2021 60th IEEE Conference on Decision and Control (CDC)
December 13-15, 2021. Austin, Texas

Constraint-based Verification of Formation Control

Julien Alexandre dit Sandretto, Alexandre Chapoutot, Christophe Garion, Xavier Thirioux and Ghiles Ziat

***Abstract*— Collision-free motion planning of formation of robots is an essential property to assess for safety purpose. We propose in this paper a new formal verification method based on abstract interpretation and constraint satisfaction problems to reach this goal. We consider state of the art control algorithms for formation maneuver to generate trajectories for a group of robots. Additionally, bounded uncertainties are considered to represent potential localization and measure errors. The collision-free property is formalized using the constraint satisfaction problem framework.**

I. INTRODUCTION

Mobile robots have seen their capabilities drastically improve over the years. For instance, autonomous displacements are now possible in unknown environments thanks to the huge development of SLAM algorithms [1]. Another example is the use of multi-agent formations to increase the efficiency of missions such as exploration [2]. In such cases, safety properties of mobile robots, like absence of collisions between them or with environment, have to be asserted in order to avoid harmful scenarios.

Model-based verification uses mathematical models of systems to verify properties on systems. Robotic systems are usually modeled using *Ordinary Differential Equations* (ODEs) to represent the dynamics of the robots. A big challenge to assert safety of such robotic systems is the capability of taking into account uncertainties appearing in all the flow of information treatment. For instance, perception cannot be done without approximating sensor measures and therefore safe robotics algorithms should take care of handling such approximations. A probabilistic approach is usually followed to solve this problem, for instance [3]. In this paper, we follow a deterministic approach when considering bounded uncertainties on the initial states of the ODEs representing the system. Classic approaches for approximating solution of ODEs exist and use Taylor model-based flow-pipe constructions [4], [5] or Runge-Kutta methods [6], [7] to compute over-approximation of the reachable states starting from some initial states. In both methods, over-approximations of states are computed based on numerical domains, *i.e.*, computer representable sets, such as intervals.

In this work, a two-step approach for the verification of safety properties of behaviors of multiple robots is proposed. The first step consists in over-approximating the solution of the ODEs representing the system using *Abstraction Interpretation theory* [8]. It has been defined to study systems through sound abstractions, considering a set of behaviors instead one particular behavior, and has successfully been used to validate for instance aircraft embedded systems [9]. The second step consists in the resolution of a constraint satisfaction problem concerning the solutions of these ODEs using *Constraint Programming* [10]. The idea of mixing techniques from Constraint Programming and Abstract Interpretation is promising and has already been exploited, for example for efficient constraint solving [11], [12]. The present work is based on the abstract solving method introduced in [11].

The contribution of this paper is the definition of a new Constraint Satisfaction Problem (CSP) approach to reason efficiently over temporal functions, `i.e.`, robot trajectories, as defined in [13] which proposes a more efficient abstract domain to handle sets of abstract elements than the *powerset abstract domain* [14] or *disjunctive completion* [15]. The resulting framework is used on a collision-free property for a multi-agent formation control, previously proposed in [16].

Some previous work has focused on the resolution of constraints involving ODEs, as [17] or [18] in which the authors extend the iSAT algorithm with safe numerical integration of ODEs as a constraint filtering mechanism. In the same spirit, in [19] the authors propose a slightly adapted filtering algorithm applied to constraints that handles ODEs. More recently, [20] proposed a framework that is able to deal with a wide class of problems based on logical combination of high-level properties, involving ODEs. The present work differs from previous ones in several ways. First, these works are largely based on interval analysis, while our framework based on abstract domains allows the natural management of more complex representations (zonotopes, polyhedra, etc). Moreover, we incorporate within our framework natural properties of ODEs, such as their chronological and their contiguous aspects, which allows to have more efficient operations and, to our knowledge, has never been done yet.

This paper is organized as follows: Section II recalls the main features of abstract interpretation, validated numerical integration and constraint programming. The tree abstract domain representing robot trajectories is defined in Section III. Verification of bearing-based formation control is given in Section IV. Section V concludes and discusses perspectives.

II. PRELIMINARY NOTIONS

A. Brief Recall on Abstract Interpretation

Abstract Interpretation [8] is a fundamental approach to soundly deal with safe approximations. A focus on numerical

This work was partially supported by the Defense Innovation Agency (AID) of the French Ministry of Defense.

J. Alexandre dit Sandretto and A. Chapoutot are with U2IS, ENSTA Paris, Institut Polytechnique de Paris, Palaiseau, France {alexandre,chapoutot}@ensta-paris.fr

C. Garion, X. Thirioux and G. Ziat are with ISAE-SUPAERO, Université de Toulouse, Toulouse, France {garion,thirioux,ziat}@isae-supaero.fr

domains is considered in this paper, especially the *interval numerical abstract domain*.

