routes to the goal.

Furthermore, if the planning horizon is short and the terminal cost is severely underestimated, there is a high risk that the motion planner chooses a trajectory such that the vehicle ends up in a bad local minimum, e.g., an alley, from which it is incapable of recovering from. An example is illustrated in Fig. 2. In the top figure, a bad local minimum to the planning problem is found since the Euclidean distance from the position of the vehicle at t_H to the goal is used as terminal cost.

The RHMPP in (3) can be separated into two distinct subproblems. The first subproblem is to find dynamically feasible trajectories towards the goal that are safe, e.g. does not collide with any static nor dynamic obstacle. The second part of the problem is to calculate the cost-to-go (4) in a systematic way. Choosing between the possible solutions such that it reflects the cost-to-go, and thus avoiding poor longterm trajectory selections. For this purpose, collisions with moving obstacles are also not as important as static obstacles, since motions further in the future are harder to predict, and a real vehicle has limited sensor range. In the next section, a principle solution to the RHMPP in (3) is proposed.

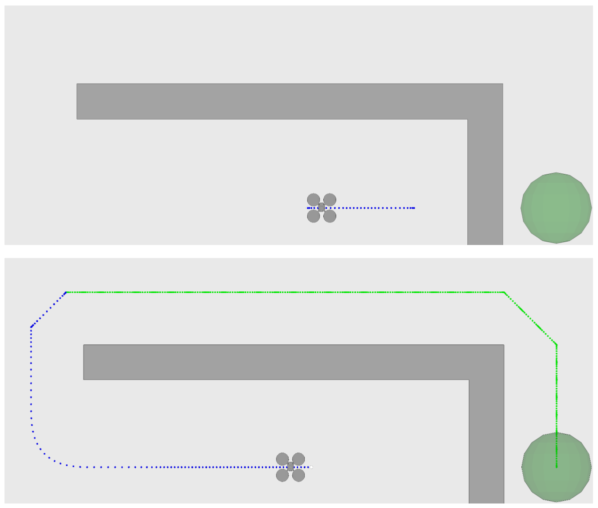
Fig. 2: Examples of a bad (top) and a good (bottom) solution to the receding-horizon motion planning problem for a quadcopter that is operating in an indoor environment. The top figure is when relevant terminal cost (cost-to-go) is underestimated, and the bottom figure when the terminal cost is calculated in a systematic way.

III. Receding-horizon lattice planner

This section presents an extension of the basic state-lattice motion planning framework [8] to also handle dynamic obstacles in cluttered environments, e.g. other vehicles or people. This is done by augmenting the state-lattice graph with an additional wait-time state, allowing a previously explored state to be visited multiple times. The standard state-lattice motion planning framework [8] is resolution optimal and resolution complete when an admissible heuristic function is used during online planning, and no time constraint is imposed on the search process. However, since temporal planning with dynamic obstacles is a computationally heavy task that has real-time requirements, we plan in a receding-horizon fashion. This sacrifices resolution optimality and completeness guarantees for real-time performance and safety.

The framework is based on the multi-resolution state-lattice framework [2], where a graph with both high resolution and connectivity is searched through during a primary planning phase, in the vicinity of the vehicle. Later a secondary search is performed with a graph of lower resolution with less connectivity, as illustrated in the lower picture in Fig. 2. The secondary search is used to estimate the remaining cost to the goal, and in order to make this search extremely fast, only static obstacles are collision checked.

A. *Temporal state-lattice planning*

The first and primary part of the planning cycle uses an augmented version of the state lattice motion planning framework [8]. This augmented lattice graph includes an extended state vector $x_t = (x, w_t)$, where a wait-time state $w_t \in \mathbb{Z}_+$ is included in the representation of the graph. The wait time w_t is a positive integer that can be viewed as a non-decreasing counter. Similarly to state-lattices [8], the temporal state-lattice planner searches for a resolution optimal solution to the DMPP in (2) by discretizing the state-space of the vehicle in a regular fashion, and constraining the motion of the vehicle to a lattice graph $\mathcal{G} = \langle \mathcal{V}, \mathcal{E} \rangle$. Each vertex $v \in \mathcal{V}$ denotes a discrete wait-time enhanced state $x_{t,d} = (x_d, w_t)$, where the subscript d denotes that the states are discrete, and each edge $e \in \mathcal{E}$ encodes a motion which respects the system dynamics (1) and its physically imposed constraints.

