Unmanned Systems, Vol. 9, No. 2 (2021) 165–198

DOI: 10.1142/S2301385021500151

UNMANNED SYSTEMS
https://www.worldscientific.com/worldscinet/us

A Survey of Motion and Task Planning Techniques for Unmanned Multicopter Systems

Menglu Lan[*,§], Shupeng Lai[†], Tong H. Lee[†], Ben M. Chen[†,‡]

[*]*Graduate School for Integrative Science and Engineering, National University of Singapore, Singapore*

[†]*Department of Electrical and Computer Engineering, National University of Singapore, Singapore*

[‡]*Department of Mechanical and Automation Engineering, Chinese University of Hong Kong, Shatin, N.T., Hong Kong*

Unmanned aerial systems provide many applications with the ability to perform flying tasks autonomously, and hence have received significant research and commercial attention in the past decade. One of the most popular unmanned aerial platforms for such tasks is the small-scale rotorcraft with multiple rotors, commonly known as multicopters. In order for these platforms to perform fully autonomous missions and tasks, they require a sophisticated low-level flight control system that is integrated with advanced task and motion planning modules, which combine together to form the complete unmanned aerial system (UAS). In this paper, the planning module of unmanned multicopter systems is discussed in detail, and a comprehensive survey on techniques for both motion and task planning reported in literature and by the Unmanned Systems Research Group at the National University of Singapore is presented.

Keywords: Task planning; motion planning; rotorcraft.

US

1. Introduction

In recent years, research and development of autonomous unmanned systems have continued to gained momentum in both theoretical domains and industrial applications. The areas where such systems can be implemented vary widely, from space satellites to aerial, terrestrial, and underwater robots. Developmental trends have shown that autonomous unmanned systems, such as unmanned aerial vehicles (UAVs), unmanned ground vehicles (UGVs), unmanned surface vehicles (USVs), and unmanned underwater vehicles (UUVs), with the integration of intelligent elements and functions, are set to play significant roles in many applications and are likely to be increasingly present in human societies.

Received 7 October 2020; Revised 25 November 2020; Accepted 27 November 2020; Published 18 February 2021. This paper was recommended for publication in its revised form by editorial board member, Lihua Xie.
Email Address: [§]*lanmenglu@u.nus.edu*

An autonomous unmanned vehicle is defined as a mechanical machine equipped with the necessary components, such as data processing units, sensors, automatic control, and communication systems, to perform autonomous missions without the interference of a human operator. The difficulty of performing autonomous operations increases in complicated environments, such as cluttered surroundings or places where the Global Positioning System (GPS) is unavailable. In order to enable full autonomy of an unmanned platform, it is necessary to fully understand the dynamic model of the hardware system and integrate the key components of the model into the overall unmanned system framework. This includes the systems for mission and task management (to manage application tasks), motion planning (to generate feasible motion paths), sensing and positioning (to detect and sense the surrounding environment), and automatic control (to control and guide the vehicle).

The initial development of UAVs was primarily driven by military applications [1], which involves tasks such as

surveillance, reconnaissance, and airstrike missions. The earliest attempt at a powered UAV began in the early 1900s, where an unmanned aircraft was used to carry an explosive payload to a predetermined target. Over the years, the development of military UAVs has resulted in sophisticated technologies and systems, with prominent examples such as the *Predator* from the United States Air Force (USAF) and rapid development of small unmanned aircraft systems [2]. In recent years, the use of single and multiple UAVs has also expanded rapidly in many civilian domains and applications [3], including agricultural applications [4], entertainment [5], remote sensing and data collection, aerial manipulation [6], parcel deliveries, and aerial photography.

One of the most popular civilian aerial platforms is the small-scale rotorcraft with multiple rotors, commonly known as the multicopter or multirotor. Based on the number and configuration of rotors, multicopters can be further classified as shown in Fig. 1. Compared to their conventional helicopter counterparts (e.g. [7, 8]), multicopters have a simpler mechanical structure and thus require relatively less effort to stabilize and control. Compared to the fixed-wing aircraft (e.g. [9]), they possess the advantage of being able to hover and perform vertical take-off and landing. The superior maneuverability and compact size of these vehicles allow them to operate in constrained spaces. As a result, they have become a popular choice for various applications such as tunnel inspection [10], environment exploration [11], and search and rescue in damaged buildings [12, 13].

A fully autonomous multicopter system is typically equipped with various avionics including onboard computers and sensors. An example of such a vehicle is depicted in Fig. 2, which is an unmanned hexacopter platform developed by the Unmanned Systems Research Group at the

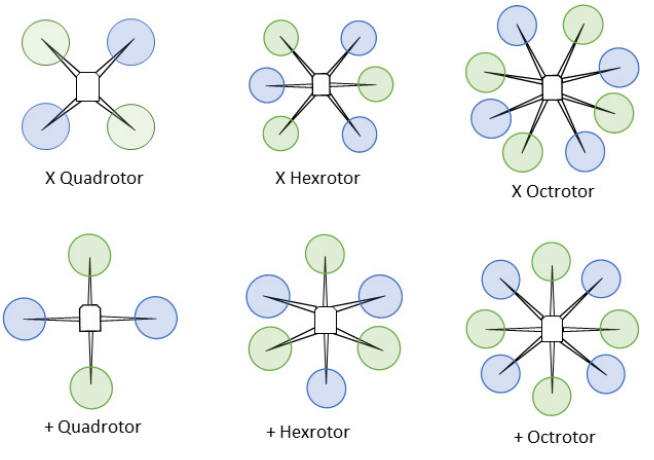


Fig. 1. Various common configurations of multicopters. Green indicates a clockwise rotating rotor while blue indicates a counter-clockwise rotating rotor.

Fig. 2. A hexacopter developed by the NUS research team.

National University of Singapore (NUS) for GPS-denied environments [12]. The hexacopter has a tip-to-tip width of 60 cm and a maximum take-off weight of 1.9 kg. It has two onboard computers: a Pixhawk flight controller, and a small-scale high-level computer. The Pixhawk flight controller [14] has a main 32-bit 216 MHz ARM processor with only 2 MB memory and 512 KB RAM. The Pixhawk also integrates various onboard sensors such as the accelerometer, gyroscope, magnetometer and barometer which provide basic measurements for flight control. The small-scale high-level computer used is the Upboard (Intel Z8350 processor). Weighing just 80 g, the Upboard is mainly responsible for running complex high-level algorithms for tasks such as planning, localization and perception. The platform is integrated with a Hokuyo UTM-30LX laser scanner, which senses obstacles in the surrounding environment within 30 m and has a 270° field of view. A camera is also installed on the platform to provide additional image information for vision-based algorithms.

The typical framework for an autonomous unmanned rotorcraft system is shown in Fig. 3. In general, there are three major sub-systems: measurement and perception, flight control, and planning. While the main focus of this paper is the planning module of the system, the following sections will first introduce each of these modules to provide a basic background on the complete unmanned system.

1.1. *Measurement*

The measurement module of the UAS is responsible for receiving data from the onboard sensors and generating both state estimates of the vehicle, such as position and velocity, and those of the environment, such as the location of geometric obstacles (environment mapping) and targets of interest (objection detection). Due to the size and payload limit of small-scale multicopters, these platforms typically adopt a lightweight microelectromechanical system (MEMS)-based inertial measurement unit (IMU) to provide

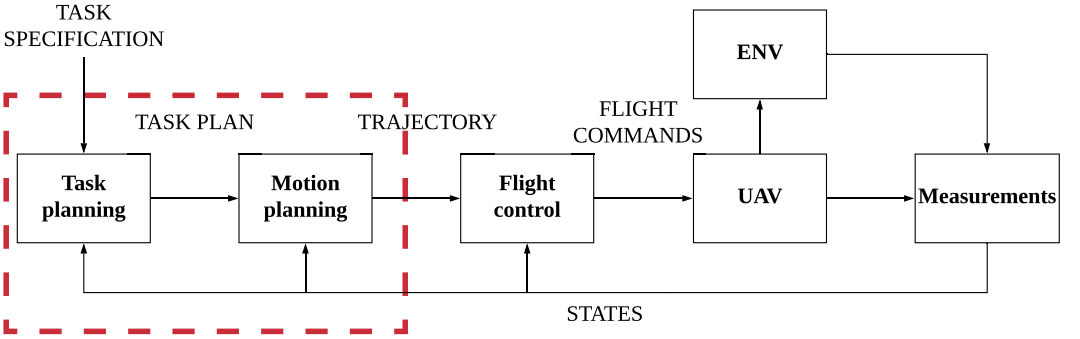


Fig. 3. A typical framework for an autonomous unmanned aerial system.

the basic state information of the vehicle. A common example of such a system is the sensor suite included in the Pixhawk flight controller. The signal inputs of each subcomponent in the IMU are fused with an extended Kalman filter (EKF) to provide the attitude, acceleration and height estimation of the vehicle, allowing the attitude of the multicopter to be controlled and stabilized through feedback control.

However, as the low quality MEMS-based accelerometer typically produces acceleration measurements that are noisy and commonly with bias, velocity and position estimations based purely on IMU measurements may diverge in seconds. To regulate and control the velocity and position of the vehicle, it is possible and sometimes preferred to use additional sensors such as GPS and Ultra-Wide Band (UWB)-based technologies to directly measure the velocity and position of the vehicle. These measurements are then further fused with the EKF to produce more accurate readings. In environments where GPS is denied and the installation of external sensors such as UWB is not feasible, sensors such as stereo cameras and laser scanners can be used to provide state estimation through the process of simultaneous location and mapping (SLAM). The use of cameras and lasers also allows the vehicle to acquire additional information of the environment, such as the location of obstacles, and this information can be further processed by perception algorithms to facilitate other functional modules such as control and planning.

1.2. *Flight control*

The basic functions of a flight control module are: (i) to stabilize the unmanned vehicle; and (ii) to track a reference trajectory with bounded error in the presence of environmental and model uncertainties. Multicopters are inherently unstable, hence a cascaded control structure, which consists of inner- and outer-loop control systems, is often adopted to stabilize the vehicle. The outer-loop position controller (20–50 Hz) tracks the reference trajectory generated from the motion planning module and outputs the desired thrust and attitude commands to the inner-loop attitude controller. The high bandwidth attitude controller (400–1000 Hz) then maps these commands to actual motor forces and stabilizes the vehicle. To handle uncertainties and external disturbances, techniques such as robust and perfect tracking (RPT) control developed in [15] can be utilized to design the outer-loop controller. The RPT control makes full use of the reference trajectory and its derivatives, when available, to reduce the tracking error significantly. In practice, when the reference trajectory generated by the motion planning algorithm is dynamically feasible, the states of the actual vehicle can be bounded inside a tube centered around the reference (nominal) trajectory. This bounded tube is useful for high-level planning processes. For example, in motion planning with obstacles, the trajectory should be at least $e + r$ away from the obstacle, where r is the radius of the vehicle, and e is the tube radius of the corresponding position trajectory. There are methods [16] to estimate the time-varying tube radius e given the model of the vehicle and the limits of the uncertainties and disturbances.

1.3. *Planning*

In autonomous systems, the primary purpose of planning is to convert the high-level task specifications from human operators to low-level instructions for the vehicle. Planning requires a predictive model that describes the behaviors of the system of interest. Based on the model, the planner can reason what will happen if the vehicle takes a particular action, and then decide what actions should be taken to fulfill the given tasks. Many planners use a state-based model, which typically consists of a state space, an input/action space, a differential/difference equation that expresses the system dynamics, and a set of initial states. Usually, there is also a set of constraints over the states and inputs, and the model describes how the states would change with the inputs over time. The task specifications, or

planning goals, can be understood as a set of additional constraints. These goals can be a simple boundary state constraint (i.e. a reachability problem) or something more complex such as temporally-extended ones. The goal of planning is to find a set of control inputs by searching in the parameterized input space of the system such that the resulted state trajectory satisfies all constraints.

Based on the modeling of the environment, there are three general types of the planning:

(i) Open loop (nonreactive): The environment is assumed to be static or the vehicle is the only agent that can change the states of the environment. In such cases, the goal of planning is to find an input sequence where the corresponding state sequence satisfies all the constraints.
(ii) Iterative: Iterative planning often interleaves with the execution. For each iteration, a prediction of the environment is made and nonreactive planning is performed. Re-planning is then triggered periodically or when the environment changes during the execution. The final state trajectory over the full execution period has to be ensured to satisfy all the requirements. Model predictive control (MPC) can be regarded as a type of iterative planning.
(iii) Reactive: In reactive planning, the environment uncertainty is explicitly modeled. The resulting plan is no longer a sequence of actions but a policy that maps each state to a set of actions. The goal of reactive planning is to find a control policy such that under any modeled environment behavior, the vehicle will take correct actions according to the policy and satisfy all the requirements.

Planning can take place at different levels by using models of different abstraction-levels. For multicopters, there are two levels of planning in general: motion planning and task planning. Motion planning usually works with a detailed dynamic model and focuses on the dynamic and geometric constraints. On the other hand, task planning works with a highly abstract discrete model and focuses on handling task-related constraints. Here, both task and motion planning are briefly introduced below, and the main differences between the two are summarized in Table 1. Although we focus on the planning for a single agent, many of the techniques are also used in multi-agent scenario [17] with different types of vehicles [18].

1.3.1. *Motion planning*

Motion planning is the task of guiding a vehicle from one place to another while avoiding obstacles. In the context of multicopters, the goal of motion planning is to compute a dynamically feasible and collision-free trajectory that starts from an initial state and ends at a specified target. The computed state trajectory is later fed into a lower-level controller as a tracking reference. For traditional motion planning, the goal specification does not involve any temporal constraints and is fairly straightforward, i.e. reaching a target state or zone. The terms *dynamically feasible* and *collision-free* specify two major constraints in motion planning: the dynamics constraints of the vehicle; and the geometric constraints induced by obstacles in the environment. A recent review of UAV path planning and obstacle avoidance can be found in [19].

The task of motion planning occurs in a continuous domain. The planning model is often an extension of the detailed dynamic model of the vehicle itself, which is described by a set of differential constraints. For example, a third-order integrator-based model can be used to describe the dynamics of a quadrotor, in which the state variables are the positions, velocities, and accelerations in the three axes, and the inputs are the corresponding jerks (i.e. the derivatives of the acceleration). Besides the differential constraints, we can enforce other dynamics-related constraints by adding extra state and input constraints. For instance, we can limit the horizontal velocity to be within 2 m/s and the acceleration to be within 0.5 m/s^2. In addition to the dynamic constraints, the planning model also has to consider the geometric constraints of the environment. In traditional motion planning, the environment states are usually the positions of obstacles. To describe these features, the model

Table 1. Comparison between traditional motion planning and traditional task planning.

	Motion Planning	Task Planning
Model	Detailed dynamic model of the vehicle; Environment map	Highly abstract symbolic model of the vehicle and environment
Goal	Reach a target state	Temporally extended goals
Main constraints	Vehicle dynamics; Geometric constraints induced by the environment obstacles;	Task related constraints
Domain	Continuous	Discrete

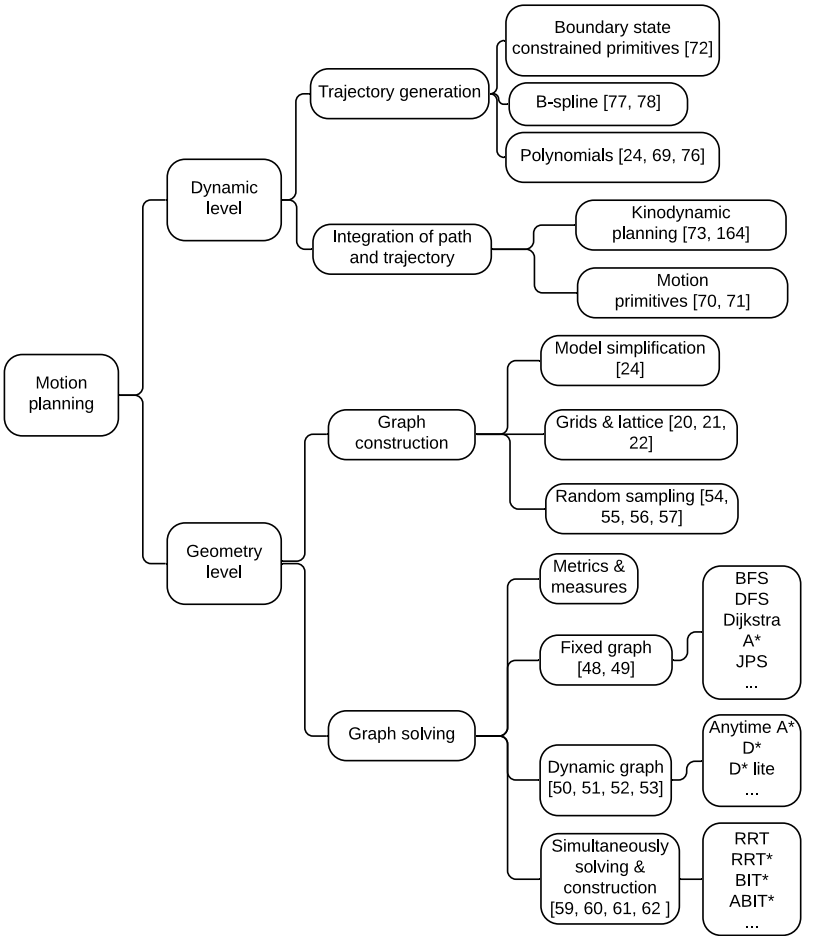


Fig. 4. Topics on motion planning for unmanned multicopters.

of the environment is often represented by an obstacle map, which is typically generated by a perception module. The most common map representation for point cloud based obstacle data is the occupancy grid map [20, 21]. For planning purposes, it is beneficial to possess additional distance information of this map [22]. These geometric constraints can be represented as a set of (possibly variant) additional position constraints that the vehicle needs to satisfy. Figure 4 presents an overview and breakdown of the various topics on motion planning for unmanned rotorcraft.

1.3.2. *Task planning*

Many real-world applications require more than simple vehicular movement from one position to another while avoiding obstacles along the way. Often, additional planning is required to be performed on top of the motion planning discussed previously. Unlike motion planning, where the goal is to reach a single position, many practical tasks involve temporal constraints, i.e. constraints over the time domain.

Consider the example shown in Fig. 5, where a drone is used for a parcel delivery task. The map contains several areas of interest, such as the warehouse where the UAV can pick-up parcels, and three drop sites: *Sites A*, *B* and *C*. The map also contains a few forbidden areas where the UAV should not enter, which are indicated by red circles. In this delivery mission, the human operator may impose temporal constraints on the visiting sequence of each site, e.g. "*Site C should be visited only after visiting Site A or Site B*". These types of constraints are out of the scope of traditional motion planning, and are typically considered as task-related constraints and handled by a task planner.

More importantly, task planning usually works with a much more abstract model, which is task-dependent rather than vehicle-dependent. A typical task planning model often completely ignores the details of the vehicle's dynamics and

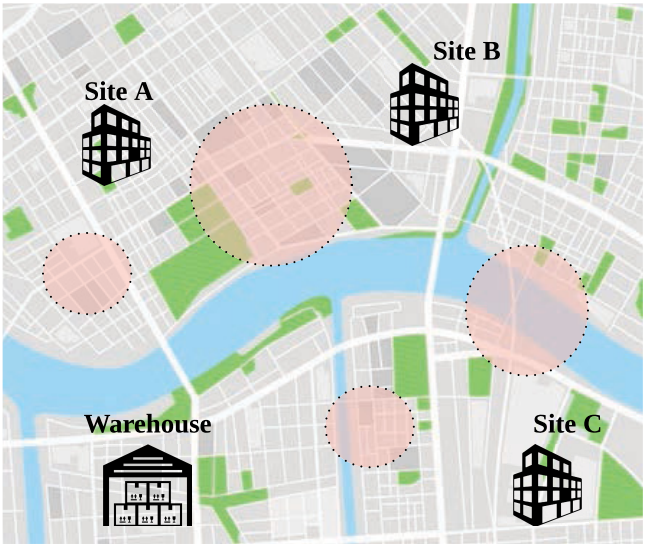


Fig. 5. An example of a task workspace for UAV parcel delivery.

geometric obstacles. It manipulates a set of abstract symbolic actions that can be mapped to a set of existing functionalities of the vehicle and finds a sequence of suitable actions that fulfills the task requirements at a symbolic level. For a multicopter, many of these actions involve flying toward or covering an area of interest, which are handled by the motion planning module.

