Software and Systems Modeling
https://doi.org/10.1007/s10270-022-00981-7

THEME SECTION PAPER

# Online adaptation for autonomous unmanned systems driven by requirements satisfaction model

Yixing Luo[1,2] · Yuan Zhou[3] · Haiyan Zhao[1,2] · Zhi Jin[1,2] · Tianwei Zhang[3] · Yang Liu[3,4] · Danny Barthaud[5] · Yijun Yu[5]

Received: 16 March 2021 / Revised: 8 January 2022 / Accepted: 17 January 2022


**Abstract**
Autonomous unmanned systems (AUSs) emerge to replace human operators for achieving better safety, efficiency, and effectiveness in harsh and difficult missions. They usually run in a highly open and dynamic operating environment, in which some unexpected situations may occur, leading to violations of predefined requirements. In order to maintain stable performance, the AUS control software needs to predict in advance whether the requirements will be violated and then make adaptations to maximize requirements satisfaction. We propose `Captain`, a model-driven and control-based online adaptation approach, for the AUS control software. At the modeling phase, apart from the system behavior model and the operating environment model, we construct a requirements satisfaction model. At runtime, based on the requirements satisfaction model, `Captain` first predicts whether the requirements will be violated in the upcoming situation; then identifies the unsatisfiable requirements that need to be accommodated; and finally, finds an optimal adaptation for the upcoming situation. We evaluate `Captain` in both simulated scenarios and the real world. For the former, we use two cases of UAV Delivery and UUV Ocean Surveillance, whose results demonstrate the `Captain`'s robustness, scalability, and real-time performance. For the latter, we have successfully implemented `Captain` in the DJI Matrice 100 UAV with real-world workloads.

**Keywords** Requirements satisfaction model · Runtime adaptation · Models@runtime · Autonomous unmanned systems

## 1 Introduction

The dramatic growth of autonomous unmanned systems (AUSs), including autonomous vehicles, unmanned aerial vehicles (UAVs), unmanned underwater vehicles (UUVs), is fundamentally changing the industries of transportation, manufacturing, and logistics [1]. Different from traditional software systems, AUSs are typically composed of physical and software components and interact with the physical environment through sensor inputs and motor outputs [2,3]. From the perspective of requirements engineering, it is far from enough for AUSs to merely assure the function correctness [4]. Considering the close interaction between AUSs and the environment, many environment-related non-functional requirements (NFRs) are extremely critical [5]. For example, any violation of safety requirements may be dangerous and lead to catastrophic results [6].

Requirements, especially those environment-related non-functional requirements [7], may not always be satisfiable at runtime as AUSs face uncertainties in the highly open and dynamic environment [8]. Such runtime uncertainties mainly arise from incomplete knowledge about the operating environment (e.g., unforeseen obstacles and private regions that can be only detected online [9]), and/or fluctuations in system characteristics [2] (e.g., degraded hardware like sensors that become less accurate over time or that consume more

Communicated by Tao Yue, Paolo Arcaini, Ji Wu, and Xiaowei Huang.

✉ Zhi Jin
zhijin@pku.edu.cn

Yixing Luo
yixingluo@pku.edu.cn

1 Key Laboratory of High-Confidence Software Technologies, Peking University, Ministry of Education, Beijing, China

2 School of Computer Science, Peking University, Beijing, China

3 School of Computer Science and Engineering, Nanyang Technological University, Singapore, Singapore

4 School of Information Science and Technology, Zhejiang Sci-Tech University, Hangzhou, China

5 School of Computing and Communications, The Open University, Milton Keynes, UK

Published online: 22 February 2022

energy than expected [3]). The effects produced by these unpredictable conditions can compound and thereby inhibit the system from fully satisfying its requirements, i.e., the occurrence of requirements violations [10].

As one of the key components in AUSs, the control software is responsible for controlling AUSs' behaviors to meet multiple requirements [3]. It faces two further challenges for meeting the requirements, when AUSs operate under unpredictable conditions. First, how to address the runtime uncertainties, as the environment and system characteristics are ever-changing and can only be partially known at development time. Second, how to detect and mitigate potential requirements violations effectively and efficiently, as this technique should be embedded into AUS online planning.

After a comprehensive analysis of related work, we found that the control-based adaptation [11,12] in which self-adaptation mechanisms are designed based on control theory, could be a promising way for tackling the first challenge as it can provide formal guarantees on the desired behavior of the controlled systems. Early research on the control-based adaptation focuses on ad-hoc solutions to control the lower-level elements and resources of the system (e.g., CPU, storage, bandwidth, etc.) [13]. These solutions require a well understanding of mathematical system models and are often done on a per-problem basis, discouraging flexibility and generality. Later, more general methods for robust control are proposed like AMOCS [14], SimCA [15], SimCA* [16] and AMOCS-MA [17], etc. However, when handling multiple requirements, these approaches either neglect the requirements conflicts [15,16] or employ a utility function with predefined priorities to make trade-offs between the satisfaction of multiple requirements [14,17]. The utility function is susceptible to biases as the priorities defined by human experts can be subjective [18].

Secondly, the goal-driven adaptation is proposed to define how requirements can be specified and accommodated in case of violations [19]. This could be a possible way to tackle the second challenge. RELAX [20] is incorporated into goal models [21–23] for self-adaptive systems, allowing requirements to be partially satisfied in response to uncertainties. To address varying environmental conditions, approaches for adapted goal models have also been proposed [24,25]. At present, the goal-driven adaptation mostly only focuses on fixed and predefined adaptation rules for requirements, rarely considers the possibility of online requirements adaptation. However, for highly configurable AUSs, it is very important to detect and mitigate requirements violations proactively and instantly, such that timely adaptations can be made, especially when the available computational capacity for AUSs is limited.

In this paper, we propose `Captain`, a model-driven and control-based online adaptation approach for addressing both the challenges. The main idea is to enable AUS control software to identify and mitigate requirements violations proactively in case of runtime uncertainties based on control theory. For the purpose of runtime identification and mitigation, apart from the system behavior model and the interactive environment model, we also explicitly establish the requirements satisfaction model, in which requirements are classified into three categories for a systematic and quantitative satisfaction evaluation: (1)*Mission*: functional requirements that AUSs need to complete; (2)*Hard constraints*: NFRs that must be satisfied by AUSs during the completion of the *Mission*; (3)*Soft Constraints*: NFRs that are not sharply defined a priori and should be satisfied as much as possible by AUSs.

At runtime, we introduce a three-step control-based mechanism to check, analyze and optimize requirements satisfaction proactively and instantly based on the requirements satisfaction model. First, we utilize model predictive control (MPC) strategy to check whether all requirements can be satisfied in the upcoming situation according to the predicted evolution of the system and environment. Second, a set of *soft constraints* that are checked as unsatisfiable is identified for accommodation. Third, an adaptive solution with the optimal requirements satisfaction is figured out through multi-objective optimization in which unsatisfiable *soft constraints* are transformed into the objective function, while the satisfaction of other requirements is assured. In this process, only soft constraints that are predicted to be violated are transformed to optimize, the search space for adaptation is accordingly scaled-down, making it efficient to find the adaptive solution with the optimal satisfaction of requirements.

The main contributions are summarized as follows.

- A requirement satisfaction model to evaluate degrees of requirements satisfaction at runtime in a quantitative and systematic way.
- A three-step control-based adaptation mechanism to identify and mitigate requirements violations proactively, i.e. requirements satisfaction checking, requirements satisfaction analyzing and requirements satisfaction optimizing.
- A multi-objective planner for the optimal requirements satisfaction of AUSs through adaptations in sensor reconfiguration and motion planning simultaneously at runtime.
- Comprehensive evaluations of `Captain` on both simulators and real-world scenarios, using diverse AUS case studies, environment, and workloads.

The remainder of the paper is organized as follows. We describe the motivation behind the proposed approach in Sect. 2. We describe the models established in Sect. 3, followed by a detailed explanation of the online adaption process of the approach in Sect. 4. Sections 5 and 6 present

**Fig. 1** A UAV of amazon prime air is delivering [28]

the assessments and experimental results of the approach. An overview of related work is summarized in Sect. 7, followed by conclusions in Sect. 8.

## 2 Motivating scenario

Recently, there have been significant advances in a variety of technologies designed to deliver cargoes by UAVs, primarily driven by industrial efforts [26]. As a motivating scenario, we describe the UAV Delivery service provided by Amazon, i.e., "Amazon Prime Air" [27]. This service uses delivery UAVs to autonomously fly individual packages to customers within 30 min of ordering, as shown in Fig. 1. This scenario will be used as a running example to illustrate the modeling (see Sect. 3), the online adaptation (see Sect. 4.4), and evaluation (see Sect. 6) of our approach.

The UAV's mission is to deliver cargo from the warehouse to customers, and this mission involves concerns from two aspects. First, requirements for the delivery mission involve various stakeholders, e.g., delivery suppliers, customers, and residents along the delivery path, and requirements from different stakeholders should be taken into account. For example, for the supplier, the completion of the delivery and energy-saving are directly related to the economic viability of delivery UAVs [26]. Customers would like to increase UAV's speed such that the parcel can be received in 30 minutes or less. For the residents along the delivery path, safety and privacy in residential areas should be preserved during the flight [29,30].

Second, during the flight, UAV is destined to face unpredictable conditions that primarily come from two sources: (1) environment changes, such as unforeseen obstacles [31] and private regions [32] that are detected online by onboard sensors; (2) fluctuations in system characteristics, such as sensor degradation or dramatic battery life reduction [33]. These unpredictable conditions may result in requirements violations during the UAV operation and need to be mitigated. For example, when new private regions are detected, the UAV may fail to compute a flight path to complete the mission within the time budget when avoiding trespassing on the private regions [9,32]. In case that the timeliness requirements would be violated, strategies should be found to minimize such violations. However, when a safe landing condition cannot be assured (e.g., on the river), adaptations (e.g., path replanning) are needed to mitigate safety violation [33].

Accordingly, the UAV Delivery mission is not trivial and requires the UAV to be adaptive to unpredictable conditions while satisfying multiple requirements. Specifically, it is critical for the UAV to monitor the changes in the operating environment and system characteristics at runtime, proactively detect requirements violations, and mitigate such violations through adjusting UAV's configurations and/or behaviors for optimal requirements satisfaction.

## 3 Modeling

We next establish models for system behavior, operating environment, and requirements of AUSs, and these models are used in the online adaptation process of `Captain`.

### 3.1 System behavior modeling

According to existing work [2,3], an AUS is a highly configurable cyber-physical system whose states include not only kinematic state but also configuration options of system characteristics (e.g., usage of sensors).

Following the guidelines [34,35] for robot system modeling, we model the kinematic state of the AUS as a sphere with safe radius $r_a$, and the center position is $\boldsymbol{x}$.

The time is discretized into time instants with an equal step length $\tau$. At any time instant $k$, the kinematic state is represented as $\boldsymbol{m}_k = (\boldsymbol{x}_k, \boldsymbol{v}_k)$, where $\boldsymbol{x}_k$ and $\boldsymbol{v}_k$ denote the vectors of its position and velocity at time instant $k$, respectively. $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \boldsymbol{v}_k\tau$, $k = 0, 1, 2, \ldots$.

As to configuration options of system characteristics, we mainly focus on the configuration of equipped sensors. We measure and quantify each sensor's functionality as a variable $w_k \in [0, 1]$, at time instant $k$. If $w_k = 1$, the sensor is unrestricted and able to function at its full working capacity. For example, in DJI Matrice 100 UAV, the Zenmuse Z3 camera's parameters contain resolution and orientation angle. The camera's orientation angle ranges from $-30°$ (up to the sky) to $90°$ (direct to the ground). Then, we linearly quantify the orientation angle at time instant $k$ as $w_k \in [1/3, 1]$ ($w_k = 1/3$ for $-30°$ and $w_k = 1$ for $90°$). The AUS's sensor configuration is defined as a vector $\boldsymbol{w}_k$ that combines the $w_k$ of all sensors' parameters. In case that the sensor parameters are discrete variables, $w_k$ can be computed through fuzzification of the reals.

Therefore, at time instant $k$, the AUS's state can be formulated as a vector $\boldsymbol{s}_k = (\boldsymbol{m}_k, \boldsymbol{w}_k)$. To adjust the behaviors

of the AUS at $k$ is to compute the control signals of velocity and sensor configuration $\boldsymbol{u}_k = (\boldsymbol{v}_k, \boldsymbol{w}_k)$.

### 3.2 Operating environment modeling

Operating environment model will reveal the runtime scenarios which may directly restrict the system behaviors [7]. Suppose that the operating environment of the AUS may contain obstacles and private regions (e.g., the residential areas) which can be detected by the AUS's onboard sensors. The AUS needs to avoid hitting all obstacles compulsorily and at the same time as far as possible not to invade private areas.

The operating environment of the AUS can be described as $M$ with the scale of $X \times Y \times Z$ in three dimensions. Let $\mathcal{O}$ and $\mathcal{C}$ be the set of obstacles and private regions, respectively, in $M$. These two sets contain information of obstacles and private regions. To be consistent with our previous work [32], we model the obstacles and private regions as spheres that are described by their center positions and radius.

During mission completion, the operating environment of AUS may change, so the AUS needs to monitor the environment and update the environmental information online. Let $\mathcal{O}_k$ be the set of obstacles and $\mathcal{C}_k$ be the set of private regions detected by the AUS at time instant $k$. Each obstacle $o \in \mathcal{O}_k$ is modeled as a sphere with a radius $r_o$ and a center $\boldsymbol{x}_o$, while each private region $c \in \mathcal{C}_k$ is modeled as a sphere with a radius $r_c$ and a center $\boldsymbol{x}_c$.

For safety, the AUS needs to keep at least an additional safe distance $D_o$ with the obstacles, i.e., $\|\boldsymbol{x}_k - \boldsymbol{x}_o\|_2 \geq r_a + r_o + D_o, \forall o \in \mathcal{O}_k$. For privacy-preservation, supposing the affecting radius of the private region $c$ is $D_c$, the distance between the AUS and the private region should be at least at $r_a + r_c + D_c$, i.e., $\|\boldsymbol{x}_k - \boldsymbol{x}_c\|_2 \geq r_a + r_c + D_c, \forall c \in \mathcal{C}_k$. $D_c$ is larger than $D_o$ as usual.

### 3.3 Requirements modeling

This subsection presents the categories of the requirements that matter in our work on how to evaluate the requirements satisfaction.

#### 3.3.1 Requirements classification

We classify the requirements that AUS control software needs to satisfy into three categories:

(1) *Missions* stand for functional requirements that AUSs need to complete, e.g., delivery, search and rescue. For example, the delivery mission can be viewed as completed when the UAV reaches the destination $\boldsymbol{x}_d$ from the initial position $\boldsymbol{x}_0$. The set of missions to complete is defined as $\mathcal{M}$.

(2) *Hard constraints* stand for non-functional requirements that must be kept during the completion of the missions as requested by ISO 10128 [36]. These requirements are mainly elicited from critical attributes of AUS, physical limitations, standards, and regulations. Violations of hard constraints (e.g., colliding with other vehicles) can lead to loss of life, property damage, or environmental harm. The set of hard constraints is defined as $\mathcal{H}$.

(3) *Soft constraints* stand for non-functional requirements that should be satisfied as much as possible. Soft constraints often determine the performance of the AUS according to user preferences, e.g., mission completion time and privacy-preservation (i.e., keep an appropriate distance with private regions or reduce sensory capability). The set of soft constraints is defined as $\mathcal{S}$.

The behaviors of an AUS are subject to various requirements composed of hard constraints and soft constraints during mission completion. The complete set of requirements for an AUS is $\mathcal{R} = \mathcal{M} \cup \mathcal{H} \cup \mathcal{S}$. In the following, we use the UAV Delivery scenario to materialize these non-functional requirements.

The temporal dimension allows us to divide the requirements set $\mathcal{R}$ into two subsets $\mathcal{R}_\mathcal{M}$ and $\mathcal{R}_\mathcal{A}$, containing the requirements that should be maintained at all time instants and the requirements which should be achieved in the future, respectively [37]. For example, requirements like safety that should be kept at any time instant belong to $\mathcal{R}_\mathcal{M}$; while requirements like timeliness that are expected to be achieved at the end of the mission completion belong to $\mathcal{R}_\mathcal{A}$.

**UAV scenario** From the Technology Roadmap for Amazon Prime Air [38], we elicit the following five representative non-functional requirements that need to be satisfied for a delivery UAV during the mission completion. We formulate these requirements as shown in Table 1, in which we use logical connectives, temporal operators $\Box$ (always) and $\Diamond$ (eventually) to formulate the soft constraints and hard constraints.

