STAE TOULOUSE
FONDATION DE COOPÉRATION SCIENTIFIQUE
Sciences et Technologies pour l'Aéronautique et l'Espace

Vérification et Validation des systèmes de commande et guidage des systèmes aéronautiques

Eric Feron
Dutton/Ducoffe Professor of Aerospace Engineering
Georgia Institute of Technology

En visite a: ENSEEIHT/INPToulouse
feron@gatech.edu

INP TOULOUSE ENSEEIHT n7 Georgia Tech College of Engineering Daniel Guggenheim School of Aerospace Engineering

Take-Home Message

Safety-critical embedded software design best tackled through proper specification, followed by automatic coding of specs AND their semantics

This is the best mechanism to leverage domain-specific knowledge

Examples: Control, collision avoidance systems

Outline

- Motivation/background
- Decision and Control Laboratory
- A simple control example
- Stability and performance analyses: Why go beyond specs and into implementation?
- What proofs for what system representations?
- Journal proofs, block diagram proofs, program proofs
- Closed-loop system properties
- Tool implementation

Safety-critical software

- Software that interacts in real time with physical system (usually big-heavy and/or very costly and/or super-dangerous) and possibly humans.
- Aircraft
- Rockets
- Missiles
- Radiotherapy machines

Some examples of why you should care

Facts

- How many lines of code produced by average software engineer for spacecraft applications

  0.6 Lines Of Code Per Hour
- F22 Raptor: 1.7M LOC
- F35 JSF: 5.7M LOC
- Boeing 787: 6.5M LOC

Accidents/Incidents

- “Some of the most widely cited *software*-related *accidents* in safety-critical systems involved a computerized *radiation therapy* machine called the Therac-25.”
- “The new US stealth fighter, the F-22 Raptor, was deployed for the first time to Asia earlier this month. On Feb. 11, twelve Raptors flying from Hawaii to Japan were forced to turn back when a software glitch crashed all of the F-22s' on-board computers as they crossed the international date line.”

Accidents/Incidents Ariane 5

- “The Ariane 5 software reused the specifications from the Ariane 4, but the Ariane 5's flight path was considerably different and beyond the range for which the reused computer program had been designed. Specifically, the Ariane 5's greater acceleration caused the back-up and primary inertial guidance computers to crash, after which the launcher's nozzles were directed by spurious data.”

Patriot disaster

- (1) the Patriot battery at Dhahran failed to track and intercept a Scud missile due to a software problem in the system's weapons control computer; (2) the software problem caused an inaccurate tracking calculation which became worse the longer the system operated; (3) at the time of the incident, the battery had operated continuously for over 100 hours and the inaccuracy was serious enough to cause the system to look in the wrong place for the incoming Scud;

(The scud killed 21 friendly soldiers)

Remedies: Analyses

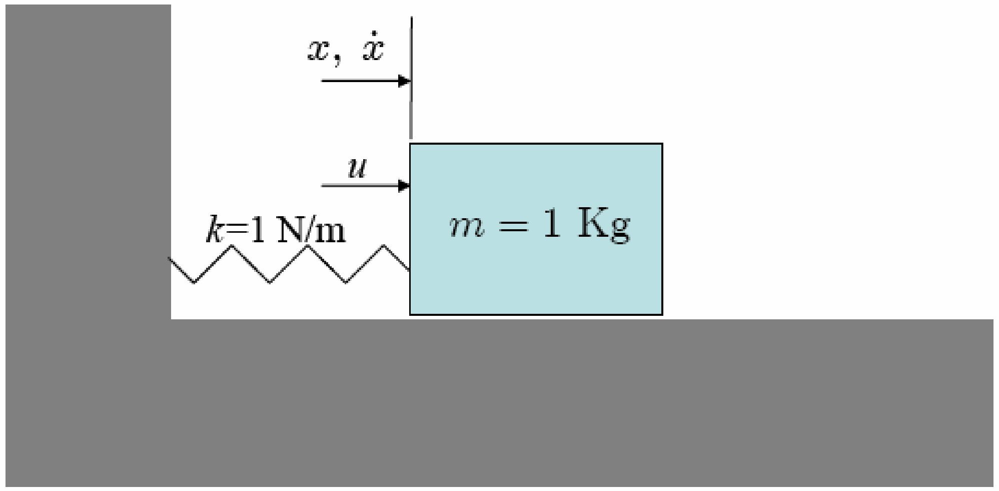
- Simulation OK: SIL, HIL.
- Enormous efforts devoted to static *program analysis*
  - Model Checking (Sifakis/Clarke/Holzmann)
  - Abstract Interpretation (Cousot, Cousot)
  - WCET analysis
  - PVS (Sankar, Owre, Rushby)
- Very strong appetite for code as input to analyzers…
- 100’s of current applications, including at ENAC
- Airbus A340/380, Ariane 5 (a posteriori)
- ENAC’s Paparazzi on NASA’s list of static analysis milestones in VVFCS program
- DO178C acknowledges power of formal methods

Remedies: Design

- Most errors arise during specification of software, not coding.
- Allow the engineer to specify, then auto-code.
- SCADE/Esterel Technologies, Picture2code/Pratt & Whitney, Realtime Workshop/Mathworks, Gene-auto/ENSEEIHT, Gryphon/Rockwell-Collins.

How do we reconcile analysis and design?

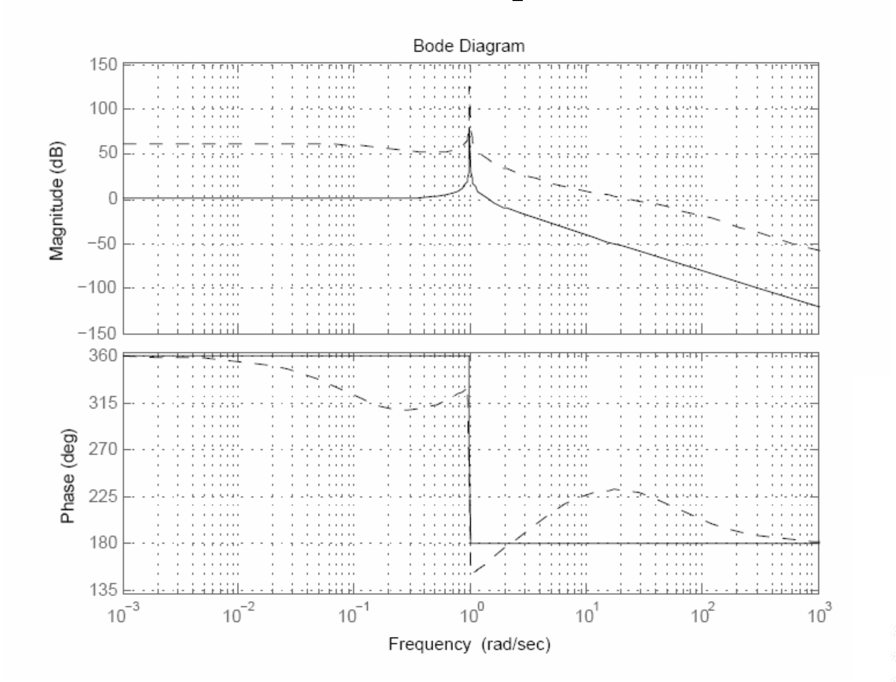
A simple control example



$$\frac{d}{dt}\begin{bmatrix} x \\ \dot{x} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}\begin{bmatrix} x \\ \dot{x} \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u, \quad x(0) = x_0, \dot{x}(0) = \dot{x}_0$$

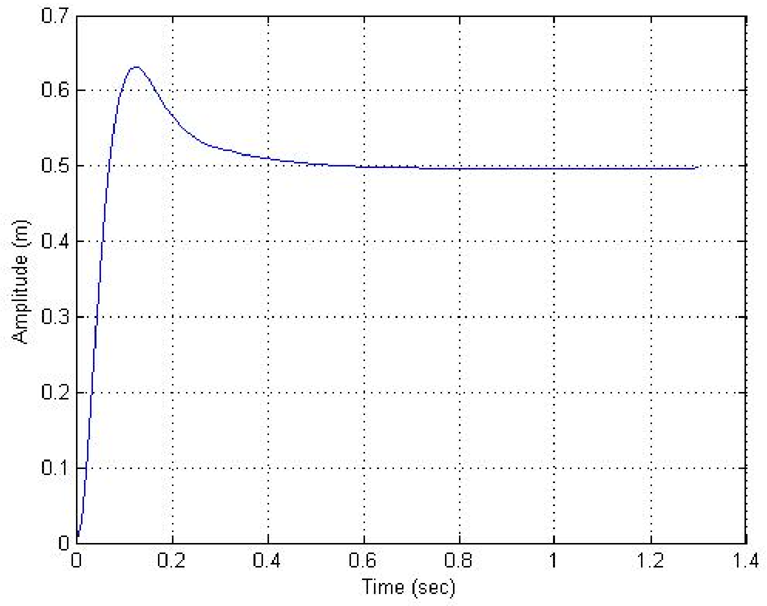
$$y = [1 \ \ 0]\begin{bmatrix} x \\ \dot{x} \end{bmatrix}.$$

A simple control example



$$\tilde{y}(t) = \mathbf{SAT}(y(t)),$$
$$u(s) = 128\frac{s+1}{s+0.1}\,\frac{s/5+1}{s/50+1}\tilde{y}(s),$$

