Principles of Cyber-Physical Systems

Chapter 9: Hybrid Systems

Instructor: Lanshun Nie

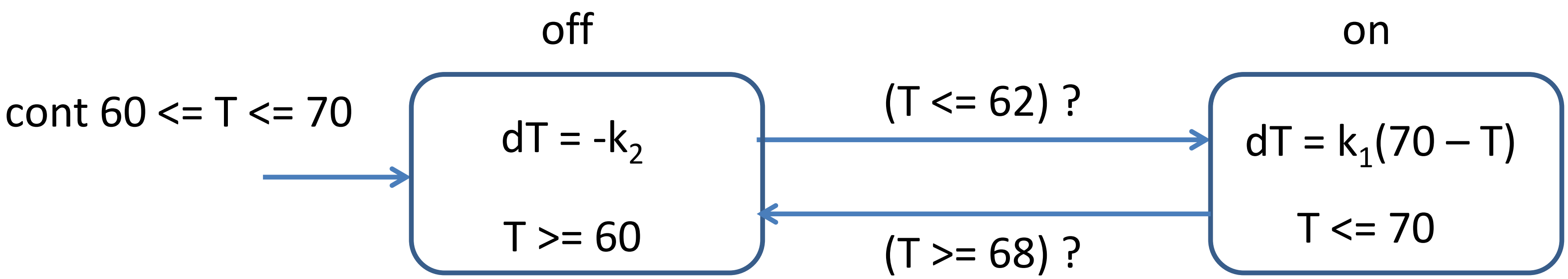
Models of Reactive Computation

- Continuous-time model for dynamical system
  - Synchronous, where time evolves continuously
  - Execution of system: Solution to algebraic / differential equations

- Timed model
  - Like asynchronous for communication of information
  - Clocks evolve continuously, and constraints on delays allow synchronous/global coordination

- Hybrid systems
  - Generalization of timed processes
  - During timed transitions, evolution of state/output variables specified using differential equations as in dynamical systems

Self-Regulating Switching Thermostat



State machine with two modes +

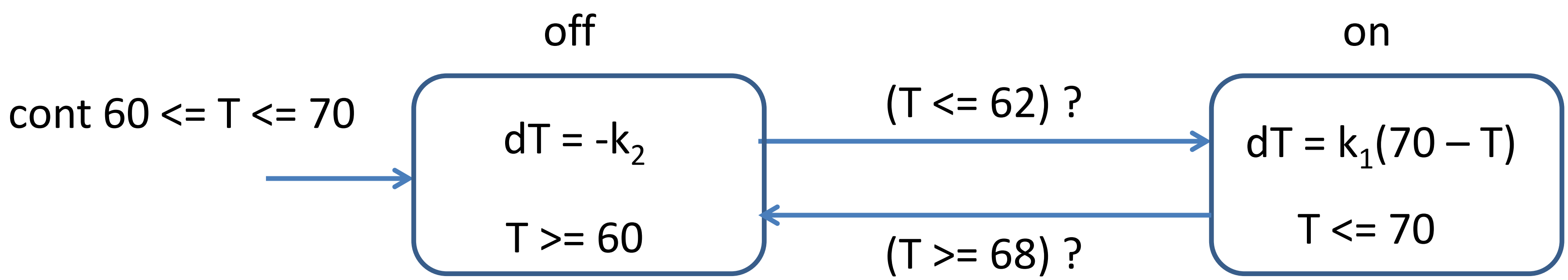
State variable T to model temperature: type **cont**

T can be tested and updated during mode-switches

T changes continuously during timed transitions given by differential equations

Invariants (as in timed model) constrain how long can a timed transition be

Executions of Thermostat



Initial state = (off, T_0) with T_0 in the interval [60,70]

During a timed transition, T decreases continuously: $\mathbf{T}(t) = T_0 - k_2\, t$

