Limiting Run-time Behavior to Improve the Verification of Autonomous Systems

Bjorn Andersson, Dionisio de Niz, and Gabriel Moreno

Why Troops Don't Trust Drones

White House Drone Crash Described as a U.S. Worker's Drunken Lark

17 drone disasters that show why the FAA hates drones

Boeing 737 Max: Software patches can only do so much

Systems architects, engineers, and management can all learn from the history of the development of this complex aircraft.

RQ-4B GLOBAL HAWK ACCIDENT INVESTIGATION RELEASED

Drone Crash in Iran Reveals Secret U.S. Surveillance Effort

Robotic surgery linked to 144 deaths in the US

Faulty pacemakers 'killing 2,000 a year': Third of unexpected deaths among patients thought to be caused by malfunctions

Death by robot: the new mechanised danger in our changing world

As the use of autonomous machines increases in society, so too has the chance of robot-related fatalities

ROBOT CANNON KILLS 9, WOUNDS 14

Robot 'goes rogue and kills woman on Michigan car parts production line'

The value that autonomous systems generate to society is limited by their lack of safety.

Technical/Scientific Questions

How do we know that an autonomous system is safe?

How do we know that the software computes the right result?

How do we know that the software computes the result at the right time?

How do we verify a system composed of both software and its physical environment (Cyber-Physical Systems (CPS)) ?

How can we do offline verification of the software in an autonomous system when we do not know precisely its physical environment?

How can we verify systems that use AI/ML techniques?

How can we design systems to continue to perform *some* function (maintain safety) even in the presence of cyber-attacks?

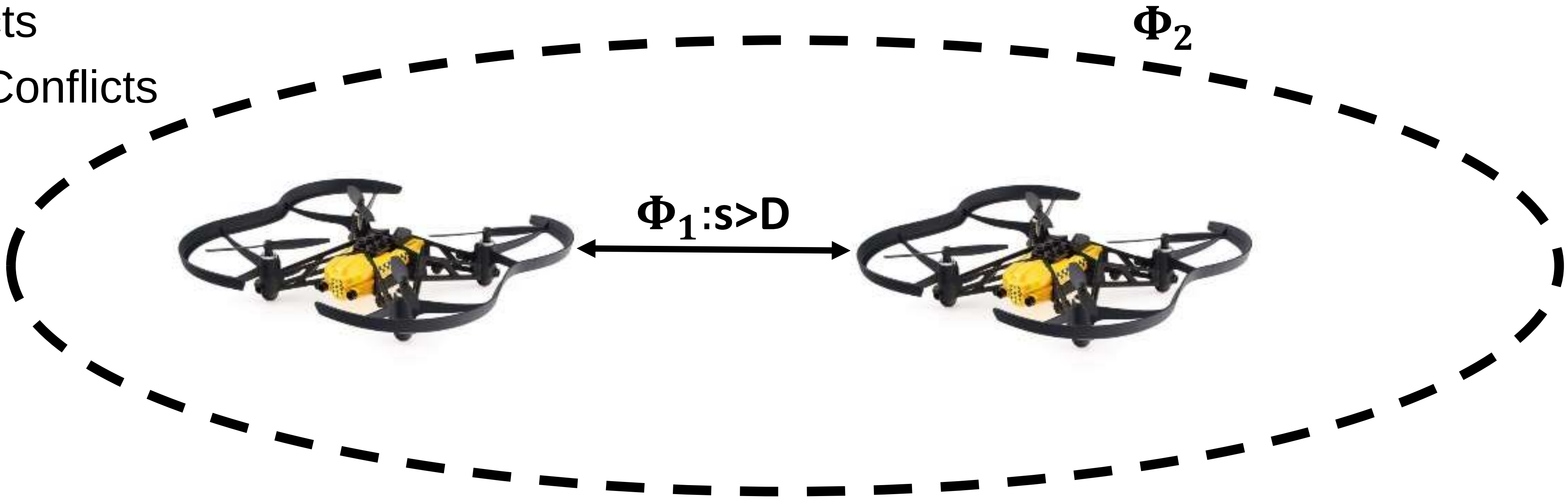
Runtime Assurance Key for Safety-Critical Autonomous-Systems

