A Component-Based Simplex Architecture for High-Assurance Cyber-Physical Systems

Dung Phan*, Junxing Yang*, Matthew Clark†, Radu Grosu‡, John Schierman§, Scott Smolka* and Scott Stoller*
*Department of Computer Science
Stony Brook University, Stony Brook, NY, USA
†Air Force Research Laboratory, Dayton, OH, USA
‡Department of Computer Science
Vienna University of Technology, Vienna, Austria
§Barron Associates Inc., Charlottesville, VA, USA

***Abstract*—We present *Component-Based Simplex Architecture* (CBSA), a new framework for assuring the runtime safety of component-based cyber-physical systems (CPSs). CBSA integrates Assume-Guarantee (A-G) reasoning with the core principles of the Simplex control architecture to allow component-based CPSs to run advanced, uncertified controllers while still providing runtime assurance that A-G contracts and global properties are satisfied. In CBSA, multiple Simplex instances, which can be composed in a nested, serial or parallel manner, coordinate to assure system-wide properties.**

Combining A-G reasoning and the Simplex architecture is a challenging problem that yields significant benefits. By utilizing A-G contracts, we are able to *compositionally* determine the switching logic for CBSAs, thereby alleviating the state explosion encountered by other approaches. Another benefit is that we can use A-G proof rules to decompose the proof of system-wide safety assurance into sub-proofs corresponding to the component-based structure of the system architecture. We also introduce the notion of *coordinated switching* between Simplex instances, a key component of our compositional approach to reasoning about CBSA switching logic.

We illustrate our framework with a component-based control system for a ground rover. We formally prove that the CBSA for this system guarantees *energy safety* (the rover never runs out of power), and *collision freedom* (the rover never collides with a stationary obstacle). We also consider a CBSA for the rover that guarantees *mission completion*: all target destinations visited within a prescribed amount of time.

***Index Terms*—Simplex architecture; Assume-guarantee reasoning; Component-based system architecture; Cyber-physical systems; Collision avoidance**

I. Introduction

Simplex [1]–[3] is a software architecture for high-assurance process-control systems. It traditionally consists of a physical plant and two versions of the controller: an *advanced controller* (AC) and a *baseline controller* (BC). The AC is in control of the plant under nominal operating conditions, and is designed to achieve *high-performance* according to certain metrics (e.g., maneuverability, fuel economy, mission-completion time). The BC is pre-certified to keep the plant within a prescribed *safety region*, i.e., a region of safe operation. A *decision module* (DM), which is also pre-certified, continually monitors the state of the plant and switches control of the plant to the BC should the plant be in imminent danger (i.e., within the next update period) of exiting the safety region. As such, Simplex is a very powerful architecture. It assures that the plant is properly controlled even if the advanced controller has bugs. As advanced controllers are increasingly more complex, more adaptive with the use of unverified algorithms such as machine-learning's, runtime assurance techniques like Simplex are becoming more important. Figure 1 illustrates the Simplex architecture.

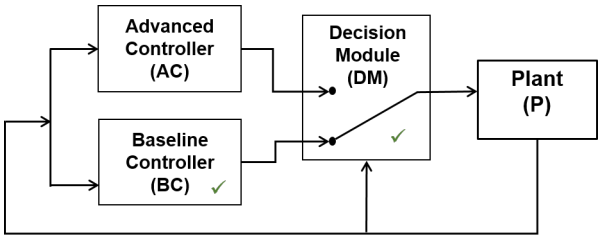

Fig. 1. The Simplex architecture. The Decision Module and Baseline Controller are pre-certified.

The Simplex architecture was developed 20 years ago. Since then, control systems have evolved to take on a more complex structure consisting of multiple controllers, each with a distinct functionality. For example, an autonomous vehicle might have a hierarchical control system consisting of controllers for (in order of decreasing level of abstraction) mission-planning, guidance, navigation, and inner-loop control. A system of this form is illustrated in Fig. 2. Furthermore, some of these controllers themselves may have a hierarchical or nested structure.

Advanced control systems are not necessarily hierarchical, but are better understood as having a *component-based* architecture. A "component" may be a software module or the distinguished physical-plant node (we do not consider multiple physical plants, because a physical plant can be arbitrarily complex). It is important to have runtime assurance techniques for component-based control systems that are equally modular, to reduce their associated complexity and cost.

Components may have different *update periods* (or *execution periods*). For the autonomous-vehicle example, the mission-planning component's update period may be a multiple of navigation's, which in turn may be a multiple of inner-

loop's. The composition of these components is a *multi-rate* component. Our framework allows this multi-rate characteristic to be modeled explicitly.

Adapting the Simplex architecture to component-based systems is non-trivial. On the one hand, wrapping a large monolithic Simplex instance around the entire control system violates modularity principles, making the design and verification of the system more difficult. On the other hand, naively wrapping each component of the system in an independent Simplex container is inadequate to ensure system-wide properties, because of interactions between components.

For example, to guarantee that a ground rover never runs out of power, the mission-planning controller might change the current target to the most recently observed power station when it detects that the battery level is low. It needs to rely on specific properties of other components to ensure that the rover can reach the power station in question before depleting the battery, and these properties might not be assured by the advanced controllers of those components. This example also highlights the need for a compositional proof technique, since a property is assured not by a single Simplex instance, but rather by multiple cooperating instances.

Assume-Guarantee (A-G) reasoning [4]–[6] is a powerful compositional proof technique. With this approach, to prove that a system composed of components M_1 and M_2 satisfies a property q, one shows that M_1 satisfies q under assumptions p and that M_2 satisfies p. Assumption p and guarantee q of M_1 are specified in M_1's *A-G contract*. A-G contracts specify a component's behavior for a single time step, and inductive reasoning about contracts is used to prove system-wide invariants. Furthermore, this approach is applicable to arbitrary safety properties, as a safety property can be expressed as an invariant by adding auxiliary variables [7].

Contributions. In this paper, we present *Component-Based Simplex Architecture* (CBSA), a new framework for assuring the runtime safety of component-based cyber-physical systems. CBSA integrates Assume-Guarantee reasoning with the core principles of the Simplex control architecture to allow component-based control systems to run advanced, uncertified controllers while still being able to provide runtime assurance that A-G contracts, and the global invariants they imply, are satisfied.

A novelty of CBSA is that by utilizing A-G contracts, we are able to *compositionally* determine Simplex switching logic, thereby alleviating the state explosion problem encountered by other approaches. Prior methods [8]–[10] for deriving Simplex switching logic are not designed to be modular, so components would need to be composed into a monolithic system before applying these approaches. In contrast, we compositionally derive Simplex switching logic for a component using assumptions it makes about its environment. These assumptions are then discharged by the guarantees specified in A-G contracts of other components.

Another feature of CBSA is its use of A-G reasoning as a proof technique for runtime assurance. In CBSA, Simplex instances are used to guarantee local, component-based A-G contracts. These contracts specify, for each component, the assumptions it makes about its environment and the guarantees it provides if the environment satisfies those assumptions. A-G reasoning, based on A-G proof rules, is used at compile time to provide runtime assurance of global safety properties from the locally assured A-G contracts.

CBSA also introduces the notion of *coordinated switching* among Simplex-based components. This is needed when the violation of a contract in one component leads to a cascade of contract violations in other components, thereby necessitating synchronized switching to ensure that the desired system properties are not violated.

Another contribution of the paper is a detailed case study that thoroughly illustrates the principles of CBSA. In particular, we show how to apply our framework to a component-based control system for the *Quickbot ground rover* [11]. The Quickbot control system includes a *Mission Planning* component, which selects targets (i.e., destinations), and a *Navigation* component, which steers the rover to its next target destination.

We use A-G reasoning to formally prove that the CBSA for this system guarantees two safety properties: (i) *energy safety* (ES), which ensures the rover never runs out of power, and (ii) *collision freedom* (CF), which ensures the rover never collides with an obstacle. This example illustrates the unique characteristics of CBSA. The switching logic in all Simplex instances is designed to guarantee component-based A-G contracts. Moreover, coordinated switching between the Mission Planning and Navigation components is required to ensure ES.