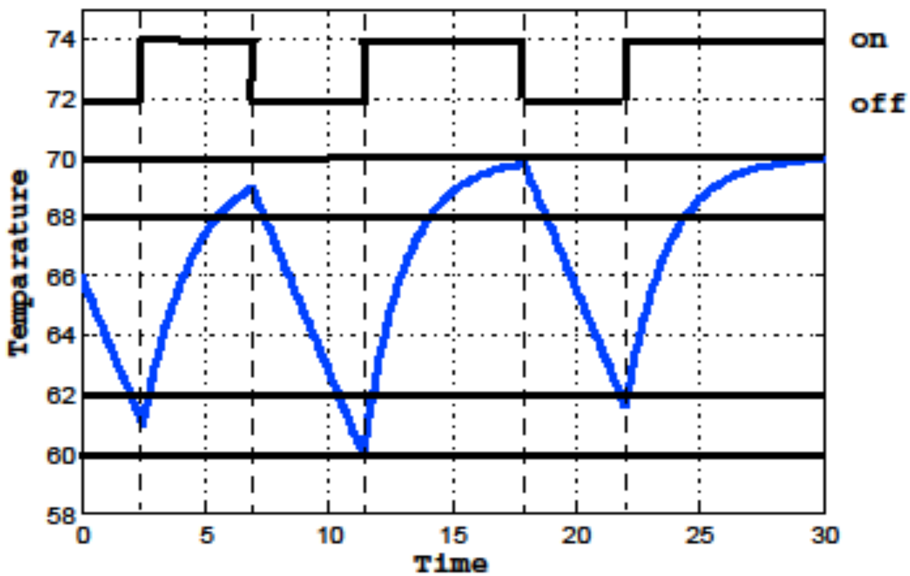
Mode-switch to on enabled when T <= 62, and must happen before T reaches 60

As time elapses in mode on, T increases according to $\mathbf{T}(t) = 70 - (70 - T^*)\, e^{-k1(t-t^*)}$,

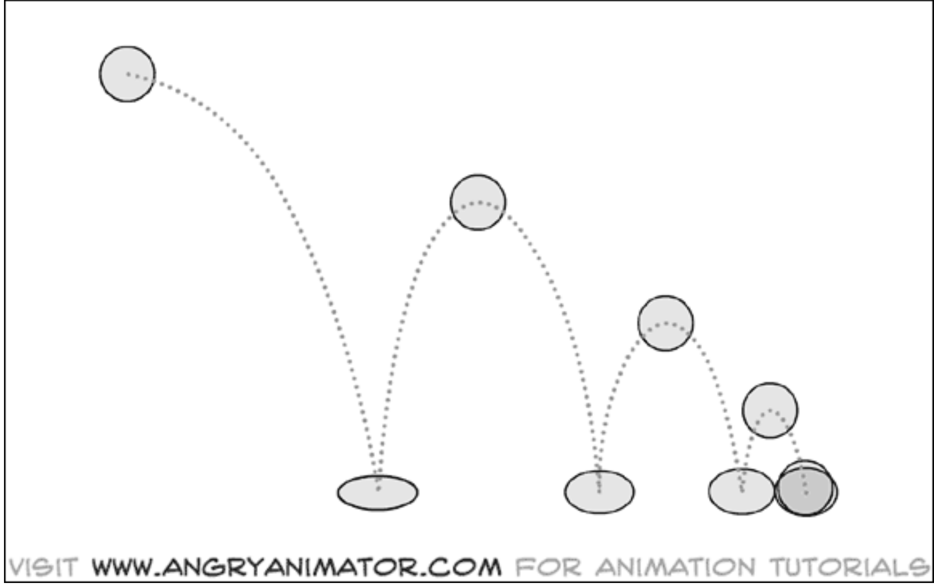
t*, T*: time and temperature upon entry to mode on

