1 System Overview

We present a computational framework for contract-based compositional design with support for incremental contract evolution through fixpoint iteration. The framework addresses the problem of analyzing how local changes to component specifications propagate through interconnected system architectures, with particular emphasis on cyber-physical systems where components interact through shared variables and exhibit feedback loops.

The implementation realizes a network model where components are described by assume-guarantee contracts $\mathcal{C} = (A, G)$, where A represents assumptions about the environment and G represents guarantees provided by the component. The framework supports three key capabilities: (i) modeling component interactions through typed interfaces with projection operators, (ii) propagating contract deviations through the network via forward and backward operators, and (iii) computing fixpoints that restore well-formedness after local changes.

The system architecture consists of five primary subsystems:

1. **Contract Model Layer**: Defines the mathematical structures for contracts, behavior sets, and deviations using zonotope-based geometric representations.

2. **Component Model Layer**: Implements physical components with mixed-integer linear programming (MILP) formulations that capture input-output relationships and enable computation of forward (post) and backward (pre) transformers.

3. **Network Model Layer**: Maintains the compositional structure as a directed graph with components as nodes and typed interfaces as edges, supporting cycle detection and well-formedness checking.

4. **Evolution Engine**: Implements the evolution operator Φ that propagates deviations through interfaces using the component transformers, and the fixpoint iteration algorithm that applies Φ repeatedly until convergence.

5. **Validation and Analysis Layer**: Provides well-formedness checking, system-level contract satisfaction verification, and iteration analytics for understanding convergence behavior.

The framework is applied to a drone system comprising five interconnected components (Battery, PowerManager, Motor, FlightController, NavigationEstimator) arranged in a topology with feedback loops. Two experimental scenarios demonstrate complementary propagation patterns: a motor upgrade scenario exhibiting pure backward propagation through assumption strengthening, and a sensor degradation scenario exhibiting pure forward propagation through guarantee relaxation.

2 Modeling Framework

2.1 Contract and Behavior Representation

A contract $\mathcal{C} = (A, G)$ consists of assumptions A and guarantees G, both represented as behavior sets. Each behavior set is formalized as a finite union of zonotopes, providing a computationally tractable representation that supports exact Minkowski operations and geometric set operations.

A zonotope $\mathcal{Z} \subset \mathbb{R}^n$ is defined in generator form as:

$$\mathcal{Z} = \left\{ c + \sum_{i=1}^{p} \xi_i g_i \,\middle|\, \xi_i \in [-1, 1] \right\} \tag{1}$$

where $c \in \mathbb{R}^n$ is the center and $g_i \in \mathbb{R}^n$ are generator vectors. This representation offers computational advantages: zonotopes are closed under Minkowski sum and linear transformation, and operations such as projection reduce to simple matrix operations on the generator representation.

A behavior set B is represented in disjunctive normal form:

$$B = \bigcup_{i=1}^{m} \mathcal{Z}_i \tag{2}$$

This allows modeling of non-convex regions through finite unions while maintaining tractability. Set operations are implemented as follows:

- **Union**: $B_1 \cup B_2$ is realized by concatenating zonotope lists, with conservative merging applied when the number of zonotopes exceeds a threshold to bound computational complexity.
- **Intersection**: Computed pairwise using constraint-based zonotope intersection, producing an over-approximation when exact intersection is not representable as a zonotope.
- **Difference**: $B_1 \setminus B_2$ is computed using geometric zonotope subtraction, which partitions the zonotope into a covering set minus the removed region. This operation is critical for implementing contract strengthening.
- **Projection**: Given a zonotope with variable ordering, projection onto a subset of variables corresponds to selecting rows of the center vector and generator matrix.

2.2 Deviation Lattice

Contract evolution is modeled through a deviation structure $\delta = (\Delta A_{\text{rel}}, \Delta A_{\text{str}}, \Delta G_{\text{rel}}, \Delta G_{\text{str}})$, where:

- ΔA_{rel} (assumption relaxation): behaviors added to assumptions, representing acceptance of a wider range of inputs

- ΔA_{str} (assumption strengthening): behaviors removed from assumptions, imposing stricter requirements on the environment
- ΔG_{rel} (guarantee relaxation): behaviors added to guarantees, representing weakened performance promises
- ΔG_{str} (guarantee strengthening): behaviors removed from guarantees, representing improved performance promises

Each deviation component is itself a behavior set. The deviation structure forms a lattice under the component-wise subset ordering $\delta_1 \sqsubseteq \delta_2$ if and only if each component satisfies $\Delta X_1 \subseteq \Delta X_2$ for $X \in \{A_{\text{rel}}, A_{\text{str}}, G_{\text{rel}}, G_{\text{str}}\}$.

Contract reconstruction from a baseline contract $\mathcal{C}_0 = (A_0, G_0)$ and deviation δ follows the semantics:

$$A^* = (A_0 \cup \Delta A_{\text{rel}}) \setminus \Delta A_{\text{str}} \tag{3}$$

$$G^* = (G_0 \cup \Delta G_{\text{rel}}) \setminus \Delta G_{\text{str}} \tag{4}$$

This formulation ensures that relaxations expand the behavior set through union while strengthenings contract it through set difference. The evolved contract is $\mathcal{C}^* = (A^*, G^*)$.

