Specification, Design and Verification of Distributed Embedded Systems

Richard M. Murray
California Institute of Technology

Sayan Mitra	**Ufuk Topcu**	**Nok Wongpiromsarn**
UIUC (CMI)	**Caltech CDS**	**Caltech ME**

V&V MURI 2009 Annual Review
17 September 2009

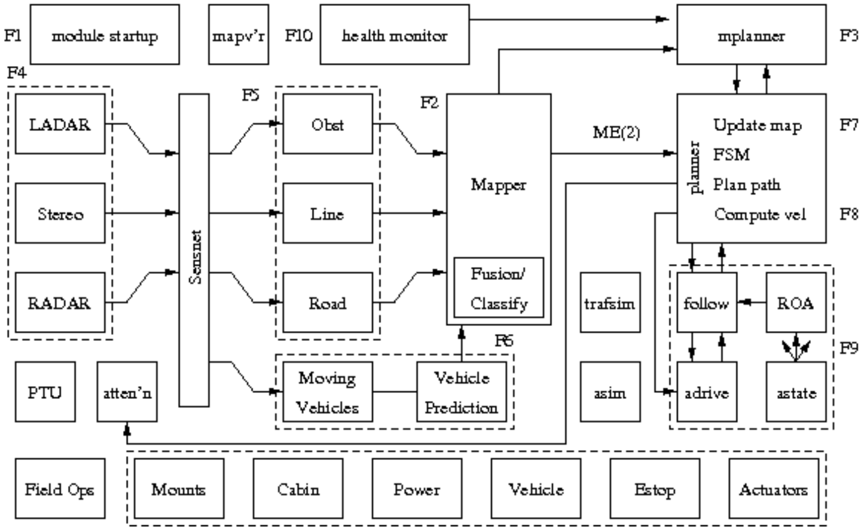
Motivating Example: Alice (DGC07)

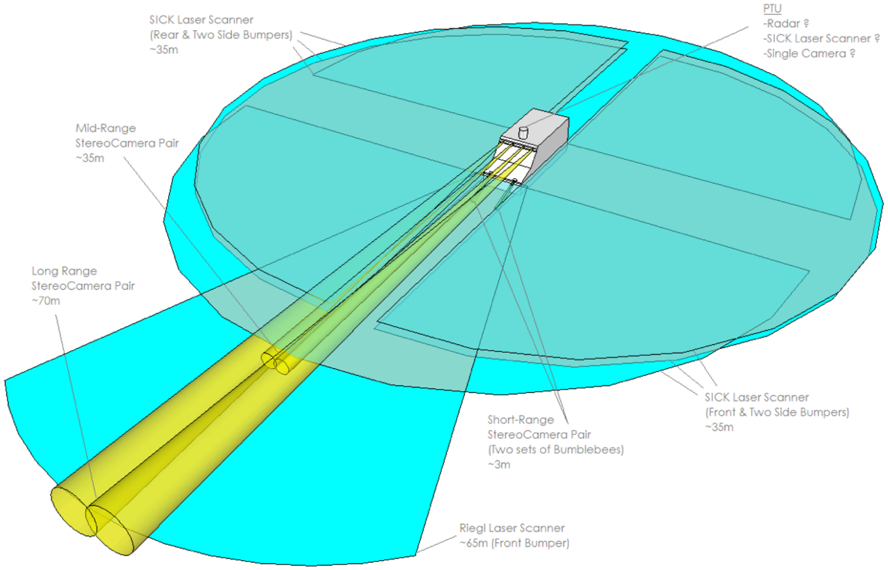
Alice

- 300+ miles of fully autonomous driving
- 8 cameras, 8 LADAR, 2 RADAR
- 12 Core 2 Duo CPUs + Quad Core
- ~75 person team over 18 months