A safe abstraction of sets of real values is based on a *Galois connection*. More precisely, let $\langle \wp(\mathbb{R}^n), \subseteq, \emptyset, \mathbb{R}^n, \cup, \cap \rangle$ be the lattice of subset of real values and let $\langle A, \sqsubseteq, \bot, \top, \sqcup, \sqcap \rangle$ be a lattice of abstract values where $\bot$ is the least element, $\top$ is the greatest element, $\sqcup$ is the join operator and $\sqcap$ is the meet operator. A Galois connection is a pair of functions (α, γ) such that: (i) $\alpha : \wp(\mathbb{R}^n) \to A$ is monotonic; (ii) $\gamma : A \to \wp(\mathbb{R}^n)$ is monotonic; (iii) $\forall S \in \wp(\mathbb{R}^n), x \in A, \quad \alpha(S) \sqsubseteq x$ and $S \subseteq \gamma(x)$. The function α is called the *abstraction function* and the function γ is called the *concretization function*. The notion of *safe abstraction* is formalized either by $\alpha(S) \sqsubseteq x$ or $S \subseteq \gamma(x)$. An abstract continuous function $F^\sharp$ is a safe abstraction of the continuous function $F^\natural$ if and only if $\forall x \in A, \left(\alpha \circ F^\natural \circ \gamma\right)(x) \subseteq F^\sharp(x)$. Note that the abstract function $F^\sharp$ is usually a redesign of $F^\natural$ defined over elements of A. In this work, $F^\natural$ will be a set of trajectories of a robot described by an ODE. Solutions of ODE can be embedded in abstract interpretation theory as shown in [6], so $F^\sharp$ will be an interval-based trajectory of a robot.

When dealing with computations involving sets of values, interval analysis [21] is a method designed to produce a sound over-approximation. Sets of intervals can be associated to order-theoretic operations and used as abstract domain in the theory of abstract interpretation [8]. Hereafter, an interval is denoted by $[x] = [\underline{x}, \overline{x}]$ with $\underline{x} \leqslant \overline{x}$ and the set of intervals on $\mathbb{R}$ is denoted by $\mathbb{IR} = \{[x] = [\underline{x}, \overline{x}] \mid \underline{x}, \overline{x} \in \mathbb{R}, \ \underline{x} \leqslant \overline{x}\}$. The abstract function $F^\sharp$ will be an *interval inclusion functions* [21].

B. Abstraction of ODEs

a) Problem Settings: Robots behavior can be modeled as solution of a set of ODEs given a set of possible initial values. More precisely, an Initial Value Problem for ODEs (IVP) is defined by

$$\begin{cases} \dot{\mathbf{y}}(t) = \mathbf{f}(t, \mathbf{y}(t), \mathbf{p}), \\ t \in \mathcal{T} := [0, t_{\text{end}}], \quad \mathbf{y}(0) \in \mathcal{Y}_0 \quad \text{and} \quad \mathbf{p} \in \mathcal{P}, \end{cases} \tag{1}$$

with $\mathbf{f}$ a non-linear function $\mathbf{f} : \mathbb{R} \times \mathbb{R}^n \times \mathbb{R}^p \to \mathbb{R}^n$, a finite time horizon $[0, t_{\text{end}}]$, $\mathcal{Y}_0$ a bounded set of initial values and $\mathcal{P}$ a bounded set of parameters. Note that this implies to deal with sets of trajectory solutions of Equation (1). We assume here classical hypotheses on $\mathbf{f}$ to ensure the existence and uniqueness of the solution of Equation (1).

The set $\mathcal{Y}(\mathcal{T}, \mathcal{Y}_0, \mathcal{P})$ stands for the set

$$\mathcal{Y}(\mathcal{T}, \mathcal{Y}_0, \mathcal{P}) = \{\mathbf{y}(t; \mathbf{y}_0, \mathbf{p}) : t \in \mathcal{T}, \mathbf{y}_0 \in \mathcal{Y}_0, \mathbf{p} \in \mathcal{P}\} \tag{2}$$

Intuitively, $\mathcal{Y}(\mathcal{T}, \mathcal{Y}_0, \mathcal{P})$ gathers all the points reached by the scalar solution $\mathbf{y}(t; \mathbf{y}_0, \mathbf{p})$ of Equation (1) starting from all scalar initial values $\mathbf{y}_0$ and all scalar parameters $\mathbf{p}$. Note that $\mathcal{Y}(\mathcal{T}, \mathcal{Y}_0, \mathcal{P})$ is hardly computable in general. The goal of a validated (or rigorous) numerical integration method is therefore to characterize the set of functions satisfying Equation (1), in the form of the values this set of functions can reach with their associated time instants, *i.e.*, $\{\mathbf{y}(t; \mathbf{y}_0) : \forall \mathbf{y}_0 \in \mathcal{Y}_0, \forall t \in [0, t_{\text{end}}]\}$. A convenient way to access those values is to use the *abstract domain of intervals* which uses interval analysis to compute an over-approximation of this set [21], [5], [6].