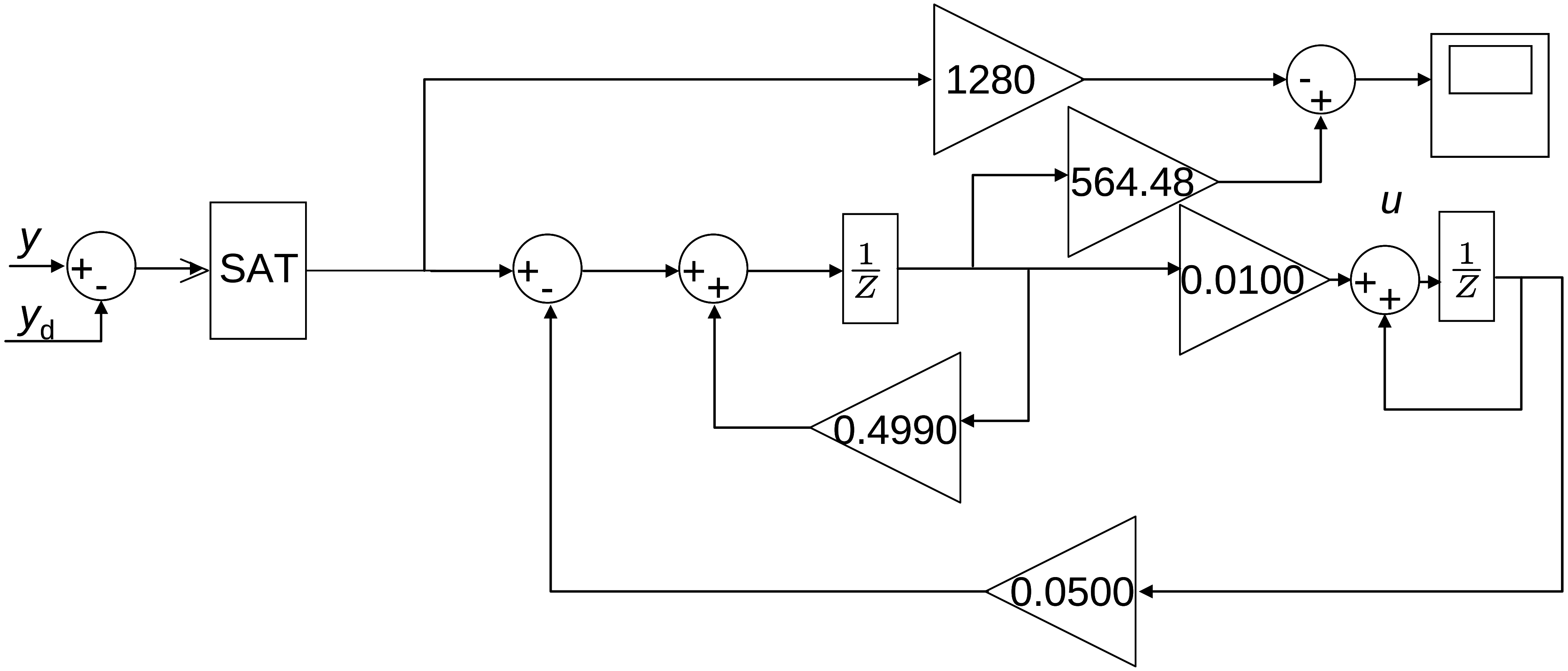
Step response



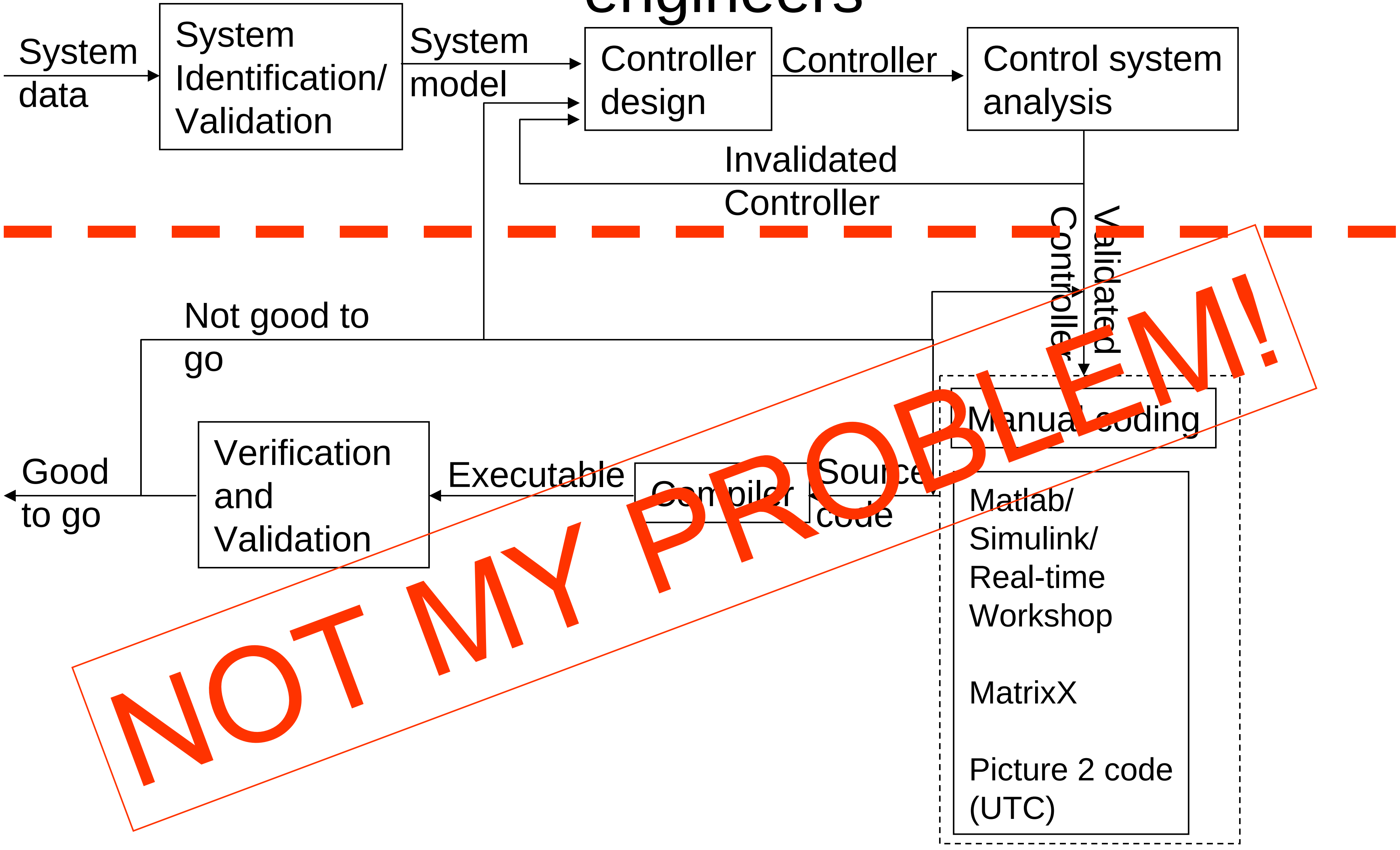
Controller implementation

$$\begin{aligned} x_{c,k+1} &= \begin{bmatrix} 0.499 & -0.050 \\ 0.010 & 1.000 \end{bmatrix} x_{c,k} + \begin{bmatrix} 1 \\ 0 \end{bmatrix} \mathbf{SAT}(y_k) \\ u_k &= -\left[564.48 \;\; 0\right] x_{c,k} + 1280\, \mathbf{SAT}(y_k) \end{aligned}$$

Discrete time Implementation 100Hz

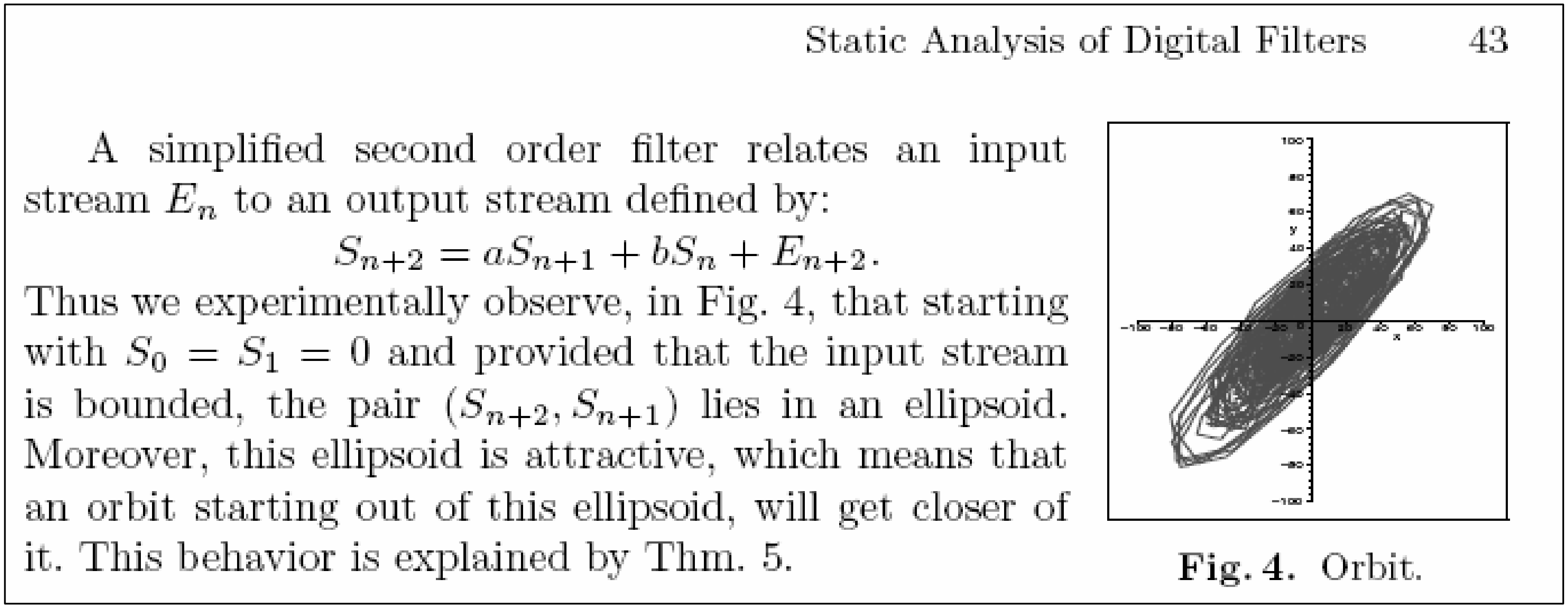


Control system design as seen by control engineers



Code-level analyses of control software

- Most significant contribution is from Patrick Cousot’s research group at Ecole Normale Superieure, Paris.
- Abstract interpretation aims at capturing semantics of programs
- Most important application is ASTREE analyzer for Airbus A380 control code.
- From Feret, “Static Analysis of Digital Filters”, 2004 (also with ASTREE).