- *Safety* ($R_S$): Safety means that the UAV needs to keep a safe distance, ensuring no damage to people, property, and the environment. When it is seen as a hard constraint, it requires to satisfy the following condition at any time instant $k$, $\|\boldsymbol{x}_k - \boldsymbol{x}_o\|_2 \geq r_a + r_o, \forall o \in \mathcal{O}_k$. As a soft constraint, it means that the UAV should keep an additional distance $D_o$ from the obstacle $o$ in case of motion disturbance.
- *Privacy* ($R_P$): Privacy preservation means that UAV cannot intrude into private regions and expose private information. When it is seen as a hard constraint, it requires the UAV not collide with private regions. As a soft constraint, it means that the UAV should keep a dis-

**Table 1** Requirements Specifications for UAV Delivery Systems

| Requirements | Hard constraints ($\mathcal{H}$) | Soft constraints ($\mathcal{S}$) |
|---|---|---|
| Safety ($R_S$) | $\square\ \|\boldsymbol{x}_k - \boldsymbol{x}_o\|_2 \geq r_a + r_o, \forall k \in [0, T], o \in \mathcal{O}_k$ | $\square\ \|\boldsymbol{x}_k - \boldsymbol{x}_o\|_2 \geq r_a + r_o + D_o, \forall k \in [0, T], o \in \mathcal{O}_k$ |
| Privacy ($R_P$) | $\square\ \|\boldsymbol{x}_k - \boldsymbol{x}_c\|_2 \geq r_a + r_c, \forall k \in [0, T], c \in \mathcal{C}_k$ | $\square\ \|\boldsymbol{x}_k - \boldsymbol{x}_c\|_2 \geq r_a + r_c + D_c, \forall k \in [0, T], c \in \mathcal{C}_k$ |
| Accuracy ($R_\varphi$) | $\Diamond\ \frac{1}{T}\sum_{k=0}^{T-1} \|\boldsymbol{w}_k\|\,\tau \geq A$ | $\Diamond\ \frac{1}{T}\sum_{k=0}^{T-1} \|\boldsymbol{w}_k\|\,\tau \geq A_t \geq A$ |
| Timeliness ($R_\xi$) | $\Diamond\ \sum_{k=0}^{T-1} \boldsymbol{v}_k\tau \leq \Delta$ | $\Diamond\ \sum_{k=0}^{T-1} \boldsymbol{v}_k\tau \leq \Delta_t \leq \Delta$ |
| Energy-saving ($R_E$) | $\Diamond\ \sum_{k=0}^{T-1} E_k \leq E$, $E_k = \|\boldsymbol{x}_{k+1} - \boldsymbol{x}_k\|_2 + \eta_1 \|\boldsymbol{v}_{k+1} - \boldsymbol{v}_k\|_2 + \eta_2 \|\boldsymbol{w}_k\|\,\tau$ | $\Diamond\ \sum_{k=0}^{T-1} E_k \leq E_t \leq E$ |

[a] Soft constraints and hard constraints are not necessarily associated with each other and can be separately specified. However, if a hard constraint shares the same QM with its corresponding soft constraint, the large violation degree of soft constraint can serve as a reminder of the hard constraint's violation in the future

tance of $D_c$ with the private region, i.e., $\|\boldsymbol{x}_k - \boldsymbol{x}_c\|_2 \geq r_a + D_c, \forall c \in \mathcal{C}_k$.

- *Accuracy* ($R_\varphi$): This is a basic requirement for the accuracy of the collected information, e.g., the quality of the images. The target accuracy of information is $A_t$, while $A$ is the threshold. Then, we have $A \leq A_t$. The quality of the images can be indicated by the capability configuration of the sensors $\boldsymbol{w}_k$, e.g., the camera's resolution and orientation angle.
- *Timeliness* ($R_\xi$): This requirement is related to the motion time. Let $\Delta$ be the deadline to finish the mission and $\Delta_t$ is the expected time that UAV takes for completing the mission. Then, we have $\Delta_t \leq \Delta$.
- *Energy-saving* ($R_E$): Energy consumption should be less than the total capacity of the energy module $E$, while the amount of energy expected to be consumed is $E_t$. The energy consumption is affected by the traveled distance, control effort, and sensor usage. In this paper, at the time duration $[k\tau, (k+1)\tau)$, the energy cost is computed as $E_k = \|\boldsymbol{x}_{k+1} - \boldsymbol{x}_k\|_2 + \eta_1 \|\boldsymbol{v}_{k+1} - \boldsymbol{v}_k\|_2 + \eta_2 \|\boldsymbol{w}_k\|\,\tau$ with the assumption that each unit of traveled distance costs one unit of energy, deceleration or acceleration costs $\eta_1$ units of energy, and sensor usage consumes $\eta_2$ units of energy.

### 3.3.2 Requirement satisfaction modeling

The absolute satisfaction of requirements is challenging for AUSs operating in complex and changing environments [39] due to runtime uncertainties [10]. In previous work, fuzzy set theory is applied to describe the requirement uncertainties [18,23]. In this paper, we use fuzzy membership functions to describe the satisfaction degree of requirements in a quantifiable way.

Given a set of requirements $\mathcal{R} = \{R_1, \ldots, R_n\}$ for an AUS, we establish a corresponding requirements satisfaction model, which supports measuring and evaluating requirements satisfaction online. We introduce $\mathcal{X}_i$ as the Quantification Measure (QM) that can be observed at runtime, to evaluate requirement $R_i$. Its value at time instant $k$ is denoted as $\mathcal{X}_i(k)$. $\mathcal{X}_i$ is defined as a function of observations $Obs$ including previous system states, current detected environmental conditions, i.e., $\mathcal{X}_i(k) = h_i(Obs_k)$, $Obs_k = \big(\langle \boldsymbol{s}_0, \boldsymbol{s}_1, ..., \boldsymbol{s}_k\rangle, \mathcal{O}_k, \mathcal{C}_k\big)$. The calculation of $\mathcal{X}_i$ for $R_i$ is manually defined. As shown in Table 1, the QM for the timeliness requirement is $\mathcal{X}_\xi(T) = \sum_{k=0}^{T-1} \boldsymbol{v}_k\tau$, where $T$ is the time instant when the mission is completed.

The hard constraints and the mission satisfaction criteria are usually clear-cut. Since each hard constraint and mission must be satisfied, we evaluate their satisfaction with the Boolean function, given in Eq. (1).

$$\mathrm{DS}^0(\mathcal{X}_i) = \begin{cases} 1, & \text{if } \mathcal{X}_i \text{ satisfy } R_i \in \mathcal{M} \cup \mathcal{H} \\ 0, & \text{otherwise} \end{cases} \tag{1}$$

To depict the satisfaction degree of soft constraints, we design a gradual measurement based on the fuzzy set theory and set up three requirements satisfaction functions according to three types of relationship between the QM and users' expectations (i.e., LESS THAN, MORE THAN, AS CLOSE AS) as follows.

$$\mathrm{DS}^1(\mathcal{X}_i) = \begin{cases} \frac{ub_i - \mathcal{X}_i}{ub_i - g_i} & g_i < \mathcal{X}_i \leq ub_i \\ 1 & \mathcal{X}_i \leq g_i \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

$$\mathrm{DS}^2(\mathcal{X}_i) = \begin{cases} \frac{\mathcal{X}_i - lb_i}{g_i - lb_i} & lb_i \leq \mathcal{X}_i < g_i \\ 1 & \mathcal{X}_i \geq g_i \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

$$\mathrm{DS}^3(\mathcal{X}_i) = \begin{cases} \frac{\mathcal{X}_i - lb_i}{g_{i^-} - lb_i} & lb_i \leq \mathcal{X}_i < g_{i^-} \\ 1 & g_{i^-} \leq \mathcal{X}_i \leq g_{i^+} \\ \frac{ub_i - \mathcal{X}_i}{ub_i - g_{i^+}} & g_{i^+} < \mathcal{X}_i \leq ub_i \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

In Eqs. (2)–(4), $lb_i$ and $ub_i$ are the lower and upper boundaries of $\mathcal{X}_i$, respectively; $g_i$, $g_{i^+}$, and $g_{i^-}$ are user-defined

target values. $\mathrm{DS}^1(\mathcal{X}_i)$ is the function to evaluate requirements satisfaction when $\mathcal{X}_i$ should LESS THAN the target value. A soft constraint $R_i \in \mathcal{S}$ is totally satisfied if $\mathcal{X}_i \leq g_i$, and the satisfaction linearly decreases to 0 if $\mathcal{X}_i > g_i$. Similarly, $\mathrm{DS}^2(\mathcal{X}_i)$ is the function for the MORE THAN relationship ($g_i \leq \mathcal{X}_i$), and $\mathrm{DS}^3(\mathcal{X}_i)$ is the function for the AS CLOSE AS POSSIBLE relationship ($g_{i^-} \leq \mathcal{X}_i \leq g_{i^+}$). Thus, the set of soft constraints $\mathcal{S}$ can be accordingly divided into three orthogonal subsets denoted as $\mathcal{S}^1$, $\mathcal{S}^2$ and $\mathcal{S}^3$, whose satisfaction is evaluated by the above three types of requirements satisfaction function, respectively. Clearly, we have $\mathcal{S} = \mathcal{S}^1 \cup \mathcal{S}^2 \cup \mathcal{S}^3$ and $\mathcal{S}^i \cap \mathcal{S}^j = \emptyset$ for $1 \leq i < j \leq 3$, $i$ and $j$ are integers.

Therefore, given a set of requirements $\mathcal{R}$, we have the four sets of functions to measure requirements satisfaction: $\mathcal{DS}^0 = \{\mathrm{DS}^0(\mathcal{X}_i) : R_i \in \mathcal{H} \cup \mathcal{M}\}$, $\mathcal{DS}^1 = \{\mathrm{DS}^1(\mathcal{X}_i) : R_i \in \mathcal{S}^1\}$, $\mathcal{DS}^2 = \{\mathrm{DS}^2(\mathcal{X}_i) : R_i \in \mathcal{S}^2\}$ and $\mathcal{DS}^3 = \{\mathrm{DS}^3(\mathcal{X}_i) : R_i \in \mathcal{S}^3\}$. The corresponding requirements satisfaction model is defined as the complete set of requirements satisfaction function, i.e., $\mathcal{DS} = \{\mathcal{DS}^0, \mathcal{DS}^1, \mathcal{DS}^2, \mathcal{DS}^3\}$.

**UAV scenario** We take the soft constraint of the privacy requirement listed in Table 1as an example to illustrate the design of the requirements satisfaction function. Based on the private region model presented in [32],privacy disclosure is affected by the distance between UAV and private regions, as well as the working state of sensors. The less the distance between UAV and the private region or the higher the sensor configuration is, the higher the possibility of private information exposure. We define the QM of privacy at time instant $k$as $\mathcal{X}_{P_c}(k) = 1 - \|\boldsymbol{w}_k\| \frac{(r_a + r_c + D_c) - \|\boldsymbol{x}_k - \boldsymbol{x}_c\|_2}{D_c}$, $\forall c \in \mathcal{C}_k$.We have the requirements satisfaction function using the formulation of $\mathrm{DS}^2$:

$$\mathrm{DS}^2(\mathcal{X}_{P_c}(k)) = \begin{cases} 1, & \mathcal{X}_{P_c}(k) \geq 1, \\ 0, & \|\boldsymbol{x}_k - \boldsymbol{x}_c\|_2 < r_a + r_c, \\ \mathcal{X}_{P_c}(k), & \text{otherwise} \end{cases}$$

Similarly, we can formulate QMs for other requirements, e.g., safety $\mathcal{X}_S$, timeliness $\mathcal{X}_\xi$, energy-saving $\mathcal{X}_E$, and accuracy $\mathcal{X}_\varphi$. Based on the soft constraints and hard constraints listed in Table 1, the sets of requirements satisfaction functions are:
$\mathcal{DS}^0$=$\{\mathrm{DS}^0(\mathcal{X}_S), \mathrm{DS}^0(\mathcal{X}_P), \mathrm{DS}^0(\mathcal{X}_\varphi), \mathrm{DS}^0(\mathcal{X}_\xi), \mathrm{DS}^0(\mathcal{X}_E)\}$,
$\mathcal{DS}^1 = \{\mathrm{DS}^1(\mathcal{X}_\xi), \mathrm{DS}^1(\mathcal{X}_E)\}$,
$\mathcal{DS}^2 = \{\mathrm{DS}^2(\mathcal{X}_S), \mathrm{DS}^2(\mathcal{X}_P), \mathrm{DS}^2(\mathcal{X}_\varphi)\}$.

The corresponding requirements satisfaction model is $\mathcal{DS} = \{\mathcal{DS}^0, \mathcal{DS}^1, \mathcal{DS}^2\}$.

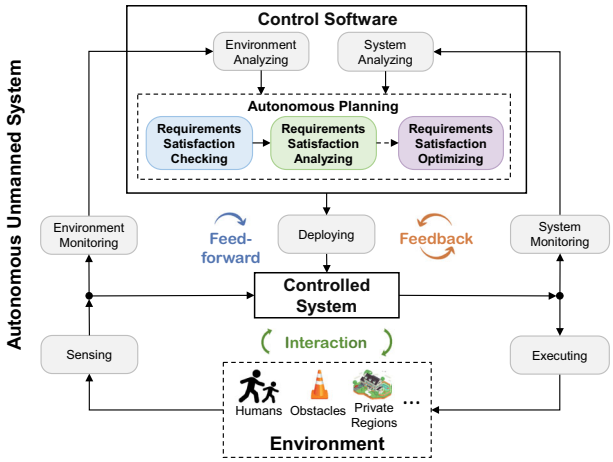



**Fig. 2** An overview of model-driven adaptation

## 4 Model-driven online adaptation

This section will describe the control-based online adaptation process of `Captain`, which is driven by the requirements satisfaction modeling established in Sect. 3.

### 4.1 Overview

Figure 2 shows the workflow of the model-driven and control-based online adaptation process of `Captain`, which is embedded into a two-layer architecture for the AUS [5]. The top layer structures the internal *Feed-forward* and *Feedback* control loops for optimal requirements satisfaction in response to runtime uncertainties, while the bottom layer shows the external *Interaction* loop between the AUS and operating environment.

At runtime, the AUS and environment are closely connected with the environment entities (humans, obstacles, private regions, etc.) through continuous sensing and execution. The *Interaction* loop is designed for the AUS to perceive changes in its operating environment (e.g., unexpected humans, obstacles, and private regions) and take corresponding actions.

The internal control loop consists of *Feed-forward* and *Feedback* loops, each of which is refined as a MAPE loop to handle runtime uncertainties through monitoring, analyzing, planning, and executing components. The *Feed-forward* and *Feedback* control loops are responsible for generating appropriate adaptation plans in response to changes in environmental conditions and system characteristics, respectively. Considering the interference between different control

loops, we set an equal step length $\tau$ to synchronize these two control loops. In addition, the analysis results of both environmental changes and system characteristics fluctuations are integrated for *Autonomous Planning* to generate the adaptation plan, such that sensor configuration and motion plan for the AUS can be generated simultaneously.

In the *Feed-forward* control loop, raw environmental data collected via *Sensing* are handled by *Environment Monitoring*. *Environment Analyzing* is for environmental evolution prediction, e.g., estimating the future positions of intruding vehicles. Based on environmental analysis, the requirements satisfaction optimization problem is treated as a three-step control problem in *Autonomous Planning*: *Requirements Satisfaction Checking* (see Sect. 4.2), *Requirements Satisfaction Analyzing* (see Sect. 4.3) and *Requirements Satisfaction Optimizing* (see Sect. 4.4). Finally, the adaptation plan, including motion plan and sensor configuration for the AUS are translated into control signals in *Deploying* and delivered to the *Controlled System* for *Executing*.

Similarly, in the *Feedback* control loop, system characteristics (e.g., usage of sensor) are monitored via *System Monitoring*. *System Analyzing* is to detect disturbance of system characteristics like component failures and sensor degradation. Once these uncertainties are detected, the three-step *Autonomous Planning* is triggered, such that an adaptation plan is expected to be generated to help the system recover from failures, even at a reduced level of requirements satisfaction.

## 4.2 Requirements satisfaction checking

Due to changes in the environment and system, some requirements may not always be satisfied. This step is to check whether all the requirements can still be satisfied proactively under the observations of these changes.

We formalize the requirements satisfaction checking problem as: Given a set of requirements $\mathcal{R} = \{R_1, \ldots, R_n\}$ and requirements satisfaction model $\mathcal{DS}$, determine whether there is a feasible system behavioral plan that enables all requirements to be totally satisfied, i.e., all QMs can achieve their target values.

To solve this problem, we construct an equivalence problem. Inspired by Lagrangian relaxation [40], we relax the demands of initial requirements with slack variables and bring them into the objective function. In this way, any violation of requirements will be penalized in the objective function.