Constraining Behavior (satisfies Φ) with Enforcers

- Verifiable Constrained Behavior
- Verifiable Enforcer Implementation

Multiple Enforcers Φ_1, Φ_2:

- Identify Conflicts
- Resolution of Conflicts



Our Work

Formalization of Enforcers Operating on Same Actuators

Algorithms to Combine Enforcers (called ***select***)

- By Priority (***select***)

Enforcers Encoding in SMT Formulae

- ***select*** implementation with SMT
- Succinct SMT enforcer encoding
- ***select*** implementation as logical operations

Online SMT enforcer implementation

- Incremental solving
- Push/Pop Formulae
- Dynamic Context

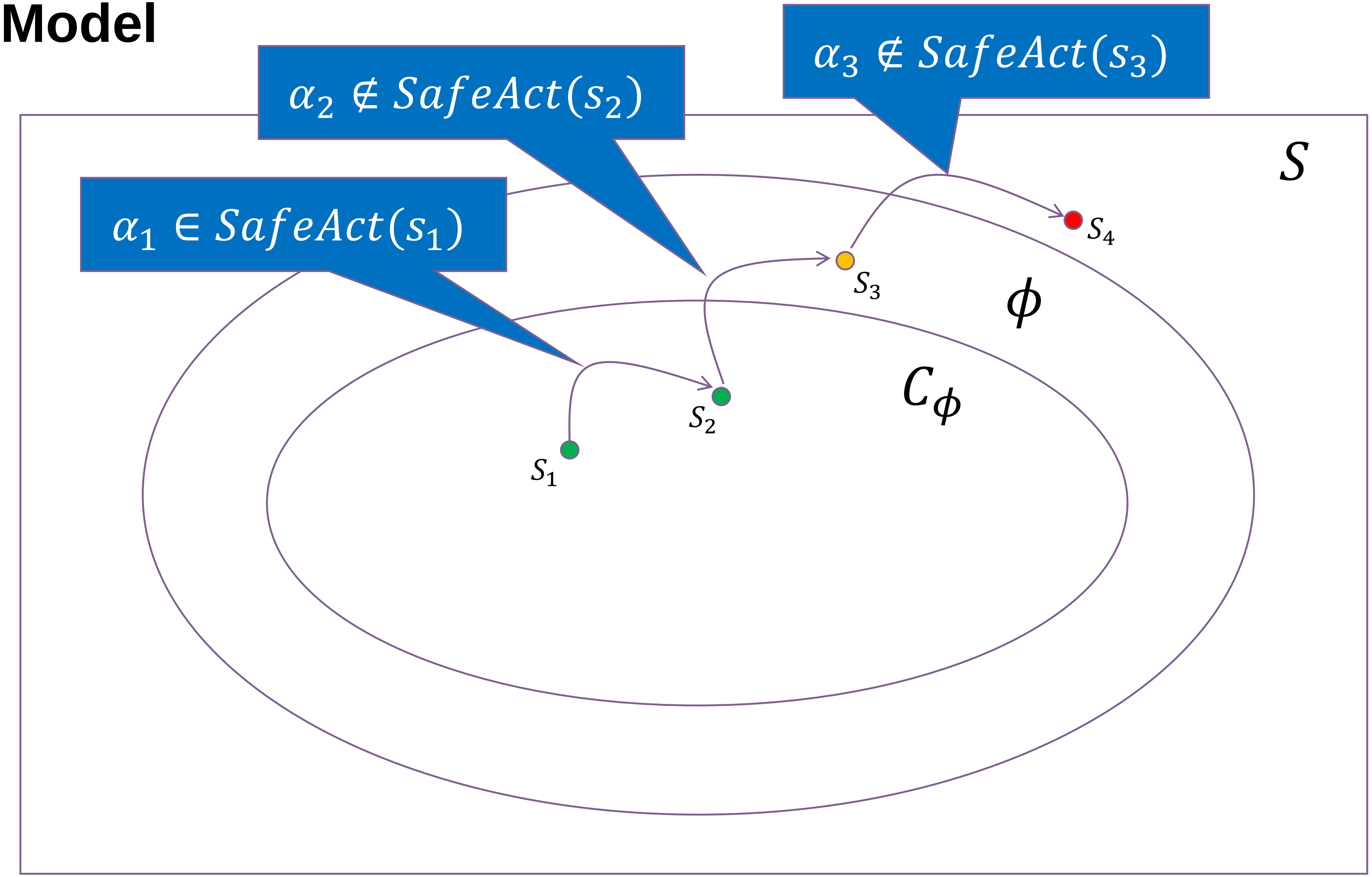
Experiments with Drones

Formal Periodic Model: Representing Time-Aware Logic

State of the system: values of variables

- State variables: V_S
- Action variables: V_Σ
- Variable values from domain: D
- System state: state variable: $s: V_S \mapsto D \in S$
- Actions: action variables valuations: $\alpha: V_\Sigma \mapsto D$
- Behavior: state transitions given actuation every period P: $R_P(\alpha) \subseteq S \times S$