In this work, the wait-time counter w_t will be counted up by one if a "wait action" is used, and this is only allowed to be used if the vehicle is standing in an equilibrium point. This results in a state trajectory where the vehicle state remains fixed and the wait time w_t is counted up by one. Unlike normal state-lattices, this augmented lattice graph allows the vehicle to wait in the same spot for a certain time duration, or revisit previously explored vehicle states, since the extended state vector $x_{t,d}$ can differ even though the vehicle states x_d are identical. This flexibility is crucial for planning safe and collision free trajectories to avoid moving obstacles in congested areas of the environment. Moreover, compared to discretizing the time in the lattice graph [2], the time duration of a motion primitive is a free parameter and when, e.g., minimum time problems are considered, the time duration of a motion primitive can be an optimization variable in the motion primitive generation.

B. Lattice creation

Moreover, the discretization of the lattice graph defines which discrete wait-time enhanced states the vehicle can reach $x_{t,d} \in \mathcal{X}_d \times \mathbb{Z}_+$, and the allowed motions of the vehicle are encoded in the motion primitive set $\mathcal{P}$. The size of the motion primitive set is M, i.e., $|\mathcal{P}| = M$, which is a finite number of possible transitions from one discrete state to neighboring states in a bounded neighborhood in free space, and thus determines the connectivity of the graph. The procedure of selecting which states to connect is application dependent and is typically performed by a system expert [13]. As in earlier state lattice planning frameworks [8], we exploit assumptions of position and time-invariance of the vehicle dynamics when generating the motion primitives $p_i \in \mathcal{P}$.

The motion primitives are generated offline by solving a finite number of Two-Point Boundary Value Problems (TPBVPs) to connect a discrete set of initial states $x_{d,i} \in \mathcal{X}_d$ to a discrete set of neighboring states $x_{d,f} \in \mathcal{X}_d$ in a bounded neighborhood in free space. The TPBVP solver guarantees that the motion primitives respect the system dynamics and its physically imposed constraints. Moreover, the time and position-invariance properties guarantee that a motion primitive $p_i \in \mathcal{P}$ is translatable to other discrete states in the graph. A motion primitive $p_i \in \mathcal{P}$ is a trajectory $(x_0^i(t), u_0^i(t))$, $t \in [0, t_f^i]$ that satisfies the following properties:

$$\dot{x}_0^i(t) = f(x_0^i(t), u_0^i(t)) \tag{5a}$$

$$x_0^i(0) = x_s^i \in \mathcal{X}_d, \quad x_0^i(t_f^i) = x_f^i \in \mathcal{X}_d \tag{5b}$$

$$x_0(t) \in \mathcal{X}, \quad u_0(t) \in \mathcal{U}, \quad \forall t \in [0, t_f^i] \tag{5c}$$

where x_s^i and x_f^i denote the initial and final state of the vehicle, respectively. An example of a motion primitive set for a quadcopter from an initial state x_s^i where the vehicle is hovering to neighboring states $x_f^i \in X_d$ can be seen in Fig. 3.

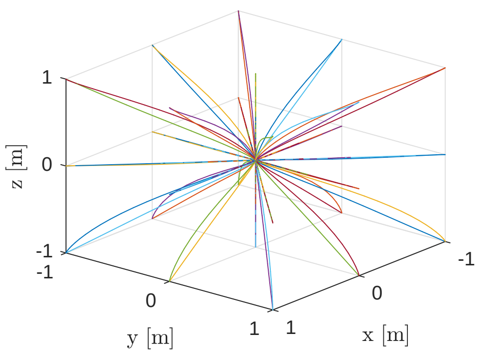

Fig. 3: Motion primitives for a quadcopter from where the vehicle is hovering to different neighboring states on the high resolution lattice grid that is used during the temporal planning phase.

Similarly to [13] the motion primitives $p_i \in \mathcal{P}$ are generated offline using numerical optimal control [14]. However, instead of using different objective functions during the motion primitive generation and online planning, the same objective function is used and each motion primitive $p_i \in \mathcal{P}$ is assigned the resulting objective function value $J_{D,i}$.