For instance, as shown in Fig. 5, we consider a task where the vehicle is required to deliver three parcels located at the warehouse to *Sites A, B* and *C*. There are multiple no-fly zones shown as red circles that the vehicle is prohibited from entering. At the task level, the detailed geometric and dynamic information have been abstracted away. The task plan may be coarsely expressed as "*fly to the warehouse*, *detect the parcel*, *pick up the parcel*, *fly to Site B*, *drop the parcel*". The details of tasks such as navigating to the warehouse, identifying and picking up the parcel are traditionally not a concern of task planning. It is assumed that other modules (including but not limited to motion planning) are responsible for the actual refinement and

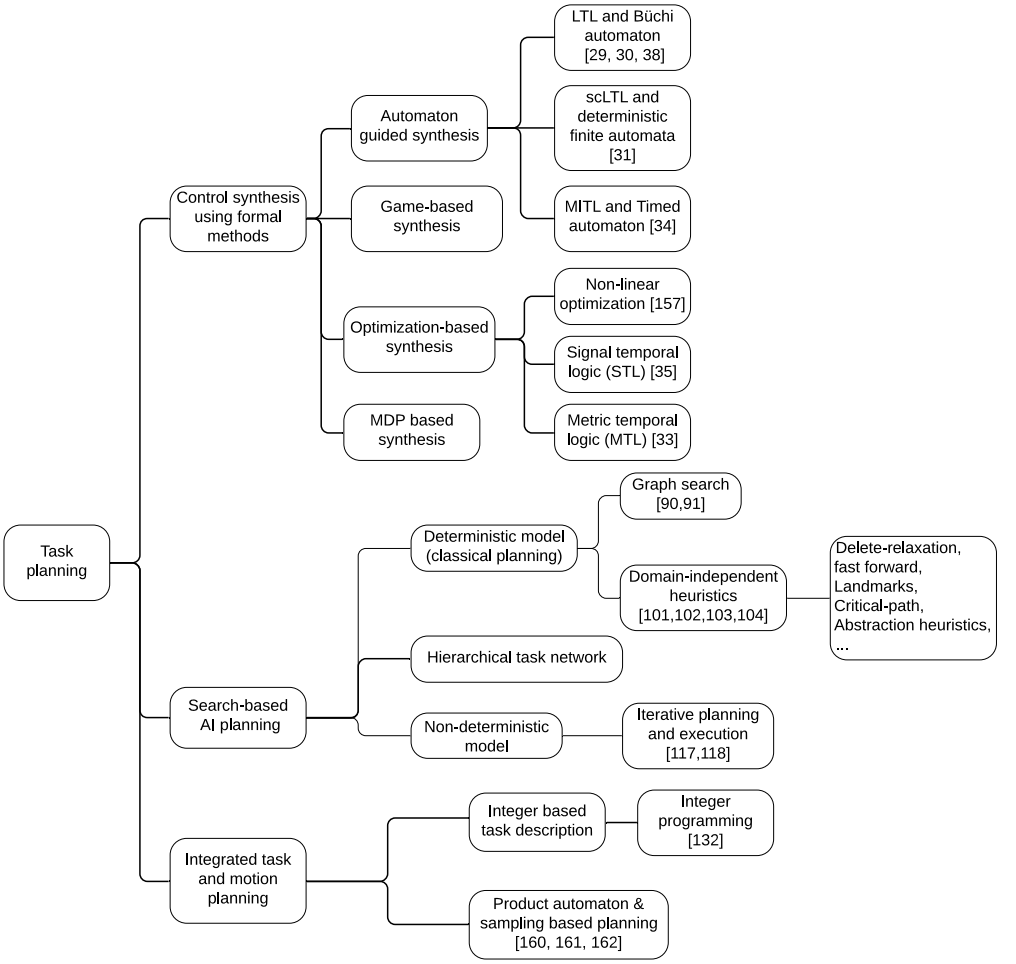


Fig. 6. Topics on task planning for unmanned multicopters.

implementation of these abstract actions. On the other hand, the motion-level is responsible for actions that involve the movement of the vehicle such as *pick up* and *fly to*. Geometric and dynamic constraints such as obstacles, no-fly-zones, the vehicle's maximum velocity, and total thrust limitations are included in the motion planning phase to generate a safe and dynamically feasible plan.

In contrast to motion planning which focuses on the dynamic and geometric constraints in the continuous domain, task planning works with a highly abstract model in the discrete domain and focuses on generating a coarse task plan. The system dynamics is typically modeled as a discrete transition system at the task level and discrete planning techniques are often applied to solve the task planning problem. An overview of the topics related to task planning for unmanned multicopters are summarized in Fig. 6.

1.4. *Outline*

Planning is an indispensable part of any autonomous system as it is responsible for the decision making and execution of required actions. As there have been many developments and proposed approaches on performing these tasks presented over the years, this paper aims to present a comprehensive survey on the current state of motion and task planning techniques for multicopters. In particular, we limit the scope of this survey to consider only the case of a single multicopter. The rest of the paper is organized as follows.

The necessary background information, such as the dynamic model of the multicopter used for motion planning, common graph search algorithms, symbolic representation of task-level system abstraction models and temporal task specifications, are first presented in Sec. 2. These provide the fundamental basis for further discussion on the planning modules. Next, in Sec. 3 we present a survey of commonly used motion planning techniques reported in literature. We then move from the motion level to task level planning, and present the general task-level symbolic representation of the system abstraction in Sec. 2.3. A wide range of temporal task specifications used to formally describe the task requirements are subsequently introduced in Sec. 2.4. Section 4 highlights a survey on task planning techniques for multicopter applications and the integration of task planning and motion planning is discussed in Sec. 5. Lastly, the concluding remarks are presented in Sec. 6.

2. Background and Preliminary Materials

In this section, preliminary information which is essential for discussion of techniques for motion and task planning are first presented. These include the dynamic model of the multicopter used for motion planning, common graph search algorithms, symbolic representation of task-level system abstraction models, and temporal task specifications.

2.1. *Dynamic model of a multicopter*

In this subsection, a dynamic model of a quadcopter based on the work in [23] is presented. The basic quadcopter has highly nonlinear dynamics and high degrees of freedom. However, it has been proven that the quadcopter is differentially flat [24], in which one can choose a set of flat outputs as the positions of the quadcopter in the 3D space together with its yaw angle. For motion planning, it is a common practice to use a simplified integrator-based model. For example, [25] used a double integrator model and [23, 26] used a third-order model, while a quadruple integrator model was used in [24, 27]. Following the work in [23], we show how a third-order integrator-based model can be used to describe the dynamics of a quadcopter. The quadcopter is modeled as a rigid body in three-dimensional space with six degrees of freedom and we use x, y, z to denote the positions in the three axes, respectively. The control inputs are the rotational rates about the vehicle body axes $\omega_x, \omega_y, \omega_z$ respectively as shown in Fig. 7), and F is the mass normalized collective thrust. It is assumed that there is a high bandwidth inner-loop controller that can perfectly track the angular rates and map these commands to actual motor forces.

We can describe the motion of the quadrotor as

$$\begin{pmatrix} \ddot{x} \\ \ddot{y} \\ \ddot{z} \end{pmatrix} = \mathbf{R}_{\mathrm{G/B}} \begin{pmatrix} 0 \\ 0 \\ F \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \\ -g \end{pmatrix}, \tag{1}$$

where $\mathbf{R}_{\mathrm{G/B}}$ denotes the rotational matrix from the body frame $\mathbf{R}_{\mathrm{B}}$ of the vehicle to the inertial frame $\mathbf{R}_{\mathrm{G}}$, and

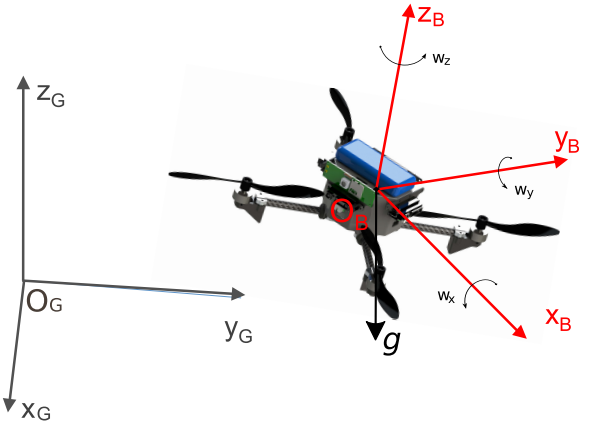


Fig. 7. Conventional UAV coordinate systems.

g represents the gravitational acceleration constant. The attitude of the vehicle is related to the rotational rates by

$$\dot{\mathbf{R}}_{\mathrm{G/B}} = \mathbf{R}_{\mathrm{G/B}} \begin{bmatrix} 0 & -\omega_z & \omega_y \\ \omega_z & 0 & -\omega_x \\ -\omega_y & \omega_x & 0 \end{bmatrix}. \tag{2}$$

We define the mass-normalized global force $H \in \mathbb{R}^3$ as

$$H := \begin{pmatrix} \ddot{x} \\ \ddot{y} \\ \ddot{z} \end{pmatrix} m + \begin{pmatrix} 0 \\ 0 \\ g \end{pmatrix} = \mathbf{R}_{\mathrm{G/B}} \begin{pmatrix} 0 \\ 0 \\ F \end{pmatrix}, \tag{3}$$

where m is the mass of the vehicle.

The multicopter is shown as differential flat [24] where all of its states can be expressed as the combination of the flat outputs $[x, y, z, \psi]$ and their derivatives. Here, the $[x, y, z, \psi]$ indicates the vehicles position $[x, y, z]$ and yaw ψ. The majority of the multi-copter motion planning literature utilizes the differential flatness to reduce the dimension of the motion planning problem, and the final trajectory can be expressed functions of $[x, y, z, \psi]$ over time. In practice, it is desirable to generate a smooth trajectory to minimize wobbling in the attitude of the vehicle as excessive erratic movements can compromise both mechanical structure and onboard sensing activities. This also reduces the undesirable pendulum effect when the vehicle is carrying payloads in the underslung position such as a heavy calligraphy brush in [28]. To generate a smooth trajectory, we can penalize the rotational speed about the vehicle body axes $\|\omega_x\|$, $\|\omega_y\|$ and $\|\omega_z\|$. Here, we focus on minimizing $\|\omega_x\|$ and $\|\omega_y\|$, since we can always make sure that there is no rotation around the body z-axis by fixing the yaw angle. Moreover, we should also constrain the global force H as the maximum thrust available is limited by the thrust that can be provided by the motors. This can be written as

$$\begin{aligned} &\|H\| = \sqrt{\ddot{x}^2 + \ddot{y}^2 + (\ddot{x} + g)^2} > H_{\min}, \\ &\|H\| < H_{\max}. \end{aligned} \tag{4}$$

As shown in [23], we can satisfy the constraints expressed in Eq. (4) by limiting the accelerations of the vehicle

$$|\ddot{x}| \le \ddot{x}_{\max}, \quad |\ddot{y}| \le \ddot{y}_{\max}, \quad |\ddot{z}| \le \ddot{z}_{\max}. \tag{5}$$

In addition, for rotational speeds $\|\omega_x\|$ and $\|\omega_y\|$, the upper bounds are set as

$$\|\omega_{x,y}\| \le \frac{\|\dot{H}\|}{\|H\|}. \tag{6}$$

Since we have already constrained the denominator $\|H\|$ (see Eq. (4)), we can indirectly penalize $\|\omega_x\|$ and $\|\omega_y\|$ by limiting or penalizing

$$\int (\dddot{x}^2 + \dddot{y}^2 + \dddot{z}^2) dt. \tag{7}$$

The quadrotor model presented above can also be applied to other multicopters such as octocopters for the trajectory generation process. This is because at the lower control level, the inner loop attitude controller has a much higher bandwidth than the outer-loop position controller. Thus the number and location of the propellers can be safely ignored during higher-level planning.

2.2. *Graph search algorithms*

Graph search serves as the backbone for various motion and task planning algorithms, and here we briefly present an outline of a general serialized graph search algorithm, as shown in Fig. 8.

First, a storage space $\mathcal{Q}$, usually a type of queue structure, is used to store and order the states that need to be examined. The properties of $\mathcal{Q}$ vary depending on the specific algorithm. If $\mathcal{Q}$ is ordered with a given priority function, the states stored by $\mathcal{Q}$ forms a search wavefront. For instance, breadth-first search uses a first-in-first-out (FIFO) queue while the depth-first search uses a stack (last-in-first-out LIFO). The optimal search algorithm Dijkstra sorts the queue by the cost-to-come of the state. A* extends the Dijkstra by considering an additional cost-to-go heuristics.

$\mathcal{Q}$ starts as an empty container and is initialized with the initial state s_0 (Line 1). While $\mathcal{Q}$ is not empty, we pop out an element of the queue, s, based on the assigned priority (Line 2–3). If the given termination condition has been satisfied, for example state if s is already in the goal set S_G, then the solution has been found (Line 4–5). Otherwise, we continue to expand the state s by applying all possible actions a (Line 6). For each successor state s', we check whether the cost of s' can be improved by changing its parent to s (Line 7– 8). Here, cost(s) and cost(s') are the costs of state s and s' respectively, and cost(s, s') is the edge cost between states s and s'. If it can be improved, we assign s as the parent of s'.

```
1:  Q ← {s_0};
2:  while Q not empty do
3:      s ← GETBESTQUEUEELEMENT(Q)
4:      if termination condition satisfied then
5:          return SUCCESS
6:      for all a ∈ Act(s) do
7:          s' ← f(s, a)
8:          if cost(s') > cost(s) + cost(s, s') then
9:              assign s as the parent of s'
10:             if s' ∉ Q then
11:                 Q ←+ {s'}
12:             else
13:                 handle duplicate s'
    return FAILURE
```

Fig. 8. A general abstract template of the graph search algorithm.

If state s' already has a parent, it is replaced by s. Then, if s' is not inside $\mathcal{Q}$, we insert s' into $\mathcal{Q}$. Otherwise, the duplication is handled typically by adjusting the priority of s' in $\mathcal{Q}$. If the goal set is never reached, we then fail to find a solution. The plan result can be obtained by simply tracing back the parents of the goal toward the initial state. The above algorithm provides a very brief outline and omits implementation details.

2.3. *Symbolic representation of task-level system abstraction models*

The symbolic representation for the system abstraction model, which is often used in task-level symbolic planning, will be introduced in this section.

Definition 1 (Labeled transition system). A *labeled transition system* TS is a tuple $\mathrm{TS} = (S, S_0, Act, \rightarrow_{\mathrm{TS}}, \Pi, L)$ defined by

(i) A set of states S
(ii) A set of initial states $S_0 \subseteq S$
(iii) A set of actions Act
(iv) A transition relation $\rightarrow_{\mathrm{TS}} \subseteq S \times Act \times S$
(v) A set of atomic propositions Π
(vi) A labeling function $L : S \rightarrow 2^{\Pi}$

The sets S and Act can be finite or infinite. TS is a *finite transition system* if the cardinalities of S and Act are finite. The TS is called a *deterministic transition system* if it has only one initial state s_0 and the transition relation $\rightarrow_{\mathrm{TS}}$ is also deterministic.

We often write $s \rightarrow^{a}_{\mathrm{TS}} s'$ to denote $(s, a, s') \in \rightarrow_{\mathrm{TS}}$, where $a \in Act$ and $s, s' \in S$. Let $\mathtt{post}_a(s)$ denote the set of all possible successor states after applying an action $a \in Act$ from s, i.e. $\mathtt{post}_a(s) = \{s' | (s, a, s') \in \rightarrow_{\mathrm{TS}}\}$. The set of all successor states of s is then given by

$$\mathtt{post}(s) = \bigcup_{a \in A} \mathtt{post}_a(s). \tag{8}$$

An *execution* of a transition system TS is a sequence of $\rho = (s_0, a_0), (s_1, a_1), (s_2, a_2), \ldots$, where $s_0 \in S_0$ and $(s_i, a_i, s_{i+1}) \in \rightarrow_{\mathrm{TS}}$ for all $i \geq 0$. A *control strategy* is a partial function $\pi : (s_0, a_0, \ldots, s_{i-1}, a_{i-1}, s_i) \rightarrow a_i$ that maps the execution history to the next action. A sequence of states $\boldsymbol{s} : s_0, s_1, s_2, \ldots$ from the execution ρ is called a *path*. For a transition system TS, $\mathtt{trace}(\tau) \in (2^{\Pi})^{\omega}$, the *trace* of an infinite path τ, is an infinite *word* emitted from the path, i.e. $\mathtt{trace}(\boldsymbol{s}) := L(s_0), L(s_1), \ldots$

In the literature, the above labeled transition system is sometimes also referred to as a *Kripke structure* defined over the atomic proposition set Π. For convenience, the action set Act is often omitted, and the transition relation is simplified as $\rightarrow_{\mathrm{TS}} \subseteq S \times S$, resulting in a tuple $\mathcal{K} = (S, S_0, \rightarrow_{\mathrm{TS}}, L)$.

We next show how to represent a discrete-time system as a transition system. Consider a discrete-time dynamical system with dynamics governed by

$$x^+ = g(x, u), \tag{9}$$

where $x \in X \subseteq \mathbb{R}^n$ is the state variable of the system with some constraints, $u \in U \subseteq \mathbb{R}^m$ is the control input, and x^+ is the next state. We can formulate the above discrete-time system as a transition system

$$\mathrm{TS}_{\mathrm{d}} = (S, S_0, Act, \rightarrow_{\mathrm{TS}_{\mathrm{d}}}, \Pi, L), \tag{10}$$

where

(i) $S = X$ and $S_0 = X_0$
(ii) $Act = U$ (each action in Act is a control input in U)
(iii) $(s, u, s') \in \rightarrow_{\mathrm{TS}_{\mathrm{d}}}$ if and only if there exists $s' = g(s, u)$
(iv) Π is the set of atomic propositions on S
(v) $L : S \rightarrow 2^{\Pi}$ is the labeling function

We can also represent a continuous-time dynamical system as a transition system. Consider a continuous-time system characterized by

$$\dot{x} = f(x, u), \tag{11}$$

where $x \in X \subseteq \mathbb{R}^n$ is the state variable with some constraints and $u \in U \subseteq \mathbb{R}^m$ is the control input. We can rewrite the above system as the following transition system:

$$\mathrm{TS}_{\mathrm{c}} = (S, S_0, Act, \rightarrow_{\mathrm{TS}_{\mathrm{c}}}, \Pi, L), \tag{12}$$

where

(i) $S = X$ and $S_0 = X_0$
(ii) $Act = \cup_{\tau \in \mathbb{R}^+} U[0, \tau]$
(iii) $(s, \boldsymbol{u}, s') \in \rightarrow_{\mathrm{TS}_{\mathrm{c}}}$ if and only if there exists $\zeta : [0, \tau] \rightarrow \mathbb{R}^n$ such that $\zeta(0) = s$, $\zeta(\tau) = s'$, and

$$\zeta(t) = s + \int_{\tilde{h}=0}^{t} f(x(\tilde{h}), \boldsymbol{u}(\tilde{h}))d\tilde{h} \tag{13}$$

for all $t \in [0, \tau]$ where $\boldsymbol{u} \in U[0, \tau] \subseteq Act$
(iv) Π is the set of atomic propositions on S
(v) $L : S \rightarrow 2^{\Pi}$ is the labeling function

We note that both TS_{c} and TS_{d} have infinitely many actions and states. For some linear systems, there are efficient control-based methods to directly work with the continuous-time dynamics without converting it to a discrete-time counterpart. There also exist analytic solutions for some systems, such as the double integrator model. However, for a general dynamical system, we often need to create many finite abstractions and plan with the abstract model. In theory, such abstractions should fulfill the so-called *simulation relationship*, i.e. the abstract system can be simulated by the original system. In other words, the abstractions can be created in such a way that as long as we can find a

feasible solution in the abstract system, we can always find a corresponding continuous and feasible trajectory in the original system.

We formally define the simulation relationship between two transition systems as follows:

Definition 2 (Simulation relation). Let

$$\mathrm{TS}_i = (S_i, S_i^0, \rightarrow_i, \Pi, L_i), \quad i = 1, 2 \tag{14}$$

be two transition systems. We note that these two transition systems have the same set of atomic proposition Π. A relationship $R \subset S_1 \times S_2$ is said to be a *simulation relationship* from TS_1 to TS_2 if the following properties hold:

(i) For any pair $(s_1, s_2) \in R$, we have $L_1(s_1) = L_2(s_2)$
(ii) For any initial state $s_1 \in S_1^0$, their exists $s_2 \in S_2^0$ such that $(s_1, s_2) \in R$
(iii) For any pair $(s_1, s_2) \in R$, if $s_1' \in \mathtt{post}(s_1)$ in TS_1 then there exists $s_2' \in \mathtt{post}(s_2)$ in TS_2 and $(s_1', s_2') \in R$

If there exists a simulation relationship R from TS_1 to TS_2, we say that TS_1 is simulated by TS_2, which is denoted as $\mathrm{TS}_1 \prec_R \mathrm{TS}_2$. In other words, for any trace in TS_1, we can find a corresponding equivalent trace in TS_2. Relation R is said to be a *bi-simulation relation* between TS_1 and TS_2 if $\mathrm{TS}_1 \prec_R \mathrm{TS}_2$ and $\mathrm{TS}_2 \prec_{R^{-1}} \mathrm{TS}_1$, which is also denoted as $\mathrm{TS}_1 \cong \mathrm{TS}_2$.

The main challenge of this process is that we are not always able to find a correct abstraction model that guarantees the original system to have a corresponding solution, especially when handling uncertain environments. One possible solution is to handle the problem by reactive synthesis. The idea is to consider all possible environment changes beforehand and generate a policy, which the robot reacts according to, during the online execution. Normally, the reactive synthesis problem is reformulated into a two-player game where the robot and environment take turns to play. Provided that the environment plays fairly, the policy generated guarantees correct behaviors. The logic specification into the GR(1) formulae (Definition 5) is often restricted since there are efficient solutions available.

