Failsafe mechanism design of multicopters based on supervisory control theory

Quan Quan1, Zhiyao Zhao2, Liyong Lin3, Peng Wang1, Walter Murray Wonham2, Kai-Yuan Cai1

1 School of Automation Science and Electrical Engineering, Beihang University, Beijing 100191, People's Republic of China
2 School of Computer and Information Engineering, Beijing Technology and Business University, No. 11/33 Fucheng Road, Haidian District, Beijing 100048, People's Republic of China
3 Department of Electrical and Computer Engineering, University of Toronto, Toronto, ON M5S 3G4, Canada

Abstract: In order to handle undesirable failures of a multicopter, which occurs in either the pre-flight process or the in-flight process, a failsafe mechanism design method based on supervisory control theory (SCT) is proposed for the semi-autonomous control mode. The failsafe mechanism is a control logic that guides what subsequent actions the multicopter should take, by taking account of real-time information from guidance, attitude control, diagnosis and other low-level subsystems. In order to design a failsafe mechanism for the multicopters, safety issues of the multicopters are introduced. Then, user requirements including functional requirements and safety requirements are textually described, where functional requirements guide the modelling of a general multicopter plant, and safety requirements cover the failsafe measures dealing with the presented safety issues. Based on these requirements, several multicopter modes and events are defined. On this basis, the multicopter plant and control specifications are modelled by automata. Then, a supervisor is synthesized by using SCT. In addition, the authors present three examples to demonstrate the potential conflicting phenomena due to the inappropriate design of control specifications. Finally, based on the obtained supervisor, an implementation method suitable for multicopters is presented, in which the supervisor is transformed into decision-making codes.

1 Introduction

Multicopters are well-suited to a wide range of mission scenarios, such as search and rescue, package delivery, border patrol, military surveillance and agricultural production [1]. In either pre-flight process or in-flight process, multicopter failures cannot be absolutely avoided. These failures may abort missions, crash multicopters and moreover, injure or even kill people. In order to handle undesirable failures in industrial systems, a technique named Prognostics and Health Management (PHM) is presented [2]. As shown in Fig. 1, an integrated PHM system generally contains three levels: monitoring, prediction and management [3]. On the one hand, the monitoring and prediction levels assess the quantitative health results from the monitoring and prediction levels and then responds to meet qualitative safety or health requirements. This paper aims to study a safety decision-making logic by using supervisory control theory (SCT) to guarantee flight safety from a qualitative perspective.

In the framework of multicopters, guidance, attitude control, PHM and other low-level subsystems work together under the coordination of a high-level decision-making module [8]. In this module, the failsafe mechanism is an important part. It is a control logic that receives information from all subsystems to decide the best flight manoeuvre from a global perspective and send flight instructions to low-level subsystems [9]. However, current academic literature covering failure-related topics of multicopters mainly focuses on fault detection techniques and fault-tolerant control algorithms [10–14], which belong to a study of low-level subsystems. For the study of the high-level decision-making module, most research focuses on path planning [15–17] and obstacle avoidance [18, 19] of an individual multicopter, or PHM-based mission allocation of a multicopter team [20, 21]. However, few studies have focused on the failsafe mechanism design of an individual multicopter subject to multiple potential failures. Ten Hamsel et al. [22] proposed an emergency flight planning for an energy-constrained situation. De Smet et al. [23] proposed a failsafe design for an uncontrollable situation. Johry and Kapoor [24] designed multiple failsafe measures dealing with different anomalies of unmanned aerial vehicles. Nevertheless, these research works only consider certain ad hoc failsafe mechanisms for certain faults or anomalies, and so far do not present a comprehensive failsafe mechanism for a multicopter. In current autopilot products (for example, DJI autopilot [25] and ArduPilot [26]), there exist comprehensive failsafe mechanisms to cope with communication, sensor and battery failures, but such mechanisms are either proprietary or can be accessed only in part. Moreover, to the best of the authors' knowledge, these failsafe mechanisms are mainly developed according to engineering experience [Engineering experience, while invaluable, may nevertheless be susceptible to subtle errors of logic when dealing with complex systems. It is here that formal methods of synthesis like SCT can play an important complementary role in enforcing rigorous design specifications. A final design should always be confirmed by both criteria.]. As a result, such a development process lacks a theoretical foundation; this will inevitably lead to man-made mistakes, logical bugs or incomplete treatment. Motivated by these, this paper first summarizes safety issues and user requirements for multicopters in the semi-autonomous control mode. Most

Fig. 1 PHM framework
multicopters have two high-level control modes: semi-autonomous control and full-autonomous control. Many open source autopilots support both modes. The semi-autonomous control mode implies that autopilots can be used to stabilize the attitude of the multicopters, and also they can help the multicopters hold the altitude and position. Under such a mode, a multicopter will be still under the control of remote pilots. On the other hand, the fully autonomous control mode implies that the multicopter can follow a pre-programmed mission script stored in the autopilot which is made up of navigation commands and also can take off and land automatically. Under such a mode, remote pilots on the ground only need to schedule the tasks [1] and then uses SCT of discrete event systems (DESs) to design a failsafe mechanism of multicopters.