C. Online planning

During online planning, a suboptimal solution to the DMPP in (2) is searched for using dynamic programming, where efficient graph-search algorithms, s.a. A* with an admissible heuristic, can be used. Since the system dynamics and its physically imposed constraints have been taken care of during the motion primitive generation, what remains during online planning is to find the best sequence of motion primitives that together generate a trajectory $(x_0(t), u_0(t))$, $t \in [t_I, t_G]$ that does not collide with any obstacle in $\mathcal{O}_{obs}(t)$, and moves the vehicle from its current position x_I towards the desired goal x_G. However, in order to be able to quickly react to changes in the environment, the temporal planning time is bounded to be less than t_T seconds, e.g., half of the overall planning time. If a solution to the goal has been found within its allowed time slot, the resulting trajectory is sent for trajectory execution.

Otherwise, the DMMP in (2) is solved in a receding horizon fashion (3) and a secondary search towards to goal is initiated. The secondary planning phase is intended to estimate the remaining cost-to-go $\Phi(x_0(t_H)), x_G, t_H)$ from the frontier of explored states in the lattice graph that satisfies a certain minimum plan duration criteria $t_{\min}$, i.e., $t_H \geq t_{\min}$ in (3). The minimum plan duration constraint is included in order to guarantee that the vehicle does not collide with any moving obstacle $t_{\min}$ seconds into the future, such that a trajectory tracking controller can safely start the execution of the partly computed plan towards the goal. Note that $t_{\min}$ is intuitive to tune based on the sensor range of the vehicle in a practical application.

D. Estimation of cost-to-go

The secondary planning phase continues the search for a plan towards to goal and is designed to be very fast, and therefore only considers static obstacles. It is intended to act as an estimation of the cost-to-go $\Phi(x_0(t_H)), x_G, t_H)$ in (4) for the set of collision free plans that satisfy the minimum time duration criteria that have been found during the computationally intense temporal planning phase. In the secondary planning phase, the resolution of the lattice graph is reduced and the introduced wait-time state w_t is neglected.

Moreover, the motion primitive set $\mathcal{P}$ is thus also significantly reduced to $\mathcal{P}_{red} \subset \mathcal{P}$ such that the lattice planner is still able to find a feasible plan to reach the desired goal position of the vehicle with short computation time. Estimation of terminal cost is critical such that the motion planner does not choose a trajectory that lead to a bad local minimum, as in Fig. 2. When a solution to the goal is found, the temporal part of the best plan is sent for trajectory execution, and the motion planner keeps continuously replanning the search for a complete dynamically-feasible and collision-free plan towards the goal.

IV. Results

To evaluate the performance of the proposed receding-horizon lattice-based motion planning framework, it is evaluated on trajectory planning for a quadcopter in dynamic 3D indoor environments. All modules that are presented here are implemented in C++ using ROS [18]. We assume that a task planner is feeding the motion planner with a fly-to command, where the desired goal x_G is assumed to be an equilibrium point, i.e., the quadcopter is hovering. The motion planning and control architecture is schematically illustrated in Fig. 4 and will now be explained in detail.

A. Quadcopter model

We use a model of the DJI Matrice 100, a common commercial quadcopter research platform[1]. Denote the coordinates of the center of gravity for the quadcopter in the world frame $\mathbb{W}$ as $\mathbf{p} = (x, y, z) \in \mathbb{R}^3$ and its velocities as $\mathbf{v} = (v_x, v_y, v_z) \in \mathbb{R}^3$. Moreover, let ϕ, θ and ψ denote the roll, pitch and yaw angle of the vehicle, respectively. The attitude angles are assumed to be controlled by low-level attitude controllers that are able to track desired attitude angles, ϕ_{ref} and θ_{ref}, with a first-order system behavior. These first-order systems can be identified using classical system identification techniques as described in [19].

By defining $R(\phi, \theta, \psi)$ as the rotation matrix from the body frame $\mathbb{B}$ of the vehicle to the world frame $\mathbb{W}$, the quadcopter can be modeled as:

$$\dot{\mathbf{p}}(t) = \mathbf{v}(t) \tag{6a}$$

$$\dot{\mathbf{v}}(t) = R(\phi, \theta, \psi)\begin{bmatrix}0\\0\\T\end{bmatrix} - K_{\text{drag}}\mathbf{v}(t) + \begin{bmatrix}0\\0\\-g\end{bmatrix} \tag{6b}$$

$$\dot{\phi}(t) = \frac{1}{\tau_\phi}(K_\phi \phi_{\text{ref}}(t) - \phi(t)) \tag{6c}$$

$$\dot{\theta}(t) = \frac{1}{\tau_\theta}(K_\theta \theta_{\text{ref}}(t) - \theta(t)) \tag{6d}$$

$$\dot{\psi}(t) = 0 \tag{6e}$$

[1]www.dji.com/matrice100

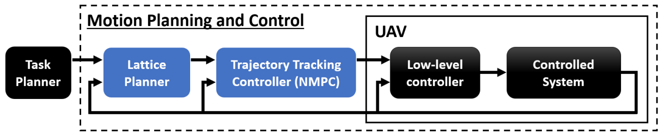

Fig. 4: Illustration of the motion planning and control architecture that is used for the quadcopter platform. The modules that are colored in blue are considered in this work.