Nevertheless, in some applications it is simply impossible and unnecessary to consider all the possible environmental changes. As such, some iterative planning techniques are often the preferred solution. The basic idea is to start with a guessed abstraction. During the execution, the robot will sense the environment, update the abstraction repeatedly and then perform re-planning when necessary. The robot is often assuming that the unknown space is obstacle-free and calculates a motion plan based on such an assumption. When the robot discovers that the reality differs from this assumption, it will modify the environment map and perform re-planning based on the updated information.

2.4. *Temporal task specifications*

Task requirements, i.e. the desired behaviors of the robot, can be viewed as a set of constraints over the system, which has to be formally described in a precise and unambiguous way. There are various different approaches to formalize these system properties. For complex requirements, temporal logics are often used to formally describe the task specifications. Temporal logics can be regarded as extended Boolean logics with temporal operators, which allow the user to reason the temporal changes of the propositions. There are various types of temporal logics, including but not limited to discrete logics, metric logics, and probabilistic logics. In this subsection, we focus only on the linear temporal logics and their metric variants. Besides temporal logics, it is also possible to use the automaton to describe the desired behaviors, which will be presented in the next subsection together with the relationship between the temporal logics and the automaton, as well as some commonly used examples.

2.4.1. *Temporal logics*

Linear temporal logic (LTL) is one of the most common discrete logics used to formally specify linear time properties. The basic concept of LTL and its important fragments, timed variants and specification patterns are introduced below.

Definition 3 (LTL syntax). Let Π be a set of atomic propositions. An LTL formula is defined over Π in accordance with the following grammar:

$$\begin{aligned} \phi ::= {} & \mathsf{true} && (\text{True}) \\ \mid {} & a && (a \in \Pi) \\ \mid {} & \neg\phi && (\text{Negation}) \\ \mid {} & \phi_1 \vee \phi_2 && (\text{Disjunction}) \\ \mid {} & \bigcirc\phi && (\text{Next}) \\ \mid {} & \phi_1 \,\mathsf{U}\, \phi_2 && (\text{Until}). \end{aligned} \tag{15}$$

Here we note that the Boolean operator $\neg$ represents the standard negation and $\vee$ represents the usual disjunction. From disjunction and negation, we can further define other Boolean operators such as the conjunction ($\wedge$), implication ($\Rightarrow$) and equivalence ($\Leftrightarrow$). Besides the Boolean operators, there are also temporal operators. The operator $\bigcirc$ represents the temporal operator Next and U denotes temporal operator Until. Other temporal operators can be further defined based on the Next and Until operator. The temporal operator Eventually is defined as $\Diamond\phi := \mathsf{true}\ \mathsf{U}\ \phi$. The temporal operator Always is defined as $\Box\phi := \neg\Diamond\neg\phi$.

The intuitive meaning of some commonly used temporal modalities are illustrated in Fig. 9.

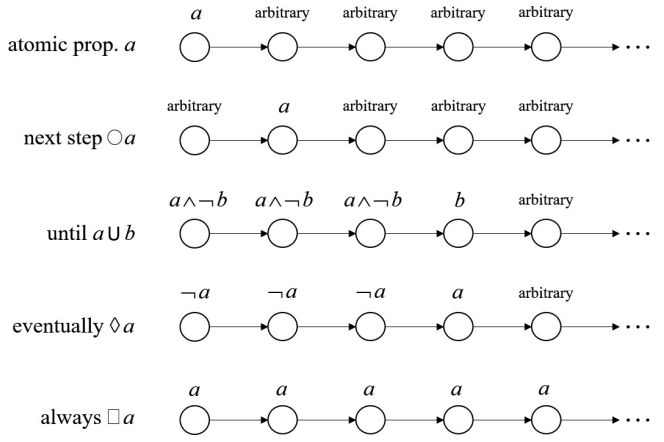


Fig. 9. The intuitive meanings of some temporal modalities.

Definition 4 (LTL semantics). The semantics of an LTL formula ϕ is interpreted over a linear structure. Let σ denote an infinite sequence of truth assignments to the proposition $a \in \Pi$ and $\sigma(i)$ denote the set of propositions that are true in position i. For an interpretation σ, we recursively define when an LTL formula ϕ is true at an instant i (written as $\sigma, i \models \phi$):

(i) $\sigma, i \models a$ iff $a \in \sigma(i)$
(ii) $\sigma, i \models \neg\phi$, iff $\sigma, i \not\models \phi$
(iii) $\sigma, i \models \phi_1 \vee \phi_2$, iff $\sigma, i \models \phi_1$ or $\sigma, i \models \phi_2$
(iv) $\sigma, i \models \bigcirc\phi$ iff $\sigma, i+1 \models \phi$
(v) $\sigma, i \models \phi_1 \, \mathsf{U} \, \phi_2$, there exist $k \geq i$ such that $\sigma, k \models \phi_2$ and for all $i \leq j < k$, we have $\sigma, j \models \phi_1$

Besides the full set of LTL, there is also interest in fragments of LTL for the sake of higher computational efficiency. Fragments of LTL are normally less expressive than the full set of LTL, but there are more efficiently specialized algorithms that handle such fragments. One important fragment of LTL is called GR(1) (the general reactivity of rank 1) fragment [29, 30]. GR(1) is frequently used in reactive synthesis, in which a whole system consists of some uncontrollable states, such as the states of the environment. The controllable plant then needs to react to the environmental changes.

For instance, a user may issue a return home command to a robot or an unmanned vehicle. Although the vehicle needs to return home after receiving the command, the situation may not always be controllable by the vehicle. One of the major challenges of reactive synthesis is its high worst-case complexity. For LTL, the complexity is the double exponential of the length of the formula. To handle such issues, [29] proposed a subset of LTL formulae, i.e. GR(1), which has a polynomial-time synthesis algorithm. The basic intuition behind GR(1) is that, if the environment satisfies some assumptions, the controlled system then has to satisfy the guarantees.

Definition 5 (GR(1) fragment). Let $\Pi = \Pi_X \bigcup \Pi_Y$, where Π_X is the set of propositions defined over the environment states and Π_Y is the set of propositions defined over the robot states. The GR(1) formulae are in the form of $\varphi = (\varphi_e \rightarrow \varphi_s)$, where φ_e describes the behavior of the environment (i.e. the assumption about the environment) and φ_s describes the behavior of the robot. The formula φ_e takes the form of

$$\varphi_e = \theta_e \wedge \Box\rho_e \wedge \bigwedge_{0<i\leq j} \Box\Diamond\phi_e^i.$$

Similarly, the formula φ_s takes the form of

$$\varphi_s = \theta_s \wedge \Box\rho_s \wedge \bigwedge_{0<i\leq j} \Box\Diamond\phi_s^i.$$

Thus, a GR(1) formula can be expressed as

$$\left(\theta_e \wedge \Box\rho_e \wedge \bigwedge_{0<i\leq j} \Box\Diamond\phi_e^i\right) \rightarrow \left(\theta_s \wedge \Box\rho_s \wedge \bigwedge_{0<i\leq j} \Box\Diamond\phi_s^i\right). \tag{16}$$

Here, we note that θ_e and θ_s are the constraints of the initial states, ρ_e and ρ_s are the safety properties, and ϕ_e and ϕ_s are the justice properties (i.e. the assertions that need to hold infinitely often) over the environment propositions Π_X and robot propositions Π_Y, respectively. The formula φ_e can simply be true, meaning that we do not put any assumptions on the environment.

Besides GR(1), another important fragment is the syntactically co-safe LTL (scLTL) [31].

Definition 6 (scLTL syntax). Let Π denote a set of atomic propositions. The scLTL formula is defined over Π in accordance with the following grammar:

$$\phi ::= \mathsf{true} \mid a \mid \neg a \mid \phi_1 \vee \phi_2 \mid \bigcirc\phi \mid \phi_1 \mathsf{U} \phi_2, \tag{17}$$

where $a \in \Pi$. We note that the key difference is that the negation $\neg$ can only appear in front of a proposition. As such, the temporal operator always $\Box$ is not a part of the scLTL formula ($\Box\phi := \neg\Diamond\neg\phi$ requires a negation of a general subformula). Though the scLTL formulas are interpreted over infinite words, it is possible to just check for a finite good prefix if the infinite word contains this good prefix.

2.4.2. *Timed temporal logics*

Though LTL is a powerful tool to specify linear time properties, it can only qualitatively describe temporal relationships. In other words, LTL can describe the ordering of events but not the precise time intervals between the events such as the bounded response time. For example, one may want to specify the deadlines between events and

response: *when a system receives a request, it has to respond within 5 time units*. Such requirements cannot be properly captured in LTL. To address this limitation, some timed specification languages are needed. The survey paper by Bouyer *et al.* [32] provides more details and further discussion on the different types of timed temporal logics and algorithms. In this section, we introduce the concept of metric temporal logic (MTL) and its important variants.

Metric temporal logic (MTL) [33] is the most widely known extension of the LTL. Here, T denotes the time domain, which is a set of non-negative real numbers. A time interval I is a nonempty convex subset in the time domain T. An interval I where $\sup(I)$ exists is called a bounded interval. Let $|I| = \sup(I) - \inf(I)$, and I and J be two intervals over the time domain T, the Minkowski sum of two intervals is defined as

$$I \oplus J := \{r + s \in T | r \in I, s \in J\}$$

which is illustrated in Table 2. Similarly, the Minkowski difference of two intervals is given by

$$I \ominus J := \{r - s \in T | r \in I, s \in J\}.$$

The syntax and semantics of MTL are defined as follows.

Definition 7 (MTL syntax). Let Π denote a set of atomic propositions. Let U_I be the time-constrained versions of the until operator U, where $I \subseteq (0, \infty)$ is a time interval with endpoints in $\mathbb{Q}_{\geq 0} \cup \{\infty\}$. The unconstrained until operator U is essentially $\mathsf{U}_{(0,\infty)}$. The MTL formulae over Π are constructed recursively as

$$\phi ::= \mathsf{true} | a | \neg\phi | \phi_1 \wedge \phi_2 | \phi_1 \ \mathsf{U}_I \ \phi_2, \tag{18}$$

where $a \in \Pi$. The temporal operator constrained eventually is then defined as $\Diamond_I \phi := \mathsf{true} \ \mathsf{U}_I \ \phi$ and the operator constrained always is defined as $\Box_I := \neg\Diamond_I \neg\phi$.

There are two types of semantics for the MTL: the continuous setting, and the point-wise continuous setting. For continuous semantics, the executions of the system are viewed as a *signal*. For point-wise semantics, the executions are discrete and viewed as a *timed word*.

Definition 8 (Signal). Let Π be a set of atomic propositions. A *signal* over Π is a function $\gamma(t) : \mathbb{R}_+ \to 2^\Pi$ which maps $t \in \mathbb{R}_+$ to a set of propositions $\gamma(t)$ holding at time t.

Definition 9 (MTL continuous semantics). Given a signal γ over 2^Π, $t \in \mathbb{R}_+$, and a formula ϕ, the satisfaction relation $\gamma, t \models \phi$ (whether the signal γ satisfies the formula ϕ at time t) is recursively defined as:

(i) $\gamma, t \models a$ iff $a \in \gamma(t)$
(ii) $\gamma, t \models \neg\phi$, iff $\gamma, i \not\models \phi$
(iii) $\gamma, t \models \phi_1 \wedge \phi_2$, iff $\gamma, i \models \phi_1$ and $\gamma, t \models \phi_2$
(iv) $\gamma, t \models \phi_1 \mathcal{U}_I \phi_2$, iff $\exists t' \in t \oplus I$, $\gamma, t' \models \phi_2$ and $\forall t'' \in (t, t'), \gamma, t'' \models \phi_1$

Definition 10 (Time sequence). A *time sequence* $\tau = \tau_0, \tau_1, \tau_2, \ldots$ is either a finite or infinite sequence of time values, which satisfies the following conditions:

(i) $\tau_i \in \mathbb{R}_+$, and $\tau_0 = 0$
(ii) $\tau_i \leq \tau_{i+1}$, $\forall i < |\tau| - 1$
(iii) If τ_i is infinite, then $\{\tau_i : i \in \mathbb{N}\}$ is unbounded

Definition 11 (Timed word). Let Σ be a finite alphabet set. A *timed word* ρ over Σ is a (either finite or infinite) sequence $\rho = (\sigma_0, \tau_0), (\sigma_1, \tau_1), (\sigma_2, \tau_2), \ldots,$ where $\sigma_i \in \Sigma$ and $\tau = \tau_0, \tau_1, \tau_2, \ldots$ is the time sequence.

Definition 12 (MTL point-wise semantics). In point-wise semantics, the MTL is interpreted over a *timed word*. Let Π be a set of atomic propositions. Given a timed word $\rho = (\sigma, \tau)$ over 2^Π and an MTL formula ϕ, the satisfaction relation $\rho, i \models \phi$ (i.e. formula ϕ is true at instant i) is defined with the standard Boolean operators and constrained until operator:

(i) $\rho, i \models a$ iff $a \in \sigma_i$,
(ii) $\rho, i \models \neg\phi$, iff $\rho, i \not\models \phi$,
(iii) $\rho, i \models \phi_1 \vee \phi_2$, iff $\rho, i \models \phi_1$ or $\rho, i \models \phi_2$,
(iv) $\rho, i \models \phi_1 \ \mathsf{U}_I \ \phi_2$, iff there exist k such that $i \leq k < |\rho|$, $\sigma, k \models \phi_2$, $\tau_k - \tau_i \in I$ and for all $i \leq j < k$, we have $\sigma, j \models \phi_1$.

We should note that the model checking of the MTL is nondecidable over the infinite continuous semantics [34]. To address this issue, a metric interval temporal logic (MITL), which is a fragment of the MTL, was proposed in [34]. Unlike MTL, the MITL does not allow punctuality, i.e. the singular time interval for temporal operators is not allowed in MITL.

Definition 13 (MITL fragment). The metric interval temporal logic (MITL) is a fragment of the MTL, where the time interval $I \subseteq (0, \infty)$ is nonsingular with rational end-points.

There are also many other variants of the MTL such as the signal temporal logic (STL) [35], which is a special extension (real-valued signals) of the bounded subset of the MITL, i.e. $\mathrm{MITL}_{[a,b]}$.

2.4.3. *Specification patterns for LTL*

Formal specification languages, such as temporal logics, are precise ways to specify the system requirements [36]. However, writing specifications still require a certain level of expertise and thus remain challenging for most users who are not familiar with these formal languages. To

address this issue, [36] and [37] created a library of specification patterns by providing a set of commonly used properties for several different languages, which assists the users in effectively translating the descriptions of requirements into a formal manner. Here, we briefly review the specification patterns for LTL.

In general, there are two major types of patterns: occurrence patterns and order patterns. Occurrence patterns are concerned with the occurrence (such as existence) of given events or states during the system execution. There are four types of occurrence patterns, namely, absence, universality, existence and bounded existence. On the other hand, the ordered patterns are concerned with the order related to the pairs of states or events during a system execution. The two basic order patterns are precedence and response. These specification patterns are summarized in Tables 3–7.

Each pattern has a scope defining the duration of execution that the pattern must hold. There are five basic types of scopes. First, the *global* scope means that the pattern has to hold over the entire program execution. The *before* scope means that the pattern must hold over the execution period up to a given state or event. Conversely, the *after* scope implies that the pattern must hold over the execution after a given state or event. The *between* scope means that the pattern must hold over any part of the execution from one given state or event to another. Last, the *after-until* scope is similar to the *between* scope, but the execution will continue even if the second state or event does not occur. The scope is essentially a left-closed and half-open interval.

Table 2. Minkowski sum of two intervals.

$\oplus$	$[c$	$(c$	$\oplus$	$d)$	$d]$
$[a$	$[a+c$	$(a+c$	$b)$	$b+d)$	$b+d)$
$(a$	$(a+c$	$(a+c$	$b]$	$b+d)$	$b+d]$

Table 3. Specification patterns of *Absence* for LTL.

	Absence (P is false)
Globally	$\Box(\neg P)$
Before R	$\Diamond R \rightarrow (\neg P \mathcal{U} R)$
After Q	$\Box(Q \rightarrow \Box(\neg P))$
Between Q and R	$\Box((Q \wedge \neg R \wedge \Diamond R) \rightarrow (\neg P \mathcal{U} R))$
After Q until R	$\Box(Q \wedge \neg R \rightarrow (\neg P \mathcal{W} R))$

Table 4. Specification patterns of *Existence* for LTL.

	Existence (P becomes true)
Globally	$\Diamond P$
Before R	$\neg R \mathcal{W}(P \wedge \neg R)$
After Q	$(\Box(\neg Q)) \vee (\Diamond(Q \wedge \Diamond P))$
Between Q and R	$\Box((Q \wedge \neg R) \rightarrow (\neg R \mathcal{W}(P \wedge \neg R)))$
After Q until R	$\Box((Q \wedge \neg R) \rightarrow (\neg R \mathcal{U}(P \wedge \neg R)))$

Table 5. Specification patterns of *Universality* for LTL.

	Universality (P is true)
Globally	$\Box P$
Before R	$\Diamond R \rightarrow (P \mathcal{U} R)$
After Q	$\Box(Q \rightarrow \Box P)$
Between Q and R	$\Box((Q \wedge \neg R \wedge \Diamond R) \rightarrow (P \mathcal{U} R))$
After Q until R	$\Box((Q \wedge \neg R) \rightarrow (P \mathcal{W} R))$

Table 6. Specification patterns of *Precedence* for LTL.

	Precedence (S precedes P)
Globally	$\neg P \mathcal{W} S$
Before R	$\Diamond R \rightarrow (\neg P \mathcal{U}(S \vee R))$
After Q	$(\Box \neg Q) \vee (\Diamond(Q \wedge (\neg P \mathcal{W} S)))$
Between Q and R	$\Box((Q \wedge \neg R \wedge \Diamond R) \rightarrow (\neg P \mathcal{U}(S \vee R)))$
After Q until R	$\Box((Q \wedge \neg R) \rightarrow (\neg P \mathcal{W}(S \vee R)))$

Table 7. Specification patterns of *Response* for LTL.

	Response (S responds to P)
Globally	$\Box(P \rightarrow \Diamond S)$
Before R	$\Diamond R \rightarrow (P \rightarrow (\neg R \mathcal{U}(S \wedge \neg R))) \mathcal{U} R$
After Q	$\Box(Q \rightarrow \Box(P \rightarrow \Diamond S))$
Between Q and R	$\Box((Q \wedge \neg R \wedge \Diamond R) \rightarrow (\neg P \mathcal{U}(S \vee R)))$
After Q until R	$\Box((Q \wedge \neg R) \rightarrow ((P \rightarrow (\neg R \mathcal{U}(S \wedge \neg R))) \mathcal{W} R))$

2.5. *Automata*

Automata are alternative mathematical models used to describe the behaviors of the system [38]. We can use an automaton to accept the languages represented by temporal logic. In this section, we introduce the automaton on finite words (i.e. the finite state automaton) and on infinite words.

Definition 14 (Finite state automaton). A finite state automaton (FSA) is a tuple $\mathcal{A} = (Z, Z_0, \Sigma, \rightarrow_{\mathcal{A}}, Z_F)$, where

(i) Z is a finite set of states
(ii) Z_0 is a set of initial states

(iii) $\Sigma = 2^{\Pi}$ is the input alphabet with Π being the atomic proposition set
(iv) $\rightarrow_{\mathcal{A}} \subseteq Z \times \Sigma \times Z$ is a nondeterministic transition relation
(v) $Z_F \subseteq Z$ is a set of accepting states

Let Σ^* denote the set consisting of all finite words over Σ. The semantics of an FSA are defined over *finite* input words in Σ^*. Let $w = w_1 w_2 \dots w_n$ denote the *input word* of the FSA. A *run* of the automaton *over* an input word w is a sequence of $r_{\mathcal{A}} = z_0, z_1, z_2, \dots, z_n$ such that $z_0 \in Z_0$, and $(z_i, w_i, z_{i+1}) \in \rightarrow_{\mathcal{A}}$ for $i = 0, 1, 2, \dots, n-1$. The word w is accepted by $\mathcal{A}$ if and only if the corresponding run ends in a final automaton state, i.e. $z_n \in Z_F$. The accepted language of $\mathcal{A}$, denoted $\mathcal{L}(\mathcal{A})$, is the set of finite words in Σ^* accepted by $\mathcal{A}$, i.e.

$$\mathcal{L}(\mathcal{A}) = \{w \in \Sigma^* | \exists \text{ an accepting run for } w \in \mathcal{A}\}.$$

It is worth noting that an scLTL formula can always be translated into a deterministic finite automaton.

The FSA accepts finite words, i.e. sequences of symbols of finite length, and can be used to check the regular safety properties. We now present the nondeterministic Büchi automata (NBA), which is a variant of the FSA and accepts infinite words. The NBA can be used to check a wider range of linear time properties.

There is a closed relationship between LTL and the Büchi automaton as it has been proven that any LTL formula can be translated into an equivalent nondeterministic Büchi automaton [39] which accepts the traces satisfying the LTL formula.