b) Guaranteed Numerical Integration: When considering the set of initial conditions as a box $[\mathbf{y}_0]$, solving Equation (1) using the interval technique framework makes possible the design of an inclusion function $[\mathbf{y}]([t]; [\mathbf{y}_0])$ for the computation of an over approximation of $\mathbf{y}([t]; [\mathbf{y}_0])$. To build it, a sequence of time instants $t_1, \ldots, t_s$ such that $t_1 < \cdots < t_s$ and a sequence of boxes $[\mathbf{y}_1], \ \ldots, [\mathbf{y}_s]$ such that $[\mathbf{y}]([t_{i+1}]; [\mathbf{y}_i]) \supseteq [\mathbf{y}_{i+1}]$ for all $i \in [0, s-1]$ are computed. From $[\mathbf{y}_i]$, computing the box $[\mathbf{y}_{i+1}]$ is a classical two-steps method (see [5]): (i) compute an *a priori* enclosure $[\tilde{\mathbf{y}}_i]$ of the set $\{\mathbf{y}(t_k; \mathbf{y}_i) \mid t_k \in [t_i, t_{i+1}], \mathbf{y}_i \in [\mathbf{y}_i]\}$ such that $\mathbf{y}(t_k; [\mathbf{y}_i])$ is guaranteed to exist and is unique, (ii) compute an enclosure of the solution $[\mathbf{y}_{i+1}]$ at time t_{i+1}. A box $[\mathbf{y}_i]$ is the Cartesian product of the time interval $[t_{i-1}, t_i]$ and the state interval $[\tilde{\mathbf{y}}_i]$ containing all the values the trajectory can reach during the time interval $[t_{i-1}, t_i]$.

c) Abstraction of Boxes as Disjunction of Linear Constraints: The solution of an IVP-ODE which is given as a set of timed boxes in the form $\{([t_1], [\tilde{\mathbf{y}}_1]), \ldots, ([t_{\text{end}}], [\tilde{\mathbf{y}}_{\text{end}}])\}$ can easily be translated into a disjunction of constraints since each pair $([t_i], [\tilde{\mathbf{y}}_i])$ corresponds to a quantified Boolean proposition:

$$([t_i], [\tilde{\mathbf{y}}_i]) \equiv (\forall t \in [t_i])(\exists \mathbf{y} \in [\tilde{\mathbf{y}}_i])(\mathbf{y}(t) = \mathbf{y}) \tag{3}$$

or defined as a constraint over the variables t and $\mathbf{x}$:

$$([t_i], [\mathbf{y}_i]) \equiv (t \in [t_i]) \wedge (\mathbf{y} \in [\mathbf{y}_i]) \tag{4}$$

where $t \in [t_i]$ means $\underline{[t_i]} \leqslant t \leqslant \overline{[t_i]}$ with $\underline{[t_i]}$ and $\overline{[t_i]}$ the lower and outer bounds of $[t_i]$ respectively. Eventually, the abstraction of the set of trajectories can be modeled as a disjunction of all the constraints from Equation (4) since at each time $t \in [t_0, t_{\text{end}}]$, $\mathbf{y}(t)$ must verify only one of these constraints.

C. Constraint Satisfaction Problems

In this paper, ODEs solutions are incorporated into a Constraint Programming framework. Constraint Programming [22], [10] is a declarative programming paradigm in which users specify the constraints of a system, generally stated as first-order logic formulæ, and then rely on a solver, which comes with constraint filtering mechanisms and choice heuristics, to establish the satisfiability of the constraints.

1) Constraint Satisfaction Problem: A continuous Constraint Satisfaction Problem (CSP) can be defined as a triplet $\langle \mathcal{V}, \mathcal{D}, \mathcal{C} \rangle$, where $\mathcal{V} = \{v_1, \ldots, v_n\}$ is a set of variables, $\mathcal{D} = \{d_1, \ldots, d_n\}$ a set of domains, each one being associated to a variable, and $\mathcal{C} = \{c_1, \ldots, c_m\}$ is a set of constraints over the variables.

a) Constraint Language: We consider a standard constraint language using a finite and fixed set $\mathcal{V}$ of real-valued variables, numeric and Boolean expressions. A constraint c is a Boolean expression whose concrete semantic corresponds

to the set of mappings, called instances, from variables to values i for which the evaluation of c, denoted $c(i)$, yields $true$. Solving a CSP usually means to find all the instances that satisfy every constraint of the problem. Because this is generally impossible when the variables domains are continuous, solvers generally compute a set of boxes (in our case any abstract element) that *covers* the solution space. In order to build this cover, such a solver alternates two main steps: (1) *filtering* which reduces the domains of the variables by removing values that cannot be solutions. (2) *exploration* when the domains cannot be reduced anymore, solvers creating then two complementary sub-problems.

As repeating these two steps in turn is not guaranteed to terminate, this procedure continues until the search space contains no solution, only solutions, or is smaller than some parameter according to a size metric. We base our work on the constraint solving framework introduced in [11] in which a solving method based on abstract domains is introduced.

2) Abstract Domains for Constraint Solving: In [11], the authors define operators and requirements over these operators an abstract domain must satisfy in order to be used in a constraint resolution scheme. The following definitions recall these requirements.

