Caltech

Lecture 10 / 11
Safety-Critical Systems / Course Summary

Richard M. Murray Caltech

Tichakorn Wongpiromsarn UT Austin/Iowa State

EECI-IGSC, 13 Mar 2020

Outline

- Introduction to safety-critical systems (aerospace focus)
- Multi-layer control system design (review of key concepts from the course)
- Thoughts and challenges for the future of self-driving cars

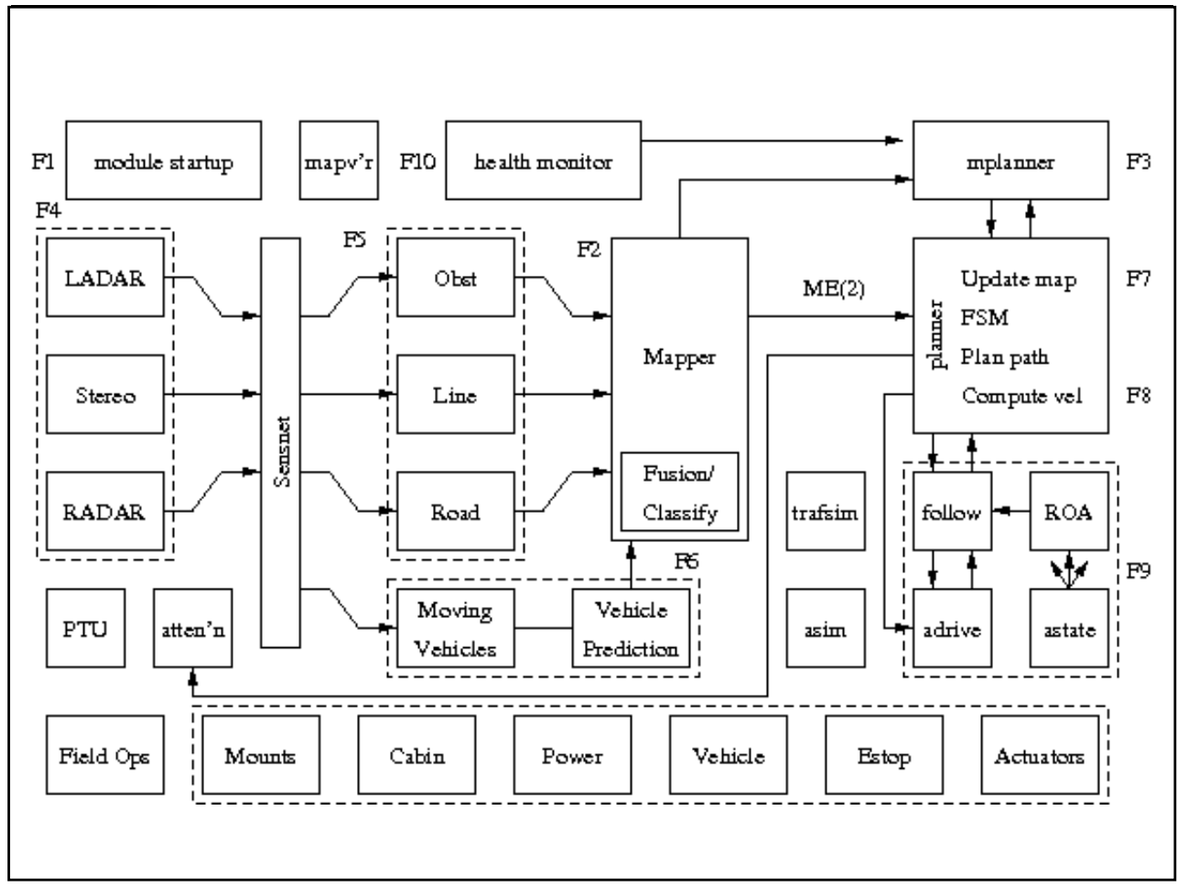
Motivating Example: Alice (2004-2007)

Alice

- 300+ miles of fully autonomous driving
- 8 cameras, 8 LADAR, 2 RADAR
- 12 Core 2 Duo CPUs + Quad Core
- 3 Gb/s data network
- ~75 person team over 18 months (x 2)

Software

- 25 programs with ~200 exec threads
- 237,467 lines of executable code



How should we design systems of this complexity?
How do we make sure they function as desired?

Current Landscape: Self-Driving Cars

SAE Levels of Automation:

0	1	2	3	4	5
No Automation	Driver Assistance	Partial Automation	Conditional Automation	High Automation	Full Automation
Zero autonomy; the driver performs all driving tasks.	Vehicle is controlled by the driver, but some driving assist features may be included in the vehicle desig	Vehicle has combined automated functions, like acceleration and steering, but the driver must remain engaged with the ing task and moni r the environ ent at all times.	Driver is a necessity, but is not required to monitor the environment. The driver must be ready take ntrol of the e at all times notice.	The vehicle is capable of performing all driving functions under certain conditions. The driver may the tion to contro he v icle.	The vehicle is capable of performing all driving functions under all conditions. The driver may have t opt to control ehicle.

Safety Critical Autonomous Systems

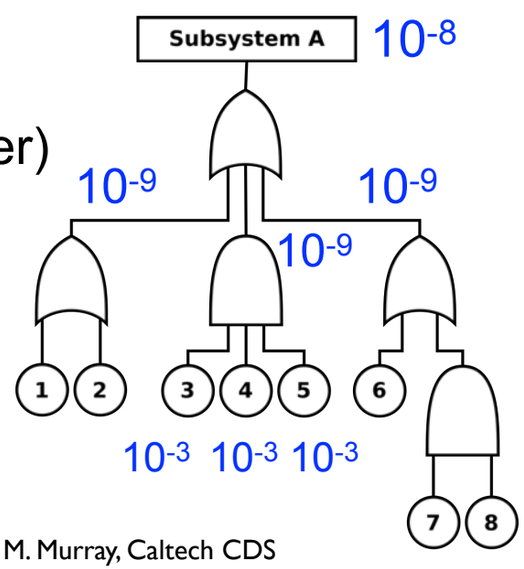
Question: How safe do autonomous vehicles need to be?

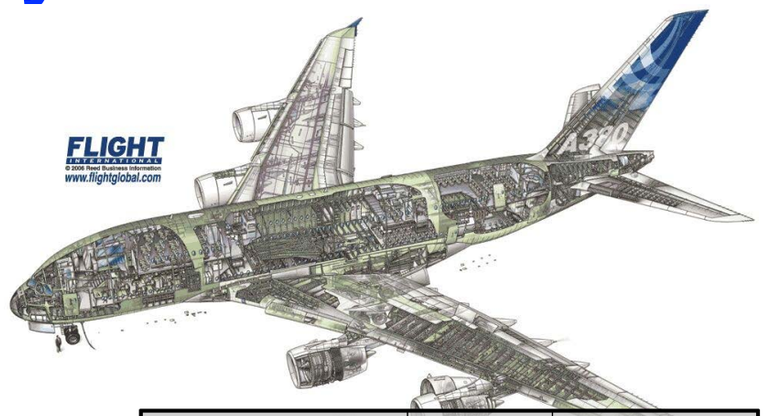
- As safe as human-driven cars (7 deaths every 10^9 miles)
- As safe as buses and trains (0.1-0.4 deaths every 10^9 miles)
- As safe as airplanes (0.07 deaths every 10^9 miles)

I. Savage, "Comparing the fatality risks in United States transportation across modes and over time", *Research in Transportation Economics*, 43:9-22, 2013.

How this is done in the aerospace industry?

- Strong certification requirements/process (DO-178C)
  - Fault tree analysis (1e-9 failure rates)
  - Model-based design + SIL, HIL testing
  - Fleet-wide analysis (⇒ rare cases matter)
- Very structured operating environments
- Well-trained personnel (pilots, FAs)
- Expensive vehicles (~$1M/passenger)





Hazard Class	SW Level	Failure/ Flight Hr
Catasophic	A	10^{-9}

DO-178C / ED-12C	
Software Considerations in Airborne Systems and Equipment Certification	
Latest Revision	01/05/2012
Prepared by	RTCA SC-205 EUROCAE WG-12

Formal methods supplement	Model-based development supplement	Object-oriented technologies supplement

What Goes "Wrong": ZA002, Nov 2010

Official Word from Boeing: ZA002 787 Dreamliner fire and smoke details

By David Parker Brown, on November 10th, 2010 at 3:46 pm

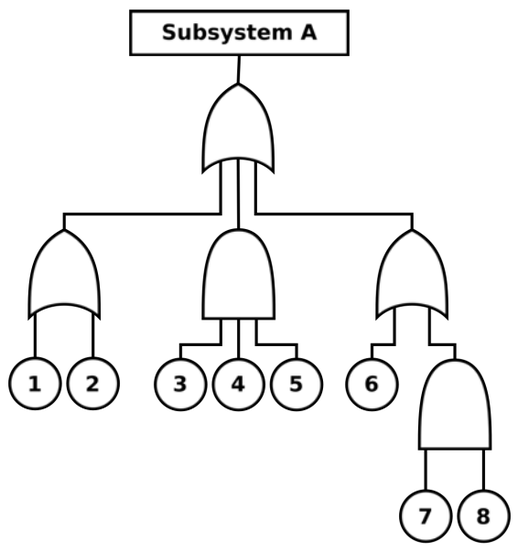
Boeing 787 Dreamliner ZA002 at Paine Field on January 27, 2010 before its first flight.

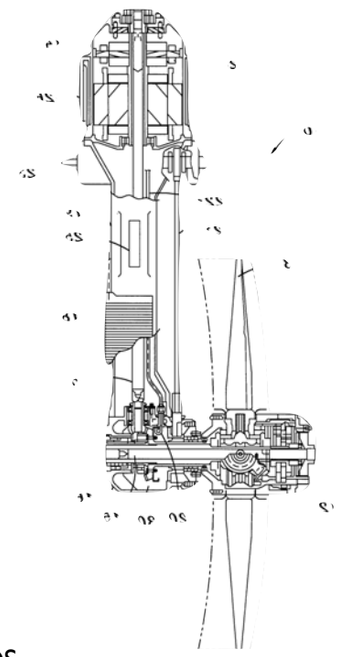
For the last day there are been bits and pieces of information coming from Boeing, inside sources and different media outlets on ZA002's sudden landing due to reported smoke in the cabin. Boeing has just released an official statement putting some of the rumors to rest and explaining what they know of ZA002's recent emergency landing in Laredo, TX.

Boeing confirms that ZA002 did lose primary electrical power that was related to an on board electrical fire. Due to the loss, the Ram Air Turbine (RAT), which provides back up power (photo of RAT from ZA003) was deployed and allowed the flight crew to land safely. The pilots had complete control of ZA002 during the entire incident.

Loss of primary electrical power => cockpit goes "dark"