Mode-switch to off enabled when T >= 68, and must happen before T reaches 70

Simulation Plot of an Execution

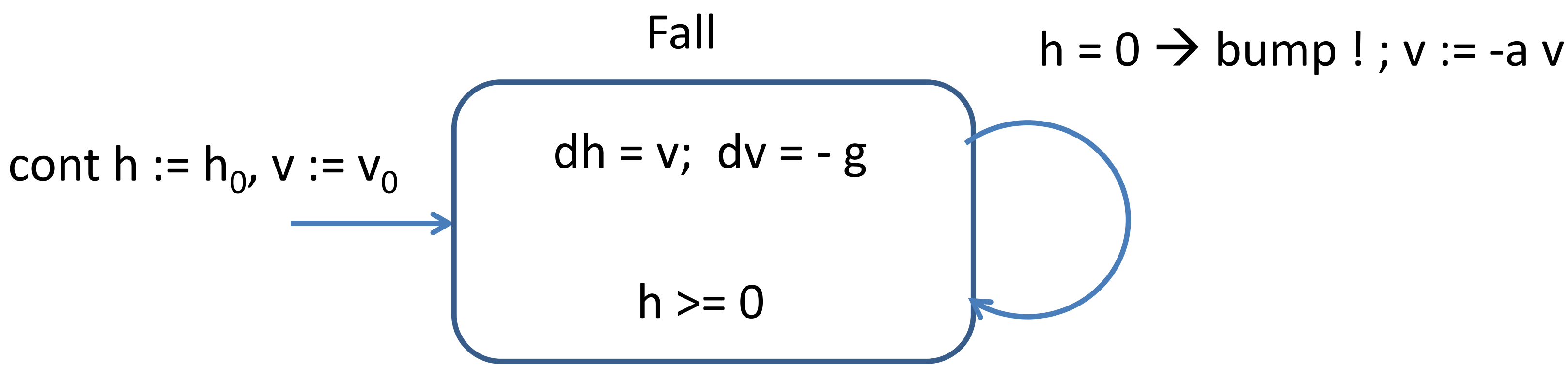


Modeling a Bouncing Ball



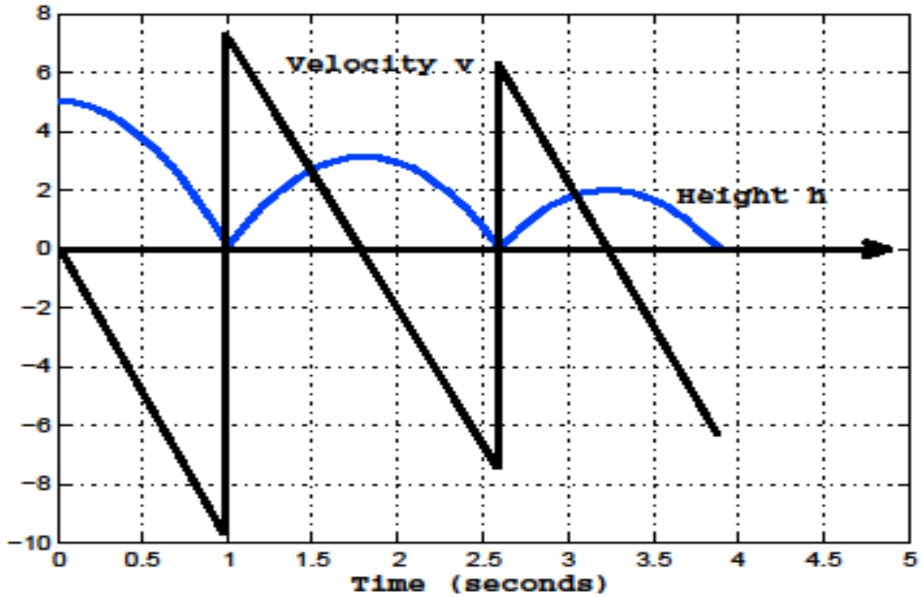
- Ball dropped from an initial height h_0 with an initial velocity v_0
- Velocity changes according to the differential equation $dv/dt = -g$
- When the ball hits the ground, that is, when height $h=0$, velocity changes discretely: $v := -a\,v$, where $0 < a < 1$ is dampening constant
- Modeled as a hybrid system: mix of discrete and continuous behaviors!

Hybrid Process for Bouncing Ball



Execution of the BouncingBall process

$h_0 = 5; v_0 = 0$



Definition of Hybrid Process: Syntax

- A hybrid process HP consists of
  1. An asynchronous process P, where some of the state variables can be type cont (ranging over real numbers)
  2. A continuous-time invariant CI which is a Boolean expression over the state variables of P
  3. For every output variable y of type cont, a Lipschitz-continuous real-valued expression that gives the value of y as a function of state variables and continuous input variables
  4. For every state variable x of type cont, a Lipschitz-continuous real-valued expression that gives the rate of change of x as a function of state variables and continuous input variables