The ES property illustrates how we use A-G contracts to compositionally construct Simplex switching logic. To ensure energy safety, the rover backtracks to the most recently visited power station when the battery level drops below a threshold. We compositionally derive this threshold for the *Mission Planning*'s switching logic by using a guarantee in the contract of the *Navigation* component. This guarantee assures that the energy the rover needs to backtrack to the most recently visited power station will not exceed the forward energy it has expended to travel from this power station to the current position.

Navigation assures this guarantee by using a Simplex instance that must switch in tandem with *Mission Planning*'s instance. Therefore, we enforce a *coordinated switching* between *Mission Planning* and *Navigation* to ensure the satisfaction of both components' A-G contracts and the global *energy safety* property they imply.

We additionally consider a *Mission Completion* (MC) property for the Quickbot system, i.e., all target destinations are visited within a prescribed amount of time. This property is interesting not only because it is a (bounded) liveness property, a type of property not traditionally handled by Simplex, but also because it is a higher-level mission-oriented property several layers of control removed from the physical plant. The MC property is also more software-oriented in nature, as it

revolves around a software data structure (a list of mission targets).

The rest of the paper is organized as follows. Section II considers related work. Section III formally introduces our CBSA framework. Sections IV and V present the Quickbot case studies. Section VI offers our concluding remarks and directions for future work.

II. Related Work

The relationship of our work with the classical Simplex architecture is discussed in Section I.

Schierman *et al*. developed a runtime assurance technique similar to Simplex, known simply as RTA [12], [13]. RTA can be applied to component-based systems, but each RTA wrapper (i.e., each Simplex-like instance) independently ensures a local safety property of a component. For example, in [12], RTA instances for an inner-loop controller and a guidance system are uncoordinated and thereby operate independently. Their work does not consider A-G contracts or cooperation between components to ensure global properties like we do in this paper.

Schierman *et al*. recently extended their work on RTA by showing how it can be used to ensure that components satisfy A-G contracts [14]. They apply RTA to, and give A-G contracts for, several components in an Unmanned Aerial Vehicle (UAV) flight control system. They do not use formal A-G proof rules to derive global invariants. Instead, they reason about the overall safety of the system using Goal-Structuring Notation (GSN), which is less formal. They discuss the potential need for coordinated switching, but coordinated switching is not used in their case study.

A-G reasoning has been extensively studied for the compositional verification of complex systems (e.g., [4]–[6], [15]–[19]). There is also significant tool support for A-G reasoning, including AGREE [20], OCRA [21], and Safety ADD [22]. AGREE is a framework that supports A-G reasoning on architectural models specified in AADL. OCRA supports the specification of A-G contracts using temporal logic and can also be used to verify the correctness of contract refinements. Safety ADD is a tool for defining A-G contracts and verifying that all guarantee-assumption pairs match and there are no unfulfilled assumptions. A-G reasoning has also been used for compositionally checking source code for preservation of the design's correctness [23].

In [24], compositional barrier functions are used to provably guarantee the simultaneous satisfaction of composed objectives. They rely on a single controller and an optimization based approach to correct the controller when violations of safety are imminent, which has limited capability and less flexibility compared to Simplex.

None of these approaches, however, consider the possibility of pairing A-G reasoning with techniques for runtime assurance of system properties. Our approach integrates A-G reasoning with the core principles of Simplex, allowing component-based systems to run advanced, uncertified controllers while still being able to provide runtime assurance that A-G contracts, and the global invariants they imply, are satisfied.

A comparison between our work and the Simplex switching-logic-derivation techniques in [8]–[10] is given in Section I. Without A-G contracts and our concept of coordinated switching, those approaches could be extended as follows to apply to systems with multiple Simplex instances. Suppose there are n Simplex instances, and s is the decision period. A global decision module can choose among 2^n possible controller configurations (baseline or advanced for each Simplex instance). For each of these 2^n configurations, do backward reachability analysis [9] with an unbounded time horizon for the system composed from the selected controllers and the plant, using maximally nondeterministic models for the advanced controllers, to obtain a recoverable region for that configuration. At each decision time, the global decision module checks whether (1) all states reachable in time at most s using the current controller configuration are safe, and (2) all states reachable in time exactly s using the current controller configuration are in the recoverable region of the current controller configuration. If so, the decision module continues to use the current configuration; otherwise, it switches to a controller configuration for which these two conditions hold.

III. Component-Based Simplex Architecture

A. Multi-rate Components

Definition 1. A *multi-rate component* M is a tuple (x, u, y, S), where $x = \{x_1, ..., x_m\}$ is the set of *state variables*, $u = \{u_1, ..., u_k\}$ is the set of *input variables*, $y = \{y_1, ..., y_n\}$ is the set of *output variables*, and $S = \{(f_1, g_1, s_1), ..., (f_l, g_l, s_l)\}$ is a sequence whose i-th element is a triple of a *next-state function* f_i, an *output function* g_i, and their *update period* s_i, which is a positive integer multiplier of the global clock tick dt. The sets of input and output variables are disjoint. The behavior of M at tick i is defined as follows.

$$\begin{cases} x(i) = f_j\left(x(i-1), u(i)\right), \forall j \in [1..l] : i \bmod s_j = 0 \\ y(i) = g_j\left(x(i), u(i)\right), \forall j \in [1..l] : i \bmod s_j = 0 \end{cases}$$

where the set assignment $A = B$ means only the variables in $A \cap B$ are assigned the corresponding values in B. The variables in A that are not updated at tick i retain the values they had at tick $i - 1$.

We give examples of components in Section IV-B.

B. Composition with Feedback

For simplicity, we make the following assumptions about multi-rate components.

1) Components communicate via shared variables. After a component writes a shared variable, the updated value is instantly available to be read by all components.
2) If a component writes and another component reads a shared variable at the same tick, then there must be a predetermined order of execution of the two components. The order of execution of $M_1 \parallel M_2$ is M_1, M_2. This means the composition operator $\parallel$ is associative but not necessarily commutative.

3) The execution time of a component is negligible.
4) A 1-tick delay is introduced in feedback loops to break circularity. A feedback loop exists between two components $M_1 = (x_1, u_1, y_1, S_1)$ and $M_2 = (x_2, u_2, y_2, S_2)$ if $y_1 \cap u_2 \neq \emptyset$ and $y_2 \cap u_1 \neq \emptyset$. If the order of execution is M_1, M_2 then the variables in y_2 that feed back into u_1 are delayed by one tick; i.e., $y_2(i-1)$ is supplied to f_i and g_i of M_1 instead of $y_2(i)$.

Two components $M_1 = (x_1, u_1, y_1, S_1)$ and $M_2 = (x_2, u_2, y_2, S_2)$ are *composable* if $x_1 \cap x_2 = \emptyset$ and $y_1 \cap y_2 = \emptyset$.

Definition 2. Let M_1 and M_2 be composable. Then their *composition* $M_1 \parallel M_2$ is the component (x, u, y, S), where $x = x_1 \cup x_2 \cup (y_1 \cap u_2) \cup (y_2 \cap u_1)$, $u = (u_1 \cup u_2) - (y_1 \cup y_2)$, $y = y_1 \cup y_2$, and $S = S_1 \cup S_2$.

Functions f_i and g_i of $M_1 \parallel M_2$ ignore variables in x and u that are not their original arguments. Note that the connected inputs $(y_1 \cap u_2) \cup (y_2 \cap u_1)$ are hidden from u and added to x. This is necessary to ensure functions f_i, g_i can still access their required variables. The variables in $x \cap y$ are synchronized and updated by functions g_i. Because the order of execution is M_1, M_2, functions f_i and g_i in S_1 are executed before those in S_2.

Remark. Our definition of component is similar to [25] except we have multi-rate. Our form of non-commutative composition can be regarded as a shorthand for a combination of cascade composition and feedback composition in [25]. For example, suppose component M_1 reads variable x and writes variable y, and component M_2 reads y and writes x. To obtain a system equivalent to our composition $M_1 \parallel M_2$ using the combinators in [25], one can first construct a cascade composition of M_1, M_2, and a unit delay state machine M_D that delays x by one tick, and then perform a feedback composition on the resulting component.