Definition 2.1: Abstract domains for constraint solving are given by (a) a partial order $\langle \mathcal{D}^\sharp, \sqsubseteq \rangle$ and the usual abstract set operators and values $\langle \top^\sharp_{\mathcal{D}}, \bot^\sharp_{\mathcal{D}}, \sqcap^\sharp, \sqcup^\sharp \rangle$ (b) an abstraction α and a concretization function γ along with:

- a size function $\tau : \mathcal{D}^\sharp \to \mathbb{R}^+$
- a splitting operator on $\mathcal{D}^\sharp$, $\oplus : \mathcal{D}^\sharp \to P(\mathcal{D}^\sharp)$,
- a constraint filtering operator $\rho^\sharp_{\mathcal{D}} : \mathcal{D}^\sharp \times \mathcal{C} \to \mathcal{D}^\sharp \cup \{\bot^\sharp\}$, which given an abstract value e and a constraint c computes the smallest abstract value (possibly empty) entailed by c and e.

The split operator $\oplus$ should respect Definition 9 of [11], which we recall here:

Definition 2.2: A split operator has the following properties: (i) $\forall d \in \mathcal{D}^\sharp, |\oplus(d)|$ is finite (ii) $\forall d \in \mathcal{D}^\sharp, \forall d_i \in \oplus(d), d_i \sqsubset d$ (iii) $\forall d \in \mathcal{D}^\sharp, \gamma(d) = \bigcup\{\gamma(d_i) \mid d_i \in \oplus(d)\}$

The first property is necessary to guarantee the termination of the procedure, the second ensures that the operator actually splits, *i.e.*, computes smaller elements than the original one, and the last one guarantees the soundness of the solving process, *i.e.*, that the splitting does not lose instances. Also, in order to have a terminating process, no infinite sequence of splits and constraint filtering must exist, and thus, every such finite sequence should yield an element smaller than a given parameter with respect to τ. More formally, $\oplus$ and τ must be compatible according to Definition 10 of [11]:

Definition 2.3 (Compatibility of $\oplus$ and τ): The split operator $\oplus$ and the size operator τ are compatible, iff for any reductive operator $\rho^\sharp$ we have:

$$\forall d \in \mathcal{D}^\sharp, \forall r \in \mathbb{R}^+, \exists k, \forall i \geq k, \tau((\oplus \circ \rho^\sharp)^i(d)) \leq r$$

III. TREE ABSTRACTION

A. A new abstract domain for ODE

We propose a new abstraction for ODEs based on the following principle: the solution of an ODE is first approximated as the union of all time frames, using the `join` operator, then is filtered using a potentially large number of constraints. Our idea is to incorporate additional information into the `join` operation that will speed up the filtering of constraints by proposing a *tree abstract domain* $\mathbb{T}(\mathcal{D}^\sharp)$. This domain can be viewed as a kd-tree in which leaves are defined using the numerical abstract domain $\mathcal{D}^\sharp$ and internal nodes, also called summaries, give information about their subtrees. Our use of summaries is similar to the one defined in [23], but applied to a CSP framework. In particular, we show that filtering a constraint can be made more efficiently using the summaries information. Moreover, the fact that ODEs are continuous objects greatly enhances the relevance of our summaries.

Definition 3.1: Given an abstract domain $\langle \mathcal{D}^\sharp, \alpha_D, \gamma_D, \sqcup_D, \sqcap_D, \oplus_D \rangle$, an element $t \in \mathbb{T}(\mathcal{D}^\sharp)$ is either (1) a leaf: $\mathcal{D}^\sharp \to \mathbb{T}(\mathcal{D}^\sharp)$, denoted by $\textbf{leaf}(d)$, (2) or an union node: $\mathcal{D}^\sharp \times \mathbb{T}(\mathcal{D}^\sharp) \times \mathbb{T}(\mathcal{D}^\sharp) \to \mathbb{T}(\mathcal{D}^\sharp)$ denoted by $\textbf{union}(u, t_1, t_2)$ with $u = \textbf{hull}(t_1) \sqcup_D \textbf{hull}(t_2)$.

hull is a summary function $\mathbb{T}(\mathcal{D}^\sharp) \to \mathcal{D}^\sharp$ such that $\textbf{hull}(leaf(d)) = d$, and $\textbf{hull}(union(u, t_1, t_2)) = u$. The concretization for this representation is given by

$$\gamma(t) = \begin{cases} \gamma_D(t'), & \text{when } t = \textbf{leaf}(t') \\ \gamma(t_1) \cup \gamma(t_2), & \text{when } t = \textbf{union}(u, t_1, t_2) \end{cases}$$

Here, we exploit the fact that $\gamma(d_1 \sqcup d_2)$ strictly contains $\gamma(d_1) \cup \gamma(d_2)$ to provide a summary u of what is contained in the sub-trees t_1 and t_2, in the sense that $\gamma(t_1) \subseteq \gamma_D(u) \wedge \gamma(t_2) \subseteq \gamma_D(u)$. This can be seen as a two-level abstraction, as u is an abstraction of both t_1 and t_2. This allows us to define a fast pre-computation for the meet and the filtering operations. We define also a partial order $\sqsubseteq_T$ for trees such that

$$t_1 \sqsubseteq_T t_2 \triangleq \forall l \in \textbf{leaves}(t_1), \exists l' \in \textbf{leaves}(t_2), l \sqsubseteq_D l'$$

We consider that a tree t_1 is smaller than a tree t_2 if each of its leaves are smaller than one of the leaves of t_2. The auxiliary function **leaves** computes the set of all leaves of a tree.