where g is the gravitational acceleration, T is the mass normalized thrust, $K_{\text{drag}} = \text{diag}(A_x, A_y, A_z)$ is the mass normalized drag coefficient matrix, τ_ϕ, K_ϕ and τ_θ, K_θ are the time constant and gain of the first-order system behavior of the inner loop for roll and pitch, respectively.

It is assumed that the yaw angle ψ of the vehicle is controlled such that it is approximately zero. Define the state vector as $x(t) = (\mathbf{p}, \mathbf{v}, \phi, \theta, \psi)$, the control signals as $u(t) = (\phi_{\text{ref}}, \theta_{\text{ref}}, T)$ and represent (6) as $\dot{x}(t) = f(x(t), u(t))$. The vehicle has physical limitations in thrust, and it is unsafe to fly with too large attitude angles. These limitations are formulated as constraints on the control signals:

$$T_{\min} \le T(t) \le T_{\max} \tag{7a}$$

$$|\phi_{\text{ref}}(t)| \le \phi_{\max} \tag{7b}$$

$$|\theta_{\text{ref}}(t)| \le \theta_{\max} \tag{7c}$$

Represent the constraints in (7) as $u \in \mathcal{U}$. These constraints have to be taken into account during motion primitive generation and will also be embedded in the MPC controller used for trajectory tracking.

B. Receding-horizon lattice planner

To generate the receding-horizon lattice planner, the state space of the vehicle has to be discretized. In the primary temporal planning phase, the allowed states of the vehicle are chosen such that $(\phi_d, \theta_d, \psi_d) = (0, 0, 0)$ and $u_d = (0, 0, g)$. This means that the acceleration of the vehicle is approximately zero at each vertex in the lattice graph. Only the position $\mathbf{p}_d$, the velocity $\mathbf{v}_d$ and the wait-time state w_t can change between each vertex in the graph. Thus, at a valid wait-time enhanced state in the lattice graph, the vehicle can be represented by a seven-dimensional vector $\mathbf{s} = (\mathbf{p}_d, \mathbf{v}_d, w_t)$. The position of the vehicle $\mathbf{p}_d$ is given a grid resolution $r = 0.5$m in all three directions, and the velocity is discretized as $\mathbf{v}_d \in \mathbb{V}$ which is a finite set of triples where the Euclidean norm of the velocity vector is either 0, 1 or 2. The wait-time state is a positive integer that can only be used when the quadcopter is hovering.

In the secondary planning phase, the velocity $\mathbf{v}_d$ of the vehicle is constrained to zero and the wait-time state is ignored. Thus, in the secondary planning phase, a vertex in the low-resolution lattice graph is represented only by the position $\mathbf{p}_d$ of the vehicle, with the same resolution as in the temporal planning phase.

1) Motion primitive generation: In a similar fashion as in [13], the motion primitive sets for the primary temporal $\mathcal{P}$ and secondary $\mathcal{P}_{\text{red}} \subset \mathcal{P}$ planning phases are generated offline using numerical optimal control. In order to generate physically feasible trajectories for the quadcopter, the model in (6), the physical constraints in (7) and additional constraints on the first and second order derivatives of the control signals are added to the optimal control problem. The objective function J_D that is used for both motion primitive generation and during online motion planning is

$$J_D = t_f + \int_0^{t_f} ||(u,\dot{u},\ddot{u})||_{\mathbf{Q}}^2 dt \tag{8}$$

where the block-diagonal matrix $\mathbf{Q} \in \mathbb{S}_+^9$ is a design parameter that acts as a weight between smoothness and plan duration. Formally, the optimal control problem that is used for motion primitive generation is

$$\begin{aligned} \underset{\ddot{u}(\cdot),t_f}{\text{minimize}} \quad & J_D \\ \text{subject to} \quad & \dot{x}(t) = f(x(t),u(t)), \\ & x(0) = x_i^d, \;\; x(t_f) = x_f^d, \\ & u(0) = u_i^d, \;\; u(t_f) = u_f^d, \\ & \dot{u}(0) = \dot{u}(t_f) = 0, \\ & |\dot{u}(t)| \leq \dot{u}_{\max}, \;\; |\ddot{u}(t)| \leq \ddot{u}_{\max}, \\ & u \in \mathbb{U}. \end{aligned} \tag{9}$$