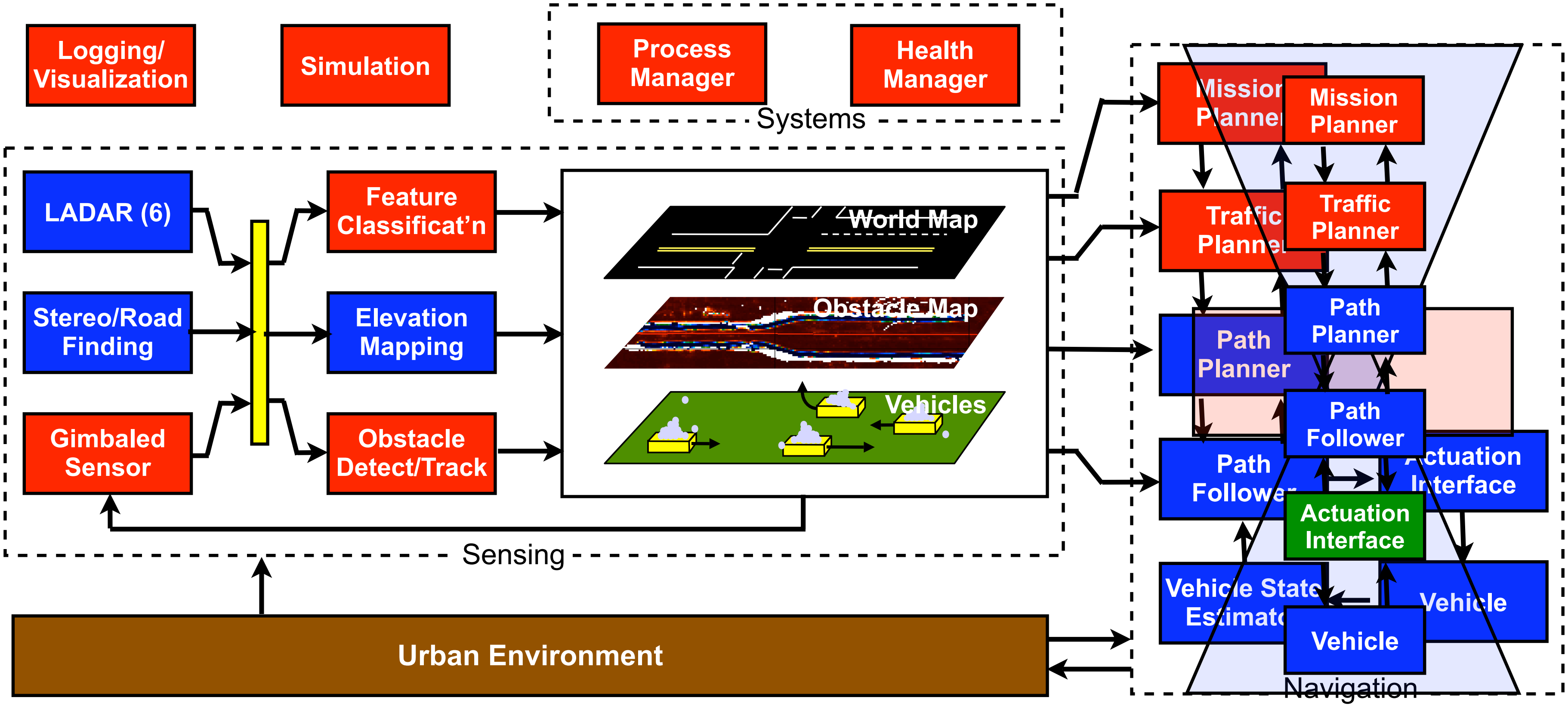
Software

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





System Architecture


Logging/ Visualization

Simulation

Process Manager

Health Manager

Systems

LADAR (6)

Stereo/Road Finding

Gimbaled Sensor

Feature Classificat'n

Elevation Mapping

Obstacle Detect/Track

World Map

Obstacle Map

Vehicles

Sensing

Urban Environment

Mission Planner

Mission Planner

Traffic Planner

Traffic Planner

Path Planner

Path Planner

Path Follower

Path Follower

Actuation Interface

Actuation Interface

Vehicle State Estimator

Vehicle

Vehicle

Navigation

V&V focus: planning "stack"

- Hourglass architecture: reasoning at interconnected layers of abstraction
- Apply different tools to verify different aspects of the design
- Evolution from verification → design for verification → proof by construction