2.3 Network Model

A contract network $\mathcal{N} = (\mathcal{V}, \mathcal{E})$ is a directed graph where:

- Each vertex $v \in \mathcal{V}$ represents a component with inputs $\text{In}(v)$, outputs $\text{Out}(v)$, and baseline contract $\mathcal{C}_v^0 = (A_v^0, G_v^0)$
- Each edge $e = (v_s, v_c, I) \in \mathcal{E}$ represents an interface from supplier v_s to consumer v_c over shared variables $I \subseteq \text{Out}(v_s) \cap \text{In}(v_c)$

Well-formedness of the network requires that for each interface (v_s, v_c, I), the consumer's assumptions (projected onto I) contain the supplier's guarantees (projected onto I):

$$G_{v_s}|_I \subseteq A_{v_c}|_I \tag{5}$$

where $B|_I$ denotes projection of behavior set B onto variables I. This condition ensures compositional reasoning: the consumer can rely on what the supplier guarantees.

The network structure may contain cycles, creating strongly connected components (SCCs). Cycles introduce feedback dependencies that require multiple iterations to resolve. The framework employs Tarjan's algorithm to identify SCCs and track propagation through cyclic structures.

3 Algorithmic Pipeline

3.1 Component Transformers

Each component implements two key transformers that enable propagation of contract changes:

- **Forward transformer (post)**: Maps input behaviors to output behaviors, computing $\text{post}(B_{\text{in}}) = B_{\text{out}}$ where B_{out} over-approximates all possible outputs given inputs in B_{in}.
- **Backward transformer (pre)**: Maps output behaviors to required input behaviors, computing $\text{pre}(B_{\text{out}}) = B_{\text{in}}$ where B_{in} under-approximates inputs that can achieve outputs in B_{out}.

These transformers are computed via MILP-based optimization. For each input or output zonotope region (axis-aligned box), the transformer solves a sequence of optimization problems to determine the extreme values of each output or input variable. Specifically, for the forward transformer:

Algorithm 1 Forward Transformer Computation

Require: Input behavior set $B_{\text{in}} = \{\mathcal{Z}_1, \ldots, \mathcal{Z}_m\}$
Ensure: Output behavior set B_{out}
  $B_{\text{out}} \leftarrow \emptyset$
  for each input zonotope $\mathcal{Z}_i$ **do**
    Convert $\mathcal{Z}_i$ to bounding box $[\ell_{\text{in}}, u_{\text{in}}]$
    for each output variable y **do**
      Solve: $y_{\min} \leftarrow \min y$ subject to component constraints and $x \in [\ell_{\text{in}}, u_{\text{in}}]$
      Solve: $y_{\max} \leftarrow \max y$ subject to component constraints and $x \in [\ell_{\text{in}}, u_{\text{in}}]$
    end for
    Construct output box $\mathcal{Z}_{\text{out}}$ from bounds $[y_{\min}, y_{\max}]$
    $B_{\text{out}} \leftarrow B_{\text{out}} \cup \{\mathcal{Z}_{\text{out}}\}$
  end for
  return B_{out}

The backward transformer follows an analogous procedure, optimizing over input variables given output constraints. Each optimization problem is formulated as a MILP with constraints encoding the component's physical behavior. For components with mode-dependent behavior, binary variables select the active mode, while linear constraints enforce mode-specific relationships between inputs and outputs.

3.2 Evolution Operator

The evolution operator $\Phi : \Delta \to \Delta$ propagates deviations through the network, where Δ is the global deviation map assigning a deviation δ_v to each component v. The operator implements two propagation rules:

Forward Propagation (Guarantee Relaxation): When a supplier relaxes its guarantees (ΔG_{rel}), this propagates forward to consumers:

1. The supplier's ΔG_{rel} projected onto interface I becomes the consumer's ΔA_{rel} (the consumer must accept the supplier's degraded output)
2. The consumer computes $\Delta G_{\text{rel}} = \text{post}(\Delta A_{\text{rel}})$, propagating the degradation to its own guarantees

Backward Propagation (Assumption Strengthening): When a consumer strengthens its assumptions (ΔA_{str}), this propagates backward to suppliers:

1. The consumer's ΔA_{str} projected onto interface I becomes the supplier's ΔG_{str} (the supplier must meet the consumer's stricter requirements)
2. The supplier computes $\Delta A_{\text{str}} = \text{pre}(\Delta G_{\text{str}})$, propagating the requirement to its own assumptions

Critically, only these two propagation paths exist in the framework. Assumption relaxation (ΔA_{rel}) does not trigger backward propagation because a consumer accepting more inputs does not impose requirements on suppliers. Similarly, guarantee strengthening (ΔG_{str}) does not trigger forward propagation because improved supplier performance does not force consumer changes.

The operator is implemented as:

Algorithm 2 Evolution Operator Φ

Require: Current deviation map Δ
Ensure: Updated deviation map Δ'

$\Delta' \leftarrow \text{copy}(\Delta)$
for each interface (v_s, v_c, I) in network **do**
  // Forward propagation
  if $\Delta G_{\text{rel}}^{v_s} \neq \emptyset$ **then**
    $\Delta A_{\text{rel}}^{v_c} \leftarrow \Delta A_{\text{rel}}^{v_c} \cup (\Delta G_{\text{rel}}^{v_s}|_I)$
    $\Delta G_{\text{rel}}^{v_c} \leftarrow \Delta G_{\text{rel}}^{v_c} \cup \text{post}_{v_c}(\Delta G_{\text{rel}}^{v_s}|_I)$
  end if
  // Backward propagation
  if $\Delta A_{\text{str}}^{v_c} \neq \emptyset$ **then**
    $\Delta G_{\text{str}}^{v_s} \leftarrow \Delta G_{\text{str}}^{v_s} \cup (\Delta A_{\text{str}}^{v_c}|_I)$
    $\Delta A_{\text{str}}^{v_s} \leftarrow \Delta A_{\text{str}}^{v_s} \cup \text{pre}_{v_s}(\Delta A_{\text{str}}^{v_c}|_I)$
  end if
end for
return Δ'

3.3 Fixpoint Iteration

The fixpoint engine computes the least fixpoint of the evolution operator starting from an initial deviation Δ_0 (typically a local change to a single component).