Since the time duration t_f is not discretized in the lattice graph, it can be a continuous optimization variable in the motion primitive generation, which is crucial when generating optimal motion primitives for minimal time problems. The initial state x_i^d and the final state x_f^d in (9) are fixed design parameters that have to obey the chosen discretization of the lattice graph. We use the open source solver ACADO Toolkit [14] to solve the OCP in (9) and generate the motion primitive set $\mathcal{P}$ which is a procedure that is done offline and can easily be parallelized. The size of the motion primitive set $|\mathcal{P}| = M$ and the boundary constraints in (9) for each motion primitive $p_i \in \mathcal{P}$ are manually specified such that:

- Each discrete state $\mathbf{s}$ is reachable in the graph
- Each resulting maneuver is a smooth and intuitive motion
- The online graph search problem remains tractable to solve in real-time

As illustration, all possible motion primitives from $\mathbf{s}_i = (0,0,0)$ to different neighboring states on the lattice grid for the temporal planning phase are presented in Fig. 3 and in Fig. 5, all motion primitives from $\mathbf{s}_i = (0,\mathbf{v}_{d,i}(0),0)$ for different initial velocities $v_{d,i}(0)$ are visualized. The size of the motion primitive set dedicated to the primary temporal planning phase is $|\mathcal{P}| = 282$. In each vertex $\mathbf{s}$, the number of possible motion primitives vary between 8 and 86. Moreover, in a valid grid point in the secondary planning phase, the number of motion primitives are $|\mathcal{P}_{\text{red}}| = 10$ where eight of the motion primitives achieve full connectivity to its closest neighbors in the horizontal plane and two are vertical movements.

2) Online planning: During online planning, the primary temporal planning phase initiate the search for a solution to the dynamic motion planning problem in (3) by searching through the temporal high-resolution lattice graph using the A* graph search algorithm with the admissible heuristic function

$$H(\mathbf{s}) = \frac{1}{v_{\max}} ||\mathbf{p}_d - \mathbf{p}_G||_2 \tag{10}$$

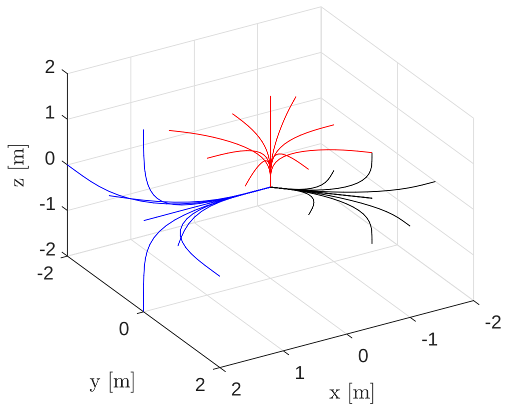

Fig. 5: High resolution motion primitives from $\mathbf{s}_i = (0,\mathbf{v}_{d,i}(0),0)$ to different states on the lattice grid. The blue, red and black trajectories are from $\mathbf{v}_{d,i}(0) = (0,-2,0)$, $\mathbf{v}_{d,i}(0) = (0,0,1)$ and $\mathbf{v}_{d,i}(0) = (0,1/\sqrt{2},1/\sqrt{2})$, respectively.

which is the Euclidean distance to the goal $\mathbf{p}_G$ from the current position of the vehicle $\mathbf{p}_d$ divided by the maximum average speed $v_{\max}$ of all motion primitives $p_i \in \mathcal{P}$. This heuristic is admissible since it is an underestimate of the remaining plan duration to the goal, and hence it is an underestimate of the remaining cost-to-go. If the temporal planning phase is unable to find a complete plan to the goal, within its allowed time budget, the dynamic motion planning problem (2) is solved in a receding horizon fashion (3) and the secondary planning phase continues the search in the low-resolution lattice graph from vertices that satisfy the minimal time duration criteria $t_H \geq t_{\min} = 5$s. In this planning phase, the same heuristic function as in (10) is used, but the maximum average speed $v_{\max}$ is of all motion primitives $p_i \in \mathcal{P}_{\text{red}}$. When a solution is found, the temporal part of the trajectory is sent for trajectory execution.