A simplified second order filter relates an input stream E_n to an output stream defined by:

$$S_{n+2} = aS_{n+1} + bS_n + E_{n+2}.$$

Thus we experimentally observe, in Fig. 4, that starting with $S_0 = S_1 = 0$ and provided that the input stream is bounded, the pair (S_{n+2}, S_{n+1}) lies in an ellipsoid. Moreover, this ellipsoid is attractive, which means that an orbit starting out of this ellipsoid, will get closer of it. This behavior is explained by Thm. 5.

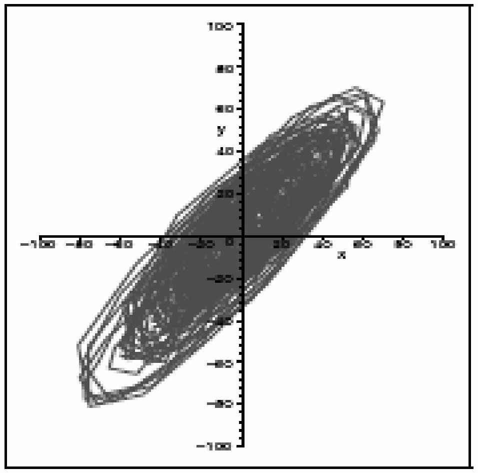
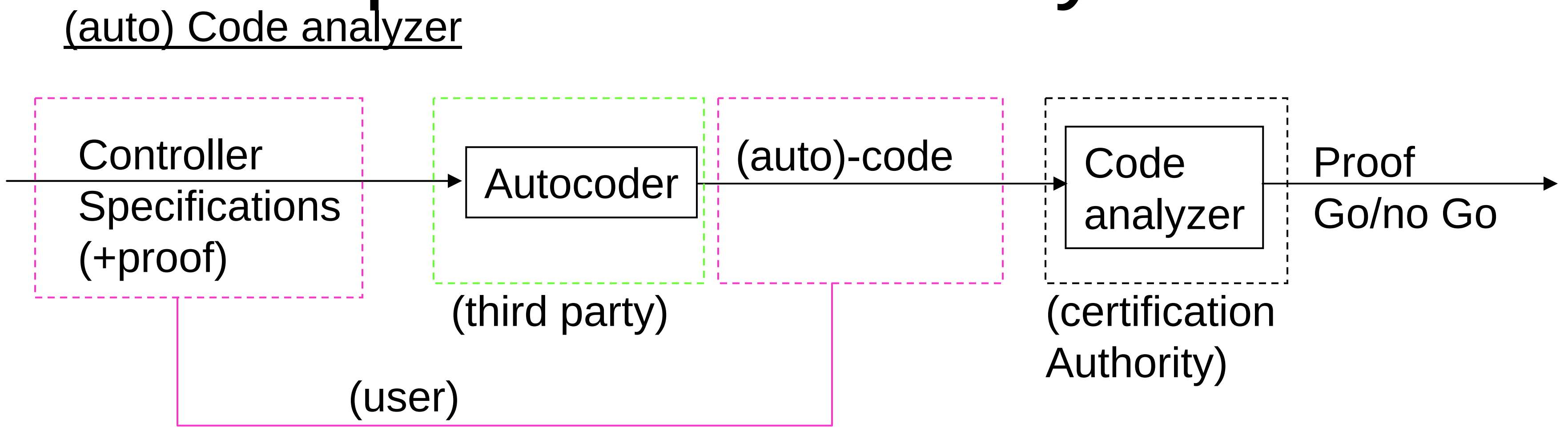
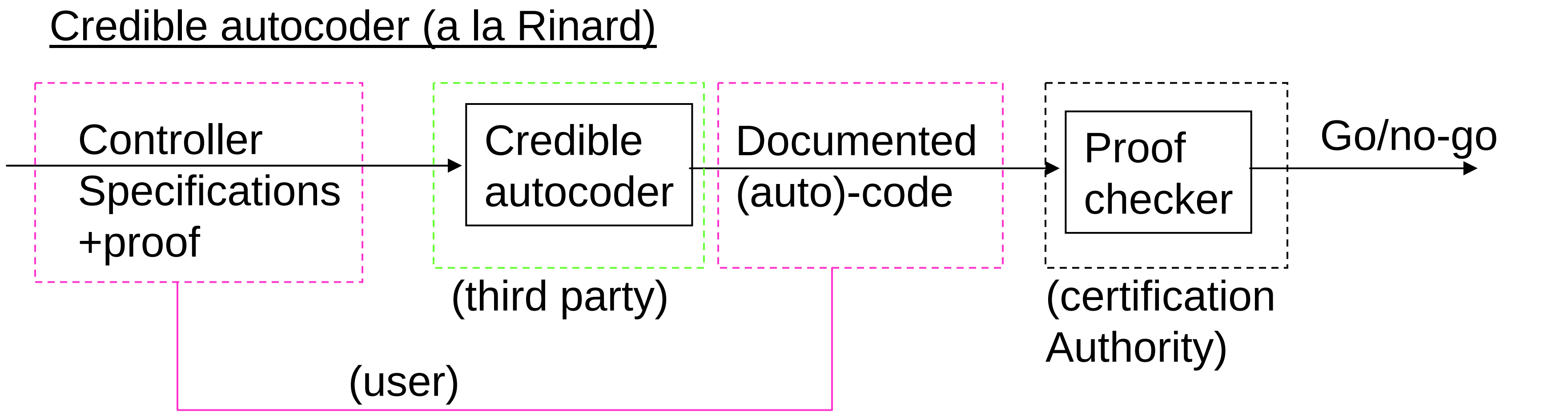


Fig. 4. Orbit.

A Paradigm Shift Enabled by Good Specification Analyses




Credible autocoder (a la Rinard)
Controller Specifications +proof
Credible autocoder
Documented (auto)-code
Proof checker
Go/no-go
(third party)
(certification Authority)
(user)

Desirable attributes of “system proofs”

- Must be expressive enough to tell nontrivial statements about system
- Must speak the language of system representation, eg: “IEEE Transactions on Automatic Control proofs” written in natural language (one wonders…), “Simulink proofs” expressed in Simulink, “Program proofs” expressed in formal languages.
- Must be “elementary enough” to be easily checked wherever necessary.

Back to the Example

The control-systemic way:

$$
\begin{aligned}
x_{c,k+1} &= \begin{bmatrix} 0.499 & -0.050 \\ 0.010 & 1.000 \end{bmatrix} x_{c,k} + \begin{bmatrix} 1 \\ 0 \end{bmatrix} \mathbf{SAT}(y_k) \\
u_k &= -\left[564.48 \;\; 0\right] x_{c,k} + 1280\, \mathbf{SAT}(y_k)
\end{aligned}
$$

Assume the controller state is initialized at $x_{c,0} = 0$

What range of values could be reached by the state $x_{c,k}$ and the control variable u_k?

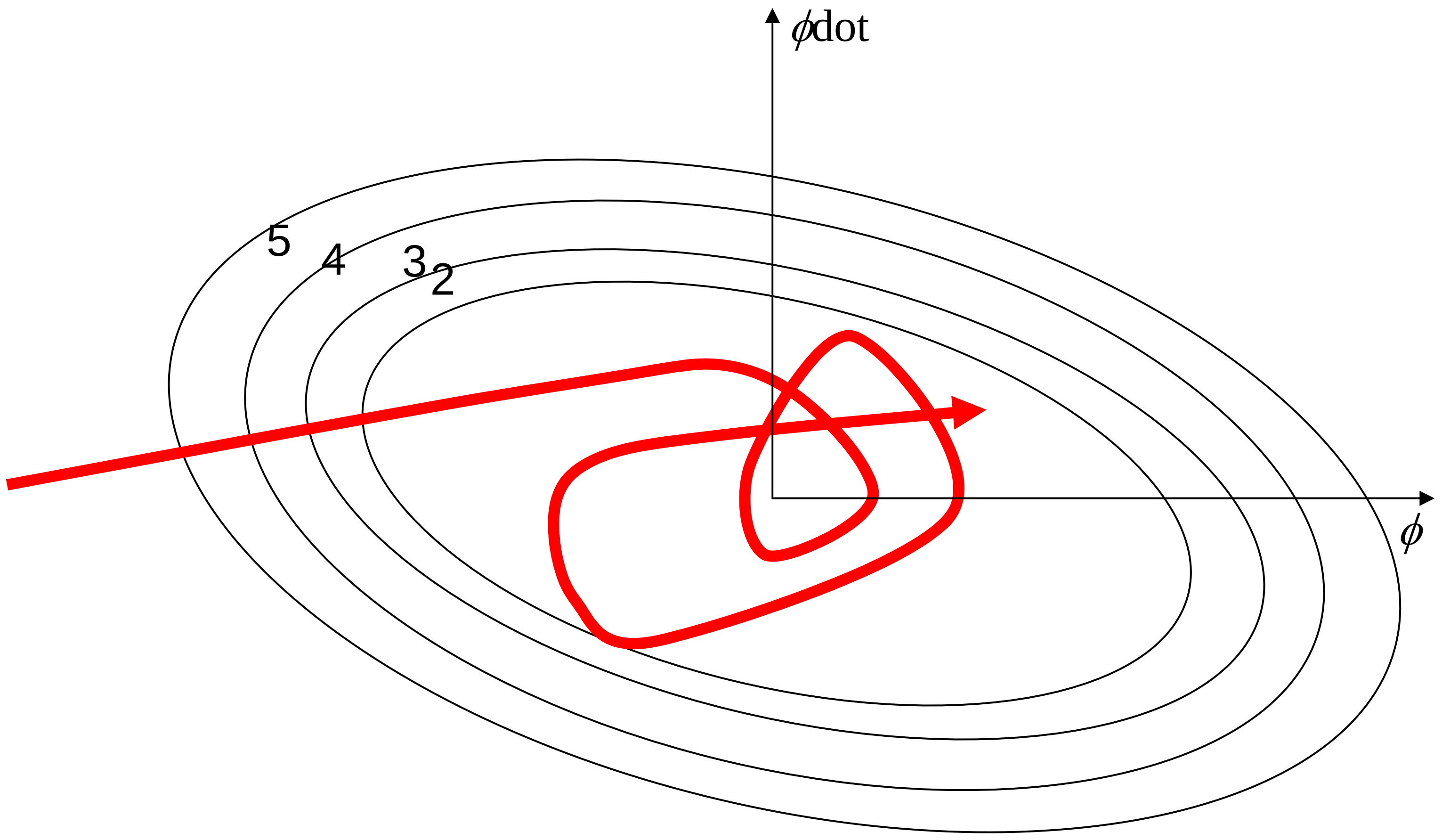
There is a variety of options, including computation of -1 norms.