The iteration proceeds as:

$$\Delta_{k+1} = \Phi(\Delta_k), \quad k = 0, 1, 2, \ldots \tag{6}$$

Convergence is detected when $\Delta_{k+1} = \Delta_k$, indicating that no further propagation occurs. The framework employs structural equality checking: two deviation maps are equal if corresponding deviation components have the same number of zonotope regions with matching bounds.

Algorithm 3 Fixpoint Iteration

Require: Initial deviation Δ_0, maximum iterations $K_{\max}$
Ensure: Final deviation Δ^* at fixpoint

```
Δ ← Δ_0
for k = 1 to K_max do
    Δ' ← Φ(Δ)
    Record iteration metrics (magnitude, propagations, time)
    if Δ' = Δ then
        return Δ' // Fixpoint reached
    end if
    Δ ← Δ'
end for
return Δ // Maximum iterations reached
```

The iteration records metrics at each step, including total magnitude (sum of zonotope counts across all deviations), per-component magnitudes, breakdown by deviation type, and execution time. These metrics enable analysis of convergence rates and identification of components most affected by propagation.

3.4 Validation

After fixpoint convergence, two validation checks are performed:

Well-Formedness Checking: For each interface (v_s, v_c, I), the checker reconstructs the evolved contracts $\mathcal{C}_s^* = \mathcal{C}_s^0 \oplus \delta_s$ and $\mathcal{C}_c^* = \mathcal{C}_c^0 \oplus \delta_c$, then verifies:

$$G_s^*|_I \subseteq A_c^*|_I \tag{7}$$

using conservative subset checking on behavior sets. Violations indicate that the fixpoint has not fully restored well-formedness, which may occur if the initial change is too large or if the baseline contracts were not initially well-formed.

System-Level Contract Checking: A system-level contract $\mathcal{C}_{\text{sys}} = (A_{\text{sys}}, G_{\text{sys}})$ specifies end-to-end requirements. The checker computes the achieved system guarantee as the union of all component guarantees:

$$G_{\text{achieved}} = \bigcup_{v \in \mathcal{V}} G_v^* \tag{8}$$

and verifies $G_{\text{sys}} \subseteq G_{\text{achieved}}$. When this fails, the framework computes:

$$\text{Gap} = G_{\text{sys}} \setminus G_{\text{achieved}} \quad \text{(required but not achieved)} \tag{9}$$

$$\text{Violation} = G_{\text{achieved}} \setminus G_{\text{sys}} \quad \text{(achieved but not required)} \tag{10}$$

providing diagnostic information about the nature of the failure.

4 Component Mathematical Models

This section presents the detailed mathematical formulations for each component in the drone system. Each component is modeled as a mixed-integer linear program (MILP) that captures physical behavior through constraints relating input and output variables. These formulations enable the automated computation of forward (post) and backward (pre) transformers via optimization.

4.1 General Transformer Formulation

For any component with input variables $\mathbf{x} \in \mathbb{R}^{n_{\text{in}}}$, output variables $\mathbf{y} \in \mathbb{R}^{n_{\text{out}}}$, and constraint set $\mathcal{F}(\mathbf{x}, \mathbf{y})$, the forward and backward transformers are defined as follows.

4.1.1 Forward Transformer (Post)

Given an input behavior set $B_{\text{in}} = \{\mathcal{Z}_1, \ldots, \mathcal{Z}_m\}$, compute the output behavior set $B_{\text{out}} = \text{post}(B_{\text{in}})$ by solving:

$$\begin{aligned} y_j^{\min} = \min_{\mathbf{x},\mathbf{y}} \quad & y_j \\ \text{subject to} \quad & \mathbf{x} \in \mathcal{Z}_i, \quad (\mathbf{x}, \mathbf{y}) \in \mathcal{F} \end{aligned} \tag{11}$$

$$\begin{aligned} y_j^{\max} = \max_{\mathbf{x},\mathbf{y}} \quad & y_j \\ \text{subject to} \quad & \mathbf{x} \in \mathcal{Z}_i, \quad (\mathbf{x}, \mathbf{y}) \in \mathcal{F} \end{aligned} \tag{12}$$

for each input region $\mathcal{Z}_i$ and each output variable y_j. The resulting output region is the axis-aligned bounding box $[y_1^{\min}, y_1^{\max}] \times \cdots \times [y_{n_{\text{out}}}^{\min}, y_{n_{\text{out}}}^{\max}]$.