C. Trajectory Tracking MPC controller

A nonlinear MPC controller that is based on the work in [19] is used for trajectory tracking of the reference trajectory $(x_0(t),u_0(t))$, $t \in [t_I,t_H]$ calculated by the receding horizon lattice planner. The objective of the nonlinear MPC controller is to track the desired reference trajectory with a small tracking error $\bar{x} = x(t) - x_0(t)$, while not deviating too far from the feed-forward control signal $u_0(t)$. The continuous-time nonlinear MPC problem is formulated as

$$\begin{aligned} \underset{u(\cdot)}{\text{minimize}} \quad & ||\bar{x}(T)||_{\mathbf{P}_N}^2 + \int_0^T \left(||\bar{x}(t)||_{\mathbf{R}_1}^2 + ||u(t) - u_0(t)||_{\mathbf{R}_2}^2 \right) dt \\ \text{subject to} \quad & \dot{x}(t) = f(x(t),u(t)), \\ & u(t) \in \mathbb{U}, \\ & x(0) = x(t_0) \end{aligned} \tag{11}$$

where $\mathbf{R}_1$, $\mathbf{R}_2$ and $\mathbf{P}_N$ are positive-definite weight matrices that are design parameters. The prediction horizon is $T = 4$s and the terminal cost matrix $\mathbf{P}_N$ is chosen as the solution to the Continuous-time Algebraic Riccati Equation (CARE) for the linearizion of (6) around hovering. The ACADO code generation tool [20] is used to automatically generate

C-code for a highly efficient discrete-time nonlinear MPC controller with the specified sampling time $T_s = 0.1$s. The generated code solves a sequence of quadratic programs using qpOASES [21], an active-set solver with warm-starts.

D. Simulation results

Here we examine simulation results of the proposed optimization-based receding-horizon lattice planner on challenging scenarios with moving obstacles, using the DJI Matrice 100 quadcopter model in (6). In the experiments, we run the receding-horizon planner with a maximal planning time of 1s, where the primary planning phase was allocated $t_T = 0.33$s. We first demonstrate the usefulness and capability of our approach to plan in time.

Consider the example in Fig. 6, a moving obstacle moves right-to-left in a narrow corridor with a small cranny. The quadcopter starts on the left side and is given a goal behind the obstacle. In this scenario, the planning time was only 0.02s and thus only a small portion of the primary planning phase was used. This scenario illustrates a case that requires planning in time to realize it has to wait in the cranny for the obstacle to pass before it can reach its goal. A planning approach that does not take obstacle motion into account, or cannot plan in time, cannot reliably solve this problem. For example, if we were to plan in pure position-velocity lattice, we could not plan to stay in the same cranny with zero velocity for two time steps, as that would be the same state. For a real-time demonstration of the planner we highly recommend the supplemental video material[2].

We have shown that planning with some notion of time can be important for safe motion planning in confined spaces with moving obstacles. Computational considerations are also crucial to allow autonomous vehicles to plan trajectories among moving obstacles in real-time. In Fig. 7 we demonstrate the computational cost of different approaches to temporal planning. We can see that the proposed wait-augmented lattice is considerably faster than naive temporal planning where accumulated time is added to the state. The right plot also shows that, on a scenario such as this that do not require temporal planning, the proposed temporal lattice only imposes a small overhead over not planning in time at all.

Finally, to demonstrate real-time performance in a setting representative of the real-world, we consider the warehouse scenario in Fig 1 (c.f. video material[2]). The quadcopter is given a random sequence of random destinations in this warehouse, for example to inspect the inventory. These result in difficult indoor 3D navigation problems, spanning several rooms containing dynamic obstacles. Both humans on the ground, and other quadcopters flying at varying altitudes, also given random destinations. What makes this instance particularly challenging is that we use a simple conservative model of the other agents, where they are non-cooperative and do not react to the quadcopter at all. As the future motions of other agents are imperfectly known, here estimated under a constant velocity assumption, the quadcopter may get boxed-in. Safety is impossible to guarantee under these circumstances. Modeling realistic interactions in crowd behavior is an open research problem on its own, nevertheless these conservative scenarios can offer insight on the relative performance of different algorithms.