Definition 15 (Büchi automaton). A Büchi automaton is a tuple $\mathcal{BA}_{\phi} = (Z, Z_0, \Sigma, \rightarrow_{\mathcal{BA}}, Z_F)$, where

(i) Z is a finite set of states
(ii) Z_0 is a set of initial states
(iii) $\Sigma = 2^{\Pi}$ is the input alphabet with Π being the set of atomic propositions
(iv) $\rightarrow_{\mathcal{BA}} \subseteq Z \times \Sigma \times Z$ is a nondeterministic transition relation
(v) $Z_F \subseteq Z$ is a set of accepting states

Let $w = w_0 w_1 w_2 \dots$ denote the *input word* of the automaton. A *run* of the automaton *over* an input word w is a sequence of $r_{\mathcal{BA}} = z_0, z_1, z_2, \dots$ such that $z_0 \in Z_0$, and $(z_i, w_i, z_{i+1}) \in \rightarrow_{\mathcal{BA}}$ for $i \in \mathbb{N}$. A run r is accepted if and only if $lim(r) \cap Z_F \neq \emptyset$, where $lim(r)$ is the set of states that occur in r infinitely often, i.e. the run is accepted if and only if it gets into Z_F infinitely many times. In other words, at least one accepting state has to be visited infinitely often. This acceptance condition is called the *Büchi acceptance*.

2.6. *Examples of specifications*

We first present some typical control specifications using LTL. Assuming we have a set of atomic propositions

$$\Pi = \{\pi_A, \pi_B, \pi_C\}, \tag{19}$$

where π_i indicates whether the vehicle is in region i, $i \in \{A, B, C\}$. Here, we will first consider the following typical specification types. More complicated examples can be found in Sec. 2.4.3.

Reach targets while avoiding obstacles

$$\neg\pi_A \wedge \neg\pi_B \vee \pi_C$$

This specification means that "*the vehicle should always avoid the regions A and B, and go to region C*".

Sequencing

$$\Diamond(\pi_A \Diamond(\pi_B \Diamond \pi_C))$$

This specification means that "*the vehicle should eventually visit A, B, C in a sequential order*".

Coverage

$$\Diamond\pi_A \wedge \Diamond\pi_B \wedge \Diamond\pi_C$$

This specification means that "*the vehicle should eventually visit A, B and C*".

Recurrence

$$\Box(\Diamond\pi_A \wedge \Diamond\pi_B \wedge \Diamond\pi_C)$$

This specification means that "*the vehicle should visit A, B and C infinitely often*".

We now revisit the parcel delivery example shown in Fig. 5 and see how different temporal logics can be used to specify the same task. Given an abstract workspace, we can create a set of atomic propositions such as

$$\begin{aligned}\Pi = \{&Warehouse, SiteA, SiteB, SiteC,\\ &ForbiddenAera1, ForbiddenAera2, ForbiddenAera3,\\ &Pickup, Deliver, GetNewRequest\}.\end{aligned} \tag{20}$$

Nonreactive co-safe LTL

$$(\neg Deliver)\ \mathcal{U}\ (Deliver \wedge (SiteB \vee SiteC))$$

This specification means that "*the vehicle should deliver a parcel to either site B or site C*".

GR (1)

$$\Box\Diamond(GetNewRequest) \rightarrow (Deliver \wedge (SiteB \vee SiteC))$$

This specification means that "*if the UAV receives a new delivery request, then deliver the parcel to either site B or site C*".

Fig. 10. An example of an automaton.

MITL

$$\Diamond_{[0,10]}(Deliver \wedge (SiteB \vee SiteC))$$

This specification means "*deliver the parcel to either site B or site C within 10 time units*".

Automaton

The automaton in Fig. 10 is converted from co-safe LTL, meaning that "*the vehicle should deliver a parcel to either site B or site C*".

3. Motion Planning Techniques

This section provides a review of advanced techniques related to motion planning for small-scale multicopters. The earlier works in this area typically used techniques from the guidance and control community, with examples of such classical approaches including path following, which is originally designed for large fixed-wing aerial vehicles [40, 41]. The basic idea of path following consists of two steps. The first step is to generate a pure geometric path as the reference. The next step is to directly design a control law to handle the dynamic constraints of the vehicle so that it follows the reference path as closely as possible. While the reference can be easily synthesized using line segments and splines, it is challenging to design such a control law for multicopters due to the complex dynamics and state constraints. Methods such as those presented in [40, 41] obtain control laws by simplifying the dynamic model of the aircraft and/or the constraints at the expense of tracking accuracy. Unfortunately, small-scale multicopters normally operate at low-altitude and in obstacle-dense environments which often require greater tracking accuracy.

To address this issue, researchers borrowed ideas from the robotics research community and proposed a trajectory tracking approach [24, 42]. A trajectory is a path with an associated timing function. With the ability to represent time information, a trajectory can incorporate both the geometric and vehicle dynamics constraints. Instead of directly sending a geometric path to a low-level controller, a collision-free and dynamically feasible trajectory is generated and sent as the reference signal to the flight control system. In this way, a simple linear control law would yield very high tracking accuracy. By tracking a trajectory instead of a geometric path, we are able to leverage the full vehicle dynamics and develop a much more aggressive flight plan. As a result, trajectory tracking soon became a dominant approach in the rotorcraft motion planning community.

The main purpose of motion planning for small-scale multicopters is to compute a trajectory to navigate the vehicle from an initial point to a target destination that is both dynamically feasible and collision-free. Following [43], we define the trajectory as a time-dependent function or mapping $\pi(t) : [0, T] \to \mathcal{X}$, where $\mathcal{X}$ is the configuration space of the vehicle. Let $\Pi(\mathcal{X}, T)$ denote the set of all continuous functions $[0, T] \to \mathcal{X}$, x_{init} denote the initial configuration and $\mathcal{X}_{\text{free}}(t) \subseteq \mathcal{X}$ denote all collision-free configurations at time t, where $t \in [0, T]$. The goal region is denoted as $\mathcal{X}_{\text{goal}} \subset \mathcal{X}$. The differential constraints are represented by a predicate $D(\pi(t), \pi'(t), \ldots)$, and the cost function is denoted as $J(\pi) : \Pi(\mathcal{X}, T) \to \mathbb{R}$. The optimal trajectory generation problem can then be stated as follows.

Problem 3.1. Given $\mathcal{X}_{\text{free}}, x_{\text{init}}, \mathcal{X}_{\text{goal}}, D$ and J, find an optimal solution π^* subject to the following optimization:

$$\begin{aligned} & \texttt{argmin} \quad J(\pi) \\ \text{subj. to} \quad & \pi(0) = x_{\text{init}}, \quad \pi(T) \in \mathcal{X}_{\text{goal}} \\ & \pi(t) \in \mathcal{X}_{\text{free}}, \quad \forall t \in [0, T] \\ & D(\pi(t), \pi'(t), \ldots), \quad \forall t \in [0, T]. \end{aligned}$$

Due to the presence of obstacles and the high-order nature of multicopter dynamics, the problem is nonconvex in nature and hence difficult to solve. Furthermore, small-scale multicopters are often restricted in terms of onboard computational power as the amount of payload it is able to carry is limited. This adds to the challenge of real-time onboard motion planning.

As a result, for computational efficiency, geometric and dynamic constraints are often either handled in different planning layers or the entire motion planning problem is decoupled into a global geometric path planning phase and a local trajectory generation phase. In the path planning phase, a simplified first-order model of the multicopter is used to search for a continuous geometric path that connects the initial and goal position. Then in the trajectory generation phase, this path is used either to simplify the obstacle free constraint $\pi(t) \in \mathcal{X}_{\text{free}}$ [27, 44, 45] or as an initial guess to start the gradient descent process [46, 47].

In the rest of this section, as illustrated in Fig. 4, we will first present the pure geometric path planning problem, and then discuss common techniques that are used to generate dynamically feasible trajectories. We will also highlight techniques for the integration of these two planning layers to end the motion planning portion of our survey.

3.1. *Path planning*

The path of a vehicle can be defined as a function $\sigma(\alpha) : [0, 1] \to X$, where X is the configuration space of the

vehicle. The path planning problem is to find a collision-free path that starts from the initial configuration to a target region while satisfying the given constraints. A path solution can be either *feasible* or *optimal* with respect to certain criterion.

Let x_{init} denote the initial configuration and $X_{\text{free}} \subseteq X$ denote all collision-free configurations. The target or goal region is denoted as $X_{\text{goal}} \subset X$. For an unmanned vehicle, there are also a set of differential constraints over the path. For instance, the path of a fixed-wing aircraft should have a minimal curvature, which can be represented by the predicate $D(\sigma(\alpha), \sigma'(\alpha), \ldots)$. Let $\Sigma(X)$ denote the set of all continuous functions $[0, 1] \to X$ and let $J(\sigma) : \Sigma(X) \to \mathbb{R}$ denote the cost function. The optimal path planning problem can be formulated as follows.

Problem 3.2 (Optimal path planning). Given X_{free}, x_{init}, X_{goal}, D and J, the optimal path planning problem is to find an optimal path σ^* such that

$$\begin{aligned} & \texttt{argmin} \quad J(\sigma) \\ \text{subj. to} \quad & \sigma(0) = x_{\text{init}}, \quad \sigma(1) \in X_{\text{goal}} \\ & \sigma(\alpha) \in X_{\text{free}} \quad \forall\, \alpha \in [0, 1] \\ & D(\sigma(\alpha), \sigma'(\alpha), \ldots) \quad \forall\, \alpha \in [0, 1]. \end{aligned}$$

Since multicopters are holonomic, path planning algorithms usually adopt a polyline path model, i.e. the path is described by a set of connected line segments. There are two major approaches to discretizing a continuous configuration space which relax the related problem to different levels of completeness. The first approach is to relax it into a resolution complete problem. The basic idea is to discretize the space into grids, and then use traditional graph search algorithms to search through the grid map. The second approach is to utilize a sampling-based planning strategy, which is probabilistically complete instead of resolution complete.

3.1.1. *Traditional grid-based graph search*

Once the environment map has been discretized into a grid, we can apply any graph search algorithm over the grid and find a path from the initial graph node to the target node. Dijkstra's algorithm [48] is one of the most commonly used optimal graph search methods. It performs the best first search over the entire graph and is able to find the shortest path from the given initial node to all other nodes. Another well-known graph search algorithm is the A* [49], which uses a heuristic function to guide the search and thus boosts computational efficiency. The resulting search performance is also dependent on the heuristic function. Both Dijkstra and A* algorithms run over a static graph. To handle dynamic graph changes, there are many variants of the original A* algorithm, such as D*, Focused D*, and D*-Lite [50, 51].

For robotics, robots often work in dynamic environments, where environmental obstacles are updated using onboard sensor data. Iterative planning strategies are often adopted to handle these dynamic obstacles. For each planning cycle, the environment is considered a static one and a fresh new path is calculated from scratch. A new planning cycle can either be triggered by a fixed time interval or by any environmental change that violates the current plan. In many applications, the environmental changes often occur on a small part of the map. Hence to improve search efficiency, it is possible to re-use some information from the previous search cycle instead of performing an entirely new one from scratch. Real-time re-planning algorithms such as the D*, Focused D*, and D*-Lite are thus developed to handle this issue. These real-time algorithms enhance the re-planning performance by focusing on the propagation of the updated environment information.

Furthermore, there are also anytime-search variants of the A*, such as the anytime A* [52] and anytime repairing A* (ARA*) [53]. The anytime-search algorithms aim to quickly find an initial sub-optimal solution and attempt to improve the quality of the solution over a further search process. The anytime-search algorithms are very useful in the field of robotics since a robot usually does not need to compute the entire optimal trajectory before its execution. The anytime planner is able to continuously optimize the trajectory when the robot is executing partially sub-optimal trajectories from time to time. For example, the anytime A* algorithm [52] can find an initial solution quickly and use its cost as the upper-bound of the optimal solution while the lower bound remains as admissible heuristics.

3.1.2. *Sampling-based methods*

The major limitation of the above methods, i.e. discretizing the continuous space into a grid and applying classical graph search algorithms to search for a suitable path, is the curse of dimensionality. Graph search algorithms generally do not scale well with the dimension of the search space. To address this issue, researchers propose to use sampling-based techniques to abstract continuous space and conduct the search over an implicit graph constructed by the samples. The sampling-based planner is also normally integrated with an extra collision detection module which is used to examine whether a graph edge is collision-free. Such a collision checking module is independent of the planner itself. Unlike grid-based graph search algorithms, which provide resolution completeness, most sampling-based planners provide *probabilistic completeness*. In other words, as the search time increases, the probability of finding a valid solution goes to one. Sampling-based planners have been shown to work well in practice and scale much better

than traditional graph search planners when handling high dimensional state spaces.

Pioneer works in sampling-based planning include techniques such as rapidly-exploring random trees (RRTs) [54], expansive-spaces tree planner (EST) [55], and probabilistic roadmaps (PRM) [56]. A formal analysis of the quality of solutions found by sampling-based planners was first presented in [57]. This work is a major milestone in the development of sampling-based planners as it formally connects the sampling-based planning algorithms with the theory of random geometric graphs, along with proposing several optimal variants of the pioneer sampling-based planners such as RRT* and PRM*. In recent years, inspired by the work in [57], researchers have re-visited traditional graph search algorithms and successfully integrated graph search techniques such as heuristics and anytime planning with traditional sampling-based planners.

(i) Basic structure of sampling-based planners

Various sampling-based planning algorithms have been proposed to enhance convergence rates and computational efficiency. Though these algorithms have different features, they often share the same basic algorithmic structure. Here, we will present a basic outline of sampling-based algorithms based on the work in [58].

Any sampling-based planner has several main sub-procedures as shown in Fig. 11. Sampling-based planners essentially construct a graph using a set of samples and then search for a valid path over the graph, which can be incrementally constructed. Let (V^k, E^k) denote an underlying graph, where V^k represents the set of graph vertices and E^k is the set of graph edges associated with the kth iteration of the process. Let V^k_{sol} denote the solution (i.e. the path from the initial vertex to the goal vertex) at the kth iteration. During the initialization process (Lines 1–3 in Fig. 11), the graph contains only the root vertex that represents the initial state x_{init}. Note that we explicitly differentiate the state in the state space and corresponding vertex in the graph. A function stateToVertex is used to map the state to the vertex (Line 2 in Fig. 11). The procedure in Lines 4–11 in Fig. 11 describes the main body of the algorithm, i.e. the iteration process. The function noTermination is used to check for the stop condition. For instance, we can set a maximum number of iterations as a termination condition. Alternatively, we can also assign a timer to limit the process. We first sample a valid state point x_{sample} from the state space by calling the function sample. The variable x_{sample} is then converted to a graph vertex structure. The details of sample varies for different planners and will be discussed later.

```
1:  k ← 0;
2:  V^k ← stateToVertex(x_init);
3:  E^k ← ∅;
4:  V^k_sol ← ∅
5:  while noTermination() do
6:      x_samp ← sample()
7:      v_samp ← stateToVertex(x_samp)
8:      V_near ← findNeighbors(x_samp, bkwd)
9:      (v_new, e_new) ←
        connectToBestVertex(v_samp, V_near)
10:     V_near ← findNeighbors(x_samp, fwd)
11:     (V^{k+1}, E^{k+1}, V^{k+1}_sol) ←
        optimizeAndUpdateGraph(v_new, e_new, V_near)
12:     k++
    return convertToTraj(V^{k+1}_sol)
```

Fig. 11. The abstract skeleton of sampling-based planning algorithms.

We note that though the algorithm in Fig. 11 only samples one state in each iteration, it is also possible to generate a batch of samples. After generating a new sample vertex, we can find the neighbors of this new sample from existing vertices in the graph. More specifically, we search in the backward reachable area of the state x_{samp} for a given time horizon by calling the function findNeighbors. Then, the new sampled vertex is added to the existing graph by connecting it with the best vertex in the graph (a vertex is the best if it minimizes the cost-to-come value of v_{samp} if v_{samp} is propagated from this vertex). Since the new vertex is added into the graph, other graph vertices may have a better cost-to-come value if they are propagated from this newly added vertex. In other words, it is necessary to perform some rewiring to optimize the current graph. Normally, the rewiring process occurs amongst the neighbors of the newly added vertex. We thus proceed to find the neighbors of x_{samp} next within its forward reachable set and use the function optimizeAndUpdateGraph to optimize the current graph and update the current best solution V_{sol}. We can then go to the next iteration by calling the function convertToTraj. Lastly, the path solution V_{sol} in the graph is mapped to the state trajectory and the optimized trajectory is captured and returned after all iterations.

(ii) Sampling strategy

There are various sampling strategies, such as uniform sampling [56, 57], biased sampling [59, 60] and informed sampling [61], documented in the literature. Of these, the most commonly used is the uniform sampling approach, where a sample or a batch of samples are drawn uniformly from the state space. However, a large number of useless samples in uniform sampling may greatly reduce the convergence rate. As such, it is more efficient to generate only useful samples, i.e. samples that can directly contribute to the solution.

Biased sampling is a frequently used technique to generate useful samples, which essentially provides heuristic information to the planning algorithm. More specifically, some parts of the state space are assigned higher

probability to be sampled as compared to other parts. For example, considering an environment with a narrow passage, it is often necessary to generate a lot of samples and perform many iterations before useful samples that connect the narrow passage are generated. For such applications, biased sampling techniques such as obstacle-based sampling can be applied. The idea is to guide the sampling process to generate more samples around the obstacle region, and thus reduce the time needed by the planner to find the narrow passage. We note that uniform sampling is the key element that ensures asymptotic optimality and probabilistic completeness of the sampling-based planner, and biased sampling has to be handled with extra care.

Another type of sampling strategy is called informed sampling. Instead of sampling inside the whole state space, we can perform uniform sampling inside the reachable set of the system or subsets of the state space that are confirmed to contain a better solution. For instance, consider a pure geometric path planning case with a shorter-path optimization objective. As in informed RRT* [62], if the initial feasible path has been found between the initial and end state after some iterations, one can safely conclude that the better path can only be contained inside an ellipsoidal subset described by the initial feasible path. To generate samples inside this region of interest, one can either perform a direct sampling inside the ellipsoid or use rejection sampling to reject the samples that are outside the ellipsoid.

(iii) Branch and bound technique

A common technique used to improve the convergence rate of sampling-based planners is the branch and bound technique. The key idea is to discard any computations that cannot further improve the current solution in order to obtain higher computational efficiency. In the context of sampling-based planning, the branch and bound technique can be, for example, (1) informed sampling, (2) pruning the tree, (3) rejecting useless samples, or (4) propagating within a bounded region. Typical examples of applying such a technique can be found in [61–66].

Heuristics are also frequently used together with the branch and bound technique. The most common heuristic used in motion planning is the Euclidean distance from the current vertex toward the goal vertex. Intuitively, such a distance is the lower bound of the true trajectory. Assume we have a graph G, where $g_G(v)$ denotes the best cost-to-come of the vertex v. Let $\hat{g}(v)$ denote the admissible cost-to-come heuristic of the vertex v and $\hat{h}(v)$ denote the admissible heuristic of the cost-to-go of the vertex. The branch and bound technique can be represented as

$$\begin{aligned} g_G(v) + \hat{h}(v) &\geq g_G(v_{\text{goal}}), \\ \hat{g}(v) + \hat{h}(v) &\geq g_G(v_{\text{goal}}). \end{aligned} \tag{21}$$

(iv) Delayed or lazy computation

Another important technique used to boost the convergence rate is to avoid or delay more expensive computation. For instance, if we assume that collision-checking is computationally more expensive than the steering function, then we can minimize the number of collision-checking procedures [66–68] by delaying the computational process. The basic idea is to build up the graph first, assuming there is no geometric obstacles, and then order the edges and perform collision-checking individually. We can order the edges by some consistent heuristic and integrate it with the branch and bound technique. The idea is that if the previous edge is collision-free (the heuristic is equal to the true cost), then there is no need to process the rest of edges in the ordered list since their heuristics (lower bound) are already larger than the true cost of the previous edge.

3.2. *Trajectory generation*

In the trajectory generation phase, our aim is to solve the optimization problem shown in Problem 3.1 and obtain the trajectory $\pi(t)$ as reference for the lower level controllers. With the geometric path provided by the path planning phase, it can either be used as a collision-free initial guess, or to further simplify the obstacle-free constraint $\pi(t) \in \mathcal{X}_{\text{free}}$. More details on the initialization and simplification techniques are covered in Sec. 3.3. Therefore, in this section, we will focus on methods that generate dynamically feasible trajectories.

In most work on multicopter trajectory generation, the vehicle is considered as a high-order integrator as shown in Eq. (1). Moreover, the trajectory's higher-order derivatives, like acceleration and jerk, need to be constrained to satisfy the vehicle's input and state constraints (see Eqs. (5) and (7)).

With the vehicle treated as a high-order integrator, a popular approach of generating a dynamically feasible trajectory is to parameterize the trajectory with polynomials and splines, and solve the resulting optimization problem [24, 27, 47, 69]. These polynomials and splines can usually be differentiated analytically, therefore the high-order derivatives can be expressed without introducing extra programming variables. On the other hand, the trajectory generation problem can also be subdivided into multiple boundary value problems (BVPs), where the sub-problem can be solved analytically [70, 71] or where a fast approximated solution exists [72, 73].