SCT [27, 28], also known as Ramadge–Wonham (RW) framework, is a method for synthesising supervisors that restrict the behaviour of a plant such that the given control specifications are fulfilled and never violated. Currently, SCT has been developed with a solid theoretical foundation [29–31], and it has been successfully applied to practical systems such as flexible manufacturing systems [32–35] and patient support systems [36]. A recent overview can be found in [37]. This paper formalizes the problem of failsafe mechanism design as a DES control problem. The solution procedure is shown in Fig. 2. In order to obtain the desired failsafe mechanism, the following steps are performed: (i) define related modes and events by studying the user requirements (including functional and safety requirements); (ii) model the multicopter plant by transforming the functional requirements to an automaton with defined modes and events; (iii) analyse the safety requirements by taking the defined modes and events into account, and transform the safety requirements to automata as control specifications; (iv) synthesise the supervisor by SCT software; (v) implement the failsafe mechanism based on the obtained supervisor.

The contributions of the paper mainly lie in two aspects.

• First, this paper introduces SCT into a new application area. The proposed SCT-based method sets a solid theoretical foundation for designing the failsafe mechanism of multicopters. In the field of aircraft engineering, especially of multicopters and drones, traditional design methods are based on engineering experience. The failsafe mechanism obtained by these methods may be problematic (for example, the failsafe mechanism may contain unintended deadlocks), especially when multiple safety issues are taken into account. Compared to existing empirical design methods, the proposed method can guarantee the correctness and effectiveness of the obtained failsafe mechanism, which is an urgent need for multicopter designers and manufacturers.

• Second, as a practical application of SCT, this paper emphasizes the modelling process of the plant and control specifications rather than developing a new theory of SCT. We believe this work is important to both the development of SCT research and practical engineering because SCT is presented with rich mathematical terminology and theory which many engineers may not understand. Motivated by this, this paper presents the procedure of applying SCT to an engineering problem, from requirements described textually, to a plant and specifications in the form of automata, then to a synthesized supervisor and finally to implementation on a real-time flight simulation platform of quadcopters developed by MATLAB. In addition, we present three examples to demonstrate the potential conflicting phenomena due to inappropriate design of control specifications. From the perspective of practitioners, this paper may serve as a guide for those engineers who are not familiar with SCT to solve their own problems in their own projects by using SCT and related software.

The remainder of this paper is organized as follows. Section 2 presents preliminaries of SCT. Section 3 lists some relevant safety issues of multicopters. Also, user requirements including functional requirements and safety requirements are textually described. In order to transform the user requirements to automata, the relevant multicopter modes and events are defined in Section 4. On this basis, a detailed modelling process of the multicopter plant and control specifications is presented in Section 5. Then, TCT software is used to perform supervisor synthesis. Section 6 uses three examples to demonstrate some possible reasons leading to a conflict and provides a brief discussion about the scope of applications and properties of the proposed method. Section 7 shows an implementation process of the proposed failsafe mechanism on the platforms MATLAB and FlightGear. Section 8 presents our conclusions and suggests some future research.

2 Preliminaries on supervisory control theory

As SCT is well established, readers can refer to textbooks [27, 38, 39] for detailed background and knowledge. This section only reviews some basic concepts and notation.

• Automaton: In RW theory [27, 28], the formal structure of DES is modelled by an automaton (generator)

\[ G = (Q, \Sigma, \delta, q_0, Q_m) \]

where \( Q \) is the finite state set; \( \Sigma \) is the finite event set (also called an alphabet); \( \delta : Q \times \Sigma \rightarrow Q \) is the (partial) transition function; \( q_0 \in Q \) is the initial state; \( Q_m \subseteq Q \) is the set of marker states. A marker state is a state that it is desired to reach. For SCT, the alphabet \( \Sigma \) is partitioned as

\[ \Sigma = \Sigma_c \cup \Sigma_a \]

where \( \Sigma_c \) is the set of controllable events that can be stopped or started by will and \( \Sigma_a \) is the set of uncontrollable events that cannot be stopped or started by will.

• Plant: Physical systems or processes in DESs are often modelled as plants in the form of automata. A plant contains all possible behaviours of the considered system or process, i.e. all possible event occurrence sequences. A single, more complex plant can be composed of many smaller components in the form of automata by using the synchronous product. [For two automata \( G_i = (Q_i, \Sigma, \delta_i, q_{i0}, Q_{i_m}), i = 1, 2 \), their interaction is captured by the synchronous product \( G = (Q, \Sigma, \delta, q_0, Q_m) \) of \( G_1 \) and \( G_2 \), denoted by \( G_i \parallel G_j \) [27, Chapter 3.3]. The synchronous product of more than two automata can be constructed similarly.]
In order to explain these concepts above, we use a simplified small factory example as shown in Fig. 3. As shown in Fig. 3(a), the transition graph represents two simplified machines named MACH1, with two states, labelled $I_1$, $W_1$ for ‘idle’ and ‘working’, respectively, $i = 1, 2$. MACH1 executes a sequence of events in accordance with its transition graph. For MACH1, the finite state set is $Q = \{I_1, W_1\}$, the finite event set is $\Sigma = \{\alpha_i, \beta_i\}$, the transition function is given by $\delta(I_1, \alpha_1) = W_1$, $\delta(W_1, \beta_1) = I_1$, the initial state is $q_0 = I_1$, and the marker state set is $Q_m = \{I_1, W_1\}$. As shown in Fig. 3(a), a small factory operates as follows. Initially, the buffer is empty. With the event $\alpha_1$, MACH1 takes a workpiece from an infinite input bin and enters $W_1$ from $I_1$. Subsequently, MACH1 returns to $I_1$ (event $\beta_1$). MACH2 operates similarly, but takes its workpiece from the BUFF (event $\alpha_2$) and enters $W_2$, and then deposits it when finished in an infinite output bin (event $\beta_2$). The text-based requirement for admissible operation is: the buffer must not overflow or underflow. The requirement is translated into a formal specification in the form of an automaton, namely BUFF as shown in Fig. 3b. BUFF has two states, labelled $E, F$ for ‘empty’ and ‘full’. Here, MACH1 and MACH2 are plants, and BUFF is the specification. The events $\alpha_1, \alpha_2$ are controllable events, but $\beta_1, \beta_2$ are uncontrollable events because they simply happen in accordance with internal machine volition. The synchronous product of two plants, namely MACH1 MACH2, is shown in Fig. 4a, where, for example, the new state $I_1W_1$ represents that MACH1 is at $I_1$ and MACH2 is at $W_1$. After applying the synthesis procedure, we can get an automaton shown in Fig. 4b, where, for example, the new state $FIW_1$, where, for example, the new state $FIW_1$, represents that BUFF is at $F$, MACH1 is at $I_1$ and MACH2 is at $W_1$. The states $FW,FW,$ and $FW,F$ are forbidden states because the buffer may overflow (two workpieces are in the buffer) due to the uncontrollable event $\beta_1$, namely the requirement will be violated. For example, MACH1 may finish its job (event $\beta_1$ happens; this is uncontrollable or unpredictable) before MACH2 takes an existing workpiece from the buffer (event $\alpha_2$ happens). To prevent this, we need to disable the controllable event $\alpha_2$ at the state $FIW_1$ so that the forbidden state $FW,F$ cannot be reached. Similarly, we will disable the controllable event $\alpha_1$ at the state $FII$. Finally, we can get the supervisor as shown in Fig. 4c. With such a supervisor, the requirement will not be violated.

