DARPA HACMS
High Assurance Spiral

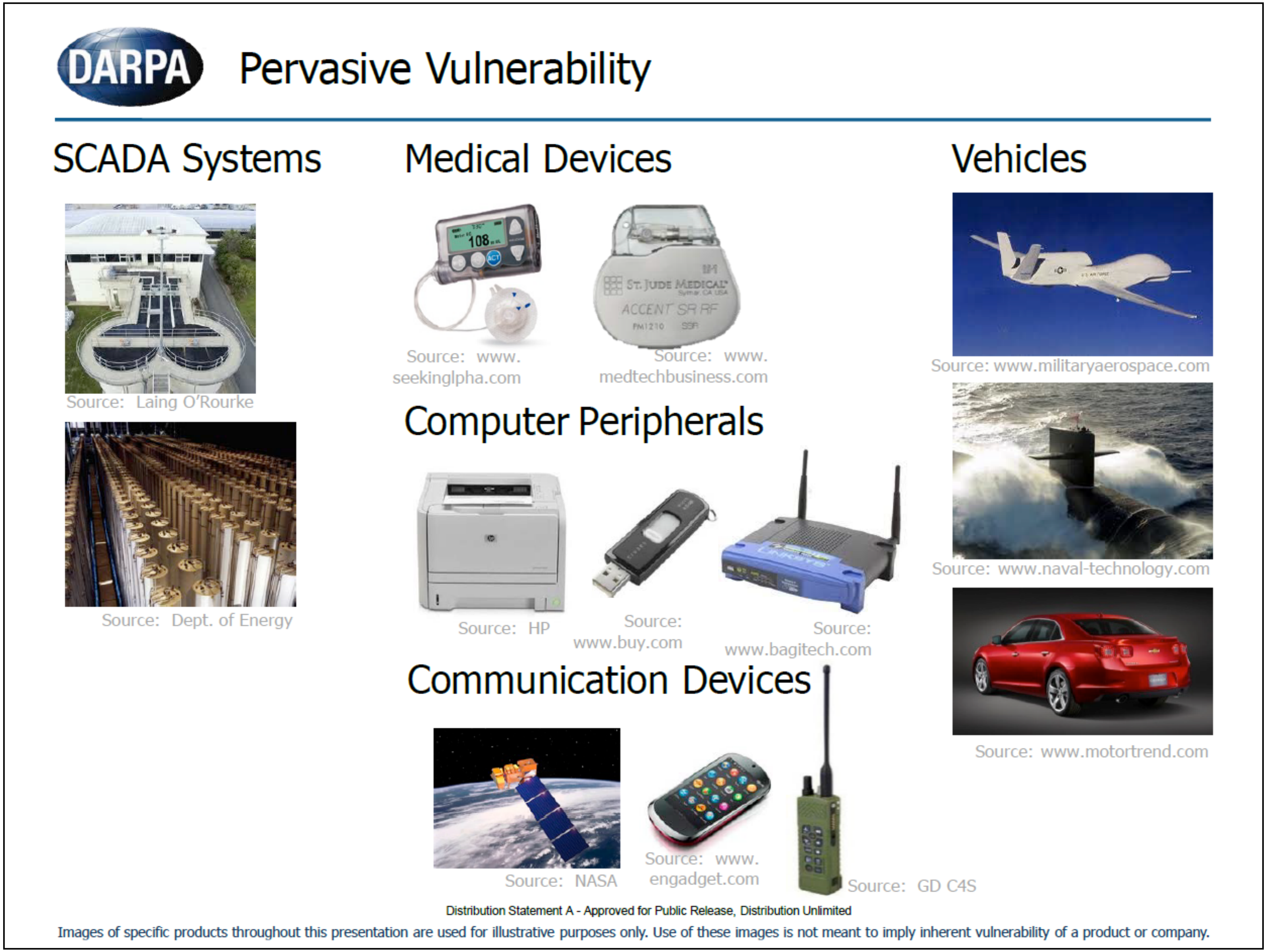
18-847E
Spiral: Formal Approaches to Hardware & Software Design & Algorithm Verification

Franz Franchetti

Carnegie Mellon University
`www.ece.cmu.edu/~franzf`

Lecture based on joint work with CMU, UIUC, Drexel, and SpirlaGen, Inc.

The DARPA HACMS Program (K. Fisher)



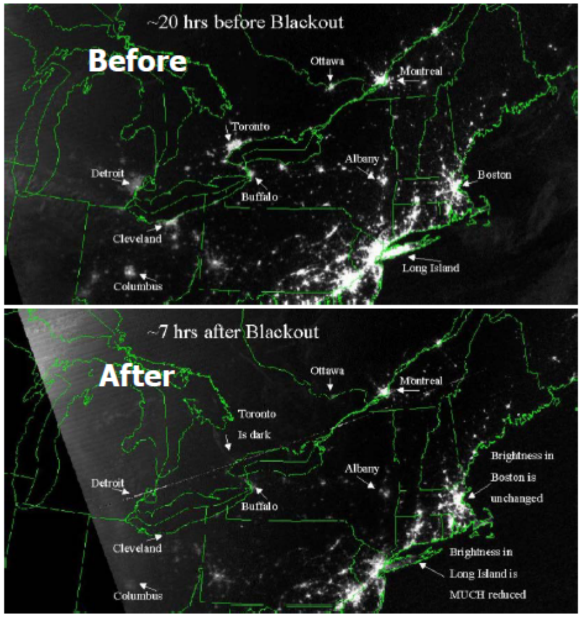
Source: DARPA-BAA-12-21 "High-Assurance Cyber Military Systems (HACMS)" Proposer's Day Slides by K. Fisher, HACMS Program Manager

The DARPA HACMS Program (K. Fisher)

DARPA Ubiquitous, Invisible, Networked, Computing Substrate

- In 2008, ~30 embedded processors per person in developed countries.
- In 2009, 98% of microprocessors were embedded [IEEE Computer `09]
- Trend: *Networked* embedded systems
- Vulnerabilities have economic and national security consequences. Extrapolating from *safety* failures:
  - June 10, 1999. Olympic Pipeline Company. 237K gallons of gasoline spilled. 3 deaths. >$45M damages. [NTSB report]
  - Aug 14 2003. Northeast Blackout cost $6B for 2 days of outage [DOE study]
  - April 26, 1986. Chernobyl Nuclear Disaster: >$300B. Belarus alone: $235B. [Chernobyl Forum]

August 14 2003
Northeast Blackout



Source: NOAA

> The growing connectivity between information systems, the Internet, and other infrastructures creates opportunities for attackers to disrupt telecommunications, electrical power, energy pipelines, refineries, financial networks, and other critical infrastructures."
> -- Dennis C. Blair, Director of National Intelligence, *Annual Threat Assessment of the Intelligence Community for the Senate Select Committee on Intelligence, Statement for the Record*, February 12, 2009

Distribution Statement A - Approved for Public Release, Distribution Unlimited

Source: DARPA-BAA-12-21 "High-Assurance Cyber Military Systems (HACMS)" Proposer's Day Slides by K. Fisher, HACMS Program Manager

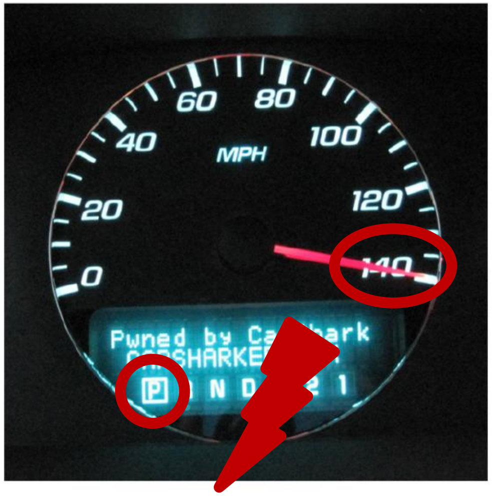
The DARPA HACMS Program (K. Fisher)

Source: DARPA-BAA-12-21 "High-Assurance Cyber Military Systems (HACMS)" Proposer's Day Slides by K. Fisher, HACMS Program Manager

Our Approach: Model-Based High Assurance

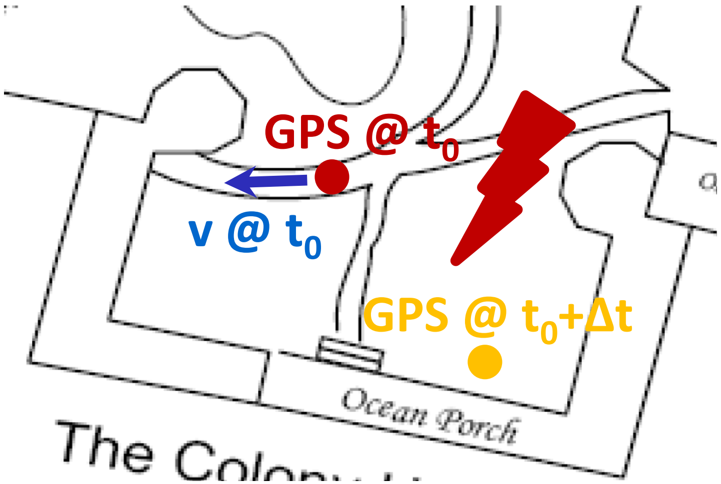
Multi-sensor UGVs

- **Multiple sensors:** GPS, compass, accelerometer, IMU, etc.
- **Control:** waypoints, joystick vector
- **Vehicle model:** laws of physics, vehicle state
- **Map data:** Terrain, possible paths, obstacles

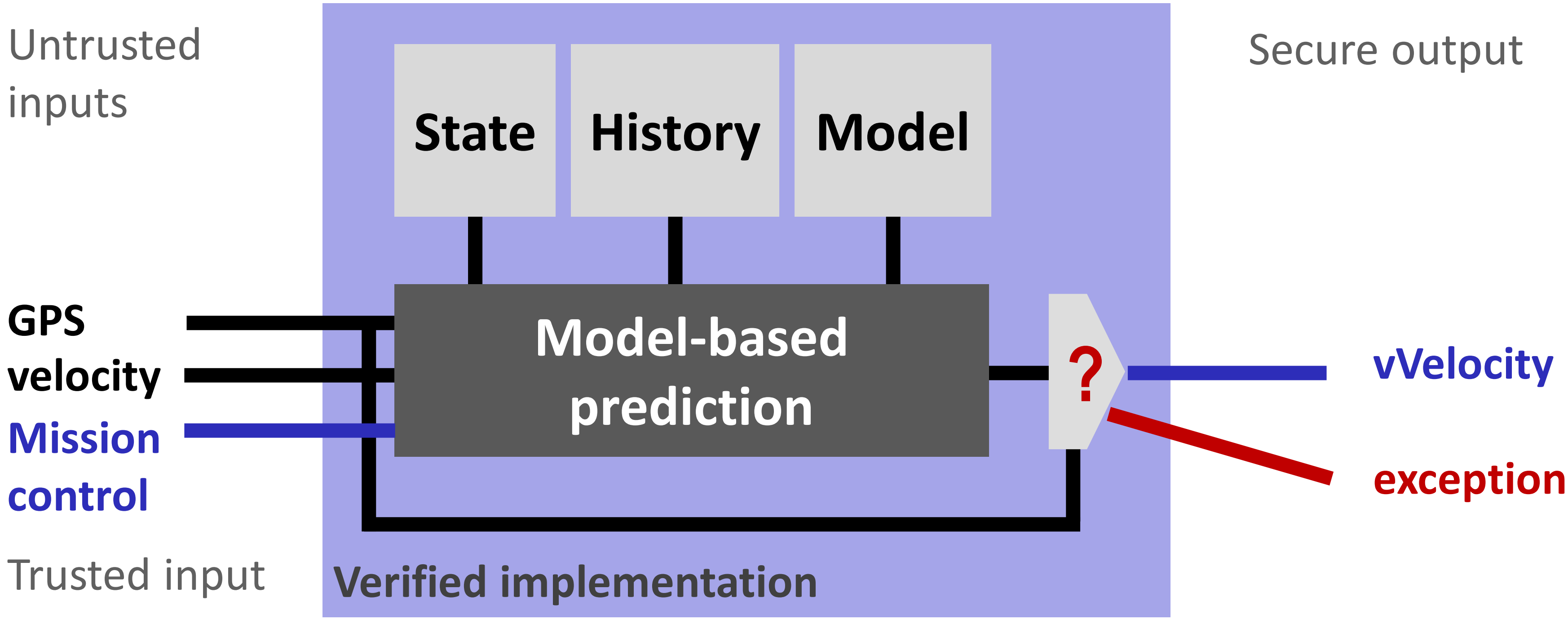


Assurance Through Consistency

- Model-based consistency checks
- Model vs. vehicle state
- Map-based path validation
- Exception signal if inconsistency threshold is exceeded



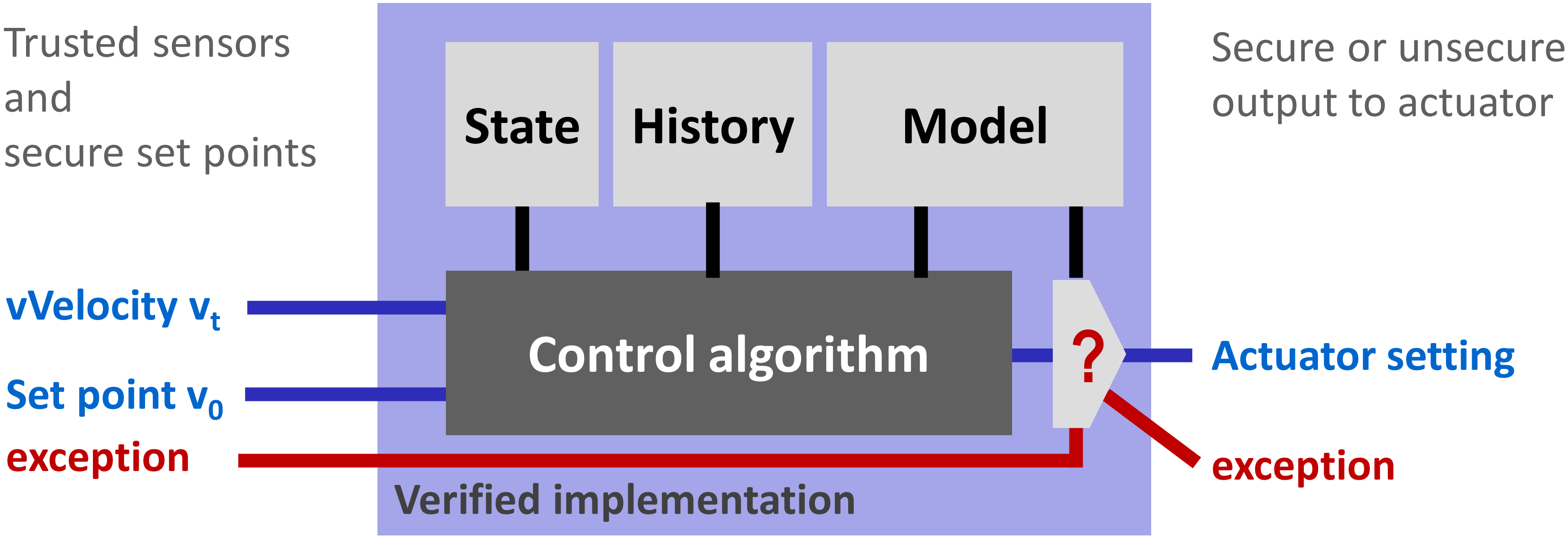
Virtual High Assurance Sensors



Assurance Through Consistency

- Model-based consistency checks Model vs. vehicle state
- Utilizes maps, physics, history, anticipated behavior, mission control
- Trusted virtual sensor output if model and sensors agree
- Exception if divergence beyond security threshold