A Lyapunov-like proof (from Boyd *et al.*, Poola*):*

The ellipsoid $\mathcal{E}_P = \left\{ x \in \mathbf{R}^2 \mid x^T P x \leq 1 \right\}.$ $\quad P = 10^{-3} \begin{bmatrix} 0.6742 & 0.0428 \\ 0.0428 & 2.4651 \end{bmatrix}.$

is invariant. None of the entries of *x* exceeds 7 in size.

Lyapunov functions and invariant ellipses



A proof for control people

$\forall t,\ x^T P x \le 1$ is equivalent to $x_k^T P x_k \le 1 \Rightarrow x_{k+1}^T P x_{k+1} \le 1$

Or $(Ax+Bw)^T P(Ax+Bw) \le 1$ whenever $x^T P x \le 1$ and $w^2 \le 1$

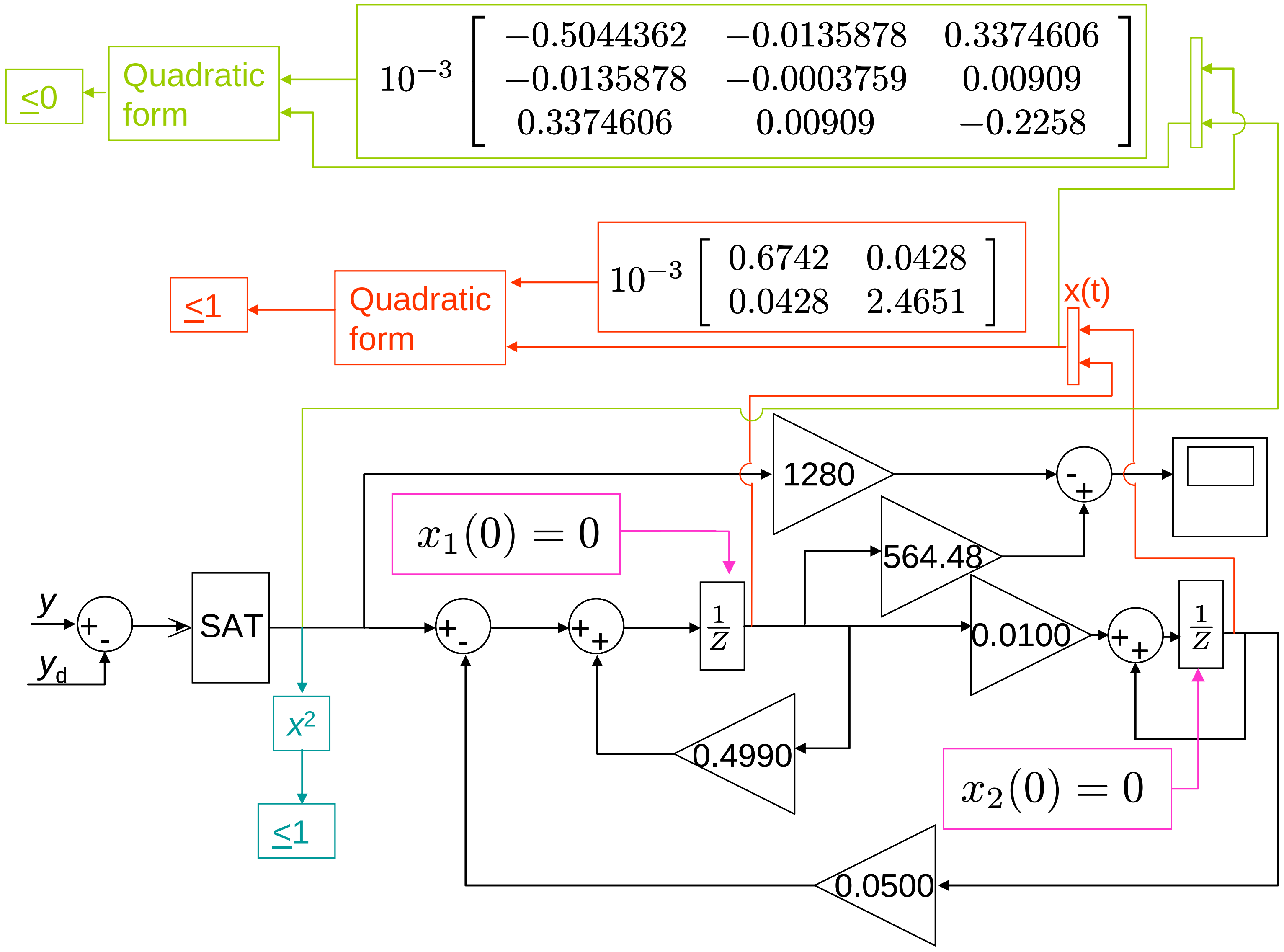
True if there exists μ such that $(Ax+Bw))^T P(Ax+Bw) - \mu x^T P x - (1-\mu) w^2 < 0$, (*) a tautology.

Indeed a linear combination of (*) and $x^T P x \le 1$ and $w^2 \le 1$ yields the desired property.

P that works is $P = 10^{-3} \begin{bmatrix} 0.6742 & 0.0428 \\ 0.0428 & 2.4651 \end{bmatrix}$, with $\mu = 0.9991$ and tautology

(*) is $10^{-3} \begin{bmatrix} x \\ w \end{bmatrix}^T \begin{bmatrix} -0.5044362 & -0.0135878 & 0.3374606 \\ -0.0135878 & -0.0003759 & 0.00909 \\ 0.3374606 & 0.00909 & -0.2258 \end{bmatrix} \begin{bmatrix} x \\ w \end{bmatrix} \le 0.$