3 Safety issues and user requirements

This section lists some relevant safety issues of multicopters. Also, user requirements including functional requirements and safety requirements are textually described.
3.1 Safety issues

Major types of multicopter failures that may cause accidents will be introduced. Here, three types of failures are considered, including communication breakdown, sensor failure and propulsion system anomaly.

- Communication breakdown: Communication breakdown mainly refers to a contact anomaly between the remote controller (RC) transmitter and the multicopter or between the ground control station and the multicopter. In this paper, for simplicity, only RC is considered.
- Sensor failure: Sensor failure mainly means that a sensor on the multicopter cannot accurately measure related signals, or cannot work properly. This paper considers the sensor failures including barometer failure, compass failure, GPS failure and inertial navigation system (INS) failure.
- Propulsion system anomaly: Propulsion system anomaly mainly refers to battery failure and propulsor failure caused by electronic speed controllers, motors or propellers.

More information about safety issues can be found in the book [1, Chapter 14.2].

3.2 User requirements

For safety, a multicopter product is required to have general functions (functional requirements), and also be capable of coping with relevant safety issues. Functional requirements and safety requirements are listed in Tables 1–4, respectively. They are summarized from the material in [26] and the authors’ knowledge and engineering experience.

(i) Functional requirements: The following functional requirements describe what behaviour the multicopter is able to perform.

(ii) Safety requirements: The safety requirements restrict what actions the user wants the multicopter to perform under specific situations when it is on the ground, in flight, or in the process of returning and landing.

4 Multicopter mode and event definition

In order to transform the user requirements to automata, several multicopter modes and events are defined in this section.

4.1 Multicopter mode

Referring to [1, 26], the whole process from taking off to landing of multicopters is divided into eight multicopter modes. They form the basis of the failsafe mechanism.

- POWER OFF MODE: This mode means that a multicopter is turned off. In this mode, the remote pilot can (possibly) disassemble, maintain and replace the hardware of a multicopter.
- STANDBY MODE: When a multicopter is connected to the power module, it enters a pre-flight status. In this mode, the multicopter has not been armed, and the remote pilot can arm the multicopter manually. Afterwards, the multicopter will perform a safety check and then transit to the next mode according to the results of the safety check.
- GROUND-ERROR MODE: This mode indicates that the multicopter has a safety problem. In this mode, the buzzer will turn on an alarm to alert the remote pilot that there exist errors in the multicopter.
- LOITER MODE: Under this mode, the remote pilot can use the control sticks of the RC transmitter to control the multicopter. When the remote pilot releases the two control sticks, the multicopter will slow to a stop, and automatically maintain the current location, heading and altitude.
- ALTITUDE-HOLD MODE: When the throttle control stick is in the mid-throttle deadzone, the throttle is automatically controlled to maintain the current altitude and the attitude is also stabilized but the horizontal position drift will occur. When the throttle control stick goes outside the mid-throttle deadzone, the multicopter will descend or climb depending upon the deflection of the control stick.
- STABILIZE MODE: This mode allows a remote pilot to fly the multicopter manually, but self-levels the roll and pitch axes. When the remote pilot releases the roll and pitch control sticks,
Table 4  Safety requirements for returning and landing