4.1.2 Backward Transformer (Pre)

Given an output behavior set $B_{\text{out}} = \{\mathcal{Z}_1, \ldots, \mathcal{Z}_m\}$, compute the input behavior set $B_{\text{in}} = \text{pre}(B_{\text{out}})$ by solving:

$$\begin{aligned} x_j^{\min} = \min_{\mathbf{x},\mathbf{y}} \quad & x_j \\ \text{subject to} \quad & \mathbf{y} \in \mathcal{Z}_i, \quad (\mathbf{x}, \mathbf{y}) \in \mathcal{F} \end{aligned} \tag{13}$$

$$\begin{aligned} x_j^{\max} = \max_{\mathbf{x},\mathbf{y}} \quad & x_j \\ \text{subject to} \quad & \mathbf{y} \in \mathcal{Z}_i, \quad (\mathbf{x}, \mathbf{y}) \in \mathcal{F} \end{aligned} \tag{14}$$

for each output region $\mathcal{Z}_i$ and each input variable x_j. The resulting input region is the axis-aligned bounding box $[x_1^{\min}, x_1^{\max}] \times \cdots \times [x_{n_{\text{in}}}^{\min}, x_{n_{\text{in}}}^{\max}]$.

4.2 Battery Model

Inputs: $p_m \in [0, 3]$ (power mode)

Outputs: $V_b \in [9.5, 12.6]$ (battery voltage, V), $I_b \in [0, 40]$ (battery current, A), $S \in [20, 100]$ (state of charge, %)

Physical Model: 3-cell LiPo battery with state-of-charge dependent open-circuit voltage, internal resistance, and mode-dependent current protection.

Constraint Set $\mathcal{F}_{\textbf{Battery}}$:

$$\text{(Open-circuit voltage)} \quad V_{\text{oc}} = f_{\text{pwl}}(S) \tag{15}$$

$$\text{(Loaded voltage)} \quad V_b = V_{\text{oc}} - R_{\text{int}} \cdot I_b \tag{16}$$

$$\text{(Mode selection)} \quad \sum_{k=0}^{3} \beta_k = 1, \quad \beta_k \in \{0, 1\} \tag{17}$$

$$\text{(Mode encoding)} \quad p_m = \sum_{k=0}^{3} k \cdot \beta_k \tag{18}$$

$$\text{(Current limit)} \quad I_b \leq I_{\text{cap}}^k + M(1 - \beta_k), \quad \forall k \in \{0, 1, 2, 3\} \tag{19}$$

where $R_{\text{int}} = 0.06\,\Omega$, $M = 1000$ is a big-M constant, $I_{\text{cap}}^0 = 40\text{A}$, $I_{\text{cap}}^1 = 30\text{A}$, $I_{\text{cap}}^2 = 22\text{A}$, $I_{\text{cap}}^3 = 15\text{A}$, and $f_{\text{pwl}} : [20, 100] \to [10.8, 12.6]$ is a piecewise linear function defined by breakpoints:

$$f_{\text{pwl}}(S) = \begin{cases} 10.8 + 0.015(S - 20) & 20 \leq S < 40 \\ 11.1 + 0.015(S - 40) & 40 \leq S < 60 \\ 11.4 + 0.03(S - 60) & 60 \leq S < 80 \\ 12.0 + 0.03(S - 80) & 80 \leq S \leq 100 \end{cases} \tag{20}$$

4.3 PowerManager Model

Inputs: $I_m \in [0, 50]$ (motor current, A), $V_b \in [9.5, 12.6]$ (battery voltage, V), $I_b \in [0, 40]$ (battery current, A)

Outputs: $V_a \in [9.0, 12.6]$ (voltage available, V), $p_m \in [0, 3]$ (power mode), $V_{\text{margin}} \in [-1.2, 2.4]$ (voltage margin, V)

Physical Model: Power distribution with resistive voltage drop, mode-based protection thresholds, and current coupling.

Constraint Set $\mathcal{F}_{\textbf{PowerManager}}$:

$$\text{(Voltage delivery)} \quad V_a = V_b - R_m \cdot I_m \tag{21}$$

$$\text{(Current coupling)} \quad I_b \geq I_m + I_{\text{base}} \tag{22}$$

$$\text{(Voltage margin)} \quad V_{\text{margin}} = V_a - V_{\min} \tag{23}$$

$$\text{(Mode selection)} \quad \sum_{k=0}^{3} \beta_k = 1, \quad \beta_k \in \{0,1\} \tag{24}$$

$$\text{(Mode encoding)} \quad p_m = \sum_{k=0}^{3} k \cdot \beta_k \tag{25}$$

$$\text{(Mode thresholds)} \quad V_{\text{margin}} \geq \tau_k^{\min} - M(1-\beta_k), \quad \forall k \tag{26}$$

$$V_{\text{margin}} \leq \tau_k^{\max} + M(1-\beta_k), \quad \forall k \tag{27}$$

where $R_m = 0.08\,\Omega$, $I_{\text{base}} = 2\text{A}$, $V_{\min} = 10.2\text{V}$, and mode thresholds are: $\tau_0 = [0.6, \infty)$, $\tau_1 = [0.3, 0.6)$, $\tau_2 = [0.1, 0.3)$, $\tau_3 = (-\infty, 0.1)$.

4.4 Motor Model

Inputs: $T_{\text{cmd}} \in [0, 100]$ (thrust command, N), $V_a \in [9.0, 12.6]$ (voltage available, V)

Outputs: $T_m \in [0, 100]$ (motor thrust, N), $I_m \in [0, 50]$ (motor current, A), $t_r \in [0.01, 3.0]$ (response time, s)

Physical Model: Brushless DC motor with voltage-dependent efficiency bands, current draw proportional to thrust and voltage deficit, and response time degradation with low voltage and high current.