High Assurance Controller



Assurance Through Guaranteed Controller Input and Output

- **Controller input:** virtual high-assurance sensor outputs
- **Controller output:** trusted or untrusted message to actuator
- **Controller algorithm:** PID or MPC, may use state, history and model
- **Failsafe:** use model-derived actuator setting if exception detected

Organization

- Overview
- **Approach**
- Example: Dynamic Window Monitor
- More HCOL examples
- Other research components
- Demos
- Concluding remarks

HCOL: Hybrid Control Operator Language

Sensor values and model-based predictions

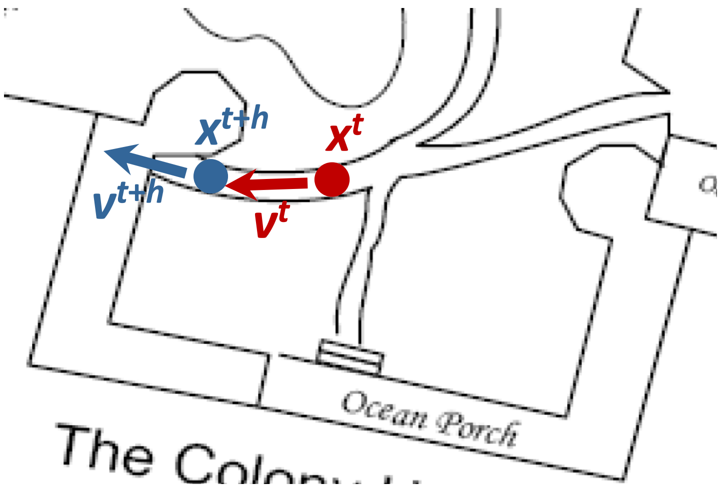
Euler step: x^{t+h}

$$\mathrm{x}^{t+h} \approx \left[\mathrm{I}_3 \,|\, h\,\mathrm{I}_3\right](\mathrm{x}^t \oplus \mathrm{v}^{t+h})$$

Numerical differentiation: v^{t+h}

$$\mathrm{v}^{t+h} \approx 1/h\left[\mathrm{I}_3 \,|\, -\mathrm{I}_3\right](\mathrm{x}^{t+h} \oplus \mathrm{x}^t)$$

I_3: 3 x 3 identity matrix
time step = matrix-vector product



Assurance through guaranteed controller input and output

- Declarative representation of physics, data and control algorithms
- Enables rule-based software synthesis and variant generation, verification and proof co-synthesis
- Extends Spiral's OL and SPL languages into the control domain

HCOL: Control Operator Examples

Time step residue: Disagreement between model and sensors

$$\mathbf{r}^{t+h} = \mathbf{R} \cdot (\mathbf{x}^t \oplus \mathbf{v}^t \oplus \mathbf{x}^{t+h} \oplus \mathbf{v}^{t+h})$$

$$\mathbf{R} = \begin{bmatrix} -\mathrm{I}_3 & h\,\mathrm{I}_3 & \mathrm{I}_3 & 0_3 \\ 1/h\,\mathrm{I}_3 & 0_3 & 1/h\,\mathrm{I}_3 & \mathrm{I}_3 \end{bmatrix}$$

Error operator: L_2 norm of time step residue

$$\mathsf{E}_h : (\mathbf{s}^t, \mathbf{s}^{t+h}) \mapsto (\mathbf{s}^t \oplus \mathbf{s}^{t+h})^\top \left(\mathbf{R}^\top \mathbf{R}\right) (\mathbf{s}^t \oplus \mathbf{s}^{t+h})$$

$$\mathbf{s}^t = (\mathbf{x}^t \oplus \mathbf{v}^t), \quad \mathbf{s}^{t+h} = (\mathbf{x}^{t+h} \oplus \mathbf{v}^{t+h})$$

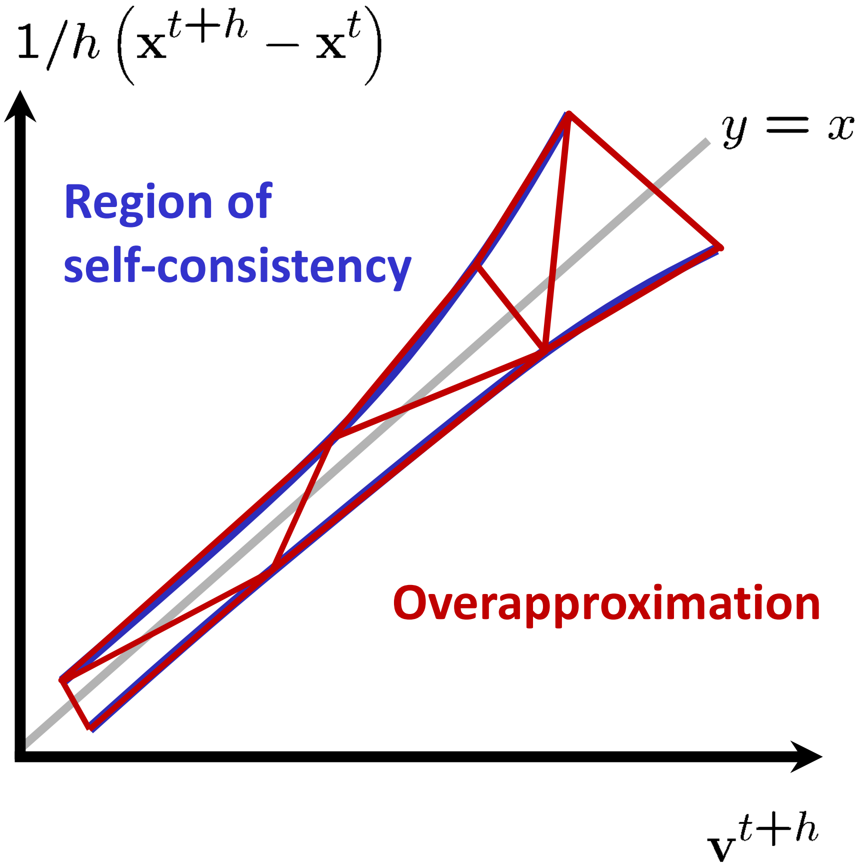
PID controller: Control velocity at set point v_0

$$\begin{pmatrix} \mathbf{u}^t \\ \mathbf{s}^t \end{pmatrix} = \left(\begin{bmatrix} k_p\,\mathrm{I}_e \,|\, k_i\,\mathrm{I}_3 \,|\, k_d/h\,\mathrm{I}_3 \end{bmatrix} \begin{bmatrix} \mathrm{I}_3 & . & . \\ \mathrm{I}_3 & . & \mathrm{I}_3 \\ \mathrm{I}_3 & -\mathrm{I}_3 & . \end{bmatrix} \begin{bmatrix} \mathrm{I}_3 & -\mathrm{I}_3 & . \\ . & . & \mathrm{I}_6 \end{bmatrix} \right) \oplus \begin{bmatrix} \mathrm{I}_3 & -\mathrm{I}_3 & . \\ \mathrm{I}_3 & -\mathrm{I}_3 & \mathrm{I}_3 \end{bmatrix} (\mathbf{v}_0 \oplus \mathbf{v}^t \oplus \mathbf{s}^{t-h})$$

$$\mathbf{e}^t = \mathbf{v}^0 - \mathbf{v}^t, \quad \mathbf{s}^t = \mathbf{e}^t \oplus \sum_{i=0}^{n-1} \mathbf{e}^{ih}$$

Usual PID controller definition: $\mathbf{u}^t = k_p \mathbf{e}^t + k_i \sum_{i=0}^{n-1} \mathbf{e}^{ih} + k_d \frac{\mathbf{e}^t - \mathbf{e}^{t-h}}{h}$

Detection Through Feasible Region of State



Self-consistency equation

$$\mathcal{F}:\begin{pmatrix}\mathbf{x}^t\\ \mathbf{v}^t\\ \mathbf{x}^{t+h}\\ \mathbf{v}^{t+h}\end{pmatrix}\mapsto\begin{pmatrix}1/h\left(\mathbf{x}^{t+h}-\mathbf{x}^{t}\right)\\ \mathbf{v}^{t+h}\end{pmatrix}$$

Inside a polyhedra

$$A_i\mathcal{F}(\vec{x})-b_i\preceq\vec{0}$$

Test: attack-free, if $\mathcal{F}\left(\mathbf{s}^t\oplus\mathbf{s}^{t+h}\right)\in\bigcup_i\mathcal{P}_i$

Rule-Based Code Synthesis

High Level Rules: Transformations within high level abstraction

$$\mathrm{I}_n \rightarrow \sum_{i=0}^{n-1} \mathrm{e}_i^n \, \mathrm{I}_1 (\mathrm{e}_i^n)^\top$$

$$\left[\sum_{i=0}^{n-1} \mathsf{S}_i \, A_i \, \mathsf{G}_i \,\Big|\, \sum_{i=0}^{n-1} \mathsf{S}_i \, B_i \, \mathsf{G}_i \right] \rightarrow \sum_{i=0}^{n-1} \mathsf{S}_i \left(\Big[A_i | B_i \Big] \begin{bmatrix} 1 \\ 1 \end{bmatrix} \right) \mathsf{G}_i$$

with $\mathrm{e}_i^{1 \times n} = [0, \ldots, 0, 1, 0, \ldots, 0]$

Code generation rules: Translate high level abstraction into code

$$\mathsf{Code}\left(y = (A\,B)x\right) \rightarrow \left\{\mathtt{Decl}(t), \mathsf{Code}\left(t = Bx\right), \mathsf{Code}\left(y = At\right)\right\}$$

$$\mathsf{Code}\left(y = \left(\sum_{i=0}^{n-1} A_i\right) x\right) \rightarrow \left\{y := \vec{0}, \mathtt{for}(i = 0..n-1)\ \mathsf{Code}\left(y+ = A_i x\right)\right\}$$

$$\mathsf{Code}\left(y = \mathrm{e}_i^{1 \times n}\, x\right) \rightarrow y[0] := x[i]$$

$$\mathsf{Code}\left(y = \mathrm{e}_i^{n \times 1}\, x\right) \rightarrow \left\{y = \vec{0}, y[i] := x[0]\right\}$$

Co-Synthesis of Code and Correctness Proofs

Code generation: rule application until convergence

$$y^{t+h} = \left[I_3 \,|h\, I_3 \right] (x^t \oplus v^{t+h})$$

```
RuleSet := rec(
   SumSAG_In := Rule(@(I(@1)),(@, @1)->Let(i := Idx(@1),
      ISum(i, @1, e(@1, i) * I(1) * e(@1, i)^T))),
   SumDist := ...,
...);
```

```
let(y:=var(TArray(TReal, 3)), xv:=var(TArray(TReal, 6)), h := TReal(1/100),
   func([inparam(xv), outparam(y)],
     loop(i, [0..3], chain(
       assign(nth(y, i), add(nth(xv, i), mul(h, nth(xv, add(i,3)))))))))
```

Proof generation: trail of rule application

```
rule: "SumSAG_In"
matched: BlockMat([[-I(3), 1/100*I(3), I(3), O(3)],
                   [100*I(3), O(3), 100*I(3), @(I(3))]])
wildcards: @="I(3)", @1="3"
rewritten: "ISum(k, 3, e(3, k) * I(1) * e(3, k)^T))"
proof: "I(3) == ISum(k, 3, e(3, k) * I(1) * e(3, k)^T))"
result: BlockMat([[-I(3), 1/100*I(3), I(3), O(3)],
                 [100*I(3), O(3), 100*I(3), ISum(k, 3, e(3, k)*I(1)*e(3, k)^T)]])
```