C. Assume-Guarantee Contracts

To prove system properties in CBSA, we use A-G reasoning in the form of A-G contracts [26]. That is, with every component M of a CBSA, we associate a contract of the form $\mathcal{C} = (I, O, A, G)$, where I is a set of typed *input* variables, O is a set of typed *output* variables, A is an *assumptions* predicate, and G is a *guarantees* predicate. Note that a contract cannot guarantee a system-wide property involving a (global) variable that does not belong to the contract. A contract should only talk about the assumptions and guarantees of its inputs and outputs. A-G contracts allow *compositional reasoning* about CBSAs using the A-G reasoning rule given in Section III-D. In CBSA, the switching logic for the Simplex instance of a component is based on determining whether the active AC might violate the component's guarantees in the next decision period. Furthermore, the composition of all active controllers must imply the safety properties of the system. This principle results in a requirement that the composition of all BCs in a CBSA must imply the safety properties of the system in case all components must switch to their BCs.

D. Assume-Guarantee Proof Rule

An *assume-guarantee triple* of the form $\langle p \rangle\, M\, [s]\, \langle q \rangle$ means component M guarantees the satisfaction of property q up to and including time $t + s \cdot dt$ under the assumption that property p is satisfied at time t, where s is an update period of M, t is the beginning of the current update period, and dt is the global tick. By induction over time, $\langle true \rangle\, M\, [s]\, \langle p \rangle$ implies M always satisfies p.

The choice of update period s in $\langle p \rangle\, M\, [s]\, \langle q \rangle$ is guided by the property q. If q is given in terms of the variables of a single-rate sub-component of M, then the update period of that sub-component is an appropriate choice for s. If q involves variables from multiple sub-components, then the shortest update period of these sub-components is a good choice for s.

We use the following Assume-Guarantee rule as a formal proof rule for multi-rate systems because it is general enough to handle most cases of interest, including our case studies. The rule we use is *asymmetric* in the sense that only one component makes assumptions about the other component. Other A-G proof rules, including *symmetric* ones (e.g. [27]), can also be used with our framework.

Rule AG.

$$\frac{\begin{array}{c}\langle p \rangle\, M_1\, [s_1]\, \langle q \rangle \\ \langle true \rangle\, M_2\, [s_2]\, \langle p \rangle\end{array}}{\langle true \rangle\, (M_1 \parallel M_2)\, [s_1]\, \langle q \rangle}$$

Rule AG allows one to prove that a system composed of components M_1 and M_2 satisfies a property q by proving that (1) component M_1 guarantees q under assumption p, and (2) component M_2 assures p unconditionally.

E. Coordinated Switching

Coordinated switching is when a switch from AC to BC in one component forces another component to also switch from AC to BC. Suppose components M_1 and M_2 use Simplex to ensure their contracts $\mathcal{C}_1 = (I_1, O_1, A_1, G_1)$ and $\mathcal{C}_2 = (I_2, O_2, A_2, G_2)$, respectively. Assuming $A_2 = true$ and $G_2 \models A_1$, where $\phi \models \varphi$ means ϕ entails φ, we have $M_1 \parallel M_2 \models G_1$ by applying Rule AG. Let AC_i and BC_i be the AC and BC, respectively, of M_i. A typical situation that requires coordinated switching is as follows. Suppose G_2 is the implication $\phi_1 \Rightarrow \phi_2$, and ϕ_1 involves a shared variable that M_1 modifies when switching from AC_1 to BC_1 such that ϕ_1 changes from *false* to *true* as a result. Suppose BC_2 ensures ϕ_2 but AC_2 does not. When M_1 uses AC_1, G_2 is vacuously true. When M_1 uses BC_1, M_2 must use BC_2 to satisfy G_2. Thus, if M_2 is using AC_2 when M_1 switches to BC_1, then M_2 must perform a coordinated switch to BC_2.

Coordinated switching is required in our case study. The *Mission Planning* component has a BC called the recharge controller, which tells the rover to go back to the most recently visited power station to recharge. When *Mission Planning* switches to its BC, it assumes that the *Navigation* component can steer the rover to that power station within a certain energy budget. The BC in the *Navigation* component

guarantees this energy constraint, but the AC does not. Thus, when the *Mission Planning* component switches to its BC, the *Navigation* component must also switch to its BC. In this example, the shared variable in ϕ_1 that communicates the need for the *Navigation* component to switch is the variable $ctlr$, which explicitly indicates which controller is in control of the *Mission Planning* component.

Coordinated switching can be generalized to be between *modes*. A component may have multiple modes with different guarantees. Each mode can use a Simplex instance to assure its guarantees. When a component M_1 switches modes, it may require component M_2 to switch to another mode so that it can use the guarantee that M_2's new mode provides. In our example above, we can view *Mission Planning* as having two modes, named go-to-target and recharge, and *Navigation* as having two modes, named go-to-target and backtrack. When *Mission Planning* switches to recharge mode, it requires *Navigation* to switch to backtrack mode. In our case study, recharge mode and backtrack mode use certified controllers, so we do not need to nest Simplex instances in them.

IV. The Quickbot Case Study

We conducted a detailed case study of our approach based on the Quickbot [11] ground rover developed for the Coursera course Control of Mobile Robots (https://www.coursera.org/course/conrob/). The rover's mission is to visit a sequence of predetermined target locations. We consider a challenging version of the problem: the rover might need to recharge at power stations during the mission, and the rover does not have a map showing the locations of power stations or the locations and shapes of obstacles.

The top-level architecture for the Quickbot case study consists of three single-rate components: *Mission Planning*, *Navigation*, *Inner-Loop* & *Plant*; see Fig. 2. *Mission Planning*'s update period s_{MP} is a multiple of *Navigation*'s s_{Nav}, which in turn is a multiple of *Inner-Loop* & *Plant*'s s_{ILP}. The Quickbot system composed of these components is therefore a multi-rate component. *Mission Planning* sends the next target position T to the *Navigation* component. *Navigation* steers the rover to T while avoiding obstacles. It does this by computing an appropriate target linear velocity vector (v_T) and target angular velocity (ω_T) at each time step. The pair of velocities (v_T, ω_T) is sent to *Inner-Loop* & *Plant*, which computes and actuates the appropriate rotational speeds for each of the two wheels to reach the desired velocities.

The rover is equipped with infrared (IR) sensors, two wheel encoders (which count revolutions of the wheels), a power-station sensor, and a battery-level sensor. *Inner-Loop* & *Plant* converts the raw sensor data into the rover's current position (p), linear velocity vector (v), angular velocity (ω), IR distances (ir) to obstacles, location of the last-detected power station (PS), and battery level (B).

We use *MP*, *Nav* and *ILP* to denote the *Mission Planning*, *Navigation* and *Inner-Loop* & *Plant* components, respectively, and *QB* to denote the entire Quickbot system. *QB* is the parallel composition of *MP*, *Nav*, and *ILP*, i.e., $QB = MP \parallel Nav \parallel ILP$. *ILP* is implicitly the parallel composition of the *Inner-Loop Control* component and the physical plant.

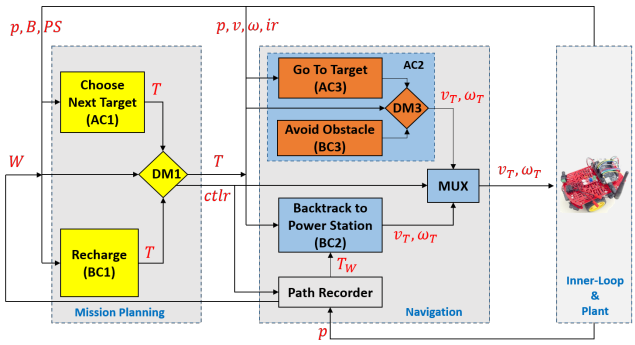

Fig. 2. Component-based Simplex architecture for Quickbot rover.

A. Problem Statement and Assumptions

We design a CBSA whose mission is to visit the specified targets while ensuring the following safety properties.

1) *Energy safety* (ES): the rover never runs out of power.
2) *Collision freedom* (CF): the rover never collides with an obstacle.

We make the following assumptions about the rover.