Ram Air Turbine (RAT) deployed and allows safe landing



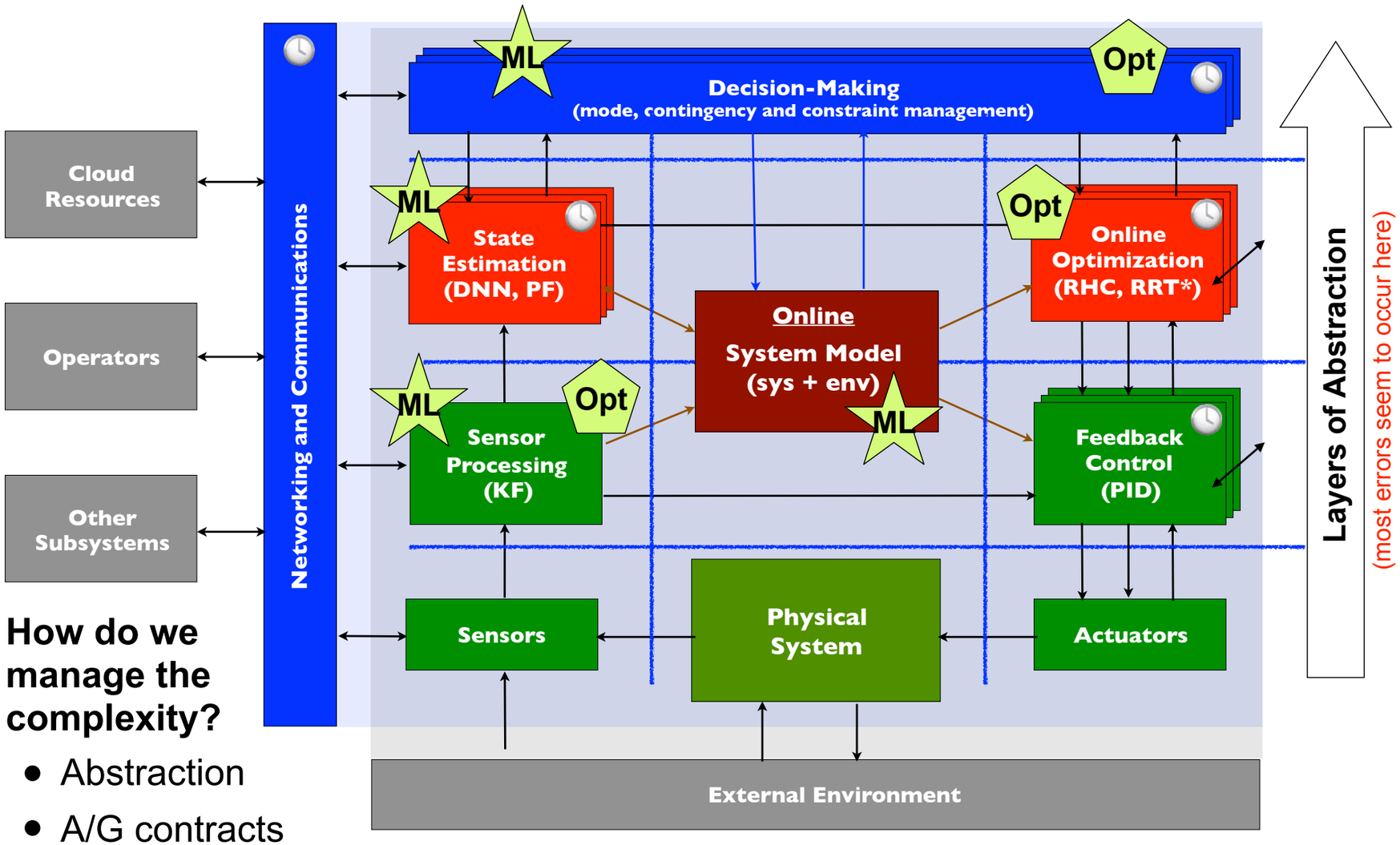


RAT stats

- ~100K flights/day globally => 35M flights/year
- ~6 documented RAT deployments in the last 20 years
- Assume 10X that amount => 3 per year => 1 in 10M flights (!)

Key point: aerospace engineers worry about the worst case

Design of Modern (Networked) Control Systems



How do we manage the complexity?

- Abstraction
- A/G contracts
- Formal methods for verification/synthesis + model- & data-driven sims/testing

Examples

- Aerospace systems
- Autonomous vehicles
- Factory automation/ process control
- Smart buildings, grid, transportation

Challenges

- How do we define the layers/interfaces (vertical contracts)
- How do we scale to *many* devices (horizontal contracts)
- Safety, robustness, security, privacy

Thoughts on ML and Control ("Easy" Problems)

ML Challenges

Failure rates are too high, w/ poor metrics

- 1 hour = 10K frames => 1B hours = …
- Classification error is not that useful

Data requirements are unknown (but large)

- Size of error vs amount of training data?
- How do we catch corner cases?

Focus on ML output vs system behavior

- Classification error is not what we actually care about; do we hit anything?

Early adoption in safety-critical settings

- Use of ML for decision-making is not ready
- Advice: ML for *performance*, optimization and control for safety and robustness

Controls Perspective

Stability margins with uncertainty balls

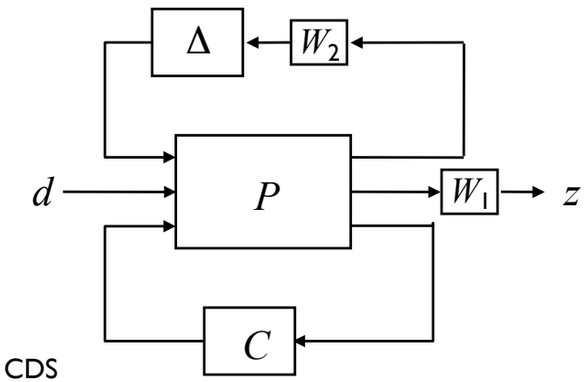
- Bounds on disturbances, uncertainty
- Model/analyze temporal response

Model-based, parametric representations

- Constrain model class (TFs, ARMAX, etc)
- Reason over worst case behavior

Input/output focus

- Focus on outputs that matter for the task and impact of uncertainty on those outputs



$$\|z\|_2 \leq \gamma \|d\|_2$$

for all

$$\|\Delta\| \leq 1$$

Thoughts on ML and Control (Hard Problems)

Autonomous Vehicles for Urban Mobility
Emilio Frazzoli, ETH Zurich & Aptiv

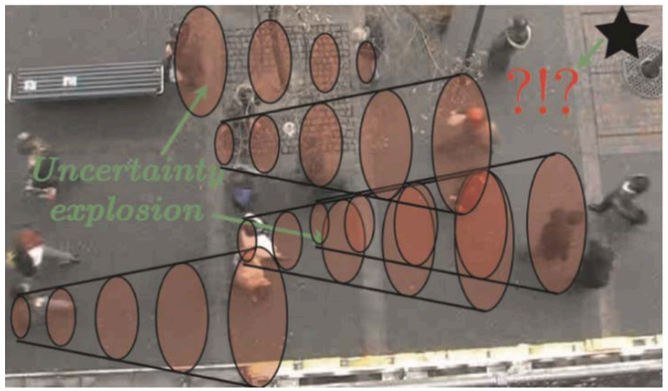
… [As] we move past the peak of the hype cycle, the industry is bracing for a development timeline that is much longer than many early predictions.

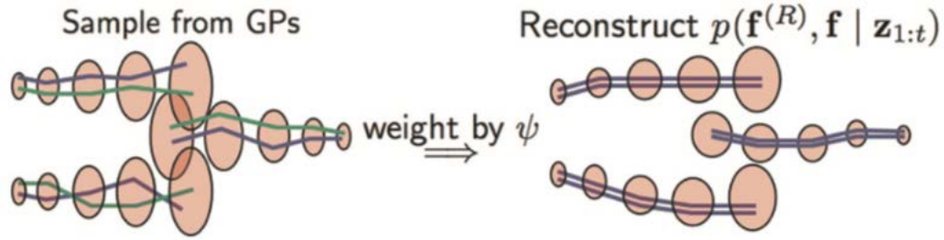
… fundamental issues that remain essentially unresolved, and will require a concerted effort by industry, academia, and regulatory bodies to address.

These issues essentially go beyond the (very hard, but in a sense "standard" and well studied) problems of control, perception, etc. and revolve around making sound decisions on precisely how we want these vehicles to behave, both at the individual, single-car level, and at the fleet level. In other words, how we want these vehicles to behave when interacting with pedestrians, cyclists, or other cars, and what effect we want them to have on urban mobility, including, e.g., their impact on the urban environment, public transit, and society.

Trautman, Ma, M and Krause
IJRR, 2014

Some Prior Work: Navigation in Crowds



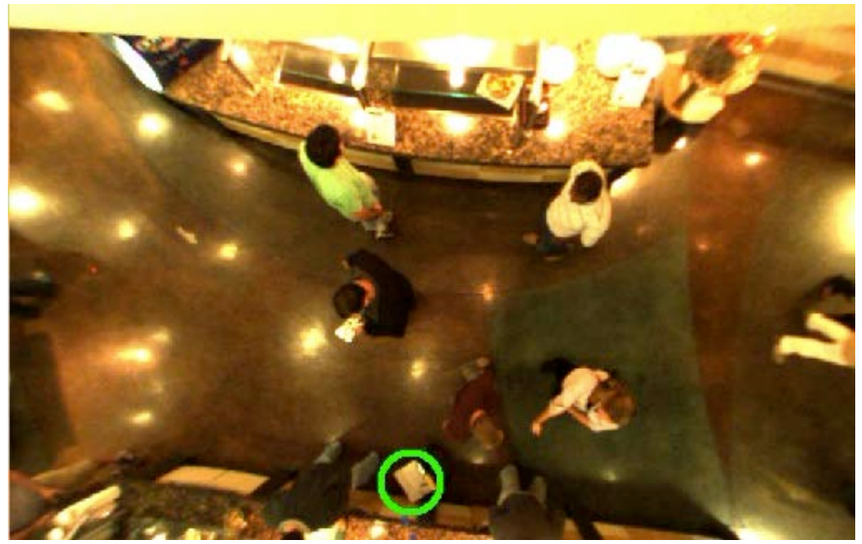


$$p(\mathbf{f}^{(R)}, \mathbf{f} | \mathbf{z}_{1:t}) = \frac{1}{Z} \psi(\mathbf{f}^{(R)}, \mathbf{f}) \prod_{i=R}^{n} p(\mathbf{f}^{(i)} | \mathbf{z}_{1:t}^{(i)})$$

Key results

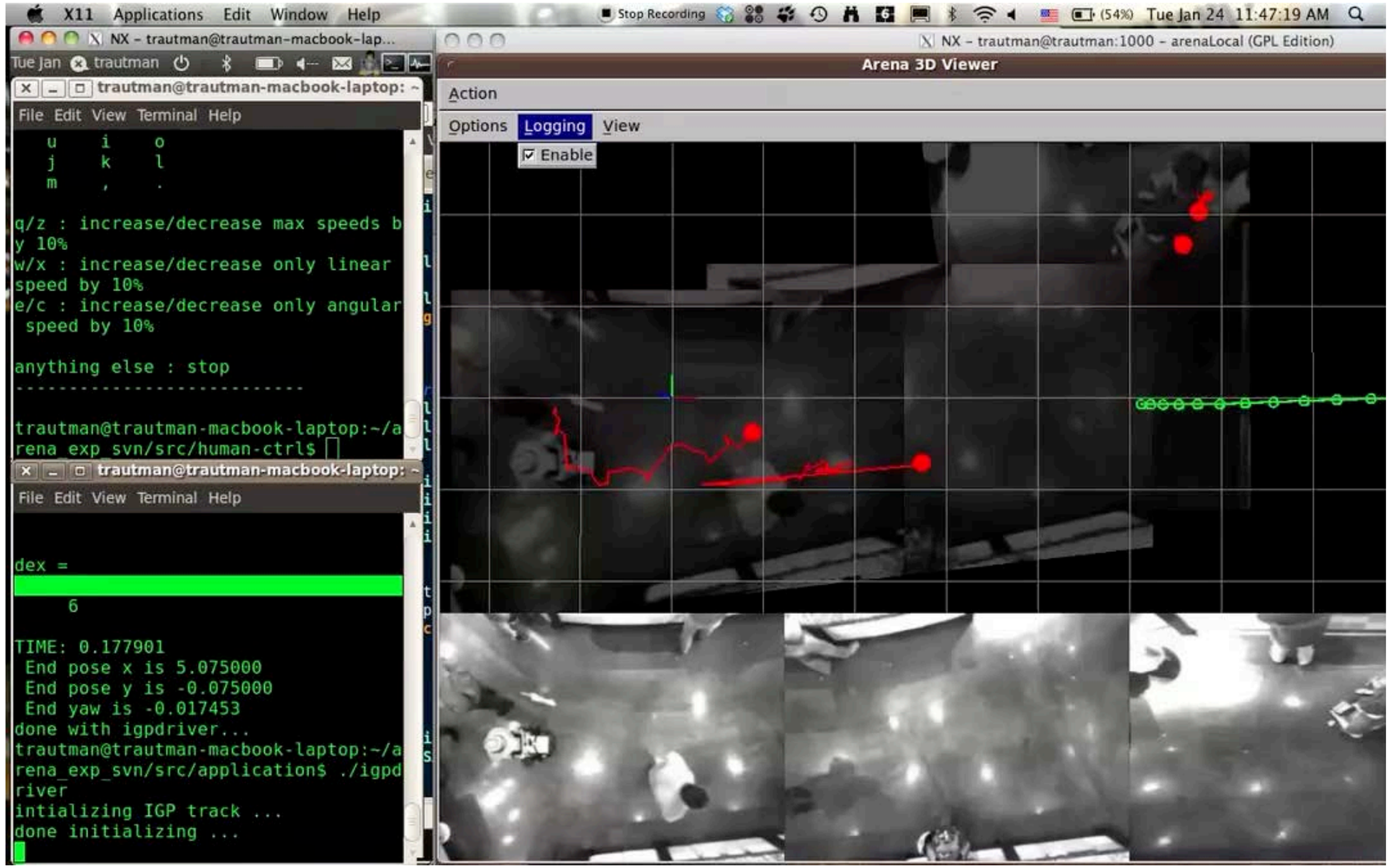
- Address "freezing robot problem": planner decides that all forward paths are unsafe and freezes in place
- Approach: *interacting Gaussian processes*
  - captures cooperative collision avoidance
  - allows goal-driven nature of human decision making
- Validation in Caltech staff cafeteria
  - Performs comparably with human teleoperators
  - non-cooperative planner exhibits unsafe behavior
  - reactive planner fails for crowd densities > 0.55 ppl/m^2

$$\psi(\mathbf{f}^{(R)}, \mathbf{f}) = \prod_{i=R}^{n} \prod_{j=i+1}^{n} \prod_{\tau=t}^{T} (1 - \alpha \exp(-\frac{1}{2h^2} |\mathbf{f}^{(i)}(\tau) - \mathbf{f}^{(j)}(\tau)|))$$