Symbolic Rule Verification

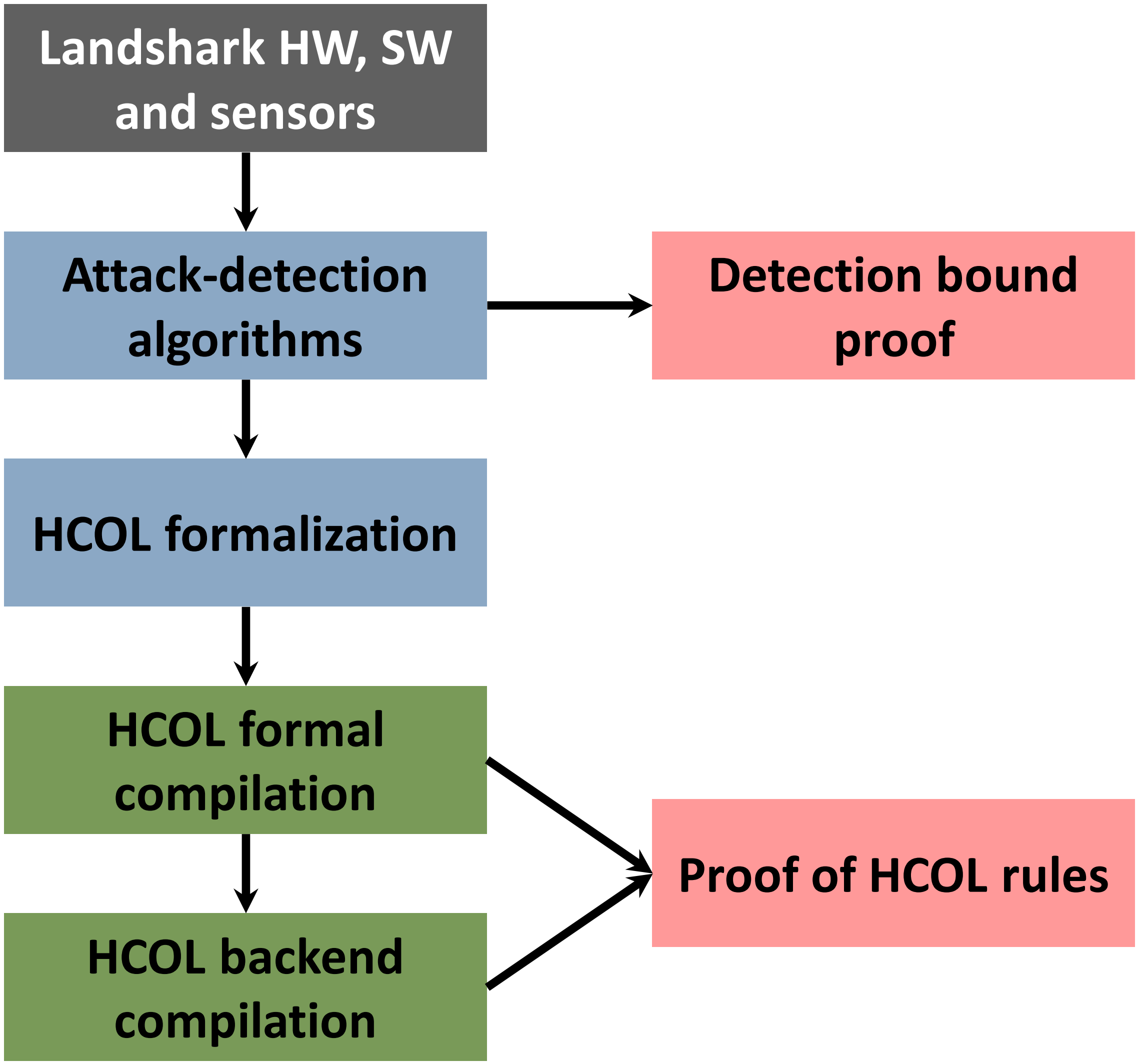
- **Rule replaces left-hand side by right-hand side when preconditions match**

$$\mathrm{I}_n \rightarrow \sum_{i=0}^{n-1} \mathrm{e}_i^n \, \mathrm{I}_1 (\mathrm{e}_i^n)^\top$$

- **Test rule by symbolically evaluating expressions before and after rule application and compare result**

$$\mathrm{I}_3 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \mathbf{= ?} \quad \sum_{i=0}^{2} \mathrm{e}_i^3 \, \mathrm{I}_1 (\mathrm{e}_i^3)^\top = [1\,0\,0][1][1\,0\,0]^\top + [0\,1\,0][1][0\,1\,0]^\top + [0\,0\,1][1][0\,0\,1]^\top$$

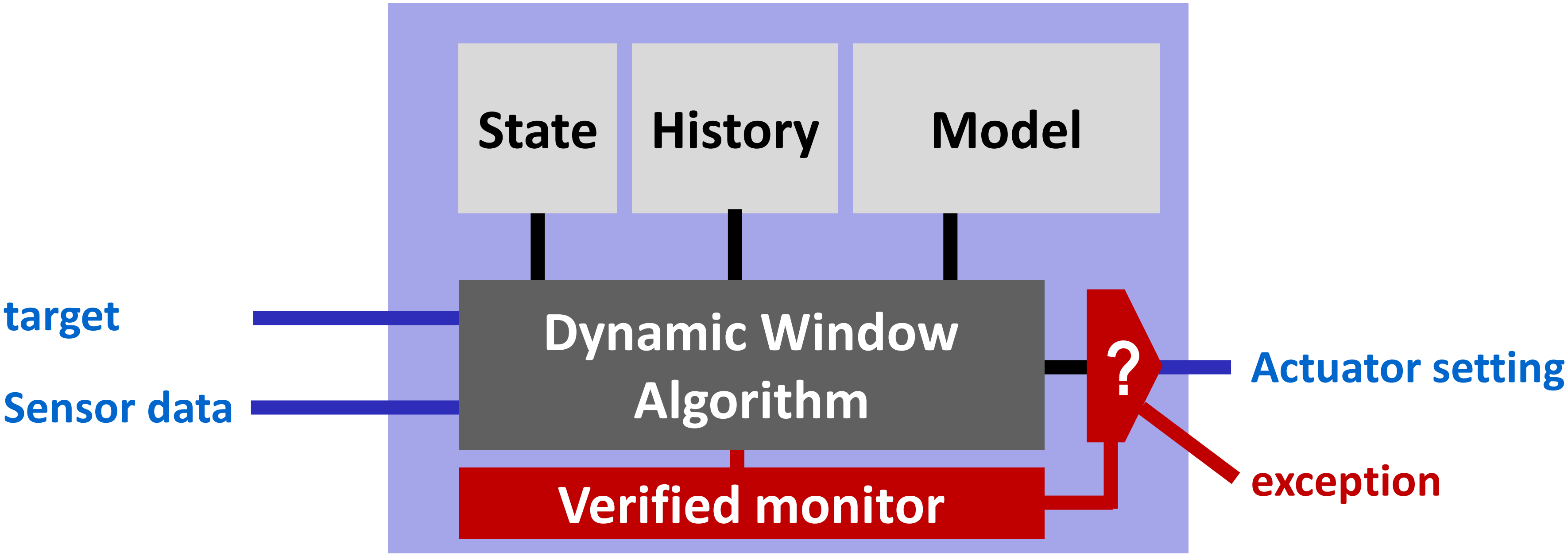
Putting It All Together



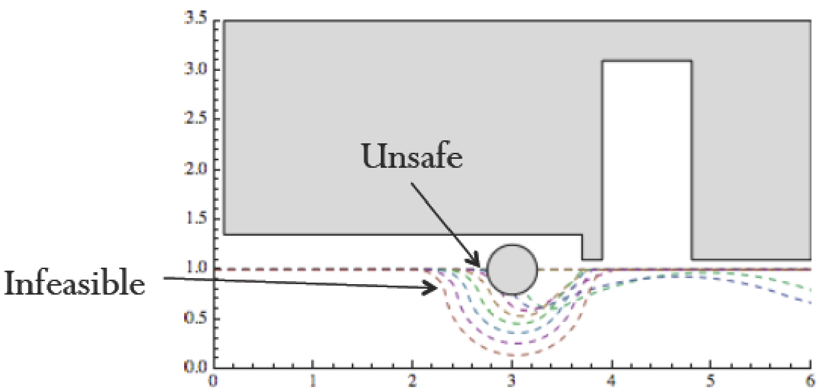
Organization

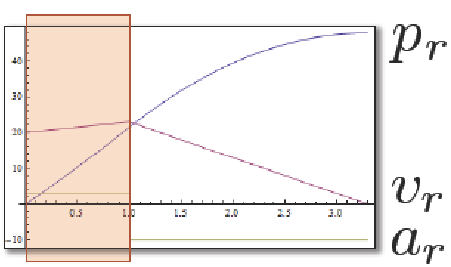
- Overview
- Approach
- **Example: Dynamic Window Monitor**
- More HCOL examples
- Other research components
- Demos
- Concluding remarks

Dynamic Window Safety Monitor



Dynamic Window Approach Primer





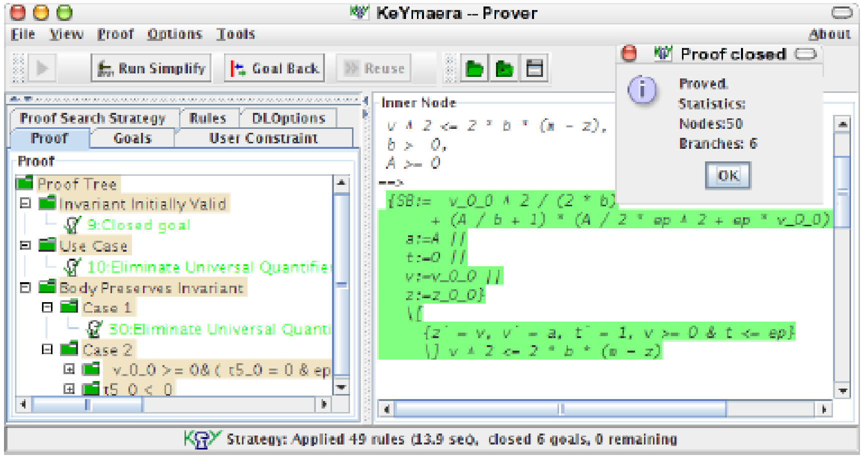
$$\|p_r - p_o\|_\infty > \frac{v_r^2}{2b} + V\frac{v_r}{b} + \left(\frac{A}{b}+1\right)\left(\frac{A}{2}\varepsilon^2 + \varepsilon(v_r+V)\right)$$

Algorithm Verified in KeYmaera

Theorem and proof

To Prove: $\psi_{ps} \rightarrow [dw_{ps}] \left((v_r = 0) \lor \left(\|p_r - p_o\| > \frac{v_r^2}{2b} + V\frac{v_r}{b} \right) \right)$

$$
\begin{aligned}
dw_{ps} &\equiv (ctrl_o \parallel ctrl_r;\ dyn)^* \\
ctrl_o &\equiv v_o = (*,*);\ ?\|v_o\| \le V \\
ctrl_r &\equiv (a_r := -b) \\
&\quad \cup (?v_r = 0;\ a_r := 0;\ \omega_r := 0) \\
&\quad \cup (a_r := *;\ ?{-b} \le a_r \le A;\ \omega_r := *;\ ?-\Omega \le \omega_r \le \Omega; \\
&\qquad p_c := (*,*);\ d_r := (*,*);\ p_o := (*,*);\ ?feasible \land safe) \\
feasible &\equiv \|p_r - p_c\| > 0 \land \omega_r \|p_r - p_c\| = v_r \land d_r = \frac{(p_r - p_c)^{\perp}}{\|p_r - p_c\|} \\
safe &\equiv \|p_r - p_o\|_\infty > \frac{v_r^2}{2b} + V\frac{v_r}{b} + \left(\frac{A}{b} + 1\right)\left(\frac{A}{2}\varepsilon^2 + \varepsilon(v_r + V)\right) \\
dyn &\equiv (t := 0;\ p_r^{x\prime} = v_r d_r^x,\ p_r^{y\prime} = v_r d_r^y,\ d_r^{x\prime} = -\omega_r d_r^y,\ d_r^{y\prime} = \omega_r d_r^x, \\
&\qquad p_o^{x\prime} = v_o^x,\ p_o^{y\prime} = v_o^y,\ v_r' = a_r,\ \omega_r' = \frac{a_r}{\|p_r - p_c\|},\ t' = 1 \\
&\qquad \&\ v_r \ge 0 \land t \le \varepsilon)
\end{aligned}
$$



	$\|p_r - p_o\|_\infty > \frac{v_r^2}{2b} + \left(\frac{A}{b}+1\right)\left(\frac{A}{2}\varepsilon^2 + \varepsilon v_r\right)$
	$v_r = 0 \lor \|p_r - p_o\|_\infty > \frac{v_r^2}{2b} + V\frac{v_r}{b} + \left(\frac{A}{b}+1\right)\left(\frac{A}{2}\varepsilon^2 + \varepsilon(v_r + V)\right)$
Sensor uncertainty	$\|\tilde{p}_r - p_o\|_\infty > \frac{v_r^2}{2b} + V\frac{v_r}{b} + \left(\frac{A}{b}+1\right)\left(\frac{A}{2}\varepsilon^2 + \varepsilon(v_r + V)\right) + U_p$
Actuator disturbance	$\|p_r - p_o\|_\infty > \frac{v_r^2}{2bU_m} + V\frac{v_r}{bU_m} + \left(\frac{A}{bU_m}+1\right)\left(\frac{A}{2}\varepsilon^2 + \varepsilon(v_r + V)\right)$
Sensor failure	$\|\tilde{p}_r - p_o\|_\infty > \frac{v_r^2}{2b} + V\frac{v_r}{b} + \left(\frac{A}{b}+1\right)\left(\frac{A}{2}\varepsilon^2 + \varepsilon(v + V)\right) + U_p + g\Delta$
	$v_r = 0 \lor \|p_r - p_o\|_\infty > \frac{v_r^2}{2b} + \frac{V^2}{2b_o} + V\left(\frac{v_r}{b} + \tau\right) + \left(\frac{A}{b}+1\right)\left(\frac{A}{2}\varepsilon^2 + \varepsilon(v_r + V)\right)$

Resulting safety monitor condition

$$\|p_r - p_o\|_\infty > \frac{v_r^2}{2b} + V\frac{v_r}{b} + \left(\frac{A}{b} + 1\right)\left(\frac{A}{2}\varepsilon^2 + \varepsilon(v_r + V)\right)$$

Proof/Code Co-Synthesis: HA Spiral

$$\mathbb{R} \times \mathbb{R}^2 \times \mathbb{R}^2 \to \mathbb{Z}_2$$
$$(v_r, p_r, p_o) \mapsto \big(p(v_r) < d_\infty(p_r, p_o)\big)$$

HCOL specification

↓ Expansion + backtracking

HCOL (dataflow) expression

↓ Recursive descent

Σ-HCOL (loop) expression

↓ Confluent term rewriting

Optimized Σ-HCOL expression

↓ Recursive descent

Abstract code

High Assurance Spiral (Beta)

```
Sigma-SPL expression:
---------------------------------------------------------------------------
Induction(3, Lambda([ r1, r2 ], mul(r1, r2)), V(1.0))


===========================================================================
EXPANSION RULE: OLCompose
---------------------------------------------------------------------------
SPL expression:
---------------------------------------------------------------------------
ScalarProd(3, D) o
Induction(3, Lambda([ r1, r2 ], mul(r1, r2)), V(1.0))
---------------------------------------------------------------------------
Sigma-SPL expression:
---------------------------------------------------------------------------
Reduction(3, (a, b) -> add(a, b), V(0.0), (arg) -> false) o
PointWise(3, Lambda([ r8, i2 ], mul(r8, nth(D, i2)))) o
Induction(3, Lambda([ r1, r2 ], mul(r1, r2)), V(1.0))


===========================================================================
EXPANSION RULE: Reduction
---------------------------------------------------------------------------
SPL expression:
---------------------------------------------------------------------------
Reduction(2, (a, b) -> max(a, b), V(0.0), (arg) -> false)
---------------------------------------------------------------------------
Sigma-SPL expression:
---------------------------------------------------------------------------
Reduction(2, (a, b) -> max(a, b), V(0.0), (arg) -> false)
```

```
func(TInt, "transform", [ X, D ],
   decl([ u1, u2, u3, u4, u5, u6, u7, u8, w1, x1, x10, x1
      chain(
         ivenv(
            assign(u5, V([ V(0.0), V(0.0) ])),
            assign(u2, vcvt_64f32f(addsub_4x32f(vdup(Real
            assign(u1, V([ V(-1.0), V(1.0) ])),
            loop(i5, [ 0 .. 2 ],
               chain(
                  assign(x6, addsub_2x64f(vdup(add(RealEP
                  assign(x1, addsub_2x64f(V([ V(0.0), V(0
                  assign(x2, mul(x1, x6)),
                  assign(x3, mul(vushuffle_2x64f(x1, vpar
                  assign(x4, neg(min(x3, x2))),
                  assign(u3, add(max(vushuffle_2x64f(x4,
                  assign(u5, add(u5, u3)),
                  assign(x7, addsub_2x64f(V([ V(0.0), V(0
```