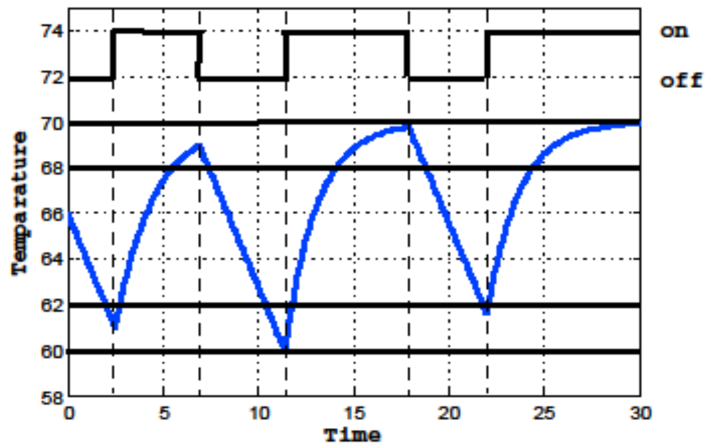
Definition of Hybrid Process: Semantics

- Define inputs, outputs, states, initial states, internal actions, input actions, output actions exactly the same as the asynchronous model
- Timed actions: Given a state s and real-valued time $\delta > 0$ and a continuous input signal **u**(t) that gives values for continuous inputs over time interval $[0, \delta]$, the corresponding state/output signal over $[0, \delta]$ is uniquely defined so that
  1. Initial state **s**(0) equals s
  2. Discrete (i.e. non-cont) state variables stay unchanged
  3. For each continuous output variable y, the value **y**(t) satisfies the corresponding algebraic equation
  4. For each continuous state variable x, the derivative d**x**(t)/dt of the signal satisfies the corresponding differential equation
  5. At all times t in $[0, \delta]$, the signal value **s**(t) satisfies the invariant constraint CI

Executions of Hybrid Processes

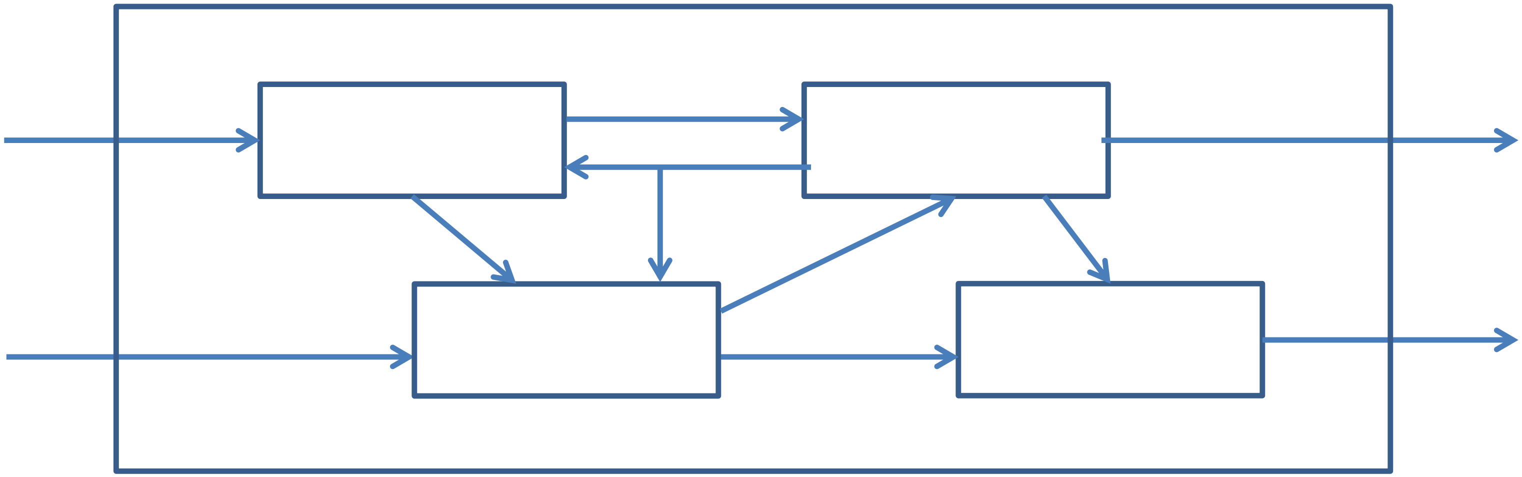
Starting from an initial state, execute either a discrete step (input, or output or internal action) or a timed step (need to solve system of differential equations)

(off, 66) –2.5→ (off, 61) → (on, 61)
–3.7→(on,69.02) → (off, 69.02)
–4.4→ (off, 60.22) → (on, 60.22)
–7.6→ (on, 69.9) → (off, 69.9)
–4.1→(off, 61.7) → (on, 61.7) …



Concepts based on transition systems such as reachable states, safety and liveness requirements, all apply to hybrid systems