3.2.1. *Spline-based numerical optimization*

A popular approach for generating a dynamically-feasible trajectory is to represent the trajectory using splines,

treating the dynamics as a set of constraints, and then formulating it as an optimization problem and solving it using a numerical solver. A general spline function can usually be expressed as

$$S_k(s) = \sum_{i=0}^{M-1} c_i N_i^k(s), \quad c_i, \quad S_k \in \mathcal{X}, \quad N_i^k \in \mathbb{R}, \tag{22}$$

where $\mathcal{X} \in \mathbb{R}^d$ is the d-dimensional work space with $d \in \mathbb{N}$, $S_k(s)$ represents the spline, k is the order of the spline, $N_i^k(s)$ is the ith base function and c_i is the corresponding weight vector. During the motion planning process, we select the best set of c_i to optimize a given cost function. An additional mapping is needed to connect the path parameter s to the time t, which can take the form of a simple linear relationship as shown in [74]:

$$\frac{s}{t} = \alpha \tag{23}$$

for an appropriate scalar α or in the form of a nonlinear function calculated through further optimization [75].

Common spline representations include polynomials [24, 69, 76] and B-splines [77, 78]. As an example, we briefly present the formulation from [24], which adopts piece-wise polynomials as the base function to approximate the trajectory. Let $\sigma(t)$ denote a polynomial of degree N, such that

$$\hat{\sigma}(t) = \hat{\sigma}_N t^N + \hat{\sigma}_{N-1} t^{N-1} + \cdots + \hat{\sigma}_0 = \sum_{i=0}^{N} \hat{\sigma}_i t^i. \tag{24}$$

The piece-wise polynomial of order N over m intervals is given as follows:

$$\sigma(t) = \begin{cases} \sum_{i=0}^{N} \sigma_{i1} t^i, & t_0 \le t < t_1, \\ \sum_{i=0}^{N} \sigma_{i2} t^i, & t_1 \le t < t_2, \\ \vdots \\ \sum_{i=0}^{N} \sigma_{im} t^i, & t_{m-1} \le t \le t_m. \end{cases} \tag{25}$$

The trajectory of the vehicle in the 3D space can be represented by three independent single-axis piece-wise polynomials. This will be denoted as $\boldsymbol{\sigma}(t) = [\sigma_x, \sigma_y, \sigma_z]^\top$.

To improve the overall smoothness of the trajectory, the optimization target is set to minimize the following cost function:

$$\begin{aligned} \min \quad & \int_{t=0}^{t=t_m} w_0 \boldsymbol{\sigma}(t)^2 + w_1 \boldsymbol{\sigma}'(t)^2 + \cdots + w_k \boldsymbol{\sigma}^k(t)^2 dt \\ s.t. \quad & \boldsymbol{\sigma}(t_i) = \boldsymbol{\sigma}_i, \quad i = 0, \ldots, m \\ & \boldsymbol{\sigma}(t) \quad \text{is of at least } C^3 \text{ continuity} \end{aligned} \tag{26}$$

in which the set of the additional linear constraints forces the trajectory and its derivatives to pass through a series of pre-defined key-frames at specific times. This optimization problem can be solved by an off-the-shelf quadratic programming (QP) solver.

3.2.2. *Optimal control approach*

The trajectory generation problem can also be expressed as one or multiple BVPs, which can be solved using the various techniques reported in the literature on optimal control. The BVPs can be solved either directly or indirectly. In [23], a two-point BVP (TPBVP) is formulated to generate time-optimal trajectories for a triple integrator system that is both input and state bounded. Since the input to a triple integrator system is the jerk, this method is thus called the jerk-limited trajectory generation. The BVP is solved analytically and is used to control the multicopter in real time with only an ARM-based flight controller. The technique is later extended in [79, 80] to generate collision-free trajectories in obstacle-strewn environments. The jerk-limited trajectory generation solves the following problem:

$$\begin{aligned} \min \quad & t_{\text{end}} \\ s.t. \quad & p(0) = p_0, \quad p(t_{\text{end}}) = p_{\text{ref}} \\ & v(0) = v_0, \quad v(t_{\text{end}}) = 0 \\ & a(0) = a_0, \quad a(t_{\text{end}}) = 0 \\ & \dot{p}(t) = v(t) \\ & \dot{v}(t) = a(t) \\ & \dot{a}(t) = j(t) \\ & -v_{\max} \le v(t) \le v_{\max}, \quad \forall t \in [0, t_{\text{end}}] \\ & -a_{\max} \le a(t) \le a_{\max}, \quad \forall t \in [0, t_{\text{end}}] \\ & -j_{\max} \le j(t) \le j_{\max}, \quad \forall t \in [0, t_{\text{end}}]. \end{aligned} \tag{27}$$

Unlike the methods in Sec. 3.2.1, it only solves a TPBVP instead of a multi-point BVP (MPBVP) as in Eq. (26), which is more computationally demanding. A similar approach can be found in [81], in which a closed-form solution is constructed for a cost function that penalizes the integration of the square of the jerk and the total time. Multiple segments of BVP trajectory generation methods can be used to derive a more complex trajectory as presented in [82]. In [72], the authors proposed to solve the BVPs offline by constructing a neural network to estimate online computational burden. This also opens the door to a wider range of cost functions and the online efficiency is no longer limited by the existence of analytical solutions.

3.3. *Integration of path and trajectory planning*

As discussed earlier, the final goal of motion planning is to generate a collision-free and dynamically feasible trajectory.

In order to increase computational efficiency, the overall problem is often decomposed into the pure geometric path level and the dynamics level. In other words, one often first focuses on dealing with the geometric constraints while ignoring the system dynamics. The obtained solution is then improved in a second phase where the detailed system dynamics are considered. There are many ways to integrate the path and trajectory planning layers, and these are identified in this section.

3.3.1. *Relaxing nonconvex optimization into convex optimization*

The most common approach of integrating path-level and trajectory-level planning is to relax the overall nonconvex optimization into a convex optimization problem. The core idea is to use a path planner to find the geometric path and then try to generate a dynamically feasible trajectory along the path.

To do so, one can convert a high-level path into a set of key waypoints and require the trajectory to pass through the key waypoints at certain specific times (see, e.g. [24]). For instance, in one of the earliest works on motion planning for multicopters [24], the authors use a polynomial spline to represent the trajectory and encode additional convex position constraints to avoid obstacles. The whole process is then formulated into a QP problem with an optimization objective that aims to minimize the integrals of the norm of the snap of the trajectory. They demonstrated the effectiveness of the proposed approach with an impressive real flight test that showed a quadrotor passing through a narrow window. Though the work presented significant contributions, it is limited to static and known environments. It also does not address how geometric constraints such as key-waypoints can be found automatically. Furthermore, the trajectory was computed offline.

In [27], a sampling-based planner RRT* [57] was used to find a collision-free geometric path, which was further pruned into a minimal set of waypoints. Then, a sequence of polynomial segments was jointly optimized to generate a smooth trajectory using similar techniques as in [24]. However, the resulted trajectory may again collide with obstacles even though the geometric path was collision-free. As a result, an additional waypoint was added halfway between the two ends and the polynomial was re-optimized. The process was repeated until a collision-free trajectory was found. Such an ad hoc solution is effective in a low obstacle density environment, but without any guarantee due to Runge's phenomenon.

An alternative approach is to extend the nominal path plan into a safe corridor, which consists of a series of connected convex regions. To guarantee that the result is collision-free, [83] used a graph search-based planner on an Octree-based map [84] and computed a collision-free corridor, which consisted of a series of axis-aligned cuboids. These cuboids provided additional linear position constraints and were directly formulated into the optimization process. As a result, the trajectory is guaranteed to stay inside the safe corridor. Chen *et al.*[85] and [45] further extended the idea of safe corridors to general 3D closed convex polyhedrons, whereas Tang and Kumar [86] extended the techniques to multi-agent systems.

3.3.2. *Direct nonconvex optimization*

We can also treat high-level geometric planning as an initial guess or heuristic for the trajectory generation process and directly formulate it as a nonconvex optimization problem. Oleynikova *et al.* [87] used a geometric planner named informed RRT* [62] to find a straight-line path as an initial guess and added a collision cost into the optimization objective, which is similar to the CHOMP technique [88] for robotic manipulation planning. The collision cost is a function of a Euclidean signed distance field at a point in the 3D work space. It is noteworthy that in such a formulation, the geometric constraints are transformed into collision cost in the optimization objective. The optimization process is thus essentially performed over the entire state space with the geometric path as an initial guess. However, due to the additional term in the objective function, the problem now becomes nonconvex and highly nonlinear, which is likely to cause numerical instability. Finding a good topological path as the initial guess is crucial to obtaining a resulting trajectory that is optimal and safe. In [89], a modified PRM method is used to find multiple topologically different initial guesses and their optimized results are compared to select the best choice.

3.3.3. *Motion primitives*

A motion primitive is a segment of a continuous trajectory. The core idea is to use motion primitives to construct a low-dimensional parameter space and thus reduce the searching dimension for the optimization problem. By using motion primitives, it is also possible to separate the dynamic and geometric constraints induced by the environment.

Motion primitives can be directly incorporated into a sampling-based planning framework (i.e. kinodynamic sampling based planning). The basic idea is to use motion primitives as the steering function of the sampling-based planner. Traditionally, such a steering function requires a TPBVP to be solved. Despite advances in TPBVP solvers, it is still difficult to achieve real-time performance as the process involves numerous calls of the steering function. To address this issue, one possible approach is to use

analytically available solutions, which can be found very efficiently when the problem comprises of certain optimization targets or state constraints. For example, the work in [82] extended the geometric RRT* algorithm [57] into a kinodynamic counterpart and used its analytically derived solution, which is similar to that in [70], as the steering function of the RRT*. It ignores state and input constraints to generate motion primitives in a fast manner, and then uses an extra checking process to get rid of the infeasible primitives during the search process. However, this method suffers from a low convergence rate when the state limits are tight, which is fairly common in the indoor environment. Since the constraints are ignored while the primitive is being generated, a large number of the generated primitives will turn out to be infeasible, which is a major issue for the motion planner.

Another way is to perform all possible computational processes offline and use a look-up table to store all the precomputed motion primitives. However, the resulting solution quality largely depends on the number of the motion primitives. Furthermore, for a 9 DOF multicopter system, the corresponding look-up table would have entries in the order of 10^{18}, which is computationally challenging in both construction and storage. To address this issue, Lan *et al.* [73] proposed the use of a neural network to approximate the motion primitives, and showed that the evaluation of the network is much faster than calling a TPBVP solver. This greatly improved the convergence rate of the searching process.

In many real-time applications, like the navigation of a multicopter in an unknown environment, it is still intractable to rely purely on sampling-based kinodynamic planning due to the high computational load. Therefore, the divide-and-conquer approach, which separates the planning problem into multiple layers, is still popular, especially in practical implementations. For example, when tasked to generate a dynamically feasible and collision-free trajectory for a multicopter, Lai *et al.* [26, 80] first plan a geometric path made of line segments, which ignores the detailed dynamics of the vehicle. Then, a third-order jerk-limited motion primitive is adopted to generate a dynamically feasible trajectory that stays close to the path. More specifically, a waypoint is picked from the line segment path. Multiple primitives that terminate within a certain radius to the waypoint are then sampled. A cost function is designed to evaluate each primitive, and the best one is selected for execution.

Finally, we note that Lai *et al.* [72] have recently extended the work of [26, 80] to propose a general framework for local motion planning with boundary state constrained motion primitives. To handle environment uncertainties and dynamic obstacles, the authors integrate the boundary state constrained motion primitives into a model predicative control framework. Unlike previous results in [26, 80] which essentially select the best motion over a set of discrete samples, Lai *et al.* [72] perform the optimization in the continuous domain.

4. Task Planning Techniques

As depicted in Fig. 6, there are two main classes of techniques used for pure task planning: artificial intelligence (AI)-based planning; and control synthesis using formal methods, which is the main focus of this section. We will highlight the key features of integrated task and motion planning in Sec. 5.

4.1. *AI-based planning*

Research on AI-based planning mostly studies various planning techniques that are used in the discrete planning domain. The main focus is on algorithms for different types of planning problems such as classical planning (fully deterministic and fully observable), fully and partially observable nondeterministic planning (FOND, POND), and probabilistic planning (MDP, POMDP). These planning problems differ from each other in terms of (i) whether the initial states and actions are deterministic; (ii) type of sensing; (iii) the presentation of uncertainties (e.g. in MDP and POMDP, one can represent uncertainties using probabilities while in FOND and POND, one represents uncertainties using sets); and (iv) the planning objective. Recently, there has been growing interest in handling temporally-extended goals. The planning problem is commonly formulated into a graph search problem [90, 91], though there are also planners that formulate the problem into a constraint satisfaction problem (CSP) [92–94].

It is possible to solve one type of planning problem using solvers that are created for another type of problem. For example, one could use a FOND planner to solve probabilistic planning problems [95], and use classical planners to solve FOND problems [96]. In AI-based planning, it is common to specify the planning problem by the Planning Domain Definition Language (PDDL) and its variants. The popularity of the PDDL is credited to the International Planning Competition, in which the PDDL is used as the standard interface to benchmark the planners. An excellent introduction to the PDDL can be found in the recent text by Haslum *et al.* [97]. In some literature, AI-based planners with PDDL input interfaces are often termed as PDDL planners. It should be noted that AI-based planning techniques have also been extended to handle hybrid systems.

4.1.1. *Classical planning*

Classical planning is the simplest type of AI-based approach. The problem is modeled as a fully deterministic and fully observable transition system with a planning goal, which can be achieved by applying deterministic actions from a fully known initial state. It is no surprise that the classical planning problem can be mapped to a graph search problem and solved by algorithms such as Dijkstra's and A* as described in Sec. 2. The challenging part is in scaling up as it involves an exponential state explosion. To address this problem, classical planners often aim to find a better heuristic [98–100] to improve the search efficiency.

The heuristic commonly used in the fast forward (FF) planner [99] is automatically generated by relaxing the plan from the delete-free relaxation of the original problem [101]. Another popular way to generate heuristics is by abstraction such as merge and shrink [102, 103]. Heuristics can also be derived from landmarks (i.e. sub-goals that we have to fulfill), as used in the LAMA planner [104]. Recently, there are classical planners [105] developed on a new measure, i.e. the *width* of the planning problem. These planners give impressive and promising results on some benchmark problems, which can be solved by the width-based planner in polynomial time if their goals are a single atom [106].

Classical planners can also be used as a basis to solve many nonclassical planning problems. With some special compilation processes, the classical planner is able to handle problems with soft goals [107] and temporally-extended goals [108, 109]. In addition, there are also works that use classical planners to solve partially observable planning problems [110, 111].

The classical planning model and problem are defined as follows.

Definition 16 (Classical planning model and problem). The state space model of *classical planning* is a tuple

$$\hat{P} =< S, s_0, S_G, Act, \rho, cost >$$

which consists of

(i) A finite set of states S
(ii) An initial state $s_0 \subseteq S$
(iii) A set of goal states $S_G \subseteq S$
(iv) A finite set of actions Act with $Act(s)$ denoting the sets of applicable actions to $s \in S$
(v) A deterministic state-transition function $\rho : S \times Act$ with $\rho(s, a)$ denoting the successor state when applying action $a \in Act(s)$ to $s \in S$
(vi) A cost function $cost : S \times Act \mapsto [0, \infty)$

A *plan* or a *solution* to a classical planning problem is a sequence of actions $\pi =< a_0, a_1, \ldots, a_n >$ that generates a state sequence $s_0, s_1, \ldots, s_{n+1}$ such that $a_i \in Act(s_i)$, $s_{i+1} = \rho(a_i, s_i)$ and $s_{n+1} \in S_G$. Note that if the cost function is not given explicitly, $cost(s, a)$ is usually set to be 1 whenever $\rho(s, a)$ is defined.

```
(define (domain warehouse)
  (:requirements :strips)
  (:predicates
   (package ?pkg)
   (vehicle ?veh)
   (location ?loc)
   (at ?x ?loc)
   (in ?pkg ?veh)
   )
  (:action load
    :parameters (?pkg ?veh ?loc)
    :precondition (and (package ?pkg) (
    vehicle ?veh) (location ?loc)
           (at ?veh ?loc) (at ?pkg ?loc))
    :effect (and (in ?pkg ?veh) (not (at ?pkg
     ?loc))))

  (:action unload
    :parameters (?pkg ?veh ?loc)
    :precondition (and (package ?pkg) (
    vehicle ?veh) (location ?loc)
           (at ?veh ?loc) (in ?pkg ?veh))
    :effect (and (at ?pkg ?loc) (not (in ?pkg
     ?veh))))

  (:action fly
    :parameters (?veh ?loc1 ?loc2)
    :precondition (and (vehicle ?veh) (
    location ?loc1) (location ?loc2)
           (at ?veh ?loc1))
    :effect (and (at ?veh ?loc2) (not (at ?
    veh ?loc1))))
  )
```

Fig. 12. The *domain* PDDL file for the warehouse example in Fig. 5.

Logic is a convenient way to compactly specify a subset of the state space where all states in a subset share some common properties. The discrete planning problem can be effectively represented by a logic-based representation. Such logic representations can also be interpreted as a sort of interface for the underlying discrete-planning problem. For classical planning, the state based planning problem and model in Definition 16 can be converted to STRIPS like logic-based representation in Definition 17. STRIPS-like representation is one of the most common logic-based representation of the planning problem, which mainly focuses on propositional logic, though there are also extensions to first-order logic. The name STRIPS refers to one of the first planning algorithm, STanford Research Institute Problem Solver, developed in the 1970s. In fact, STRIPS representation remains relevant till today and inspired more

```
(define (problem warehouse-problem)
  (:domain warehouse)
  (:objects uav warehouse siteA siteB siteC
    pkg1 pkg2)
  (:init
       (package pkg1)
       (package pkg2)
       (vehicle uav)
       (location warehouse)
       (location siteA)
       (location siteB)
       (location siteC)
       (at pkg1 warehouse)
       (at pkg2 warehouse)
       (at uav  siteC)
)
  (:goal (and (at pkg1 siteA) (at pkg2 siteB)
     (at uav warehouse)))
  )
```

Fig. 13. The *problem* PDDL file for the warehouse example in Fig. 5.

comprehensive PDDL representations that were developed around 1998 for the very first international planning competition. The STRIPS remains the simplest yet most important subset of the PDDL.

There are several key components of a STRIPS-like representation. The first is what is known as *instances*, which refer to real-life objects such as robots, rooms, cups and tables. The second component is a predicate or proposition whose value can either be true or false. For example, a predicate *on (cup, table)* can describe a situation that there is a cup placed on the table if it is true. Similarly, *inRoom (robot, kitchen)* can indicate whether the robot is inside the kitchen. Note that each predicate actually maps to a subset of underlying continuous state space.

A *literal* is an atomic formula (positive literal) or the negation of an atomic formula (negative literal). A state of the planning problem can then be described by a set of literals. An operator or action manipulates the states by adding literals or deleting literals over the current state. More specifically, an operator has a form of $op = (Pre, Eff)$, where *Pre* is the set of literals that have to be true when applying this action, and is often called the *precondition*, and *Eff* is the set of effects after applying the action. In STRIPS, *Eff* can be described in the form $Eff := (Add, Del)$, where *Add* is the set of literals to be added and *Del* is the set of literals to be deleted. There are many variants of the STRIPS, and additional features include but are not limited to timed actions, sensor models, and conditional effects.

Definition 17 (Classical planning in logic representation). The logic representation of the classical planning model in Definition 16, i.e. $\hat{P} =< S, s_0, S_G, Act, f, cost >$, is given by

$$P =< F, I, A, G >,$$

where

(i) *Ins* is a set of instances.
(ii) F is a finite set of predicates. A predicate is a partial function of one of more instances, which can be either true or false. A literal is any predicate (or its negation) applied to a specific set of instances.
(iii) A is a finite set of operators or actions. Each operator has 1) preconditions which are set of literals must be hold when applying this operator; 2) effects which are a set of literals of the result of this operator.
(iv) I is a set of initial conditions, which is a set of literals over F
(v) G is the planning goal, which is a set of literals over F.

Here, we revisit the example shown in Fig. 5 and show the entire planning problem written in standard STRIPS-like PDDL format (see Figs. 12 and 13). We assume that the multicopter is originally at *siteC* and needs to fly to the warehouse to collect two packages *pkg1* and *pkg2*. These two packages need to be delivered to *siteA* and *siteB*, respectively. For simplicity, we assume that the multicopter is capable of executing three abstract actions, namely *load*, *unload* and *fly*.

There are two PDDL files. One is the *domain* file which defines the proposition set F and action set A in Definition 17 (see Fig. 12). The other is the *problem* file which defines the initial state I and end goal G of the planning problem in Definition 17 (see Fig. 13).

The STRIPS representation can also be translated into linear temporal logic (LTL). As discussed above, STRIPS actions typically consists of three parts: (i) the action name, (ii) a set of preconditions that must be true before this action is applied, and (iii) a set of effects after the action. The effects of an action a consist of two parts: an add list, $Add(a)$, which describes a set of states that have to be true after action a; and a delete list, $Del(a)$, which consists of a set of facts that have to be deleted (i.e. be false) in the effects.