  - Next state given action: $R_P(\alpha, s) = \{s' | (s, s') \in R_P(\alpha)\}$
- Property to verify subset of all possible states: $\phi \subseteq S$
- Enforceable state: $C_\phi \subseteq \phi \wedge C_\phi = \{s \mid \exists \alpha \in \Sigma: \mathrm{R_P}(\alpha, s) \in C_\phi\}$
- Safe actuation : $SafeAct(s) = \{\alpha | R_P(\alpha, s) \in C_\phi\}$

Formal Model



Enforcer Definition

Enforcer: $E = \left(P, C_{\phi}, \mu, U\right)$

- P: period of the enforcer
- C_{ϕ}: set of ϕ-enforceable states
- $\mu\!:\mathrm{C}_{\phi} \mapsto 2^{\Sigma}$: mapping from enforceable states to actions
  - $\forall s \in C_{\phi} \cdot \mu(s) \subseteq SafeAct(s)$
- $U\!:\mathrm{C}_{\phi} \times \Sigma \hookrightarrow \mathbb{R}$ maps each ϕ-enforceable state and its corresponding actuation to its utility
  - $Dom(U) = \{(s, \alpha) | \left(s \in C_{\phi}\right) \wedge (\alpha \in \mu(s))\}$

Utility Agnostic Enforcer Operation

$$\tilde{\alpha} = \begin{cases} \alpha & if\ \alpha \in \mu(s) \\ pick\big(\mu(s)\big) & otherwise \end{cases}$$

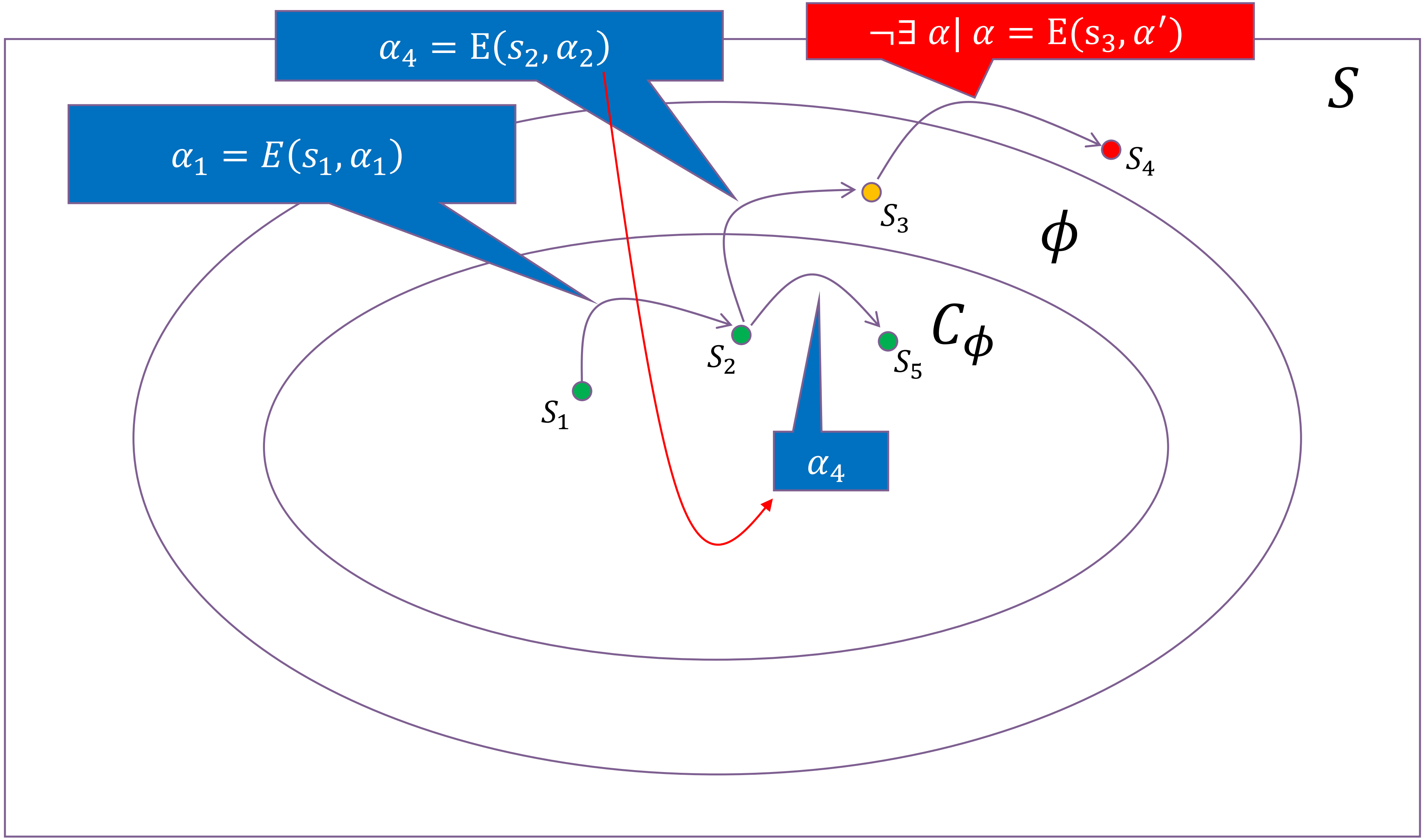
where:

$pick(X)$: arbitrary element of X

α :actuation from unverified software

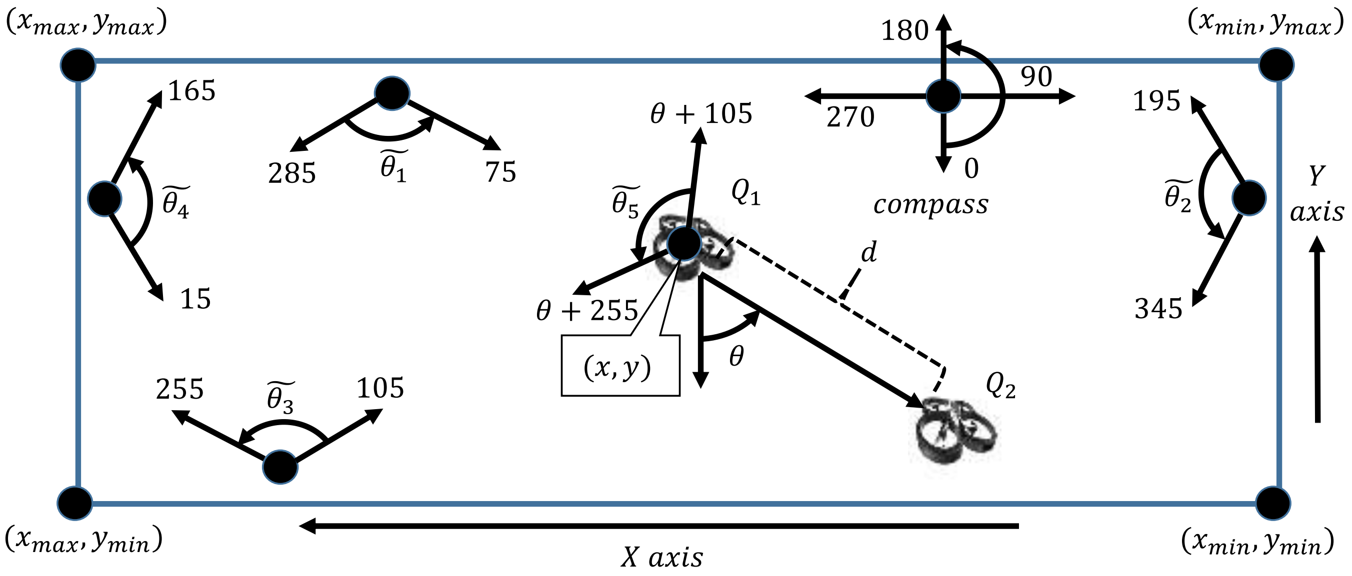
s: state of system before α

Enforcer



$\alpha_4 = \mathrm{E}(s_2, \alpha_2)$
$\neg\exists\ \alpha |\ \alpha = \mathrm{E}(s_3, \alpha')$
S
$\alpha_1 = E(s_1, \alpha_1)$
s_4
s_3
ϕ
C_ϕ
s_2
s_5
s_1
α_4