Block Diagrams

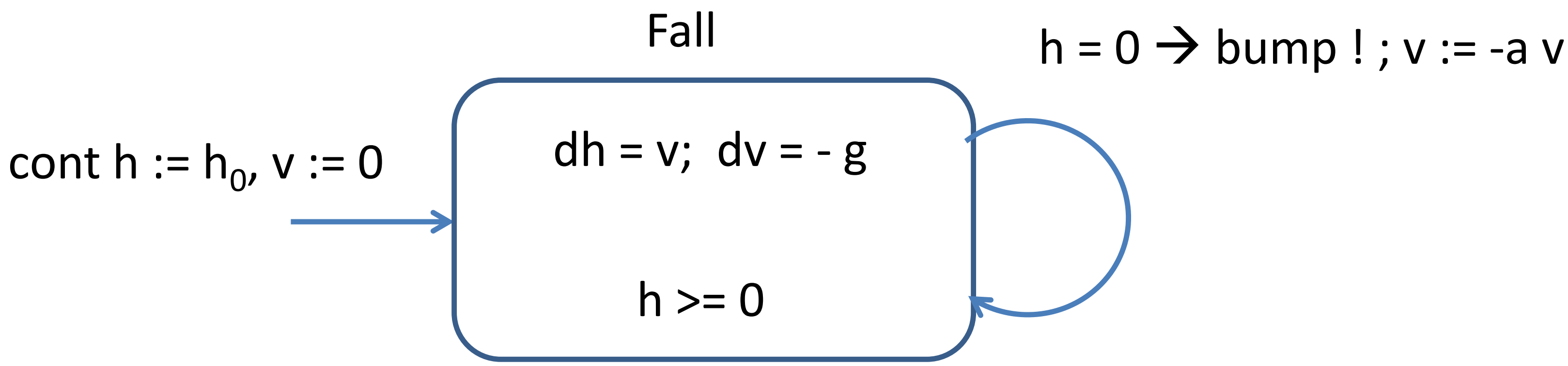


- ❑ Component processes can now be hybrid processes
  - ▪ Need to define Instantiation, Composition, Output Hiding
- ❑ Channels connecting processes of two types
  1. Sender/receiver communication of values during discrete steps as in the asynchronous model
  2. Continuously evolving signals during timed steps as in the model of continuous-time dynamical systems

Summary of the Model

- Generalizes timed model
  - Variables evolving continuously during a timed action can have complex dynamics, clocks being a very special case
- Generalizes continuous-time dynamical systems
  - Discontinuous changes to system state now can be modeled
- Generalizes asynchronous model
  - Distributed/multi-agent systems can be modeled
- Suitable for modeling of cyber-physical systems (in full generality)
- Existing commercial tool support: Modelica, Stateflow/Simulink
  - Simulink now supports Hybrid Automata (hybrid processes described by state machines)
- Challenge for analysis
  - Even if dynamics in individual modes is linear, due to discrete changes, not possible to obtain closed-form solutions, or general theorems about stability

Analysis of Bouncing Ball Model



Change in height during first bounce: $\mathbf{h}(t) = h_0 - gt^2/2$

Time at which first bump occurs: $t_1 = \mathrm{Sqrt}\ (\ 2\ h_0 / g)$

Velocity just before first bump occurs: $- \mathrm{Sqrt}\ (\ 2g\ h_0) = - v_1$

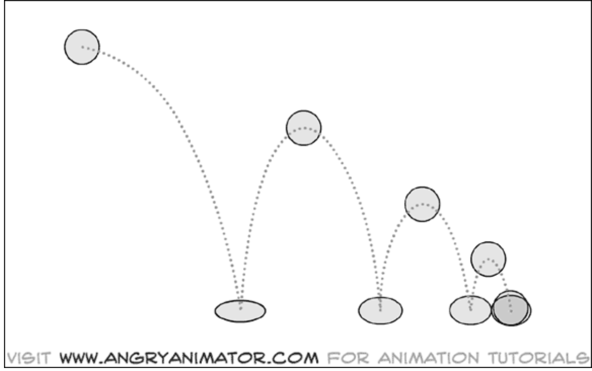
Velocity just after first bump : $v_2 = a\ v_1$

Evolution of height during second bounce: $\mathbf{h}(t) = v_2\, t - gt^2/2$

Time between first and second bump: $t_2 = 2\ v_2 / g$

Velocity just before second bump occurs: $- v_2$ and after second bump $v_3 = a\ v_2$

Modeling a Bouncing Ball



- Velocity after k bumps = $a^k v_1$
- Duration between k-th and following bump $a^k v_1 / g$
- Sum of durations between successive bumps converges to $v_1 (1+a)/(1-a)$
- Infinitely many discrete actions in finite time = Zeno behavior!