Let f denote a literal. For an action $a \in A$, the effects are

$$\bigwedge_{f \in Add(a)} f \wedge \bigwedge_{f \in Del(a)} \neg f.$$

Let ϕ denote the precondition set. We then have:

(i) The preconditions have to be always satisfied before the action can be applied, i.e. $\square(\bigcirc a \rightarrow \phi)$ (if the action is to occur in the next step, the preconditions have to be satisfied now);
(ii) The effects have to be applied

$$\square\left(\bigcirc a \rightarrow \bigcirc\left(\bigwedge_{f \in Add(a)} f \wedge \bigwedge_{f \in Del(a)} \neg f\right)\right)$$

(iii) The rest of literals have to remain unchanged

$$\Box a \rightarrow \bigwedge_{f \notin \{Add(a) \bigcup Del(a)\}} (f \Longleftrightarrow \bigcirc f)$$

(iv) At each step, only one action should be activated, i.e.

$$\Box \left(\left(\bigvee_{a \in A} a \right) \wedge \left(\bigwedge_{a_i, a_j \in A, i \neq j} a_i \rightarrow \neg a_j \right) \right)$$

(v) The initial condition

$$\bigwedge_{f \in I} \wedge \bigwedge_{f \notin I} \neg f$$

(vi) The goal has to be eventually reached, i.e. $\Diamond \phi_{goal}$.

The most common way to solve the classical planning problem is to re-formulate the problem into a graph search problem. During the planning process, the state transition graph is often only generated incrementally by applying a sequence of actions and it is unnecessary to specify the full transition graph beforehand.

4.2. *Control synthesis*

Given a system abstraction and a set of formal high-level task specifications, control synthesis refers to using *formal methods* to automatically find a correct-by-construction controller, i.e. finding a control policy such that at the given abstraction level, the controller will result in a controlled system that satisfies all task specifications with respect to a modeled environment behavior. Usually, these tasks are formalized using various types of temporal logics. Compared to AI-based planning where most goals are not temporally extended, researchers in the control synthesis community focus more on handling the temporal specifications in a formal way. In some works (see e.g. [112]), both the task specification and the abstract system model are formalized using temporal logics. In [113, 114], the authors provide an excellent review on applying formal methods to robotics systems.

4.2.1. *Automata-based control synthesis*

One popular approach of synthesizing a controller under temporal specification is based on the automaton theory [115–118]. This method often applies to a deterministic system abstraction model. The basic procedure can be summarized as follows: first, the temporal logic is translated into an equivalent automaton, in which the language satisfying the temporal formula is also accepted by the automaton. A product automaton is then created between the system abstraction (which is represented as a labeled transition system) and the specification automaton. Lastly, it searches for a satisfying run over the product automaton.

It has been proven that any LTL formula can be mapped to a Büchi automaton, and there are various open source tools such as LTL2BA [119] and Spot [120] which are available for performing such translations. Given a labeled transition system TS and a Büchi automaton $\mathcal{BA}$, one can create a product automaton $\mathcal{P} = \text{TS} \otimes \mathcal{BA}$, which itself is also a nondeterministic Büchi automaton. Since the Büchi acceptance condition requires the run to visit one of the accepting states infinitely often, one can then search for a *prefix-suffix* structure over the product automaton. The *prefix* starts from an initial state to an accepting state. The *suffix* part loops around this accepting state (i.e. starting from the accepting state, and ending in the exact same accepting state). The *prefix-suffix* structure can either be found by an algorithm that is similar to a nested depth-first search (DFS), or an algorithm that is based on finding the strongly connected component (SCC) [121]. On the other hand, co-safe LTL can be translated into a deterministic finite automaton. One can then directly search a run that starts from the initial state and ends in an accepting state over the product finite automaton.

The synthesis of the automata-based approach is frequently combined with iterative planning techniques to handle partially unknown environments [117, 118]. There are also research efforts focusing on partial specification satisfaction based on the automata-based synthesis approach [118, 122, 123].

4.2.2. *Game-based synthesis*

Game-based synthesis is usually used for reactive tasks where the proposition set contains both the robot and environment propositions (observable but uncontrollable). The key idea is to model the synthesis problem as a two-player game, in which one player is the robot and the other is the environment. The objective is to find a control strategy for the robot such that it can react accordingly to the behavior of the environment model and the resulting trajectory satisfies the given specifications. Reactive synthesis for the full set of LTL can be computationally challenging [124]. There is thus a significant amount of research focusing on the GR(1) subset instead [112, 125–127]. There are also various efficient solvers for the GR(1) subset, such as the Slugs (a complete GR (1) synthesizer) [128].

4.2.3. *Markov decision processes-based synthesis*

Markov decision process (MDP)-based synthesis is mainly used for two scenarios: to handle the temporal specifications expressed in probabilistic logics such as probabilistic

computation tree logic (PCTL) [129]; and to handle the discrete temporal logics such as LTL for the dynamical system modeled in MDP [130, 131]. The goal of the MDP-based synthesis is to find a policy that maximizes the probabilities of fulfilling a temporal specification.

4.2.4. *Control synthesis from an optimization perspective*

The temporal specification can also be translated directly into a set of constraints over the dynamic system. Often, the overall formulation is solved in a mixed integer linear programming (MILP) or a mixed integer quadratic programming (MIQP) framework, assuming that the predicates are linear. Pioneer work of such approaches can be found in [132], where the authors encode the LTL formulae into a set of mixed-integer constraints. There are also various results focusing on the encoding of signal temporal logics (STL) [133]. Instead of Boolean satisfaction, STL can provide quantitative satisfaction. For each STL formula, one can assign a robustness score over it to evaluate the robustness of the satisfaction. This robustness score can be seamlessly integrated into an optimization problem.

In [133], Raman *et al.* extend the synthesis problem into a receding horizon control framework. The main drawback of the MILP approach is its high computational burden due to the large set of binary variables. Work in [134] adopts an iterative planning approach, which deals with fewer binary variables in each iteration. The idea is to guess a trajectory first and identify any violation points. They then add new constraints over the violation points and solve the problem repeatedly. The idea is similar to that in incremental constraints solving [135]. In [136], the authors provide a differentiable approximation of the robustness function, and thus formulate it as a sequential quadratic programming (SQP) problem for higher computational speed. In [137], Mehdipour *et al.* went one step further by defining a new robustness score (i.e. the arithmetic-geometric mean robustness), with gradient information that is readily available.

4.3. *Bridging the gaps*

Recently, there has been an increasing amount of research that aims to bridge the gap between AI-based planning and control synthesis approaches. Researchers who used to work on AI-based planning have begun to investigate the reformulation of temporal logic synthesis problems into various types of common planning problems and then solve them using existing PDDL planners.

For instance, the authors in [109] proposed handling the nonreactive LTL synthesis problem by compiling it into a standard PDDL-based classical planning problem. The fundamental idea behind this is still converting the LTL specification into an automaton and searching for a prefix-suffix structure in the product automaton. More specifically, the transition system is first modeled into the standard PDDL format and the specification automaton generated from the LTL is converted into a PDDL action with conditional effects. The transition of the product automaton is thus modeled as two successive actions. The first move is taken by the specification automaton and the second move is taken by the transition system. To handle the infinite plan, a special loop action is created. With additional auxiliary propositions and actions, the entire search problem can be represented as a classical planning problem. The resulting problem can then be solved by any standard PDDL classical planner such as those in [99, 104].

In [138], the authors propose to handle the full reactive LTL synthesis problem by compiling the two-player game into a standard fully observable planning problem (FOND). Any PDDL FOND planner can thus be adopted to solve the problem. In [139], the task temporal constraints and the continuous optimization variables are combined in a hierarchical structure and solved with a dynamic backtracking method to achieve activity planning. The system has been proven highly efficient and reliable with its implementation on the Mars Rover.

4.4. *Direct task plan specified by human operators*

In the above sections, we discussed a wide range of techniques used for automated task planning. However, in practice, it is also important to allow human operators to directly specify the task plan instead of going through an automated planner. We identify two reasons for this need. First of all, automated planning requires a sophisticated planning model and a properly defined planning goal. For nonexperts who do not have sufficient understanding in planning, it is difficult to create a proper planning model and formally create the task specifications. For instance, in Sec. 2.4, we presented various temporal logics used to represent task requirements. We also provided a specification pattern for LTL (see Sec. 2.4.3) to assist users in effectively translating the description requirements into formalism. For nonexperts, however, it remains challenging to understand and use these formalisms. More importantly, in some cases, human operators might simply not wish to use an automated planner, as they might not be able to produce specific types of behaviors. For example, a path planner might always return the shortest path between two points on a map. The human operator on the other hand might actually prefer to follow a self-designed nonoptimal path. If we are forced to use an automated planner, we will have to modify the optimization target by adding many additional constraints.

Currently, the most popular choice for plan representation is a finite state machine (FSM) [140–142] which is a well-studied mathematical tool. In [140], the authors create a powerful graphical user interface (GUI) named RAFCON which allows the user to specify tasks for mobile robots using a hierarchical state machine. In fact, many task planners generate the task policy directly in the form of a state transition system [112]. The main drawback of the FSM is that the readability for human operators will decrease with the size of state machine. Editing a particular state or transition manually in the FSM is never straightforward since the transition itself is state dependent in nature. As pointed out by [143], the state transition in FSM is essentially an on-way transfer of control, which behaves in a similar way as the outdated programming statement GoTo [144]. In fact, the low readability is the reason the GoTo statement has received a wide range criticism and its usage has been significantly restricted in modern programming languages [145]. As pointed out in Edsger Dijkstra's well-known letter "GoTo Statement Considered Harmful" dated back in 1968 [144], "the GoTo statement as it stands is just too primitive; it is too much an invitation to make a mess of one's program".

In this survey, we introduce another plan representation, namely the behavior tree, which partially addresses the challenges faced by the FSM. The behavior tree is a tree-based model that organizes a group of modular behaviors which was originally developed in the game industry to model the behaviors of nonplayer characters [146]. It allows the user to easily specify and execute the task plan in a tree form. Figure 14 depicts a typical behavior tree, in which the leaf node represents a primitive behavior that maps to a control command directly executable by an agent. The internal node groups the primitive behaviors into a composite one in accordance to a set of rules that are defined. Similar to the hierarchical task network (HTN), the behavior tree can be regarded as a framework that represents a set of refinement models. The key difference, however, is that the behavior tree comes with additional execution policies. Each node defines a set of rules on how its children nodes should be traversed. In general, different node types have different traversal rules. In other words, the behavior tree not only represents a hierarchy of models, but also defines a control policy associated with the decomposition process.

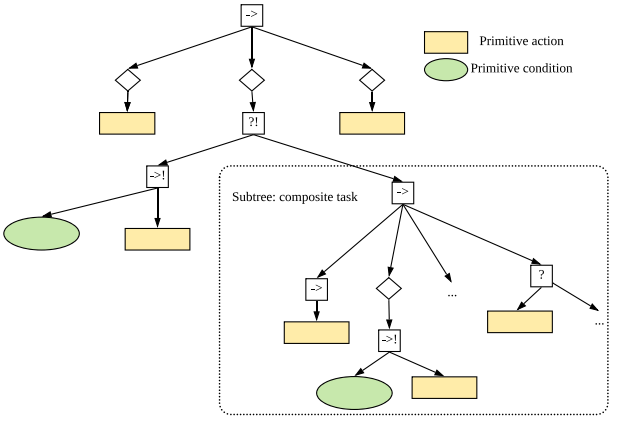


Fig. 14. An example of a behavior tree.

As a type of refinement acting engine (RAE) [147], the behavior tree gradually refines compound tasks (behaviors) into a set of executable primitive tasks (behaviors) by traversing the tree. With such an *acting* feature, it is then possible to interleave the planning and execution via the tree [143] [148].

It has been shown in [149] that the behavior tree is able to generalize other structures such as subsumption architecture and decision trees. The key advantage of the behavior tree is that each tree node essentially behaves like a two-way function. As a result, it is able to decouple the node implementation and node usage, which makes the behavior tree highly modular. The code is thus also re-usable due to the modular tree node implementation.

Recently, research on the behavior tree has attracted more attention because of its hierarchical and modular structure as well as natural execution policy. The concept has been applied to a wide range of real robotic applications, such as object manipulation [150], human–robots collaboration [151], underwater vehicles [152], humanoid control [143] and even semi-autonomous robotic surgeries [153]. For instance, in [150], the authors use the behavior tree as the main architecture to integrate several core functionalities (e.g. perception, planning, control and audio processing) of a manipulation task. In [153], the behavior tree is used to systematically model and implement the surgical procedure for a brain tumor ablation.

In [151, 154], the authors create a behavior-tree based system, named CoSTAR, for end-users to specify task plans to industrial collaborative robots. The main motivation behind the CoSTAR is the observation that many industrial robots are often underutilized in small manufacturers due to the high cost and user-unfriendly interface of reprogramming the robots for other tasks that are not prespecified. The CoSTAR addresses this issue by using a behavior-tree based framework, which combines the basic capabilities of the robots in a modular manner. This helps nonexperts to create new tasks on the fly. The perception module has been integrated into the system, which allows users to create a task plan that is more robust to the environmental changes. SARAFun [155] is another project that aims to help nonexperts to specify an assembly task from scratch, where the behavior tree is used as a control architecture that combines and executes a set of modular learned or planned actions.

Due to its high modularity and hierarchical structure, behavior trees are also used to specify and execute complex tasks for autonomous multicopters. In [156], a behavior tree is used as the core underlying structure to construct a modular mission management system for small-scale multicopters. This system was successfully demonstrated at the 2017 International Micro Aerial Vehicle Competition (IMAV 17). Specifically, in the competition, the multicopter completed a series of tasks including moving through a narrow window, navigating through a set of obstacles, flying in front of an industrial fan, searching for a target, dropping a payload and finally landing on a moving platform.

5. Integrated Task and Motion Planning

The integration of task and motion planning is not a trivial issue due to possible mismatch among the different abstraction models at different planning levels. In a common hierarchical layered planning framework where the task-level model often completely ignores the dynamics and geometric details, the high-level task plan may simply be infeasible for the lower-level motion planner to implement. The following are some general approaches to properly integrating the task planning layer and the motion planning layer.

5.1. *Direct optimization with task and motion constraints*

The most straightforward approach to integrating task planning and motion planning is by directly considering all constraints (task-related constraints, geometric constraints, and dynamics constraints) in one planning layer. For example, we can use the optimization synthesis technique presented in Sec. 4.2.4 to directly generate a dynamically feasible trajectory that fulfills the temporal task constraints with a complete dynamics model. The authors in [157] use signal temporal logic for the formalism of the task requirement. A polynomial was used to parameterize the motions of the vehicle and a smooth operator was further adopted to approximate the original MIQP problem into an SQP problem. Although this approach is sound and complete, the main drawback is the high computational load. For robotics, the fundamental reason for decoupling task and motion into two layers is to gain computational efficiency. Although direct optimization can generate feasible or even an optimal solution in one run, such an algorithm often requires significantly larger computational power, which may not be feasible for real-time applications.

5.2. *Enforcement of the simulation relationship*

Another possible approach is to ensure that the simulation relationship holds from the abstract model to the original system. In other words, we need to design a special system abstraction strategy and special motion planning or control algorithms such that the motion and control algorithms can implement the abstract task plan [158, 159]. The key challenge is to find a correct abstraction, which in general is not a straightforward task.

5.3. *Sampling-based integration of task and motion planning*

There is a large amount of research interest in applying sampling-based techniques to solve planning under temporal logic constraints as these techniques are convenient tools for abstracting the original dynamic system. Pioneering work on applying sampling-based techniques to solve a planning task under temporal logic constraints can be found in [160]. The key idea is to abstract the robot dynamic system into a discrete counterpart by using a sampling-based method. More specifically, the rapidly-exploring random graph (RRG) is used to create a sampling-based Kripke structure, which is essentially an approximation of the underlying dynamics. A model checking tool is then used to check whether this model satisfies the specification expressed in μ-calculus and contains a satisfying trajectory. A product automaton is created and a cyclic pattern is then found in the graph by a graph algorithm. If the abstract model is not rich enough to contain a satisfying run, more samples are added and the above procedure is repeated. The main issue with this is that the model checking process is independent from the graph construction. In other words, for each checking iteration, a new independent Kripke structure has to be provided, and a complex graph search algorithm has to be executed from scratch. In [161], Vasile and Belta improve the work in [160] by incrementally building up the RRG and maintaining the product graph, which only requires incremental searching for a satisfying run.

This was further extended in [162] to handle reactive planning. Specifically, a run is first found to satisfy the global specification, and then a local path is patched to the global run to handle the reactive local specification. It should be noted that these algorithms can only handle path planning. In other words, they can only handle system dynamics without differential constraints. In addition, these algorithms are only capable of producing a feasible path and not an optimal one.

Both works in [160, 161] require an exact local steering function to obtain a graph-based Kripke structure, which

requires a two-point boundary value problem to be solved. However, for some types of systems, such steering functions might not be readily available. To address this issue, [163] extends the work in [160] by adopting a tree-based planner, which relies on forward simulation instead of steering, specifically, the SST* from [164]. Multiple tree-based Kripke structures are propagated by the SST* in parallel from different initial states in the state space. Each initial state is sampled from a region that corresponds to a positive atomic proposition appearing in temporal specifications. Later, all Kripke structures are merged into a single abstract Kripke structure that contains the solution.

There are also works that use an RRT*-like algorithm to search for an optimal satisfying run (see e.g. [165]). However, the work in [165] focuses on applying sampling techniques to further abstract a large transition system into a simpler one, instead of abstracting a continuous state space into a discrete counterpart. The application of [165] is mainly for large-scale multi-agent task planning, in which it is assumed that the task dynamics of each robot are already provided as a transition system. The sampling domain is then the product of each robot transition system, which is already in discrete form. As a result, the algorithm in [165] cannot be directly applied to a continuous dynamical system.

5.4. *Iterative planning with model updating*

An increasingly popular approach is to start with a guessed abstraction and then apply an iterative planning strategy. More specifically, we can start with a guessed task-level abstraction and then plan with the guessed model. If the lower-level motion planner cannot implement the resulting task plan, we need to find the underlying reasons and update the abstraction accordingly. We then repeat the planning with the updated task model. Usually, the whole process is facilitated by an independent task-action layer. For general tasks which require a wide range of functions, the term *motion* is extended to a more general one named *action*.

Figure 15 presents a general framework for integrating high-level planning with lower-level planning using a task-action interface. The task specification interface, as shown graphically in pink, allows the user to properly specify the task requirements. The yellow sections represent the model-related modules. This consists of two major submodules. The first one is the symbolic abstract model used for the planning process, and it is possible for the planning model to have different levels of abstraction. Besides the symbolic model itself, one has to assign meaning to each symbol, i.e. perform *symbol grounding*. Specifically, each symbol used in the abstract model needs to be assigned a physical meaning and mapped to a subset of the underlying continuous full state space. It is not a trivial task to ground the symbols in general and complex algorithms are often required [166, 167]. In practice, it is also common to directly channel human knowledge to ground the symbol. For instance, in the example shown in Fig. 5, to ground the symbol *site A*, the human operator can specify the GPS coordinates of the center of the site, assuming the site area can be represented as a circle. The blue section in Fig. 15 represents the main high-level task planner, which computes the task-level plan given a task specification and

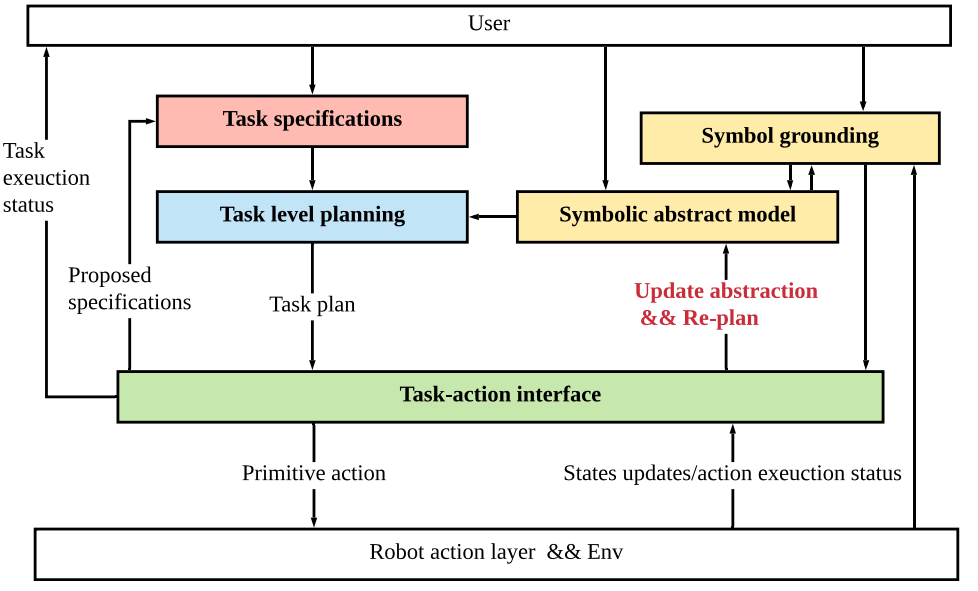


Fig. 15. A general framework of integrating high-level planning with lower-level planning using a task-action interface.

planning model. The green section is the task acting engine, which is the main interface between high-level planning and low-level implementation.