Specifying Behavior with Temporal Logic

Description

- State of the system is a snapshot of values of all variables
- Reason about *behaviors* σ: sequence of states of the system
- No strict notion of time, just ordering of events
- *Actions* are relations between states: state s is related to state t by action a if a takes s to t (via prime notation: $x' = x + 1$)
- *Formulas* (specifications) describe the set of allowable behaviors
- Safety specification: what actions are allowed
- Fairness specification: when can a component take an action (eg, infinitely often)

- $\Box p \equiv$ **always** p (invariance)
- $\Diamond p \equiv$ **eventually** p (guarantee)
- $p \rightarrow \Diamond q \equiv$ p **implies eventually** q (response)
- $p \rightarrow q \,\mathcal{U}\, r \equiv$ p **implies** q **until** r (precedence)
- $\Box\Diamond p \equiv$ **always eventually** p (progress)
- $\Diamond\Box p \equiv$ **eventually always** p (stability)
- $\Diamond p \rightarrow \Diamond q \equiv$ **eventually** p **implies eventually** q (correlation)

Example

- Action: $a \equiv x' = x + 1$
- Behavior: $\sigma \equiv x := 1,\ x := 2,\ x := 3,\ \ldots$
- Safety: $\Box x > 0$ (true for this behavior)
- Fairness: $\Box(x' = x + 1 \vee x' = x) \wedge \Box\Diamond\,(x' \neq x)$

Properties

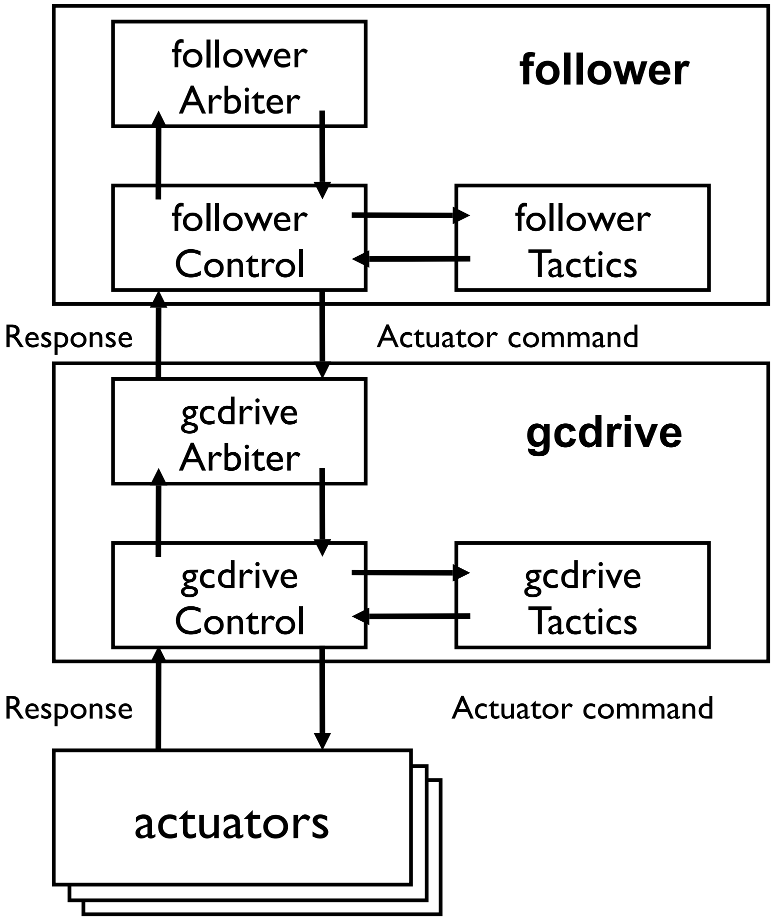
- Can reason about time by adding "time variables" ($t' = t + 1$)
- Specifications and proofs can be difficult to interpret by hand, but computer tools existing (eg, TLC, Isabelle, PVS, etc)