Concretely, for requirement $R_i$ at time instant $k$, a nonnegative slack variable $\delta_{i,k}$ (resp., $\zeta_{i,k}$) is introduced to measure the distance between $\mathcal{X}_i(k)$ and $g_i$ in case $\mathcal{X}_i(k)$ is larger (resp., smaller) than $g_i$. With the slack variable, the demand of initial requirements satisfaction can be transformed to a slacked one $\psi_{i,k}$. For example, for $R_i \in \mathcal{S}^1$, the initial requirement demand is $\mathcal{X}_i(k) \leq g_i$. The slacked one becomes $\psi_{i,k} : \mathcal{X}_i(k) \in [lb_i, g_i + \delta_{i,k}] \wedge \delta_{i,k} \in [0, ub_i - g_i]$. The slack variables for missions and hard constraints should always be zero, i.e., $\psi_{i,k} : \mathrm{DS}^0(\mathcal{X}_i(k)) = 1$. Therefore, the requirements satisfaction checking problem can be transformed into an optimization problem, where the objective function is to minimize violations of all the requirements. The formulations of slacked requirements are listed as follows:

$$\psi_{i,k} : \begin{cases} \mathcal{X}_i(k) \in [lb_i, g_i + \delta_{i,k}] \wedge \delta_{i,k} \in [0, ub_i - g_i] \\ \quad \forall R_i \in \mathcal{S}^1 \\ \mathcal{X}_i(k) \in [g_i - \zeta_{i,k}, ub_i] \wedge \zeta_{i,k} \in [0, g_i - lb_i] \\ \quad \forall R_i \in \mathcal{S}^2 \\ \mathcal{X}_i(k) \in [g_{i^-} - \zeta_{i,k}, g_{i^+} + \delta_{i,k}] \wedge \\ \zeta_{i,k} \in [0, g_{i^-} - lb_i] \wedge \delta_{i,k} \in [0, ub_i - g_{i^+}] \\ \quad \forall R_i \in \mathcal{S}^3 \\ \mathrm{DS}^0(\mathcal{X}_i(k)) = 1, \ \forall R_i \in \mathcal{H} \cup \mathcal{M} \end{cases} \tag{5}$$

To identify unsatisfiable requirements proactively, we use the model predictive control (MPC) technique in a time-sliding mode. At time instant $k_0$, the `Captain` reasons the evolution of the environment and system in the next $N$ steps (i.e., the prediction horizon) based on the operating environment model, the system behavior model, and $Obs_k$. For $R_i \in \mathcal{R}_\mathcal{M}$, slacked requirements are constructed via estimating QMs based on the predicted evolution of the environment and system during the period $[k_0+1, k_0+N]$. For $R_i \in \mathcal{R}_\mathcal{A}$, slacked requirements are constructed based on the estimation of $\mathcal{X}_i(T)$, assuming that all missions end at $k = T$. Therefore, the requirements satisfaction problem (i.e., whether it is possible to meet all the requirements given the prediction horizon $N$) is transformed into a multi-objective optimization problem (i.e., the minimal requirements violations) described as follows:

$$\begin{aligned} \min_{\boldsymbol{u},\boldsymbol{\delta},\boldsymbol{\zeta}} f_k = & \sum_{R_i \in \mathcal{R}_\mathcal{M}} \sum_{k=k_0+1}^{k_0+N} (\overline{\delta_{i,k}} + \overline{\zeta_{i,k}}) \\ & + \sum_{R_i \in \mathcal{R}_\mathcal{A}} (\overline{\delta_{i,T}} + \overline{\zeta_{i,T}}) \\ s.t. \ & \forall R_i \in \mathcal{R}_\mathcal{M}, \forall k \in [k_0+1, ..., k_0+N], \ \psi_{i,k}; \\ & \forall R_i \in \mathcal{R}_\mathcal{A}, \ \psi_{i,T}. \end{aligned} \tag{6}$$

As shown in Eq. (6), the multi-objective optimization problem is scalarized into a single-objective problem with a combined sum of normalized slack variables ($\overline{\delta_{i,k}} = \frac{\delta_{i,k}}{ub_i - g_i}$, $\overline{\zeta_{i,k}} = \frac{\zeta_{i,k}}{g_i - lb_i}$) for no preference between requirements. Requirements are transformed into the slacked ones through Eq. (5). The planning result is a behavioral plan $\langle \boldsymbol{u}_{k_0}, ..., \boldsymbol{u}_{k_0+N-1} \rangle$ with the minimal requirements violations. Through solving Eq. (6), `Captain` can check require-

ments satisfaction during AUS operation. If the value of the objective function for the optimal solution exceed zero, i.e., $f_k(\boldsymbol{\delta}^*, \boldsymbol{\zeta}^*) > 0$, theoretically there exist requirements violations, i.e., requirements whose slack variables are positive.

**UAV scenario** We continue the UAV example in Sect. 2 to illustrate the construction of optimization problem for Requirements Satisfaction Checking. The delivery mission of the UAV is to travel from position $\boldsymbol{x}_0$ to the destination $\boldsymbol{x}_d$ within time period $T$. Suppose the sensing range is $l$, meaning that only the obstacles and private regions whose distances to the UAV are shorter than $l$ can be detected. At current instant $k_0$, the UAV needs to plan its velocities and sensor configurations within the range of the minimal and maximal values, i.e., $\boldsymbol{v}_k \in [\boldsymbol{v}_{\min}, \boldsymbol{v}_{\max}]$ and $\boldsymbol{w}_k \in [\boldsymbol{w}_{\min}, \boldsymbol{w}_{\max}]$ for the following consecutive $N$ time steps, where $N \leq \left\lfloor \frac{l}{v_{\max}\tau} \right\rfloor$. For mission completion, we have $\boldsymbol{x}_{k_0+N} + \boldsymbol{v}_{k_0+N} \cdot (T - k_0 - N)\tau = \boldsymbol{x}_d$ to ensure the UAV could reach the destination finally. In this scenario, the safety and privacy requirements should be met at every time instant, while the timeliness, energy-saving, and accuracy requirements should be met at the end of the mission, i.e., $\mathcal{R}_{\mathcal{M}} = \{R_S, R_P\}, \mathcal{R}_{\mathcal{A}} = \{R_\xi, R_\varphi, R_E\}$. Thus, the slacked safety and privacy requirements for each obstacle and private region during planning horizon are constructed as shown in Eqs. (6a)–(6b); while the slacked timeliness, energy-saving and accuracy requirements are constructed based on the estimation of their QMs at the end of the mission, as shown in Eqs. (6c)–(6e). Based on Eq. (6), the Requirements Satisfaction Checking problem at $k_0$ can be formulated as below:

$$
\begin{aligned}
\min_{s,\boldsymbol{\delta},\boldsymbol{\zeta}} f_k = & \sum_{k=k_0+1}^{k_0+N} \Big( \sum_{o\in\mathcal{O}_k} \zeta_{S_o,k} + \sum_{c\in\mathcal{C}_k} \zeta_{P_c,k} \Big) + \frac{\delta_{\xi,T}}{\Delta - \Delta_t} \\
& + \frac{\delta_{E,T}}{E - E_t} + \frac{\zeta_{\varphi,T}}{A_t - A} \\
s.t. & \forall k \in [k_0, k_0 + N - 1], \\
& \boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \boldsymbol{v}_k \cdot \tau, \\
& \boldsymbol{v}_k \in [\boldsymbol{v}_{min}, \boldsymbol{v}_{max}], \boldsymbol{w}_k \in [\boldsymbol{w}_{min}, \boldsymbol{w}_{max}], \\
& \boldsymbol{x}_{k_0+N} + \boldsymbol{v}_{k_0+N} \cdot (T - k_0 - N)\tau = \boldsymbol{x}_d; \\
& \forall k \in [k_0 + 1, k_0 + N], \\
& 1 - \zeta_{S_o,k} \leq \mathcal{X}_{S_o}(k),\ 0 \leq \zeta_{S_o,k} \leq 1, \forall o \in \mathcal{O}_k, \qquad \text{(6a)} \\
& 1 - \zeta_{P_c,k} \leq \mathcal{X}_{P_c}(k),\ 0 \leq \zeta_{P_c,k} \leq 1, \forall c \in \mathcal{C}_k, \qquad \text{(6b)} \\
& \mathcal{X}_\xi(T) \leq \Delta_t + \delta_{\xi,T}, 0 \leq \delta_{\xi,T} \leq \Delta - \Delta_t, \qquad \text{(6c)} \\
& \mathcal{X}_E(T) \leq E_t + \delta_{E,T}, 0 \leq \delta_{E,T} \leq E - E_t, \qquad \text{(6d)} \\
& A_t - \zeta_{\varphi,T} \leq \mathcal{X}_\varphi(T),\ 0 \leq \zeta_{\varphi,T} \leq A_t - A. \qquad \text{(6e)}
\end{aligned}
$$

## 4.3 Requirements satisfaction analyzing

To reduce the risk of overly strict demands of requirements satisfaction [41], we need to decide which requirements should be accommodated for satisfaction optimization. The initial requirements set $\mathcal{R}$ is divided into the $\boldsymbol{\varepsilon}$-*Flexible* and $\boldsymbol{\varepsilon}$-*Inflexible* sets of requirements based on the result in the previous step.

Given the optimal solution of the slack variables for Eq. (6), denoted as $(\boldsymbol{\delta}^*, \boldsymbol{\zeta}^*)$, the requirements whose slack variables exceed zero are predicted to be unsatisfiable. To avoid the frequent reporting of unsatisfiable requirements and high time cost for the adaptation process, we only select requirements whose violation degree (defined as $V_i$) exceed a threshold to adjust. We define such as threshold as the violation tolerance $\varepsilon_i$ for each requirement $R_i$. Clearly, for hard constraints and missions, their violation tolerance should always be zero.

**Definition 1** (**$\boldsymbol{\varepsilon}$-Flexible and $\boldsymbol{\varepsilon}$-Inflexible sets of requirements**) Given a set of requirements $\mathcal{R} = \{R_1, \ldots, R_n\}$ with violation tolerance $\boldsymbol{\varepsilon} = (\varepsilon_1, \ldots, \varepsilon_n)$ where $\varepsilon_i \to 0^+$, the $\boldsymbol{\varepsilon}$-Flexible set of requirements is $\mathcal{R}_f = \{R_i \in \mathcal{R} : V_i > \varepsilon_i\}$, and the set of $\boldsymbol{\varepsilon}$-Inflexible requirements is $\mathcal{R}_{nf} = \mathcal{R} - \mathcal{R}_f$.

The violation degree $V_i$ for a requirement $R_i \in \mathcal{R}_{\mathcal{A}}$ can be computed as the sum of its normalized slack variables: $V_i = \overline{\delta^*_{i,T}}$ (or $\overline{\zeta^*_{i,T}}$). For $R_i \in \mathcal{R}_{\mathcal{M}}$, its violation degree is $V_i = \frac{1}{N}\sum_{k=k_0+1}^{k_0+N} \overline{\delta^*_{i,k}}$ (or $\frac{1}{N}\sum_{k=k_0+1}^{k_0+N} \overline{\zeta^*_{i,k}}$). Based on Definition 1, requirements in $\mathcal{M}$ and $\mathcal{H}$ should belong to $\mathcal{R}_{nf}$, while soft constraints whose violation degree exceeds threshold belong to $\mathcal{R}_f$. Otherwise, there are violations of hard constraints or missions to report.

**UAV scenario** In this example, the violation degrees for soft constraints of safety, privacy, timeliness, accuracy, and energy-saving requirements, which can be calculated as below:

*Safety:* $V_S = \sum_{k=k_0+1}^{k_0+N} \sum_{o\in\mathcal{O}_k} \frac{\zeta^*_{S_o,k}}{N|\mathcal{O}_k|}$

*Privacy:* $V_P = \sum_{k=k_0+1}^{k_0+N} \sum_{c\in\mathcal{C}_k} \frac{\zeta^*_{P_c,k}}{N|\mathcal{C}_k|}$

*Accuracy:* $V_\varphi = \frac{\zeta^*_\varphi}{A_t - A}$, *Timeliness:* $V_\xi = \frac{\delta^*_\xi}{\Delta - \Delta_t}$

*Energy-Saving:* $V_E = \frac{\delta^*_e}{E - E_t}$.

## 4.4 Requirements satisfaction optimizing

The purpose of this step is to mitigate the violation of unsatisfiable soft constraints while keeping requirements in $\mathcal{R}_{nf}$ fully satisfied. For $R_i \in \mathcal{R}_f$, their initial satisfaction demands (i.e., $\mathcal{X}_i$ achieves $g_i$) are relaxed, and they

are brought into the objective function to optimize. For $R_i \in \mathcal{R}_{nf}$, their initial satisfaction demands are kept. The reason is our insight that for the soft constraints whose violation degrees are less than their tolerances, they are likely to be satisfied by relaxing those in $\mathcal{R}_f$. To improve the real-time performance of the AUS, the objective function to optimize is formulated as the sum of satisfaction degree for all $\varepsilon$-*Flexible* requirements, while the constraints in the optimization problem are $\varepsilon$-*Inflexible* requirements. The requirements satisfaction optimization problem is formulated as:

$$\begin{aligned} \max_{\boldsymbol{u}} \quad & h_k = \sum_{R_i \in \mathcal{R}_M \cap \mathcal{R}_f} \frac{1}{N} \sum_{k=k_0+1}^{k_0+N} \mathcal{DS}(\mathcal{X}_i(k)) \\ & \quad + \sum_{R_i \in \mathcal{R}_A \cap \mathcal{R}_f} \mathcal{DS}(\mathcal{X}_i(T)) \\ s.t. \; & \forall R_i \in \mathcal{R}_{\mathcal{M}} \cap \mathcal{R}_{nf}, \forall k \in [k_0+1, ..., k_0+N], \\ & \mathcal{DS}(\mathcal{X}_i(k)) = 1; \\ & \forall R_i \in \mathcal{R}_{\mathcal{A}} \cap \mathcal{R}_{nf}, \; \mathcal{DS}(\mathcal{X}_i(T)) = 1. \end{aligned} \tag{7}$$

As shown in Eq. (7), the multi-objective optimization problem is scalarized into a single-objective problem with a combined sum of requirements satisfaction degree $\mathcal{DS}(\mathcal{X}_i(k))$ for $R \in \mathcal{R}_f$, while the constraints in the optimization problem are the fully satisfaction for $R \in \mathcal{R}_{nf}$. To ease the bias of the satisfaction between $R_i \in \mathcal{R}_{\mathcal{A}}$ and $R_i \in \mathcal{R}_{\mathcal{M}}$ with respect to the length of horizon $N$, we use the average satisfaction degree of $R_i \in \mathcal{R}_{\mathcal{M}}$ during the horizon as part of the objective function. Although there is no preference between requirements, predefined priorities are allowed in case of specific scenarios, and the functions satisfaction degree can be combined with weights in Eq. (7). As $\mathcal{R}_f$ is dynamically determined, the formulation of the objective function is also generated according to specific runtime situations.

### 4.5 Online adaptation of `Captain`

We here describe the online adaptation process of `Captain`, i.e., how to combine the *Requirements Satisfaction Checking* (Sect. 4.2), the *Requirements Satisfaction Analyzing* (Sect. 4.3), and the *Requirements Satisfaction Optimizing* (Sect. 4.4) during the adaptation process. Algorithm 1 shows the process of generating the adaptation plan for the AUS to complete the mission with the optimal requirements satisfaction. The algorithm receives a set $\mathcal{R}$ of requirements and its requirements satisfaction model $\mathcal{DS}$ as inputs. The output of `Captain` is an adaptation plan for the AUS.

During the operation, both the environment and system are continuously monitored until the assigned missions

**Algorithm 1:** Requirements Satisfaction Model Driven Online Adaptation for AUSs

**Input**: Time instant $k$, requirements set $\mathcal{R}$ and requirements satisfaction model $\mathcal{DS}$.
**Output**: The adaptation plan for AUS to complete the mission.

```
1  Obs_k = (O_k, C_k, ⟨s_0, s_1, ..., s_k⟩);
2  (u*_k, δ*, ζ*) = arg min f_k;
3  if u*_k = ∅ then
4      Report false and take some emergency actions;
5      break;
6  else
7      Compute requirement violation degree {V_i} for R based on
       (δ*, ζ*);
8      Determine R_f and R_nf;
9      if R_f ≠ ∅ then
10         u'*_k = arg max h_k;
11         if u'*_k ≠ ∅ then
12             u*_k ← u'*_k;
13 return adaptation plan of AUS u*_k;
```

are completed, e.g., reach the destination. Once changes in the environment or system are detected, the online adaptation process is triggered. The `Captain` first perceives the changes in environmental conditions and system states (Line 1). Second, the optimization problem for *Requirements Satisfaction Checking* (Line 2) is solved. No feasible solution to the optimization problem indicates the violations of hard constraints or missions. This failure is reported for the human pilot to take emergency action (Line 4), such as landing or surrendering control to a human pilot. If the optimal solution $\boldsymbol{u}_k^*$ exists, in *Requirements Satisfaction Analyzing*, the violation degree of each requirement is computed (Line 7), and $\mathcal{R}$ is divided into $\boldsymbol{\varepsilon}$-*Flexible* and $\boldsymbol{\varepsilon}$-*Inflexible* sets of requirements (Line 8). In case $\mathcal{R}_f = \emptyset$, we skip the *Requirements Satisfaction Optimizing* step and leverage the solution $\boldsymbol{u}_k^*$ directly. Otherwise, the optimization problem for *Requirements Satisfaction Optimizing* (Line 10) is constructed and solved. If another optimal solution $\boldsymbol{u'}_k^*$ exists, $\boldsymbol{u}_k^*$ is updated with $\boldsymbol{u'}_k^*$ (Line 12) for the AUS to act.