[2]https://sites.google.com/view/rhlmp

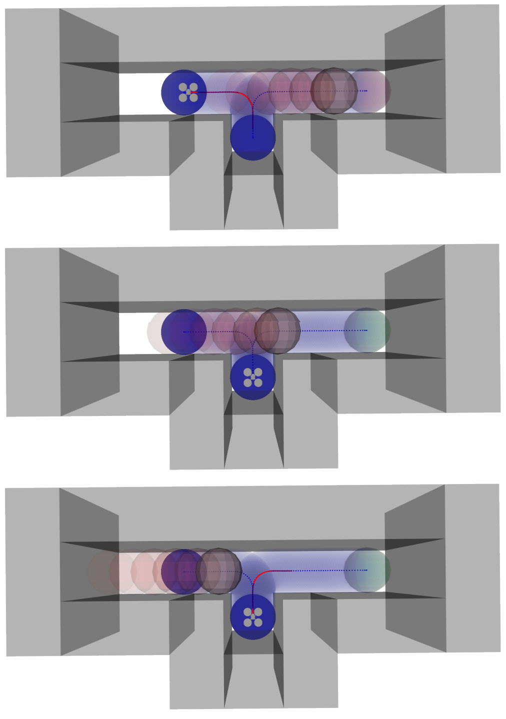

Fig. 6: This figure show a scenario where the UAV has to find a safe spot to move to, and wait at, until an approaching obstacle has passed by. Such situations necessitates a temporal lattice planner with a possible stand-still or wait action.

The results of the proposed receding-horizon lattice planner (RHLP) on this difficult scenario are shown in Tab. I. We compare the full algorithm, as presented in Section III, to three variants representing common assumptions in motion planning literature. The first is without obstacle predictions, and therefore no consideration of time or moving obstacles. As can be seen this generates considerably more collisions.

The second variant is with the receding horizon t_H fixed only allow one high-resolution motion primitive. This represents a more reactive avoidance behavior for moving obstacles. We can clearly see that a longer horizon contributes to

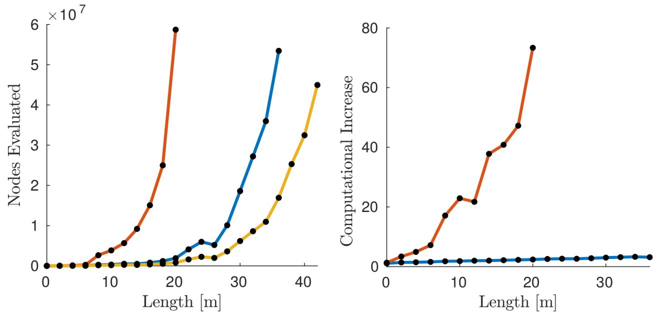

Fig. 7: The number of state evaluations required to plan around an obstacle (wall of increasing length) using different approaches. A baseline lattice *grid without time* (**Yellow**), a *grid with wait time* (**Blue**) and a *grid with time* (**Red**). **Right:** Proportion between the two time-based variants against the baseline.

safety. The average time to goal is also longer as the low-resolution primitives $\mathcal{P}_{\text{red}}$ used beyond the horizon do not admit planning with regard to velocity.

The third and perhaps most relevant variant is to instead let the horizon $t_H \to \infty$, such that high-resolution primitives $\mathcal{P}$ are used all the way to the goal. It was too slow for real-time operation, taking over a minute to generate a plan and expanded 3505800 nodes. This clearly demonstrates the benefit of a receding-horizon approach to the motion-planning problem.

TABLE I: Results from 100min of the difficult indoor warehouse scenario with moving non-cooperative obstacles in 3D. Ablation study of proposed Receding Horizon Motion Planner (RHLP) against three restricted baselines. *Time to goal* and *Nodes evaluated* are averages per randomized goal, and plan, respectively.

Motion planner	Collisions/min	Time to goal $[s]$	Nodes eval.
RHLP	0.24	57.3	36394
$\text{RHLP}_{\text{no prediction}}$	0.96	85.0	40769
$\text{RHLP}_{\text{horizon}=1\,\text{prim.}}$	1.12	127.2	2745
$\text{RHLP}_{\text{horizon}=\infty}$	not real-time	not real-time	3505800

V. Conclusions and Future Work

We proposed a general optimization-based receding-horizon lattice-based motion planning framework with collision avoidance functionality for both complex 3D environments and moving obstacles. This includes planning with both dynamics and time, such that the quadcopter can plan trajectories around, and move out of the way of, other agents. This was highlighted by the narrow corridor example, where the quadcopter has to use crannies or side passages, to wait for other agents to pass. We also demonstrated real-time performance on a difficult warehouse scenario with multiple moving obstacles, both people and other UAVs flying at varying altitudes. To the best of our knowledge, no other optimization-based and dynamically-feasible approach has demonstrated this capability in real-time.