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SR9</td>
<td>when the multicopter is in the process of returning to base, the multicopter can be manually switched by the RC transmitter to normal flight or landing</td>
</tr>
<tr>
<td>SR10</td>
<td>when the multicopter is in the process of returning to the base, if the distance to base is less than a given threshold, the multicopter should switch to landing; if the battery's capacity becomes inadequate and unable to return to base, the multicopter should switch to landing; if the INS, GPS, barometer, compass or propulsors are unhealthy, the multicopter should switch to landing</td>
</tr>
<tr>
<td>SR11</td>
<td>when the multicopter is in the process of landing, the multicopter can be manually switched by the RC transmitter to normal flight. This switch requires that the INS and propulsors are both healthy, the connection to the RC transmitter is normal, and the battery's capacity is adequate. Otherwise, the switch is ignored by the multicopter</td>
</tr>
<tr>
<td>SR12</td>
<td>when the multicopter is in the process of landing, the multicopter can be manually switched by the RC transmitter to returning to base. This switch requires that the INS, GPS, barometer, compass, propulsors are all healthy, the battery's capacity is able to support the multicopter to return to base, and the distance to the base is not less than a given threshold. Otherwise, the switch is ignored by the multicopter</td>
</tr>
<tr>
<td>SR13</td>
<td>when the multicopter is in the process of landing, if the multicopter's altitude is lower than a given threshold, or the multicopter's throttle is less than a given threshold over a time horizon, the multicopter can be automatically disarmed</td>
</tr>
</tbody>
</table>

Table 5  MIE definition

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MIE1</td>
<td>the power is on</td>
</tr>
<tr>
<td>MIE2</td>
<td>the power is off</td>
</tr>
<tr>
<td>MIE3</td>
<td>arm action is executed by remote pilots</td>
</tr>
<tr>
<td>MIE4</td>
<td>disarm action is executed by remote pilots</td>
</tr>
<tr>
<td>MIE5</td>
<td>other actions on the sticks by remote pilots. These actions also include the do-nothing operation</td>
</tr>
<tr>
<td>MIE6</td>
<td>switch to normal flight. In normal flight, the multicopter can be in either LOITER MODE, ALTITUDE-HOLD MODE or STABILIZE MODE</td>
</tr>
<tr>
<td>MIE7</td>
<td>switch to RTL MODE</td>
</tr>
<tr>
<td>MIE8</td>
<td>switch to AL MODE</td>
</tr>
</tbody>
</table>

Table 6  MCE definition

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MCE1</td>
<td>multicopter switched to POWER OFF MODE</td>
</tr>
<tr>
<td>MCE2</td>
<td>multicopter switched to STANDBY MODE</td>
</tr>
<tr>
<td>MCE3</td>
<td>multicopter switched to GROUND ERROR MODE</td>
</tr>
<tr>
<td>MCE4</td>
<td>multicopter switched to LOITER MODE</td>
</tr>
<tr>
<td>MCE5</td>
<td>multicopter switched to ALTITUDE-HOLD MODE</td>
</tr>
<tr>
<td>MCE6</td>
<td>multicopter switched to STABILIZE MODE</td>
</tr>
<tr>
<td>MCE7</td>
<td>multicopter switched to RTL MODE</td>
</tr>
<tr>
<td>MCE8</td>
<td>multicopter switched to AL MODE</td>
</tr>
</tbody>
</table>

The multicopter automatically stabilizes its attitude but position drift may occur.

- **RETURN-TO- LAUNCH (RTL) MODE**: Under this mode, the multicopter will return to the base location from the current position, and hover there.
- **AUTOMATIC-LANDING (AL) MODE**: In this mode, the multicopter realizes landing automatically by adjusting the throttle according to the estimated height [Even if the barometer fails, the height estimation is acceptable within a short time. Similarly, the other estimates generated by filters could continue to be used for a short time, even if related sensors fail.].

4.2 Event definition

Three types of events are defined here: manual input events (MIEs), mode control events (MCEs) and automatic trigger events (ATEs). The failsafe mechanism takes MIEs and ATEs as inputs, and outputs MCEs to decide which mode the multicopter should stay in or switch to. Here, MIEs and MCEs are controllable, while ATEs are uncontrollable in the sense of SCT.

(i) **MIEs**: MIEs are instructions from the remote pilot sent through the RC transmitter. This part defines eight MIEs as shown in Table 5. Here, MIE3, MIE4 are realized by certain successive actions on the control sticks of the RC transmitter as shown in Fig. 5; MIE6, MIE7 and MIE8 are realized by the three-position switch (namely the flight mode switch) on the RC transmitter as shown in Fig. 5. It is noticed that the elements in (MIE1, MIE2) are defined mutually exclusively, namely, one and only one is always true for all time. Also, the elements in (MIE3, MIE4, MIE5) and the elements in (MIE6, MIE7, MIE8) are defined mutually exclusively.

(ii) **MCEs**: MCEs are instructions from multicopter's autopilot. As shown in Table 6, these events will control the multicopter to switch to a specified multicopter mode defined in Section 4.1. It is noticed that the events in (MCE1, ..., MCE8) are mutually exclusive. So, enabling one event is equivalent to disabling others.

(iii) **ATEs**: ATEs are independent of the remote pilot's operations. As shown in Table 7, these events contain the health check results and flight status of a multicopter. It is noticed that the events in (ATE1, ATE2), ..., (ATE11, ATE12), (ATE13, ATE14, ATE15), (ATE16, ATE17), ..., (ATE20, ATE21) are mutually exclusive.

Here, note that this paper assumes the component health check can be performed by effective fault diagnosis and health evaluation methods. For simplified presentation, the statements of ‘check result of’ and ‘measured’ are omitted in the subsequent sections.

5  Failsafe mechanism design

In this section, functional requirements guide the modelling of the multicopter plant based on defined multicopter modes and events. Then, from the safety requirements, multiple control specifications are represented by automata. These control specifications should indicate the preferable failsafe measures that are consistent with the textually described safety requirements. After the plant and control specifications have been obtained, a supervisor is synthesized by using monolithic (namely fully centralized) supervisory control [27, Chapter 4.6].

