Verifying Cyber-Physical Systems by Combining Software Model Checking with Hybrid Systems Reachability

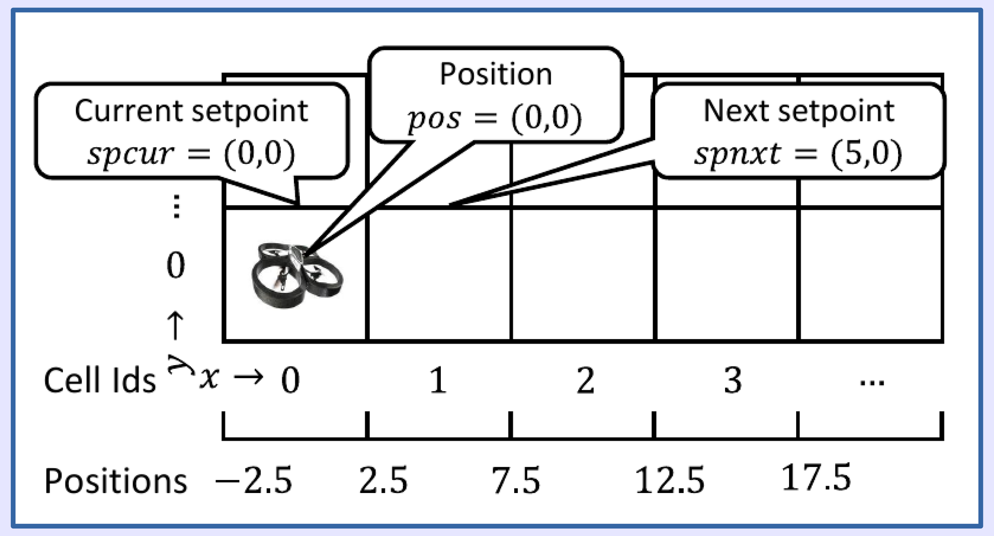
Stanley Bak[1], Sagar Chaki[2]

[1]Air Force Research Laboratory (AFRL), USA, [2]Software Engineering Institute (SEI)

Motivating Example

Cyber-Physical Systems combine discrete computation with physical environmental interactions. Some popular models for these components are:

- Software Code (Discrete Logic)
- Hybrid Automaton (Controller + Plant)

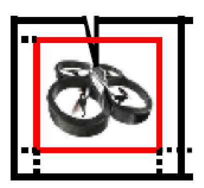


Consider a distributed quadcopter system where each agent moves in a 2d world. Each helicopter has continuous dynamics. We want to prove distributed, end-to-end collision avoidance.

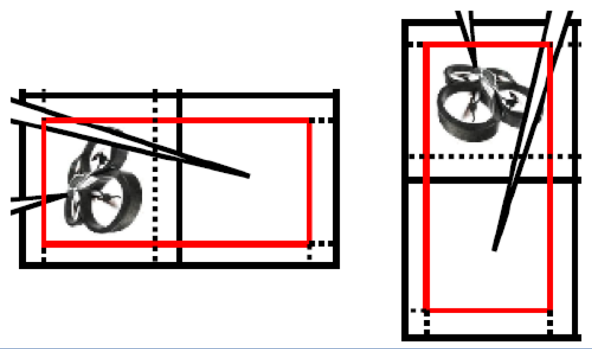
Single-Agent Continuous Property

$$(\Phi_{hover} \wedge spnxt = spcur) \vee (\Phi_{move} \wedge (|spnxt - spcur| = (5,0) \vee |spnxt - spcur| = (0,5)))$$

$$\Phi_{hover} \equiv |pos - spcur| \leq (1.5, 1.5)$$



$$\begin{aligned}\Phi_{move} \equiv\ & min(spcur_x, spnxt_x) - 1.5 \leq pos_x \\ & \leq max(spcur_x, spnxt_x) + 1.5 \\ \wedge\ & min(spcur_y, spnxt_y) - 1.5 \leq pos_y \\ & \leq max(spcur_y, spnxt_y) + 1.5\end{aligned}$$



Each agent maintains a current set-point (spcur) and a next set-point (spnxt). In the HOVER mode, the agent must stay near spcur. In MOVE mode, the agent must stay near the set-points' connecting line.

Contract Automaton

The contract automaton encodes the single-agent continuous property, as well as the restrictions on the changes between modes and updates to spcur and spnxt.

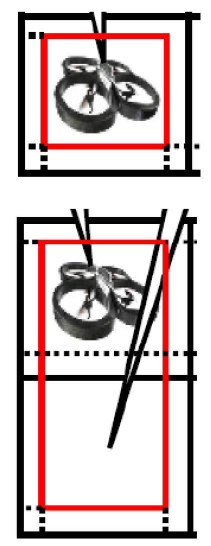
(C1) The application always calls $update\_setpoint(x, y)$, with arguments that satisfy the condition $|(x, y) - spcur| = (5, 0) \vee |(x, y) - spcur| = (0, 5)$.