DGC Example: Changing Gear

Wongpiromsarn and M
CDC 2008

Verify that we can't drive while shifting or drive in the wrong gear

- Five component: follower Control, gcdrive Arbiter, gcdrive Control, actuators and network
- Construct temporal logic models for each component (including network)



follower

follower Arbiter

follower Control

follower Tactics

Response

Actuator command

gcdrive

gcdrive Arbiter

gcdrive Control

gcdrive Tactics

Response

Actuator command

actuators

Asynchronous operation

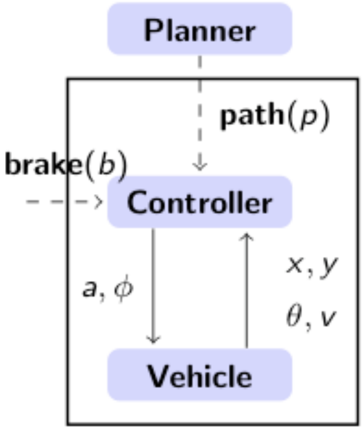
- Notation: $Message_{mod,dir}$ - message to/from a module; Len = length of message queue
- Verify: follower has the right knowledge of the gear that we are currently in, or it commands a full brake.
  - $\Box\,((Len(TransResp_{f,r}) = Len(Trans_{f,s})) \wedge TransResp_{f,r}[Len(TransResp_{f,r})] = \text{COMPLETED} \Rightarrow Trans_f = Trans))$
  - $\Box\,(Trans_f = Trans \vee Acc_{f,s} = -1)$
- Verify: at infinitely many instants, follower has the right knowledge of the gear that we are currently in, or we have hardware failure.
  - $\Box\Diamond\,(Trans_f = Trans = Trans_{f,s}[Len(Trans_{f,s})] \vee HW\ failure)$

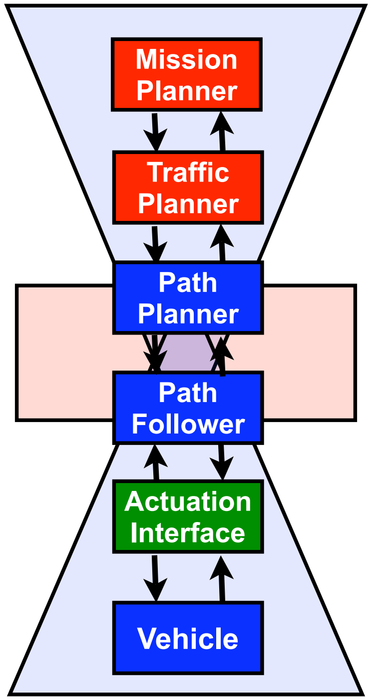
Wongpiromsarn, Mitra and M
HSCC09

Verification of Periodically Controlled Hybrid Systems

Hybrid system: continuous dynamics + discrete updates

- Vehicle
  - Captures the state (position, orientation and velocity) of the vehicle.
  - Specifies the dynamics of the autonomous ground vehicle with respect to the acceleration and the angle of the steering wheel.
  - Limits the magnitude of the steering input to ϕ_{max}.
- Controller
  - Receives the state of the vehicle, a path and an externally triggered brake input.
  - Periodically computes the input steering
  - Restricts the steering angle to δv for mechanical protection of the steering.
  - Sampling period: $\Delta \in R_+$.
- Desired properties
  - (Safety) At all reachable states, the deviation of the vehicle from the current path is upper-bounded by e_{max}.
  - (Progress) The vehicle reaches successive waypoints.