5.1 Multicopter plant modelling

(i) Modelling principles: Modelling the multicopter plant in the form of an automaton is to mathematically describe what behaviour the multicopter is able to perform, which is mainly based on functional requirements. In this paper, the modelling principles of the multicopter plant include: (i) modelling the ‘on ground’ component and the ‘in air’ component one by one; (ii) events of each transition modelled mutually exclusively, namely one and only one event will be always executed. As shown in Fig. 6, the transition from S₆ to S₇ is related to the arm action, disarm action and other actions on the sticks, associated with three mutually exclusive events. The plant allows that the three events can happen at S₆. As shown in Fig. 6, the transition from S₇ to S₆ is related to...
the five multicopter mode decisions, after which health checking (the loop from $S_0$ to $S_0$) is required. The plant allows that the five events can happen. However, in our resulting supervisor, only one event is allowed to happen at $S_0$, if the transition from $S_0$ to $S_1$ is performed.

(ii) Model details: As shown in Fig. 7, the plant describes basic functions of a multicopter. Specifically, Plant contains 27 states ($S_0$–$S_{26}$), 37 events and 63 transitions. Here, the states $S_0$, $S_1$, $S_2$, $S_4$, $S_5$ are marker states. The state $S_0$ can be understood as POWER OFF MODE; the state $S_1$ can be understood as STANDBY MODE; the state $S_4$ can be understood as GROUND-ERROR MODE; the state $S_5$ can be understood as another multicopter modes, including LOITER MODE, ALTITUDE-HOLD MODE, STABILIZE MODE, RTL MODE and AL MODE. Since the event occurrence sequences are of principal interest, the meaning of states, in fact, can be ignored. The desired behaviour will be defined in specifications. Plant can be divided into two parts: one (consists of states $S_0$–$S_{13}$ and transitions among them) describes the multicopter behaviour on the ground (‘on ground’ component), and the other one (consists of states $S_{13}$–$S_{26}$ and transitions among them) describes the behaviour during the flight (‘in air’ component). The model of the plant is not easy to divide into many small components (each one corresponds to a physical component), so we use a monolithic model. This is unlike many existing examples. As in the small factory example in Section 2, the plant is composed of three physical components explicitly, namely two machines and a buffer.

5.2 Control specification design

(i) Modelling principle: In this part, control specifications are designed to restrict the behaviour of Plant according to the description of the safety requirements. In order to guarantee the completeness and only one MCE to occur, [A multicopter itself has to decide to enter one of eight multicopter modes defined in Section 4.1 depending on the specifications. This may be unlike some SCT problems that allow multiple events to be chosen at one state.] the control specifications must cover all possible strings in the plant.

(ii) Control specification design ‘on ground’: Through a study of textual safety requirements, it can be seen that SR1 in Table 2 describes the intended failsafe measure when the multicopter is on the ground. In other words, SR1 restricts what action the user...
In safety requirement SR1 in Table 2, the user lists the required conditions for a successful arm. In order to model it with an automaton, the key is to split the branches in the ‘on ground’ component of Plant, and enable only one mode which the user expects the multicopter to switch to. Following this principle, a control specification named Specification 1 is designed as shown in Fig. 8. It contains eight states $S_0$, $S_1$, 24 events and 68 transitions. Here, the states $S_0$, $S_1$ are marker states. The state $S_0$ can be understood as STANDBY MODE, and the state $S_1$ can be understood as other multicopter modes. The details of other control specifications are presented in the support material available in http://rfly.buaa.edu.cn/resources. It should be noted that MIE3 and MIE4 (arm and disarm actions on the RC transmitter) can be also enabled by the remote pilot (maybe a pilot error), but it does not imply that the propellers of the multicopter must be locked. In most cases, MIE3 and MIE4 will be ignored (according to our specifications) when a multicopter is in the air, namely, these actions will not be responded to by the multicopter.

5.3 Supervisor synthesis on TCT software

Related algorithms in SCT can be performed on software platforms such as TCT software [27], Supremica [40] and Discrete Event Control Kit [41]. The algorithms and operations in this part are performed on TCT software. The multicopter plant is named as ‘PLANT’, and the 25 control specifications are named as ‘$E_j$', $j = 1, 2, \ldots, 25$. The source files are given in http://rfly.buaa.edu.cn/resources.

Step 1: In the monolithic supervisory control framework, all the control specifications should be synchronized into a monolithic one, i.e.

$$E = \text{sync}(E_1, E_2, \ldots, E_{25}).$$

It turns out that $E$ is non-blocking, and contains 133 states and 2092 transitions.

Step 2: Here, note that PLANT contains 37 events, while the number of events in each $E_j$ is <37 (i.e. the alphabet of each $E_j$ is different from that of the PLANT). This is because the given textual safety requirements only emphasise the events we are concerned with and ignore the remaining events. For supervisory control, the alphabet of $E$ should be equal to the alphabet of PLANT. Thus, the control specification should be completed by the following TCT instructions:

$$\text{EVENTS} = \text{all events(PLANT)}$$

where $\text{EVENTS}$ is a self-loop automaton containing all events in the alphabet of PLANT. Then, for $E$, we have

$$E = \text{sync}(E, \text{EVENTS}).$$
Fig. 9 Traversal relation between 24 control specifications and the structure of the ‘in air’ component of Plant

Here, the events present in PLANT but not in E are added into E in the form of self-loops.