1) The rover does not have a map of the environment.
2) There are gaps (blind spots) between the fields-of-view of the distance sensors.
3) The distance sensors have a limited sensing range $[0, R_s]$.
4) The rover can stop instantaneously.
5) The linear speed v is bounded by $v \in [0, v_{max}]$.
6) The rotational speed ω is bounded by $\omega \in [-\omega_{max}, \omega_{max}]$.
7) The rover can reach any pair of values (v, ω) instantaneously, provided they are within bounds.
8) The rover starts at a power station.
9) The rover can detect a power station when passing within distance d_{PS} of it.
10) The rover can recharge when it is within distance d_{PS} of a power station.
11) The power consumption of the rover is a monotonically increasing function of the rotational speeds of the two wheels.

We make the following assumptions about the environment.

1) Obstacles are stationary polyhedra.
2) There is a known lower bound on the internal angles between edges of an obstacle.
3) There is a known lower bound on the edge lengths of an obstacle.
4) The separation between obstacles is such that whenever two adjacent sensors simultaneously detect an obstacle, they are detecting the same obstacle.

Assumptions 1-7 about the rover and 1-4 about obstacles are the same as in [28]. This allows us to reuse the Simplex instance in [28] to help ensure the CF property. Let A_P denote the conjunction of the above assumptions about the rover. We assume *ILP* satisfies A_P.

B. Quickbot CBSA

The architecture of the Quickbot CBSA is shown in Fig. 2. A novelty of our case study is that we apply the Simplex architecture to the *Mission Planning* component; Simplex is traditionally applied to lower-level controllers that interact directly with the plant. In our component-based architecture, *Mission Planning* controls a *virtual plant* comprising the *Navigation* component, the *Inner-Loop Control* component, and the physical plant.

Quickbot Components We illustrate Definition 1 with the Quickbot components. The single-rate *Mission Planning* component is $MP = (x_{MP}, u_{MP}, y_{MP}, S_{MP})$, where $x_{MP} = \{T, ctlr\}$, $u_{MP} = \{p, B, PS, W\}$, $y_{MP} = \{T, ctlr\}$, and $S_{MP} = (f_{MP}, g_{MP}, s_{MP})$. Here f_{MP} and g_{MP} are the next-state and output functions representing the algorithm for *MP* described later in this section, and s_{MP} is the update period of *MP*. The behavior of *MP* at global tick i is specified by:

$$\begin{cases} x_{MP}(i) = f_{MP}\left(x_{MP}(i-1), u_{MP}(i)\right) \text{ if } i \bmod s_{MP} = 0 \\ y_{MP}(i) = g_{MP}\left(x_{MP}(i), u_{MP}(i)\right), \text{ if } i \bmod s_{MP} = 0 \end{cases}$$

The single-rate *Navigation* component is $Nav = (x_{Nav}, u_{Nav}, y_{Nav}, S_{Nav})$, where $x_{Nav} = \{v_T, \omega_T, W\}$, $u_{Nav} = \{T, ctlr, p, v, \omega, ir\}$, $y_{Nav} = \{v_T, \omega_T, W\}$, and $S_{Nav} = (f_{Nav}, g_{Nav}, s_{Nav})$. Here f_{Nav} and g_{Nav} are the next-state and output functions representing the algorithm for *Nav* described later in this section, and s_{Nav} is the update period of *Nav*. The behavior of *Nav* at global tick i is specified by:

$$\begin{cases} x_{Nav}(i) = f_{Nav}\left(x_{Nav}(i-1), u_{Nav}(i)\right) \text{ if } i \bmod s_{Nav} = 0 \\ y_{Nav}(i) = g_{Nav}\left(x_{Nav}(i), u_{Nav}(i)\right), \text{ if } i \bmod s_{Nav} = 0 \end{cases}$$

The single-rate *Inner-Loop* & *Plant* component is $ILP = (x_{ILP}, u_{ILP}, y_{ILP}, S_{ILP})$, where $x_{ILP} = \{p, v, \omega, ir, PS, B\}$, $u_{ILP} = \{v_T, \omega_T\}$, $y_{ILP} = \{p, v, \omega, ir, PS, B\}$, and $S_{ILP} = (f_{ILP}, g_{ILP}, s_{ILP})$. Here f_{ILP} and g_{ILP} are the next-state and output functions representing the dynamics and sensing of *ILP*, and s_{ILP} is the update period of *ILP*. The behavior of *ILP* at global tick i is specified by:

$$\begin{cases} x_{ILP}(i) = f_{ILP}\left(x_{ILP}(i-1), u_{ILP}(i)\right) \text{ if } i \bmod s_{ILP} = 0 \\ y_{ILP}(i) = g_{ILP}\left(x_{ILP}(i), u_{ILP}(i)\right), \text{ if } i \bmod s_{ILP} = 0 \end{cases}$$

where f_{ILP} and g_{ILP} are the next-state and output functions representing the dynamics and sensing of *ILP*. The order of execution is MP and then Nav (in time steps when they both execute) and then ILP. We assume s_{MP} is a multiple of s_{Nav}, and s_{Nav} is a multiple of s_{ILP}.

Quickbot Controllers Recall that we assume that the rover can detect a power station within distance d_{PS}, for example, by using a camera to read a QR code or by using an RFID reader to read a tag. When the rover passes near a power station, it can determine with IR sensors whether the power station is accessible from its current location; i.e., there are no intervening obstacles. If the power station is accessible, the rover remembers it as the last-visited (i.e., the most recently passed) power station.

The *Mission Planning* AC informs the *Navigation* component of the next target to be visited to fulfill the mission goals, while the BC (the recharge controller) instructs the *Navigation* component to navigate to the most recently detected power station. We say that the rover is in *recharge mode* when DM_1 has switched control to the recharge controller. To guarantee that recharge mode will drive the rover to a power station without depleting the battery, we need a certified backtrack-to-power-station controller in the *Navigation* component. Since the rover lacks a map of the environment, backtracking is the most realistic way to guarantee that the rover can reach a power station without running out of power.

We also briefly considered a variant of the problem in which the rover has a map showing locations of power stations. If the rover does not also have a map of obstacles, backtracking to the last-visited power station might still be necessary, because nearby power stations could be blocked by unknown obstacles that would take too much power to circumnavigate. More complex strategies for the AC in the *Mission Planning* component could help avoid backtracking. For example, instead of going as directly as possible to the targets, it might add waypoints that are slightly out of the way but help the rover determine accessibility of nearby power stations.

Note that there is coordinated switching between the *Mission Planning* component and the *Navigation* component. When *Mission Planning* switches to the recharge controller, *Navigation* must switch to the backtrack controller. The recharge controller in *Mission Planning* assumes that the *Navigation* component can navigate the rover to a power station using at most a specified amount of energy. The go-to-target controller does not provide such guarantees, whereas the backtrack-to-power-station controller (backtrack controller, for short) does. This cascading switch is implemented by hardwiring a decision signal from *Mission Planning*'s DM to *Navigation*'s DM, as shown in Fig. 2.

We introduce a recorder module that records the positions and heading angles (waypoints) at every time step of *Navigation*. When in recharge mode, the backtrack controller tracks these recorded positions in reverse order. We also want the backtrack controller to preserve the *CF* property so that the rover is guaranteed not to collide with any static obstacles on the way back to the last-visited power station. We consider three designs for the backtrack controller.

1) Instead of recording waypoints, we can record the control inputs (v, ω) for *ILP* in every *Navigation* time step. When backtracking, we replay $(v, -\omega)$. This controller will trace the forward path exactly provided that there are no actuation or state estimation errors. Otherwise, the error will accumulate.
2) The rover can backtrack from its current position to a recorded waypoint by first turning in place to point to the target waypoint and then moving in a straight line to it. This approach is robust, but it is slow, consumes more energy, and replaces a curve in the forward trajectory

with a line segment in the backward trajectory. Such a discrepancy could lead to collisions unless the avoid-obstacles controller is invoked.

3) The rover can backtrack from its current position to a recorded waypoint by computing the appropriate values of (v, ω). This means we need to solve the *inverse kinematics problem*, which in general has no analytical solution. We can, however, obtain the least-squares solution, which has several benefits. If there are no errors then the least-squares solution is an exact solution; i.e., this approach falls back to the first one. If there are errors, the errors will be minimized in the least-squares sense and bounded, instead of accumulating. Assuming the errors are small, the trajectory during backtracking is always close to the forward trajectory, and the energy consumed during backtracking is within a bounded margin of the energy consumed going forward. We adopt this approach for the backtrack controller because of these benefits.