Considering that the optimization problems in Eq. 6 and Eq. 7 can be non-convex with nonlinear constraints, sequential convex programming (SCP) as an optimization method for non-convex problems can be leveraged. Among SCP, sequential quadratic programming (SQP) is a typical nonlinear programming method that is realized through solving a set of approximate convex problems [42]. We use SQP to solve Eq. 6 and Eq. 7, as it outperforms other optimization methods in terms of efficiency, accuracy, and success rate [43].

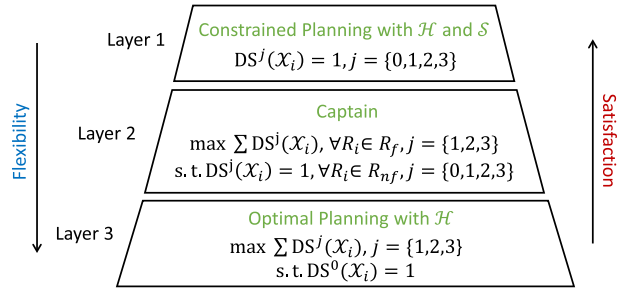



**Fig. 3** The hierarchical layout of adaptation strategies for AUSs with multiple requirements

## 5 Assessments for `Captain`

Figure 3 demonstrates a hierarchical layout of adaptation strategies for the AUS with multiple requirements to satisfy. The underlying idea of the adaptation strategy in the top layer is to find a solution that can satisfy all hard constraints $\mathcal{H}$ and soft constraints $\mathcal{S}$ for mission completion in the feasible region $\mathscr{F} = \{\mathcal{X}|\mathrm{DS}^j(\mathcal{X}) = 1, j = \{0, 1, 2, 3\}\}$. The strategy at the bottom layer is to find optimal planning with $\mathcal{H}$ satisfied by transforming all soft constraints into the objective function to optimize such that the feasible region expands to $\mathscr{F}' = \{\mathcal{X}|\mathrm{DS}^j(\mathcal{X}) = 1, j = \{0\}\}$. It is worth noting that `Captain` adopts a new adaptation strategy, located in the middle layer, which identifies and transforms only unsatisfiable soft constraints in $\mathcal{R}_f$ into the objective function to optimize while keeping the other requirements $\mathcal{R}_{nf}$ fully satisfied.

### 5.1 Optimality of `Captain`

If constrained planning with $\mathcal{H}$ and $\mathcal{S}$ (Layer 1 in Fig. 3) is solvable, the solution can keep all the hard and soft constraints fully being satisfied. However, uncertainties in the environment may lead to requirements violations, resulting in $\mathscr{F} = \emptyset$. In that case, the AUS cannot find any solution, and may be trapped into mission failure. One remedy to this situation is to expand $\mathscr{F}$ for higher flexibility, i.e., optimizing satisfaction of all soft constraints together in Layer 3. However, the feasible domain $\mathscr{F}'$ is so large, and it can be biased to make trade-offs between soft constraints according to predefined preferences. Hence, as shown in Layer 2, `Captain` tries to achieve a balance between flexibility and requirements satisfaction by selecting unsatisfiable soft constraints to optimize while keeping the other requirements fully satisfied. In this way, the optimality of `Captain` is assured when a feasible solution is found.

### 5.2 Flexibility of `Captain`

In `Captain`, only selected soft constraints in $\mathcal{R}_f$ need to be accommodated, and such a selection is made based on the results of the two steps introduced previously: *Requirements Satisfaction Checking* and *Requirements Violation Analyzing*. In case that $\mathscr{F} \neq \emptyset$, the *Requirements Satisfaction Checking* step will also generate an optimal solution in which all requirements are fully satisfied and this solution is the same as that solved by Layer 1 in Fig. 3. Otherwise, the *Requirements Satisfaction Optimization* step is launched to find a sub-optimal solution. In the worst case, when all soft constraints are unsatisfiable, `Captain` degenerates to Layer 3 in Fig. 3. In this way, `Captain` is flexible to be compatible with other control-based methods. Such flexibility ensures the applicability of `Captain` in complex situations with multiple requirements.

### 5.3 Proactive detection of requirements violations

Apart from handle violations of soft constraints, `Captain` can also detect violations of hard constraints and missions in advance. In `Captain`, both $\mathcal{M}$ and $\mathcal{H}$ are modeled as constraints that must be met in Eqs. 6 and 7. The two optimization problems in Eqs. 6 and 7 are constructed as convex optimization and solved by SQP. The infeasibility of these two optimization problems implies violations of hard constraints or mission failures, as SQP solver can produce feasible results with respect to bounds. In that case, the AUS is required to report the failures and take emergency pause/stop action, or system control is transferred to a human pilot, as shown in Line 4 in Algorithm 1.

### 5.4 Real-time performance

Since our approach needs to solve optimization problems during AUS operation, it is critical to consider timing issues that could prolong the system's execution. For example, for UAVs hovering in the sky, the adaptation plan should be calculated timely for their future actions. In our approach, Eqs. 6 and 7 are solved within prediction horizon $N$ based on MPC. A receding horizon technique [17] is adopted such that only the first action of the plan is applied. In dynamic environments, the system will follow the calculated plan until information about new changes in the environment and system is available or at the end of the horizon; at that time a new iteration of `Captain` is launched to generate a fresh plan. The prediction horizon $N$ is determined based

on the sensing range $l$ and the span of time instant $\tau$, i.e., $N \leq \left\lfloor \frac{l}{v_{\max}\tau} \right\rfloor$. $\tau$ represents the granularity of control, which can also affect the real-time performance. Given the span of the time instant, the prediction horizon can be tuned according to the computational capability of the UAV. Additionally, the two optimization problems are solved by SQP, which outperforms other optimization methods in terms of efficiency, accuracy, and percentage of successful solutions [43]. Further, the hot-start strategy [44] can be used to accelerate the solving of optimization problems.

## 6 Experimental evaluation

We evaluate the practicality, robustness, scalability, and real-time performance of `Captain` in two AUS cases using MATLAB as well as an implementation on a real UAV (DJI Matrice 100)[1]. Our experiments focus on the following questions:

*RQ1* (*Practicality*) Can `Captain` be implemented in simulated and real-world scenarios to detect unsatisfiable requirements and provide adaptive solutions for the optimal requirements satisfaction at runtime?
*RQ2* (*Robustness*) How robust is `Captain` against different types of unpredictable conditions?
*RQ3* (*Scalability*) How scalable is `Captain` in large environment (e.g. extracted from real urban scenarios)?
*RQ4* (*Real-time performance*) What is the real-time performance of `Captain` in terms of computation overhead?

### 6.1 Experimental design and settings

#### 6.1.1 Scenarios

We consider two AUS scenarios, i.e., the UAV Delivery scenario mentioned in Sect. 2 and the UUV ocean surveliance scenario extracted from [15,16]. We also implement `Captain` on the DJI Matrice 100 UAV in the real world.

**UAV Delivery scenario** This example is given in Sect. 2. The requirements in this scenario are listed in Table 1, with violation tolerances of $10^{-20}$ (safety), $10^{-20}$ (privacy), $10^{-10}$ (accuracy), $10^{-20}$ (timeliness), and 0.005 (energy-saving). We evaluate `Captain` in three environment settings for the UAV case, whose parameters are listed in Table 2. We assume that the obstacles in the same setting have the same radius, as do the private regions. The volume ratios of the obstacles and private regions to the c-space [34] are characterized by $\rho_o$ and $\rho_p$, respectively. The higher the ratio is, the more cluttered the environment is. Once the adaption plan is generated for the UAV, the low-level commands to the actuators/motors are generated via a PD (proportional-differential) controller modified from the quadrotor PD controller for CrazyFlie 2.0 [45] [2]. In Setting 1, the obstacles and the private regions are randomly generated, in contrast to the environment in Settings 2 and 3, which are extracted from two real urban scenarios according to the open building dataset of Portland, USA [46].

**UUV oceanic surveillance scenario** This example originates from [15,16], in which the UUV is used for oceanic surveillance for a time period, i.e., $\Delta = 10$ hours. Rather than considering single-objective optimization for the UUV scenario in [15,16], we extend the original scenario to multiple requirements satisfaction under runtime uncertainties through `Captain`, while other settings like sensor parameters are kept the same. The requirements for the UUV to achieve are listed as follows:

- Accuracy ($R_\varphi$): $\mathcal{S}_\varphi$: the accuracy of sensor measurements is targeted at $A_t = 90\%$; $\mathcal{H}_\varphi$: the threshold is set as $A = 80\%$.
- Scanning Distance ($R_L$): $\mathcal{S}_L$: A segment of the surface over a distance of $L_t = 100$ km is expected to be examined by the UUV; $\mathcal{H}_L$: the threshold of surveillance distance is $L = 90$ km.
- Energy-saving ($R_E$): $\mathcal{S}_E$: A total amount of energy $E_t = 5.4$ MJ is expected to be consumed; $\mathcal{H}_E$: the maximum amount of energy is $E = 6$ MJ.

**DJI Matrice 100 UAV scenario** We implement `Captain` on a DJI Matrice 100 UAV in real world. The UAV's mission is to move along a reference path while avoiding intruding privacy regions and saving motion time. A Lenovo laptop controls UAV motion via an Android application developed with the DJI mobile SDK on a mobile phone.

#### 6.1.2 Baseline approaches

Considering the importance of adaptation strategy in terms of the effectiveness and efficiency in requirements satisfaction, we compare the online adaptation process of `Captain` with two approaches: one is the control-based requirements adaptation algorithm, and the other is a variant of `Captain`.

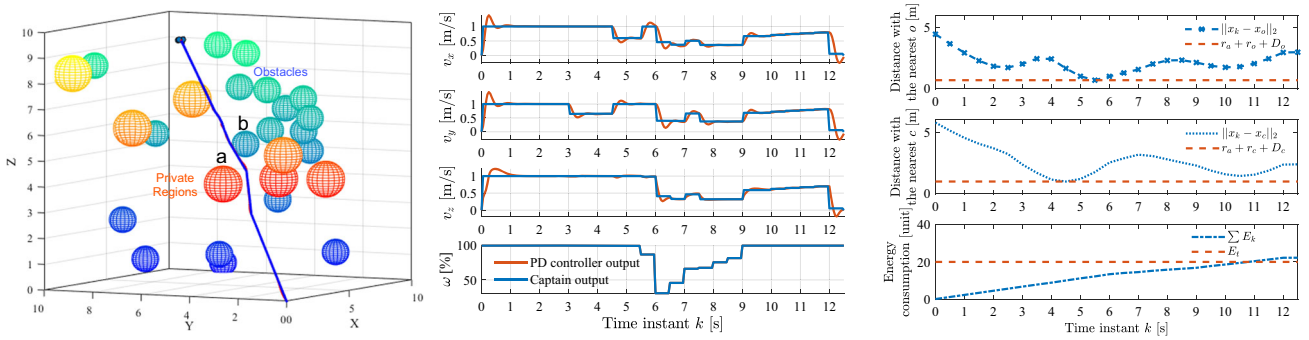
- AMOCS-MA: it adapts from the state-of-the-art control-based adaptation approaches for multiple requirements [17] and generates adaptation plans using the maximization of satisfaction degree of all soft constraints as the

[1] https://github.com/YixingLuo/Captain.

[2] https://github.com/yrlu/quadrotor.

**Table 2** Parameters of UAV Delivery case

| Parm. | Setting 1 | Setting 2 | Setting 3 |
|---|---|---|---|
| $l$ | 2.5 m | 20 m | 20 m |
| $\boldsymbol{v}$ | $[-1, 1]$ m/s | $[-10, 10]$ m/s | $[-10, 10]$ m/s |
| $\boldsymbol{w}$ | 0–100% | 0–100% | 0–100% |
| $r_a$, $\tau$ | 0.2 m, 0.5 s | 0.2 m, 0.5 s | 0.2 m, 0.5 s |
| $\eta_1$, $\eta_2$ | 0.5, 0.2 | 0.5, 0.2 | 0.5, 0.2 |
| Scale | (10, 10, 10) m | (500, 500, 100) m | (1000, 1000, 100) m |
| $r_o$, $r_c$ | 0.3 m, 0.5 m | 5 m, 5 m | 5 m, 5 m |
| $\rho_o$, $\rho_c$ | [0%, 100%] | (1.64%, 2.41%) | (2.87%, 2.84%) |
| $D_o$, $D_c$ | 0.2 m, 0.3 m | 5 m, 10 m | 5 m, 10 m |
| $\Delta_t$, $\Delta$ | 15 s, 30 s | 60 s, 90 s | 90 s, 150 s |
| $A_t$, $A$ | 90%, 80% | 90%, 80% | 90%, 80% |
| $E_t$, $E$ | 20 unit, 40 unit | 100 unit, 150 unit | 200 unit, 300 unit |




**(a)** Flight Path of UAV. **(b)** Velocity and Sensor Configuration. **(c)** Requirements Satisfaction during the Flight.

**Fig. 4** RQ1—Adaption planning results of `Captain` in UAV case

objective function. This approach is chosen as it represents the adaptation strategy as shown in Layer 3 of Fig. 3.

- GSlack: it generates adaptation plans through the application of optimization problem in the *Requirements Satisfaction Checking* step. The objective function is the minimization of the violation of soft constraints for all requirements. This approach is compared with the whole online adaptation process of `Captain` to demonstrate the necessity of *Requirements Satisfaction Analyzing* and *Requirements Satisfaction Optimizing* in the optimality of requirements satisfaction.

#### 6.1.3 Configurations and implementations

For the UAV Delivery scenario and UUV Ocean Surveillance scenario, we conduct simulations based on MATLAB 2019b on a desktop equipped with an Intel(R) Core(TM) i7-7700U CPU@3.60GHz and 32GB RAM. The constructed optimization problems are solved by the Optimization Toolbox of MATLAB.

In the DJI Matrice 100 UAV scenario, UAV is equipped with a Zenmuse Z3 camera, whose orientation angle $\theta$ can be reconfigured from $-90°$ (vertically towards the ground) to 30° (up to the sky). The Lenovo laptop used to control DJI Matrice 100 UAV is equipped with an Intel(R) Core(TM) i7-7500U CPU@2.70GHz and NVIDIA GeForce 940MX.

### 6.2 RQ1: practicality

We investigate the practicality of `Captain` with response to runtime uncertainties through the implementation of `Captain` in two simulated scenarios, i.e., the UAV Delivery and UUV Ocean Surveillance, and a real-world scenario with a DJI Matrice 100 UAV.

#### 6.2.1 UAV Delivery scenario

We simulate the UAV scenario in Setting 1, where $\rho_o = 1\%$ (19 obstacles) and $\rho_p = 1\%$ (7 private regions). As shown in Fig. 4a, the small blue spheres represent obstacles, and the large red ones represent private regions. The blue curves show the planned path via `Captain`. The velocity and sensor configuration during operation are shown in Fig. 4b, where the planned behaviors and actual outputs are blue and red

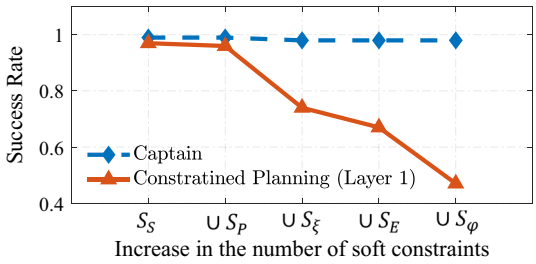



**Fig. 5** RQ1—Success rate with increase of soft constraints

lines, respectively. In Fig. 4c, the three blue lines (from top to bottom) indicate the distances between UAV and the nearest obstacle, the distances between UAV and the nearest private region, and accumulated energy consumption, respectively. The red lines are the target values of soft constraints listed in Table 2.

During the flight, UAV detects private region $a$ at $k = 3$s. To avoid intruding into $a$, UAV reduces velocity in the y-axis. At $k = 4$s, the UAV then detects obstacle $b$. To achieve the optimal satisfaction of safety and privacy requirements, UAV plans a new velocity and changes direction. Additionally, at $k = 4.5$s, the parameters $\eta_1$ and $\eta_2$ increase to 0.53 and 0.23, respectively, and UAV plans to reduce its sensor configuration to conserve energy since $k = 6$s. However, at $k = 6.5$s, the energy-saving requirement is checked to no longer be satisfied as $V_e = 0.015 > 0.005$. So the target energy consumption is relaxed to 40 units. Furthermore, at $k = 8$s, $\boldsymbol{v}_{\max}$ is limited to (0.96, 0.96, 0.96), and at $k = 9.5$s the time target $\Delta_t$ is tightened to 12s. With such changes, UAV increases its speed to minimize the degree to which timeliness is violated. Finally, UAV reaches the destination with $\mathcal{X}_\xi = 12.5$s, $\mathcal{X}_\phi = 90\%$, $\mathcal{X}_e = 22.25$ units and no violations of safety and privacy requirements.