Step 3: A monolithic supervisor is synthesized by

\[ S = \text{supcon}(\text{PLANT}, E) \]

The obtained supervisor is the expected failsafe mechanism. It contains 790 states, 37 events and 1566 transitions. There are eight marker states, which correspond to eight multicopter modes, respectively. Besides monolithic supervisory control, the supervisor can also be synthesized by decentralized supervisory control [42], and a supervisor reduction [43] process can also be carried out for an easier realisation in practice. The synthesis is also carried out in the software Supremica with the same result as with TCT. These source files are presented in http://rfly.buaa.edu.cn/resources.

6 Examples and discussion

In SCT, plants and specifications are supposed to be correct even if specifications may conflict with each other. The resulting supervisor will automatically remove conflicts such as blocking states and satisfy all requirements. In fact, we may consider such requirements and specifications not correct because conflict is not expected by the designers. In this sense, the design of control specifications is a process to understand and re-organize these requirements. If designers synthesise an empty supervisor or find conflict in specifications by SCT software, the correctness of requirements and specifications needs to be rechecked and then corresponding modifications need to be made. This section illustrates three examples to demonstrate some possible reasons leading to conflict and gives a brief discussion about the scope of applications and properties of the method.

6.1 Examples

Example 1: The aim of this example is to show that missing information in specifications may lead to conflict. In this example, we delete transitions ‘S_6 \rightarrow ATE13 \rightarrow S_6’, ‘S_6 \rightarrow ATE14 \rightarrow S_6’ and ‘S_6 \rightarrow ATE15 \rightarrow S_6’ in Specification 1. In this case, Specification 1 is changed to an automaton named as Specification 1’. This implies that ATE13/14/15 cannot occur at state S_6. By replacing Specification 1 with Specification 1’, a conflict will arise after the occurrence of ATE11/12, because the next possible event defined in the plant model is ATE13/14/15, which are all disabled by Specification 1’. This will be shown by some SCT software, such as Supremica (‘purge result’ is not used). The problematic trace is depicted in Fig. 11. Through this, we can locate problems in specifications and make modifications.

Example 2: The aim of this example is to show the conflict in specifications. In this example, we replace the transition ‘S_6 \rightarrow MCE2 \rightarrow S_6’ with a transition ‘S_6 \rightarrow MCE3 \rightarrow S_6’ in Specification 1, resulting in Specification 1”. This implies that Specification 1 and Specification 1” will have a conflict. By adding Specification 1” to the whole control specification, Specification 1 and Specification 1” will exist simultaneously. A conflict will arise after the occurrence of ATE13/14/15 because the next event is MCE2 according to Specification 1 but MCE3 according to Specification 1”. This will be shown by some SCT software, such as Supremica (‘purge result’ is not used). The problematic trace is depicted in Fig. 12. Through this, we can locate problems in specifications and make modifications.

Example 3: The aim of this example is to show the conflict in user requirements. Assume that we have a new safety requirement described as follows: ‘when the multicopter is flying, the multicopter can be manually switched to returning to base by the RC transmitter. This switch requires that the INS, GPS, barometer, compass and propulsors be all healthy. Otherwise, the switch cannot occur for the multicopte’ Then, this safety requirement is transformed to an automaton named Specification 7’. Compared to Specification 7, ATE13/14/15 are all removed in Specification 7’ because they are not considered. By adding Specification 7’ to the whole control specification, Specification 7 and Specification 7’ will exist simultaneously. Then, a conflict will arise with the problematic trace depicted in Fig. 13 for Specification 7. Specification 7 indicates that this switch requires that the INS, GPS, barometer, compass, propulsors be all healthy, and the battery’s capacity be able to support the multicopter to return to base’. However, Specification 7’ does not restrict the condition of battery’s capacity. So, when battery's capacity is very low (ATE15 is true in Fig. 13), Specification 7 will not allow MCE7 to occur, but Specification 7’ will allow MCE7 to occur. Therefore, this conflict arises.

Remark 1: From the above examples, it can be seen that an incorrect failsafe mechanism might be obtained during the design process due to conflicting safety requirements or incorrect and inappropriate design of control specifications. The mistake might
be introduced inadvertently, and the designer cannot easily detect the problem by using empirical design methods. However, by relying on the SCT-based method, we can check the correctness of the obtained failsafe mechanism, and make modifications if the conflict is observed. This is a big advantage of the proposed method over empirical design methods. Once the conflict is removed, the resulting failsafe mechanism in the form of the supervisor is logically correct with respect to the plant and specifications, and able to deal with all relevant safety issues during flight.

6.2 Discussion

This paper aims to apply the SCT based method to guarantee the correctness in the design of the failsafe mechanism. It can be understood that, given a correct plant and correct specifications for a multicopter, synthesis approaches can automatically generate a correct protocol (or strategy) to control the multicopter. This process is also called ‘correct-by-design’ [44]. In this domain, various formal methods and techniques, such as SCT and formal synthesis based on linear temporal logic, are used to design control protocol of autonomous systems, including autonomous cars [45], aircraft [46–48] and swarm robots [49]. With a precise description of both the multicopter and its correct behaviour, the proposed method allows a failsafe mechanism that guarantees the correct behaviour of the system to be automatically designed.