Example



Quadrotors Q_1, Q_2

State Variables: $V_S = \{x, y, \theta, d\}$

Action $V_\Sigma = \{\theta_\alpha\}$: move in direction θ_α

Z: Virtual Fence Zone

$C_{\phi_1} = \{(x, y, \theta, d) | (x + \delta_{B1}, y + \delta_{B1}) \in Z \wedge (x - \delta_{B1}, y - \delta_{B1}) \in Z\}$

- δ_{B1}: braking distance

$C_{\phi_2} = \{(x, y, \theta, d) | d + \delta_{B2} \geq D\}$

- δ_{B2}: largest reduction in d once separation enforcement applied

Combining Enforcers: Priority

$$select(s, \langle E_1 \rangle, \alpha) = \begin{cases} \alpha, & \alpha \in \mu_1(s) \\ pick(\mu_1(s)), & otherwise \end{cases}$$

$$select(s, \langle E_1, \dots, E_n \rangle, \alpha) = \begin{cases} \alpha, & s \in \bigcap_{i=2,\dots n} C_{\phi_i} \wedge \alpha \in \bigcap_{i=1,\dots n} \mu_i(s) \\ pick(\bigcap_{i=1,\dots,n} \mu_i(s)), & s \in \bigcap_{i=2,\dots,n} C_{\phi_i} \wedge \bigcap_{i=1,\dots n} \mu_i(s) \neq \emptyset \\ select(s, \langle E_1, \dots, E_{n-1} \rangle, \alpha), & otherwise \end{cases}$$

Shorthand

- Sat(): Boolean -- formula is true
- Soln() assignment of V_Σ (action) values to satisfy formula

Symbolic Implementation

Enforcer Operations	Symbolic Implementation
$s \in \bigcap_{i=2,\dots,n} C_{\phi_i}$	$sat(V_s = s \wedge \bigwedge_{i=2,\dots,n} C_{\phi_i})$
$\alpha \in \bigcap_{i=1,\dots,n} \mu_i(s)$	$sat(V_s = s \wedge V_\Sigma = \alpha \wedge \bigwedge_{i=1,\dots,n} \mu_i)$
$\bigcap_{i=1,\dots,n} \mu_i(s) = \emptyset$	$\neg sat(V_s = s \wedge \bigwedge_{i=1,\dots,n} \mu_i)$
$pick(\bigcap_{i=1,\dots,n} \mu_i(s))$	$soln(V_S = s \wedge \bigwedge_{i=1,\dots,n} \mu_i , V_\Sigma)$

SMT Implementation

Logical formulae over theory of linear arithmetic over rationals with operations:

z3 to solve $sat(F)$ and $soln(F, V)$

Online SMT

- Parameterized context θ (conjunctions of formulas of current context)
- Context modified through push(F), pop()
- Shorthands
  - $b \coloneqq sat^{\dagger}(\Gamma) \equiv push(\Gamma); b \coloneqq sat^{\dagger}(); pop()$
  - $\alpha \coloneqq soln^{\dagger}(\Gamma, \mathrm{V}) \equiv push(\Gamma); \alpha \coloneqq soln^{\dagger}(V); pop()$

Algorithms

```
proc select(s, α, n) {
  if (n = 1) {
    if (sat(V_S = s ∧ V_Σ = α ∧ μ_1))
      return α;
    return soln(V_S = s ∧ μ_1, V_Σ);
  }
  b := sat(V_S = s ∧ C^n);
  if (¬b) return select(s, α, n − 1);
  if (sat(V_S = s ∧ V_Σ = α ∧ μ^n))
    return α;
  α' := soln(V_S = s ∧ μ^n, V_Σ);
  if (α' ≠ ⊥) return α';
  return select(s, α, n − 1);
}
```