In [168], the authors use an independent interface to communicate the geometric details from the motion level to the task-level in the form of a logical predicate. In [118, 169, 170], the state space is partitioned into several regions to form an abstraction. The discrete plan helps the lower-level sampling-based motion tree to identify the best possible region to explore. The motion tree also sends back utility information to the discrete planner, i.e. updates the discrete abstraction by adjusting region weights. The work in [171] unifies sampling-based planning for integrated task and motion planning by abstracting the motion planner into PDDL actions. By performing conditional sampling in the task-motion space, a sampling-based abstraction is constructed. Discrete planning is then performed on this abstraction. If it fails to find a solution, more samples will be drawn to provide a denser abstraction.

In [172], the authors used the behavior tree discussed in Sec. 4.4 as the main task-action interface. The behavior tree is responsible for refining the high-level task plan into primitive actions, executing the actions, providing necessary execution feedbacks, updating the task-level model and facilitating the re-planning process. Although the system used a PDDL-based classical planner as the core high-level planner, it is also able to handle LTL task specifications in addition to standard PDDL specifications. The basic idea involves applying the techniques mentioned in Sec. 4.3, which compiles the nonreactive LTL synthesis problem into a classical planning problem. By doing so, the system is able to use any off-the-shelf classical planner and easily benefit from the recent advances of the planning community. The linear task plan is later translated into a behavior tree form and is further refined into a set of more primitive actions. With the help of a behavior tree-based task-action interface, it is able to perform iterative planning and thus handle the reactive LTL task specifications by assuming a determined task planning model for each planning cycle. Moreover, by using a behavior tree as the task-action interface, the system is able to directly handle a human specified behavior tree plan without going through an automated planner.

6. Conclusion

In this paper, we presented a survey of the various background materials and techniques used in task and motion planning for unmanned multicopters. This includes the dynamic model of the multicopter, common graph search algorithms, symbolic representation of task-level system abstraction models and temporal task specifications, and commonly used motion planning and task planning techniques. We hope that this survey provides a comprehensive overview of the area and can serve as a reference for beginners looking to enter the field of the motion and task planning.

Acknowledgments

The work of Shupeng Lai is supported in part by Peng Cheng Laboratory. The work of Ben M. Chen is supported in part by the Research Grants Council of Hong Kong SAR (Grant No: 14209020), in part by the Key Program of National Natural Science Foundation of China (Grant No: U1613225), and in part by Peng Cheng Laboratory.

The authors would like to thank Miss Yu-Herng Tan for proofreading the whole manuscript and for her constructive comments that helps improving the overall presentation of this manuscript.

References

[1] D. H. Lyon, A military perspective on small unmanned aerial vehicles, *IEEE Instrum. Measur. Mag.* **7**(3) (2004) 27–31.
[2] M. D. Zollars, R. G. Cobb and D. J. Grymin, Optimal suas path planning in three-dimensional constrained environments, *Unmanned Syst.* **07**(02) (2019) 105–118.
[3] G. Skorobogatov, C. Barrado and E. Salamí, Multiple UAV systems: A survey, *Unmanned Syst.* **08**(02) (2020) 149–169.
[4] The dji agras mg-1 octocopter https://www.dji.com/sg/mg-1, accessed on 2020-02-10.
[5] M. El-Jiz and L. Rodrigues, Trajectory planning and control of a quadrotor choreography for real-time artist-in-the-loop performances, *Unmanned Syst.* **06**(01) (2018) 1–13.
[6] A. Mohiuddin, T. Tarek, Y. Zweiri and D. Gan, A survey of single and multi-UAV aerial manipulation, *Unmanned Syst.* **08**(02) (2020) 119–147.
[7] E. Johnson and D. Schrage, The georgia tech unmanned aerial research vehicle: GTMax, in *AIAA Guidance, Navigation, and Control Conference and Exhibit*, p. 5741.
[8] G. Cai, B. M. Chen, T. H. Lee and M. Dong, Design and implementation of a hardware-in-the-loop simulation system for small-scale uav helicopters, in *IEEE Int. Conf. on Automation and Logistics* (IEEE, 2008), pp. 29–34.
[9] J. M. Levin, A. A. Paranjape and M. Nahon, Agile maneuvering with a small fixed-wing unmanned aerial vehicle, *Robot. Autonom. Syst.* **116** (2019) 148–161.
[10] T. Özaslan, G. Loianno, J. Keller, C. J. Taylor, V. Kumar, J. M. Wozencraft and T. Hood, Autonomous navigation and mapping for inspection of penstocks and tunnels with mavs, *IEEE Robot. Autom. Lett.* **2**(3) (2017) 1740–1747.
[11] L. Heng, D. Honegger, G. H. Lee, L. Meier, P. Tanskanen, F. Fraundorfer and M. Pollefeys, Autonomous visual mapping and exploration with a micro aerial vehicle, *J. Field Robot.* **31**(4) (2014) 654–675.
[12] Y. Bi, M. Lan, J. Li, S. Lai and B. M. Chen, A lightweight autonomous mav for indoor search and rescue, *Asian J. Control* **21**(4) (2019) 1732–1744.

[13] J. Q. Cui, S. K. Phang, K. Z. Ang, F. Wang, X. Dong, Y. Ke, S. Lai, K. Li, X. Li, J. Lin *et al.*, Search and rescue using multiple drones in post-disaster situation, *Unmanned Syst.* **4**(01) (2016) 83–96.

[14] Pixhawk http://pixhawk.org/, accessed on 2019-09.

[15] B. M. Chen, T. H. Lee, K. Peng and V. Venkataramanan, *Hard Disk Drive Servo Systems* (Springer Science & Business Media, 2006).

[16] A. Majumdar and R. Tedrake, Funnel libraries for real-time robust feedback motion planning, *Int. J. Robot. Res.* **36**(8) (2017) 947–982.

[17] A. T. Hafez and M. A. Kamel, Cooperative task assignment and trajectory planning of unmanned systems via HFLC and PSO, *Unmanned Syst.* **07**(02) (2019) 65–81.

[18] J. Gibson, T. Schuler, L. McGuire, D. M. Lofaro and D. Sofge, Swarm and multi-agent time-based a* path planning for lighter-than-air systems, *Unmanned Syst.* **08**(03) (2020) 253–260.

[19] M. Radmanesh, M. Kumar, P. H. Guentert and M. Sarim, Overview of path-planning and obstacle avoidance algorithms for uavs: A comparative study, *Unmanned Syst.* **06**(02) (2018) 95–118.

[20] S. H. Park, B. Kim, C. M. Kang, C. C. Chung and J. W. Choi, Sequence-to-sequence prediction of vehicle trajectory via LSTM encoder-decoder architecture, in *IEEE Intelligent Vehicles Symp. (IV)* (IEEE, 2018), pp. 1672–1678.

[21] A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss and W. Burgard, OctoMap: An efficient probabilistic 3D mapping framework based on octrees, *Autonom. Robots* **34**(3) (2013) 189–206.

[22] H. Oleynikova, Z. Taylor, M. Fehr, R. Siegwart and J. Nieto, Voxblox: Incremental 3D Euclidean signed distance fields for on-board mav planning, in *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)* (Vancouver, BC, 2017), pp. 1366–1373.

[23] M. W. Mueller, M. Hehn and R. D'Andrea, A computationally efficient motion primitive for quadrocopter trajectory generation, *IEEE Trans. Robot.* **31**(6) (2015) 1294–1310.

[24] D. Mellinger and V. Kumar, Minimum snap trajectory generation and control for quadrotors, *IEEE Int. Conf. Robotics and Automation (ICRA)* (IEEE 2011), pp. 2520–2525.

[25] F. Augugliaro, A. P. Schoellig and R. D'Andrea, Generation of collision-free trajectories for a quadrocopter fleet: A sequential convex programming approach, *IEEE/RSJ Int. Conf. Intelligent Robots and Systems (IROS)* (IEEE, 2012), pp. 1917–1922.

[26] S. Lai, K. Wang, H. Qin, J. Q. Cui and B. M. Chen, A robust online path planning approach in cluttered environments for micro rotorcraft drones, *Control Theory Technol.* **14**(1) (2016) 83–96.

[27] C. Richter, A. Bry and N. Roy, Polynomial trajectory planning for aggressive quadrotor flight in dense indoor environments, *Robot. Res.* (2016) 649–666.

[28] S. K. Phang, S. Lai, F. Wang, M. Lan and B. M. Chen, Systems design and implementation with jerk-optimized trajectory generation for UAV calligraphy, *Mechatronics* **30** (2015) 65–75.

[29] N. Piterman, A. Pnueli and Y. Saar, Synthesis of reactive (1) designs, in *Int. Workshop on Verification, Model Checking, and Abstract Interpretation* (Springer, 2006), pp. 364–380.

[30] R. Bloem, B. Jobstmann, N. Piterman, A. Pnueli and Y. Saar, Synthesis of reactive (1) designs, *J. Comput. Syst. Sci.* **78**(3) (2012) 911–938.

[31] C. Vasile, J. Tumova, S. Karaman, C. Belta and D. Rus, Minimum-violation scLTL motion planning for mobility-on-demand, in *IEEE Int. Conf. Robotics and Automation (ICRA)* (Singapore, 2017), pp. 1481–1488.

[32] P. Bouyer, F. Laroussinie, N. Markey, J. Ouaknine and J. Worrell, Timed temporal logics, *Models, Algorithms, Logics and Tools* (Springer, 2017), pp. 211–230.

[33] Y. Zhou, D. Maity and J. S. Baras, Timed automata approach for motion planning using metric interval temporal logic, in *European Control Conf. (ECC)* (Aalborg, 2016), pp. 690–695.

[34] R. Alur, T. Feder and T. A. Henzinger, The benefits of relaxing punctuality, *J. ACM* **43**(1) (1996) 116–146.

[35] O. Maler and D. Nickovic, Monitoring temporal properties of continuous signals, in *Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems* (Springer, 2004), pp. 152–166.

[36] M. B. Dwyer, G. S. Avrunin and J. C. Corbett, Patterns in property specifications for finite-state verification, in *Proc. Int. Conf. Software Engineering* (IEEE, 1999), pp. 411–420.

[37] M. B. Dwyer, G. S. Avrunin and J. C. Corbett, Property specification patterns for finite-state verification, in *Proc. 2nd workshop on Formal Methods in Software Practice* (ACM, 1998), pp. 7–15.

[38] E. M. ClarkeJr, O. Grumberg, D. Kroening, D. Peled and H. Veith, *Model Checking* (MIT Press, 2018).

[39] C. Baier and J.-P. Katoen, *Principles of Model Checking* (The MIT Press, 2008).

[40] G. Conte, S. Duranti and T. Merz, Dynamic 3D path following for an autonomous helicopter, *IFAC Proceedings Volumes* **37**(8) (2004) 472–477.

[41] D. R. Nelson, D. B. Barber, T. W. McLain and R. W. Beard, Vector field path following for miniature air vehicles, *IEEE Trans. Robot.* **23**(3) (2007) 519–529.

[42] G. Hoffmann, S. Waslander and C. Tomlin, Quadrotor helicopter trajectory tracking control, in *AIAA Guidance, Navigation and Control Conference and Exhibit*, Honolulu, 2012, p. 7410.

[43] B. Paden, M. Cap, S. Z. Yong, D. Yershov and E. Frazzoli, A survey of motion planning and control techniques for self-driving urban vehicles, in *IEEE Transactions on Intelligent Vehicles*, Vol. 1, No. 1 (2016), pp. 33–55.

[44] J. Chen and S. Shen, Improving octree-based occupancy maps using environment sparsity with application to aerial robot navigation, in *IEEE Int. Conf. Robotics and Automation (ICRA)* (Singapore, 2017), pp. 3656–3663.

[45] S. Liu, M. Watterson, K. Mohta, K. Sun, S. Bhattacharya, C. J. Taylor and V. Kumar, Planning dynamically feasible trajectories for quadrotors using safe flight corridors in 3-d complex environments, *IEEE Robot. Autom. Lett.* **2** (2017) 1688–1695.

[46] F. Gao *et al.*, Optimal trajectory generation for quadrotor teach-and-repeat, in *IEEE Robotics and Automation Letters*, Vol. 4, No. 2 (2019), pp. 1493–1500.

[47] F. Gao, Y. Lin and S. Shen, Gradient-based online safe trajectory generation for quadrotor flight in complex environments, in *IEEE/RSJ Int. Conf. Intelligent Robots and Systems (IROS)* (2017), pp. 3681–3688.

[48] E. W. Dijkstra *et al.*, A note on two problems in connexion with graphs, *Numer. Math.* **1**(1) (1959) 269–271.

[49] P. E. Hart, N. J. Nilsson and B. Raphael, A formal basis for the heuristic determination of minimum cost paths, *IEEE Trans. Syst. Sci. Cybernet.* **4**(2) (1968) 100–107.

[50] A. Stentz *et al.*, The focussed D* algorithm for real-time replanning, in *Proceedings of the International Joint Conference on Artificial Intelligence* (1995), pp. 1652–1659.

[51] S. Koenig and M. Likhachev, Fast replanning for navigation in unknown terrain, in *IEEE Transactions on Robotics*, Vol. 21, No. 3, June 2005, pp. 354-363.

[52] M. Likhachev, D. I. Ferguson, G. J. Gordon, A. Stentz and S. Thrun, Anytime dynamic A*: An anytime, replanning algorithm, in *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)* (2005), pp. 262–271.

[53] M. Likhachev, G. J. Gordon and S. Thrun, ARA*: Anytime a* with provable bounds on sub-optimality, *Adv. Neural Inform. Process. Syst.* (2004) 767–774.

[54] S. M. Lavalle, Rapidly-exploring random trees: A new tool for path planning, technical report (1998).

[55] D. Hsu, J.-C. Latombe and R. Motwani, Path planning in expansive configuration spaces, in *Proc. Int. Conf. Robotics and Automation*, Vol. 3 (Albuquerque, NM, USA, 1997), pp. 2719–2726.

[56] L. E. Kavraki, P. Švestka, J.-C. Latombe and M. H. Overmars, Probabilistic roadmaps for path planning in high-dimensional configuration spaces, *IEEE Trans. Robot. Autom.* **12**(4) (1996) 566–580.

[57] S. Karaman and E. Frazzoli, Sampling-based algorithms for optimal motion planning, *Int. J. Robot. Res.* **30**(7) (2011) 846–894.

[58] J. hwan Jeon, Sampling-based motion planning algorithms for dynamical systems, PhD thesis, Massachusetts Institute of Technology (2015).

[59] J. Bialkowski, M. Otte and E. Frazzoli, Free-configuration biased sampling for motion planning, in *IEEE/RSJ Int. Conf. Intelligent Robots and Systems* (IEEE, 2013), pp. 1272–1279.

[60] S. Thomas, M. Morales, X. Tang and N. M. Amato, Biasing samplers to improve motion planning performance, in *Proc. IEEE Int. Conf. Robotics and Automation* (IEEE, 2007), pp. 1625–1630.

[61] J. D. Gammell, S. S. Srinivasa and T. D. Barfoot, Batch informed trees (BIT*): Sampling-based optimal planning via the heuristically guided search of implicit random geometric graphs, in *IEEE Int. Conf. Robotics and Automation (ICRA)* (IEEE, 2015), pp. 3067–3074.

[62] J. D. Gammell, S. S. Srinivasa and T. D. Barfoot, Informed RRT*: Optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic, arXiv:1404.2334.

[63] S. Karaman, M. R. Walter, A. Perez, E. Frazzoli and S. Teller, Anytime motion planning using the RRT*, in *IEEE Int. Conf. Robotics and Automation* (2011), pp. 1478–1483.

[64] O. Arslan and P. Tsiotras, Use of relaxation methods in sampling-based algorithms for optimal motion planning, in *IEEE Int. Conf. Robotics and Automation* (2013), pp. 2421–2428.

[65] O. Salzman and D. Halperin, Asymptotically-optimal motion planning using lower bounds on cost, in *IEEE Int. Conf. Robotics and Automation (ICRA)* (Seattle, WA, 2015), pp. 4167–4172.

[66] L. Janson, E. Schmerling, A. Clark and M. Pavone, Fast marching tree: A fast marching sampling-based method for optimal motion planning in many dimensions, *Int. J. Robot. Res.* **34**(7) (2015) 883–921.

[67] R. Bohlin and L. E. Kavraki, Path planning using lazy PRM, in *Proc. ICRA. Millennium Conf. IEEE Int. Conf. Robotics and Automation. Symposia Proceedings* (Cat. No.00CH37065), Vol. 1 (2000), pp. 521–528.

[68] K. Hauser, Lazy collision checking in asymptotically-optimal motion planning, in *IEEE Int. Conf. Robotics and Automation (ICRA)* (Seattle, WA, 2015), pp. 2951–2957.

[69] F. Gao and S. Shen, Online quadrotor trajectory generation and autonomous navigation on point clouds, in *IEEE Int. Symp. Safety, Security, and Rescue Robotics (SSRR)* (IEEE, 2016), pp. 139–146.

[70] M. Hehn and R. Andrea, Quadrocopter Trajectory Generation and Control, in *Proceedings of 18th IFAC World Congress*, Vol. 44, No. 1 (2011), pp. 1489–1491.

[71] M. Hehn and R. D'Andrea, Real-time trajectory generation for quadrocopters, *IEEE Trans. Robot.* **31** (2015) 877–892.

[72] S. Lai, M. Lan and B. M. Chen, Model predictive local motion planning with boundary state constrained primitives, *IEEE Robot. Autom. Lett.* **4**(4) (2019) 3577–3584.

[73] M. Lan, S. Lai and B. M. Chen, Towards the realtime sampling-based kinodynamic planning for quadcopters, in *11th Asian Control Conf. (ASCC)* (Gold Coast, QLD, 2017), pp. 772–777.

[74] H. Kano, H. Fujioka and C. F. Martin, Optimal smoothing and interpolating splines with constraints, *Appl. Math. Comput.* **218**(5) (2011) 1831–1844.

[75] D. Lam, C. Manzie and M. Good, Model predictive contouring control, in *49th IEEE Conf. Decision and Control (CDC)* (Atlanta, GA, 2010), pp. 6137–6142.

[76] F. Gao, W. Wu, W. Gao and S. Shen, Flying on point clouds: Online trajectory generation and autonomous navigation for quadrotors in cluttered environments, *J. Field Robot.* **36**(4) (2019) 710–733.

[77] B. Zhou, F. Gao, L. Wang, C. Liu and S. Shen, Robust and efficient quadrotor trajectory generation for fast autonomous flight, *IEEE Robot. Autom. Lett.* **4**(4) (2019) 3529–3536.

[78] S. Lai, M. Lan and B. M. Chen, Optimal constrained trajectory generation for quadrotors through smoothing splines, in *IEEE/RSJ Int. Conf. Intelligent Robots and Systems (IROS)* (IEEE, 2018), pp. 4743–4750.

[79] S.-P. Lai, M.-l. Lan, Y.-x. Li and B. M. Chen, Safe navigation of quadrotors with jerk limited trajectory, *Front. Inform. Technol. Electron. Eng.* **20**(1) (2019) 107–119.

[80] B. T. Lopez and J. P. How, Aggressive 3D collision avoidance for high-speed navigation, in *IEEE Int. Conf. Robotics and Automation (ICRA)* (Singapore, 2017), pp. 5759–5765.

[81] N. Bucki and M. W. Mueller, Rapid collision detection for multicopter trajectories, in *IEEE/RSJ Int. Conf. Intelligent Robots and Systems (IROS)* (IEEE, 2019), pp. 7234–7239.

[82] D. J. Webb and J. Van Den Berg, Kinodynamic RRT*: Asymptotically optimal motion planning for robots with linear dynamics, in *IEEE Int. Conf. Robotics and Automation* (IEEE, 2013), pp. 5054–5061.

[83] J. Chen, K. Su and S. Shen, Real-time safe trajectory generation for quadrotor flight in cluttered environments, in *IEEE Int. Conf. Robotics and Biomimetics (ROBIO)* (IEEE, 2015), pp. 1678–1685.

[84] H. Samet and R. A. Earnshaw, An overview of quadtrees, octrees, and related hierarchical data structures, in *Theoretical Foundations of Computer Graphics and CAD* (Springer Berlin Heidelberg, 1988), pp. 51–68.

[85] J. Chen, T. Liu and S. Shen, Online generation of collision-free trajectories for quadrotor flight in unknown cluttered environments, *IEEE Int. Conf. Robotics and Automation (ICRA)* (IEEE, 2016), pp. 1476–1483.

[86] S. Tang and V. Kumar, Safe and complete trajectory generation for robot teams with higher-order dynamics, in *IEEE/RSJ Int. Conf. Intelligent Robots and Systems (IROS)* (Daejeon, 2016), pp. 1894–1901.

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

[88] M. Zucker, N. Ratliff, A. D. Dragan, M. Pivtoraiko, M. Klingensmith, C. M. Dellin, J. A. Bagnell and S. S. Srinivasa, Chomp: Covariant hamiltonian optimization for motion planning, *Int. J. Robot. Res.* **32**(9-10) (2013) 1164–1193.