Here, the generated supervisor by SCT satisfies the following properties:

(i) **Deterministic**: This property has two aspects. First, there exists no situation that one event triggers a transition from a single source state to different target states in the obtained supervisor. Second, after the occurrence of MIEs and ATEs, SCT can guarantee that only one MCE is enabled by disabling other MCEs due to deliberate design of control specifications. In this case, after the

![Fig. 10](image1.png)

**Fig. 10** Automaton model of Specification 7. Specification 7 is triggered under the two successive conditions: (i) the multicopter is in LOITER MODE (MCE4 occurs), (ii) and then the remote pilot normally manipulates the sticks of the RC transmitter (MIE5 occurs). In this case, when the remote pilot uses the flight mode switch to manually switch the multicopter to RTL MODE (MCE7 occurs), if the INS, GPS, barometer, compass, propulsors are all healthy (ATE1, ATE3, ATE5, ATE7 and ATE9 occur), the connection to the RC transmitter is normal (ATE11 occurs), the battery’s capacity is able to support the multicopter to return to the base (ATE13 or ATE14 occurs), and the multicopter’s distance from the base is not less than a given threshold (ATE19 occurs), then the multicopter enters RTL MODE (MCE4 occurs); otherwise, the multicopter stays in LOITER MODE (MCE4 occurs). Furthermore, when the remote pilot uses the flight mode switch to manually switch the multicopter to AL MODE (MIE8 occurs), the multicopter enters AL MODE (MCE5 occurs).

![Fig. 11](image2.png)

**Fig. 11** Problematic trace in synthesized supervisor due to Specification 1

![Fig. 12](image3.png)

**Fig. 12** Problematic trace in synthesized supervisor due to Specification 1

This is an open access article published by the IET and Zhejiang University Press under the Creative Commons Attribution License (http://creativecommons.org/licenses/by/3.0/)
occurrence of certain MIEs and ATEs, the mode which the multicopter should enter is deterministic.

(ii) Non-conflicting [27, Chapter 3.6]: SCT often can find the maximally permissive non-blocking supervisor satisfying all requirements. If some requirements are fundamentally contradictory, then a supervisor does not exist (this can be observed by using SCT software). If so, the designer should check in part (i) the correctness of control specifications transformed from user textual requirements; or (ii) the reasonableness of the user requirements.

(iii) Logically correct: SCT is a mature and effective tool to be used in the area of decision-making. If the plant and control specifications are correctly modelled, the logic of the generated supervisor will correctly satisfy user requirements without introducing man-made mistakes and bugs in the high level.

7 Implementation and simulation

Based on the obtained supervisor generated by TCT software or Supremica, an implementation method suitable for multicopters is presented, in which the supervisor is transformed into decision-making codes. Here, for any state, we only focus on enabling certain controllable events here rather than disabling controllable events. The two ways are equivalent to each other.

7.1 A simple semi-autonomous autopilot (SAA) system architecture

In order to introduce the implementation, we need to know where our supervisor is implemented in a SAA system architecture for multicopters. A simple SAA system architecture is shown in Fig. 14. The communication management module is used to detect the command sent by remote pilots, namely MIEs in Table 5. PHM module is used to detect the safety issues, namely communication breakdown, sensor failure and propulsion system anomaly, with the output being ATEs in Table 7. The mission management module is used to decide which mode the multicopter will switch to, based on MIEs, ATEs and the obtained supervisor. According to the selected mode (eight multicopter modes defined in Section 6.1), eight low-level flight controllers correspond to them, namely C1, C2, ..., C8. A multicopter with such an SAA is always controlled by one of them. The sensor management module is used to estimate the information of multicopters, such as motion information and actuator information. It serves for feedback control and PHM. PHM is in turn used to help sensor management module to separate some failed sensors so that their information will not invalidate fusion results.

7.2 Failsafe mechanism implementation

On the one hand, we would want to avoid manual implementation of the calculated supervisors, since this may introduce errors and is also difficult for complex cases. On the other hand, we look for an easy way to generate an application programming interface (API) function, with events MIEs, ATEs as the input and modes (MCEs) as output, so that it can be easily put in the mission management module shown in Fig. 14. The information required from a synthesized supervisor is a transition matrix, which is an $m \times 3$ matrix where $m$ is the number of transitions in the synthesized supervisor. (We have developed a function to export the transition matrix based on the output file of Supremica, available in http://rfly.buaa.edu.cn/resources.) As shown in Table 8, in each row, it consists of a source state, a destination state and a triggered event. For example, if the multicopter is in source state 1 and the triggered event is 1, then the destination state will be 2. By taking the synthesized supervisor of multicopters as an example, it contains 790 states, 37 events and 1566 transitions. So, the transition matrix is a $1566 \times 3$ matrix. In fact, we only need to consider eight modes (corresponding to eight marker states in the supervisor), namely POWER OFF MODE, STANDBY MODE, ...
As per the decision period, the system must be in one state, then they will be fed into the mission management module according to the occurrence order in Plant. So, the same event may be overwritten by the new one during the mode decision period. Therefore, after one decision period, the system must be in one of those eight states. By recalling Fig. 7, since the events in every transition are mutually exclusive, one and only one event must be triggered for any transition. As a result, the system does not stop at any non-marker state. The computation is affordable by current autopilots. For example, if an autopilot has a 20 MHz CPU, the maximum time for a mode decision will only take about 1.566 ms.

Actually the high-level decision-making is a relatively slow process in practice. Thus, the failsafe mechanism implementation is not synchronized with the low-level flight control system. In practice, for example, the events MIEs, ATEs will be detected every 0.01 s, while the mode decision period may be 1 s. All triggered events are collected together in every mode decision period. So, the same event may be overwritten by the new one during the mode decision period. We do not need to consider the past sequence of all detected events, because they will be fed into the mission management module according to the occurrence order in Plant shown in Fig. 7. For example, in Fig. 7, if the initial state is S14 and the events ATE3, ATE1, ATE5, ATE7, ATE9, ATE11, ATE13, ATE16, ATE18, ATE20, MIE5, MIE6 are collected one by one, then they will be fed into the mission management module according to the occurrence order in Plant, namely MIE5, MIE6, ATE1, ATE3 ATE5, ATE7, ATE9, ATE11, ATE13, ATE16, ATE18, and ATE20. Then the system will go to S26 in Plant. Consequently, only one MCEi will be enabled by the autopilot according to the specifications, i = 1, 2, ..., 8. Therefore, the system will finally have a definite mode chosen. For our case, the failsafe mechanism is implemented as shown in Table 9, where Δ > 0 represents a mode decision period.