Constraint Set $\mathcal{F}_{\text{Motor}}$:

$$\text{(Efficiency bands)} \quad \sum_{k=0}^{2} \beta_k = 1, \quad \beta_k \in \{0,1\} \tag{28}$$

$$\text{(Band 0: } V_a \geq 11.5\text{)} \quad V_a \geq 11.5 - M(1-\beta_0) \tag{29}$$

$$\text{(Band 1: } 10.5 \leq V_a < 11.5\text{)} \quad V_a \geq 10.5 - M(1-\beta_1), \quad V_a \leq 11.5 + M(1-\beta_1) \tag{30}$$

$$\text{(Band 2: } V_a < 10.5\text{)} \quad V_a \leq 10.5 + M(1-\beta_2) \tag{31}$$

$$\text{(Thrust with efficiency)} \quad T_m = \eta_k \cdot T_{\text{cmd}} \text{ when } \beta_k = 1 \tag{32}$$

$$\text{(Voltage deficit)} \quad V_{\text{def}} \geq V_{\text{nom}} - V_a, \quad V_{\text{def}} \geq 0 \tag{33}$$

$$\text{(Current draw)} \quad I_m \geq 0.5T_m + 2V_{\text{def}} \tag{34}$$

$$I_m \leq 0.6T_m + 2.5V_{\text{def}} + 1 \tag{35}$$

$$\text{(Response time)} \quad t_r \geq t_{\text{base}} + K_V(V_{\text{nom}} - V_a) + K_I I_m \tag{36}$$

$$t_r \leq t_{\text{base}} + K_V(V_{\text{nom}} - V_a) + K_I I_m + 0.05 \tag{37}$$

where $\eta_0 = 1.0$, $\eta_1 = 0.9$, $\eta_2 = 0.8$ are efficiency factors, $V_{\text{nom}} = 12\text{V}$, $t_{\text{base}} = 0.05\text{s}$, $K_V = 0.05\text{s/V}$, $K_I = 0.01\text{s/A}$.

4.5 FlightController Model

Inputs: $T_m \in [0, 100]$ (motor thrust, N), $t_r \in [0.01, 3.0]$ (motor response time, s), $e_{\text{nav}} \in [0, 50]$ (navigation position error, m), $p_m \in [0, 3]$ (power mode)

Outputs: $T_{\text{cmd}} \in [0, 100]$ (thrust command, N), $e_c \in [0, 100]$ (control error)

Physical Model: Mode-dependent thrust authority controller with saturation tracking and error composition from navigation error, response time lag, and control saturation effects.

Constraint Set $\mathcal{F}_{\textbf{FlightController}}$:

$$\text{(Demanded thrust)} \quad T_{\text{dem}} = K_{\text{nav}} \cdot e_{\text{nav}} \tag{38}$$

$$\text{(Mode selection)} \quad \sum_{k=0}^{3} \beta_k = 1, \quad \beta_k \in \{0, 1\} \tag{39}$$

$$\text{(Mode encoding)} \quad p_m = \sum_{k=0}^{3} k \cdot \beta_k \tag{40}$$

$$\text{(Authority limit)} \quad T_{\text{cmd}} \leq A_k + M(1 - \beta_k), \quad \forall k \in \{0, 1, 2, 3\} \tag{41}$$

$$\text{(Demand limit)} \quad T_{\text{cmd}} \leq T_{\text{dem}} \tag{42}$$

$$\text{(Saturation slack)} \quad s_{\text{sat}} \geq T_{\text{dem}} - T_{\text{cmd}}, \quad s_{\text{sat}} \geq 0 \tag{43}$$

$$\text{(Control error)} \quad e_c \geq e_{\text{nav}} + K_r t_r + K_s s_{\text{sat}} \tag{44}$$

$$e_c \leq e_{\text{nav}} + K_r t_r + K_s s_{\text{sat}} + 0.5 \tag{45}$$

where $K_{\text{nav}} = 2.0$, $A_0 = 100$N, $A_1 = 85$N, $A_2 = 65$N, $A_3 = 45$N are mode-dependent authority limits, $K_r = 5.0$ and $K_s = 0.2$ are error contribution coefficients.

4.6 NavigationEstimator Model

Inputs: $e_c \in [0, 100]$ (control error), $I_m \in [0, 50]$ (motor current, A), $p_m \in [0, 3]$ (power mode)

Outputs: $e_{\text{nav}} \in [0, 50]$ (navigation position error, m), $d_{\text{nav}} \in [0, 10]$ (navigation drift, m/s)

Physical Model: 10-tier sensor degradation model where tier activation depends on motor current and power mode. Higher tiers represent worse sensor conditions with increased position error and drift.

Constraint Set $\mathcal{F}_{\textbf{NavigationEstimator}}$:

$$\text{(Tier selection)} \quad \sum_{k=0}^{9} \beta_k = 1, \quad \beta_k \in \{0,1\} \tag{46}$$

$$\text{(Tier level)} \quad \ell = \sum_{k=0}^{9} k \cdot \beta_k, \quad \ell \in \mathbb{Z},\, 0 \le \ell \le 9 \tag{47}$$

$$\text{(Activation: current)} \quad \gamma_k^I = \mathbb{I}(I_m \ge I_{\text{base}} + k \cdot I_{\text{step}}), \quad \gamma_k^I \in \{0,1\} \tag{48}$$

$$\text{(Activation: mode)} \quad \gamma_k^p = \mathbb{I}(p_m \ge k/3), \quad \gamma_k^p \in \{0,1\} \tag{49}$$

$$\text{(Activation: OR)} \quad \gamma_k = \gamma_k^I \vee \gamma_k^p \tag{50}$$

$$\text{(Tier constraint)} \quad \ell \ge k - M(1-\gamma_k), \quad \forall k \tag{51}$$