Simulink, Discrete Time Formal Semantics

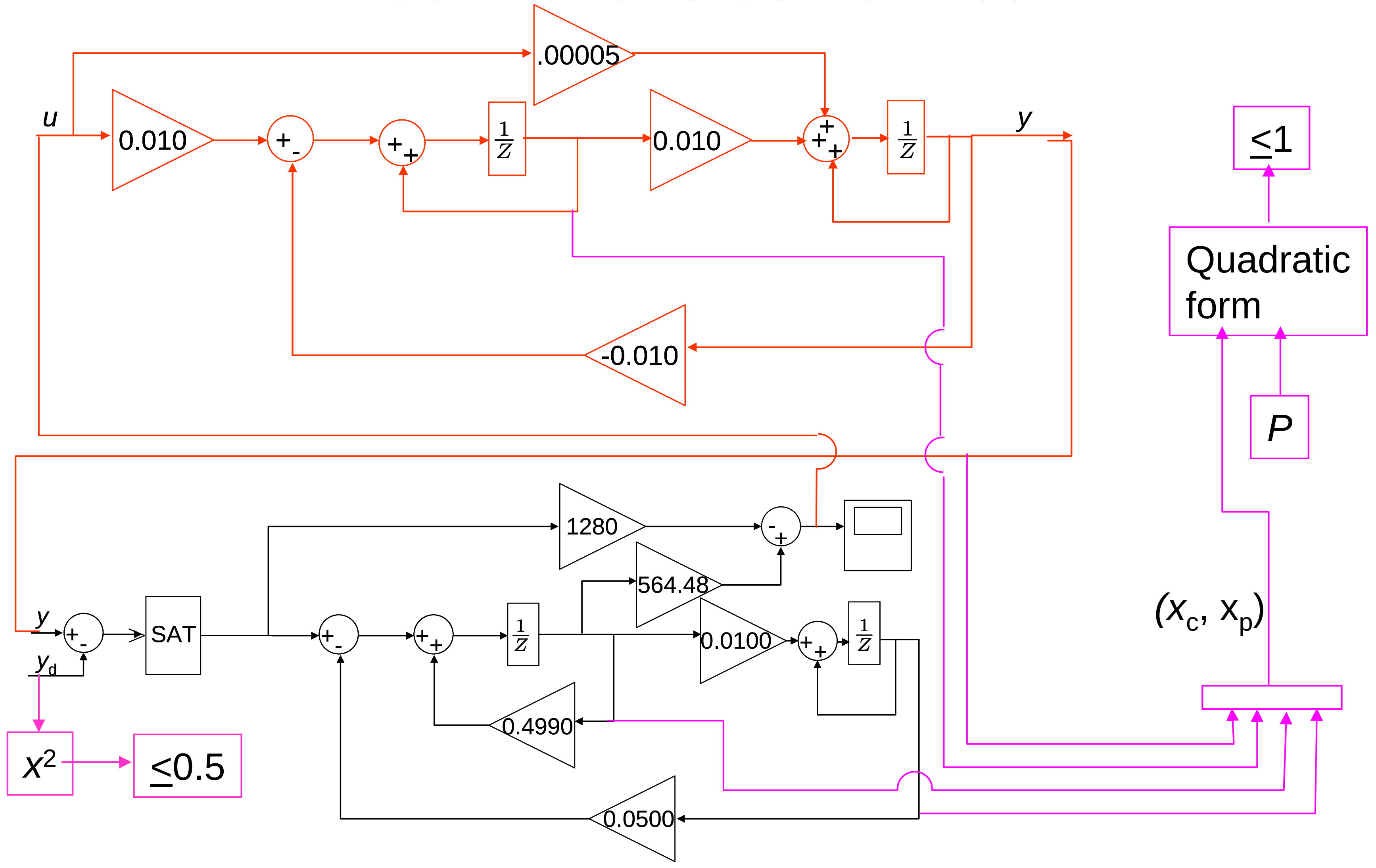


$10^{-3}\begin{bmatrix} -0.5044362 & -0.0135878 & 0.3374606 \\ -0.0135878 & -0.0003759 & 0.00909 \\ 0.3374606 & 0.00909 & -0.2258 \end{bmatrix}$
Quadratic form
≤ 0
$10^{-3}\begin{bmatrix} 0.6742 & 0.0428 \\ 0.0428 & 2.4651 \end{bmatrix}$
Quadratic form
≤ 1
x(t)
y
y_d
SAT
x^2
≤ 1
$x_1(0) = 0$
$\frac{1}{Z}$
1280
564.48
0.0100
0.4990
$x_2(0) = 0$
0.0500

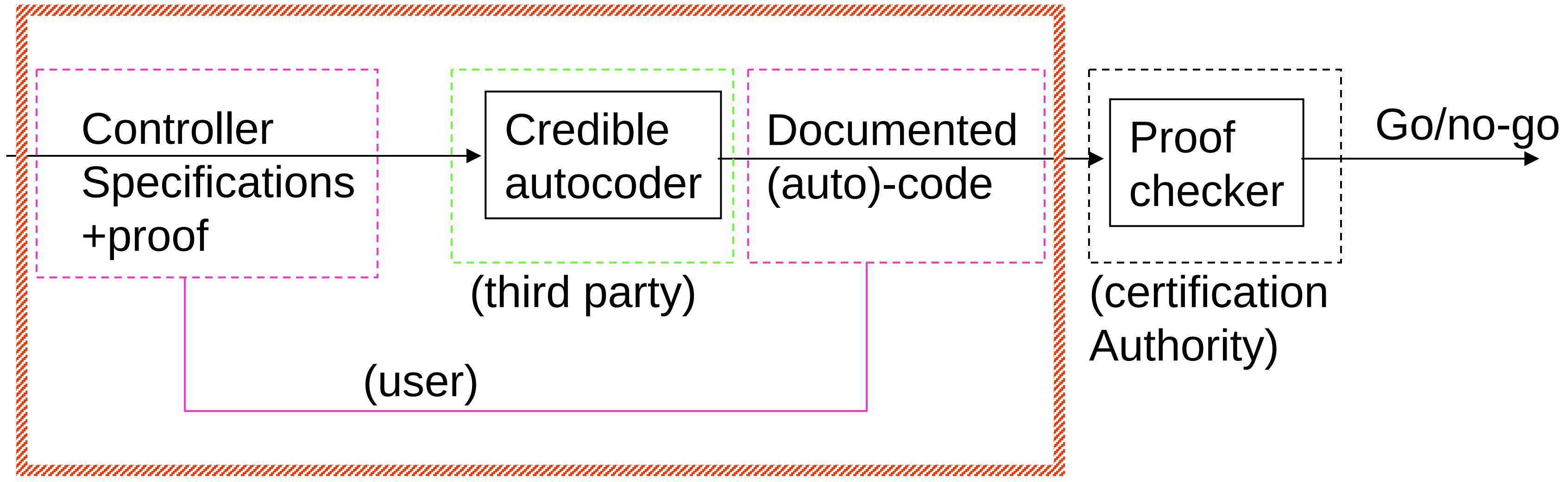
Commented code

```
$\{\text{true}\}$
A = [0.4990, -0.0500; 0.0100, 1.0000];
$\{\text{true}\}$
C = [-564.48, 0];
$\{\text{true}\}$
B = [1;0];D=1280
$\{\text{true}\}$
x = zeros(2,1);
$\{x \in \mathcal{E}_P\}$
while 1
$\{x \in \mathcal{E}_P\}$
y = fscanf(stdin,"%f")
$\{x \in \mathcal{E}_P\}$
y = max(min(y,1),-1);
$\{x \in \mathcal{E}_P,\ y^2 \leq 1\}$
u = C*x+D*y;
$\{x \in \mathcal{E}_P,\ u^2 \leq 2(CP^{-1}C^T + D^2),\ y^2 \leq 1\}$
fprintf(stdout,"%f\n",u)
$\{x \in \mathcal{E}_P,\ y^2 \leq 1,\ (Ax+By)^T P(Ax+By) - 0.01x^T Px - 0.99y^2 \leq 0\}$
$skip$
$\{Ax + By \in \mathcal{E}_P,\ y^2 \leq 1\}$
 x = A*x + B*y;
$\{x \in \mathcal{E}_P\}$
 end
```

Adding the controlled plant as part of the controller's semantics



Front End: Formal comment writing



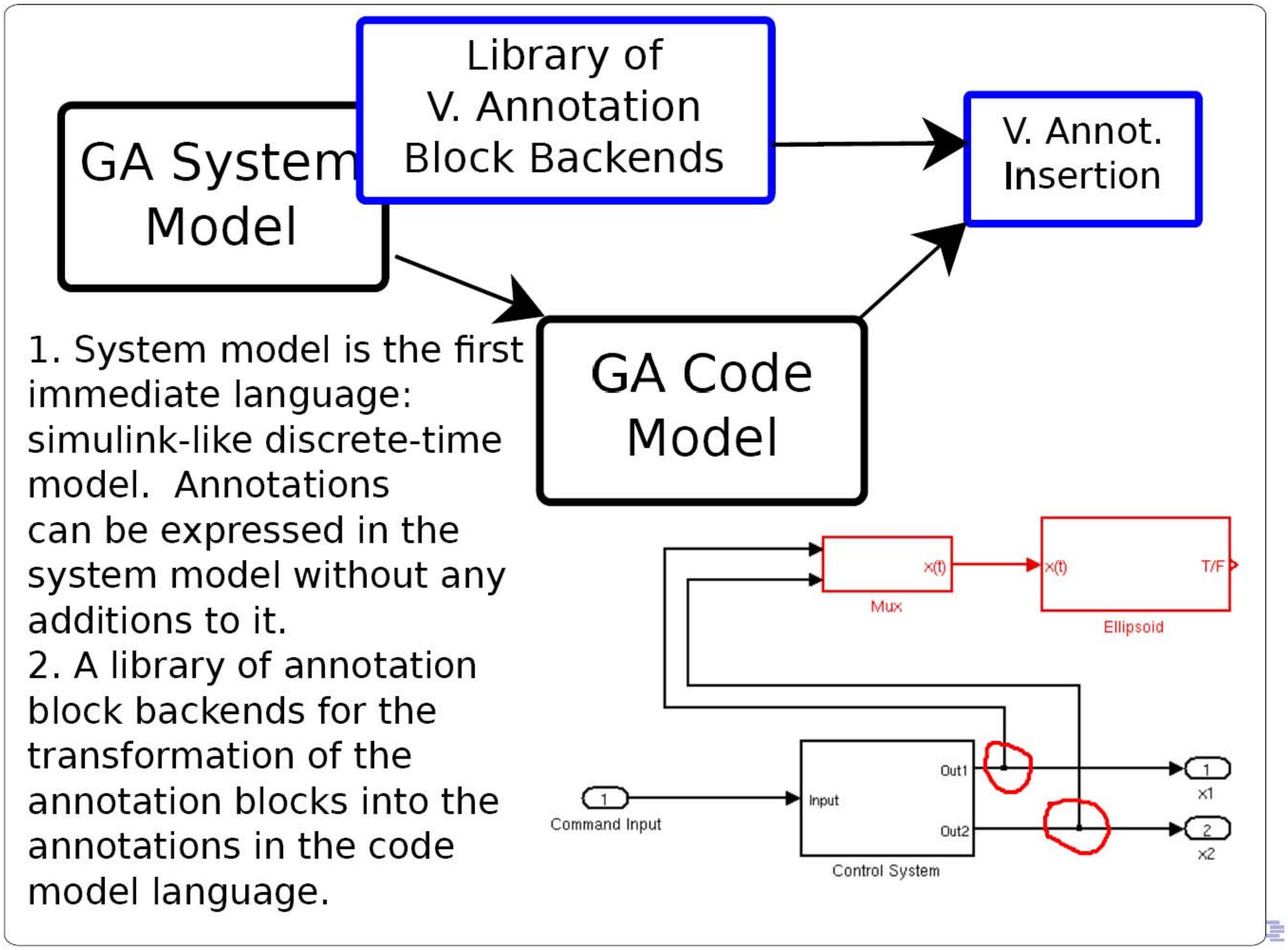
- ANSI/ISO C Specification Language (ACSL) can be used to formally comment C programs and can be handled by Frama-C.
- Start from Simulink
- End with commented C code

ANNOTATION LANGUAGE

- On the Simulink Side
  - Must be able to write system semantics and proofs supporting semantics.
- On the C side
  - Same requirements of expressivity, but annotations must be readable by certification software.
  - We express everything in ACSL.