Our representation being able to encode exactly disjunctions, we define the join operator as

$$t_1 \sqcup_T t_2 = \textbf{union}(\textbf{hull}(t_1) \sqcup_D \textbf{hull}(t_2), t_1, t_2)$$

The greater the intersection of two elements is, the more accurate the summary is. Because of the continuous nature of ODEs, this property is particularly interesting as two successive elements always intersect.

For the definition of the meet operator for trees, we exploit the summaries to benefit from a pre-computation. If two summaries do not intersect, then the whole corresponding trees do not either.

B. Split and Measure

To embed our abstract domain in the constraint solving framework, we must define a split operation $\oplus$, along with a measure function τ and a constraint filtering operator ρ. The operator $\oplus$, given by Definition 3.2, performs a symbolic cut when the tree is a **union** node, and otherwise uses $\oplus_D$.

Definition 3.2: Split operator for $T(\mathcal{D}^\sharp)$ is defined by

$$\oplus(t) = \begin{cases} \{\textbf{leaf}(d') \mid d' \in \oplus_D(d)\}, & \text{if } t = \textbf{leaf}(d) \\ \{t_1, t_2\} & \text{if } t = \textbf{union}(u, t_1, t_2) \end{cases}$$

Proposition 3.1: The operator $\oplus$ is a split operator according to Definition 2.2

We now define the size function $\tau : \mathbb{T}(D) \to \mathbb{R}$ for trees, which is given in Definition 3.3.

Definition 3.3: Size operator is defined by

$$\tau(e) = \begin{cases} \tau_D(d), & \text{when } e = \textbf{leaf}(d) \\ +\infty & \text{when } e = \textbf{union}(u, t_1, t_2) \end{cases}$$

The splitting is done as long as there are disjunctions in the representation and when a leaf is reached, a branching to (τ_D) is performed.

Proposition 3.2: The split operator $\oplus_T$ and the size operator τ_T are compatible for any reductive operator ρ, according to Definition 2.3

C. Constraint Filtering

Finally, note that fast precomputation is not only available for the meet operation but also for the filtering of a constraint:

Definition 3.4: Given a tree t and a constraint c, the filtering operator $\rho_T : \mathbb{T}(\mathcal{D}^\sharp) \to \mathcal{C} \to \mathbb{T}(\mathcal{D}^\sharp)$ is such that:

$$\rho(t, c) = \begin{cases} \textbf{leaf}(\rho_D(t')) & \text{when } t = \textbf{leaf}(t') \\ \bot & \text{when } t = \textbf{union}(u, t_1, t_2) \wedge \\ & \quad \rho(u) = \bot_D \\ \sqcup_T(\rho(t_1), \rho(t_2)) & \text{when } t = \textbf{union}(u, t_1, t_2) \wedge \\ & \quad \rho(u) \neq \bot_D \end{cases}$$

Instead of filtering a constraint for each atom of a disjunction, we do it first for their summary. If we can prove that a summary u violates the constraint c, *i.e.*, $\forall i \in \gamma_D(u), \neg c(i)$ (resp. $\forall i \in \gamma_D(u), c(i)$), then the child elements will also violate it. In the case where we cannot draw a definitive conclusion, we propagate the filtering towards the sub-trees.

Proposition 3.3: The filtering operator ρ contracts and is complete, *i.e.*,

- $\rho_T(t, c) \sqsubseteq_T t$ (contraction)
- $\forall i \in \gamma_T(t), c(i) \implies i \in \gamma_T(\rho(t))$ (completeness)

The contraction property ensures that values are only removed from the abstract element, and the completeness property guarantees that no solution is removed from it.

The tree abstraction that has been defined in this section exploits the continuity aspect of ODEs solutions to propose a fast pre-computation for the filtering of a constraint and the meet operation. It can thus be seen as an incremental powerset, that gradually augments its precision, starting from the precision of a base abstract domain D to the precision of its powerset $P(D)$ if needed only. This will greatly speed-up the solving process in most cases.

IV. VERIFICATION OF FORMATION MANEUVER

A. Tools used

We present here the two tools used for verification of formation maneuver.

DynIbex is used for abstraction of solutions of ODE [7]. It is a library written in C++ using Ibex, a library for constraint processing over real numbers. It adds a validated numerical integration method to compute an over-approximation of the reachable set of an ODE in a time interval. It returns a set of timed boxes in the form $\{([t_1], [\tilde{\mathbf{x}}_1]), \ldots, ([t_{\text{end}}], [\tilde{\mathbf{x}}_{\text{end}}])\}$ (and also tighter approximations at given time steps) using Runge-Kutta methods with interval analysis.

AbSolute is a constraint solver based on abstract domains [11]. It is built upon the Abstract Interpretation framework, and features several techniques and classical heuristic from Constraint Programming. The solver is written in OCaml and is usable with several numeric abstract domains (intervals, congruences, octagons, polyhedra), and domain combinators (products), some of which are based upon the Apron library [24]. We use AbSolute as an oracle for DynIbex to verify if the constraints over dynamic objects described as ODEs hold. The tree abstract domain, presented in Section III, is implemented as plugins of AbSolute.