$$\text{(Position error bounds)} \quad e_{\text{nav}} \ge e_{\text{base}}^{\min} + k \cdot \Delta e - M(1-\beta_k), \quad \forall k \tag{52}$$

$$e_{\text{nav}} \le e_{\text{base}}^{\max} + k \cdot \Delta e + M(1-\beta_k), \quad \forall k \tag{53}$$

$$\text{(Drift bounds)} \quad d_{\text{nav}} \ge d_{\text{base}}^{\min} + k \cdot \Delta d - M(1-\beta_k), \quad \forall k \tag{54}$$

$$d_{\text{nav}} \le d_{\text{base}}^{\max} + k \cdot \Delta d + M(1-\beta_k), \quad \forall k \tag{55}$$

$$\text{(Control coupling)} \quad e_{\text{nav}} \ge 0.5 e_c \tag{56}$$

where $I_{\text{base}} = 4\text{A}$, $I_{\text{step}} = 2\text{A}$, $e_{\text{base}} = [0.5, 3.0]\text{m}$, $d_{\text{base}} = [0.0, 0.5]\text{m/s}$, $\Delta e = 0.8\text{m}$ per tier, $\Delta d = 0.15\text{m/s}$ per tier, and $\mathbb{I}(\cdot)$ denotes the indicator function implemented via big-M constraints.

4.7 Optimization Problem Summary

For each component, the post and pre transformers are computed by solving a sequence of linear or mixed-integer linear programs with the constraint sets defined above. The transformers over-approximate reachable behaviors due to the use of axis-aligned bounding boxes and conservative linearizations of bilinear terms. This ensures soundness: if the actual component can produce an output given an input, the transformer will include that output in its result (though it may also include infeasible outputs as over-approximations).

The MILP formulations capture discrete mode transitions through binary variables β_k with big-M constraints that enforce mode-specific bounds and relationships. The choice of $M = 1000$ is sufficiently large to deactivate constraints for inactive modes while remaining small enough to maintain numerical stability in the solver. All optimization problems are solved using the CBC (Coin-or Branch and Cut) solver or Gurobi when available, with a 5-second timeout per optimization.

5 Implementation Architecture

5.1 Software Structure

The implementation is structured in Python with the following module organization:

Contracts Module (`src/contracts/`): Implements the core contract abstractions. The `behavior.py` module defines the `BehaviorSet` class with zonotope-based operations (union, intersection, difference, projection). The `contract.py` module defines the `Contract` class with projection methods. The `deviation.py` module implements the `Deviation` and `DeviationMap` classes with lattice operations and the `reconstruct_contract` function.

Zonotope Operations Module (`src/zonotope_ops.py`): Provides low-level geometric operations. The `Zonotope` class implements generator-form representation with methods for construction from boxes, bounding box computation, containment checking, and vertex enumeration. Key operations include `zonotope_union` (computing convex hull), `zonotope_intersection` (constraint-based intersection), and `zonotope_subtract` (geometric difference).

Components Module (`src/components/`): Implements physical components as subclasses of `BaseComponent`. Each component defines input/output variables, MILP constraints via `get_constraints`, and inherits `post` and `pre` transformers from the base class. The base class orchestrates MILP solving using the PuLP library with CBC or Gurobi solvers.

Network Module (`src/network/`): Implements the compositional structure. The `contract_network.py` module defines `ContractNetwork` with graph operations and cycle detection. The `interface.py` module defines typed interfaces between components. The `evolution.py` module implements the `EvolutionOperator` and `FixpointEngine`. The `validation.py` module provides `WellFormednessChecker` and `SystemLevelChecker`.

Scenarios Module (`src/scenarios/`): Defines experimental scenarios that instantiate networks, configure initial deviations, and specify system-level contracts.

Main Entry Point (`main.py`): Orchestrates scenario execution, including network construction, fixpoint iteration, validation, and output generation (reports, traces, visualizations).

5.2 Component Physical Models

The drone system comprises five components with physics-based MILP models:

Battery: A 3-cell LiPo battery (11.5-12.6V nominal) with state-of-charge (SOC) dependent voltage output and current capacity limits. The model uses binary variables to select operating modes and linear constraints to enforce voltage-current-SOC relationships.

PowerManager: Implements a 4-mode state machine (Normal, Degraded, Critical, Emergency) with voltage and current thresholds. Mode selection uses binary variables with big-M constraints. Voltage drop due to internal resistance is modeled linearly. Outputs include available voltage (accounting for losses) and voltage margin indicators.

Motor: A brushless DC motor with voltage-dependent efficiency (three bands: optimal, degraded, critical). Current draw depends on thrust demand and voltage deficit. Response time increases with lower voltage and higher

current. The model uses three binary variables for band selection with bilinear constraint linearization via big-M.

FlightController: Authority-based controller with 4 tiers depending on navigation error and power mode. Control error and thrust command increase with tier. The model includes logical constraints linking tier selection to input conditions and output bounds.

NavigationEstimator: A 10-tier sensor model where tier selection depends on control error, motor current, and power mode. Position error and drift increase with tier. The model employs 10 binary variables with constraints partitioning the input space into tier regions.

All components implement baseline contracts that characterize nominal operating regions. These contracts are instantiated as zonotope-based behavior sets with one or a few regions per assumption/guarantee.

5.3 Data Flow and Execution

The execution pipeline follows these stages:

1. **Scenario Instantiation**: A scenario creates the contract network by adding components with baseline contracts and interfaces. It defines the initial deviation (typically on a single component) and optionally a system-level contract.