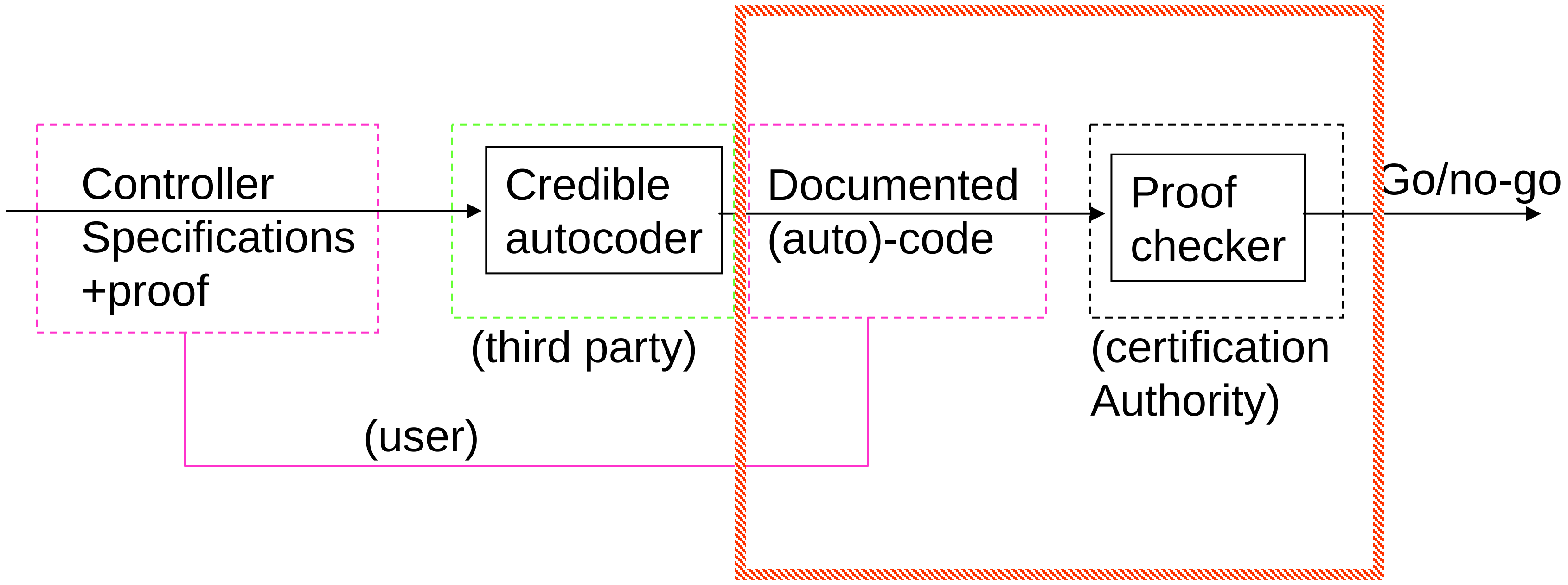
A prototype front-end built on Gene-Auto

- Thank you Marc Pantel, Arnaud Dieumegard, Andres Toom



1. System model is the first immediate language: simulink-like discrete-time model. Annotations can be expressed in the system model without any additions to it.
2. A library of annotation block backends for the transformation of the annotation blocks into the annotations in the code model language.

Back End: Verification of Code Semantics



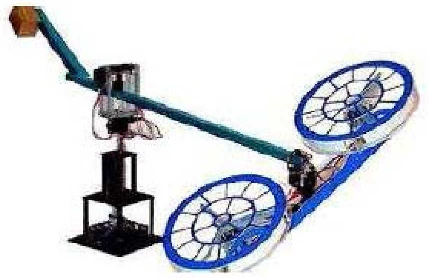
FRAMA-C

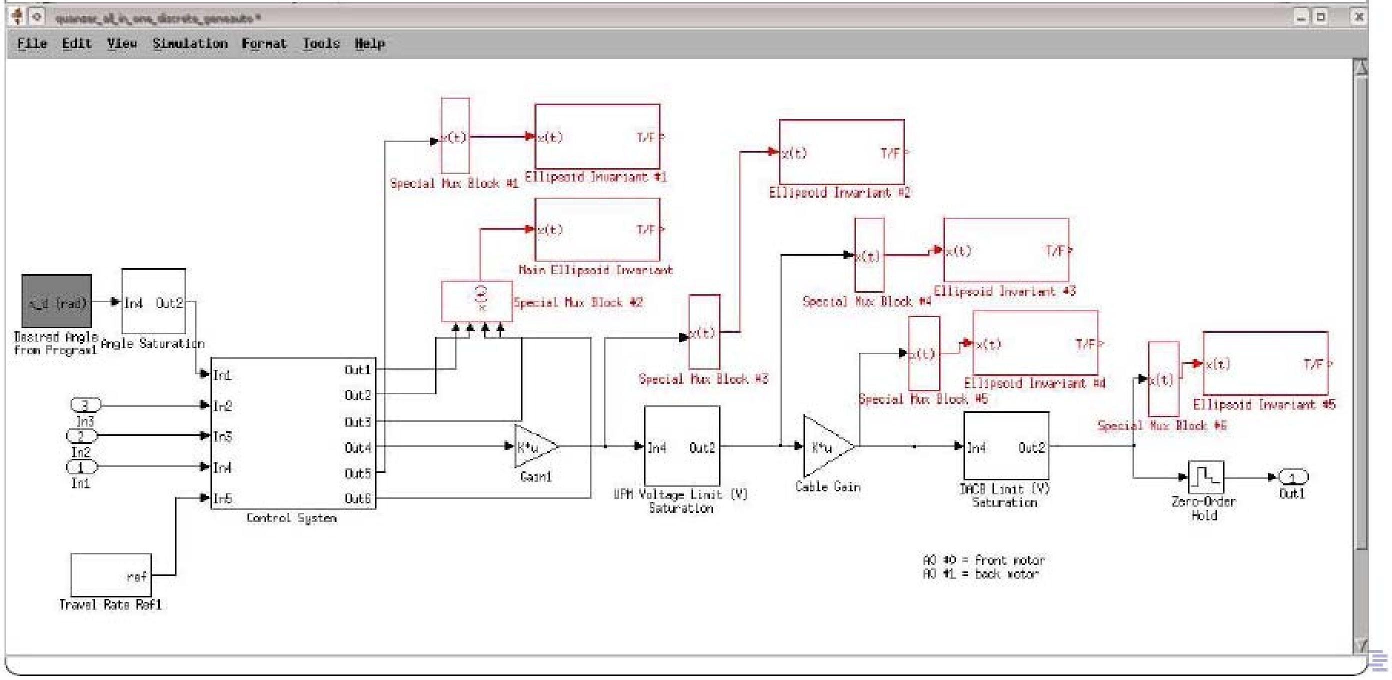
- Developed by CEA-LIST and INRIA
- Hoare Style annotation language.
- Can interface with manual and automated proving software (e.g., PVS).
- Has the required expressivity.

INTERFACING WITH VERIFICATION TOOLS

- We use Frama-C because it can generate verification conditions for various pieces of software
- The interface with PVS allows us to use the work done at the National Institute of Aerospace (Heber Herencia) and SRI (Sam Owre) on verification of linear algebraic systems. (NFM 2012)

A physical example: 3 DOF helicopter



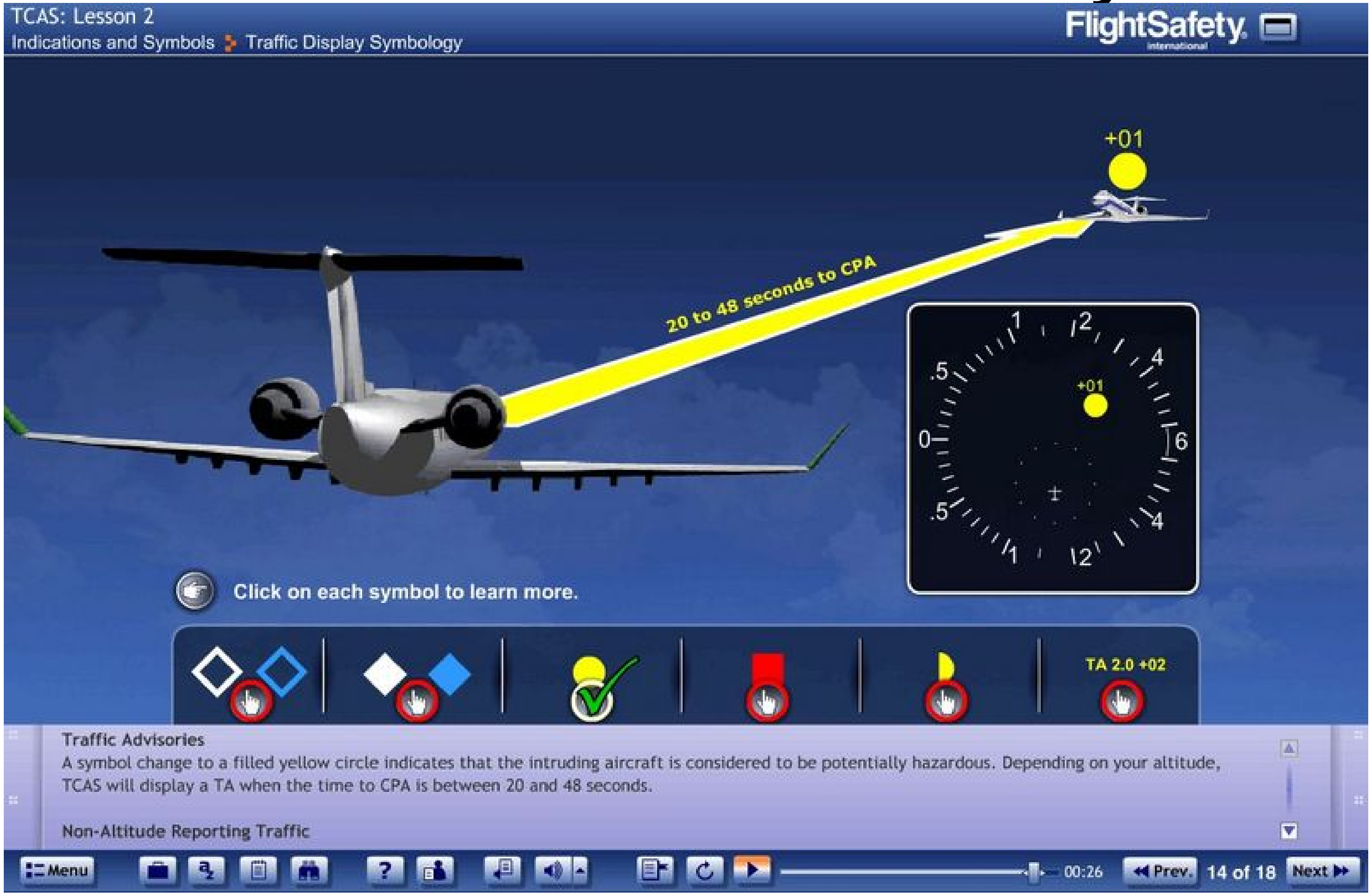


And it still works!!!