Details: Formal Compilation

- **HCOL Breakdown Rules**

$$\text{SafeDist}_{V,A,b,\varepsilon}(.,.,.) \rightarrow \Big(P[x,(a_0,a_1,a_2)](.) < d_\infty^2(.,.)\Big)(.,.,.)$$

$$\text{with} \quad a_0=\frac{1}{2b},\ a_1=\frac{V}{b}+\varepsilon\Big(\frac{A}{b}+1\Big),\ a_2=\Big(\frac{A}{b}+1\Big)\Big(\frac{A}{2}\varepsilon^2+\varepsilon V\Big)$$

$$d_\infty^n(.,.) \rightarrow \|.\|_\infty^n \circ (-)_n$$

$$(\diamond)_n \rightarrow \text{Pointwise}_{n\times n,(a,b)\mapsto a\diamond b}$$

$$\|.\|_\infty^n \rightarrow \text{Reduction}_{n,(a,b)\mapsto \max(|a|,|b|)}$$

$$<.,.>_n \rightarrow \text{Reduction}_{n,(a,b)\mapsto a+b} \circ \text{Pointwise}_{n\times n,(a,b)\mapsto ab}$$

$$P[x,(a_0,\ldots,a_n)] \rightarrow <(a_0,\ldots,a_n),.> \circ (x^i)_n$$

$$(x^i)_n \rightarrow \text{Induction}_{n,(a,b)\mapsto ab,1}$$

- **Fully Expanded HCOL Expression**

$$\text{SafeDist}_{V,A,b,\varepsilon} \rightarrow \text{Atomic}_{(x,y)\mapsto x<y} \circ \Big(\big(\text{Reduction}_{3,(x,y)\mapsto x+y} \circ \text{Pointwise}_{3,x\mapsto a_i x} \circ \text{Induction}_{3,(a,b)\mapsto ab,1}\big) \times \big(\text{Reduction}_{2,(x,y)\mapsto \max(|x|,|y|)} \circ \text{Pointwise}_{2\times 2,(x,y)\mapsto x-y}\big)\Big)$$

Final Synthesized C Code

```
int dwmonitor(float  *X, double  *D) {
    __m128d u1, u2, u3, u4, u5, u6, u7, u8 , x1, x10, x13, x14, x17, x18, x19, x2, x3, x4, x6, x7, x8, x9;
    int w1;
    {
        unsigned _xm = _mm_getcsr();
        _mm_setcsr(_xm & 0xffff0000 | 0x0000dfc0);
        u5 = _mm_set1_pd(0.0);
        u2 = _mm_cvtps_pd(_mm_addsub_ps(_mm_set1_ps(FLT_MIN), _mm_set1_ps(X[0])));
        u1 = _mm_set_pd(1.0, (-1.0));
        for(int i5 = 0; i5 <= 2; i5++) {
            x6 = _mm_addsub_pd(_mm_set1_pd((DBL_MIN + DBL_MIN)), _mm_loaddup_pd(&(D[i5])));
            x1 = _mm_addsub_pd(_mm_set1_pd(0.0), u1);
            x2 = _mm_mul_pd(x1, x6);
            x3 = _mm_mul_pd(_mm_shuffle_pd(x1, x1, _MM_SHUFFLE2(0, 1)), x6);
            x4 = _mm_sub_pd(_mm_set1_pd(0.0), _mm_min_pd(x3, x2));
            u3 = _mm_add_pd(_mm_max_pd(_mm_shuffle_pd(x4, x4, _MM_SHUFFLE2(0, 1)), _mm_max_pd(x3, x2)), _mm_set1_pd(DBL_MIN));
            u5 = _mm_add_pd(u5, u3);
            x7 = _mm_addsub_pd(_mm_set1_pd(0.0), u1);
            x8 = _mm_mul_pd(x7, u2);
            x9 = _mm_mul_pd(_mm_shuffle_pd(x7, x7, _MM_SHUFFLE2(0, 1)), u2);
            x10 = _mm_sub_pd(_mm_set1_pd(0.0), _mm_min_pd(x9, x8));
            u1 = _mm_add_pd(_mm_max_pd(_mm_shuffle_pd(x10, x10, _MM_SHUFFLE2(0, 1)), _mm_max_pd(x9, x8)), _mm_set1_pd(DBL_MIN));
        }
        u6 = _mm_set1_pd(0.0);
        for(int i3 = 0; i3 <= 1; i3++) {
            u8 = _mm_cvtps_pd(_mm_addsub_ps(_mm_set1_ps(FLT_MIN), _mm_set1_ps(X[(i3 + 1)])));
            u7 = _mm_cvtps_pd(_mm_addsub_ps(_mm_set1_ps(FLT_MIN), _mm_set1_ps(X[(3 + i3)])));
            x14 = _mm_add_pd(u8, _mm_shuffle_pd(u7, u7, _MM_SHUFFLE2(0, 1)));
            x13 = _mm_shuffle_pd(x14, x14, _MM_SHUFFLE2(0, 1));
            u4 = _mm_shuffle_pd(_mm_min_pd(x14, x13), _mm_max_pd(x14, x13), _MM_SHUFFLE2(1, 0));
            u6 = _mm_shuffle_pd(_mm_min_pd(u6, u4), _mm_max_pd(u6, u4), _MM_SHUFFLE2(1, 0));
        }
        x17 = _mm_addsub_pd(_mm_set1_pd(0.0), u6);
        x18 = _mm_addsub_pd(_mm_set1_pd(0.0), u5);
        x19 = _mm_cmpge_pd(x17, _mm_shuffle_pd(x18, x18, _MM_SHUFFLE2(0, 1)));
        w1 = (_mm_testc_si128(_mm_castpd_si128(x19), _mm_set_epi32(0xffffffff, 0xffffffff, 0xffffffff, 0xffffffff)) –
             (_mm_testnzc_si128(_mm_castpd_si128(x19), _mm_set_epi32(0xffffffff, 0xffffffff, 0xffffffff, 0xffffffff))));
        __asm nop;
        if (_mm_getcsr() & 0x0d) {
            _mm_setcsr(_xm);
            return -1;
        }
        _mm_setcsr(_xm);
    }
    return w1;
}
```

Assembly Generated By Intel C Compiler

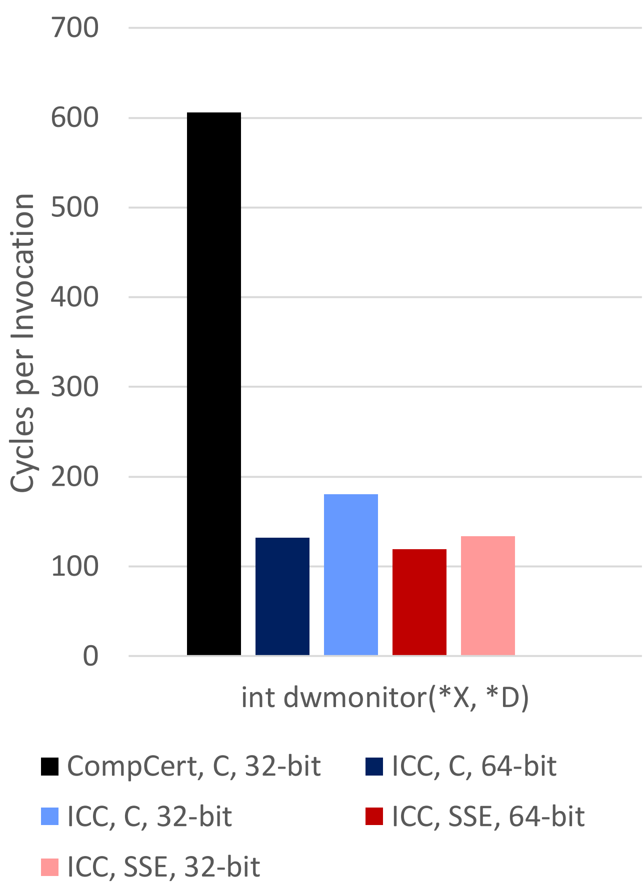
```
dwmonitor  PROC
        sub        rsp, 120
        vstmxcsr   DWORD PTR [112+rsp]
        mov        r8d, DWORD PTR [112+rsp]
        mov        eax, r8d
        and        eax, -65536
        or         eax, 57280
        mov        DWORD PTR [112+rsp], eax
        vldmxcsr   DWORD PTR [112+rsp]
        vmovaps    xmm3, XMMWORD PTR [_2il0floatpacket.2]
        vmovss     xmm0, DWORD PTR [rcx]
        vshufps    xmm1, xmm0, xmm0, 0
        vmovaps    xmm0, XMMWORD PTR [_2il0floatpacket.3]
        vxorps     xmm5, xmm5, xmm5
        vmovaps    xmm2, xmm5
        vaddsubps  xmm4, xmm3, xmm1
        vmovaps    xmm1, XMMWORD PTR [_2il0floatpacket.4]
        vcvtps2pd  xmm4, xmm4
        xor        eax, eax
        vmovaps    XMMWORD PTR [32+rsp], xmm11
        vmovaps    xmm11, XMMWORD PTR [_2il0floatpacket.5]
        ...
        vmovddup   xmm15, QWORD PTR [rdx+rax*8]
        inc        rax
        vaddsubpd  xmm13, xmm1, xmm15
        vaddsubpd  xmm15, xmm5, xmm0
        vminpd     xmm13, xmm14, xmm12
        ...
        <100 more lines>
        ...
        add        rsp, 120
        ret
        ALIGN      16
dwmonitor ENDP
```

64-bit mode
AVX/VEX encoding
3 operand instructions
SSE 4.1
1-1 mapping to C source
150 lines of assembly

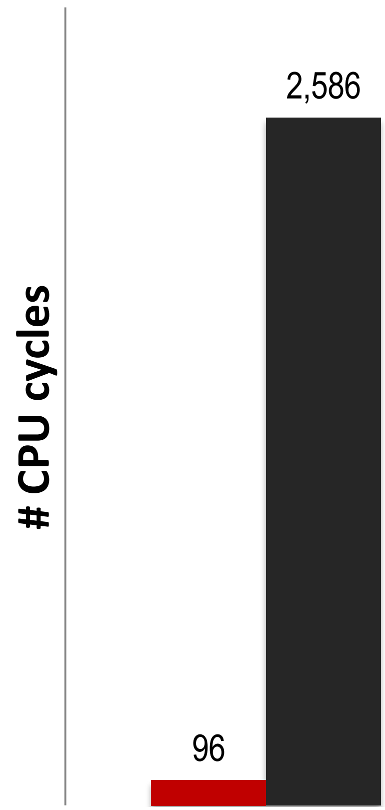
On SandyBridge:
100 – 240 cycles
30ns – 80ns @ 3 GHz

Spiral Interval Arithmetic Code Quality

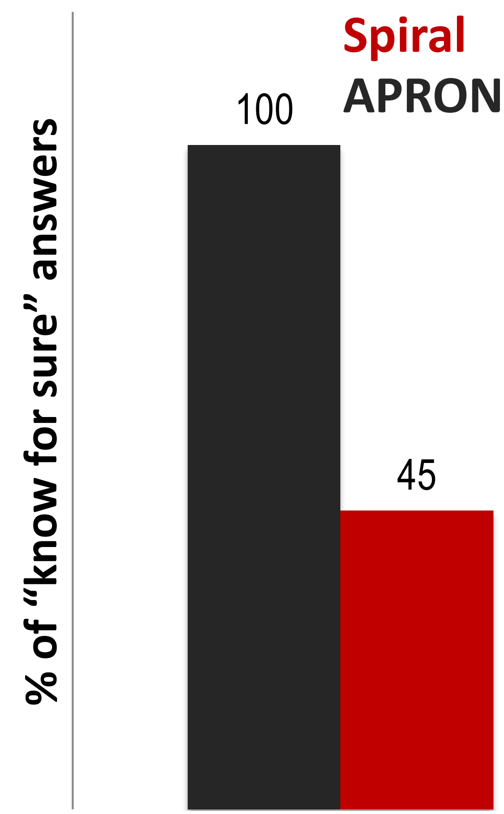
Intel C vs. CompCert



Performance



Precision at boundary



a < b ? for
$a - b \approx 10^{-15}$

SandyBridge CPU, Intel C Compiler, CompCert, APRON Interval Arithmetic Library

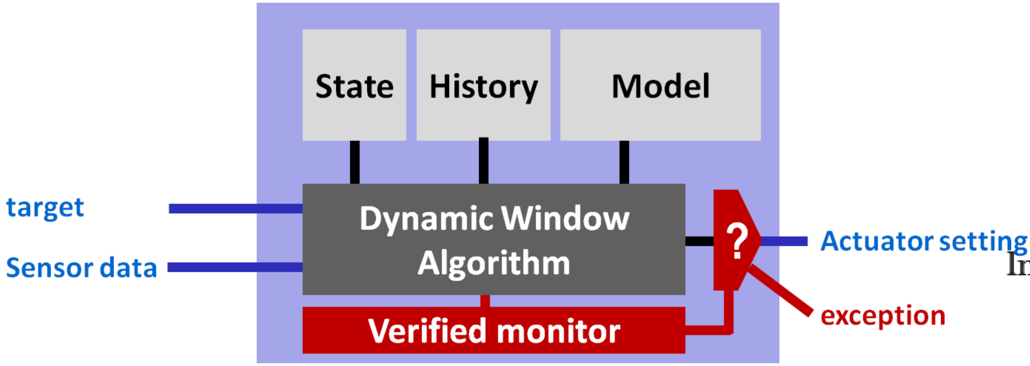
Organization

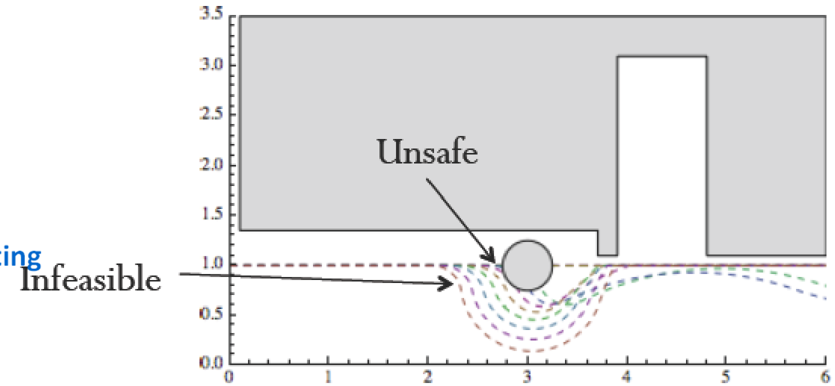
- Overview
- Approach
- Example: Dynamic Window Monitor
- **More HCOL examples**
- Other research components
- Demos
- Concluding remarks