2. **Baseline Validation**: Well-formedness of baseline contracts is checked to ensure the network is valid before evolution.

3. **Component Initialization**: Component instances are created to provide post and pre transformers. These are passed to the evolution operator.

4. **Fixpoint Iteration**: The engine applies the evolution operator repeatedly, recording iteration history (deviation maps, metrics) until convergence or maximum iterations.

5. **Contract Reconstruction**: Final contracts are reconstructed from baseline contracts and final deviations using the reconstruction semantics.

6. **Validation**: Well-formedness of evolved contracts is checked, and system-level contract satisfaction is verified with gap/violation computation if applicable.

7. **Output Generation**: The framework produces detailed text reports with iteration-by-iteration traces, summary metrics, and validation results. Visualizations include network diagrams annotated with deviations, iteration analytics plots (magnitude vs. iteration), and delta breakdowns (stacked bar charts showing deviation types).

5.4 Testing and Validation

The implementation includes several testing layers:

MILP Model Tests (`test_milp_models.py`): Validates that component MILP formulations are solvable and produce feasible solutions for representative inputs. Each component model is instantiated, constraints are added to a test problem, inputs are fixed, and the solver is invoked to verify optimality.

Unit Tests (`tests/`): Includes tests for behavior set operations (`test_behavior.py`), contract reconstruction semantics (`test_reconstruction.py`), and well-formedness checking (`test_well_formedness.py`). These tests verify correctness of low-level operations using small synthetic examples with known expected outcomes.

Integration Tests: The main execution validates end-to-end behavior by running complete scenarios and checking convergence, iteration counts, and well-formedness results against documented expectations.

6 Use Case: Autonomous Drone System

6.1 System Description

The use case demonstrates contract evolution on a quadrotor drone system with five interconnected components arranged in a topology containing a 3-node strongly connected component (SCC). The network structure is:

$$\begin{aligned}
\text{Battery} &\to \text{PowerManager} \to \text{Motor} \to \text{FlightController} \\
\text{FlightController} &\to \text{Motor} \quad \text{(feedback)} \\
\text{Motor} &\to \text{PowerManager} \quad \text{(feedback)} \\
\text{PowerManager} &\to \text{FlightController} \quad \text{(completes 3-SCC)} \\
\text{NavigationEstimator} &\to \text{FlightController} \\
\text{FlightController} &\to \text{NavigationEstimator} \quad \text{(feedback)}
\end{aligned}$$

The 3-node SCC (FlightController $\to$ Motor $\to$ PowerManager $\to$ FlightController) creates a feedback loop where voltage degradation affects motor performance, which affects control quality, which affects power demand, cycling back through the loop. The NavigationEstimator forms a separate feedback pair with the FlightController.

Baseline contracts for each component capture nominal operating ranges:

- Battery: SOC $\in [60, 100]\%$, voltage $\in [11.5, 12.6]$V, current $\in [0, 30]$A
- PowerManager: Mode $\in [0, 1]$, voltage available $\in [10.5, 12.6]$V, margin $\in [0.3, 2.4]$V
- Motor: Thrust $\in [0, 25]$N, current $\in [0, 15]$A, response time $\in [0.05, 0.5]$s
- FlightController: Control error $\in [0, 15]$, thrust command $\in [0, 30]$N
- NavigationEstimator: Position error $\in [0.5, 6]$m, drift $\in [0, 1]$m/s

6.2 Scenario 1: Motor Upgrade (Backward Propagation)

Physical Narrative: The motor is upgraded to a higher-quality model with improved bearings, tighter tolerances, and better efficiency. The upgraded motor has stricter requirements (demands cleaner power and more precise control) but provides better performance (tighter response time and current bounds).

Deviation Model: The initial deviation is pure strengthening on the Motor component:

$$\Delta A_{\text{str}} : \text{thrust\_command} \in [0, 30] \to [5, 25],\ \text{voltage\_available} \in [10.5, 12.6] \to [11.5, 12.6]$$
$$\Delta G_{\text{str}} : \text{motor\_thrust} \in [0, 25] \to [5, 25],\ \text{motor\_current} \in [0, 15] \to [2, 12],$$
$$\text{response\_time} \in [0.05, 0.5] \to [0.08, 0.35]$$

The initial magnitude is 22 (sum of zonotope counts across ΔA_{str} and ΔG_{str}).

Propagation Pattern: The scenario demonstrates pure backward propagation. The Motor's ΔA_{str} (stricter voltage requirement) propagates to PowerManager, forcing it to strengthen guarantees (ΔG_{str}). The PowerManager's resulting ΔA_{str} (needs better battery performance) propagates to Battery. Similarly, the Motor's stricter thrust command assumption propagates to FlightController, which must strengthen its guarantee. The FlightController's ΔA_{str} on navigation inputs propagates to NavigationEstimator.

Experimental Results: The fixpoint iteration converges in 7 iterations with total execution time of 55.8 seconds. The final magnitude reaches 179, with all 5 components exhibiting deviations. The breakdown shows exclusive backward propagation: every component has non-zero ΔA_{str} and ΔG_{str} while $\Delta A_{\text{rel}} = \Delta G_{\text{rel}} = \emptyset$ throughout.

However, the final well-formedness check fails, indicating that the reconstruction operator (using zonotope difference for strengthening) produces empty or insufficient assumption sets. This is a known issue with the difference operation on complex zonotope unions and represents a direction for future refinement of the geometric operations.