B. Bearing-based Formation Control

Distributed multi-agent formation control is mainly driven by inter-neighbor relative position or distance constraints. In recent work, the bearing-based approach has been considered (see [16]): the target formation is constrained by inter-neighbor bearings. Indeed, bearings are invariant in rotation, translation and scaling and therefore offer a convenient approach to define formation control. Nonetheless, using bearing-based formations adds additional constraints between robots in order to ensure expected. Moreover, in such framework, at least two leaders have to be defined as described in [16]. Figure 1 presents an example of square formation with four robots in 2D plan (back centered square).

A brief review on the formation maneuver control as defined in [16] is given. Consider n agents in $\mathbb{R}^d$ ($n \geqslant 2$ and $d \geqslant 2$). Their position at time t is denoted by $p_i(t) \in \mathbb{R}^d$ with $i \in \{1, \ldots, n\}$. Interactions between agents is described by a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with a vertex set $\mathcal{V} = \{1, \ldots, n\}$ and an edge set $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$. An edge $(i, j) \in \mathcal{E}$ means that agent i can measure the relative bearing of agent j so it is a neighbor of Agent j. The set of all neighbors of Agent i is denoted by $\mathcal{N}_i = \{j \in \mathcal{V} : (i, j) \in \mathcal{E}\}$. Edges are not directed: if i is a neighbor of j, then j is also a neighbor of i. A formation denoted by $\mathcal{G}(p)$ is a graph $\mathcal{G}$ such that each vertex i of $\mathcal{G}$ is associated to $p_i(t)$. The relative bearing of p_j with respect to p_i is defined by $e_{ij} := p_j - p_i$, $g_{i,j} := \frac{e_{ij}}{\|e_{ij}\|}$, with $\| \, . \, \|$ being the Euclidean norm.

Suppose the velocity of n_ℓ agents is given. Such agents are called *leaders* and the rest of n_f agents are called *followers* ($n_l + n_f = n$). $\mathcal{V}_\ell = \{1, \ldots, n_\ell\}$ and $\mathcal{V}_f = \{n_\ell + 1, \ldots, n_f\}$ respectively stands for the index set for leaders and followers. The dynamics of leaders and followers follows a single integrator model

$$\begin{aligned} \dot{p}_i &= v_i^*(t) && i \in \mathcal{V}_\ell \\ \dot{p}_i &= -k_p \sum_{j \in \mathcal{N}_i} P_{g_{ij}^*}(p_i(t) - p_j(t)) - k_i \xi(t) && i \in \mathcal{V}_f \\ \dot{\xi}_i &= \sum_{j \in \mathcal{N}_i} P_{g_{ij}^*}(p_i(t) - p_j(t)) && i \in \mathcal{V}_f \end{aligned}$$

with $v_i^*(t)$ the leader velocity reference, g_{ij}^* stands for the target formation to reach by the group of robots and $P_{g_{ij}^*} = I_d - g_{ij}^*(g_{ij}^*)^T$. This controller can be used to follow a predefined plan made of different way-points.

In the following, a group of four robots will be considered in the particular context of a square formation pattern in 2D plan. A trajectory r can be defined as a disjunction of predicates $r = (p_1 \wedge t_1) \vee (p_2 \wedge t_2) \cdots \vee (p_n \wedge t_n)$ where each p_i represents the position of agent i at a time instant t_i. This can be understood as: the agent is either at point p_1 during the time frame t_1, or at position p_2 during the time frame t_2, etc. This property is always true, as at least one of its atoms will be true. When dealing with a formation of n agents, we have to define n trajectories $r_1, \ldots r_n$ and the constraint corresponding to the possible collision between agents is expressed by $(r_1 \wedge r_2) \vee \ldots (r_1 \wedge r_n) \vee (r_2 \wedge r_3) \vee \cdots \vee (r_{n-1} \wedge r_n)$.: there is a collision if two constraints representing two trajectories are true at the same time and therefore the global formula is true.

C. Atomic displacements

A set of atomic displacements is considered in this step to validate the inter-agent collision-free trajectories. Three displacements respecting the square pattern are considered: translation, scaling and rotation. A composition of these three displacements is performed to generate trajectories of the different agents following bearing-based formation control.

A finite number of values for each displacement is considered: north, south, east, west direction are considered for translation; contraction and dilation for scaling; four rotations of the formation over the centroid are available with angles $\pi/4$, $\pi/2$, $3\pi/4$ and π. See Figure 1 for some examples of displacements. In total, 108 atomic displacements can be considered.

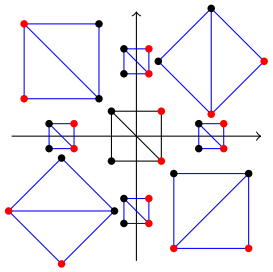

Fig. 1. Examples of atomic displacements combining translation, rotation and scaling: black dots are followers and red dots are leaders, black inner square is initial formation and blue squares are formations resulting of atomic displacements.