Algorithms Formalized in HA Spiral

- **Dynamic Window Approach Monitor**
  Passive safety monitor, formally derived in KeYmaera
- **Set calculus: Sensor self-consistency in state space**
  Check that set of self-consistent true state values permitted by measurements is non-empty
- **Multi-timescale Z-test for redundant sensors**
  Test for zero mean of difference between multiple sensors on multiple time scales
- **Mathematical infrastructure ROS code**
  Coordinate transformations, data filtering, ODE integration

Dynamic Window Safety Monitor



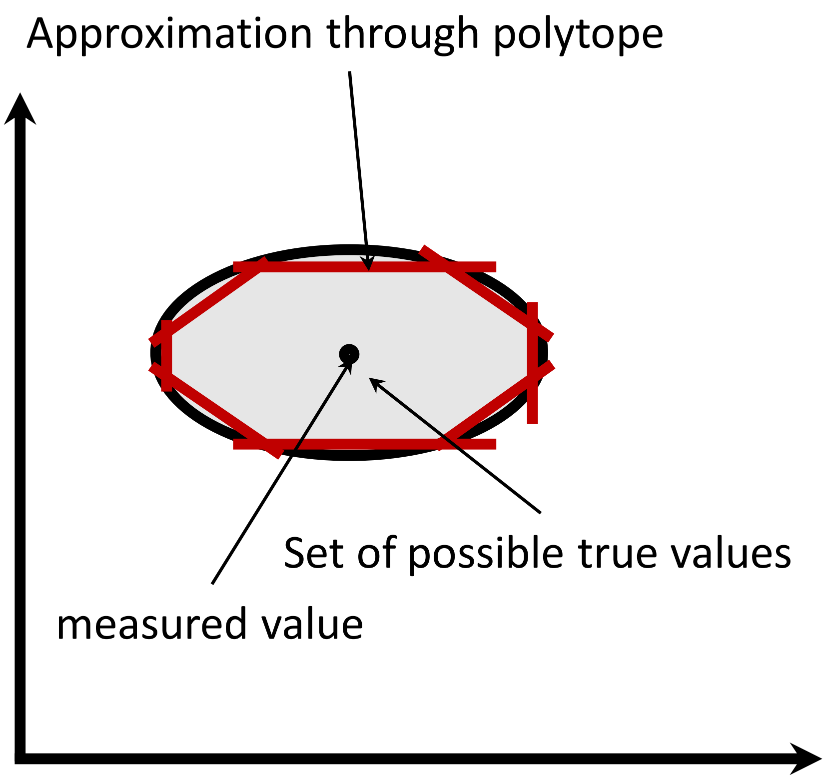


KeYmaera verification: monitors

Safety	Invariant + Safe Control (RSS'13)
static	$\lVert p_r - p_o \rVert_\infty > \frac{v_r^2}{2b} + \left(\frac{A}{b}+1\right)\left(\frac{A}{2}\varepsilon^2 + \varepsilon v_r\right)$
passive	$v_r = 0 \lor \lVert p_r - p_o \rVert_\infty > \frac{v_r^2}{2b} + V\frac{v_r}{b} + \left(\frac{A}{b}+1\right)\left(\frac{A}{2}\varepsilon^2 + \varepsilon(v_r + V)\right)$
+ sensor	$\lVert \hat{p}_r - p_o \rVert_\infty > \frac{v_r^2}{2b} + V\frac{v_r}{b} + \left(\frac{A}{b}+1\right)\left(\frac{A}{2}\varepsilon^2 + \varepsilon(v_r + V)\right) + U_p$
+ disturb	$\lVert p_r - p_o \rVert_\infty > \frac{v_r^2}{2bU_m} + V\frac{v_r}{bU_m} + \left(\frac{A}{bU_m}+1\right)\left(\frac{A}{2}\varepsilon^2 + \varepsilon(v_r + V)\right)$
+ failure	$\lVert \hat{p}_r - p_o \rVert_\infty > \frac{v_r^2}{2b} + V\frac{v_r}{b} + \left(\frac{A}{b}+1\right)\left(\frac{A}{2}\varepsilon^2 + \varepsilon(v + V)\right) + U_p + g\Delta$
friendly	$\lVert p_r - p_o \rVert_\infty > \frac{v_r^2}{2b} + \frac{V^2}{2b_o} + V\left(\frac{v_r}{b} + \tau\right) + \left(\frac{A}{b}+1\right)\left(\frac{A}{2}\varepsilon^2 + \varepsilon(v_r + V)\right)$

Sensor Self-Consistency in State Space

Set calculus and approximation

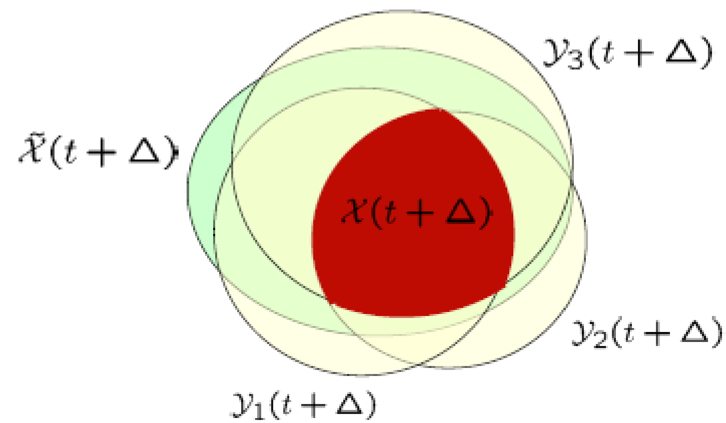


Inside a polytope ⇒ inside feasible set

$$A\vec{x} - b \preceq \vec{0}$$

Time step and physics modeling

Intersect feasible sets of all sensors



$$\mathcal{X}(t+\Delta) = \tilde{\mathcal{X}}(t+\Delta) \cap \bigcap_{k=1}^{K} \mathcal{Y}_k(t+\Delta)$$

Last intersection evolves with physics

$$\mathbf{p}(t+\Delta) = \mathbf{p}(t) + \Delta\mathbf{v}(t) + \int_t^{t+\Delta}\int_t^{s} \mathbf{a}(\tau)d\tau ds$$

$$\mathbf{v}(t+\Delta) = \mathbf{v}(t) + \int_t^{t+\Delta} \mathbf{a}(s)ds$$

HCOL Specification and Expansion

- **HCOL Specification**

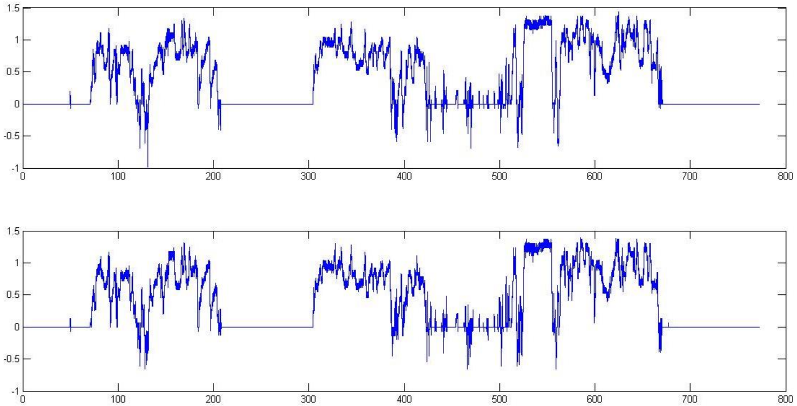
$$\mathsf{ForAny}_{i=0}^{k-1}\left(\mathsf{InsidePoly}_{m,n}(\mathbf{A}_i,\mathbf{b}_i,.)\right) : \mathbb{R}^n \to \mathbb{Z}_2$$
$$x \mapsto (\exists i : \mathbf{A}_i x - \mathbf{b}_i \preceq \vec{0})$$

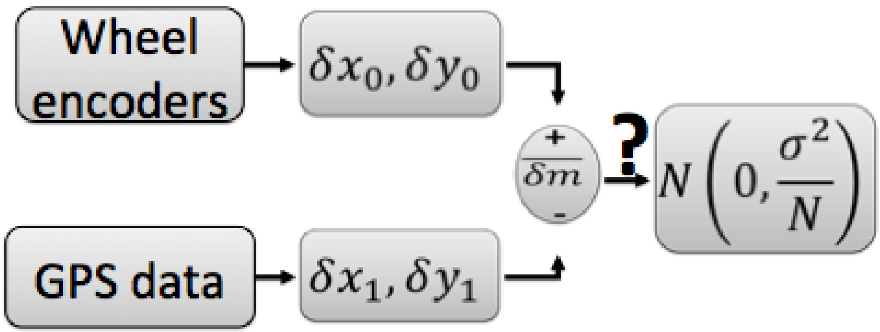
- **Expansion into HCOL expression**

$$\mathsf{ForAny}_{i=0}^{k-1}\left(\mathsf{InsidePoly}_{m,n}(\mathbf{A}_i,\mathbf{b}_i,.)\right) \to$$
$$\mathsf{Reduction}_{n,(a,b)\mapsto a\vee b}$$
$$\circ \begin{bmatrix}\cdot\\\cdot\end{bmatrix}_{i=0}^{k-1}\Big(\mathsf{Reduction}_{n,(a,b)\mapsto a\wedge b}$$
$$\circ\, \mathsf{Pointwise}_{n,x_i\mapsto x_i\le 0} \circ \mathsf{Pointwise}_{n,x_i\mapsto x_i-b_i}$$
$$\circ \begin{bmatrix}\cdot\\\cdot\end{bmatrix}_{i=0}^{n-1}\left(\mathsf{Reduction}_{n,(a,b)\mapsto a+b} \circ \mathsf{Pointwise}_{n,x_i\mapsto a_i x_i}\right)\Big)$$

Multi-Timescale Z-Test

Receive a new residual value x.
for Window size $w \in \{2^0, 2^1, \ldots, 2^{15}, \infty\}$ **do**
  Update number of samples $N_w \leftarrow N_w + 1$
  Update the residual sample average $\bar{x}_w$ to include x.
  if $N_w > w$ **then**
    Update $\bar{x}_w$ to exclude the oldest sample in w.
    $N_w \leftarrow w$
  end if
  {Compute a z-statistic for $\bar{x}_w$:}
  if $\bar{x}_w > \delta\mu_+$ **then**
    $z = \frac{\bar{x}_w - \mu_+}{\sigma / N_w}$
  else if $\bar{x}_w < \delta\mu_-$ **then**
    $z = \frac{\bar{x}_w - \mu_-}{\sigma / N_w}$
  else
    $z = 0$
  end if
  Extract p value using a Z-test on z.
  if $p < p_{thresh}$ **then**
    return Failure
  end if
end for
return Not Failure





HCOL Expansion

HCOL Operator Definition

$$Z^n_{w_{\max},\sigma,p_{\text{thresh}},\delta,\mu_+,\mu_-} : \mathbb{R}^n \to \mathbb{Z}_2$$

$$(x_0,\ldots,x_{n-1}) \mapsto \bigvee_{w\in\{2^0,2^1,\ldots,w_{\max},n\}} \left(|z_w| > \Phi^{-1}\left(1-\frac{p_{\text{thresh}}}{2}\right)\right)$$

with $N_w = \min(n,w)$

$$\overline{x}_w = \sum_{i=\max(0,n-w)}^{n-1} x_i$$

$$z_w = \begin{cases} \sigma^{-1}N_w(\overline{x}_w - \mu_+) & \text{if } \overline{x}_w > \delta\mu_+ \\ \sigma^{-1}N_w(\overline{x}_w - \mu_-) & \text{if } \overline{x}_w < \delta\mu_- \\ 0 & \text{else} \end{cases}$$

HCOL Breakdown Rule

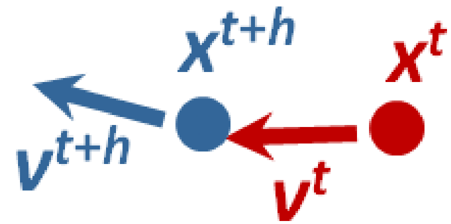
$$Z^n_{w_{\max},\sigma,p_{\text{thresh}},\delta,\mu_+,\mu_-} \to \text{Reduction}_{2+\log_2 w_{\max},(a,b)\mapsto a\vee b}$$
$$\circ\ \text{Pointwise}_{2+\log_2 w_{\max},\tau^{p_{\text{thresh}}}\circ z_w^{n,\sigma,\delta,\mu_+,\mu_-}}$$
$$\circ \begin{bmatrix}\cdot\\ \cdot\end{bmatrix}_{w\in\{2^0,2^1,\ldots,w_{\max},n\}} \overline{x}^n_w$$

Mathematial ROS Infrastructure Code

Example: (x,y) position from odometer

Euler step: x^{t+h}

$$\mathbf{x}^{t+h} \approx \left[\mathrm{I}_2 \,|\, h\, \mathrm{I}_2 \right] (\mathbf{x}^t \oplus \mathbf{v}^{t+h})$$