The rover can backtrack from its current position to a recorded waypoint by computing the appropriate values of (v, ω). It can do this by solving the *inverse kinematics problem* for the least-squares solution. Suppose the rover is currently at point $A = (x_1, y_1, \theta_1)$ and wants to reach a recorded waypoint $B = (x_2, y_2, \theta_2)$ in one Navigation time step t_{Nav}. The kinematics equation is given below.

$$\begin{cases} x_2 &= x_1 + \int_{t_1}^{t_1+t_{Nav}} v\cos(\theta_1 + \omega t)dt \\ y_2 &= y_1 + \int_{t_1}^{t_1+t_{Nav}} v\sin(\theta_1 + \omega t)dt \\ \theta_2 &= \theta_1 + \omega t_{Nav} \end{cases} \quad (1)$$

The solution in the trivial case where $\theta_1 = \theta_2$ is $\omega = 0$ and $v = AB/t_{Nav}$. If $\theta_1 \neq \theta_2$, Eq. 1 becomes:

$$\begin{cases} x_2 &= x_1 + \frac{v}{\omega}(\sin\theta_2 - \sin\theta_1) \\ y_2 &= y_1 + \frac{v}{\omega}(\cos\theta_1 - \cos\theta_2) \\ \theta_2 &= \theta_1 + \omega t_{Nav} \end{cases} \quad (2)$$

We solve Eq. 2 for the least-squares solution using the standard least-squares method.

We reuse the Simplex instance in [28] to help ensure the CF property. We nest this instance inside the AC of the Simplex instance in *Navigation*. The intuition behind this nested composition of Simplex instances is that the backtrack controller also helps ensure the CF property, because it retraces a collision-free path the rover has already traveled. Therefore, the Simplex instance involving AC_2 and BC_2 in Fig. 2 assures the CF property.

Contracts of Quickbot Components Based on the above analysis, we specify the following contracts. The *Mission Planning* component's contract is:

Inputs: p, PS, B, W
Outputs: $T, ctlr$
Assumption: $ctlr = \text{BC} \Rightarrow$
$BE(p, PS, W) \leq (1 + \epsilon_{BE})FE(PS, p)$
Guarantee: $B > E(p, PS)$

where $BE(p, PS, W)$ is the energy needed to backtrack from the current position p to the most recently visited power station PS going through the sequence of recorded waypoints W; ϵ_{BE} is a constant defined in Section IV-C; $FE(PS, p)$ is the energy expended on the forward path from PS to p; $E(p, PS)$ is the energy needed to go from p to PS, including the amount of energy needed to turn 180° before backtracking. The *Navigation* component's contract is:

Inputs: $T, ctlr, p, ir, v, \omega$
Outputs: v_T, ω_T, W
Assumption: A_P
Guarantee: $(d_o > 0) \wedge (ctlr = \text{BC} \Rightarrow$
$BE(p, PS, W) \leq (1 + \epsilon_{BE})FE(PS, p))$

where d_o is the shortest distance to an obstacle. The *Inner-Loop & Plant*'s contract is:

Inputs: v_T, ω_T
Outputs: p, PS, B, ir, v, ω
Assumption: $true$
Guarantee: A_P

The input and output variables of each component are shown in Fig. 2.

C. Switching Logic for the ES Property

The ES property can be formally expressed as $G(B > 0)$, where G is the *always* (global) operator in linear temporal logic. In our design, this is ensured by having the rover recharge at a power station whenever the battery level is low. So, we enforce the stricter property that the rover always has enough energy to return to the last-visited power station. This property is formalized as $G(B > E(p, PS))$, where $E(p, PS)$ is the amount of energy needed to go from the current position p to the last-visited power station PS, including the amount of energy needed to turn 180° (E_{180}) before going back. Hereafter, we refer to $G(B > E(p, PS))$ as the ES property.

We assume the following constants are known:

1) ϵ_{BE}: a constant such that the backtracking energy *BE* (p, *PS*, W) needed to backtrack from the current position p to the last-visited power station *PS* through a sequence of recorded waypoints W is bounded by $(1+\epsilon_{BE})$ times the forward energy *FE* (*PS*, p) expended since PS was last visited, i.e., $BE(p, PS, W) \leq (1 + \epsilon_{BE})FE(PS, p)$. If there are no errors in actuation or state estimation, we can take $\epsilon_{BE} = 0$.
2) E_{MP}: the worst-case energy expended in one update period of *Mission Planning* by an arbitrary controller.
3) E_{180}: the energy needed to turn in place by 180°.
4) BE_{MP}: a bound on the energy needed to backtrack from the rover's position at the end of the next update period of *Mission Planning* to the current position.

Let E_{MP} be the amount of energy expended when both wheels rotate at maximum rotational speed for one update period of *Mission Planning*. BE_{MP} equals E_{MP}.

The DM in the *Mission Planning* component decides whether to continue to go forward or to return to *PS*. If it decides to go forward, it must ensure that the ES property holds at its next decision point. The worst-case amount of energy needed to travel to a position within one update period

and then backtrack from that position to *PS* is $E_{MP} + E_{180} + BE_{MP} + (1 + \epsilon_{BE})FE(PS, p)$. The switching condition is thus:

$$B \leq E_{MP} + E_{180} + BE_{MP} + (1 + \epsilon_{BE})FE(PS, p) \quad (3)$$

This switching condition is derived compositionally by relying on *Mission Planning*'s assumption, an assumption that is later discharged by composing *Mission Planning* with *Navigation*. We do not need to consider implementation details of other components in the derivation process.

D. Proof Outlines of ES and CF

Lemma 1. For any property ϕ, $\langle A_P \rangle\, Nav\, [s_{Nav}]\, \langle \phi \rangle \Rightarrow \langle true \rangle\, (Nav \parallel ILP)\, [s_{Nav}]\, \langle \phi \rangle$.

Proof. The proof follows from our assumptions and one application of Rule AG.

$$\frac{\langle A_P \rangle\, Nav\, [s_{Nav}]\, \langle \phi \rangle \quad \langle true \rangle\, ILP\, [s_{ILP}]\, \langle A_P \rangle}{\langle true \rangle\, Nav \parallel ILP\, [s_{Nav}]\, \langle \phi \rangle}$$

□

Theorem 1. $\langle true \rangle\, QB\, [s_{MP}]\, \langle ES \rangle$

Proof outline

1) We prove $\langle A_{BE} \rangle\, MP\, [s_{MP}]\, \langle ES \rangle$, where A_{BE} is the assumption in MP's contract given in Section IV-B.
2) We prove $\langle A_P \rangle\, Nav\, [s_{Nav}]\, \langle A_{BE} \rangle$. By Lemma 1, this implies $\langle true \rangle\, (Nav \parallel ILP)\, [s_{Nav}]\, \langle A_{BE} \rangle$.
3) Applying Rule AG, we conclude that $\langle true \rangle\, (MP \parallel Nav \parallel ILP)\, [s_{MP}]\, \langle ES \rangle$.

Theorem 2. $\langle true \rangle\, QB\, [s_{MP}]\, \langle CF \rangle$

Proof outline

1) We prove $\langle A_P \rangle\, Nav\, [s_{Nav}]\, \langle CF \rangle$. By Lemma 1, this implies $\langle true \rangle\, (Nav \parallel ILP)\, [s_{Nav}]\, \langle CF \rangle$.
2) Applying Rule AG, we conclude that $\langle true \rangle\, (MP \parallel Nav \parallel ILP)\, [s_{MP}]\, \langle CF \rangle$.

The full proofs of the ES and CF properties are provided in Appendix A.