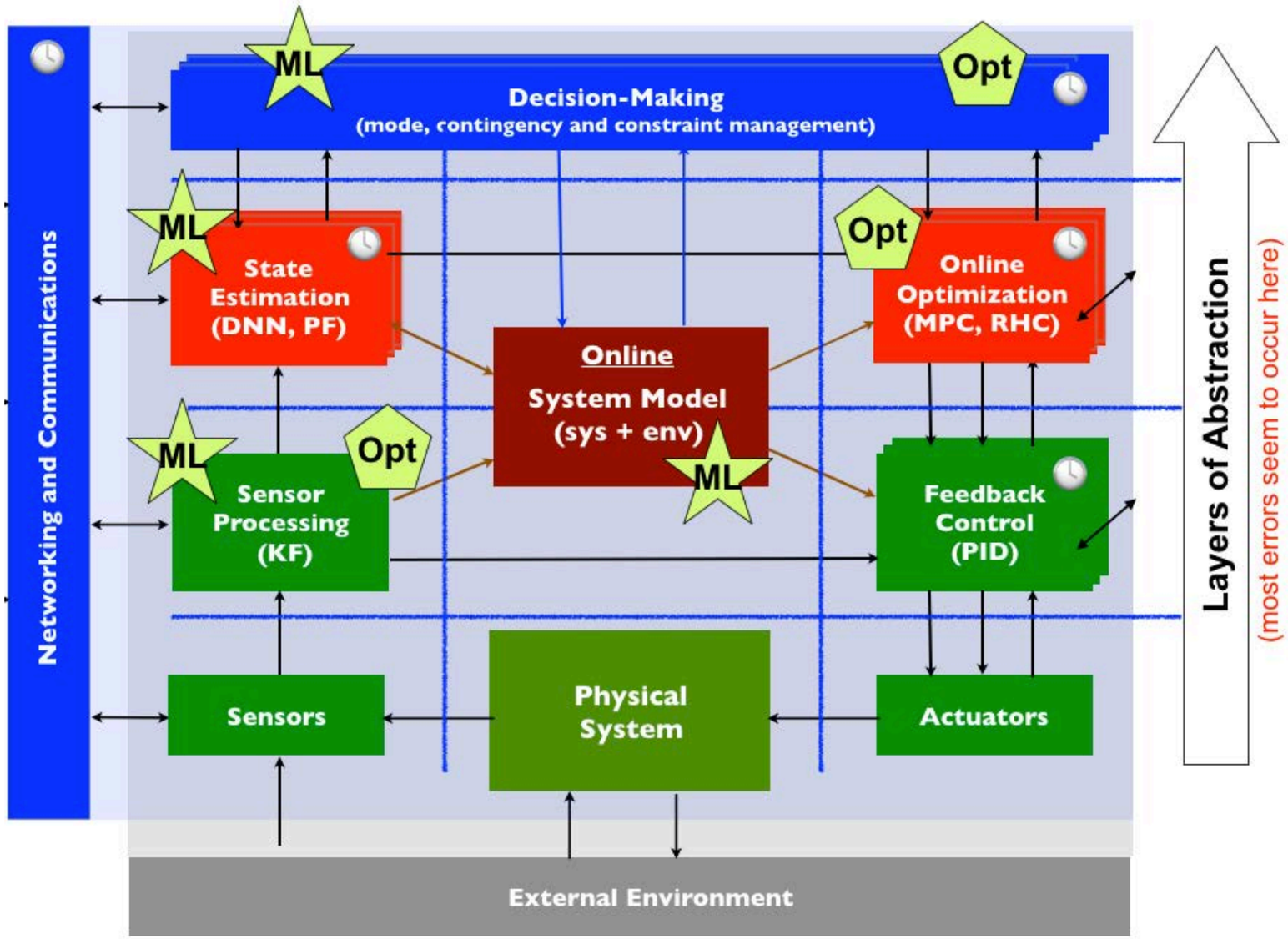


Trautman, Ma, M and Krause
IJRR, 2014

Some Prior Work: Navigation in Crowds



RMM Assessment: Wait for Others to Figure out ML…



Assume/guarantee contracts

- Assume: properties of other components in the system
- Guarantee: properties that will hold for my component

$$A_i \Rightarrow G_i$$

$G_2 \wedge G_3 \Rightarrow A_1$, $G_1 \wedge G_3 \Rightarrow A_2$, …

- Contracts can be horizontal (within a layer) or vertical (between two layers)

Integrating ML (eventually)

- Wait for smart people to create ML w/ A/G contracts
- Think about how to best integrate these into the larger NCS architecture

Machine Learning in Safety-Critical Systems

Hazard Class	SW Level	Failure/ Flight Hr
Catasophic	A	10^{-9}
Hazardous	B	10^{-7}
Major	C	10^{-5}
Minor	D	—
No Effect	E	—

0.07 deaths every 10^9 miles ⟵ ?? ⟶ 7 deaths every 10^9 miles

35K/year (US)

Claim: ML can solve problems that we can't solve otherwise

Q: How do we move ML into safety-critical applications?

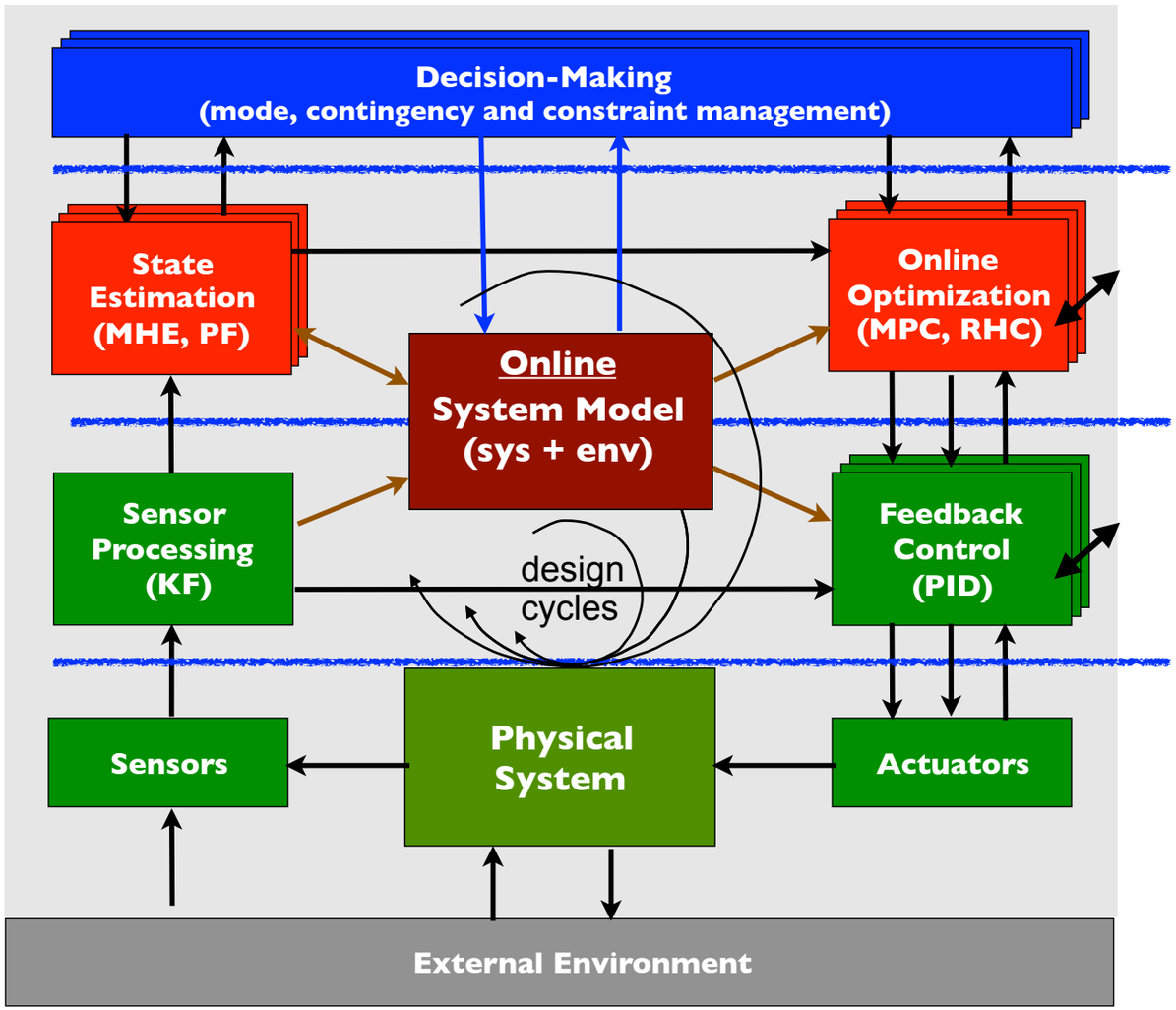
- Certification methodology for ML-based components
- Error rates (of decisions) measured in 1 per billions of hrs/miles
- Robust operation across wide range of conditions

DO-178C / ED-12C	
Software Considerations in Airborne Systems and Equipment Certification	
Latest Revision	01/05/2012
Prepared by	RTCA SC-205 EUROCAE WG-12

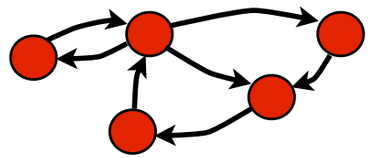
Formal methods supplement	Model-based development supplement	Object-oriented technologies supplement

Layered Approaches to Design

Multi-layer Networked Control System



Model



$$\dot{x} = f_\alpha(x, u)$$
$$g_\alpha(x, u, z) \leq 0$$

$$y = P_{yu}(s)\, u + P_{yd}(s)\, d$$
$$\|W(s)d(s)\| \leq 1$$

$$\dot{x}^i = f_\alpha(x^i, u^i, d^i)$$
$$x \in \mathcal{X}, u \in \mathcal{U}, d \in \mathcal{D}$$

Specs

$$(\phi_{\text{init}} \wedge \Box\phi_{\text{env}}) \implies (\Box\phi_{\text{safe}} \wedge \Box\Diamond_{\leq T}\phi_{\text{live}})$$

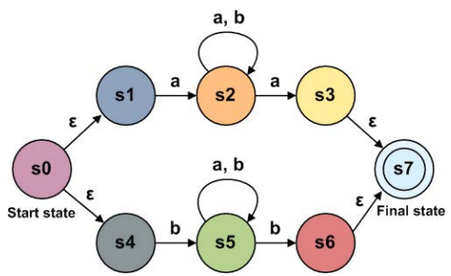
$$\min J = \int_0^T L_\alpha(x, u)\, dt + V(x(T))$$