Moreover, to illustrate `Captain`'s effectiveness to generate in adaptive solution generation, we investigate the mission success rate as the increasing number of soft constraints, from $\{\mathcal{S}_S\}$, $\{\mathcal{S}_S, \mathcal{S}_P\}$ to $\{\mathcal{S}_S, \mathcal{S}_P, \mathcal{S}_\xi, \mathcal{S}_E, \mathcal{S}_\varphi\}$. We simulate 100 randomly generated uncertain conditions and diverse environments. Thus, we could estimate the success rate of solution generation of `Captain` in unpredictable and unaccommodating environments. We also compare the results with the approach we mentioned in Fig. 3 Layer 1. Under constrained planning with $\mathcal{H}$ and $\mathcal{S}$, all of the requirements, both soft and hard constraints, must be satisfied. However, this could lead to a higher possibility of no solutions, as shown in Fig. 5 where the success rate drops to 47% as the number of soft constraints increases. However, `Captain` can maintain a high success rate ($\geq 95\%$), because `Captain` is able to transform unsatisfiable soft constraints into optimizable objectives flexibly when necessary.

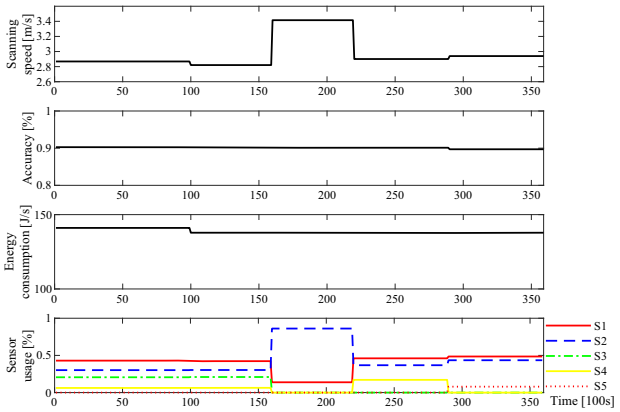



**Fig. 6** RQ1—Adaption planning results of `Captain` in UUV case

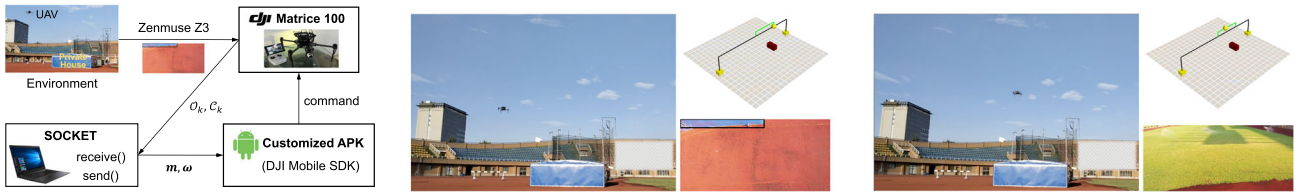
### 6.2.2 UUV oceanic surveillance scenario

To demonstrate the practicality of `Captain` in the UUV Oceanic Surveillance Scenario, we construct the requirement satisfaction model in this scenario: $\mathrm{DS}^1(\mathcal{X}_L(T))$ (scanning distance), $\mathrm{DS}^2(\mathcal{X}_E(T))$ (energy-saving) and $\mathrm{DS}^2(\mathcal{X}_\varphi(T))$ (accuracy). There are trade-offs between the satisfaction of three requirements, e.g., when sensors (e.g., sensor 1) with a higher quality of surveillance are chosen, more energy is consumed, resulting in less distance scanned.

Figure 6 shows the adaptation process of the UUV with `Captain` during operation. At $k = 100$, the available energy $E_t$ drops 5.0 MJ and at $k = 160$, the distance to be scanned $L_t$ increase to 105 km. These two changes trigger the online adaptation process of `Captain`, which leads to corresponding changes in sensor usage arrangement as the time portion for $S2$ increases. Figure 6 also shows how `Captain` reacts to disturbances of system characteristics like sensor failures. At $k = 220$, the measurement accuracy of sensor $S3$ drastically decreases from 83 to 43%, and at $k = 290$, $S4$ stops working. These changes lead to the adaptive solution that $S1$ is more exploited. Finally, the mission ends with an average measurement accuracy of 90.1%, scanning distance at 106.7 km, and energy consumption at 4.98 MJ.

### 6.2.3 DJI Matrice 100 UAV scenario

We implement `Captain` on a DJI Matrice 100 UAV. The experiment setups are depicted in Fig. 7a. `Captain` is deployed on a Lenovo laptop which can capture the environment contexts from the UAV's onboard camera and generate the trajectory for the UAV to follow. The trajectory is sent to the DJI Matrice 100 UAV via an Android application developed with the DJI mobile SDK on a mobile phone. As the UAV is controlled by a customized Android application deployed on a remote controller, a communication




**(a)** Experiment setups.

**(b)** UAV detects the private region (blue house) and *Requirements Satisfaction Checking* and *Analyzing* is launched.

**(c)** UAV changes its path and sensor configuration according to the planning results of *Requirements Satisfaction Optimizing*.

**Fig. 7** Experiment results with DJI Matrice 100 UAV

delay needs to be taken into consideration, and the UAV will wait for the next command.

The operating environment is discretized with the scale of $20 \times 20 \times 10\,\mathrm{m}^3$. The UAV moves at a constant speed of 0.5m/s, in steps of time span $\tau = 2$s, with a target distance from private regions of $D_c = 4.5$ m. The prediction horizon is $N = 2$. The orientation angle of the camera ranges from $-30°$ to $90°$, which is normalized as $w \in [1/3, 1]$. The flight mission is to travel from a starting point to the destination in a timely manner without intruding into private regions. As shown in Fig. 7b, c, the pictures on the left are screenshots of UAV during the flights. The pictures at the top right corner are the environment models: the starting position $\boldsymbol{x}_s$ and destination $\boldsymbol{x}_d$ are marked as yellow cubes, the detected private regions are marked as red cubes, the yellow spheres indicate current positions of UAV, the black line is the reference trajectory that UAV is expected to follow, the green path is the replanned trajectory for optimal requirements satisfaction. The pictures at the lower right corner of Fig. 7b, c are the views of onboard camera at current time step.

During the flight, the onboard camera takes pictures of the surroundings and sends pictures to the laptop. Then, private regions are identified and localized according to the method in [32]. As shown in Fig. 7b, at time instant $k = 14$ s, UAV detects and locates a private region within the black box in the photo of camera view. So `Captain` is launched first to check whether the timeliness and privacy requirements could be satisfied. After *Requirements Violation Analysis*, `Captain` finds that those two requirements could not be satisfied simultaneously, and it provides a sub-optimal trajectory, i.e., flying up and tuning the orientation angle and resolution of the camera to avoid exposing the private house (Fig. 7c). Finally, UAV reaches the destination after 52 s at the velocity of 0.5 m/s with timeliness satisfaction of $DS^1(\mathcal{X}_\varphi) = 91.7\%$ and privacy preservation satisfaction of $DS^2(\mathcal{X}_P) = 96.6\%$. The results show that `Captain` can generate an adaptive trajectory to mitigate requirements violations effectively. Videos of simulations and real-world experiments can be found on the website[3].

According to our experience, to apply `Captain` in the real context needs the following three steps: (1) elicit requirements for the AUS in real context and establish the requirements satisfaction model; (2) determine what types of runtime uncertainties need to handle and enable the AUS to perceive these changes during operation; (3) deploy `Captain` in AUS to generate the adaptation plan at runtime. To our knowledge, the main difference between simulations and real experiments is the delay of perception and actuation in the real context, which may affect the effectiveness of the adaptive planning process of `Captain`.

**Answer to RQ1**: `Captain` is practical both in simulated scenarios and real-world workloads to detect unsatisfiable requirements and generate effective adaptation solutions in response to runtime uncertainties.

## 6.3 RQ2: robustness

To demonstrate the robustness of `Captain` against dynamic disturbances in system characteristics and variations from the environment, in terms of higher requirements satisfaction degree under runtime uncertainties, we compare the planning results of `Captain`, AMOCS-MA, and GSlack.

### 6.3.1 UAV Delivery scenario

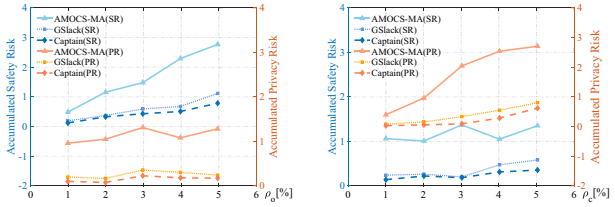
We measure the UAV's behaviors in the environment with varied densities of obstacles and private regions, variations of sensor parameters ($\eta_1, \eta_2$, speed limit $\boldsymbol{v}_{\max}$, and sensor reconfiguration range $\boldsymbol{w}_{\max}$). We simulate runtime disturbances with 1000 randomly generated scenarios.

[3] https://yixingluo.github.io/Captain.github.io/.

**Table 3** RQ2—Requirements satisfaction results of UAV with disturbances in system characteristics

| Requirements approaches | | Accuracy ($\mathcal{S}_\varphi$) ($\mathcal{X}_\varphi$ [%], $\mathrm{DS}^2(\mathcal{X}_\varphi(T))$) | | | Timeliness ($\mathcal{S}_\xi$) ($\mathcal{X}_\xi$ [$s$], $\mathrm{DS}^1(\mathcal{X}_\xi(T))$) | | | Energy-saving ($\mathcal{S}_E$) ($\mathcal{X}_E$ [$unit$], $\mathrm{DS}^1(\mathcal{X}_E(T))$) | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | AMOCS-MA | GSlack | `Captain` | AMOCS-MA | GSlack | `Captain` | AMOCS-MA | GSlack | `Captain` |
| # Of incidents | 2 | 88.48 | 89.52 | 89.86 | 15.48 | 13.60 | 12.53 | 24.57 | 22.53 | 21.93 |
| | | 85.98% | 93.71% | 95.88% | 96.23% | 99.75% | 99.80% | 76.85% | 87.18% | 90.17% |
| | 4 | 87.03 | 89.16 | 89.57 | 15.43 | 13.29 | 12.34 | 24.57 | 22.59 | 21.99 |
| | | 78.63% | 89.68% | 93.39% | 92.86% | 99.45% | 99.73% | 74.90% | 86.86% | 89.87% |
| | 6 | 86.85 | 88.78 | 89.14 | 15.62 | 13.30 | 12.54 | 24.85 | 22.79 | 22.22 |
| | | 74.30% | 86.96% | 90.15% | 92.63% | 99.39% | 99.83% | 74.18% | 85.96% | 88.84% |

[a] The violation tolerance is $\epsilon_{S,P,\varphi,\xi,E} = \{10^{-20}, 10^{-20}, 10^{-10}, 10^{-20}, 0.005\}$ with $\rho_o = 2\%$, $\rho_p = 2\%$




**(a)** $\rho_c = 2\%$. **(b)** $\rho_o = 2\%$.

**Fig. 8** RQ2—Accumulated safety and privacy risks with different disturbances of $\rho_o$ and $\rho_c$

As shown in Fig. 8, $\rho_o$ and $\rho_c$ range from 1% (i.e., $|\mathcal{O}| = 19$, $|\mathcal{C}| = 7$) to 5% (i.e., $|\mathcal{O}| = 95$, $|\mathcal{C}| = 35$). For different combinations of $\rho_o$ and $\rho_c$, we compare the three approaches in terms of the accumulated safety risk $SR = \sum_{k=0}^{T}(1-\mathcal{X}_S(k))$ and privacy risk $PR = \sum_{k=0}^{T}(1-\mathcal{X}_P(k))$. The higher risks are, the lower satisfaction of safety and privacy requirements. Fig. 8a shows the results for different values of $\rho_o$ with a fixed $\rho_c = 2\%$. Due to the requirements modeling for AUSs, we find that UAV with `Captain` performs the best in satisfying safety and privacy requirements, with the lowest accumulated safety and privacy risks on average. Fig. 8b shows the results of different $\rho_c$ when $\rho_o = 2\%$. It shows that the planning results of `Captain` also have the lowest accumulated safety and privacy risks. Hence, compared to the two baseline approaches, `Captain` is more robust against environmental changes, such as unknown obstacles and privacy regions.

To test the robustness of `Captain` against the disturbances in system characteristics, we randomly add several disturbances ($\sim 10\%$) of sensor parameters at different time instants during the flight of UAV. The simulation results are shown in Table 3, where the number of incidents indicates the frequency of disturbances, the upper value of each cell in Table 3 is the average value of $\mathcal{X}_i$ of the requirements, and the percentage below represents the average requirements satisfaction degree $\mathrm{DS}^j(\mathcal{X}_i)$. We find that as the number of introduced disturbances increases, there is a decrease in requirements satisfaction degree. However, the planning results of `Captain` show the highest requirements satisfaction degree than the two baseline approaches.

We next perform a statistical test of the requirements satisfaction of three soft constraints for the planning results of the approaches across the 1000 runs. Following a guideline [47], we use the Wilcoxon signed-rank test [48] and the Vargha-Delaney's $\hat{A}_{12}$ effect size [49]. Using significance level $\alpha$=0.01, we observe that `Captain` is always significantly better than all the other approaches. We also use Vargha-Delaney's $\hat{A}_{12}$ [49] to assess effect size. Table 5 reports the statistical test results comparing the requirements satisfaction degree of the soft constraints (i.e., accuracy $\mathcal{S}_\varphi$, timeliness $\mathcal{S}_\xi$, energy-saving $\mathcal{S}_E$) under the different number of incidents (i.e., 2, 4, 6) for trajectories planned by AMOCS-MA, GSlack, and `Captain`. As shown in the table 5, `Captain` and GSlack outperform AMOCS-MA with a large effect size in the requirements satisfaction of $\mathcal{S}_\varphi$, $\mathcal{S}_\xi$, and $\mathcal{S}_E$. Moreover, we notice that `Captain` outperforms GSlack with a large effect size in requirements satisfaction $\mathcal{S}_\varphi$ and $\mathcal{S}_E$, and medium effect size in $\mathcal{S}_\xi$. The reason for the medium effect size is that the timeliness requirement is more likely to be satisfiable during the flight, and `Captain` outperforms GSlack narrowly in $\mathcal{S}_\xi$ (close to 100%), as shown in Table 3.

### 6.3.2 UUV oceanic surveillance scenario

In this scenario, the robustness of the three approaches is compared while adding random disturbances to parameters of sensors, i.e., sensor accuracy, scanning distance per second, and energy consumption per second. For each approach, we add different times of random disturbances at different time instants during the simulation. For a determined number of disturbances, we simulate 1000 rounds and compute the average accuracy, scanning distance, and energy consumption. The simulation results are shown in Table 4, where the number of incidents indicates the frequency of disturbances, the upper value of each cell is the average value of $\mathcal{X}_i$, and the percentage below represents the average requirements satisfaction degree $\mathrm{DS}^j(\mathcal{X}_i)$.