E. Experimental Results

We implemented the CBSA for the Quickbot rover in Matlab using the following parameter values: (1) number of distance sensors $N = 8$; (2) angle of detection of the sensors $\beta_s = 5°$; (3) maximum range of the sensors $R_s = 0.8m$; (7) radius of the wheels $r = 0.0325m$; (8) distance between the centers of the two wheels $l = 0.09925m$; (9) maximum linear velocity $v_{max} = 0.8m/s$; (10) maximum angular velocity $\omega_{max} = 7\pi\ rad/s$; (11) power station sensor detection range $d_{PS} = 0.1m$; (12) maximum battery level $B_{max} = 100$. We use a power model that is an affine function of the angular velocities of the two wheels: $P(\omega_l, \omega_r) = p_1(|\omega_l| + |\omega_r|) + p_2$, where p_1 and p_2 are constants and ω_l, ω_r are the rotational speeds of the two wheels. These are calculated from linear velocity v and angular velocity ω as follows.

$$\begin{cases} \omega_l &= \frac{2v - \omega l}{2r} \\ \omega_r &= \frac{2v + \omega l}{2r} \end{cases} \quad (4)$$

In our experiments, we use $p_1 = 0.15, p_2 = 0.01$. The constants used in Eq. 3 are $E_{MP} = BE_{MP} = 2.032$, and $E_{180} = 1.524$, which are computed based on v_{max} and ω_{max}. We also choose $\epsilon_{BE} = 0$ as we assume there are no errors. The global tick is $dt = 0.05s$. The update periods of *Mission Planning*, *Navigation*, and *Inner-Loop* & *Plant* are $4dt, 2dt$ and dt, respectively. We adopt the algorithm from the Coursera course "Control of Mobile Robots" for the go-to-target controller. The avoid-obstacles controller simply stops the rover, as in [28]. The backtrack controller implements the least-squares approach described in Section IV-B. In *Mission Planning*, the choose-next-target controller picks the next target in a sequence of pre-determined target locations when the rover arrives at the previous target. The recharge controller simply sets $T = PS$.

Fig. 3 shows the complete trajectory the rover takes with the following configuration. The rover's starting position is $P_0 = PS_1 = (-1, 0, 0)$, its initial heading angle is $\theta_0 = 0$, and the targets are $T_1 = (1.2, 0)$ and $T_2 = (0.3, 1.2)$, which must be visited in that order. The power stations are located at $PS_1 = (-1, 0), PS_2 = (0.8, -0.5), PS_3 = (0.4, 0.9)$. The black lines represent the forward paths and the red line represents the backward path. The rover is able to reach T_1 without having to recharge. At position $BT = (0.895, 0.570)$ on its way from T_1 to T_2, the battery level drops to $B = 29.07$, triggering recharge mode. The rover re-traces the forward path segment PS_2BT exactly (the red and black lines are indistinguishable) to recharge at power station PS_2. The battery level when it reaches PS_2 is $B = 0.07$, indicating a tight switching condition. After recharging to full capacity at PS_2, the rover takes a different path to the last target T_2. A video of the simulation can be viewed at https://youtu.be/i8WGVD5Vk7U.

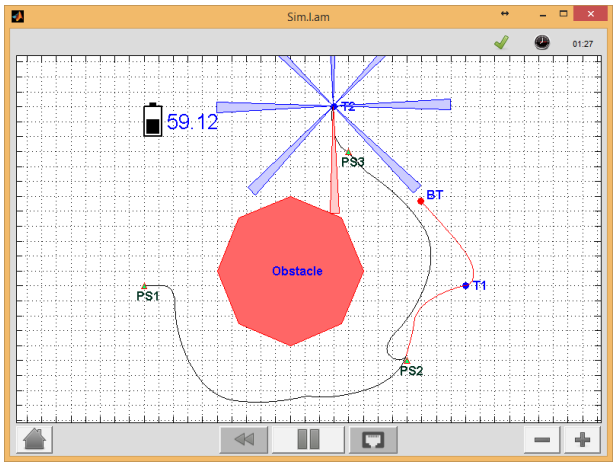

Fig. 3. Snapshot from a simulation showing the trajectory the rover takes to visit targets T_1 and T_2 while ensuring the ES and CF properties.

V. QUICKBOT MISSION COMPLETION

The Simplex architecture is not intended for assuring general liveness properties. This is reflected by the fact that a Decision Module looks ahead one update period to determine if the (safety) property in question is about to be violated; as such, it will not be able to detect a violation of a liveness property if the property's time horizon is unbounded. As we show here, however, the Simplex architecture can be made to work with *bounded* liveness properties. In particular, we design a CBSA for the Quickbot that assures a property called *Mission Completion* (MC).

Recall that the mission of the Quickbot rover is to visit a sequence of predetermined targets. The MC property ensures that the rover completes its mission within a given amount of time. This property is interesting in the context of Simplex not only because it is a (bounded) liveness property, but also because it is a higher-level mission-oriented property, several layers of control removed from the physical plant. The only physical parameter of the plant in question is (real) time. The MC property is also more software-oriented in nature, as it revolves around a data structure (a list of mission targets). The MC property can be formally expressed as $G(F_{<T}(\text{all targets are visited}))$, where G and F are the *always* and *eventually* operators in linear temporal logic, and T is the upper bound on amount of time the rover needs to complete its mission. The unbounded version of the MC property is $G(F(\text{all targets are visited}))$. In other words, it is always the case that the rover will eventually visits all targets.

In designing the CBSA for the MC property, we assume that obstacles are stationary and there is a *map* showing the targets and the location and shape of every obstacle. For simplicity, we also assume that the rover has enough battery to operate in the given mission-completion timeframe.

Similar to the case study in Section IV, the CBSA consists of three single-rate components: *Mission Planning*, *Navigation*, and *Inner-Loop* & *Plant*. The main objective of the *Mission Planning* component is to inform *Navigation* where the next target is. The BC for *Mission Planning* simply outputs one target after another, whereas the AC can generate intermediate *waypoints* in between targets. The *Mission Planning* component does not output the next waypoint or target until the rover arrives at the current waypoint/target.

The AC can choose waypoints to optimize some criteria according to its path-planning algorithm. Since the algorithm used by AC can be uncertified, it may generate a time-consuming path that violates the MC property. Another factor that may cause a violation of MC is the manner in which *Navigation*'s AC drives the rover through the waypoints to get to targets.

The *Navigation* component steers the rover to the target or waypoint computed by the *Mission Planning* component. As such, it plays an important role in assuring the MC property. *Navigation* alone, however, cannot guarantee MC because it does not know the sequence of targets. What it can do is to guarantee a bound on the time needed to move from one location to another on the map when *Mission Planning* activates its BC. The BC in *Navigation* is designed to guarantee this bound.

The DM in *Mission Planning* uses the bound to check whether a violation of MC is imminent. We say the rover is about to violate the MC property if, in the next time step, it may end up at a location from which the *Navigation*'s BC is unable to navigate through the remaining targets in time. When *Mission Planning*'s DM decides to switch, the switching is cascaded to *Navigation*.

The contracts for *Mission Planning* and *Navigation* are as follows. The *Mission Planning* component's contract is:

Inputs: p
Outputs: $T, ctlr$
Assumption: $ctlr = \text{BC} \Rightarrow$
$\forall(p_1, p_2), t(p_1, p_2) \leq tu(p_1, p_2)$
Guarantee: $t(p, Tseq) \leq remaining\_time$

where p is the current position of the rover, T is the output waypoint or target, $ctlr \in \{\text{AC}, \text{BC}\}$ indicates which controller is active in *Mission Planning*, $t(p_1, p_2)$ is the actual time needed to go from location p_1 to p_2, $tu(p_1, p_2)$ is the upper bound on $t(p_1, p_2)$ and is based on *Navigation*'s BC, $Tseq$ is the sequence of remaining targets, $t(p, Tseq)$ is the actual time needed to complete the mission from p, i.e., $t(p, Tseq) = t(p, Tseq[1]) + \sum_{i=1}^{n-1} t(Tseq[i], Tseq[i+1])$ where n is the size of $Tseq$, and $remaining\_time$ is the time until the mission completion deadline. The *Navigation* component's contract is:

Inputs: $T, ctlr, p, ir, v, \omega$
Outputs: v_T, ω_T
Assumption: A_P
Guarantee: $ctlr = \text{BC} \Rightarrow$
$\forall(p_1, p_2), t(p_1, p_2) \leq tu(p_1, p_2)$

where ir, v, ω, v_T, and ω_T are as described in Section IV, A_P is the conjunction of assumptions 2-7 about the rover given in Section IV.