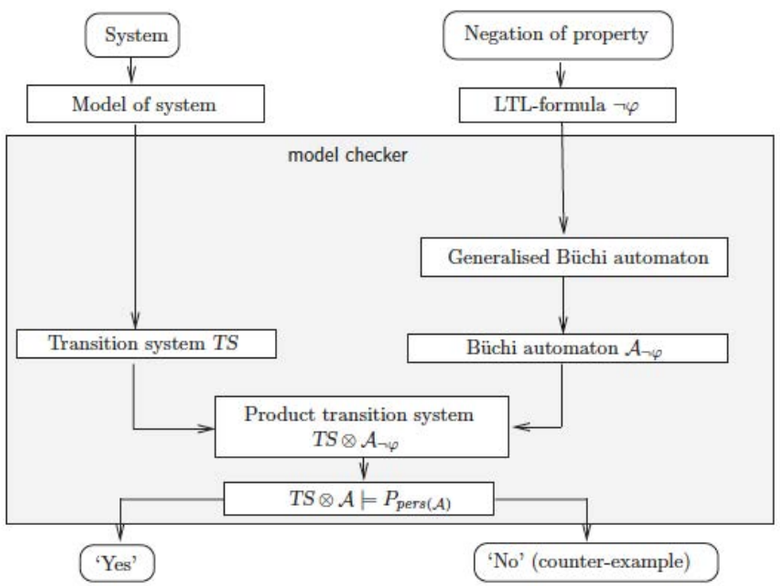
$$\|W_1 S + W_2 T\|_\infty < \gamma$$

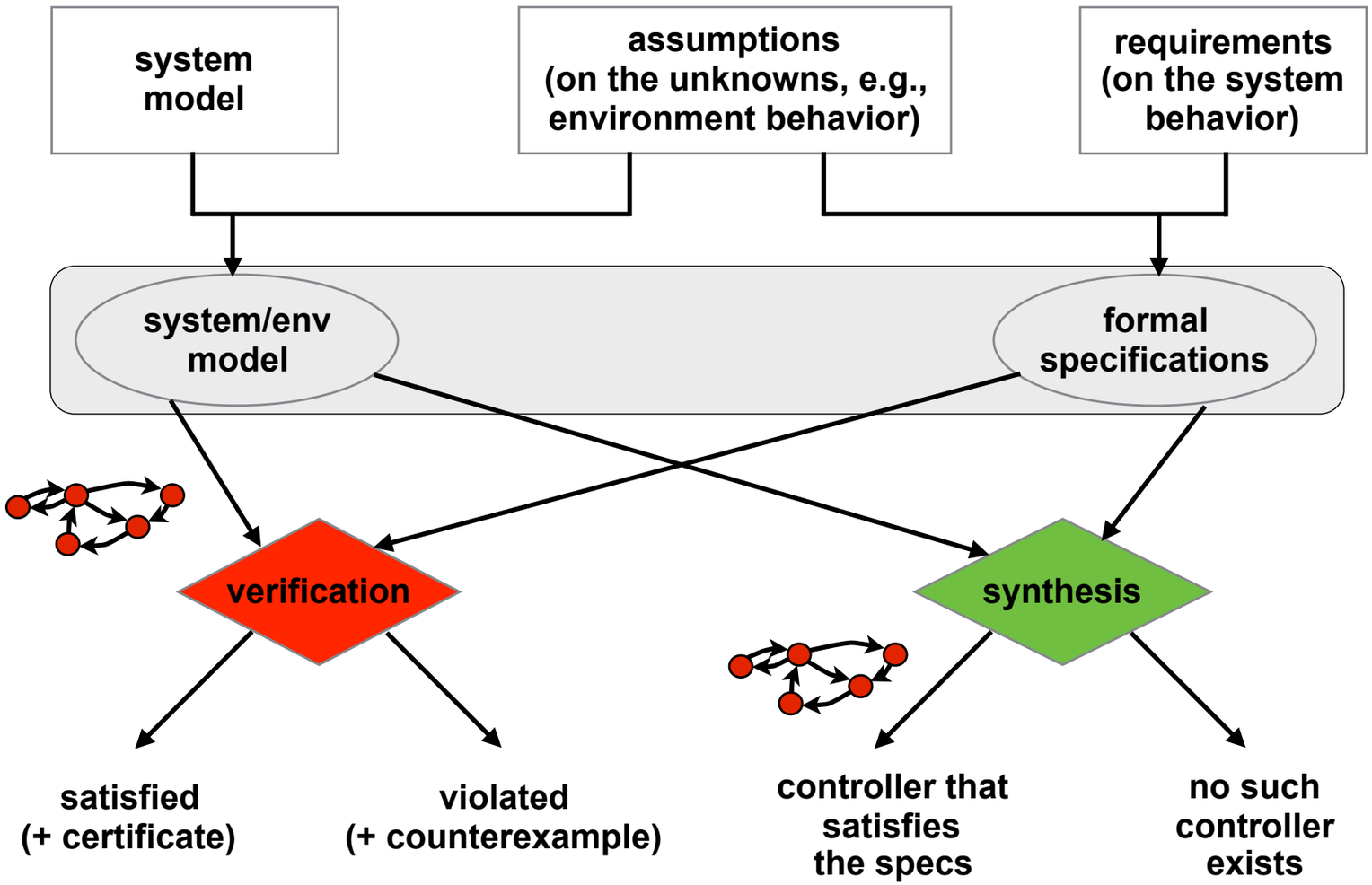
Operating Envelope
Energy Efficiency
Actuator Authority

Formal Methods for System Design

$$\Box\{(\tilde{c}=1\wedge c=0\wedge(x_C<T_{c_{min}}))\rightarrow (\bigcirc c=0\wedge\bigcirc x_C=x_C+\delta)\},$$
$$\Box\{(\tilde{c}=1\wedge c=0\wedge(x_C\geq T_{c_{min}}))\rightarrow (\bigcirc c=1\vee\bigcirc x_C=x_C+\delta)\},$$
$$\Box(x_C\leq T_{c_{max}}).$$



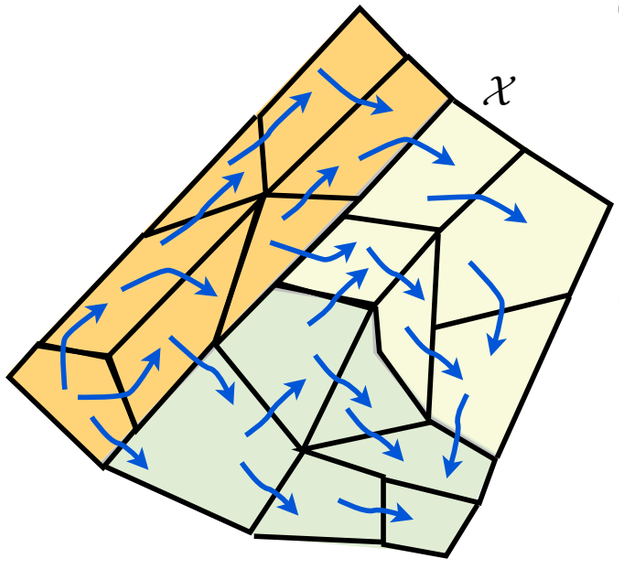




Ready to be applied now
(SoS, SPIN, TLC, nuSMV, PRISM, SMT)

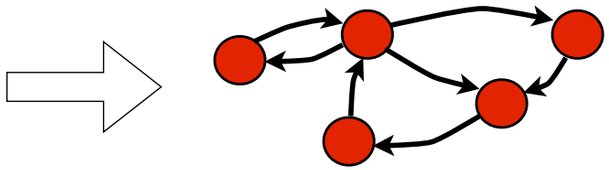
Next generation tools (in progress)

Discrete Abstractions for (Hybrid) Dynamical Systems



Continuous states → discrete abstractions

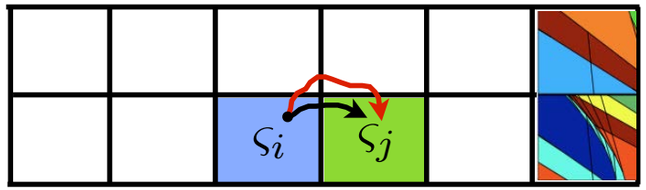
$$\dot{x} = f_\alpha(x, u)$$
$$g_\alpha(x, u, z) \le 0$$



Use formal tools to create abstractions

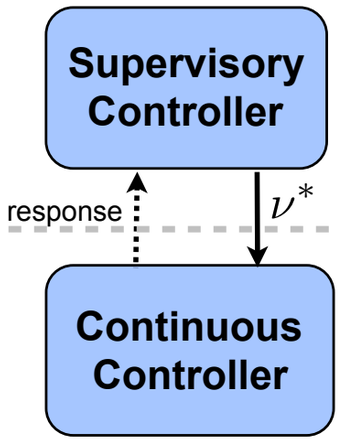
- Use reachability analysis (trajectory gener'n) to compute regions, transitions
- Account for disturbances, uncertainty, failures (using, for example, MPT)

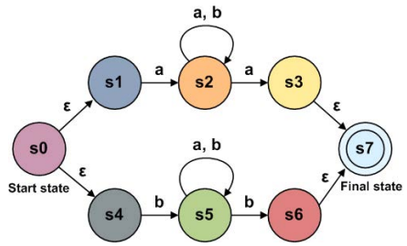
- Look for regions such that we can move from one region to another w/out leaving the union of two regions

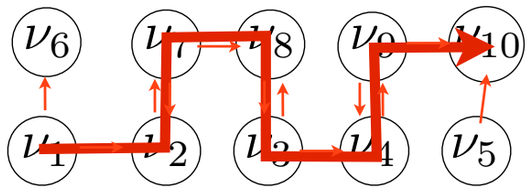


- Solve via trajectory generation algorithm: piecewise linear dynamics w/ disturbances:

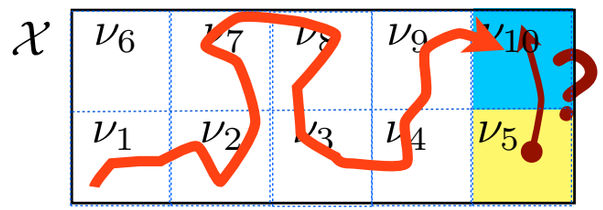
$$s[t+1] = As[t] + Bu[t] + Ed[t]$$
$$s[t] \in \varsigma_i, s[N] \in \varsigma_j, u[t] \in U$$







$$\min J = \int_0^T L_\alpha(x, u)\, dt + V(x(T))$$



$$L \begin{bmatrix} s[0] \\ u[0] \\ \vdots \\ u[N-1] \end{bmatrix} \le M - G \begin{bmatrix} d[0] \\ \vdots \\ d[N-1] \end{bmatrix}$$

Synthesis of Reactive (Feedback) Controllers

Reactive Protocol Synthesis

- Find control action that insures that specification is always satisfied
- For LTL, complexity is doubly exponential (!) in the size of system specification

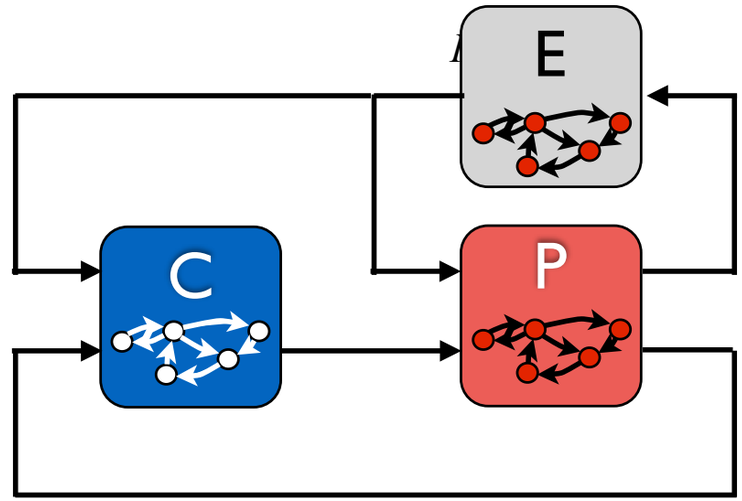
GR(1) synthesis for reactive protocols

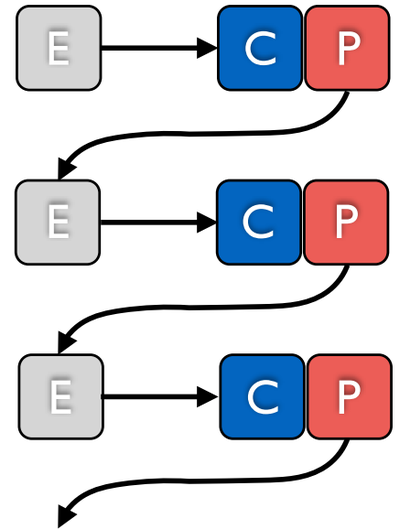
- Piterman, Pnueli and Sa'ar, 2006
- Assume environment fixes action before controller (breaks symmetry)
- For certain class of specifications, get complexity cubic in # of states (!)

$$(\phi^{\mathrm{e}}_{\mathrm{init}} \wedge \Box\phi^{\mathrm{e}}_{\mathrm{safe}} \wedge \Box\Diamond\phi^{\mathrm{e}}_{\mathrm{prog}}) \rightarrow (\phi^{\mathrm{s}}_{\mathrm{init}} \wedge \Box\phi^{\mathrm{s}}_{\mathrm{safe}} \wedge \Box\Diamond\phi^{\mathrm{s}}_{\mathrm{prog}})$$