Experiment – Setup

Controller & Enforcer running on Laptop

- Ubuntu 16.04
- Processor
- Two XBox controllers

Two Parrot mini-drones Travis

Optitrack localization system with 8 cameras @ 120Hz

- (x,y,z) + roll,pitch,yaw

Three-dimensional enforcement (fence & separation)

Operators Flying Drones with Xbox controllers

Fixed-Priority with Rate-Monotonic Priority Assignment

Experiment Behavior

Individual Enforcers

- Virtual Fence:
  - Enforcer Prevents Drone from Leaving Fenced Area
- Separation (Two Drones)
  - Enforcer prevents drones from getting closer than the separation

Combined Enforcers (priority)

- Higher Priority: Virtual Fence:
  - Drone “pushed” by separation enforcer when other drone approaches
  - Until pushed drone reaches fence where it stops & violates separation
- Higher Priority: Separation
  - Drone “pushed” by separation enforcer when other drone approaches
  - When pushed drone reaches fence it does not stops violating virtual fence

Enforcer Performance

Thread	Per	Prio	Flt-Time	#Jobs	DL-Miss	RespTime	ExecTime
T_{FL}	5	9	2358.22	530841	0	1.099/0.151/0.039	1.099/0.150/0.039
T_{CL}	50	2	2358.22	53059	26	250.994/0.146/4.281	0.101/0.014/0.008
T_{RS}	40	7	2358.22	64996	19	238.114/0.118/2.842	0.776/0.030/0.015
T_{Log}	1000	1	2358.22	2656	0	31.849/1.198/3.598	0.895/0.330/0.114
$\langle E_1 \rangle$	20	8	145.98	7743	572	83.626/5.579/7.709	39.164/5.415/7.453
$\langle E_1 \rangle^{\dagger}$	20	8	147.99	8397	0	7.196/0.323/0.439	3.553/0.322/0.434
$\langle E_1 \rangle^{*}$	20	8	197.11	8295	2564	33.910/9.798/10.237	32.722/9.558/9.984
$\langle E_1 \rangle^{*\dagger}$	20	8	353.03	19684	0	7.539/1.015/1.435	7.310/1.012/1.427
$\langle E_2 \rangle$	20	8	219.07	11338	660	45.079/5.752/7.515	42.942/5.611/7.329
$\langle E_2 \rangle^{\dagger}$	20	8	146.55	8368	0	2.732/0.361/0.480	2.732/0.361/0.480
$\langle E_2 \rangle^{*}$	20	8	188.14	8099	2327	36.035/9.940/10.264	34.776/9.705/10.018
$\langle E_2 \rangle^{*\dagger}$	20	8	234.75	13258	0	11.623/0.999/1.856	11.242/0.986/1.817
$\langle E_1, E_2 \rangle$	20	8	100.77	3479	2118	46.066/15.415/11.633	44.547/15.088/11.384
$\langle E_1, E_2 \rangle^{\dagger}$	20	8	101.23	5605	0	3.834/0.637/0.787	3.834/0.637/0.788
$\langle E_1, E_2 \rangle^{*}$	20	8	130.74	4396	2657	48.932/16.053/12.269	47.564/15.731/12.017
$\langle E_1, E_2 \rangle^{*\dagger}$	20	8	89.79	5009	0	13.640/1.815/2.579	13.157/1.796/2.537
$\langle E_2, E_1 \rangle$	20	8	55.61	2447	920	57.623/10.631/11.434	56.112/10.416/11.192
$\langle E_2, E_1 \rangle^{\dagger}$	20	8	81.71	4629	0	3.898/0.561/0.762	3.899/0.561/0.762
$\langle E_2, E_1 \rangle^{*}$	20	8	69.50	2795	1152	45.360/13.066/13.464	44.214/12.801/13.176
$\langle E_2, E_1 \rangle^{*\dagger}$	20	8	96.15	5315	0	16.940/2.656/3.770	16.371/2.586/3.647

Conclusions

Algorithms to combined CPS enforcers with conflicting actuations

- Design-time prioritization

Symbolic Enforcer Encoding in SMT

Online Incremental SMT implementation

Experiments with Drones

- Scheduled with RMS