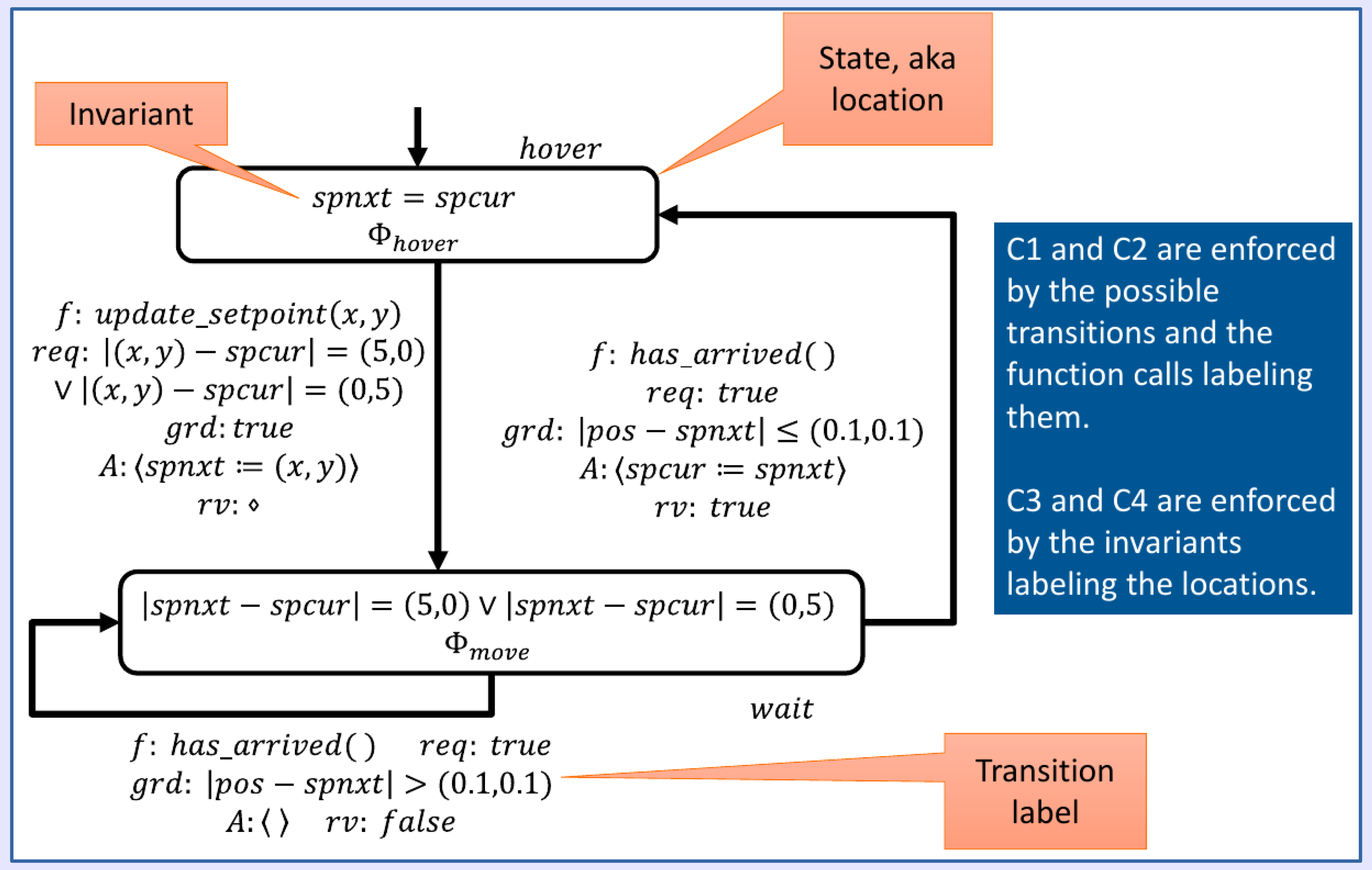
(C2) Once the application calls $update\_setpoint(x, y)$, it can keep calling $has\_arrived()$ until it gets a return value of TRUE; once $has\_arrived()$ returns TRUE, the application can only then start to call $update\_setpoint(x, y)$ again.

(C3) When the quadcopter is hovering (i.e., $spnxt = spcur$), the controller must maintain the following invariant: $\Phi_{hover} \equiv |pos - spcur| \leq (1.5, 1.5)$.