Two different sources of uncertainties are considered to check collision-free property for these atomic displacements: uncertainties on initial positions of agents and on inter-agent distances. This allows different scenarios with: (i) no uncertainties; (ii) only initial position uncertainties; (iii) only distance measure uncertainties; (iv) both initial position and distance measure uncertainties. These scenarios try to detect which combination of displacements can generate a collision and the corresponding robustness w.r.t. uncertainties.

TABLE I

COLLISION-FREE CHECKING ON ATOMIC DISPLACEMENTS

	T1T2	T1T3	T1T4	T2T3	T2T4	T3T4
NO	27/81	1/107	4/104	3/105	3/105	4/104
EI 0.01	27/81	1/107	8/100	9/99	3/105	17/91
EI 0.1	27/81	12/96	41/67	52/56	26/82	54/54
ED 0.01	27/81	3/105	15/93	17/91	11/97	52/56
ED 0.1	27/81	28/30	78/30	51/57	50/58	61/47
EID 0.01	27/81	4/104	25/83	28/80	15/93	54/54
EID 0.1	27/81	36/72	91/17	68/40	55/53	68/40

A summary of the verification of collision-free property on atomic displacements is given in Table I. In this table, No stands for no uncertainty, EI stands for uncertainty on initial conditions, ED stands for uncertainty on distance measure and EID stands for uncertainties on both initial position and distance measures. Two values are considered for uncertainties, namely 0.01 and 0.1. The number of satisfiable and unsatisfiable problems (read SAT/UNSAT) is reported for each scenario and each pair of trajectory between agents (T1T2 stands for trajectory of agent 1 and agent 2). An UNSAT scenario means that no collision have been found and so the atomic displacement is safe while a SAT scenario implies that a potential collision has been found.

Looking at this experimental evaluation of the bearing-based formation control, we note that increasing uncertainties increases the number of possible collisions. More precisely, rotations of pi has to be avoid as it generates most of the possible collisions with or without uncertainty considered. Once the uncertainty is 0.1, atomic displacements including rotations of π, $3\pi/4$ or $\pi/2$ generates collisions when combined with other displacements. In consequence, a global planning of a bearing-based formation should consider to perform rotations only when necessary (especially of angle $1/4\pi$) and not a combination with other kinds of displacements to avoid collisions.

D. Complete mission

A path following mission is considered in this section, for which a sequence of atomic displacements is defined to reach a target position. Two scenarios have been considered:

- the first scenario considers way-points corresponding to atomic displacements merging scaling, translation and rotation together. In this scenario, 11 way-points have been defined. The simulation result is given in Figure 2 considering no uncertainty;
- the second scenario considers way-points corresponding to either a pure rotation or a possible combination of scaling and translation. In this scenario, 14 way-points have been defined. The simulation result is given in Figure 3 considering no uncertainty.

Applying the proposed method to detect collision on Scenario 1 found a possible collision at time $t = 40.3012$s between the trajectories of agent 2 and agent 3 (at coordinates $x = 75.3000150552$ and $y = -24.3395564986$). This collision may happen during the second atomic displacement merging rotation and other displacements between way-

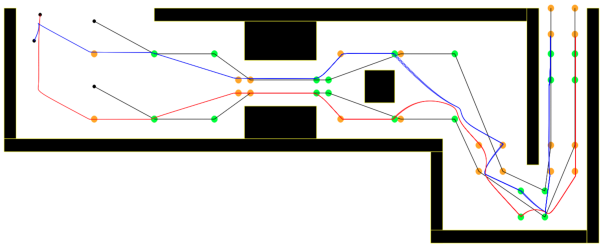

Fig. 2. Path following mission, scenario 1 way-points associated to all possible atomic displacements together. In particular way point 7 is a rotation, a scaling reduction and a translation. No uncertainties are considered.

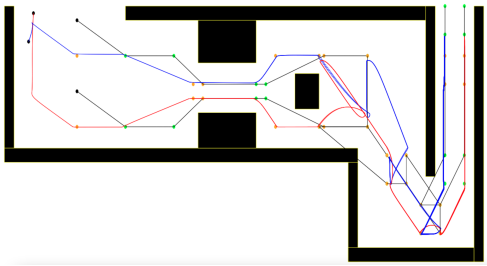

Fig. 3. Path following mission, scenario 2 way points associated to pure rotation or translation/scaling displacements. No uncertainties are considered. More complex trajectories are generated but without collision.

points 7 and 8. No collision is found in Scenario 2 when rotations have been considered alone for the definition of atomic displacements. Increasing the uncertainties on initial position and measure distances on Scenario 2 implies possible collisions during rotation displacements. Our proposed method can then be used to validate global path for robot formation by asserting the absence of inter-agent collisions.

V. CONCLUSION

We have shown that it is possible to validate formation control using techniques from different areas, namely Constraint Programming, Abstract Interpretation and Interval Analysis. Our work can be summarized in two parts: a) a correct over-approximation of the reachability states of sets of ODEs, and b) a resolution of constraint satisfaction problems implying such ODEs. For the latter, we have defined an abstract domain able to take advantage of the characteristics of ODE solutions, particularly their continuous nature. We have demonstrated the usefulness of our techniques on a realistic application implying the absence of collisions in a formation of robots.