Usual Euler definition: $(x^{t+h}, y^{t+h}) = (x^t + hv_x, y^t + hv_y)$

PID controller: Control velocity at set point v_0

$$\begin{pmatrix} \mathbf{u}^t \\ \mathbf{s}^t \end{pmatrix} = \left(\left[k_p \mathrm{I}_e \,|\, k_i \mathrm{I}_3 \,|\, k_d/h\, \mathrm{I}_3 \right] \begin{bmatrix} \mathrm{I}_3 & \cdot & \cdot \\ \mathrm{I}_3 & \cdot & \mathrm{I}_3 \\ \mathrm{I}_3 & -\mathrm{I}_3 & \cdot \end{bmatrix} \begin{bmatrix} \mathrm{I}_3 & -\mathrm{I}_3 & \cdot \\ \cdot & \cdot & \mathrm{I}_6 \end{bmatrix} \right) \oplus \begin{bmatrix} \mathrm{I}_3 & -\mathrm{I}_3 & \cdot \\ \mathrm{I}_3 & -\mathrm{I}_3 & \mathrm{I}_3 \end{bmatrix} (\mathbf{v}_0 \oplus \mathbf{v}^t \oplus \mathbf{s}^{t-h})$$

$$\mathbf{e}^t = \mathbf{v}^0 - \mathbf{v}^t, \quad \mathbf{s}^t = \mathbf{e}^t \oplus \sum_{i=0}^{n-1} \mathbf{e}^{ih}$$

Usual PID controller definition: $\mathbf{u}^t = k_p \mathbf{e}^t + k_i \sum_{i=0}^{n-1} \mathbf{e}^{ih} + k_d \frac{\mathbf{e}^t - \mathbf{e}^{t-h}}{h}$

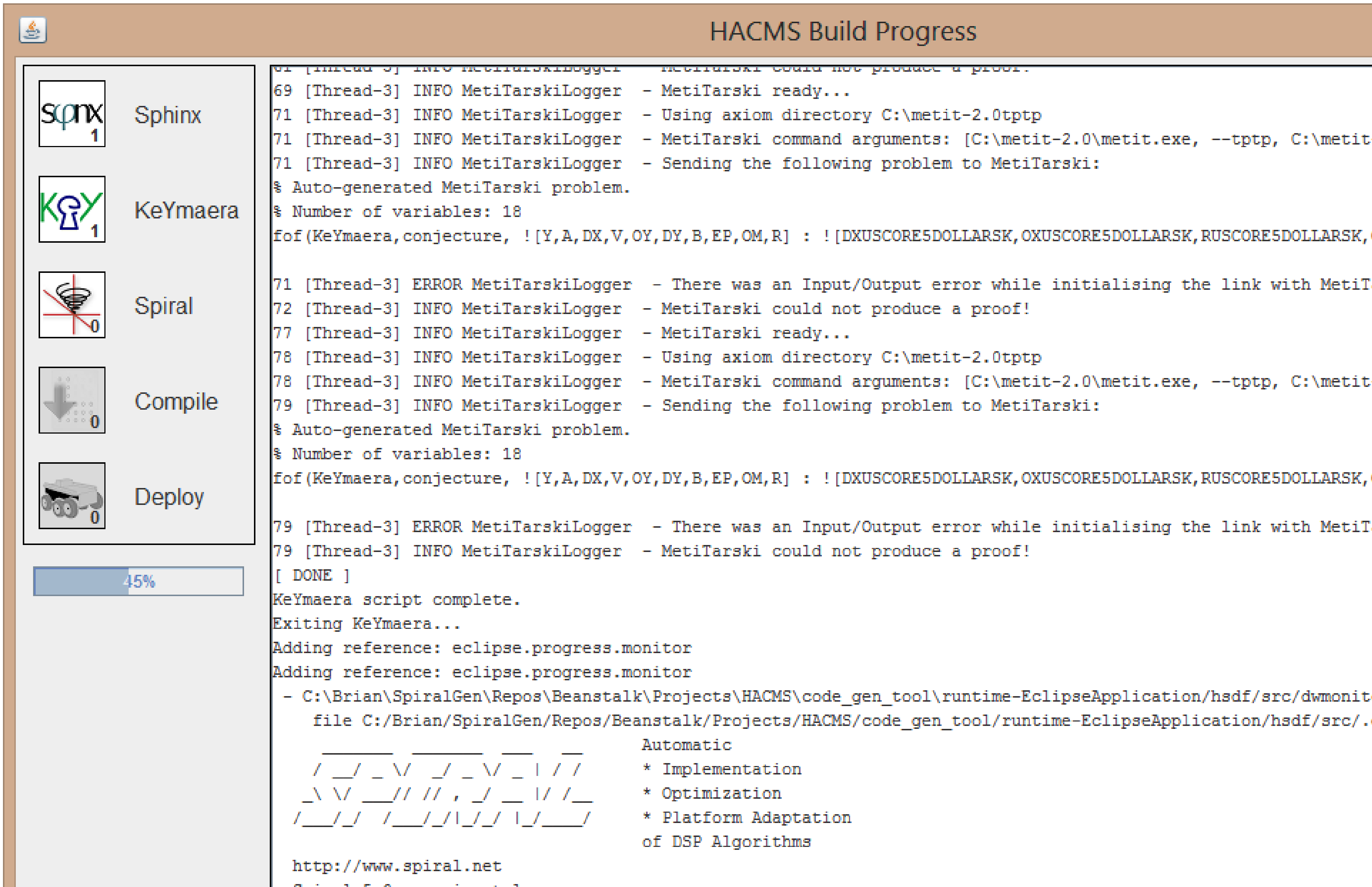
High Assurance Spiral Code Generation

High Assurance Spiral (Beta)

```
spiral> s := OLCompose(BinOp(3, Lambda([a, b], add(a, b))), PointWise(6, Lambda(
[x,i], cond(leq(i,V(2)), x, mul(x,h)))));
BinOp(3, Lambda([ a1, b1 ], add(a1, b1))) o
PointWise(6, Lambda([ r1, i1 ], cond(leq(i1, V(2)), r1, mul(r1, param(TReal, "h"
)))))
spiral> opts := HACMSopts.getOpts(rec(params := [h]));;
spiral> c := HACMSProof_Codegen(s, opts);;
spiral> c2 := Rewrite(Copy(c), RulesCodeHACMS, opts);
func(TVoid, "transform", [ Y, X, param(TReal, "h") ],
   decl([ T2 ],
      chain(
         loop(i4, [ 0 .. 5 ],
            assign(nth(T2, i4), cond(leq(i4, V(2)), nth(X, i4), mul(nth(X, i4),
param(TReal, "h"))))
         ),
         loop(i5, [ 0 .. 2 ],
            assign(nth(Y, i5), add(nth(T2, i5), nth(T2, add(i5, V(3)))))
         )
      )
   )
)
spiral> c3 := Rewrite(Copy(c2), RulesCodeUnrollHACMS, opts);;
spiral> PrintCode("euler", c3, opts);

void euler(int  *Y, double  *X, double h) {
    double q10, q11, q12, q7, q8, q9;
    q7 = X[0];
    q8 = X[1];
    q9 = X[2];
    q10 = (X[3]*h);
    q11 = (X[4]*h);
    q12 = (X[5]*h);
    Y[0] = (q7 + q10);
    Y[1] = (q8 + q11);
    Y[2] = (q9 + q12);
}
spiral> _
```

SpiralGen's High Assurance Spiral Tool Chain

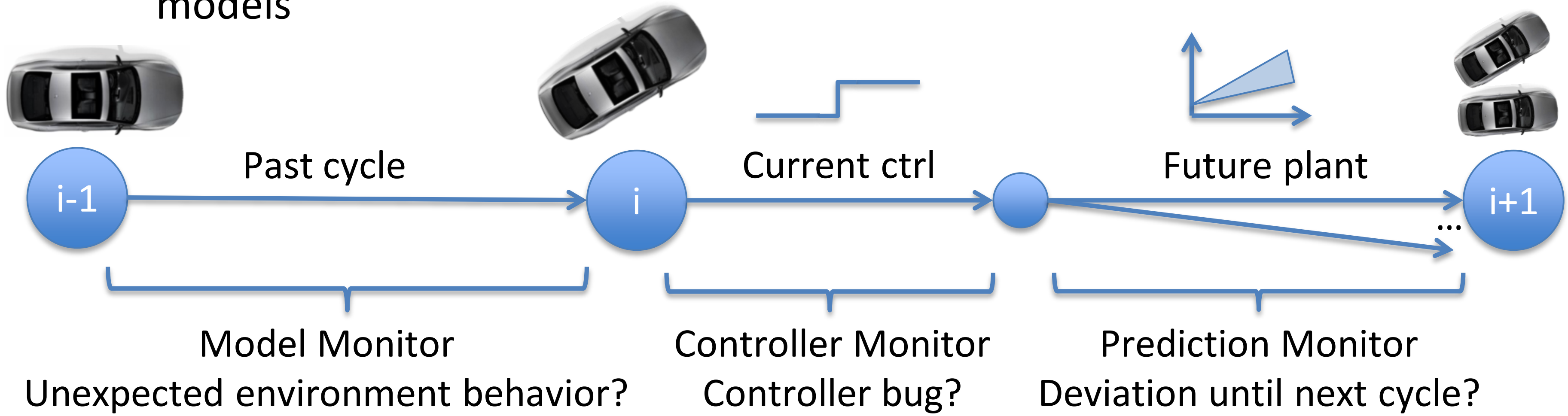


Organization

- Overview
- Approach
- Example: Dynamic Window Monitor
- More HCOL examples
- **Other research components**
- Demos
- Concluding remarks

ModelPlex Runtime Validation

- ModelPlex **ensures that proofs about models apply to real CPS**
- **Synthesize** provably correct monitors **to check CPS at runtime**
- Correct-by-construction monitor conditions instead of manual annotation in models



Offline Synthesis by Theorem Proving

Semantic $(x^-, x^+) \in \rho(\alpha)$

$\Updownarrow$

Logic (dL) $(x = x^-) \rightarrow \langle \alpha_{(x)} \rangle (x = x^+)$

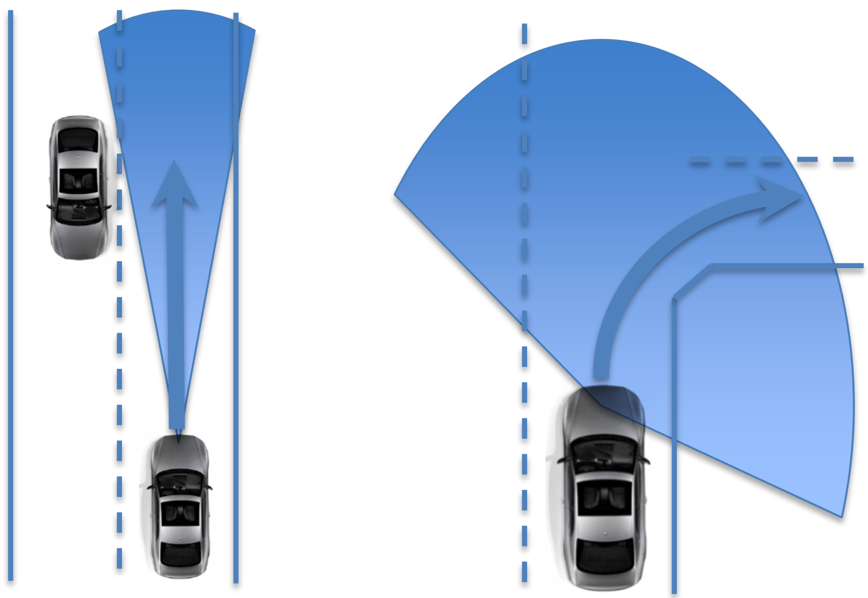
Starting at x=x⁻, exists a run of model α to state x=x⁺?

$\Uparrow$

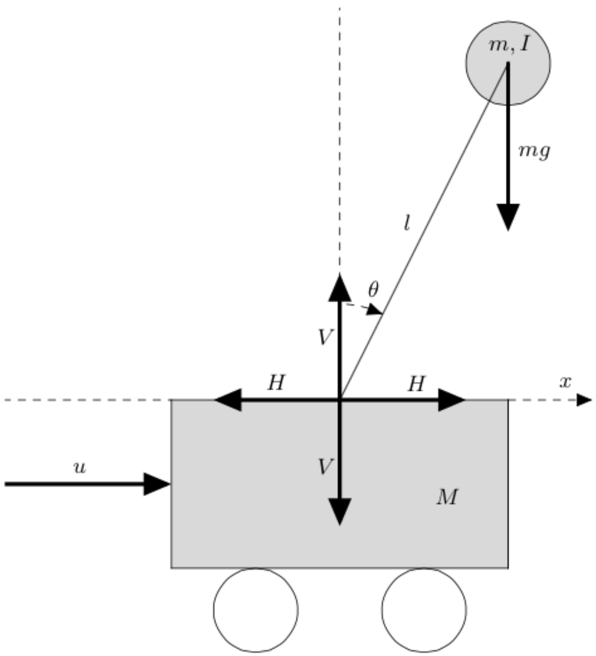
Arithmetic $F(x^-, x^+)$ Check at Runtime (efficient)

Directional Collision Avoidance

- **Field of view** and **orientation**
  - Vehicle only **responsible** for collisions **inside field of view**
  - Allows **more aggressive driving:** ignores obstacles outside visible area
  - **Narrow vision** cone **on straight lanes**: fast with limited steering
  - **Broad** vision cone at **intersections:** sharp turns at slow speed
- Multiple **obstacle kinds**
  - Pedestrians vs. other cars
  - Moveable vs. stationary
- Safety despite **velocity uncertainty**



Formal Verification of PD Controller: Inverted Pendulum



Hybrid Model

$$\mathrm{m3} \equiv (\mathrm{ctrl}_{\mathrm{m3}};\ \mathrm{plant}_{\mathrm{m3}})^*$$
$$\mathrm{ctrl}_{\mathrm{m3}} \equiv r := *;\ ?\,(\varphi_{m3} \wedge \zeta_{\mathrm{m3}})$$
$$\mathrm{plant}_{\mathrm{m3}} \equiv \{\theta' = \omega,\ \omega' = a(\theta - r) + b\omega\}$$

Automatically Derived Safety Conditions

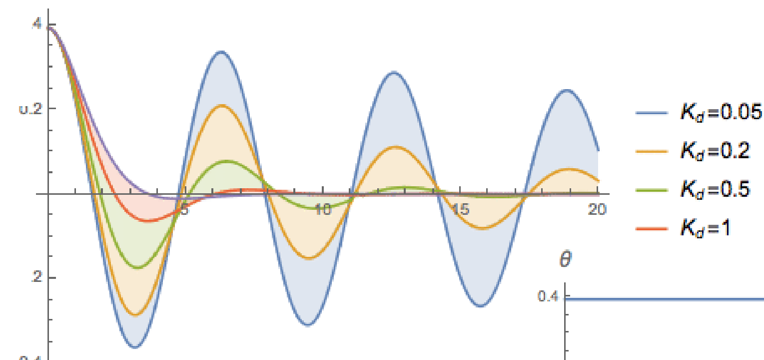
$$: K_p > (M+m)g$$
$$\wedge\, 0 < K_d < 2Ml\sqrt{\frac{K_p - (M+m)g}{Ml}}$$