(C4) When the quadcopter is moving (i.e., $|spnxt - spcur| = (5, 0) \vee |spnxt - spcur| = (0, 5)$), the controller must maintain the following invariant:

$$\begin{aligned}\Phi_{move} \equiv\ & min(spcur_x, spnxt_x) - 1.5 \leq pos_x \\ & \leq max(spcur_x, spnxt_x) + 1.5 \\ \wedge\ & min(spcur_y, spnxt_y) - 1.5 \leq pos_y \\ & \leq max(spcur_y, spnxt_y) + 1.5\end{aligned}$$





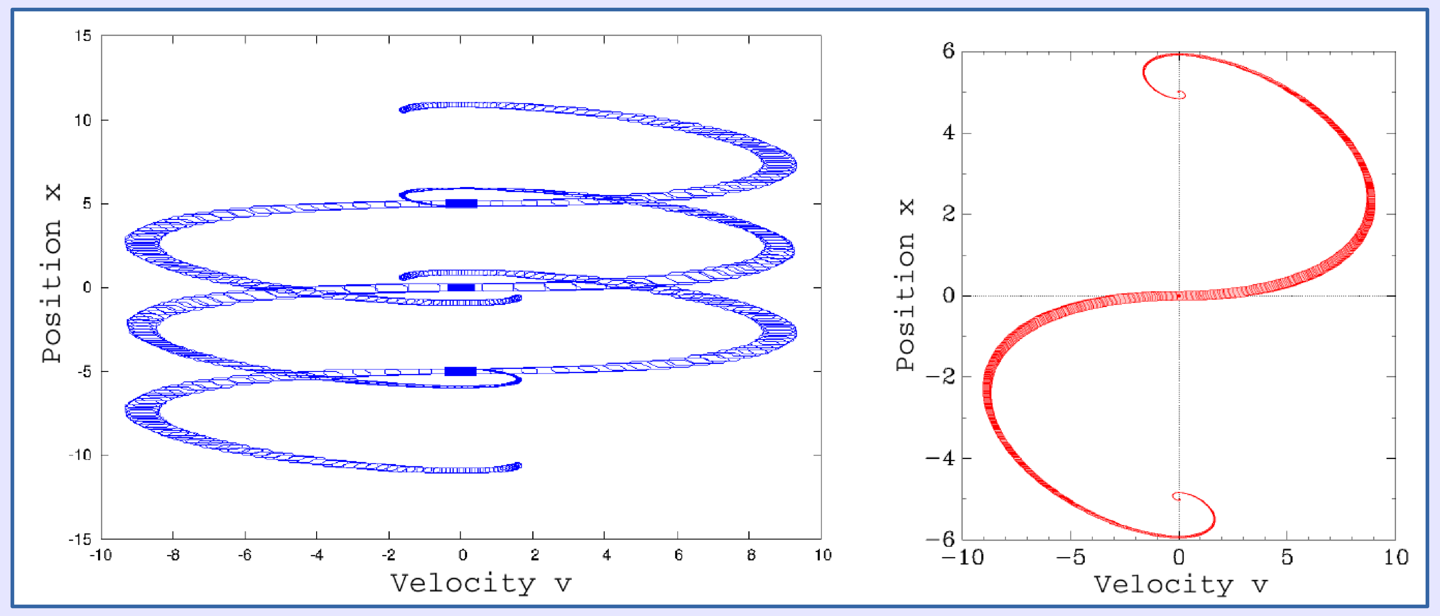
Proving Correctness

The contract automaton is used to create <u>software specifications</u> which we prove with a software model checker (CBMC).

```
enum Loc {hover, wait};
Loc loc = hover;

void update_setpoint(double x, double y) {
  pos = *; //-- assign non-deterministic value
  if (loc == hover) {
    assume(INV_hover); assert(REQ_hover_wait);
    spnxt = (x,y); assert(INV_wait);
    loc = wait; return;
  }
  assert(0);
}
```

The contract automaton is used to create <u>continuous specifications</u> which we prove using a hybrid systems reachability tool.



Showing distributed collision avoidance requires additional distributed properties proven by assuming a synchronous middleware and using CBMC (sequentialization).

The single-agent and distributed properties can then be composed into an SMT problem to check if collisions are possible, which is discharged using Z3.

```
(<= (abs (- (pos i) (pos j))) (* 2.0 HELI_RADIUS))
```

Potential Error	Detection
Software bug modifies setpoint twice in a row	SW
Software bug changes setpoint by both x and y	SW
Controller's gains are too high causing quadcopter to overshoot into neighboring cell	HY
Controller logic unstable	HY
Real-time period of low-level controller too low	HY
`has_arrived` condition too aggressive	HY
Barrier synchronization incorrectly used in communication protocol	DIST
Software does not reason about loss of communication	DIST
Buffer distances in cells too small	SMT
Helicopters too large for a given grid size	SMT