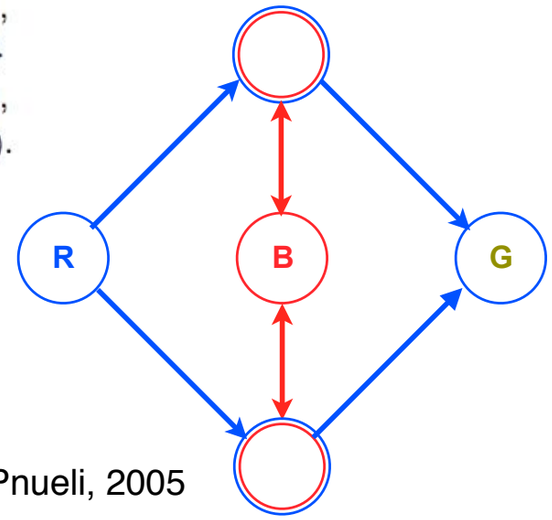
Environment assumption System guarantee

- GR(1) = general reactivity formula
- Assume/guarantee style specification

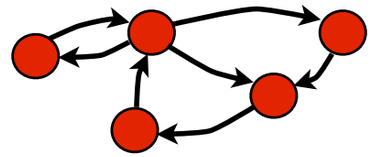




$$\Box\{(\tilde{c}=1 \wedge c=0 \wedge (x_C < T_{c_{min}})) \rightarrow (\bigcirc c=0 \wedge \bigcirc x_C = x_C + \delta)\},$$
$$\Box\{(\tilde{c}=1 \wedge c=0 \wedge (x_C \geq T_{c_{min}})) \rightarrow (\bigcirc c=1 \vee \bigcirc x_C = x_C + \delta)\},$$
$$\Box(x_C \leq T_{c_{max}}).$$



A. Pnueli, 2005

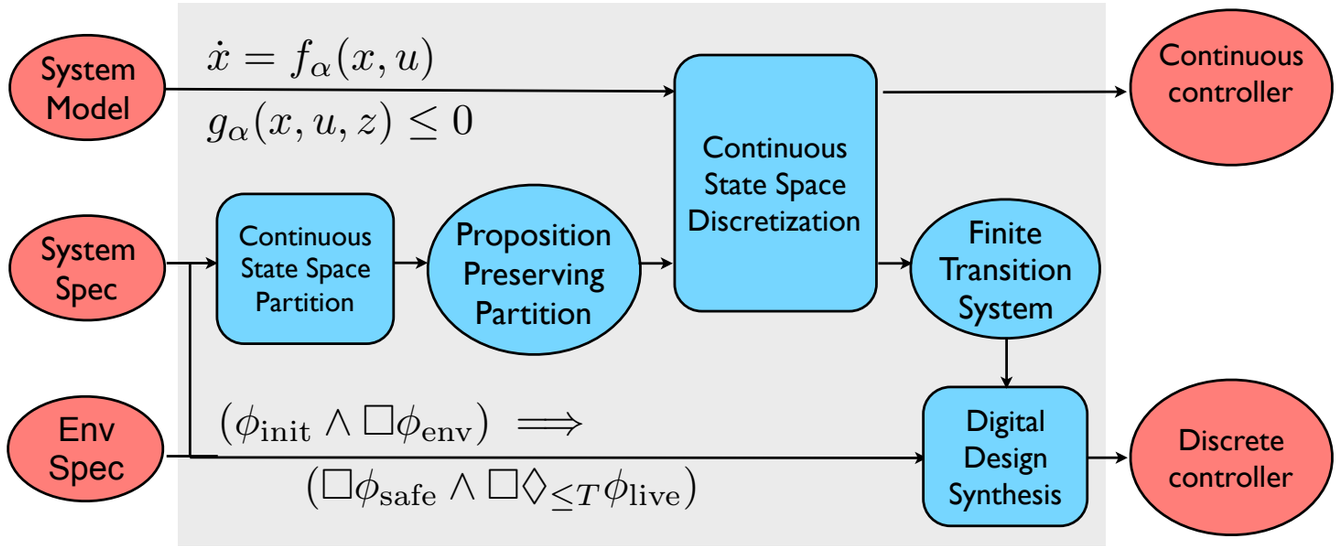


Temporal Logic Planning (TuLiP) toolbox

http://tulip-control.org

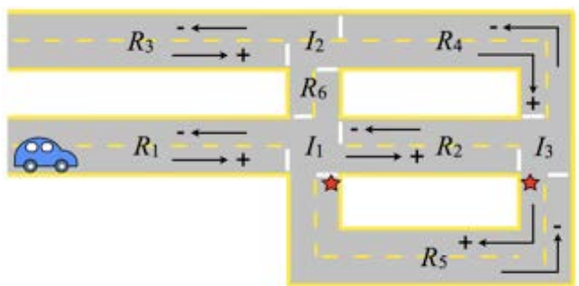
Python Toolbox

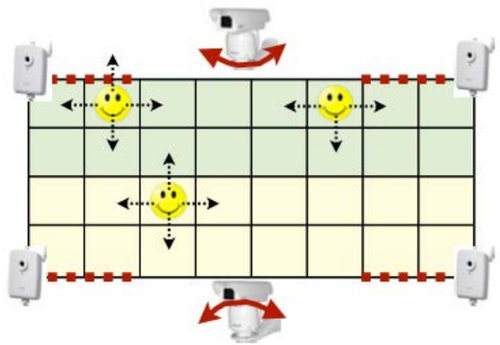
- Transition systems, automata
- GR(1), LTL specs
- Nonlinear dynamics, discretization
- Synthesis: probabilistic (stormpy), reactive, minimum violation planning

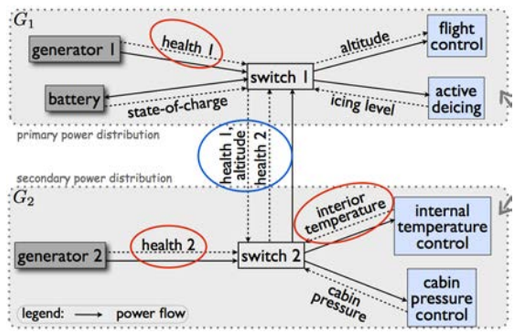


Applications of TuLiP

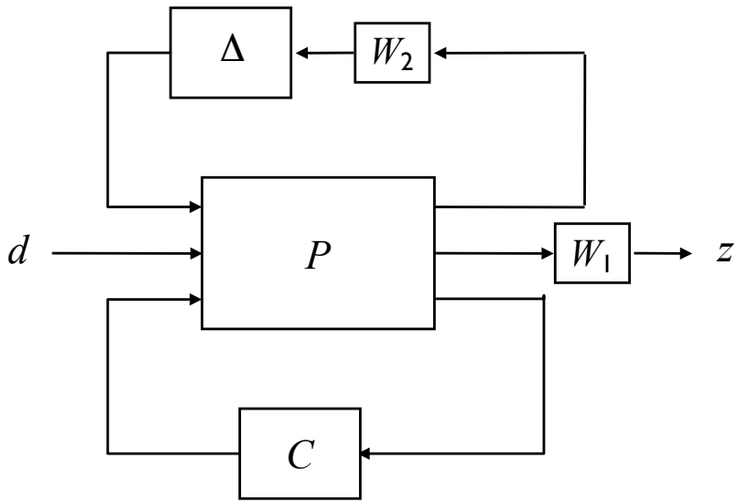
- Autonomous vehicles - traffic planner (intersections and roads, with other vehicles)
- Distributed camera networks - cooperating cameras to track people in region
- Electric power transfer - fault-tolerant control of generator + switches + loads

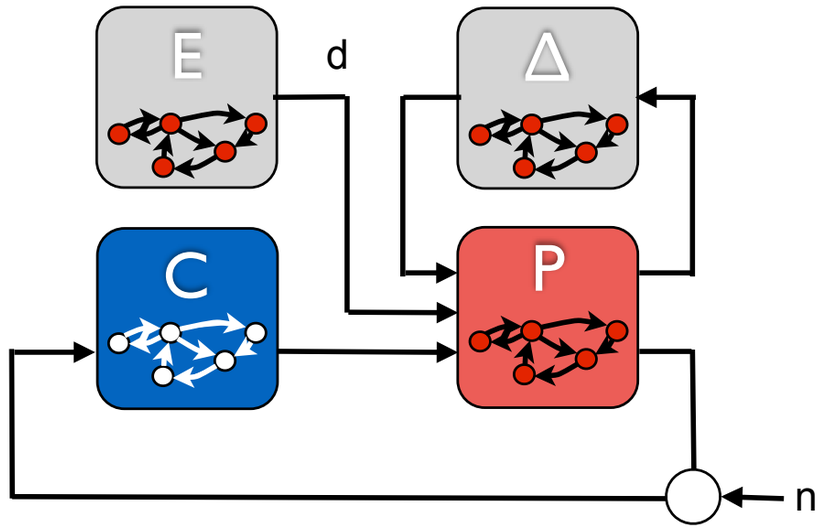






Rapprochement Between Formal Methods and Control





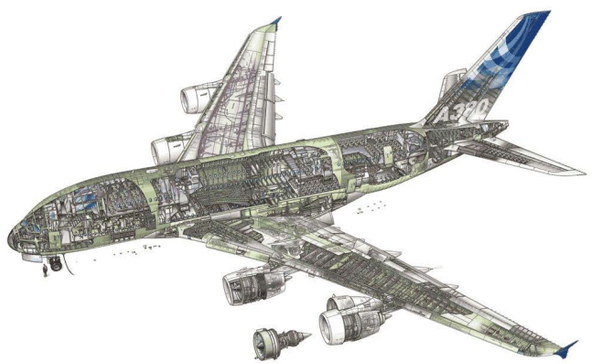
$$\|z\|_2 \le \gamma \|d\|_2 \quad \text{for all} \quad \|\Delta\| \le 1$$

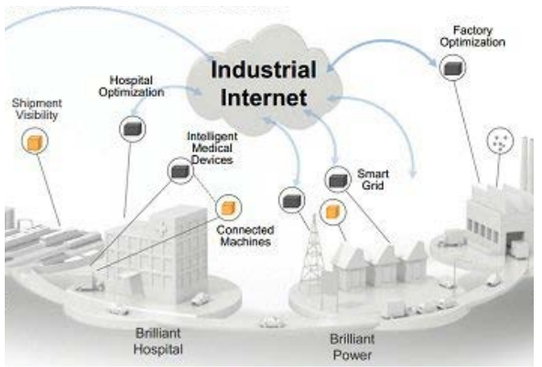
$$\Box\phi^{\mathrm{e}}_{\mathrm{safe}} \wedge \Box\Diamond\phi^{\mathrm{e}}_{\mathrm{prog}} \rightarrow \Box\phi^{\mathrm{s}}_{\mathrm{safe}} \wedge \Box\Diamond_{\le T}\phi^{\mathrm{s}}_{\mathrm{prog}}$$

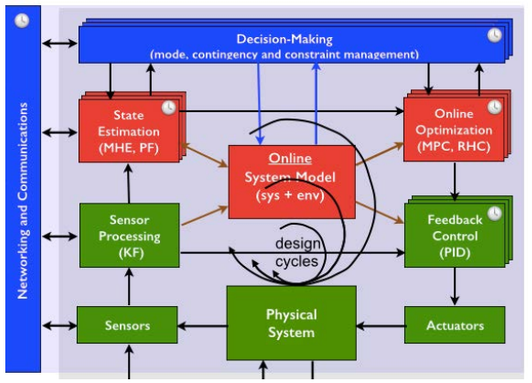
Getting more rigorous about control of reactive systems

- Systems are too complex to be tested by trial and error
- Systems are too safety-critical to be tested by trial and error
- Move from "design then verify" to
  - specify then synthesize
  - synthesis of contracts

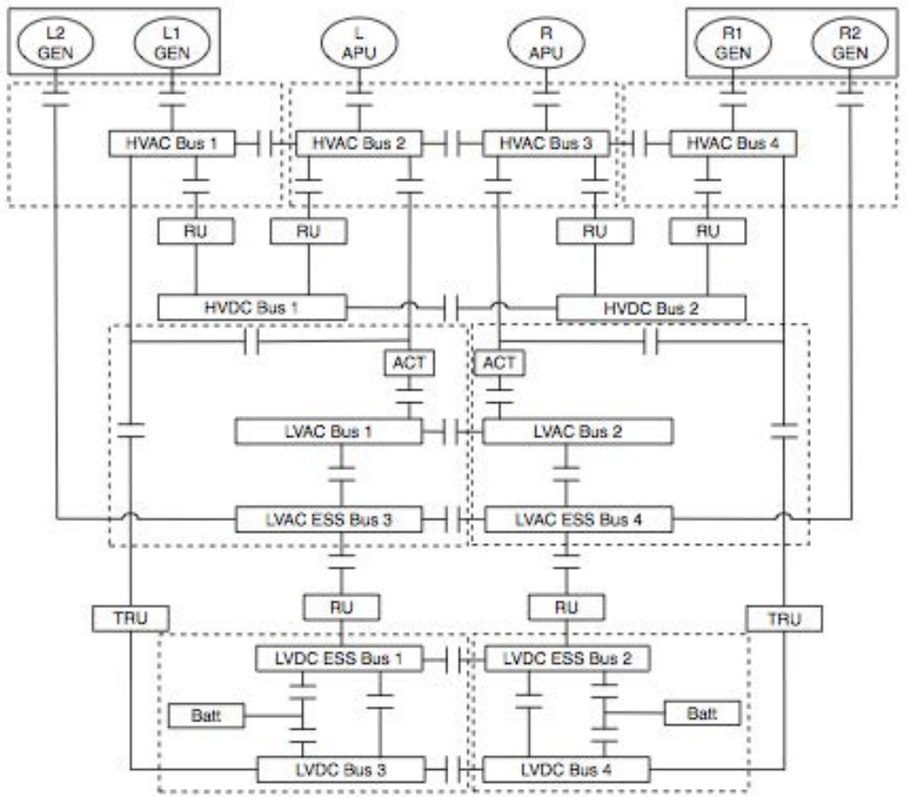
Controlling cyberphysical systems requires solving *both* problems







Example: Electric Power Systems



R. G. Michalko, "Electrical starting, generation, conversion and distribution system architecture for a more electric vehicle," US Patent 7,439,634 B2, Oct. 2008.