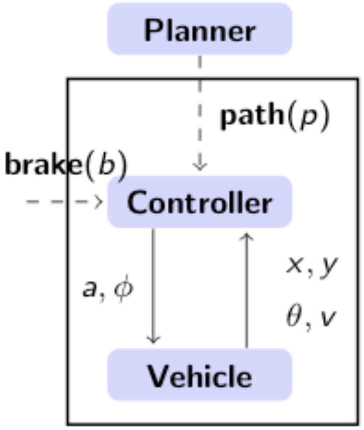


Wongpiromsarn, Mitra and M
HSCC09

Periodically Controlled Hybrid Automata (PCHA)

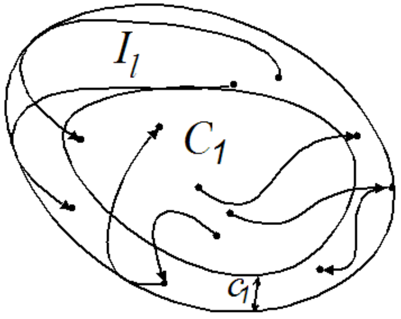
PCHA setup

- Continuous dynamics with piecewise constant inputs
- Controller executes with period $T \in [\Delta_1, \Delta_2]$
- Input commands are received asynchronously
- Execution consists of trajectory segments + discrete updates
- Verify safety (avoid collisions) + performance (turn corner)



Proof technique: verify invariant (safe) set via barrier functions

- Let I be an (safe) set specified by a set of functions $F_i(x) \geq 0$
- Step 1: show that the control action renders I invariant
- Step 2: show that between updates we can bound the continuous trajectories to live within appropriate sets
- Step 3: show progress by moving between nested collection of invariant sets $I_1 \rightarrow I_2$, etc