Next step: F-18 replica from Rockwell-Collins

http://www.youtube.com/watch?v=QJkIONTzbNM

Application to Collision avoidance TCAS / last resort safety net

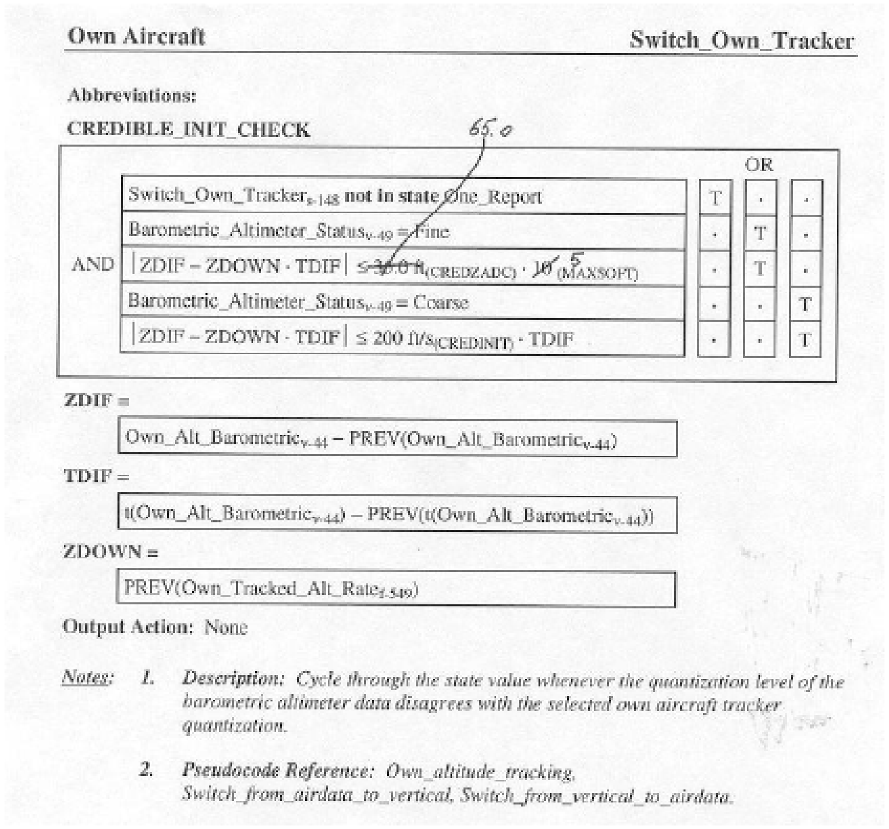


Traffic Advisories
A symbol change to a filled yellow circle indicates that the intruding aircraft is considered to be potentially hazardous. Depending on your altitude, TCAS will display a TA when the time to CPA is between 20 and 48 seconds.

Non-Altitude Reporting Traffic

Vehicle guidance and collision avoidance

- Current TCAS designed as computer pseudo-code and specifications



Own Aircraft **Switch_Own_Tracker**

Abbreviations:

CREDIBLE_INIT_CHECK

		OR		
AND	Switch_Own_Tracker$_{s\text{-}148}$ **not in state** One_Report	T	·	·
	Barometric_Altimeter_Status$_{v\text{-}49}$ = Fine	·	T	·
	$\lvert ZDIF - ZDOWN \cdot TDIF \rvert \le$ ~~30.0~~ 65.0 ft$_{(CREDZADC)}$ · ~~10~~ 5$_{(MAXSOFT)}$	·	T	·
	Barometric_Altimeter_Status$_{v\text{-}49}$ = Coarse	·	·	T
	$\lvert ZDIF - ZDOWN \cdot TDIF \rvert \le$ 200 ft/s$_{(CREDINIT)}$ · TDIF	·	·	T

ZDIF =

Own_Alt_Barometric$_{v\text{-}44}$ – PREV(Own_Alt_Barometric$_{v\text{-}44}$)

TDIF =

t(Own_Alt_Barometric$_{v\text{-}44}$) – PREV(t(Own_Alt_Barometric$_{v\text{-}44}$))

ZDOWN =

PREV(Own_Tracked_Alt_Rate$_{f\text{-}549}$)

Output Action: None

Notes: 1. ***Description:*** *Cycle through the state value whenever the quantization level of the barometric altimeter data disagrees with the selected own aircraft tracker quantization.*

2. ***Pseudocode Reference:*** *Own_altitude_tracking, Switch_from_airdata_to_vertical, Switch_from_vertical_to_airdata.*

Very hard to formally <u>prove</u> anything about TCAS.

Where are the invariants? Good luck with that. A nice challenge for static analyzers.

ACAS-X: A new development

- An airborne, embedded collision avoidance system like TCAS. Same functionality.
- Developed by Lincoln Laboratory, Lexington, Massachusetts, and MIT.
- Reportedly improvement over TCAS.
- Development encouraged by Federal Aviation Administration, and discussed by FAA/EASA/DGAC-DTI groups.

New Development: ACAS-X

- Designed according to sound theoretical, model-based principle of *Dynamic Programming*:

$$J^*(x) = \min_u \left(c(x) + J^*(x^+)\right)$$

$$x^+ = f(x,u)$$

Think of J as total probability of collision during encounter, $c(x)$ as probability of collision at state x during small instant. Need other terms to prevent aircraft from making, e.g. Split S maneuver.

What’s a Split-S?

2002: First autonomous aerobatic split-S (Gavrilets, Feron, Mettler)