REQUIREMENTS:

1. No AC bus shall be simultaneously powered by more than one AC source.
2. The aircraft electric power system shall provide power with the following characteristics: 115 +/- 5 V (amplitude) and 400 Hz (frequency) for AC loads and 28 +/-2V for DC loads.
3. Buses shall be powered according to the priority tables.
4. AC buses shall not be unpowered for more than 50ms.
5. The overall system failure probability must be less than 10^{-9} per flight hour.
6. Never lose more than one bus for any single failure.
7. Total load must be within the capacity of the generator

Component models/specifications:

1. Failure probabilities for contactors, generators, etc. (not much on failure modes)
2. Contactor closure times are between 15-25 ms and opening times are between 10-20 ms.

Properties can be formulated in GR(1)

- Safety: supply power, avoid shorts/ paralleling
- Progress: all loads eventually powered

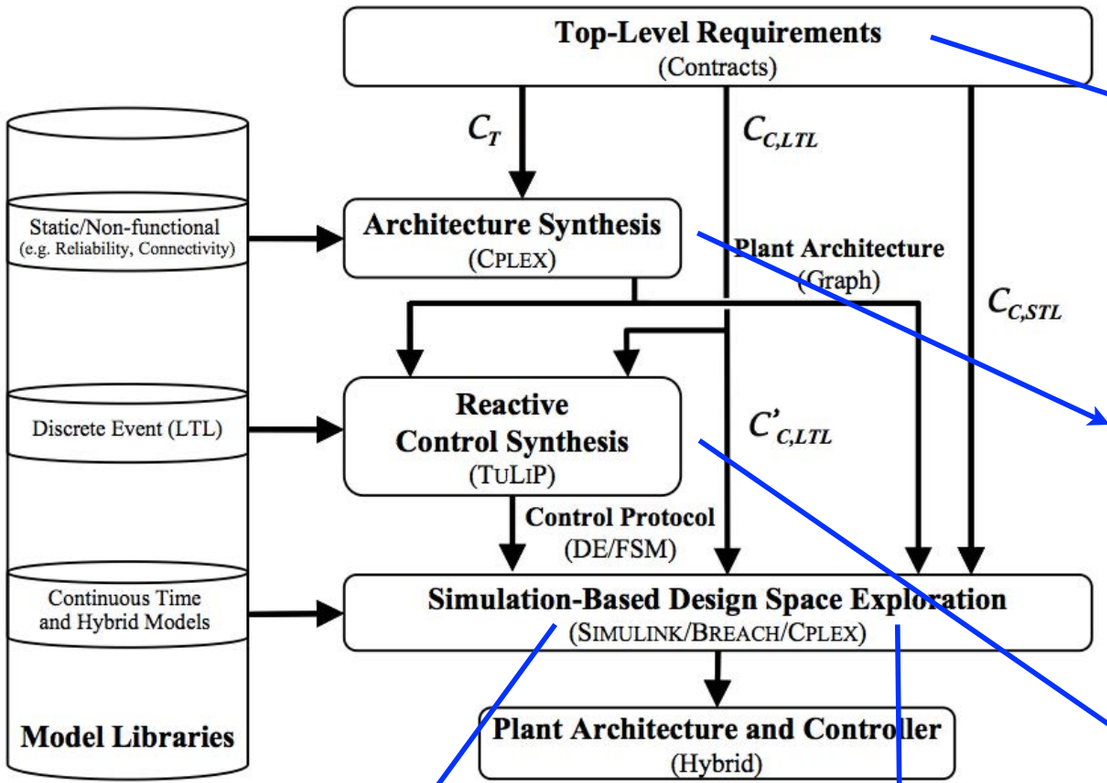
Verification

- Given properties + logic, ensure that specs are satisfied

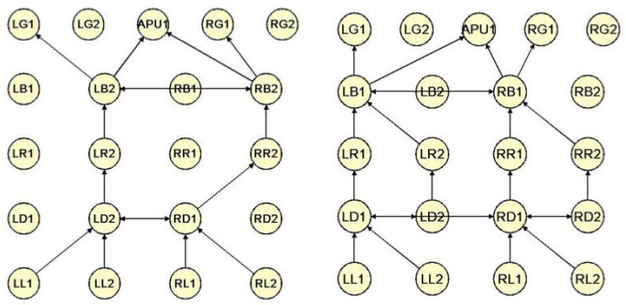
Synthesis

- Given properties and topology + actuators, *synthesize* switching logic

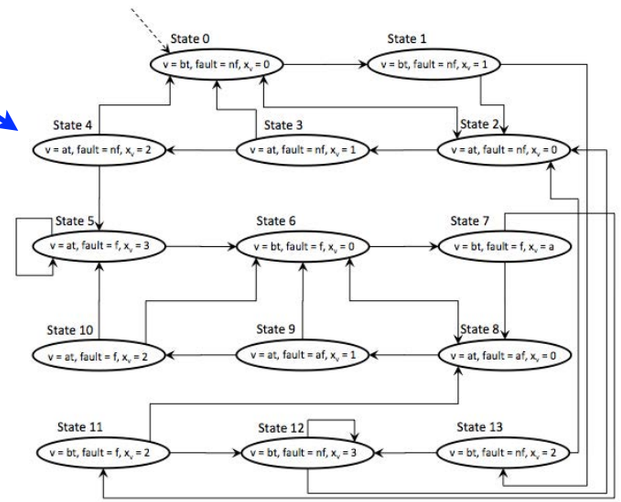
EPS Design Space Exploration

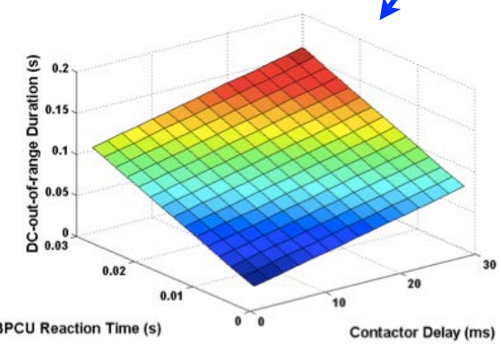


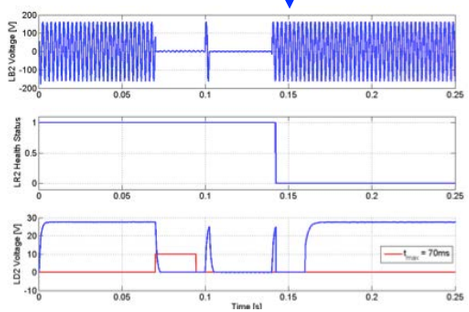
$$\square\{(\tilde{c}=1 \wedge c=0 \wedge (x_C < T_{c_{min}})) \rightarrow (\bigcirc c=0 \wedge \bigcirc x_C = x_C + \delta)\},$$
$$\square\{(\tilde{c}=1 \wedge c=0 \wedge (x_C \geq T_{c_{min}})) \rightarrow (\bigcirc c=1 \vee \bigcirc x_C = x_C + \delta)\},$$
$$\square(x_C \leq T_{c_{max}}).$$



(b) Topology 3 (c) Topology 4







Design workflow

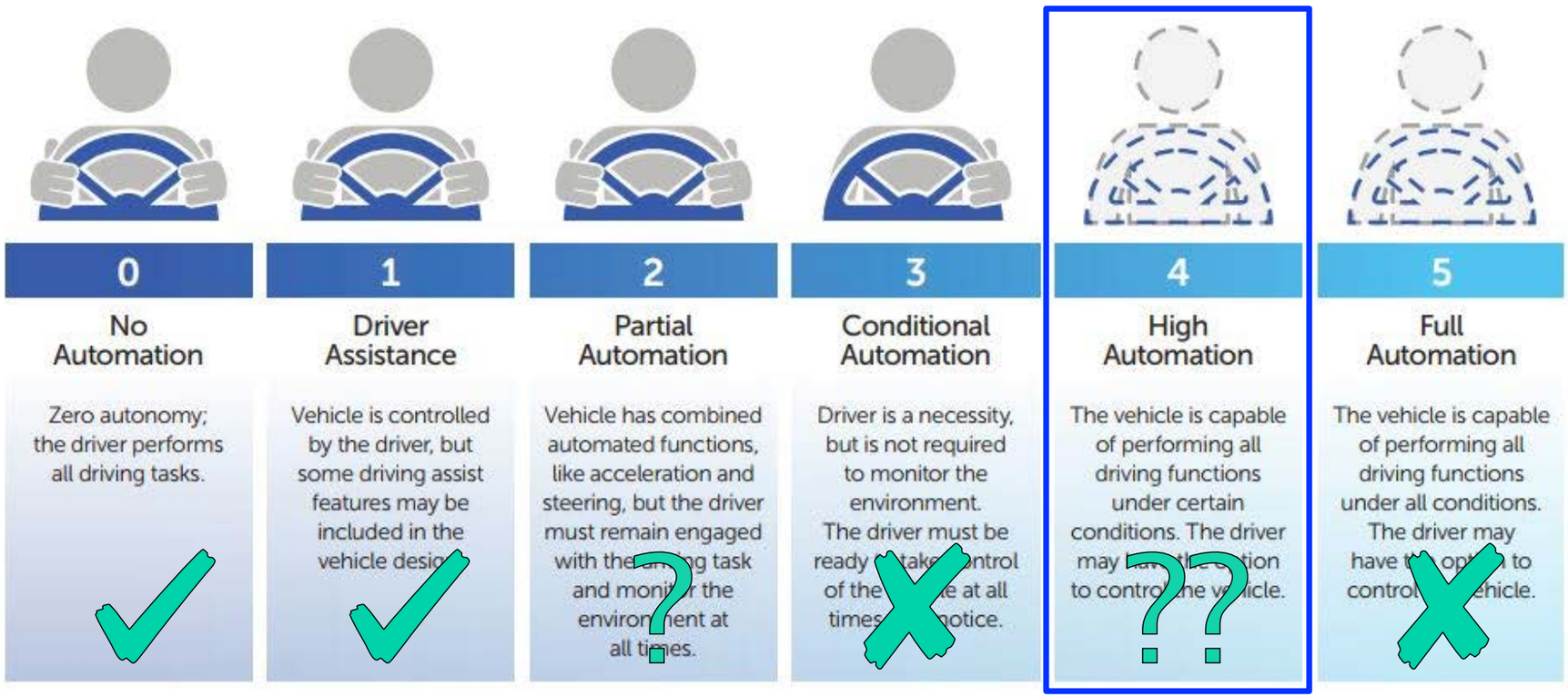
- Formalize specs as a A/G contracts
- Synthesize possible EPS topologies
- Synthesize control logic, if possible
- Use more complex models to verify continuous time properties