**Table 4** RQ2—Requirements satisfaction results of the UUV with disturbances in system characteristics

| Requirements | | Accuracy ($\mathcal{S}_\varphi$) $\left(\mathcal{X}_\varphi\,[\%], \mathrm{DS}^2(\mathcal{X}_\varphi(T))\right)$ | | | Scanning Distance ($\mathcal{S}_L$) $\left(\mathcal{X}_L\,[km], \mathrm{DS}^2(\mathcal{X}_L(T))\right)$ | | | Energy-saving ($\mathcal{S}_E$) $\left(\mathcal{X}_E\,[MJ], \mathrm{DS}^1(\mathcal{X}_E(T))\right)$ | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Approaches | | AMOCS-MA | GSlack | `Captain` | AMOCS-MA | GSlack | `Captain` | AMOCS-MA | GSlack | `Captain` |
| # Of incidents | 3 | 88.64 | 89.60 | 89.64 | 99.52 | 101.59 | 102.36 | 5.41 | 5.32 | 5.29 |
| | | 86.38% | 95.94% | 96.42% | 95.24% | 99.12% | 99.81% | 98.81% | 100% | 100% |
| | 6 | 87.41 | 88.80 | 88.95 | 98.71 | 100.99 | 101.38 | 5.43 | 5.37 | 5.35 |
| | | 74.10% | 88.02% | 89.52% | 87.12% | 98.60% | 98.90% | 95.60% | 100% | 100% |
| | 9 | 85.95 | 87.55 | 87.77 | 97.43 | 100.06 | 100.45 | 5.45 | 5.38 | 5.36 |
| | | 59.54% | 75.47% | 77.69% | 74.31% | 92.84% | 93.83% | 91.05% | 99.27% | 99.72% |

[a] The violation tolerance are $\epsilon_\varphi = 10^{-3}, \epsilon_L = 0, \epsilon_E = 0$

**Table 5** RQ2—Statistical test results comparing the requirements satisfaction degrees of adaptation plans generated by AMOCS-MA, GSlack and `Captain` in the UAV Delivery and UUV Ocean Surveillance scenarios

| approaches | UAV Delivery ($\hat{A}_{12}$) Requirements | # Of incidents | | | UUV Ocean Surveillance ($\hat{A}_{12}$) Requirements | # of incidents | | |
|---|---|---|---|---|---|---|---|---|
| | | 2 | 4 | 6 | | 3 | 6 | 9 |
| GSlack versus AMOCS-MA | Accuracy | 1 (L) | 1 (L) | 1 (L) | Accuracy | 1 (L) | 1 (L) | 1 (L) |
| | Timeliness | 1 (L) | 1 (L) | 1 (L) | Scanning Distance | 1 (L) | 1 (L) | 1 (L) |
| | Energy-saving | 1 (L) | 1 (L) | 1 (L) | Energy-saving | 1 (L) | 1 (L) | 1 (L) |
| `Captain` vs. AMOCS-MA | Accuracy | 1 (L) | 1 (L) | 1 (L) | Accuracy | 1 (L) | 1 (L) | 1 (L) |
| | Timeliness | 1 (L) | 1 (L) | 1 (L) | Scanning Distance | 1 (L) | 1 (L) | 1 (L) |
| | Energy-saving | 1 (L) | 1 (L) | 1 (L) | Energy-saving | 1 (L) | 1 (L) | 1 (L) |
| `Captain` vs. GSlack | Accuracy | 0.84 (L) | 0.86 (L) | 0.88 (L) | Accuracy | 0.78 (L) | 0.87 (L) | 0.93 (L) |
| | Timeliness | 0.76 (L) | 0.70 (M) | 0.68 (M) | Scanning Distance | 0.56 (S) | 0.64 (M) | 0.71 (L) |
| | Energy-saving | 0.95 (L) | 0.95 (L) | 0.96 (L) | Energy-saving | 0.58 (S) | 0.63 (S) | 0.73 (L) |

[a] L: large effect size, with $\hat{A}_{12} \geq 0.71$; M: medium effect size, with $\hat{A}_{12} \geq 0.64$, S: small effect size, with $\hat{A}_{12} \geq 0.56$

We can see that under `Captain`, the UUV can scan a longer distance with higher accuracy and less energy consumption than AMOCS-MA and GSlack on average. Moreover, `Captain` outperforms AMOCS-MA and GSlack with higher requirements satisfaction degree of the three requirements, especially for the accuracy requirement $\mathcal{S}_\varphi$. The reason behind this is that in the *Requirements Satisfaction Analysis* step, the accuracy is selected as the requirement for accommodation while the other two remain as their original specifications. We also compare the planning results in Table 4, for each pair of approaches soft constraints (i.e., accuracy $\mathcal{S}_\varphi$, scanning distance $\mathcal{S}_L$, energy-saving $\mathcal{S}_E$) under the different number of incidents (i.e., 3, 6, 9), using the same statistical tests used in UAV Delivery scenario. With significance level $\alpha$=0.01, we observe a significant difference between the `Captain` and baseline approaches. We also use effect size measure $\hat{A}_{12}$ for the statistical test. Table 5 shows that `Captain` and GSlack outperform AMOCS-MA with large effect size in the requirements satisfaction of $\mathcal{S}_\varphi$, $\mathcal{S}_L$, and $\mathcal{S}_E$. Moreover, in most cases, `Captain` outperforms GSlack in requirements satisfaction with large effect size, and we notice that as the number of introduced disturbances increases, the planning results generated by `Captain` are more significant to achieve higher requirements satisfaction than GSlack.

**Answer to RQ2**: With `Captain`, UAV and UUV have more robust performance to achieve higher requirements satisfaction than baseline approaches under different runtime uncertainties.

### 6.4 RQ3: scalability

To demonstrate the scalability of `Captain` in different scales of environment, we simulate the UAV case on two selected real urban environments of different scales from the open building dataset of Portland in USA [46]. In the dataset, we obtain the longitude and latitude of the center, average height, and usage (i.e., industrial and commercial buildings, houses, and apartments for residents) of each building. We use ArcGIS map to set up a 3D model according to the method in [50]. Buildings for commercial or industrial use are viewed as obstacles, while residential buildings are viewed as private regions. The original spaces we selected are $500 \times 500 \times 100$

$m^3$ and $10^3 \times 10^3 \times 100\ \mathrm{m}^3$, but these spaces are compressed into the scales of $50 \times 50 \times 10$ and $100 \times 100 \times 10$, respectively. The detailed parameters are given in Settings 2 and 3 in Table 2. In each scale of the environment, the flight mission of UAV is to travel from the position $[0, 0, 0]$ to the destinations $[49, 49, 0]$ and $[99, 99, 0]$, respectively, within the budget of accuracy, time, and energy, as well as no safety and privacy risks.

From our simulation results, in Setting 2, the flight mission is completed in 60.07 s, consuming 106.27 units of energy without any risk of safety or privacy. In Setting 3, the mission is finished in 101.5 s, consuming 210.94 units of energy. There are no safety risks but four positions along the trajectory where the soft constraint of privacy-preserving is violated. In contrast, as the feasible domain of UAV behaviors increases along with the increase in working space, AMOCS-MA fails to compute adaptive behaviors at runtime when the workspace's size increases. One of the reasons for `Captain`'s scalability is that the two optimization problems are solved by SQP that can handle large-scale optimization problems, and the flexibility of `Captain` as we illustrated in Sect. 5. Videos of the simulations can be found on the website[3].

**Answer to RQ3**: `Captain` is scalable and can be generalized into different environment scales.

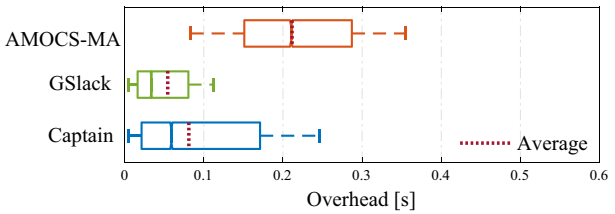
## 6.5 RQ4: real-time performance

To answer RQ4, we analyze the real-time performance of the online requirements-driven adaptation process in the UAV Delivery and UUV Ocean Surveillance scenarios. Figure 9 shows the statistical results of the computation overhead for 10000 executions of each approach.

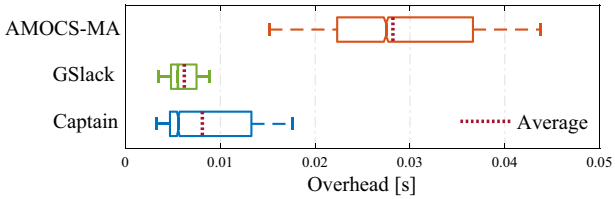
### 6.5.1 UAV Delivery scenario

From Fig. 9a, we find that `Captain` outperforms AMOCS-MA in terms of the average overhead in this scenario, i.e., the gap in average computation time between AMOCS-MA and our approach is about 130 ms (0.212 s versus 0.08 s) This gap is because AMOCS-MA optimizes the satisfaction of all soft constraints. By contrast, in `Captain`, to reduce the adaptation search space, only soft constraints in $\mathcal{R}_f$ need optimization while other requirements are still kept as constraints. So AMOCS-MA requires more computational resources, especially when there are many requirements in the form of soft constraints to handle simultaneously.

Compared with GSlack, `Captain` requires more time to solve Eq. 7. There is a trade-off between optimality in requirements satisfaction and computation time: GSlack is faster, but its solution is not optimal, while `Captain` needs more time to generate the solution to improve the




**(a)** UAV Delivery ($\rho_o = 2\%, \rho_p = 2\%$)




**(b)** UUV Ocean Surveillance (# of incidents = 9)

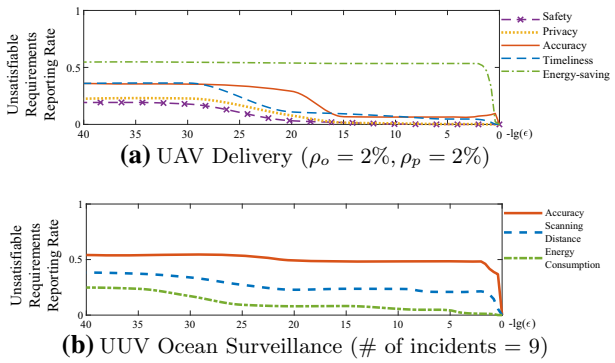
**Fig. 9** RQ4—Empirical distribution of the computation time for the UAV and UUV cases

requirements satisfaction. Thanks to the setting of violation tolerances, the gap in average computation time between GSlack and `Captain` is only 27 ms in the UAV case. As the soft constraints with a slight violation are filtered, this helps determine the appropriate $\boldsymbol{\epsilon}$-*flexible* set of requirements to scale down the adaptation space for optimization. In the worst case, when all the soft constraints need to be accommodated, the computation time of `Captain` would exceed 200 ms in the UAV case, but from our experiment results, we see this case is very rare.

Given the violation tolerances, the average rates of soft constraints violation reporting are recorded per simulation, as shown in Fig. 10a. We choose $\epsilon_{S,P,\varphi,\xi,E} = \{10^{-20}, 10^{-20}, 10^{-10}, 10^{-20}, 0.005\}$ for the UAV case, as the average rates of the reporting of soft constraints violations are not so frequent, at which 16.0%, 14.7%, 29.0%, 25.5%, and 1.1% for violations of safety, privacy, timeliness, accuracy, and energy-saving, respectively, are filtered. This choice shows that right violation tolerances can help determine the appropriate set of unsatisfied requirements with a smaller impact on the algorithm's efficiency. Considering that the choice of tolerance parameter may affect the convergence in the *Requirements Satisfaction Optimization* step and result in no feasible solutions, `Captain` can also directly leverage the planning results from the *Requirements Satisfaction Checking* step, as illustrated in Algorithm 1.

### 6.5.2 UUV oceanic surveillance scenario

Figure 9b shows the empirical distribution of the computation time for 10000 executions of each approach, where the red dotted lines represent the average computation time. We find that `Captain` outperforms AMOCS-MA in terms of the




**(a)** UAV Delivery ($\rho_o = 2\%, \rho_p = 2\%$)

**(b)** UUV Ocean Surveillance (# of incidents = 9)

**Fig. 10** RQ4—Unsatisfied requirements reporting rate for the UAV and UUV cases

average overhead, i.e., 20 ms (0.028 versus 0.008 s) in this scenario. Thanks to the setting of violation tolerances, the gap in average computation time between GSlack and `Captain` is 2 ms in the UUV case. We can also find the gaps of overhead between approaches are minimized, and the average overhead of `Captain` in the UUV case is smaller than that in the UAV case. This is because the number of requirements handled by each AUS is different, i.e., there are three types of soft constraints and hard constraints in the UUV case, while there are five types of soft constraints and hard constraints in the UAV case. As the number of requirements increases, the advantage of `Captain` in real-time performance is more obvious against AMOCS-MA, considering the exponential increase of search space for the optimization problem. Given the violation tolerances, the average rates of soft constraints violation reporting are recorded per simulation, as shown in Fig. 10b. In the UUV case, we choose $\epsilon_{\varphi,e,l} = \{10^{-3}, 0, 0\}$, as the average rates of violation reporting are not so frequent.

**Answer to RQ4**: `Captain` incurs acceptable computation overhead and is suitable for multiple requirements adaptation at runtime.

## 6.6 Threats to validity

The experiments show that our model-driven and control-based adaptation approach has better performance in the requirements satisfaction of AUSs. We are aware of threats to the validity of our evaluation of the proposed approach and discuss them below. There are three types of threats to validity in evaluation.

**Construct validity** We specify and evaluate five representative requirements for the UAV Delivery system as listed in Table 1. Requirements handled by `Captain` are required to be assessed or quantified, which may not be easy for all attributes [51].

**Internal validity** The main threat to internal validity is that the division of $\mathcal{R}_f$ and $\mathcal{R}_{nf}$ depends on the setting of violation tolerances. The values of $\boldsymbol{\epsilon}$ cannot be too low; otherwise, it would result in a large number of soft constraints in $\mathcal{R}_f$ and sacrifice the real-time performance and optimality when solving Eq. 7. The values cannot be too high either; otherwise, it would filter more soft constraints into $\mathcal{R}_{nf}$, which are treated as satisfiable and kept as constraints in Eq. 7. This type of inappropriate partition of $\mathcal{R}_f$ and $\mathcal{R}_{nf}$ may result in unsolvable for the step of *Requirements Satisfaction Optimizing*. In experiments, we empirically determine the proper values through trial-and-error (see Sect. 6.5).

Considering the bias in weights for requirements in AMOCS-MA and GSlack, for a fair comparison with `Captain`, we choose equal weights for requirements in the baseline approaches (i.e., AMOCS-MA and GSlack), and use the sum of the satisfaction degree of all soft constraints as the objective function in AMOCS-MA. Thus, the baseline approaches (i.e., AMOCS-MA and GSlack) are implemented by ourselves. The re-implementation of baseline approaches may lead to faults in the prototypes, which might bias our evaluation results. To reduce this threat, we have iteratively inspected the intermediate results and revised codes when implementing AMOCS-MA and GSlack.

**External validity** Currently, `Captain` is realized with a UAV Delivery simulation and deployed on UAV for real flights, which poses a threat in terms of the generality of the proposed approach. We mitigate this threat by extending `Captain` in a UUV Ocean Surveillance scenario [15]. In the simulation, `Captain` is evaluated with two kinds of runtime uncertainties, i.e., the changes in environmental conditions, variations in system characteristics of sensor parameters. In the future, other kinds of runtime uncertainties (e.g., weather, human in-the-loop [19]) will be introduced to evaluate `Captain` in more complex scenarios and different types of AUSs.

## 6.7 Limitations

With consideration to execution time and nonlinear constraints in the two optimization problems, i.e., Eqs. 6 and 7, the optimization method SQP is adopted. However, there is a risk that the solution falls into a local optimum, as planning results may be affected by different initial values. To mitigate this risk, we can adopt the motion planner of Baidu Apollo [52] which applies a more complicated planning method with a combination of dynamic programming and spline-based quadratic programming. Additionally, heuristic algorithms for a global optimum in multi-dimensional space like genetic algorithms [53] may also be helpful.

As some functionalities and attributes of AUSs may not change drastically (e.g., timeliness), they are not necessary

to be checked as frequently as the others (e.g., safety). To simplify the *Requirements Satisfaction Checking* process, we set the same prediction horizon $N$ for each requirement. In the future, we plan to adopt a two-layer hierarchical MPC with different prediction horizons [54] to mitigate this concern.

## 7 Related work

Our work is related to the following areas: (1) requirements monitoring, (2) requirements runtime management, (3) models at runtime, (4) goal-driven adaption approaches, and (5) control-based adaption approaches.

### 7.1 Requirement monitoring

Techniques in requirement monitoring are necessary for software systems to detect requirement deviations at runtime [55–57]. Model-based approaches can be applied to formalize requirements such that requirements can be checked at runtime [58,59], e.g., *Athena* leverages utility functions to measure the satisfaction of requirements based on the goal model [60]. Alternatively, through analyzing traces of system execution and data logs, violations of assumptions and constraints can be detected, e.g., *ACon* uses data mining to detect inconsistent requirements due to runtime uncertainty [61]. *REMINDS* monitoring framework is proposed to check event-based constraints for cyber-physical systems online [62]. However, these existing requirement monitoring approaches focus mainly on detecting violations of expected behaviors, while the diagnosis of root causes of requirements violations and countermeasures are limited [55]. In `Captain`, conditions lead to requirements violation are identified proactively through *Requirements Satisfaction Checking*. Thus, the AUS is aware of when and which requirements are likely to be violated through *Requirements Violation Analysis*, and is able to find appropriate solutions in *Requirements Satisfaction Optimization*.

### 7.2 Requirements runtime management

To cope with uncertain environments while providing functionality efficiently and dependably, software systems need to make autonomous decisions in requirements management so that requirements can be easily added, removed, and changed. In runtime requirements management, software systems continuously evaluate requirements achievement and autonomously perform adaptations for re-achievement online. The requirements management mechanism is first proposed in a three-layer self-managed software architecture [63] and refined in the MORPH [64]. Based on the modular and generic goal model, automatic quantitative goal evaluation and management is available from [65]. Such methodology for requirements management at runtime usually follows user-defined preference or hand-specified rules to adjust requirements in real-time, which may result in sub-optimal. The runtime requirements management in `Captain` is realized in a way that unsatisfiable soft constraints are identified to accommodate such that runtime uncertainties can be mitigated.

Methods like SimCA [15], SimCA* [16] also utilizes control theory to realize the dynamic requirements management process. Through transforming requirements into *STO-reqs* in the Goal Transform step, requirements are described as setpoints or thresholds for parameters or the optimization goal in the control methods. Compared with these approaches, `Captain` is more flexible in handling multiple requirements, as the unsatisfiable soft constraints can be dynamically transformed from setpoints into objectives to optimize at runtime.