### Table 9 Decision-making logic implementation

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>export a transition matrix from the supervisor synthesized by TCT software or Supremica; k = 0, Δ &gt; 0 is a positive integer representing a mode decision period; the initial state s = s0</td>
</tr>
<tr>
<td>2</td>
<td>k = k + 1</td>
</tr>
<tr>
<td>3</td>
<td>detect the instruction from the RC transmitter, health status of all considered equipment and flight status of the multicopter. If mod (k, Δ) = 0, goes to step 4; otherwise, go to step 2</td>
</tr>
<tr>
<td>4</td>
<td>collect events occurring in the mode decision period Δ</td>
</tr>
<tr>
<td>5</td>
<td>by starting at state s with the events input according to the occurrence order in Plant one by one, search the transition matrix when an event is an input. After all the MIEs and ATEs are fed completely, search the transition matrix again, and only one match will be found, where the triggered event is an MCE and the destination state is s1</td>
</tr>
<tr>
<td>6</td>
<td>s = s1, go to step 2</td>
</tr>
</tbody>
</table>

**Fig. 15 Simulation diagram**

GROUND-ERROR MODE, LOITER MODE, ALTITUDE-HOLD MODE, STABILIZE MODE, RTL MODE and AL MODE in Section 6.1. For each of them, a corresponding low-level flight controller, namely C1, C2, ..., or C8, is used to control the multicopter (see Fig. 14). Also, there exist many non-marker states in the transition matrix (790 – 8 = 782 non-marker states for the considered multicopter), to which no low-level flight controllers correspond. Therefore, after one decision period, the system must be in one of those eight states. By recalling Fig. 7, since the events in every transition are mutually exclusive, one and only one event must be triggered for any transition. As a result, the system does not stop at any non-marker state. The computation is affordable by current autopilots. For example, if an autopilot has a 20 MHz CPU, then the maximum time for a mode decision will only take about 1.566 ms.

Actually the high-level decision-making is a relatively slow process in practice. Thus, the failsafe mechanism implementation is not synchronized with the low-level flight control system. In practice, for example, the events MIEs, ATEs will be detected every 0.01 s, while the mode decision period may be 1 s. All triggered events are collected together in every mode decision period. So, the same event may be overwritten by the new one during the mode decision period. We do not need to consider the past sequence of all detected events, because they will be fed into the mission management module according to the occurrence order in Plant shown in Fig. 7. For example, in Fig. 7, if the initial state is S14 and the events ATE3, ATE1, ATE5, ATE7, ATE9, ATE11, ATE13, ATE16, ATE18, ATE20, MIE5, MIE6 are collected one by one, then they will be fed into the mission management module according to the occurrence order in Plant, namely MIE5, MIE6, ATE1, ATE3 ATE5, ATE7, ATE9, ATE11, ATE13, ATE16, ATE18, and ATE20. Then the system will go to S26 in Plant. Consequently, only one MCEi will be enabled by the autopilot according to the specifications, i = 1, 2, ..., 8. Therefore, the system will finally have a definite mode chosen. For our case, the failsafe mechanism is implemented as shown in Table 9, where Δ > 0 represents a mode decision period.

### 7.3 Simulation

In this part, we put the failsafe mechanism into a real-time flight simulation platform of quadcopters developed by MATLAB. Although it is realized by MATLAB, this method is applicable to any programming language. The simulation diagram is shown in Fig. 15. This simulation contains three main functions: (i) the failsafe mechanism can determine the flight mode according to the health check result, instruction of RC transmitter and quadcopter status; (ii) the remote pilot can fly the quadcopter through RC transmitter; (iii) the flight status of quadcopter can be visually displayed by FlightGear. Thus, this simulation can be viewed as an SAA simulation of quadcopters. A video of this simulation is presented in https://www.youtube.com/watch?v=b1-K2xWbwF8&feature=youtu.be and http://t.cn/RXmhnu6. It contains three scenarios: (i) the remote pilot manually controls the quadcopter to arm, fly, RTL, and land; (ii) anomalies of GPS, barometer, and INS occur during flight; (iii) the connection of RC transmitter is abnormal during flight.

### 8 Conclusions

This paper proposes an SCT-based method to design a failsafe mechanism of multicopters. The modelling process of the plant and control specifications is presented in detail. The failsafe mechanism is obtained by synthesising a supervisor in the monolithic framework. It ignores the detailed dynamic behaviour underlying each multicopter mode. This is reasonable because the failsafe mechanism belongs to the high-level decision-making module of a multicopter, while the dynamic behaviour can be characterized and controlled in the low-level flight control system. Also, we discuss the meaning of correctness and the properties of the obtained supervisor. The formal method based on SCT extends our confidence in the correctness of the designed failsafe mechanism. In future research, it is deserved to study how to normalise and simplify requirement analysis and related modelling so that the proposed formal method for multicopters can be easy to extend to the failsafe mechanism for other similar safety-critical systems.