Applications

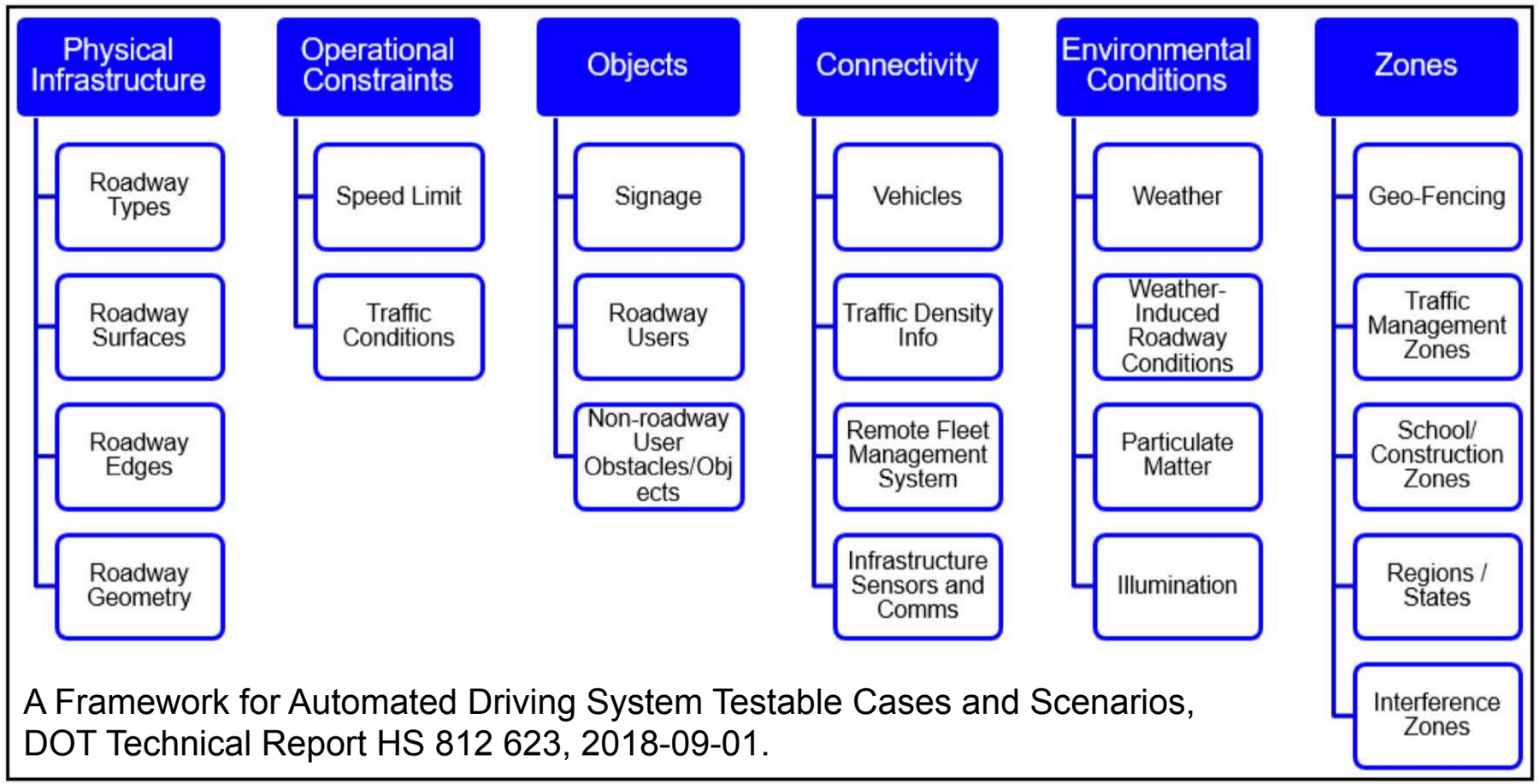
- Aircraft electric power systems
- Environmental control systems

Self-Driving Cars

SAE Levels of Automation:



Operational Design Domains (ODDs)



A Framework for Automated Driving System Testable Cases and Scenarios, DOT Technical Report HS 812 623, 2018-09-01.

SAE J3016: The specific conditions under which a given driving automation system or feature thereof is designed to function, including, but not limited to, driving modes.

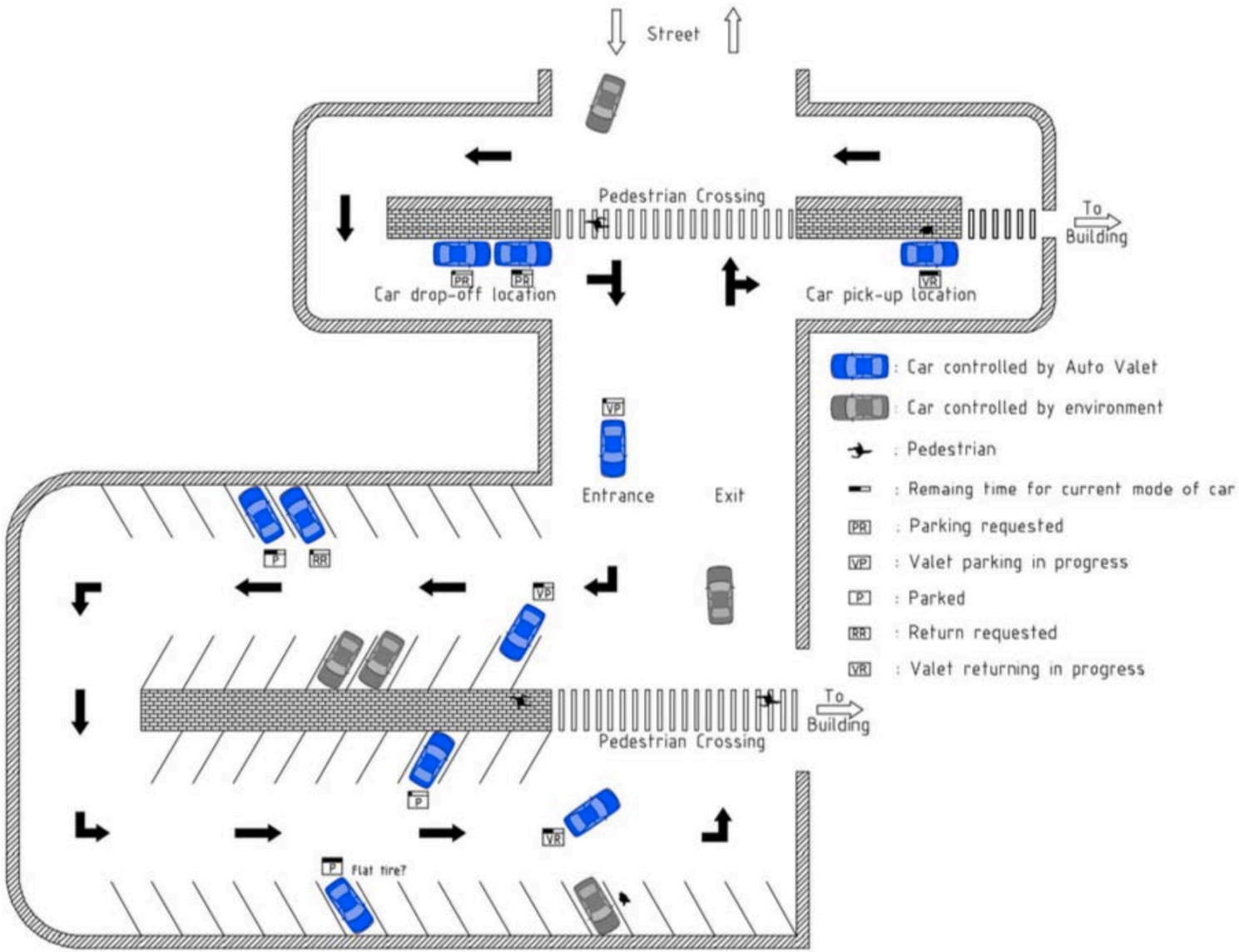
ODD as a tool for L4 autonomy

- Use ODD to describe conditions under which vehicle can operate
- Use online monitors to detect out of ODD events; bring car to a stop

Definition/use of ODDs still evolving

- SAE J3016 + DOT report give concept
- May also be useful for defining vehicle internal status?

Example: Autonomous Valet Parking - Specification



Layer 3 - supervisory protocol

- Respond to requests to deposit or retrieve a car
- Specification: STL formulas using TLA+ (temporal logic of actions)
- Controller: finite state automata

Layer 2 - trajectory optimization

- Find optimal trajectory minimizing fuel and time + avoid obstacles
- Specification: simplified model + cost function and constraints
- Controller: receding horizon (MPC)

Layer 1 - feedback regulation

- Tracking, disturbance rejection
- Controller: PID w/ gain scheduling

Autonomous Valet Parking - Design and Verification

MODULE *ValetParking*

EXTENDS *Naturals*, *Reals*, *Sequences*, *FiniteSets*

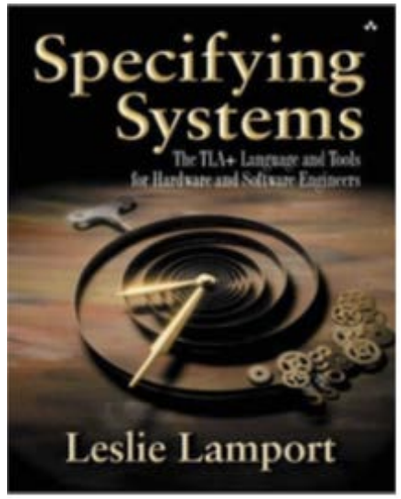
CONSTANTS *P*, The total number of parking spots.
LIC_NUM, The set of license plate numbers of all cars in the universe.
PARK_TL, Time limit for parking a car.
RETRIEVE_TL, Time limit for retrieving a car.
WAIT_TL, Maximum waiting time allowed.
SPACES, All parking spaces.
SPACE_STATUS, Set of all possible status of a parking space.
QUEUE_STATUS Set of all possible status of a car in queue.

Anything that is not a license plate number.

$NOT\_LIC\_NUM \triangleq \text{CHOOSE } num : num \notin LIC\_NUM$

ASSUME $\wedge P \in Nat$
$\wedge PARK\_TL \in Real$
$\wedge PARK\_TL > 0$
$\wedge RETRIEVE\_TL \in Real$
$\wedge RETRIEVE\_TL > 0$
$\wedge WAIT\_TL \in Real$
$\wedge WAIT\_TL > 0$
$\wedge SPACES = 1 \,..\, P$
$\wedge SPACE\_STATUS = [car : LIC\_NUM \cup \{NOT\_LIC\_NUM\},$
$return\_requested : \text{BOOLEAN},$
$been\_waiting\_for : Real]$
$\wedge QUEUE\_STATUS = [car : LIC\_NUM,$
$been\_waiting\_for : Real]$

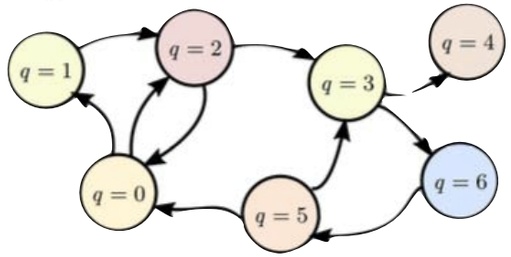
VARIABLES *lot_info*, An array of status of each parking spot.
queue_info, An array of status of the parking queue.
cars_in_lot, All cars currently in the lot



$Spec = Init \wedge \Box[Next]_{vars} \wedge TimeFlow$

THEOREM

$Spec \Rightarrow \wedge TypeInvariant$
$\wedge ReasonableWaitTimes$
$\wedge CarsPreserved$



Layer 3 - supervisory protocol

- Respond to requests to deposit or retrieve a car
- Specification: STL formulas using TLA+ (temporal logic of actions)
- Controller: finite state automata

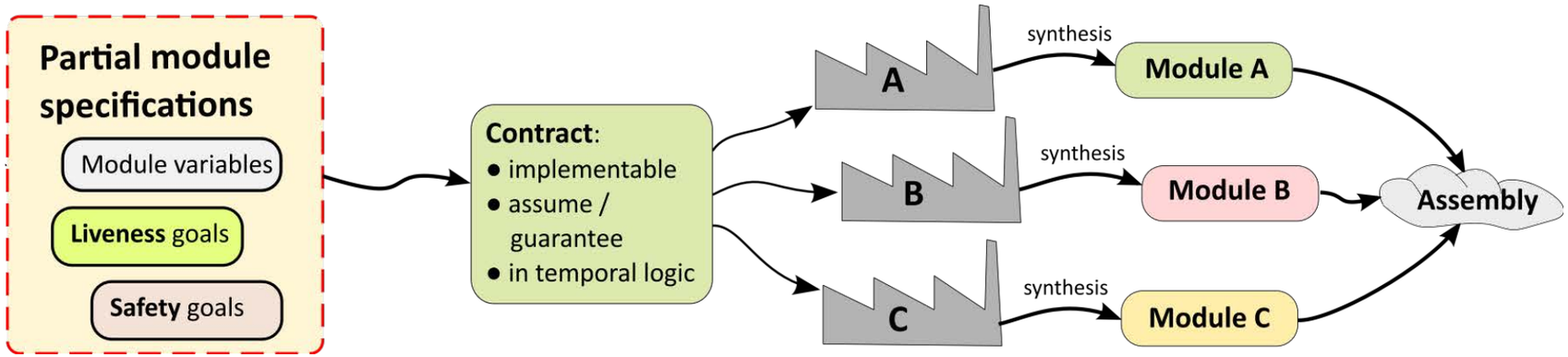
Layer 2 - trajectory optimization

- Find optimal trajectory minimizing fuel and time + avoid obstacles
- Specification: simplified model + cost function and constraints
- Controller: receding horizon (MPC)