[89] B. Zhou, F. Gao, J. Pan and S. Shen, Robust real-time uav replanning using guided gradient-based optimization and topological paths, in *Proceedings of 2020 IEEE International Conference on Robotics and Automation (ICRA)* (Paris, France, 2020), pp. 1208–1214.

[90] B. Bonet and H. Geffner, Planning as heuristic search, *Artif. Intelli.* **129**(1-2) (2001) 5–33.

[91] N. Lipovetzky and H. Geffner, Best-first width search: Exploration and exploitation in classical planning, in *31st AAAI Conf. Artificial Intelligence* (2017).

[92] A. Gerevini and I. Serina, Planning as propositional CSP: from walksat to local search techniques for action graphs, *Constraints* **8**(4) (2003) 389–413.

[93] M. Cashmore, M. Fox, D. Long and D. Magazzeni, A compilation of the full PDDL+ language into SMT, in *Workshops at the 30th AAAI Conf. Artificial Intelligence* (2016).

[94] J. Rintanen, Planning as satisfiability: Heuristics, *Artif. Intelli.* **193** (2012) 45–86.

[95] A. Camacho, C. Muise and S. A. McIlraith, From fond to robust probabilistic planning: Computing compact policies that bypass

avoidable deadends, in *26th Int. Conf. Automated Planning and Scheduling* (2016).

[96] J. Fu, A. C. Jaramillo, V. Ng, F. B. Bastani and I.-L. Yen, Fast strong planning for fully observable nondeterministic planning problems, *Ann. Math. Artifi. Intelli.* **78**(2) (2016) 131–155.

[97] P. Haslum, N. Lipovetzky, D. Magazzeni and C. Muise, An introduction to the planning domain definition language, *Synth. Lect. Artifi. Intelli. Mach. Learn.* **13**(2) (2019) 1–187.

[98] B. Bonet, G. Loerincs and H. Geffner, A robust and fast action selection mechanism for planning, in *AAAI/IAAI* (1997), pp. 714–719.

[99] J. Hoffmann, FF: The fast-forward planning system, *AI Mag.* **22**(3) (2001) 57.

[100] E. R. Keyder, J. Hoffmann and P. Haslum, Semi-relaxed plan heuristics, in *22nd Int. Conf. Automated Planning and Scheduling* (2012).

[101] E. Keyder, J. Hoffmann and P. Haslum, Improving delete relaxation heuristics through explicitly represented conjunctions, *J. Artifi. Intelli. Res.* **50** (2014) 487–533.

[102] M. Helmert, P. Haslum, J. Hoffmann and R. Nissim, Merge-and-shrink abstraction: A method for generating lower bounds in factored state spaces, *J. ACM* **61**(3) (2014) p. 16.

[103] S. Sievers, Merge-and-shrink heuristics for classical planning: Efficient implementation and partial abstractions, in *11th Annual Symp. Combinatorial Search* (2018).

[104] S. Richter and M. Westphal, The LAMA planner: Guiding cost-based anytime planning with landmarks, *J. Artifi. Intelli. Res.* **39** (2010) 127–177.

[105] N. Lipovetzky and H. Geffner, Width and serialization of classical planning problems, in *Proc. 20th European Conf. Artificial Intelligence* (IOS Press, 2012), pp. 540–545.

[106] N. Lipovetzky and H. Geffner, A polynomial planning algorithm that beats LAMA and FF, in *Proc. 27th Int. Conf. Automated Planning and Scheduling (ICAPS)* (2017).

[107] E. Keyder and H. Geffner, Soft goals can be compiled away, *J. Artifi. Intelli. Res.* **36** (2009) 547–556.

[108] S. Cresswell and A. Coddington, Compilation of LTL goal formulas into PDDL, in *Proc. 16th European Conf. Artificial Intelligence* (IOS Press, 2004), pp. 985–986.

[109] F. Patrizi, N. Lipoveztky, G. De Giacomo and H. Geffner, Computing infinite plans for ltl goals using a classical planner, in *22nd Int. Joint Conf. Artificial Intelligence* (2011).

[110] A. Albore, H. Palacios and H. Geffner, A translation-based approach to contingent planning, in *21st Int. Joint Conf. on Artificial Intelligence* (2009).

[111] B. Bonet and H. Geffner, Planning under partial observability by classical replanning: Theory and experiments, in *Proc. Int. Joint Con. Artificial Intelligence* (Barcelona, 2011), pp. 1936–1941.

[112] H. Kress-Gazit, G. E. Fainekos and G. J. Pappas, Temporal-logic-based reactive mission and motion planning, *IEEE Trans. Robot.* **25**(6) (2009) 1370–1381.

[113] H. Lin, Mission accomplished: An introduction to formal methods in mobile robot motion planning and control, *Unmanned Syst.* **2**(02) (2014) 201–216.

[114] H. Kress-Gazit, M. Lahijanian and V. Raman, Synthesis for robots: Guarantees and feedback for robot behavior, *Annu. Rev. Control, Robot. Autonom. Syst.* **1** (2018) 211–236.

[115] S. L. Smith, J. Tumová, C. Belta and D. Rus, Optimal path planning under temporal logic constraints, in *IEEE/RSJ Int. Conf. Intelligent Robots and Systems* (IEEE, 2010), pp. 3288–3293.

[116] S. L. Smith, J. Tumová, C. Belta and D. Rus, Optimal path planning for surveillance with temporal-logic constraints, *Int. J. Robot. Res.* **30**(14) (2011) 1695–1708.

[117] M. R. Maly, M. Lahijanian, L. E. Kavraki, H. Kress-Gazit and M. Y. Vardi, Iterative temporal motion planning for hybrid systems in partially unknown environments, in *Proc. 16th Int. Conf. Hybrid Systems: Computation and Control* (ACM, 2013), pp. 353–362.

[118] M. Lahijanian, M. R. Maly, D. Fried, L. E. Kavraki, H. Kress-Gazit and M. Y. Vardi, Iterative temporal planning in uncertain environments with partial satisfaction guarantees, *IEEE Trans. Robot.* **32**(3) (2016) 583–599.

[119] P. Gastin and D. Oddoux, Fast LTL to Büchi automata translation, in *Proc. 13th Int.Conf. Computer Aided Verification (CAV'01)*, eds. G. Berry, H. Comon and A. Finkel, Lecture Notes in Computer Science, Vol. 2102, (Springer, Paris, France, 2001), pp. 53–65.

[120] A. Duret-Lutz, A. Lewkowicz, A. Fauchille, T. Michaud, E. Renault and L. Xu, Spot 2.0 — a framework for LTL and ω-automata manipulation, in *Proc. 14th Int. Symp. Automated Technology for Verification and Analysis (ATVA'16)*, Lecture Notes in Computer Science, Vol. 9938 (Springer, 2016), pp. 122–129.

[121] S. Schwoon and J. Esparza, A note on on-the-fly verification algorithms, in *Int. Conf. Tools and Algorithms for the Construction and Analysis of Systems* (Springer, 2005), pp. 174–190.

[122] M. Lahijanian, S. Almagor, D. Fried, L. E. Kavraki and M. Y. Vardi, This time the robot settles for a cost: A quantitative approach to temporal logic planning with partial satisfaction, in *29th AAAI Conf. Artificial Intelligence*, (2015).

[123] K. Kim, G. Fainekos and S. Sankaranarayanan, On the minimal revision problem of specification automata, *Int. J. Robot. Res.* **34**(12) (2015) 1515–1535.

[124] A. Pnueli and R. Rosner, On the synthesis of a reactive module, in *Proc. 16th ACM SIGPLAN-SIGACT Symp. Principles of Programming Languages* (ACM, 1989), pp. 179–190.

[125] R. Ehlers, R. Könighofer and R. Bloem, Synthesizing cooperative reactive mission plans, in *IEEE/RSJ Int. Conf. Intelligent Robots and Systems (IROS)* (IEEE, 2015), pp. 3478–3485.

[126] S. Dathathri and R. M. Murray, Decomposing gr (1) games with singleton liveness guarantees for efficient synthesis, in *IEEE 56th Annual Conf. Decision and Control (CDC)* (IEEE, 2017), pp. 911–917.

[127] S. Chinchali, S. C. Livingston, U. Topcu, J. W. Burdick and R. M. Murray, Towards formal synthesis of reactive controllers for dexterous robotic manipulation, in *IEEE Int. Conf. Robotics and Automation* (IEEE, 2012), pp. 5183–5189.

[128] R. Ehlers and V. Raman, Slugs: Extensible gr (1) synthesis, in *Int. Conf. Computer Aided Verification* (Springer, 2016), pp. 333–339.

[129] M. Lahijanian, S. Andersson and C. Belta, Control of Markov decision processes from PCTL specifications, in *Proc. American Control Conf.* (IEEE, 2011), pp. 311–316.

[130] X. C. Ding, S. L. Smith, C. Belta and D. Rus, Mdp optimal control under temporal logic constraints, in *50th IEEE Conf. Decision and Control and European Control Conf.* (IEEE, 2011), pp. 532–538.

[131] B. Lacerda, D. Parker and N. Hawes, Optimal and dynamic planning for Markov decision processes with co-safe LTL specifications, in *IEEE/RSJ Int. Conf. Intelligent Robots and Systems* (IEEE, 2014), pp. 1511–1516.

[132] S. Karaman, R. G. Sanfelice and E. Frazzoli, Optimal control of mixed logical dynamical systems with linear temporal logic specifications, in *47th IEEE Conf. Decision and Control* (IEEE, 2008), pp. 2117–2122.

[133] V. Raman, A. Donzé, M. Maasoumy, R. M. Murray, A. Sangiovanni-Vincentelli and S. A. Seshia, Model predictive control with signal temporal logic specifications, in *53rd IEEE Conf. Decision and Control* (IEEE, 2014), pp. 81–87.

[134] S. Saha and A. A. Julius, An MILP approach for real-time optimal controller synthesis with metric temporal logic specifications, in *American Control Conf.* (IEEE, 2016), pp. 1105–1110.

[135] N. T. Dantam, Z. K. Kingston, S. Chaudhuri and L. E. Kavraki, Incremental task and motion planning: A constraint-based approach, *Robotics: Sci. Syst.* (2016) 1–6.

[136] Y. V. Pant, H. Abbas and R. Mangharam, Smooth operator: Control using the smooth robustness of temporal logic, in *IEEE Conf. Control Technology and Applications (CCTA)* (IEEE, 2017), pp. 1235–1240.

[137] N. Mehdipour, C.-I. Vasile and C. Belta, Arithmetic-geometric mean robustness for control from signal temporal logic specifications, in *Proceedings of 2019 American Control Conference (ACC), Philadelphia* (PA, USA, 2019), pp. 1690–1695.

[138] A. Camacho, J. A. Baier, C. Muise and S. A. McIlraith, Finite LTL synthesis as planning, in *28th Int. Conf. Automated Planning and Scheduling* (2018).

[139] J. L. Bresina, A. K. Jónsson, P. H. Morris and K. Rajan, Activity planning for the mars exploration rovers, in *Proc. 15th Int. Conf. Automated Planning and Scheduling, ICAPS'05* (AAAI Press, 2005), pp. 40–49.

[140] S. G. Brunner, F. Steinmetz, R. Belder and A. Dömel, Rafcon: A graphical tool for engineering complex, robotic tasks, in *IEEE/RSJ Int. Conf. Intelligent Robots and Systems (IROS)* (IEEE, 2016), pp. 3283–3290.

[141] V. de Araujo, A. P. G. Almeida, C. T. Miranda and F. de Barros Vidal, A parallel hierarchical finite state machine approach to uav control for search and rescue tasks, in *11th Int. Conf. Informatics in Control, Automation and Robotics (ICINCO)*, Vol. 1 (IEEE, 2014), pp. 410–415.

[142] M. Schütt, P. Hartmann, J. Holsten and D. Moormann, Mission control concept for parcel delivery operations based on a tiltwing aircraft system, in *Advances in Aerospace Guidance, Navigation and Control* (Springer, 2018), pp. 475–494.

[143] A. Marzinotto, M. Colledanchise, C. Smith and P. Ögren, Towards a unified behavior trees framework for robot control, in *IEEE Int. Conf. Robotics and Automation (ICRA)* (IEEE, 2014), pp. 5420–5427.

[144] E. W. Dijkstra, Letters to the editor: go to statement considered harmful, *Commun. ACM* **11**(3) (1968) 147–148.

[145] R. W. Sebesta*Concepts of Programming Languages* (Pearson Education Inc, 2012).

[146] M. Colledanchise and P. Ögren, *Behavior Trees in Robotics and AI: An Introduction* (CRC Press, 2018).

[147] M. Ghallab, D. Nau and P. Traverso, *Automated Planning and Acting* (Cambridge University Press, 2016).

[148] M. Colledanchise, D. Almeida and P. Ögren, Towards blended reactive planning and acting using behavior trees, in *Int. Conf. Robotics and Automation (ICRA)* (IEEE, 2019), pp. 8839–8845.

[149] M. Colledanchise and P. Ogren, How behavior trees modularize hybrid control systems and generalize sequential behavior compositions, the subsumption architecture, and decision trees, *IEEE Trans. Robot.* **33** (2017) 372–389.

[150] J. A. Bagnell, F. Cavalcanti, L. Cui, T. Galluzzo, M. Hebert, M. Kazemi, M. Klingensmith, J. Libby, T. Y. Liu, N. Pollard *et al.*, An integrated system for autonomous robotics manipulation, in *IEEE/RSJ Int. Conf. Intelligent Robots and Systems* (IEEE, 2012), pp. 2955–2962.

[151] C. Paxton, A. Hundt, F. Jonathan, K. Guerin and G. D. Hager, Costar: Instructing collaborative robots with behavior trees and vision, in *IEEE Int. Conf. Robotics and Automation (ICRA)* (IEEE 2017), pp. 564–571.

[152] C. I. Sprague, Ö. Özkahraman, A. Munafo, R. Marlow, A. Phillips and P. Ögren, Improving the modularity of AUV control systems using behaviour trees, in *IEEE/OES Autonomous Underwater Vehicle Workshop (AUV)* (IEEE, 2018), pp. 1–6.

[153] D. Hu, Y. Gong, B. Hannaford and E. J. Seibel, Semi-autonomous simulated brain tumor ablation with ravenii surgical robot using behavior tree, in *IEEE Int. Conf. Robotics and Automation (ICRA)* (IEEE, 2015), pp. 3868–3875.

[154] K. R. Guerin, C. Lea, C. Paxton and G. D. Hager, A framework for end-user instruction of a robot assistant for manufacturing, in *IEEE Int. Conf. Robotics and Automation (ICRA)* (IEEE, 2015), pp. 6167–6174.

[155] Smart assembly robot with advanced functionalities.

[156] M. Lan, Y. Xu, S. Lai and B. M. Chen, A modular mission management system for micro aerial vehicles, in *IEEE 14th Int. Conf. Control and Automation (ICCA)* (IEEE, 2018), pp. 293–299.

[157] Y. V. Pant, H. Abbas, R. A. Quaye and R. Mangharam, Fly-by-logic: Control of multi-drone fleets with temporal logic objectives, in *ACM/IEEE 9th Int. Conf. Cyber-Physical Systems (ICCPS)* (IEEE, 2018), pp. 186–197.

[158] M. Kloetzer and C. Belta, A fully automated framework for control of linear systems from temporal logic specifications, *IEEE Trans. Automatic Control* **53**(1) (2008) 287–297.

[159] L. Habets and J. H. Van Schuppen, A control problem for affine dynamical systems on a full-dimensional polytope, *Automatica* **40** (1) (2004) 21–35.

[160] S. Karaman and E. Frazzoli, Sampling-based motion planning with deterministic μ-calculus specifications, in *Proc. 48h IEEE Conf. Decision and Control (CDC) Held Jointly with 2009 28th Chinese Control Conf.* (IEEE, 2009), pp. 2222–2229.

[161] C. I. Vasile and C. Belta, Sampling-based temporal logic path planning, in *IEEE/RSJ Int. Conf. Intelligent Robots and Systems*, (IEEE, 2013), pp. 4817–4822.

[162] C. I. Vasile and C. Belta, Reactive sampling-based temporal logic path planning, in *2014 IEEE In. Conf. Robotics and Automation (ICRA)* (IEEE, 2014), pp. 4310–4315.

[163] L. Larocque and J. Liu, Sampling-based motion planning with μ-calculus specifications without steering, in *Proceedings of 2018 IEEE International Conference on Robotics and Automation (ICRA)*, Brisbane, QLD, 2018, pp. 4355–4360.

[164] Y. Li, Z. Littlefield and K. E. Bekris, Asymptotically optimal sampling-based kinodynamic planning, *Int. J. Robot. Res.* **35**(5) (2016) 528–564.

[165] Y. Kantaros and M. M. Zavlanos, Sampling-based optimal control synthesis for multirobot systems under global temporal tasks, *IEEE Trans. Automatic Control* **64**(5) (2018) 1916–1931.

[166] S. Coradeschi, A. Loutfi and B. Wrede, A short review of symbol grounding in robotic and intelligent systems, *KI-Künstliche Intelligenz* **27**(2) (2013) 129–136.

[167] S. Penkov, A. Bordallo and S. Ramamoorthy, Physical symbol grounding and instance learning through demonstration and eye tracking, in *IEEE Int. Conf. Robotics and Automation (ICRA)* (IEEE, 2017), pp. 5921–5928.

[168] S. Srivastava, E. Fang, L. Riano, R. Chitnis, S. Russell and P. Abbeel, Combined task and motion planning through an extensible planner-independent interface layer, in *IEEE Int. Conf. Robotics and Automation (ICRA)* (IEEE, 2014), pp. 639–646.

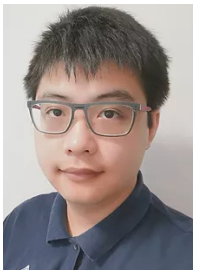
[169] E. Plaku, L. E. Kavraki and M. Y. Vardi, Discrete search leading continuous exploration for kinodynamic motion planning, *Robotics: Sci. Syst.* (2007) 326–333.

[170] E. Plaku and G. D. Hager, Sampling-based motion and symbolic action planning with geometric and differential constraints, in *IEEE Int. Conf. Robotics and Automation (ICRA)* (IEEE, 2010), pp. 5002–5008.

[171] C. R. Garrett, T. Lozano-Pérez and L. P. Kaelbling, Sampling-based methods for factored task and motion planning, *Int. J. Robotics Research* **37**(13-14) (2018) 1796–1825.

[172] M. Lan, S. Lai, T. H. Lee and B. M. Chen, Autonomous task planning and acting for micro aerial vehicles, in *15th Int. Conf. Control and Automation (ICCA)* (IEEE, 2019), pp. 738–745.

Menglu Lan received her B.Eng. from the Department of Electrical and Computer Engineering, National University of Singapore, and her Ph.D. degree from the Graduate School for Integrative Sciences and Engineering, National University of Singapore. Her research interests include task planning, motion planning for aerial vehicles and the application of formal methods to robotic systems.



Shupeng Lai received his B.Eng. degree from the School of Electrical and Electronic Engineering, Nanyang Technological University, Singapore and his Ph.D. degree from the Graduate School for Integrative Sciences and Engineering, National University of Singapore. His research interests include motion planning, nonlinear model predictive control and multi-agent robotic systems.

Tong H. Lee received the B.A. degree with First Class Honors in the Engineering Tripos from Cambridge University, England, in 1980; the M.Eng. degree from NUS in 1985; and the Ph.D. degree from Yale University in 1987. He is a Professor in the Department of Electrical and Computer Engineering at the National University of Singapore (NUS); and also a Professor in the NUS Graduate School, NUS NGS. He was a Past Vice-President (Research) of NUS.

Dr. Lee's research interests are in the areas of adaptive systems, knowledge-based control, intelligent mechatronics and computational intelligence. He currently holds Associate Editor appointments in the IEEE Transactions in Systems, Man and Cybernetics; Control Engineering Practice (an IFAC journal); and the International Journal of Systems Science (Taylor and Francis, London). In addition, he is the Deputy Editor-in-Chief of IFAC Mechatronics journal.

Ben M. Chen is currently a Professor of Mechanical and Automation Engineering at the Chinese University of Hong Kong and a Professor of Electrical and Computer Engineering (ECE) at the National University of Singapore (NUS). He was a Provost's Chair Professor in the NUS ECE Department, where he also served as the Director of Control, Intelligent Systems and Robotics Area, and Head of Control Science Group, NUS Temasek Laboratories. He was an Assistant Professor at the State University of New York at Stony Brook, in 1992–1993. His current research interests are in unmanned systems, robust control and control applications.

Dr. Chen is an IEEE Fellow. He has authored/co-authored more than 450 journal and conference articles, and a dozen research monographs in control theory and applications, unmanned systems and financial market modeling. He had served on the editorial boards of a dozen international journals including Automatica and IEEE Transactions on Automatic Control. He currently serves as an Editor-in-Chief of Unmanned Systems. Dr. Chen has received a number of research awards. His research team has actively participated in international UAV competitions and won many championships in the contests.