The *Inner-Loop* & *Plant*'s contract is similar to the one in Section IV. For the actual implementation of *Navigation*'s BC, we can use the A* algorithm [29] to generate a collision-free path between two points on the map. To bound the time $tu(p_1, p_2)$ needed to traverse this path, the BC can rely on just two motion primitives: turning in-place and moving straight ahead at a constant speed. The *Mission Planning*'s DM needs a bound on the time needed for *Navigation*'s BC to complete the entire mission. It can use $tu(p_1, p_2)$ for this bound, which can be calculated on-the-fly. If the map is not too large, we can divide the map into grid cells and pre-compute the maximum time needed for mission completion from each cell.

The switching condition is then derived as follows. We first compute the region reachable by the rover in one update period of *Mission Planning*. We then compute the maximal $t(p', Tseq)$, for p' in the reachable region. The switching condition is then $t(p', Tseq) < remaining\_time - s_{MP}dt$, where $s_{MP}dt$ is the update period of *Mission Planning*. We now provide a proof outline of the MC property.

Theorem 3. $\langle true \rangle\ QB\ [s_{MP}]\ \langle MC \rangle$

Proof outline

1) We prove $\langle A_{TU} \rangle\ MP\ [s_{MP}]\ \langle MC \rangle$, where A_{TU} is the assumption in MP's contract given above.
2) We prove $\langle A_P \rangle\ Nav\ [s_{Nav}]\ \langle A_{TU} \rangle$. By Lemma 1, this implies $\langle true \rangle\ (Nav \parallel ILP)\ [s_{Nav}]\ \langle A_{TU} \rangle$.
3) Applying Rule AG, we conclude that $\langle true \rangle\ (MP \parallel Nav \parallel ILP)\ [s_{MP}]\ \langle MC \rangle$.

VI. Conclusions

We have presented a component-based Simplex architecture for assuring the runtime safety of component-based cyber-physical systems, and a detailed case study that illustrates how our proposed CBSA helps the Quickbot ground rover assure two safety properties: energy safety and collision freedom. We also presented a CBSA for Quickbot that ensures a bounded liveness property called mission completion. As future work, we plan to extend our case study to accommodate uncertainties in actuation and sensor readings, and to investigate the application of our component-based Simplex architecture to UAVs and squadrons of UAVs. Another direction for future work is to develop a design process for CBSA that can be used to determine what guarantees each component should provide to ensure a global property.

Acknowledgment

This work is supported in part by AFOSR Grant FA9550-14-1-0261, NSF Grants CNS-1421893, and CCF-1414078, and ONR Grant N00014-15-1-2208. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of these organizations.

References

[1] D. Seto, B. Krogh, L. Sha, and A. Chutinan, "The Simplex architecture for safe online control system upgrades," in *Proc. 1998 American Control Conference*, vol. 6, 1998, pp. 3504–3508.

[2] ——, "Dynamic control system upgrade using the simplex architecture," *Control Systems, IEEE*, vol. 18, no. 4, pp. 72–80, Aug 1998.

[3] L. Sha, "Using simplicity to control complexity," *IEEE Software*, vol. 18, no. 4, pp. 20–28, 2001.

[4] M. Abadi and L. Lamport, "Conjoining specifications," *ACM Trans. Program. Lang. Syst.*, vol. 17, no. 3, pp. 507–535, May 1995.

[5] K. L. McMillan, "A compositional rule for hardware design refinement," in *Proceedings of the 9th International Conference on Computer Aided Verification*, ser. CAV '97, 1997, pp. 24–35.

[6] E. W. Stark, "A proof technique for rely/guarantee properties," in *Fifth Conf. on Foundations of Software Technology and Theoretical Computer Science*, ser. Lecture Notes in Theoretical Computer Science, vol. 206. Springer-Verlag, Dec. 1985, pp. 369–391.

[7] F. B. Schneider, *On Concurrent Programming*. Springer-Verlag, 1997.

[8] S. Bak, T. T. Johnson, M. Caccamo, and L. Sha, "Real-time reachability for verified simplex design," in *35th IEEE Real-Time Systems Symposium (¡a href="http://2014.rtss.org/"¿RTSS 2014¡/a¿)*. Rome, Italy: IEEE Computer Society, Dec. 2014.

[9] S. Bak, K. Manamcheri, S. Mitra, and M. Caccamo, "Sandboxing controllers for cyber-physical systems," in *Proc. 2011 IEEE/ACM International Conference on Cyber-Physical Systems ICCPS*. IEEE Computer Society, 2011, pp. 3–12.

[10] D. Seto, D. Seto, L. Sha, L. Sha, N. L. Compton, and L. Col, "A case study on analytical analysis of the inverted pendulum real-time control system," 1999.

[11] "QuickBot MOOC v2," 2014. [Online]. Available: http://o-botics.org/robots/quickbot/mooc/v2/

[12] M. Aiello, J. Berryman, J. Grohs, and J. Schierman, "Run-time assurance for advanced flight-critical control systems*," in *Guidance, Navigation, and Control and Co-located Conferences*. American Institute of Aeronautics and Astronautics, Aug 2010, 0.

[13] J. Schierman, D. Ward, B. Dutoi, A. Aiello, J. Berryman, M. DeVore, W. Storm, and J. Wadley, "Run-time verification and validation for safety-critical flight control systems," in *Guidance, Navigation, and Control and Co-located Conferences*. American Institute of Aeronautics and Astronautics, Aug 2008.

[14] J. D. Schierman, M. D. DeVore, N. D. Richards, N. Gandhi, J. K. Cooper, K. R. Horneman, S. D. Stoller, and S. A. Smolka, "Runtime assurance framework development for highly adaptive flight control systems," SBIR Phase III Final Report AFRL-RQ-WP-TR-2016-0001, Dec. 2015.

[15] R. Alur and T. A. Henzinger, "Reactive modules," *Form. Methods Syst. Des.*, vol. 15, no. 1, pp. 7–48, Jul. 1999.

[16] K. M. Chandy, *Parallel Program Design: A Foundation*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1988.

[17] E. Clarke, D. Long, and K. McMillan, "Compositional model checking," in *Proc. of the Fourth Annual Symp. on Logic in Computer Science*, 1989, pp. 353–362.

[18] O. Grumberg and D. E. Long, "Model checking and modular verification," *ACM Trans. Program. Lang. Syst.*, vol. 16, no. 3, pp. 843–871, May 1994.

[19] A. Pnueli, "Logics and models of concurrent systems," K. R. Apt, Ed., 1985, ch. In Transition from Global to Modular Temporal Reasoning About Programs, pp. 123–144.

[20] D. Cofer, A. Gacek, S. Miller, M. W. Whalen, B. LaValley, and L. Sha, "Compositional verification of architectural models," in *Proceedings of the 4th International Conference on NASA Formal Methods*, ser. NFM'12, 2012, pp. 126–140.

[21] A. Cimatti, M. Dorigatti, and S. Tonetta, "OCRA: A tool for checking the refinement of temporal contracts," in *Automated Software Engineering (ASE), IEEE/ACM 28th Inter. Conf. on*, Nov 2013, pp. 702–705.

[22] F. Warg, B. Vedder, M. Skoglund, and A. Söderberg, "Safety ADD: A tool for safety-contract based design," in *Proceedings of the 2014 IEEE International Symposium on Software Reliability Engineering Workshops*, ser. ISSREW '14, 2014, pp. 527–529.

[23] D. Giannakopoulou, C. S. Pasareanu, and J. M. Cobleigh, "Assume-guarantee verification of source code with design-level assumptions," in *Proceedings of the 26th International Conference on Software Engineering*, ser. ICSE '04, 2004, pp. 211–220.

[24] L. Wang, A. D. Ames, and M. Egerstedt, "Multi-objective compositions for collision-free connectivity maintenance in teams of mobile robots," *ArXiv e-prints*, Aug. 2016.