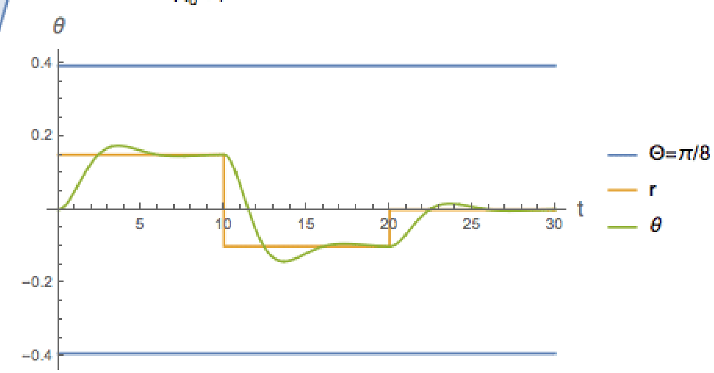
Dynamics

$$\left(I + \frac{mMl^2}{M+m}\right)\theta'' + \frac{ml}{M+m}u = mgl\theta$$

PD Controller

$$u(t) = K_p\theta + K_d\theta'$$





Simulations

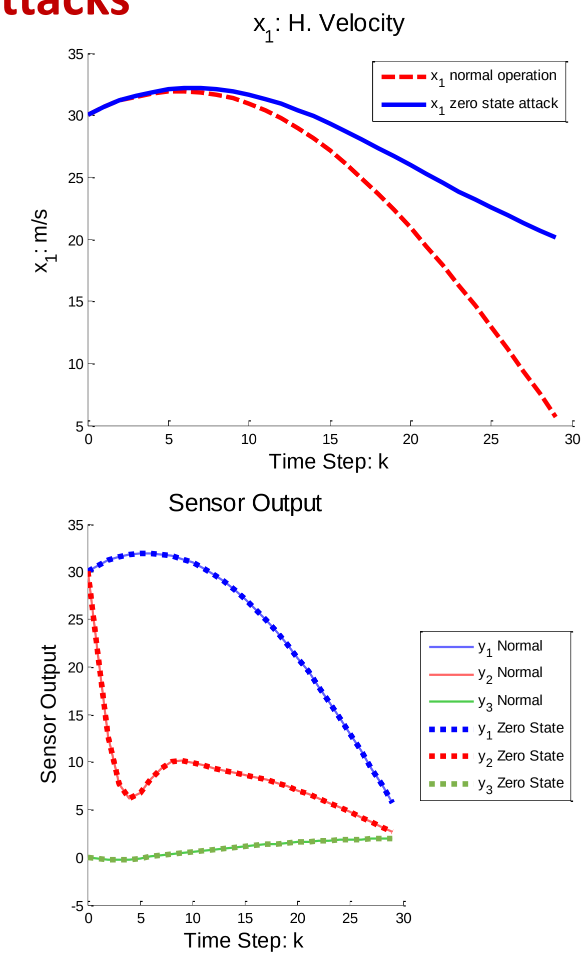
A. Platzer, J.-B. Jeannin, and K. Ghorbal

Detection of Actuator + Sensor Attacks

$$x(k+1) = Ax(k) + Bu(k) + \Gamma e(k),$$ ← **Actuator attacks**

$$y(k) = Cx(k) + \Psi e(k).$$ ← **Sensor attacks**

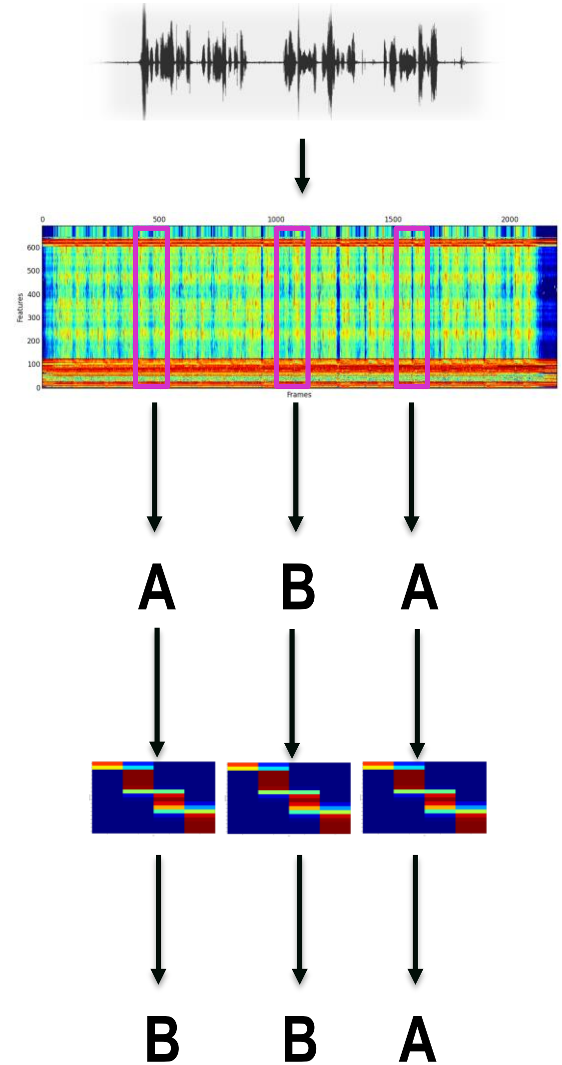
- **Limitations of attack detection addressed as geometric control problems**
- **Detector performance depends on knowledge of system initial state**
- **One form of attack is undetectable when detector exactly knows system initial state:**
  - Changes system's physical state (e.g. true velocity)
  - Does NOT change system sensor output (e.g., Odometer reading)



Attack against remotely piloted aircraft

S. Kar, J. Moura, and Y. Chen

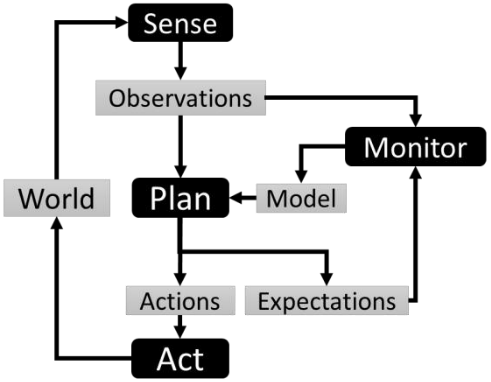
Estimating ABCar's Speed From Audio



- Recorded at HRL
- Multi microphone setup
- Audio classification
- Physics constraints (gear vs. speed)
- Good speed estimate (±2.5 km/h)

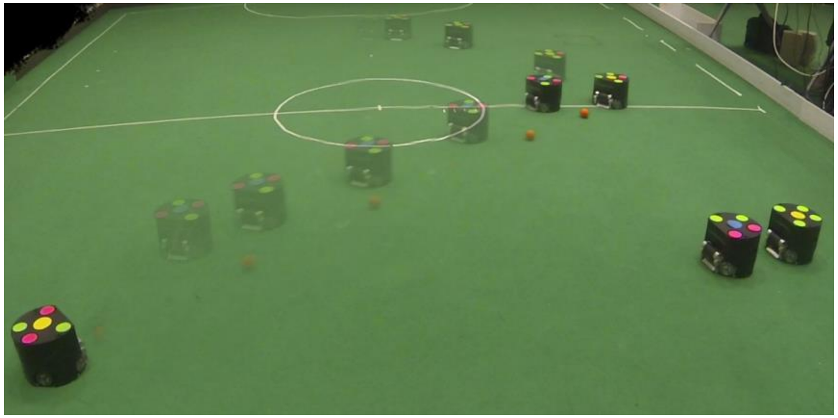
Adversaries and Space Inhomogeneity

Execution Monitoring

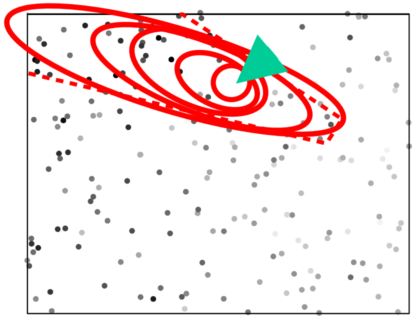


Online execution monitoring to correct planning models about an adversary

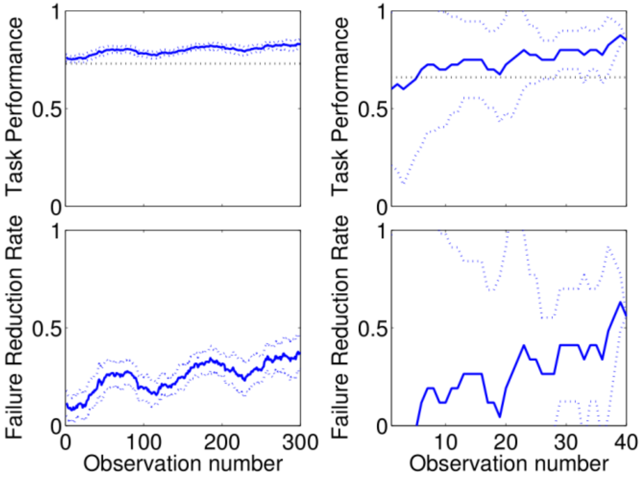
Demonstrated in Robosoccer



Online Detection of Anomalies



$$\text{anom}(R) = \frac{\text{P}(obs|\text{maximum likelihood anomaly in } R)}{\text{P}(obs\ |\text{no anomaly in } R)}$$

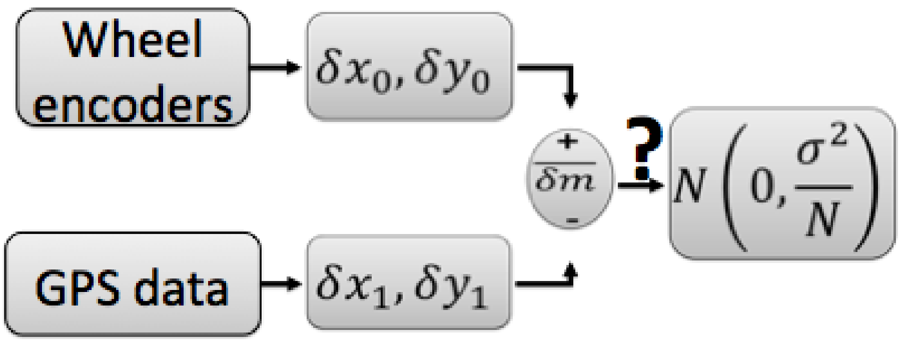


(a) Simulation (b) Real robots

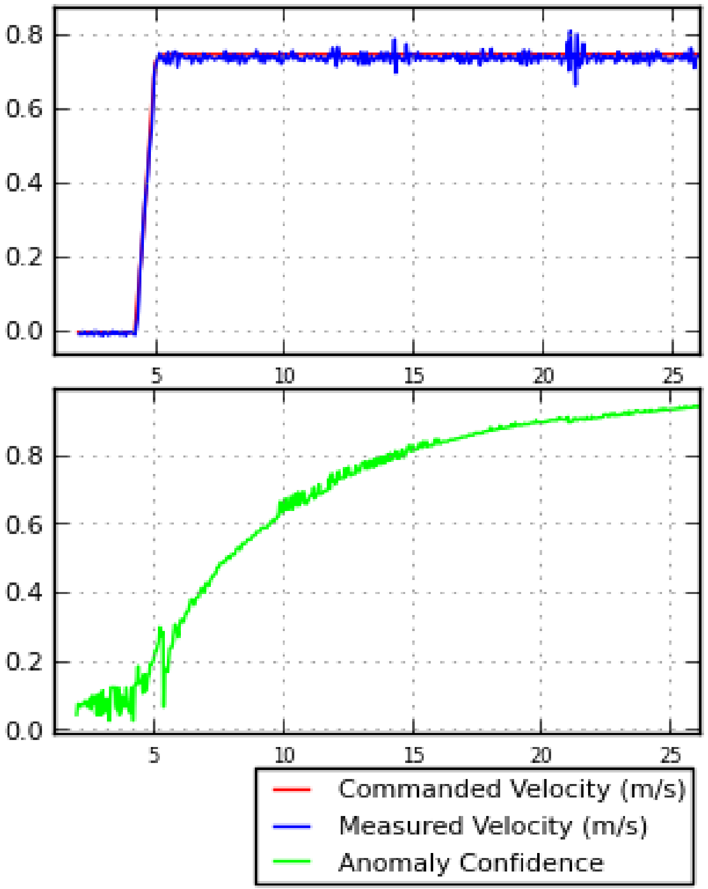
M. Veloso, J. P. Mendoza, and R. Simmons

Sensor Fusion And Data Consistency

Abnormal := not normal



Confidence in data



Learn state-dependent bad data

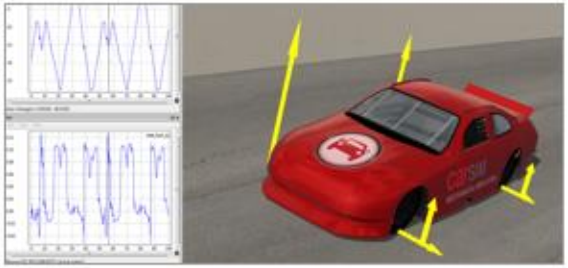
Set-Bases Sensor Inconsistency Checks

- Fuse wheel encoders and GPS to detect inconsistencies
- Models noise and attack (strength, type)
- Matlab implementation calibrated with Carsim runs