Layer 1 - feedback regulation

- Tracking, disturbance rejection
- Controller: PID w/ gain scheduling

Structure of Specifications for a System



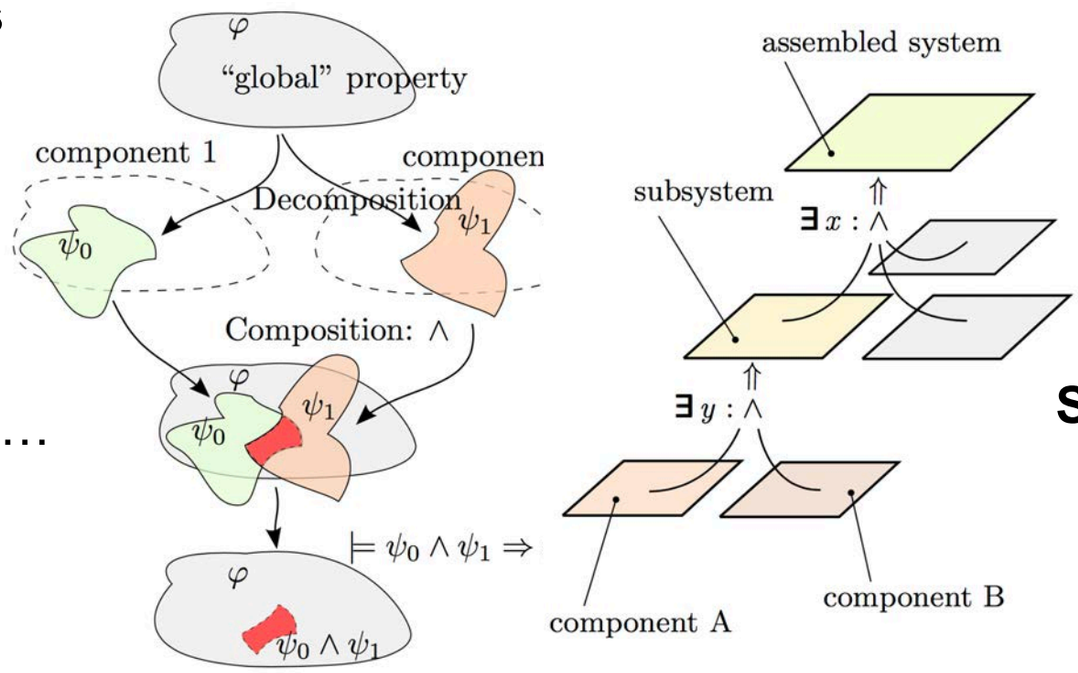
Assume/guarantee contracts

- Assume: properties of other components in the system
- Guarantee: properties that will hold for my component

$$A_i \Rightarrow G_i$$

$$G_2 \wedge G_3 \Rightarrow A_1,\ G_1 \wedge G_3 \Rightarrow A_2,\ \ldots$$

- Contracts can be horizontal (within a layer) or vertical (between two layers)



Synthesis of contracts

- Given a set of (LTL) properties, *synthesize* GR(1) contracts for components
- Key component is amount of information that must be shared
  - Can minimize subject to constraints

Software (I. Filippidis)

- omega - synthesis of controllers/contracts
- dd - binary decision diagrams in Python

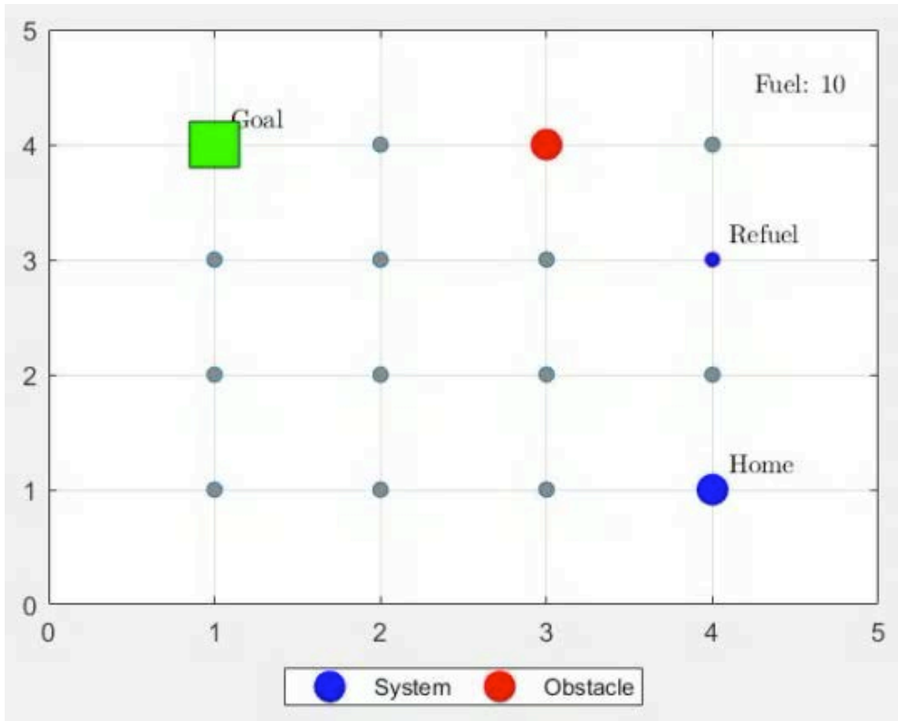
Generating Test Sequences for GR(1) Specifications

Desired properties for test sequences

- Coverage of system/environment states and actions
- Generation of environmental states/actions that limit the number of satisfying actions the process can take

Gridworld example:

- System spec:
  - Avoid obstacle (perfect knowlege)
  - Maintain fuel > 0
  - Follow rules of the road (grid)
  - Park = ON $\Longrightarrow$ System must leave HOME and eventually reach GOAL
  - Park = OFF $\Longrightarrow$ System must leave GOAL and eventually reach HOME
- Environment spec
  - Obstacle: stay in restricted region
  - Park signal can turn on/off at any time



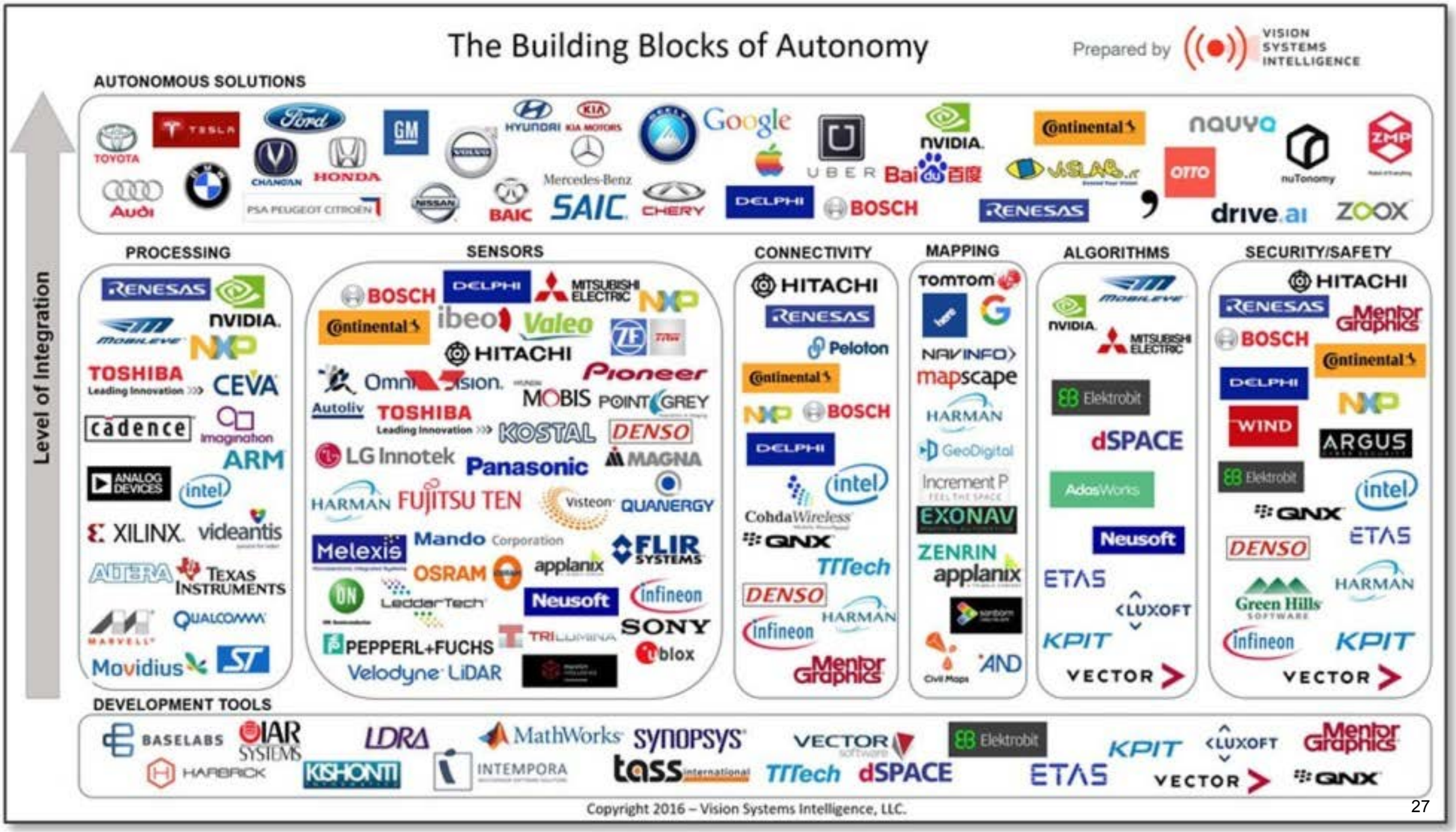
$$\square\phi^{\mathrm{e}}_{\mathrm{safe}} \wedge \square\lozenge\phi^{\mathrm{e}}_{\mathrm{prog}} \rightarrow \square\phi^{\mathrm{s}}_{\mathrm{safe}} \wedge \square\lozenge_{\leq T}\phi^{\mathrm{s}}_{\mathrm{prog}}$$

Test plan generation

- Try to choose obstacle trajectories that force system toward its limits

The Building Blocks of Autonomy

Prepared by VISION SYSTEMS INTELLIGENCE



Challenges in Self Driving

Solved (or solvable):

- Driving on city streets/free-ways with normal traffic and everyone obeying the rules
- Obtaining accident rates on par w/ humans (~7 per 10^9)
- 99%+ of miles completed in self-driving mode
- Using ML to improve performance
- 1-2% of cars are self-driving, w/ humans as a "backup", accidents OK

Work remains to be done:

- Driving in crowded and/or unstructured environments (rules may not exist/hold)
  - Requires understanding and prediction, not just mimicking
- Obtaining 10-100X better safety
  - Higher than MTBF of parts
- 99%+ of all *trips* completed, in environments with humans
  - Accurate predictions + asserting "intent" to make progress
- Using ML to guarantee safety
  - Hard to learn rare events
  - Combine w/ formal methods?
- 1-80% of cars are fully self driving (L4), with fewer accidents than humans
  - 90%+ $\Longrightarrow$ probably gets *easier*

http://www.path.berkeley.edu/publications/national-automated-highway-systems-consortium

Self-Driving Cars: Value Proposition (?)

Scenario #1: purchase by individuals

- Extension of current L2 functionality to more complex functions
- Examples: Tesla, traditional manufacturers?
- Value: less attention/skill required to “drive”
- Challenge: cost, reliability, handoff

Scenario #2: robo-taxis

- L4 autonomy for ride-share in urban environments
- Value: fleet-level economics; replace human, extend utilization
- Examples: Cruise, Waymo, Zoox
- Challenge: cost, reliability, public perception (?)

Scenario #3: limited/structured environments

- L4/L5 autonomy in structure/constructed environments
- Examples: airports (Heathrow), mines, highways, new cities?
- Value: automate dirty, dull, dangerous tasks
- Challenge: cost of purpose-build infrastructure, market size

Summary: Safety-Critical Autonomous Systems

Aerospace Systems

- Software failure rate of 1 in 10^9 flight hours
- Challenges in terms of design time and cost

Some math and (control) theory

- Multi-layer hierarchies, contracts to manage complexity, formal methods
- Verification and synthesis tools to ensure correctness

Self-driving cars

- Very complex problem, especially for humans + autonomous systems
- Not clear if we have the tools to design safe systems at reasonable cost…
- Good area for fundamental research in (safety-critical) real-time decision making