[25] E. Lee and P. Varaiya, *Structure and Interpretation of Signals and Systems*. Addison-Wesley, 2003. [Online]. Available: https://books.google.com/books?id=R2mBQgAACAAJ

[26] A. Benveniste, B. Caillaud, D. Nickovic, R. Passerone, J.-B. Raclet, P. Reinkemeier, A. Sangiovanni-Vincentelli, W. Damm, T. Henzinger, and K. G. Larsen, "Contracts for system design," INRIA, Research Report RR-8147, Nov. 2012. [Online]. Available: https://hal.inria.fr/hal-00757488

[27] H. Barringer, D. Giannakopoulou, and C. S. Pasareanu, "Proof rules for automated compositional verification through learning," in *Proc. of the 2nd Workshop on Specification and Verification of Component-Based Systems (SAVCBS'03)*, Helsinki, Finland, 2003.

[28] D. Phan, J. Yang, D. Ratasich, R. Grosu, S. A. Smolka, and S. D. Stoller, "Collision avoidance for mobile robots with limited sensing and limited information about the environment," in *Proc. 15th International Conference on Runtime Verification (RV 2015)*, ser. Lecture Notes in Computer Science. Springer-Verlag, Sep. 2015.

[29] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, July 1968.

Appendix

The proofs below follow the proof outlines given in Section IV-D.

A. Proof of ES property

Proof of $\langle A_{BE}\rangle\, MP\, [s_{MP}]\, \langle ES\rangle$ We prove $\langle A_{BE}\rangle\, MP\, [s_{MP}]\, \langle ES\rangle$ by showing that if the ES property holds at the end of time step i, then it holds at the end of time step $i + s_{MP}$, i.e., the next period of ***MP***, where $i \bmod s_{MP} = 0$. The proof is easily extended to show that the ES property holds continuously during the time step. Let $B', p', W', ctlr'$ denote the values of $B, p, W, ctlr$, respectively, at the end of time step $i + s_{MP}$. We prove that if $B > E(p, PS)$ then $B' > E(p', PS)$. Since the rover traces through a sequence of recorded waypoints W when going back to PS, we have

$$E(p, PS) = \begin{cases} E_{180} + BE(p, PS, W), \text{ if } ctlr = \text{AC} \\ BE(p, PS, W), \text{ if } ctlr = \text{BC} \end{cases} \quad (5)$$

There are four cases:

Case 1: $ctlr =$ BC and $ctrl' =$ BC. The rover is in recharge mode. The backtracking energy needed to backtrack from current position p to the next position p' is $BE(p, p', W)$. Therefore the battery level at the end of the next decision period of ***MP*** is

$$B' = B - BE(p, p', W) \quad (6)$$

We assume that the ES property holds at current time, i.e.,

$$B > BE(p, PS, W) \quad (7)$$

We want to prove that the ES property still holds at end of the next decision period of ***MP*** i.e.,

$$B' > BE(p', PS, W') \quad (8)$$

The backtracking energy from current position p to PS can be partitioned into $BE(p, p', W)$ and $BE(p', PS, W')$, i.e.,

$$BE(p, PS, W) = BE(p, p', W) + BE(p', PS, W') \quad (9)$$

Re-arranging this equation gives:

$$BE(p', PS, W') = BE(p, PS, W) - BE(p, p', W) \quad (10)$$

Combining Eq. 7 and Eq. 10, we have:

$$BE(p', PS, W') < B - BE(p, p', W) \quad (11)$$

Combining Eq. 6 and Eq. 11, we have:

$$BE(p', PS, W') < B' \quad (12)$$

According to Eq. 5, $E(p', PS) = BE(p', PS, W')$, therefore:

$$B' > E(p', PS). \quad (13)$$

Case 2: $ctlr =$ AC and $ctrl' =$ AC. The energy needed to go from current position p to the next position p' is $FE(p, p') \leq E_{MP}$. The battery level at the end of the next decision period of ***MP*** is:

$$B' \geq B - E_{MP} \quad (14)$$

We assume that the ES property holds at the current time, i.e.,

$$B > E_{180} + BE(p, PS, W) \quad (15)$$

We want to prove that the ES property still holds at the end of the next decision period of ***MP*** i.e.,

$$B' > E_{180} + BE(p', PS, W') \quad (16)$$

Note that if the rover detects a new power station PS' on the way from p to p' then the ES property still holds at the end of the next decision period, because $BE(p', PS', W') \leq BE(p', PS, W')$. $ctrl =$ AC means the switching condition is false, i.e.,

$$B > E_{MP} + E_{180} + BE_{MP} + (1 + \epsilon_{BE})FE(PS, p) \quad (17)$$

Since $BE(p, PS, W) \leq (1 + \epsilon_{BE})FE(PS, p)$, we have:

$$B > E_{MP} + E_{180} + BE_{MP} + BE(p, PS, W) \quad (18)$$

Combining Eq. 14 with Eq. 18, we have:

$$B' > E_{180} + BE_{MP} + BE(p, PS, W) \quad (19)$$

$BE(p', PS, W')$ can be partitioned into $BE(p', p, W')$ and $BE(p, PS, W)$, i.e.,

$$BE(p', PS, W') = BE(p', p, W') + BE(p, PS, W) \quad (20)$$

Energy needed to backtrack from p' to p is bounded by BE_{MP}, thus:

$$BE(p', PS, W') \leq BE_{MP} + BE(p, PS, W) \quad (21)$$

Combining Eq. 19 with Eq. 21, we have:

$$B' > E_{180} + BE(p', PS, W') \quad (22)$$

According to Eq. 5, $E(p', PS) = E_{180} + BE(p', PS, W')$, therefore,

$$B' > E(p', PS). \quad (23)$$

Case 3: $ctlr =$ AC and $ctrl' =$ BC. The rover is switching from AC to BC, which means it must turn 180° before backtracking. The battery level in the next decision period of ***MP*** is:

$$B' = B - E_{180} - BE(p, p', W) \quad (24)$$

We assume that the ES property holds at the current time, i.e.,

$$B > E_{180} + BE(p, PS, W) \quad (25)$$

We want to prove that the ES property still holds at the end of the next decision period of ***MP*** i.e.,

$$B' > BE(p', PS, W') \quad (26)$$

Observe that if we let $B_{180} = B - E_{180}$ then Case 3 becomes Case 1 with the same proof.

Case 4: $ctlr =$ BC and $ctrl' =$ AC. When ***MP*** switches from BC to AC, the switching condition must evaluate to false, i.e.,

$$B > E_{MP} + E_{180} + BE_{MP} + (1 + \epsilon_{BE})FE(PS, p) \quad (27)$$

Thus, this case folds back to Case 2 with the same proof.

Proof of $\langle A_P\rangle\, Nav\, [s_{Nav}]\, \langle A_{BE}\rangle$

Since the decision $ctlr$ from ***MP*** is hardwired to the switch in *Nav*, we only need to prove that the backtrack controller

in *Nav* satisfies $BE < (1 + \epsilon_{BE})FE(PS, p)$. As discussed in Section IV-B, the sequence of values of (v, ω) on the backtrack path will have the same magnitudes as the ones on the forward path. Therefore the backtrack energy will be the same as the forward energy.

B. Proof of CF Property

Proof of $\langle A_P \rangle \, Nav \, [s_{Nav}] \, \langle CF \rangle$

We prove $\langle A_P \rangle \, Nav \, [s_{Nav}] \, \langle CF \rangle$ by proving that $\langle A_P \rangle \, AC2 \, [s_{Nav}] \, \langle CF \rangle$ and $\langle A_P \rangle \, BC2 \, [s_{Nav}] \, \langle CF \rangle$. Since AC2 is the same Simplex instance in [28], the proof of $\langle A_P \rangle \, AC2 \, [s_{Nav}] \, \langle CF \rangle$ is already available there. We show that $\langle A_P \rangle \, BC2 \, [s_{Nav}] \, \langle CF \rangle$. Similar to the proof of $\langle A_P \rangle \, Nav \, [s_{Nav}] \, \langle A_{BE} \rangle$, BC2 will track the forward path exactly, in the reverse direction. Since the forward path is collision-free as guaranteed by the proof that AC2 satisfies the CF property, it immediately follows that the backtracking path is collision-free as well, i.e., $\langle A_P \rangle \, BC2 \, [s_{Nav}] \, \langle CF \rangle$.