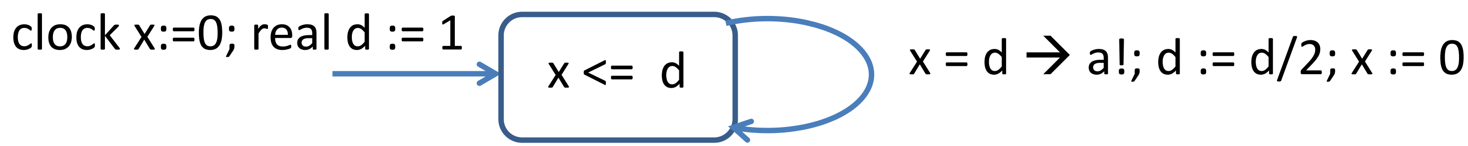
Zeno' Paradox

- Described by Greek philosopher Zeno in context of a race between Achilles and a tortoise
- Tortoise has a head start over Achilles, but is much slower
- In each discrete round, suppose Achilles is d meters behind at the beginning of the round
- During the round, Achilles runs d meters, but by then, tortoise has moved a little bit further
- At the beginning of the next round, Achilles is still behind, by a distance of a.d meters, where a is a fraction $0<a<1$
- By induction, if we repeat this for infinitely many rounds, Achilles will never catch up!
- If the sum of durations between successive discrete actions converges to a constant K, then an execution with infinitely many discrete actions describes behavior only upto time K (and does not tell us the state of the system at time K and beyond)

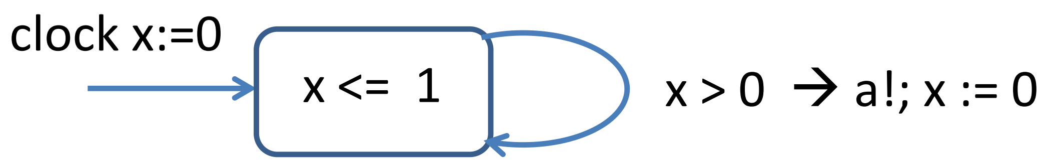
Formalization

- An infinite execution of a hybrid process HP is of the form $s_0 - t_1 \rightarrow s_1 - t_2 \rightarrow s_2 - t_3 \rightarrow s_3$ …, where t_i is duration of i-th step
  - Input/output/internal actions are instantaneous (duration 0)
- An infinite execution is called Zeno if the infinite sum of all the durations is bounded by a constant, and non-Zeno if the sum diverges
- A state s of the process HP is called
  - Zeno if every execution starting in state s is Zeno
  - Non-Zeno if there exists some non-Zeno execution starting in s
- A hybrid process HP is called non-Zeno if every reachable state of HP is non-Zeno
  - At every point during an execution it is possible for time to diverge
- Zeno system: Could end up in a state from which duration between successive steps must get smaller and smaller
- Thermostat: non-Zeno; BouncingBall: Zeno

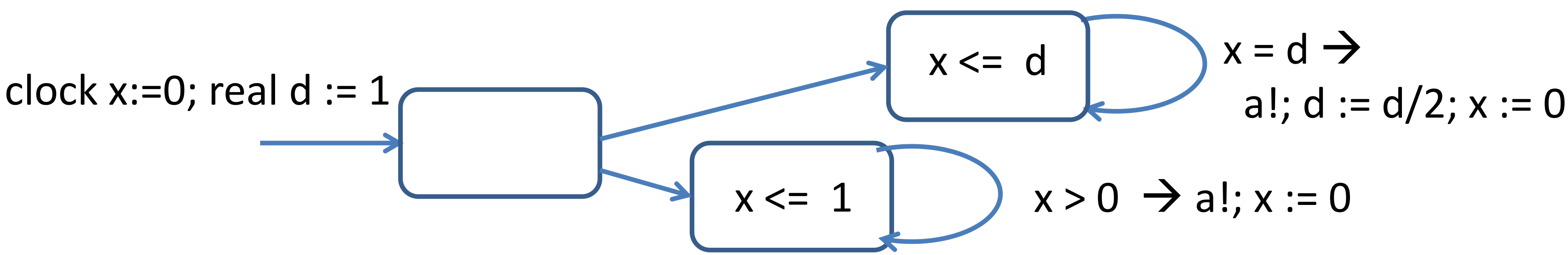
Zeno Vs Non-Zeno



Zeno ! Every possible execution is Zeno



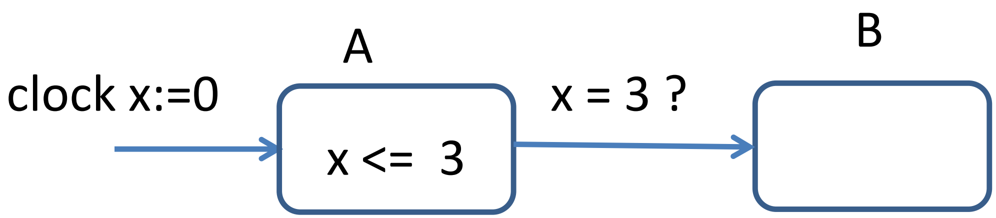
Non-Zeno ! Some executions are Zeno and some are non-Zeno