The system-level contract (requiring battery voltage $\geq$ 11V, control error $\leq$ 15, response time $\leq$ 0.4s, position error $\leq$ 8m) is not satisfied, with a gap of 1 box indicating that the evolved system cannot meet all requirements simultaneously after the motor upgrade forces tighter constraints.

6.3 Scenario 2: Navigation Drift Increase (Forward Propagation)

Physical Narrative: The NavigationEstimator's sensors degrade due to GPS antenna corrosion, IMU calibration drift, and electromagnetic interference. The result is worse position estimates with larger errors and increased drift.

Deviation Model: The initial deviation is pure guarantee relaxation on

the NavigationEstimator:

ΔG_{rel} : Three staged boxes modeling progressive degradation:

Stage 1: position_error $\in [0.5, 6] \to [0.5, 8]$, drift $\in [0, 1] \to [0, 1.5]$

Stage 2: position_error $\in [5, 12]$, drift $\in [0.8, 2.5]$

Stage 3: position_error $\in [8, 15]$, drift $\in [1.5, 3.5]$

The initial magnitude is 7 (baseline plus 3 new regions with overlaps removed).

Propagation Pattern: The scenario demonstrates pure forward propagation. NavigationEstimator's ΔG_{rel} (worse position errors) propagates to FlightController, forcing it to relax assumptions (ΔA_{rel}). The FlightController's post transformer computes the resulting ΔG_{rel} (worse control error and higher thrust). This cascades to Motor (higher current draw), then to PowerManager (lower voltage, worse modes). The degraded voltage cycles back to Motor through the 3-SCC, and worse power modes feed back to both FlightController and NavigationEstimator, progressively activating higher sensor tiers.

Experimental Results: The fixpoint iteration converges in 12 iterations with total execution time of 90.5 seconds. The final magnitude reaches 176, with all 5 components showing substantial deviations. The breakdown confirms pure forward propagation: every component has non-zero ΔA_{rel} and ΔG_{rel} while $\Delta A_{\text{str}} = \Delta G_{\text{str}} = \emptyset$ throughout.

The final well-formedness check passes, indicating that the network has successfully adapted to the sensor degradation through cascading relaxations that restore the well-formedness invariant. The final contracts show:

- Battery: 17 assumption boxes, 17 guarantee boxes (expanded operating range)
- PowerManager: 17 assumption boxes, 17 guarantee boxes (accepts higher currents, provides degraded voltage)
- Motor: 18 assumption boxes, 17 guarantee boxes (operates at degraded conditions)
- FlightController: 20 assumption boxes, 17 guarantee boxes (tolerates worse navigation, provides coarser control)
- NavigationEstimator: 20 assumption boxes, 17 guarantee boxes (accepts feedback, provides degraded estimates)

The system-level contract is not satisfied, with a gap of 1 box and violations of 20 boxes. The gap indicates that some required behaviors (tight position error bounds with good power modes) are no longer achievable after sensor degradation. The violations reflect the expanded operating regions that now include degraded states not present in the baseline requirements.

6.4 Complementary Nature of Scenarios

The two scenarios demonstrate perfect duality in the evolution operator:

Property	**Motor Upgrade**	**Nav Drift Increase**
Initial Component	Motor	NavigationEstimator
Deviation Type	$\Delta A_{\text{str}} + \Delta G_{\text{str}}$	ΔG_{rel}
Physical Cause	Component upgrade	Sensor degradation
Propagation	Backward only	Forward only
Iterations	7	12
Final Magnitude	179	176
Final Well-Formedness	Failed	Passed
Propagation Path	$\Delta A_{\text{str}} \to \Delta G_{\text{str}}$	$\Delta G_{\text{rel}} \to \Delta A_{\text{rel}}$

The Motor Upgrade scenario isolates backward propagation through assumption strengthening, showing how a component demanding better inputs forces upstream suppliers to improve. The Navigation Drift scenario isolates forward propagation through guarantee relaxation, showing how component degradation cascades downstream as consumers must tolerate worse inputs. Together, they provide comprehensive empirical validation of the two propagation rules in the evolution operator.

The presence of the 3-node SCC creates multi-iteration feedback loops in the forward propagation case (12 iterations vs. 7 in the backward case), demonstrating the framework's ability to handle cyclic dependencies and converge despite the presence of strongly connected components.

7 Conclusion

This implementation demonstrates a complete computational framework for contract-based design with support for incremental evolution through fixpoint iteration. The use of zonotope-based geometric representations enables exact modeling of non-convex behavior sets while maintaining computational tractability. The MILP-based component models capture realistic physical behaviors with mode-dependent dynamics and enable automated computation of forward and backward transformers. The evolution operator correctly implements the two propagation rules (forward through guarantee relaxation, backward through assumption strengthening), and the fixpoint iteration successfully converges in the presence of feedback cycles.

The drone system use case validates the framework on a realistic cyber-physical system with 5 components, 11 interfaces, and a 3-node SCC. The two experimental scenarios demonstrate pure backward and pure forward propagation, respectively, providing clear empirical evidence of the duality in the evolution operator. Iteration counts (7-12) confirm the non-trivial nature of the propagation through the cyclic network structure. Well-formedness and system-level checking provide rigorous validation of the evolved contracts.

The implementation provides comprehensive outputs including iteration-by-iteration traces, convergence analytics, and diagnostic information (gap/violation computation) that support understanding of the evolution process. The framework is publicly available and documented for reproducibility.