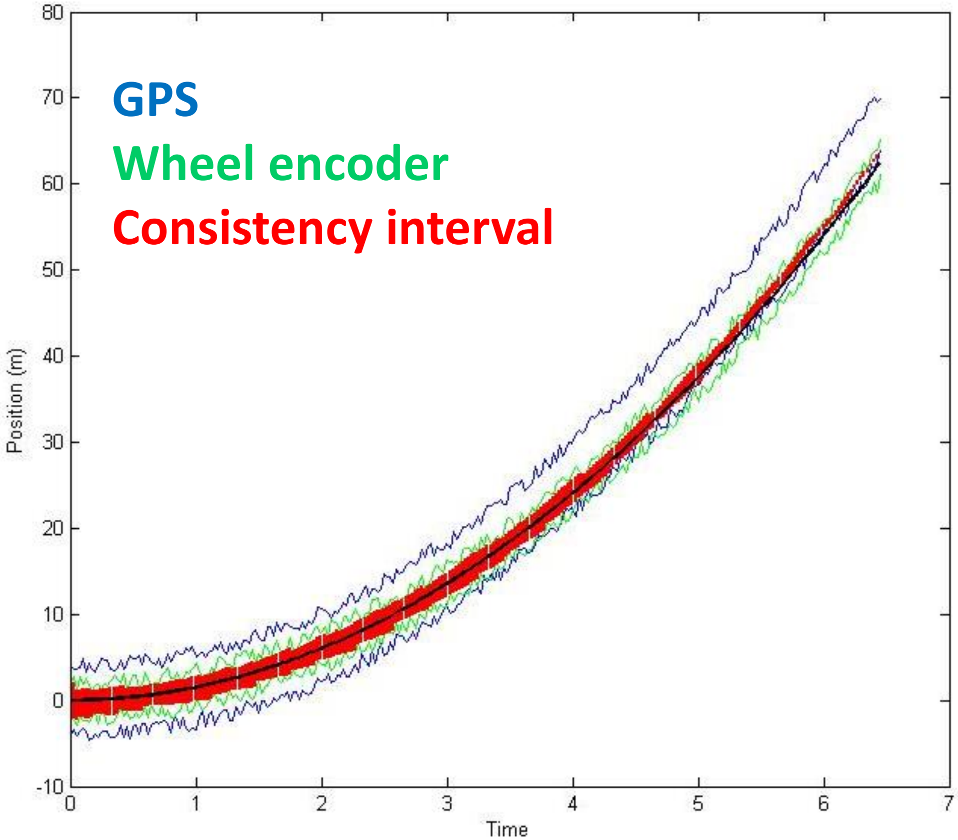
Physics and noise model

$$p((k+1)\Delta) = p(k\Delta) + \int_{k\Delta}^{(k+1)\Delta} v(s)ds$$

$$y(k\Delta) = Hx(k\Delta) + v(k\Delta) + b(k\Delta)$$



Accelerating car with slight GPS attack



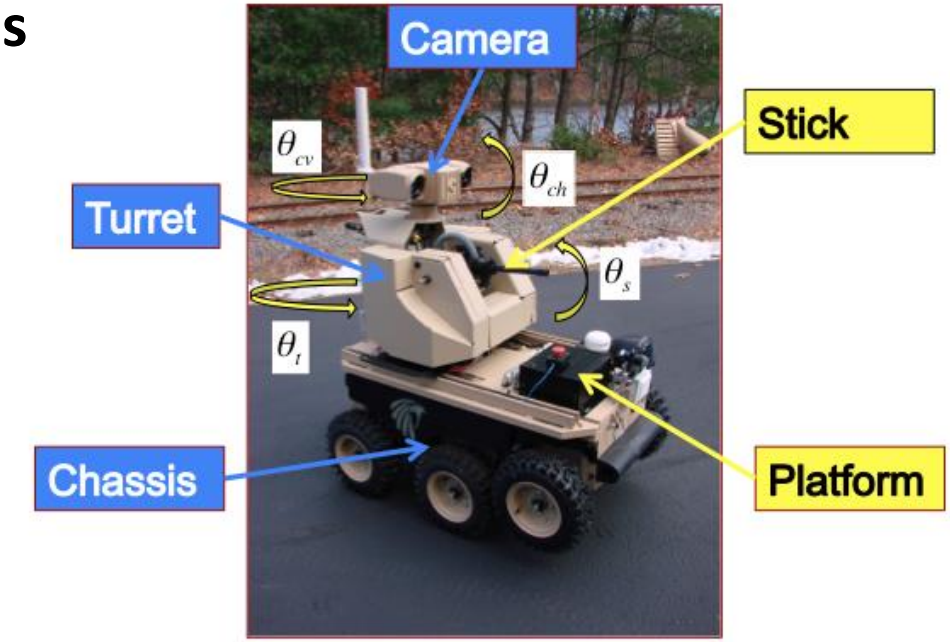
S. Kar, F. Franchetti, A. Sandryhaila, and T.M. Low

Camera/Image Sensor Consistency

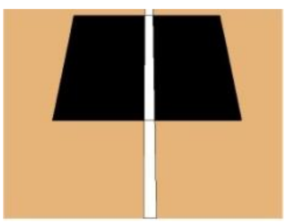
Projection and rotation matrices

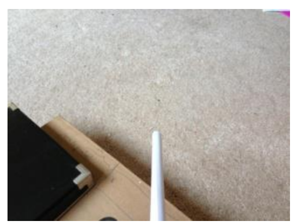
$$M = KR[I, -C]$$

$$R = R_Y(q_t + q_{cv})R_X(q_{ch})$$



Cartoon and real image





Consistency check: compare cartoon image and camera image

Rotation Parameters (by sensors)
+
Dimension Parameters of LandShark
+
Camera Parameters
→ Cartoon Images
+
Real Images (by Camera)
→ Consistency

Organization

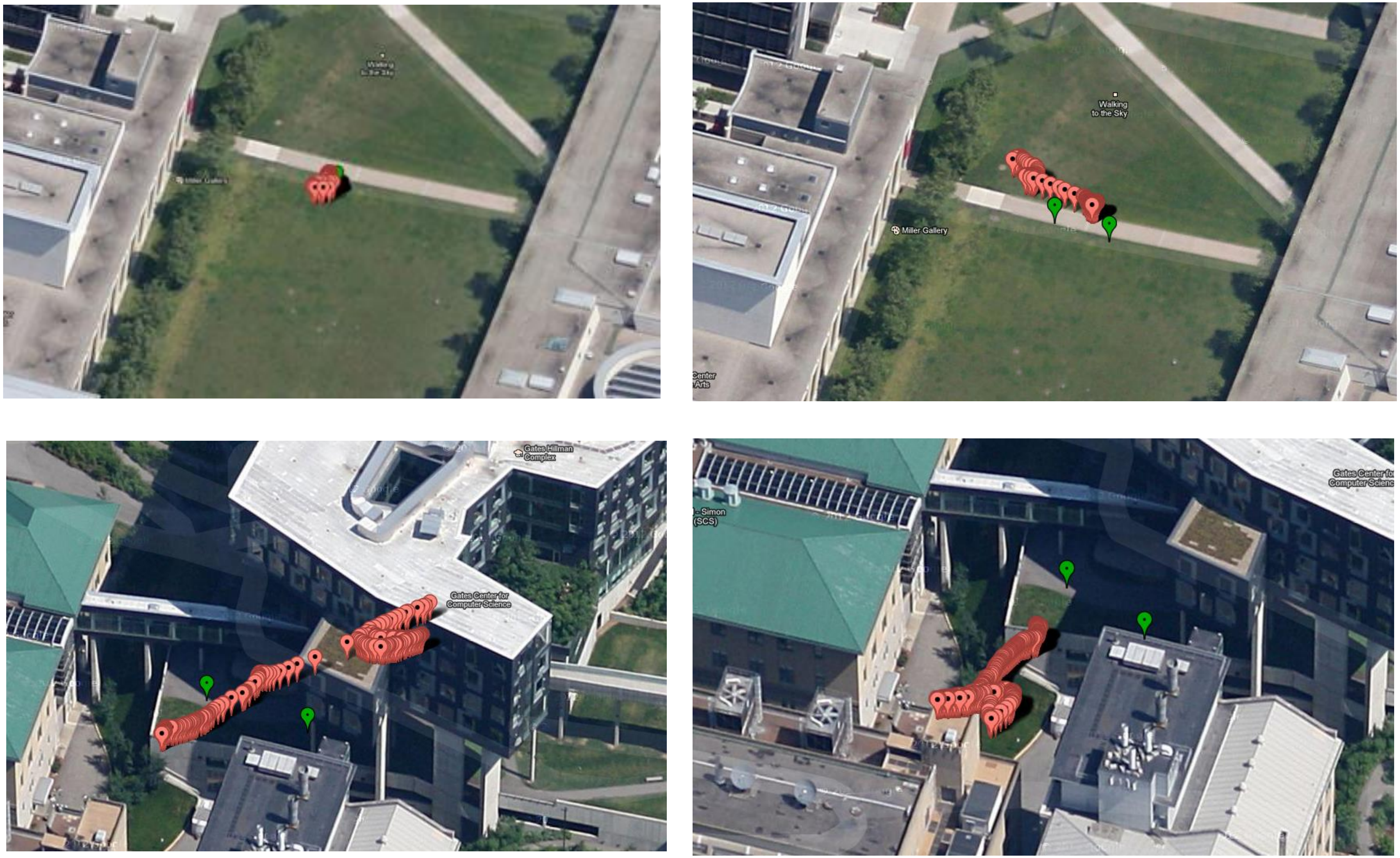
- Overview
- Approach
- Example: Dynamic Window Monitor
- More HCOL examples
- Other research components
- **Demos**
- Concluding remarks

HACMS Phase 1 Demo on Landshark

- **Setup:** Drive Landshark with/without spoofing detection and obstacle avoidance ,
  show impact of drive error and GPS attack
- **Attack:** Drift GPS to drive Landshark into obstacle while obstacle avoidance is engaged. Then show defense.
- **Tool:** Code synthesized with HA Spiral and KeYmaera/Sphinx
  - **Run 1:** no spoofing, no obstacle avoidance
  - **Run 2:** obstacle avoidance on
  - **Run 3:** obstacle avoidance, GPS spoofing attack
  - **Run 4:** obstacle avoidance + spoofing detection


1. No Monitors

Calibrating The LandShark GPS



Landshark Waypoint GPS Following

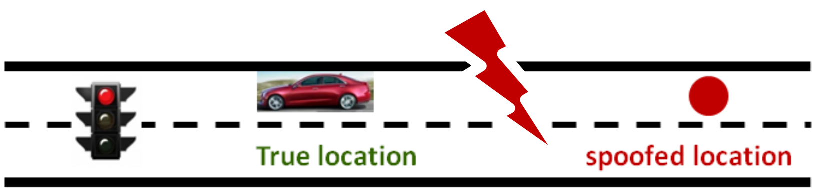
Organization

- Overview
- Approach
- Example: Dynamic Window Monitor
- More HCOL examples
- Other research components
- Demos
- **Concluding remarks**

Summary: High Assurance Spiral

Problem and main idea

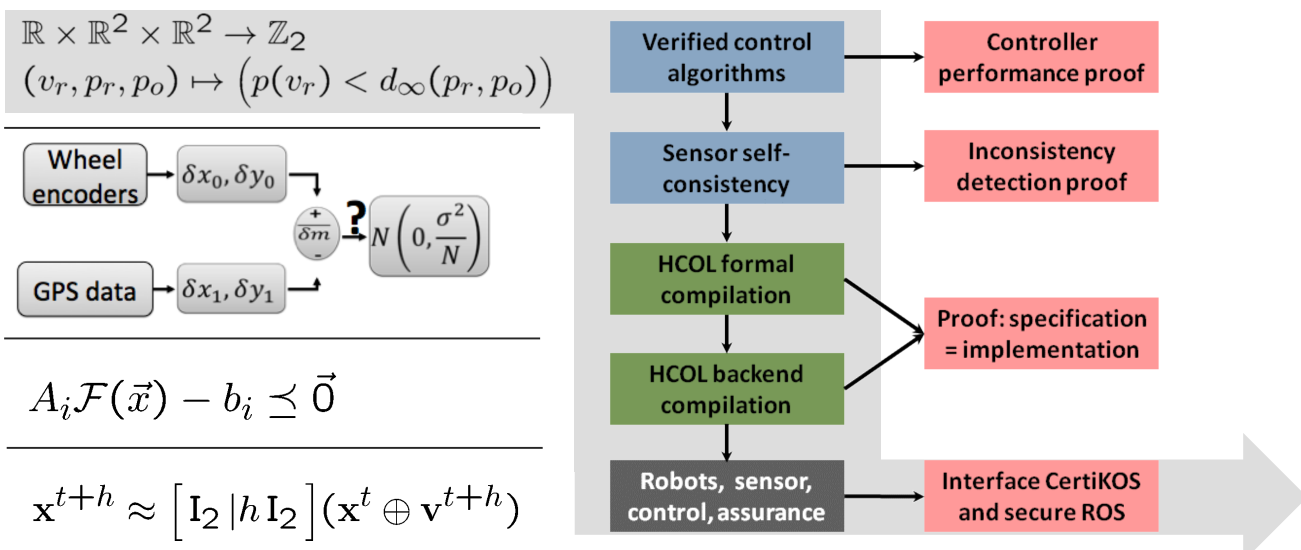
Co-synthesize high-quality code and proof for sensor-fusion based self-consistency algorithms



Results

- **Four algorithms in HA Spiral formalized/in library**
  dynamic window monitor, statistical tests, feasible state set test, infrastructure math code
- **HA Spiral Tool/GUI**
  ready for beta testers
- **End-to-end proof/code co-synthesis and deployment**
  deployed on Landshark and ABCar Simulator
- **Rule based backend compiler proof of concept**
  Spiral/Coq interface

Approach




```
int dwmonitor(float  *X, double  *D)
    __m128d u1, u2, u3, u4, u5, u6,
    unsigned _xm = _mm_getcsr();
    _mm_setcsr(_xm & 0xffff0000 | 0x
    u5 = _mm_set1_pd(0.0);
    u2 = _mm_cvtps_pd(_mm_addsub_ps(
    _mm_set1_ps(FLT_MIN), _mm_set1_p
    u1 = _mm_set_pd(1.0, (-1.0));
    for(int i5 = 0; i5 <= 2; i5++) {
        x6 = _mm_addsub_pd(_mm_set1_
            +DBL_MIN)), _mm_loaddup_
        x1 = _mm_addsub_pd(_mm_set1_
        x2 = _mm_mul_pd(x1, x6);
    ...
        }
    __asm nop;
    if (_mm_getcsr() & 0x0d) {
        _mm_setcsr(_xm);
```

Acknowledgement

S. Kar (PI), J. Moura (PI), A. Platzer (PI), M. Veloso (PI)
Y. Chen, F. Faruq, K. Ghorbal , L. Gui, C. Van den Hauwe, J.-B. Jeannin, T. M. Low, J.P. Mendoza, S. Mitsch, J.-D. Quesel, A. Sandryhaila, R. Veras, V. Zaliva
Carnegie Mellon University

J. Johnson (PI), LC Meng
Drexel

D. Padua (PI), A. Phaosawasdi, S. Seo
UIUC

M. Franusich (PI), B. Duff, J. Larkin
SpiralGen, Inc.

More Information:

www.spiral.net

www.spiralgen.com