Zeno ! System may end up in a state from which only Zeno executions are possible

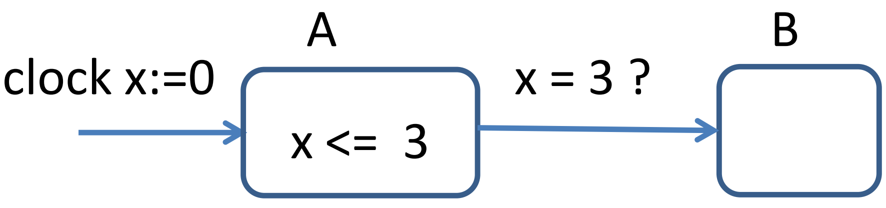
Zeno Processes and Reachability

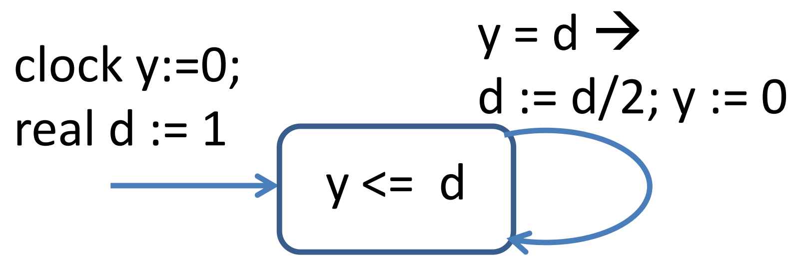
- How does existence of Zeno processes influence analysis?
- Recall: A state s is said to be reachable if there exists a finite execution starting in an initial state and ending in state s
- Safety: A property φ is an invariant if all reachable states satisfy φ



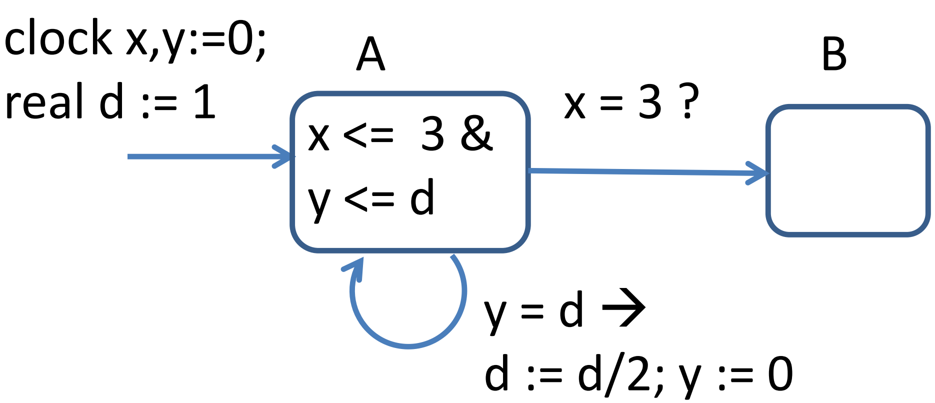
Is mode B reachable ?