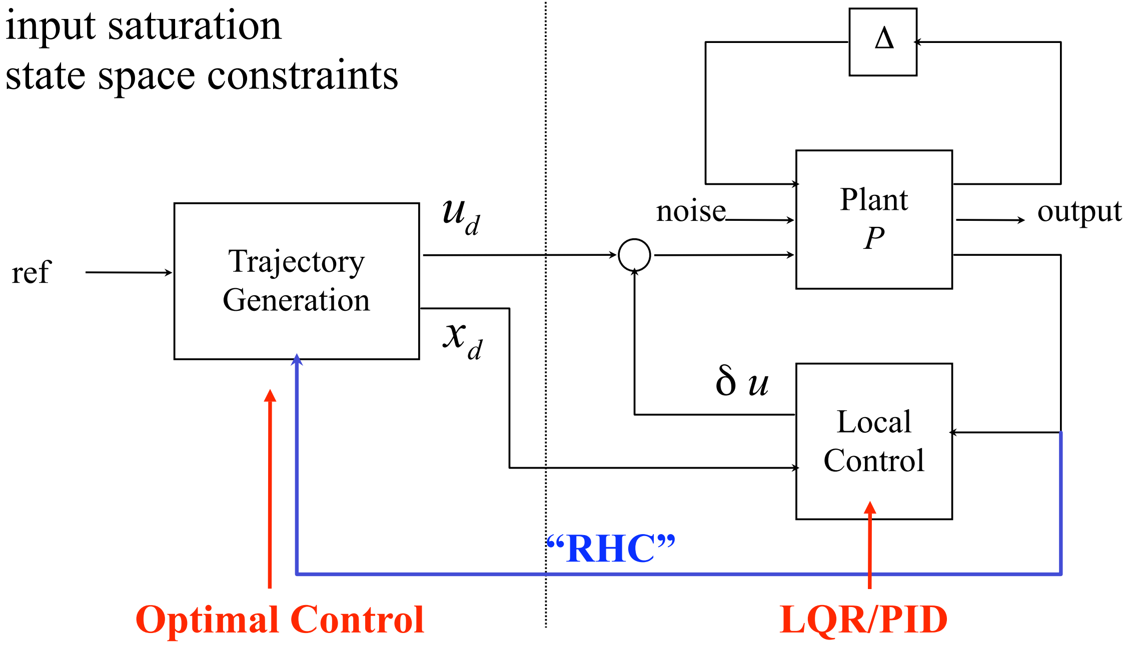
Remarks

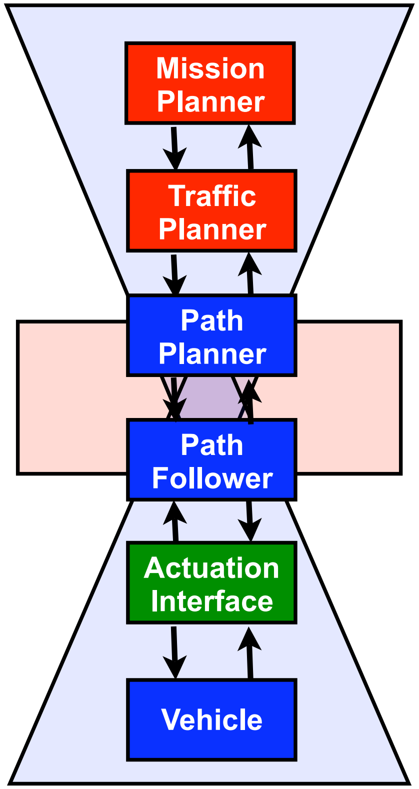
- Can use this to show that settings in Alice were not properly chosen; modified settings lead to proper operation (after the fact)
- Very difficult to find invariant sets (barrier functions) for given control system...

Moving up the Planning Stack

Nonlinear design
- global nonlinearities
- input saturation
- state space constraints

Local design





Extending RHC to planning is tricky

- Modes as integers => MILP (slow)
- Hard to encode temporal logic specifications as cost functions
  - Eg, intersection operations

Approach: *rapidly* explore feasible paths

- Enumerate all executions, then eliminate executions that violate LTL specs
- Issue: state space explosion, especially due to environment

Wongpiromsarn, Topcu and M
CDC 09 (s)

Receding Horizon Control for Linear Temporal Logic

Find planner (logic + path) to solve general control problem

$$(\varphi_{init} \ \wedge \ \Box\varphi_e) \implies (\Box\varphi_s \ \wedge \ \Diamond\varphi_g)$$

- φ_{init} = init conditions
- φ_e = envt description
- φ_s = safety property
- φ_g = planning goal

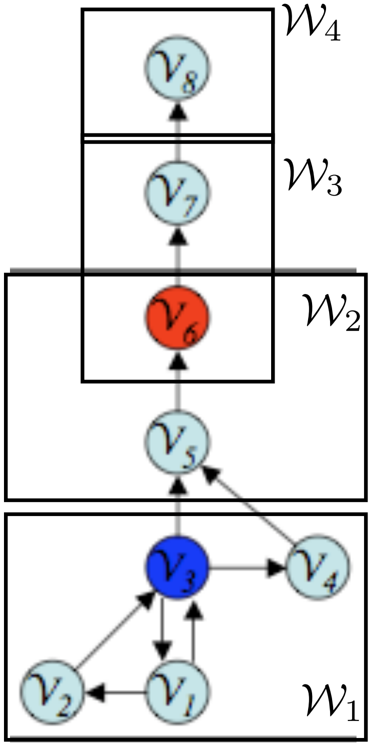
- Can find automaton to satisfy this formula in $O((nm|\Sigma|^3)$ time (!)

Basic idea

- Discretize state space into regions $\{\mathcal{V}_i\}$ + interconnection graph
- Organize regions into a partially ordered set $\{\mathcal{W}_i\}$; $\mathcal{W}_j \preceq_{\varphi_g} \mathcal{W}_i$
  $\Rightarrow$ if state starts in $\mathcal{W}_i$, must transition through $\mathcal{W}_j$ on way to goal
- Find a finite state automaton $\mathcal{A}_i$ satisfying

$$\Psi_i = ((v \in \mathcal{W}_i) \ \wedge \ \Phi \ \wedge \ \Box\varphi_e) \implies (\Box\varphi_s \ \wedge \ \Diamond(v \in \mathcal{W}_{g_i}) \ \wedge \ \Box\Phi)$$

  - Φ describes receding horizon invariants (eg, no collisions)
  - Automaton states describe sequence of regions we transition through; $\mathcal{W}_{g_i} \preceq_{\phi_g} \mathcal{W}_i$ is intermediate (fixed horizon) goal
  - Planner generates trajectory for each discrete transition
  - Partial order condition guarantees that we move closer to goal