The work we have done may be deepened in several ways: it could be effective to use some more expressive abstract domains for the nodes than for the leaves (*e.g.*, relational representation like polyhedra for nodes and intervals for leaves) to minimize the loss of precision due to join operation, and thus further improve the capabilities of our tree abstract domain. Moreover, the framework presented could also be used to define a global planner for formation control algorithm based on [25] on switched systems.

REFERENCES

[1] C. Cadena, L. Carlone, H. Carrillo, Y. Latif, D. Scaramuzza, J. Neira, I. Reid, and J. Leonard, "Past, present, and future of simultaneous localization and mapping: Towards the robust-perception age," *IEEE Transactions on Robotics*, vol. 32, no. 6, p. 1309–1332, 2016.

[2] K.-K. Oh, M.-C. Park, and H.-S. Ahn, "A survey of multi-agent formation control," *Automatica*, vol. 53, no. C, p. 424–440, 2015.

[3] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics*. The MIT Press, 2005.

[4] X. Chen, E. Ábrahám, and S. Sankaranarayanan, "Flow*: An analyzer for non-linear hybrid systems," in *Computer Aided Verification*. Springer, 2013, pp. 258–263.

[5] N. S. Nedialkov, K. R. Jackson, and G. F. Corliss, "Validated solutions of initial value problems for ordinary differential equations," *Applied Mathematics and Computation*, vol. 105, no. 1, pp. 21–68, 1999.

[6] O. Bouissou and M. Martel, "Abstract interpretation of the physical inputs of embedded programs," in *Proc. of Verification, Model Checking, and Abstract Interpretation*. Springer, 2008.

[7] J. Alexandre dit Sandretto and A. Chapoutot, "Validated explicit and implicit Runge–Kutta methods," *Reliable Computing*, vol. 22, no. 1, pp. 79–103, 07 2016.

[8] P. Cousot and R. Cousot, "Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints," in *Proc. of Symposium on Principles of Programming Languages*. ACM Press, 1977, pp. 238–252.

[9] B. Blanchet, P. Cousot, R. Cousot, J. Feret, L. Mauborgne, A. Miné, D. Monniaux, and X. Rival, "A static analyzer for large safety-critical software," in *Conference on Programming Language Design and Implementation*. ACM Press, 2003.

[10] F. Rossi, P. Beek, and T. Walsh, *Handbook of constraint programming*. Elsevier, 2006.

[11] M. Pelleau, A. Miné, C. Truchet, and F. Benhamou, "A constraint solver based on abstract domains," 01 2013.

[12] M. Pelleau, C. Truchet, and F. Benhamou, "The octagon abstract domain for continuous constraints," *Constraints*, vol. 19, no. 3, pp. 309–337, 2014.

[13] G. Ziat, O. Mullier, J. A. d. Sandretto, C. Garion, A. Chapoutot, and X. Thirioux, "Abstract domains for constraint programming with differential equations," in *NSAD 2020: Proceedings of the 9th ACM SIGPLAN International Workshop on Numerical and Symbolic Abstract Domains*. Virtual USA, France: ACM, 2020.

[14] G. Filé and F. Ranzato, "The powerset operator on abstract interpretations," *Theoretical Computer Science*, vol. 222, no. 1, 1999.

[15] P. Cousot and R. Cousot, "Systematic design of program analysis frameworks," in *Proc. of Symposium on Principles of Programming Languages*. ACM Press, 1979.

[16] S. Zhao and D. Zelazo, "Bearing-based formation stabilization with directed interaction topologies," *54th IEEE Conference on Decision and Control*, pp. 6115–6120, 2015.

[17] J. Cruz and P. Barahona, "Constraint satisfaction differential problems," 09 2003, pp. 259–273.

[18] A. Eggers, M. Fränzle, and C. Herde, "Application of constraint solving and ode-enclosure methods to the analysis of hybrid systems," *AIP Conference Proceedings*, vol. 1168, 09 2009.

[19] A. Goldsztejn, O. Mullier, D. Eveillard, and H. Hosobe, "Including ordinary differential equations based constraints in the standard CP framework," in *Proc. of Principles and Practice of Constraint Programming*. Springer, 2010, pp. 221–235.

[20] J. Alexandre dit Sandretto, A. Chapoutot, and O. Mullier, *Constraint-Based Framework for Reasoning with Differential Equations*. Springer, 2018, pp. 23–41.

[21] R. E. Moore, *Interval Analysis*. Prentice Hall, 1966.

[22] U. Montanari, "Networks of constraints: fundamental properties and applications to picture processing," *Information Science*, vol. 7, no. 2, pp. 95–132, 1974.

[23] A. Becchi and E. Zaffanella, "Revisiting polyhedral analysis for hybrid systems," in *Proc. of Static Analysis Symposium*. Springer, 2019.

[24] B. Jeannet and A. Miné, "Apron: A library of numerical abstract domains for static analysis," in *Proc. of the 21th International Conference Computer Aided Verification*, 2009.

[25] A. Le Coënt, J. Alexandre dit Sandretto, A. Chapoutot, and L. Fribourg, "An improved algorithm for the control synthesis of nonlinear sampled switched systems," *Formal Methods in System Design*, vol. 53, no. 3, pp. 363–383, 2018.