As future work we would like to increase robustness by explicitly considering uncertainty in obstacle motion, as well as to implement the proposed framework on real quadcopter hardware.

Acknowledgments

This work was partially supported by FFI/VINNOVA, the Wallenberg Artificial Intelligence, Autonomous Systems and Software Program (WASP) funded by Knut and Alice Wallenberg Foundation, the Swedish Foundation for Strategic Research (SSF) project Symbicloud, the ELLIIT Excellence Center at Linköping-Lund for Information Technology, Swedish Research Council (VR) Linnaeus Center CADICS, and the National Graduate School in Computer Science, Sweden (CUGS).

References

[1] J. Petereit, T. Emter, and C. W. Frey, "Mobile robot motion planning in multi-resolution lattices with hybrid dimensionality," *IFAC Proceedings Volumes*, vol. 46, no. 10, pp. 158–163, 2013.

[2] M. Pivtoraiko and A. Kelly, "Differentially constrained motion replanning using state lattices with graduated fidelity," in *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Sept 2008, pp. 2611–2616.

[3] P. Fiorini and Z. Shiller, "Motion planning in dynamic environments using velocity obstacles," *The International Journal of Robotics Research*, vol. 17, no. 7, pp. 760–772, 1998.

[4] 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)*, May 2016, pp. 4597–4604.

[5] D. Mellinger and V. Kumar, "Minimum snap trajectory generation and control for quadrotors," in *2011 IEEE International Conference on Robotics and Automation*, May 2011, pp. 2520–2525.

[6] C. Richter, A. Bry, and N. Roy, "Polynomial trajectory planning for aggressive quadrotor flight in dense indoor environments," in *Robotics Research*. Springer, 2016, pp. 649–666.

[7] M. Burri, H. Oleynikova, , M. W. Achtelik, and R. Siegwart, "Real-time visual-inertial mapping, re-localization and planning onboard mavs in unknown environments," in *proceedings of the international conference on Intelligent Robots and Systems (IROS)*, Sept 2015.

[8] 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.

[9] 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.

[10] E. Frazzoli, M. A. Dahleh, and E. Feron, "Real-time motion planning for agile autonomous vehicles," *Journal of guidance, control, and dynamics*, vol. 25, no. 1, pp. 116–129, 2002.

[11] J. Ziegler and C. Stiller, "Spatiotemporal state lattices for fast trajectory planning in dynamic on-road driving scenarios," in *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Oct 2009, pp. 1879–1884.

[12] M. Likhachev and D. Ferguson, "Planning long dynamically feasible maneuvers for autonomous vehicles," *The International Journal of Robotics Research*, vol. 28, no. 8, pp. 933–945, 2009.

[13] O. Ljungqvist, N. Evestedt, M. Cirillo, D. Axehill, and O. Holmer, "Lattice-based motion planning for a general 2-trailer system," in *Proceedings of the 2017 IEEE Intelligent Vehicles Symposium (IV), Los Angeles*, June 2017, pp. 2455–2461.

[14] 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, pp. 298–312, 2011.

[15] M. Pivtoraiko, D. Mellinger, and V. Kumar, "Incremental micro-uav motion replanning for exploring unknown environments," in *Robotics and Automation (ICRA), 2013 IEEE International Conference on*. IEEE, 2013, pp. 2452–2458.

[16] S. Liu, N. Atanasov, K. Mohta, and V. Kumar, "Search-based motion planning for quadrotors using linear quadratic minimum time control," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Sept 2017, pp. 2872–2879.

[17] S. M. LaValle, *Planning algorithms*. Cambridge university press, 2006.

[18] M. Quigley, K. Conley, B. P. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "ROS: an open-source robot operating system," in *ICRA Workshop on Open Source Software*, 2009.

[19] 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.

[20] B. Houska, H. J. Ferreau, and M. Diehl, "An auto-generated real-time iteration algorithm for nonlinear mpc in the microsecond range," *Automatica*, vol. 47, no. 10, pp. 2279 – 2285, 2011.

[21] H. J. Ferreau, C. Kirches, A. Potschka, H. G. Bock, and M. Diehl, "qpoases: a parametric active-set algorithm for quadratic programming," *Mathematical Programming Computation*, vol. 6, no. 4, pp. 327–363, Dec 2014.