### 7.3 Models at runtime

Researches on models at runtime aim to provide abstractions of useful information about the software system, and models serve as the driver and enabler for automatic reasoning and planning during operation [10]. These researches extend the applicability of model-driven engineering techniques to the runtime environment [66,67]. Approaches to reason and plan based on runtime models can be generally classified into two types [68]: rule-based and search-based. In rule-based approaches, the reasoning and planning process is specified by the form of event-condition-action rules or policies [69,70]. Search-based approaches depend on utility functions to find the optimal plan that fulfills the requirements [3,71]. However, these two types of approaches suffer either from scalability issues as the number of rules to be managed and validated increases or from the costly reasoning and planning processes [72]. Based on an MPC strategy, we support requirements satisfaction analyzing and optimizing in a timely fashion, saving the effort spent on reasoning.

### 7.4 Goal-driven adaptation

The development of self-adaptive systems results from uncertainties at runtime, e.g., disturbances in the execution environment, system parameter fluctuations, requirement changes, and human-related uncertainty (beyond this paper) [19]. In goal-driven adaptation, goal models are associated with system design [24]. To express requirement uncertainties, RELAX is proposed as a textual language [20] incorporated into goal models [21] for self-adaptive systems. AutoRELAX [23] targets a relaxed goal model with the least number of relaxed goals and the minimal adaptation cost online. A similar approach introduces FLAGS [22], which allows requirements to be partially satisfied to mitigate uncer-

**Table 6** Comparison with automated control-theoretical software adaptation approaches

| Approach | Multi-requirements handling | | | Sources of uncertainties | | |
|---|---|---|---|---|---|---|
| | $\mathcal{S}$ | $\mathcal{H}$ | Objective function | Requirement | System | Environment |
| AMOCS | $n$ | $n$ | Settled (combined by priority) | / | / | / |
| SimCA | 1 | $n$ | Settled (single) | ✓ | ✓ | / |
| AMOCS-MA | $n$ | $n$ | Settled (combined by priority) | ✓ | / | ✓ |
| SimCA* | 1 | $n$ | Settled (single) | ✓ | ✓ | / |
| `Captain` | $n$ | $n$ | Flexible (determined by case) | ✓ | ✓ | ✓ |

[a] $n$ means multiple requirements of certain type are handled by the approach, while 1 means only one requirements are handled.
[b] settled means the formulation of the objective function to optimize is fixed, while flexible means the objective function adapts to the uncertainties

tainties and extend the initial goal model. These approaches use goal models as the knowledge base of adaptation which presumes a predefined adaptation space. The selection and adjustment of goals-to-adapt are made based on specified rules and preferences. While `Captain` supports both invariant ($\mathcal{R}_{\mathcal{M}}$) and non-invariant ($\mathcal{R}_{\mathcal{A}}$) requirements satisfaction evaluation, and the relaxation of requirements are enforced only when they are predicted as unsatisfiable. Therefore, alternatives are automatically generated in response to unexpected conditions in a real-time performance.

### 7.5 Control-based adaptation

In the wave of control-based adaptation, strategies for guaranteed self-adaptation are widely studied and summarized by Shevtsov et al. [11]. So far, most research on control-based adaptation has focused on ad-hoc solutions to control the lower-level elements and resources of the system (e.g., CPU, storage, bandwidth, etc.) [13]. These solutions require a well understanding of mathematical system models and are often done on a per-problem basis, discouraging flexibility and generality. Thus, general and automated methods for robust control are proposed like AMOCS [14], SimCA [15], SimCA* [16] and AMOCS-MA [17], etc.

In AMOCS [14], multiple requirements are satisfied through cascaded controllers, as requirements are prioritized for a chain of push-button methodology controllers to produce adaptation decisions. Thus, the requirements are prioritized based on their position in the chain. This may result in sub-optimal solutions [16]. SimCA [15] and its variants [16,73] provide formal guarantees to behavior adaptation of software systems but cannot handle conflicts when handling multiple requirements. Moreover, they do not support uncertainties in the operating environment which are common for AUSs.

Recently, researchers investigated the use of model predictive control (MPC) in control-theoretic software adaptation to eliminate these limitations when handling multiple requirements. MPC is a technique based on the optimization of a cost function according to the prediction of future outcomes, which is considered as an effective and flexible solution for multi-objective problems with optimization [11]. At first, control-based requirements-oriented adaptation (CobRA) [74,75] provides a framework to apply MPC to computing systems, and the model in CobRA has to be generated manually and fed to the system. Similarly, the proactive latency-aware (PLA) approach [76,77] uses models of the environment and the software to determine the best strategy to be followed using a model checker with the ability to look into the future expectations for the system. CobRA suits more for continuous inputs, while PLA works better with discrete control [78]. Following these semi-automated approaches, a fully automated model predictive control strategy was developed, i.e., Automated Multi-objective Control of Software with Multiple Actuators (AMOCS-MA) [17]. By minimizing a predefined cost function in the form of weighted sums, trade-offs are made between different requirements. However, it is susceptible to biases [18] as the weights predefined by human experts are usually subjective. Additionally, the efficiency and effectiveness of many-objective optimization decrease as the number of requirements increases [51].

In contrast, `Captain` can handle multiple requirements, including optimizable soft constraints, missions, and hard constraints that are must be assured. The form of the objective function to optimize is determined by specific runtime situations. In other words, we detect unsatisfiable requirements in the upcoming future and transform them into objective functions to optimize. Runtime uncertainties, including environment changes (e.g., unknown obstacles) and system parameters disturbances (e.g., component failure) are considered in `Captain`. Finally, the key properties of the state-of-the-art control-based adaptation methods are summarized in Table 6. As shown in the table, we compare the number of soft constraints $\mathcal{S}$ and hard constraints $\mathcal{H}$ and the formulation of objective functions that can be handled by each approach.

## 8 Conclusions

In this paper, we present `Captain`, a model-driven online adaptation approach based on control theory. It is designed for the optimal requirements satisfaction of AUSs in response to runtime uncertainties of environmental changes and system characteristics fluctuations. Driven by the manually designed requirements satisfaction model, `Captain` can detect unsatisfiable requirements proactively, analyze their violation degree and optimize their satisfaction through an autonomous adaptation process. Besides, `Captain` supports the availability of tuning sensor configuration and physical motions simultaneously in the online adaptation process of AUSs. The approach is evaluated in two AUS cases with simulations and the DJI Matrice 100 UAV in real scenarios. Experiments show that our approach outperforms other control-based adaptation approaches in the robustness of requirements satisfaction, scalability, and overhead.

Currently, the prediction of unsatisfiable requirements is based on the comparison between computed violation degree and thresholds of violation tolerance. In the near future, we will investigate approaches to improve the efficiency of requirements violation identification and reduce the failure rate at the *Requirements Satisfaction Optimization* step. Our long-term goal is to extend our approach to handle more complex scenarios and different types of autonomous self-adaptive systems with various requirements.

**Acknowledgements** The work is supported in part by the National Natural Science Foundation of China under Grant Nos. 62192731 and 61751210, UK EPSRC (SAUSE), and EU H2020 Engage KTN on Drone Identity.

## Appendix

In this part, we illustrate the detailed requirements satisfaction functions we used in the UAV Delivery scenario and UUV surveillance scenario.

### UAV Delivery scenario

- Safety: The indicator to evaluate the safety requirement is the collision risk of UAV during the flight. Supposing that the obstacles detected by the UAV at time instant $k$ is $\mathcal{O}_k$, while the current state of UAV is $s_k$. Thus, the QM of safety is $\mathcal{X}_{S_o}(k) = \frac{\|\boldsymbol{x}_k - \boldsymbol{x}_o\|_2 - r_a - r_o}{D_o}, \forall o \in \mathcal{O}_k$. Such that the average distance between UAV and the center of obstacle reflects the safety risk.

$$\mathrm{DS}^2(\mathcal{X}_{S_o}(k)) = \begin{cases} 1, & \mathcal{X}_{S_o}(k) \geq 1 \\ 0, & \mathcal{X}_{S_o}(k) < 0 \\ \mathcal{X}_{S_o}(k), & \text{otherwise} \end{cases}$$

- Timelines: The total traveling time from time instant $i$ to $j$ is denoted as $\xi_{ij} = \sum_{k=i}^{j-1} \frac{\|\boldsymbol{x}_{k+1} - \boldsymbol{x}_k\|_2}{\boldsymbol{v}_k}$. The indicator of timeliness is $\mathcal{X}_\xi(T) = \xi_{0T}$, the degree of satisfaction of the timeliness requirement of the whole trajectory is:

$$\mathrm{DS}^1(\mathcal{X}_\xi(T)) = \begin{cases} 1, & \mathcal{X}_\xi(T) \leq \Delta_t \\ \dfrac{\Delta - \mathcal{X}_\xi(T)}{\Delta - \Delta_t}, & \Delta_t < \mathcal{X}_\xi(T) \leq \Delta \\ 0, & \mathcal{X}_\xi(T) > \Delta \end{cases}$$

- Accuracy: The average quality of the information collected during the mission is denoted as $\mathcal{X}_\varphi(T) = \frac{1}{\xi_{0T}} \sum_{k=0}^{T-1} \|\boldsymbol{\omega}\| \tau$, the degree of satisfaction is $DS_\varphi$ is:

$$\mathrm{DS}^2(\mathcal{X}_\varphi(T)) = \begin{cases} 1, & \mathcal{X}_\varphi(T) \geq A_t \\ \dfrac{\mathcal{X}_\varphi(T) - A}{A_t - A}, & A \leq \mathcal{X}_\varphi(T) < A_t \\ 0, & \mathcal{X}_\varphi(T) < A \end{cases}$$

- Energy-saving: The total energy consumption from time instant $i$ to $j$ is denoted as $e_{ij} = \sum_{k=i}^{j-1} \|\boldsymbol{x}_{k+1} - \boldsymbol{x}_k\|_2 + \eta_1 \cdot \|\boldsymbol{v}_{k+1} - \boldsymbol{v}_k\|_2 + \eta_2 \cdot \|\boldsymbol{\omega}_k\| \tau$. The indicator of energy consumption is $\mathcal{X}_E(T) = e_{0T}$, the degree of satisfaction of energy requirement $DS_e$ is:

$$\mathrm{DS}^1(\mathcal{X}_E(T)) = \begin{cases} 1, & \mathcal{X}_E(T) \leq E_t \\ \dfrac{E - \mathcal{X}_E(T)}{E - E_t}, & E_t < \mathcal{X}_E(T) \leq E \\ 0, & \mathcal{X}_E(T) > E \end{cases}$$

### UUV surveillance scenario

The UUV is equipped with 5 sensors for ocean surveillance. The scanning time 10 hours is 360 time instance, $x_i, i \in [1, 5]$ is the portion of time the sensor $i$ should be used during system operation in each instance. $Acc_i$ is the accuracy of sensor $i$; $E_i$ is the energy consumed by sensor; $V_i$ is the scanning speed of sensor. $q_i$ is portion of accuracy of sensor and $p_i$ is for scanning speed, respectively, in decimals. The energy consumed is related with working accuracy and speed of sensor. The corresponding measures are listed as follows: $\mathcal{X}_L(T) = \sum_{k=0}^{T} \sum_{i=0}^{N} x_i q_i V_i \tau$, $\mathcal{X}_E(T) = \sum_{k=0}^{T} \sum_{i=0}^{N} x_i E_i \cdot \frac{e^{p_i + q_i} - 1}{e^2 - 1} \tau$, and $\mathcal{X}_\varphi(T) = \sum_{k=0}^{T} \sum_{i=0}^{N} x_i p_i Acc_i$, where $T = 360$, i.e., adaptations are performed every 100 surface measurements of the UUV state, and the time instance $k$ incremented by $1 \sim 100$. The requirements satisfaction functions are listed as follows:

- Scanning distance: A segment of surface over a distance of $L_t = 100$ km is expected to be examined by the UUV

within $\Delta = 10$ hours, while the distance threshold is $L = 90$ km.

$$\mathrm{DS}^2(\mathcal{X}_L(T)) = \begin{cases} 1, & \mathcal{X}_L(T) \geq L_t \\ \dfrac{\mathcal{X}_L(T) - L}{L_t - L}, & L \leq \mathcal{X}_L(T) < L_t \\ 0, & \mathcal{X}_L(T) < L \end{cases}$$

- Energy-saving: A total amount of energy $E_t = 5.4$ MJ is expected to be consumed, while the maximum amount of energy is $E = 6$ MJ.

$$\mathrm{DS}^1(\mathcal{X}_E(T)) = \begin{cases} 1, & \mathcal{X}_E(T) \leq E_t \\ \dfrac{E - \mathcal{X}_E(T)}{E - E_t}, & E_t < \mathcal{X}_E(T) \leq E \\ 0, & \mathcal{X}_E(T) > E \end{cases}$$

- Accuracy: The accuracy of sensor measurements is targeted at $A_t = 90\%$, while the accuracy threshold is set as $A = 80\%$.

$$\mathrm{DS}^2(\mathcal{X}_\varphi(T)) = \begin{cases} 1, & \mathcal{X}_\varphi(T) \geq A_t \\ \dfrac{\mathcal{X}_\varphi(T) - A}{A_t - A}, & A \leq \mathcal{X}_\varphi(T) < A_t \\ 0, & \mathcal{X}_\varphi(T) < A \end{cases}$$

## References


1. Erdelj, M., Natalizio, E., Chowdhury, K.R., Akyildiz, I.F.: Help from the sky: leveraging uavs for disaster management. IEEE Pervasive Comput. **16**(1), 24–32 (2017)
2. Aldrich, J., Garlan, D., Kästner, C., Goues, C.L., Mohseni-Kabir, A., Ruchkin, I., Samuel, S., Schmerl, B.R., Timperley, C.S., Veloso, M., Voysey, I., Biswas, J., Guha, A., Holtz, J., Cámara, J., Jamshidi, P.: Model-based adaptation for robotics software. IEEE Softw. **36**(2), 83–90 (2019)
3. Jamshidi, P., Cámara, J., Schmerl, B. R., Kästner, C., Garlan, D.: Machine learning meets quantitative planning: enabling self-adaptation in autonomous robots. In: Proceedings of the 14th IEEE/ACM international symposium on software engineering for adaptive and self-managing systems (SEAMS). pp 39–50, (2019)
4. Leveson, N.G.: Engineering a safer world: systems thinking applied to safety. The MIT Press, Cambridge (2016)
5. Luo, Y., Yu, Y., Jin, Z., Zhao, H.: Environment-centric safety requirements for autonomous unmanned systems. In: Proceedings of the 27th IEEE international requirements engineering conference (RE). pages 410–415 (2019)
6. Dalpiaz, F., Niu, N.: Requirements engineering in the days of artificial intelligence. IEEE Softw. **37**(4), 7–10 (2020)
7. Jin, Z.: Environment Modeling based Requirements Engineering for Software Intensive Systems. Elsevier, Morgan Kaufmann Publisher (2018)
8. Zhang, M., Ali, S., Yue, T.: Uncertainty-wise test case generation and minimization for cyber-physical systems. J. Syst. Softw. **153**, 1–21 (2019)
9. Kim, H., Ben-Othman, J., Mokdad, L.: UDiPP: a framework for differential privacy preserving movements of unmanned aerial vehicles in smart cities. IEEE Trans. Veh. Technol. **68**(4), 3933–3943 (2019)
10. Giese, H., Bencomo, N., Pasquale, L., Ramirez, A.J., Inverardi, P., Wätzoldt, S., Clarke, S.: Living with uncertainty in the age of runtime models. In: Models@run.time: foundations, applications, and roadmaps., volume 8378 of Lecture Notes in Computer Science. pp 47–100 (2011)
11. Shevtsov, S., Berekmeri, M., Weyns, D., Maggio, M.: Control-theoretical software adaptation: a systematic literature review. IEEE Trans. Software Eng. **44**(8), 784–810 (2017)
12. Shevtsov, S., Weyns, D., Maggio, M.: Self-adaptation of software using automatically generated control-theoretical solutions. In: Engineering adaptive software systems. pp 35–55. Springer (2019)
13. Klein, C, Maggio, M, Årzén, K.-E., Hernández-Rodriguez, F.: Brownout: building more robust cloud applications. In: Proceedings of the 36th international conference on software engineering (ICSE). pp 700–711 (2014)
14. Filieri, A., Hoffmann, H., Maggio, M.: Automated multi-objective control for self-adaptive software design. In: Proceedings of the 10th joint meeting on foundations of software engineering (ESEC/FSE). pp 13–24, (2015)
15. Shevtsov, S., Weyns, D.: Keep it SIMPLEX: satisfying multiple goals with guarantees in control-based self-adaptive systems. In: Proceedings of the 24th international symposium on foundations of software engineering (ESEC/FSE). pp 229–241, (2016)
16. Shevtsov, S., Weyns, D., Maggio, M.: SimCA*: a control-theoretic approach to handle uncertainty in self-adaptive systems with guarantees. ACM Trans. Autonom. Adapt. Syste. **13**(4), 17 (2019)
17. Maggio, M., Papadopoulos, A. V., Filieri, A., Hoffmann, H.: Automated control of multiple software goals using multiple actuators. In Proceedings of the 11th joint meeting on foundations of software engineering (ESEC/FSE), pp 373–384, (2017)
18. Edwards, R., Bencomo, N.: DeSiRE: further understanding nuances of degrees of satisfaction of non-functional requirements trade-off. In Proceedings of the 13th IEEE/ACM international symposium on software engineering for adaptive and self-managing systems (SEAMS), pages 12–18, (2018)
19. Weyns, D.: Software engineering of self-adaptive systems. In: Handbook of software engineering. pp 399–443. Springer (2019)
20. Whittle, J., Sawyer, P., Bencomo, N., Cheng, B.H., Bruel, J.-M.: RELAX: a language to address uncertainty in self-adaptive systems requirement. Requir. Eng. **15**(2), 177–196 (2010)
21. Whittle, J., Sawyer, P., Bencomo, N., Cheng, B.H., Bruel, J.-M.: Relax: Incorporating uncertainty into the specification of self-adaptive systems. In: Proceedings of the 17th IEEE international requirements engineering conference (RE). pp 79–88, (2009)
22. Baresi, L., Pasquale, L., Spoletini, P.: Fuzzy goals for requirements-driven adaptation. In Proceedings of the 18th IEEE international requirements engineering conference (RE). pp 125–134, (2010)
23. Fredericks, E.M., DeVries, B., Cheng, B.H.C.: AutoRELAX: automatically relaxing a goal model to address uncertainty. Emp. Softw. Eng. **19**(5), 1466–1501 (2014)
24. Solano, G.F., Caldas, R.D., Rodrigues, G.N., Vogel, T., Pelliccione, P.: Taming uncertainty in the assurance process of self-adaptive systems: a goal-oriented approach. In: Proceedings of the 14th IEEE/ACM international symposium on software engineering for adaptive and self-managing systems (SEAMS), pages 89–99 (2019)
25. Alrajeh, D., Cailliau, A., van Lamsweerde, A.: Adapting requirements models to varying environments. In: Proceedings of the 42nd international conference on software engineering (ICSE), pp 50–61 (2020)
26. Frachtenberg, E.: Practical drone delivery. Computer **52**(12), 53–57 (2019)