Properties

- Provably correct behavior according to spec

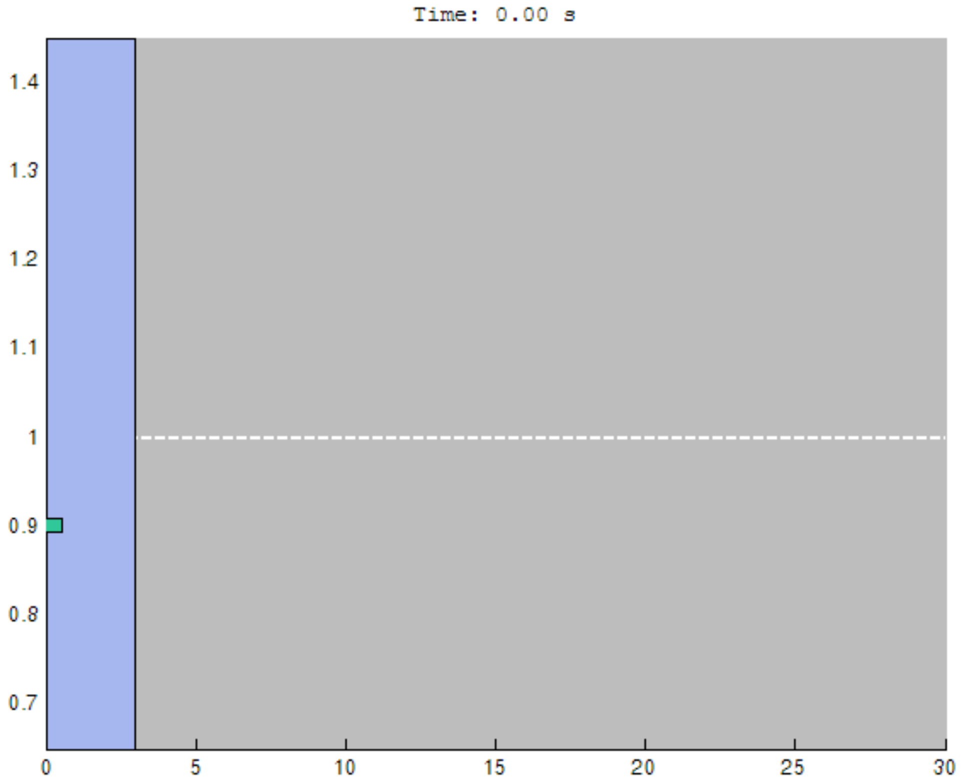
Wongpiromsarn, Topcu and M
CDC 09 (s)

Comments and Example

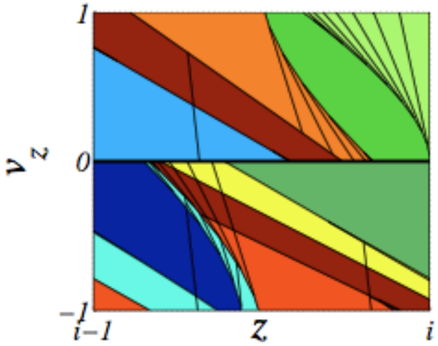
Comments and caveats

- Automaton synthesis is basically searching thru all feasible trajectories (efficiently)
- Complexity is polynomial, but can still get large ⇒ receding horizon is a huge help!
- Discretization of the state space is important and non-trivial

Example: driving down a lane with unknown obstacles



- Demonstrates basic feasibility of approach
- Lots of tuning required to get everything to work
- Clever discretization + RHC are key enablers...



Summary and Next Steps

Specification, Design and Verification for Alice

- Most of the actual design was ad hoc; with lots of testing
- Starting to develop tools for systematic design, verification

Analysis techniques based on invariants & model checking

- Specify desired behavior in terms of temporal logic
- Model checking using existing tools (TLA+, TLC, SPIN, ...)
- Theorem proving techniques using Lyapunov fcns, lattices

Synthesis techniques for LTL specifications using receding horizon planning

- Convert the specification into a design criterion
- Use fast solvers to find trajectories that satisfy constraints (including temporal logic specifications)
- Manage complexity using receiding horizon approach

Next steps

- More systematic design of regions, lattices, invariants
- Better integration of trajectory planning and logic planning