ACAS-X certification

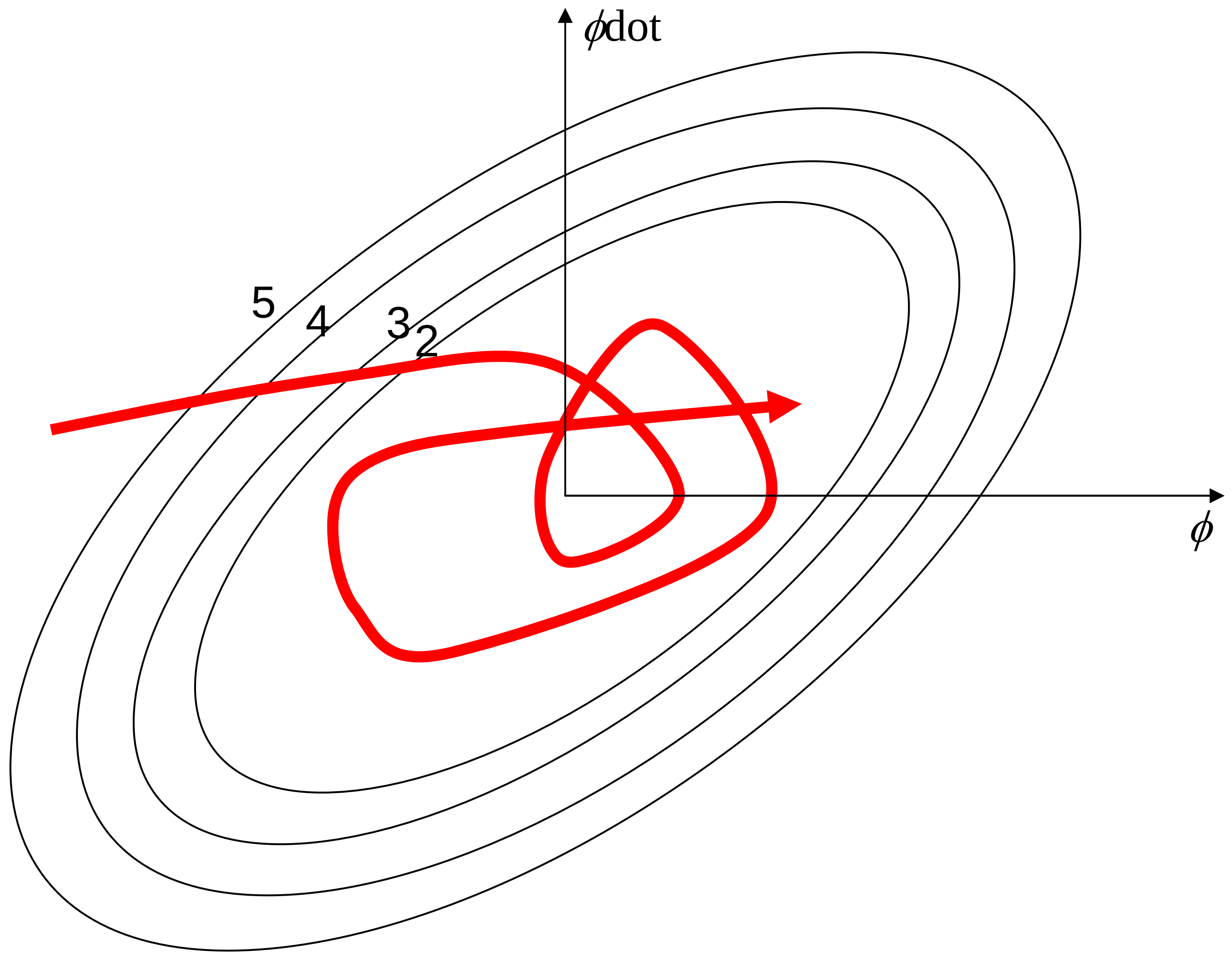
From Kochenderfer, 2010

“In particular, since this is a new approach to TCAS logic development, the certifiability of the resulting logic is of particular concern. If this new approach is to be used simply as an aid to engineers who are developing or revising collision avoidance pseudocode, then there would be little impact on the certification process. However, if the logic produced by dynamic programming or some other automated process is to be used directly in a future version of TCAS, then the certification process may be somewhat different. The core of the certification process will be the same, involving rigorous simulation studies and flight tests to prove safety and demonstrate operational acceptability. However, the vetting of the logic itself will involve more than just studying the logic that will be deployed on the system. Depending on the representation of the logic, it may not be directly comprehensible by an engineer. Therefore, confidence would need to be established in the safety community that the methods used to generate the logic are sound.”

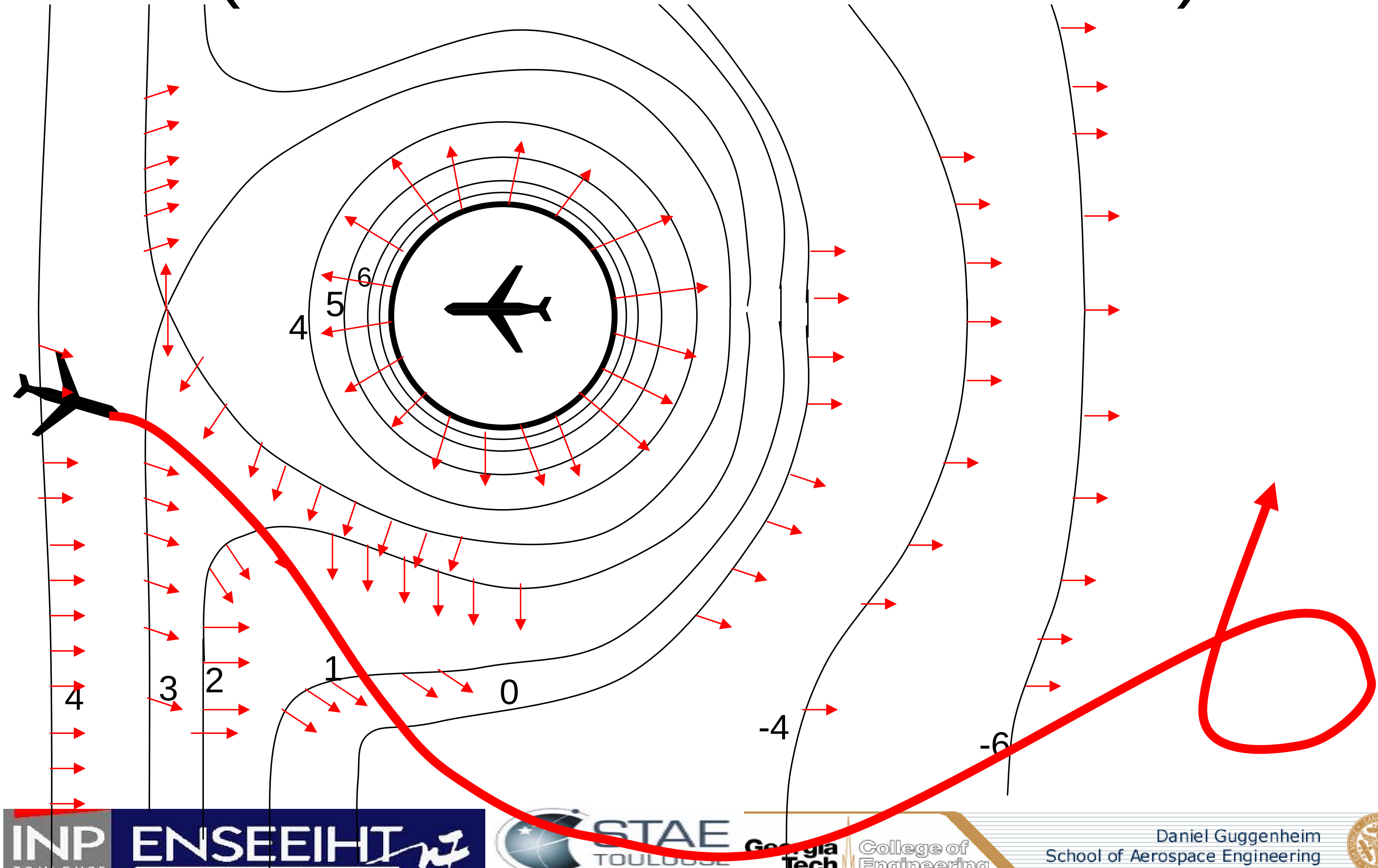
Solution in part via close designer/software analysis cooperation

- Under “optimal” decision policy, J^* , the optimal cost, *decays along trajectories.*
- ie J^* acts a bit like a…. Lyapunov function.
- So plenty of opportunities to extract essential ACAS-X software properties at design phase.

Lyapunov functions yield software invariants…

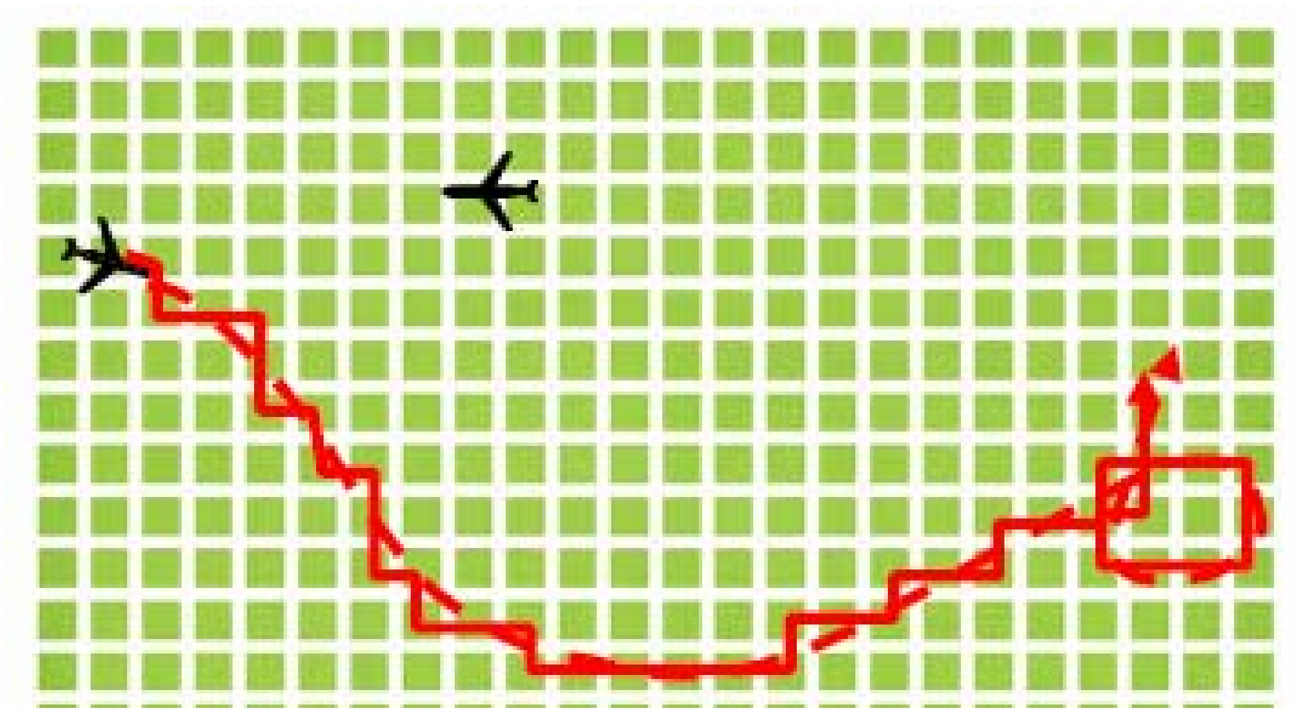


So do optimal cost functions…
(Note: This is NOT ACAS-X)



Same challenges as for inner-loop control functions

- Lincoln Lab's ACAS-X is designed via discretized state-space.
- Specification-level models used to design system are not identical to reality



Conclusion

- It is possible to generate safety-critical control code from specifications, all-equipped with semantics and proofs.
- Code-level analyses are possible, and much easier than analyses from code alone.

Acknowledgements

- Army Research Office
- Dutton/Ducoffe professorship at Georgia Tech
- Fondation STAE Toulouse
- Institut National Polytechnique de Toulouse
- National Science Foundation
- NASA
- *Fernando Alegre, Arnaud Dieumegard, Alwyn Goodloe, Heber Herencia, Pierre-Loic Garoche, Romain Jobredeaux, Sam Owre, Marc Pantel, Pierre Roux, Andres Toom, Arnaud Venet, Tim Wang.*

INP TOULOUSE ENSEEIHT n7 STAE TOULOUSE Georgia Tech College of Engineering Daniel Guggenheim School of Aerospace Engineering