27. Kimchi, G., Buchmueller, D., Green, S.A., Beckman, B. C, Isaacs, S., Navot, A., Hensel, F., Bar-Zeev, A., Jean-Michel R., Severan S.: Unmanned aerial vehicle delivery system (2017). US Patent 9,573,684
28. ABC NEWS. Amazon's Drone Delivery Idea Launches Funny Tweets, (2013)
29. Chang, V., Chundury, P., Chetty, M.: Spiders in the sky: User perceptions of drones, privacy, and security. In: Proceedings of the 35th acm conference on human factors in computing systems (CHI). pp 6765–6776 (2017)
30. Li, Z., Gao, C., Yue, Q., Fu, X.: Toward drone privacy via regulating altitude and payload. In: Proceedings of the 8th IEEE international conference on computing, networking and communications (ICNC). pp 562–566 (2019)
31. Yel, E., Lin, T. X., Bezzo, N.: Self-triggered adaptive planning and scheduling of uav operations. In: Proceedings of the 35th IEEE international conference on robotics and automation (ICRA). pp 7518–7524 (2018)
32. Luo, Y., Yu, Y., Jin, Z., Li, Y., Ding, Z., Zhou, Y., Liu, Y.: Privacy-aware uav flights through self-configuring motion planning. In: Proceedings of the 37th IEEE international conference on robotics and automation (ICRA). pp 1169–1175 (2020)
33. Maia, P. H., Vieira, L., Chagas, M., Yu, Y., Zisman, A., Nuseibeh, B.: Cautious adaptation of defiant components. In: Proceedings of the 34th IEEE/ACM international conference on automated software engineering (ASE). pages 974–985, (2019)
34. Spong, M.W., Hutchinson, S., Vidyasagar, M.: Robot Modeling and Control. Wiley, NY (2020)
35. Siciliano, B., Sciavicco, L., Villani, L., Oriolo, G.: Robotics: Modelling, Planning and Control. Springer, Berlin (2010)
36. ISO: Robots and robotic devices: safety requirements for industrial robots—Part 1: Robots (2011)
37. Neace, K., Roncace, R., Fomin, P.: Goal model analysis of autonomy requirements for unmanned aircraft systems. Requir. Eng. **23**(4), 509–555 (2018)
38. Singireddy, S. R. R., Daim, T. U.: Technology roadmap: Drone delivery—amazon prime air. In: Infrastructure and technology management. pp 387–412. Springer (2018)
39. Morse, J., Araiza-Illan, D., Eder, K., Lawry, J., Richards, A.: A fuzzy approach to qualification in design exploration for autonomous robots and systems. In: Proceedings of the 26th IEEE international conference on fuzzy systems (FUZZ-IEEE), pp 1–6 (2017)
40. Lemaréchal, C.: Lagrangian relaxation. In: Computational combinatorial optimization. pp12–156. Springer, (2001)
41. Lutz, R., Cleland-Huang, J.: The risk of overly strict requirements. IEEE Softw. **34**(2), 26–29 (2017)
42. Gill, P. E., Wong, E.: Sequential quadratic programming methods. In: Mixed integer nonlinear programming
43. Schittkowski, K.: NLPQL: a fortran subroutine solving constrained nonlinear programming problems. Ann. Oper. Res. **5**(2), 485–500 (1986)
44. Konnik, M., De Doná, J.: Hot-start efficiency of quadratic programming algorithms for fast model predictive control: a comparison via an adaptive optics case study. In: Proceedings of the 4th IEEE Australian control conference (AUCC), pp 95–100, (2014)
45. Corke, P.: Robotics, vision and control: fundamental algorithms in MATLAB® second, completely revised, volume 118. Springer, (2017)
46. Burian, S.J., Velugubantla, S. P., Chittineni, K., Maddula, S. R. K. Brown, M. J.: Morphological analyses using 3d building databases: Portland, oregon. Technical report, Utah. LA-UR, Los Alamos National Laboratory, Los Alamos, NM, (2002)
47. Arcuri, A., Briand, L. C.: A practical guide for using statistical tests to assess randomized algorithms in software engineering. In: Proceedings of the 33rd international conference on software engineering, (ICSE). pp–10, (2011)
48. Capon, J.A.: Elementary Statistics for the Social Sciences: Study Guide. Wadsworth Publishing Company Belmont, CA, USA (1991)
49. Vargha, A., Delaney, H.D.: A critique and improvement of the CL common language effect size statistics of McGraw and Wong. J. Edu. Behav. Stat. **25**(2), 101–132 (2000)
50. Hidalgo-Panagua, A., Vega-Rodríguez, M.A., Ferruz, J., Pavón, N.: Solving the multi-objective path planning problem in mobile robotics with a firefly-based approach. Soft. Comput. **21**(4), 949–964 (2017)
51. Ramírez, A., Raúl Romero, J., Ventura, S.: A survey of many-objective optimisation in search-based software engineering. J. Syst. Softw. **149**, 382–395 (2019)
52. Fan, H., Zhu, F., Liu, C., Zhang, L., Zhuang, L., Li, D., Zhu, W., Hu, J., Li, H., Kong, Q.: Baidu apollo EM motion planner. arXiv:1807.08048 (2018)
53. Liu, Y., Chen, Z., Jiao, W.: A multi-goal oriented approach for adaptation rules generation. In: Proceedings of the 25th IEEE Asia-Pacific software engineering conference (APSEC), pages 249–257, (2018)
54. Picasso, B., De Vito, D., Scattolini, R., Colaneri, P.: An mpc approach to the design of two-layer hierarchical control systems. Automatica **46**(5), 823–831 (2010)
55. Vierhauser, M., Cleland-Huang, J., Rabiser, R., Krismayer, T., Grünbacher, P.: Supporting diagnosis of requirements violations in systems of systems. In: Proceedings of the 26th IEEE international requirements engineering conference (RE)
56. Salifu, M., Yu, Y., Nuseibeh, B.: Specifying monitoring and switching problems in context. In: Proceedings of the 15th IEEE international requirements engineering conference (RE), pp 211–220, (2007)
57. Wang, Y., Mcilraith, S.A., Yu, Y., Mylopoulos, J.: Monitoring and diagnosing software requirements. Autom. Softw. Eng. **16**(1), 3 (2009)
58. Silva S., Vítor E., Lapouchnian, A., Robinson, W. N., Mylopoulos, J.: Awareness requirements for adaptive systems. In: Proceedings of the 6th IEEE/ACM international symposium on software engineering for adaptive and self-managing systems (SEAMS), pp 60–69, (2011)
59. DeVries, B., Cheng, B. H. C.: Using models at run time to detect incomplete and inconsistent requirements. In: Proceedings of the 20th ACM/IEEE international conference on model driven engineering languages and systems (MODELS), pp 201–209 (2017)
60. Ramirez, A. J., Cheng, B. H..: Automatic derivation of utility functions for monitoring software requirements. In: Proceedings of the 14th ACM/IEEE international conference on model driven engineering languages and systems (MODELS). pp 501–506 (2011)
61. Knauss, A., Damian, D., Franch, X., Rook, A., Müller, H.A., Thomo, A.: ACon: a learning-based approach to deal with uncertainty in contextual requirements at runtime. Inf. Softw. Technol. **70**, 85–99 (2016)
62. Vierhauser, M., Cleland-Huang, J., Bayley, S., Krismayer, T., Rabiser, R., Grünbacher, P.: Monitoring CPS at runtime: a case study in the UAV domain. In: Proceedings of the 44th IEEE Euromicro conference on software engineering and advanced applications (SEAA). pp 73–80, (2018)
63. Kramer, J., Magee, J.: Self-managed systems: an architectural challenge. In: Workshop on the future of software engineering (FOSE), pp 259–268, (2007)
64. Braberman, V. A., D'Ippolito, N., Kramer, J., Sykes, D., Uchitel, S.: An extended description of MORPH: A reference architecture for configuration and behaviour self-adaptation. In: Software engineering for self-adaptive systems III., volume 9640 of Lecture Notes in Computer Science, pp 377–408, (2013)

65. Klös, V., Göthel, T., Glesner, S.: Runtime management and quantitative evaluation of changing system goals in complex autonomous systems. J. Syst. Softw. **144**, 314–327 (2018)
66. Blair, G., Bencomo, N., France, R.B.: Models@ run. time. Computer **42**(10), 22–27 (2009)
67. Aßmann, U., Götz, S., Jézéquel, J.-M., Morin, B., Trapp, M.: A reference architecture and roadmap for models@run.time systems. In: Models@run.time, volume 8378 of *Lecture Notes in Computer Science*, pp 1–18, (2011)
68. Fleurey, F., Solberg, A.: A domain specific modeling language supporting specification, simulation and execution of dynamic adaptive systems. In: Proceedings of the 12th ACM/IEEE international conference on model driven engineering languages and systems (MODELS), pp 606–621, (2009)
69. Dubus, J., Merle, P.: Applying OMG d&c specification and ECA rules for autonomous distributed component-based systems. In: Proceedings of the 9th ACM/IEEE international conference on model driven engineering languages and systems (MODELS), pp 242–251, (2006)
70. Fleurey, F., Dehlen, V., Bencomo, N., Morin, B., Jézéquel, J.-M.: Modeling and validating dynamic adaptation. In: Proceedings of the 11th ACM/IEEE international conference on model driven engineering languages and systems (MODELS), pp 97–108, (2008)
71. Cugola, G., Ghezzi, C., Pinto, L.S., Tamburrelli, G.: Selfmotion: a declarative approach for adaptive service-oriented mobile applications. J. Syst. Softw. **92**, 32–44 (2014)
72. Bennaceur, A., France, R., Tamburrelli, G., Vogel, T., Mosterman, P. J., Walter C., Costa, F. M., Alfonso P., Matthias T., Mehmet A., et al. Mechanisms for leveraging models at runtime in self-adaptive software. In: Models@run.time. pp 19–46. (2014)
73. Shevtsov, S., Weyns, D., Maggio, M.: Handling new and changing requirements with guarantees in self-adaptive systems using simca. In: Proceedings of the 12th IEEE/ACM international symposium on software engineering for adaptive and self-managing systems (SEAMS), pp 12–23, (2017)
74. Angelopoulos, K., Papadopoulos, A. V., Vítor, E S., Mylopoulos, J.: Model predictive control for software systems with cobra. In: Proceedings of the 11th IEEE/ACM international symposium on software engineering for adaptive and self-managing systems (SEAMS), pp 35–46, (2016)
75. Angelopoulos, K., Papadopoulos, A.V., Souza, V.E.S., Mylopoulos, J.: Engineering self-adaptive software systems: from requirements to model predictive control. ACM Trans. Autonom. Adapt. Syst. **13**(1), 1–27 (2018)
76. Moreno, G. A., Cámara, J., Garlan, D., Schmerl, B.: Proactive self-adaptation under uncertainty: a probabilistic model checking approach. In: Proceedings of the 10th joint meeting on foundations of software engineering (ESEC/FSE), pp 1–12, (2015)
77. Moreno, G. A, Cámara, J., Garlan, D., Schmerl, B.: Efficient decision-making under uncertainty for proactive self-adaptation. In: Proceedings of the 13th IEEE international conference on autonomic computing (ICAC), pp 147–156, (2016)
78. Moreno, G. A., Papadopoulos, A. V., Angelopoulos, K., Cámara, J., Schmerl, B.: Comparing model-based predictive approaches to self-adaptation: Cobra and pla. In: Proceedings of the 12th IEEE/ACM international symposium on software engineering for adaptive and self-managing systems (SEAMS), pp 42–53, (2017)




**Yixing Luo** received B.Sc. degree in software engineering from University of Electronic Science and Technology of China. She is working towards her Ph.D. in Computer Software and Theory under Prof. Zhi Jin at Peking University. Her research interests include self-adaptive systems, cyber-physical systems, safety modeling, and testing. She was a visiting research student at Nanyang Technological University (NTU).

**Yuan Zhou** received M.S. degree in computational mathematics from Zhejiang Sci-Tech University in 2015 and the PhD degree in computer science from Nanyang Technological University in 2019. He is currently a research fellow with the School of Computer Science and Engineering, Nanyang Technological University. His research interests include multi-robot systems, self-adaptive systems, and autonomous vehicles.

**Haiyan Zhao** received her Ph.D. in information engineering from University of Tokyo in 2003. She is currently an associate professor of computer science at Peking University. Her research interests include requirements engineering, requirements-driven software adaptation, cyber-physical systems, and programming languages. She is member of both ACM and IEEE and senior member of China Computer Federation (CCF).

**Zhi Jin** received the Ph.D. degree in computer science from National University of Defense Technology in 1992. She is currently a full professor of computer science at Peking University. She is deputy director of Key Lab of High Confidence Software Technologies (Peking University), Ministry of Education. Her research interests include requirements Engineering, self-adaptive systems and machine learning. She (co-)authors three books and has more than 200 publications in these areas. She is currently senior member of the IEEE, standing board member of China Computer Federation (CCF), the director of CCF Technical Committee of System Software and was elected to

CCF fellow in 2012. She serves as associated editor of IEEE TSE, ACM TAAS, and IEEE TR and editorial board member of ESEM and RE.

**Tianwei Zhang** received the bachelor's degree from Peking University in 2011, and the Ph.D. degree from Princeton University in 2017. He is an Assistant Professor with the School of Computer Science and Engineering, Nanyang Technological University, Singapore. His research focuses on computer system security. He is particularly interested in security threats and defenses in machine learning systems, auto -nomous systems, computer architecture, and distributed systems.

**Yang Liu** received the Ph.D. degree from National University Singapore in 2010. He joined Nanyang Technological University (NTU) as a Nanyang assistant professor in 2012. He is currently a full professor and director of the Cyber Security Lab with NTU. He is a specialist in software verification, security, and software engineering. His research has bridged the gap between the theory and practical usage of formal methods and program analysis to evaluate the design and implementation of software for high assurance and security. By now, he has more than 300 publications in top tier conferences and journals. He has received a number of prestigious awards including MSRA Fellowship, TRF Fellowship, Nanyang assistant professor, Tan Chin Tuan Fellowship, and eight best paper awards in top conferences like ASE, FSE, and ICSE.

**Danny Barthaud** is a research software engineer in Department of Computing and Communications at the Open University. His current research focuses on cyber-physical systems and methods of gathering and recording data in a forensically sound way.

**Yijun Yu** received his Ph.D. in Computer Science from Fudan University in 1998, worked as a postdoc in Ghent University till 2002, and as a Research Associate and Lecturer in University of Toronto. Since 2006, he is a Senior Lecturer in Computing and Communications at the Open University. His research focuses on improving aviation safety, privacy, and security through distributed ledger technology, which has led to intensive media coverage in the wake of missing flight MH370 (e.g., BBC) and Microsoft and Amazon Research Awards in 2017. His research on Requirements-Driven Self-Adaptation was recognized by a 10 year Most Influential Paper Award from IBM CASCON in 2016 and 8 Best Paper Awards in this area. He has managed knowledge transfer projects with industry partners (IBM, CA, Microsoft, Huawei, UK NATS) and research projects funded by EPSRC, Royal Society, ERC, EU, and the QNRF. He is member of IEEE and AIAA, chair of BCS RESG, and founding member of World Forum on IoT.