Zeno Processes and Reachability



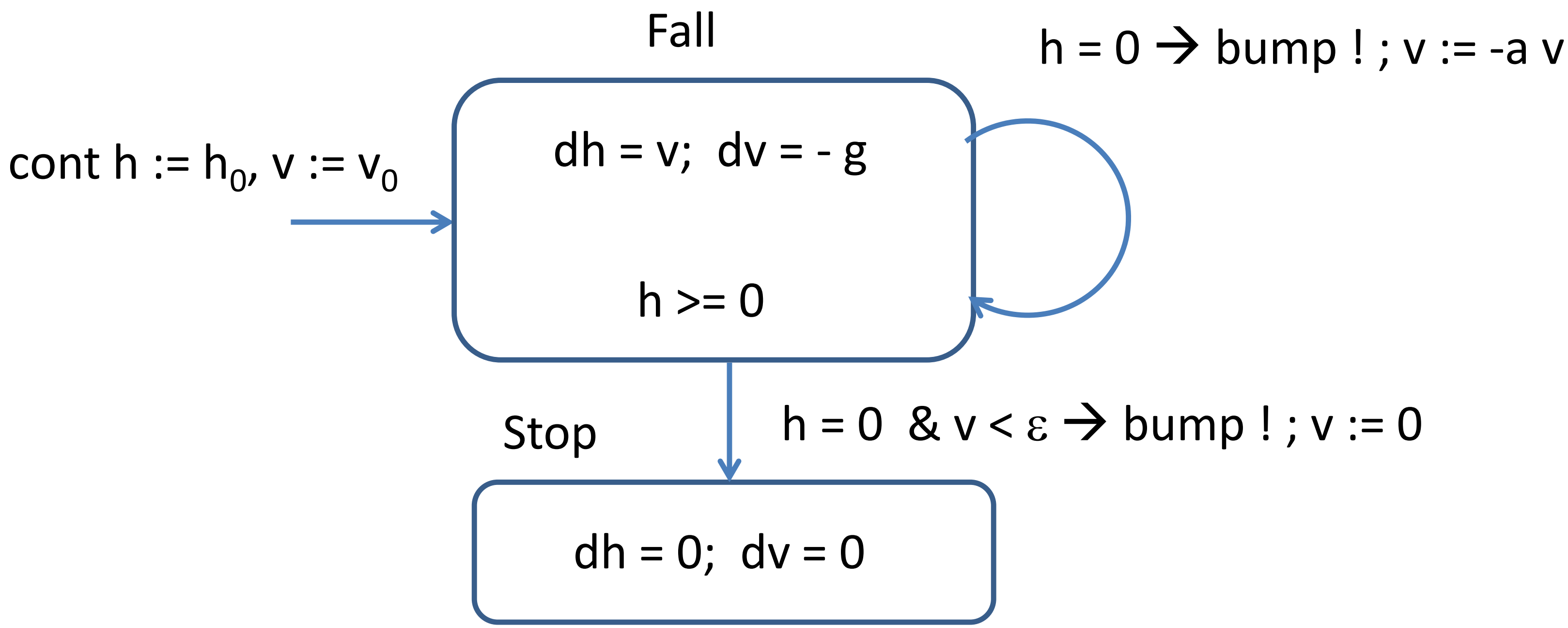


Is mode B reachable ?



Presence of a Zeno process in the system can stop time from advancing, and make states of other processes unreachable !

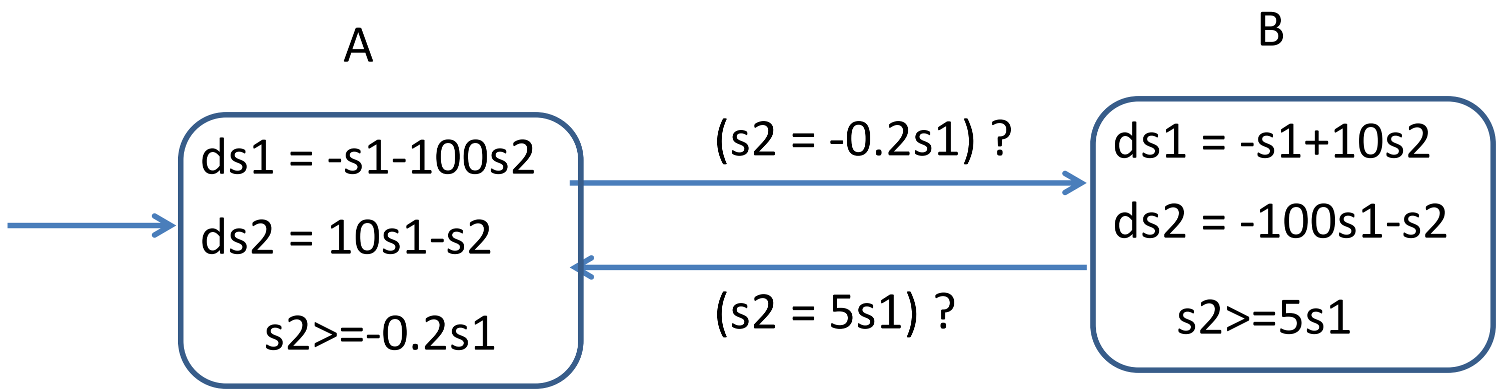
Making Bouncing Ball Non-Zeno



If velocity is too small, stop modeling dynamics accurately

In this model, there is a lower bound on duration between successive bumps

Stability of Hybrid Systems

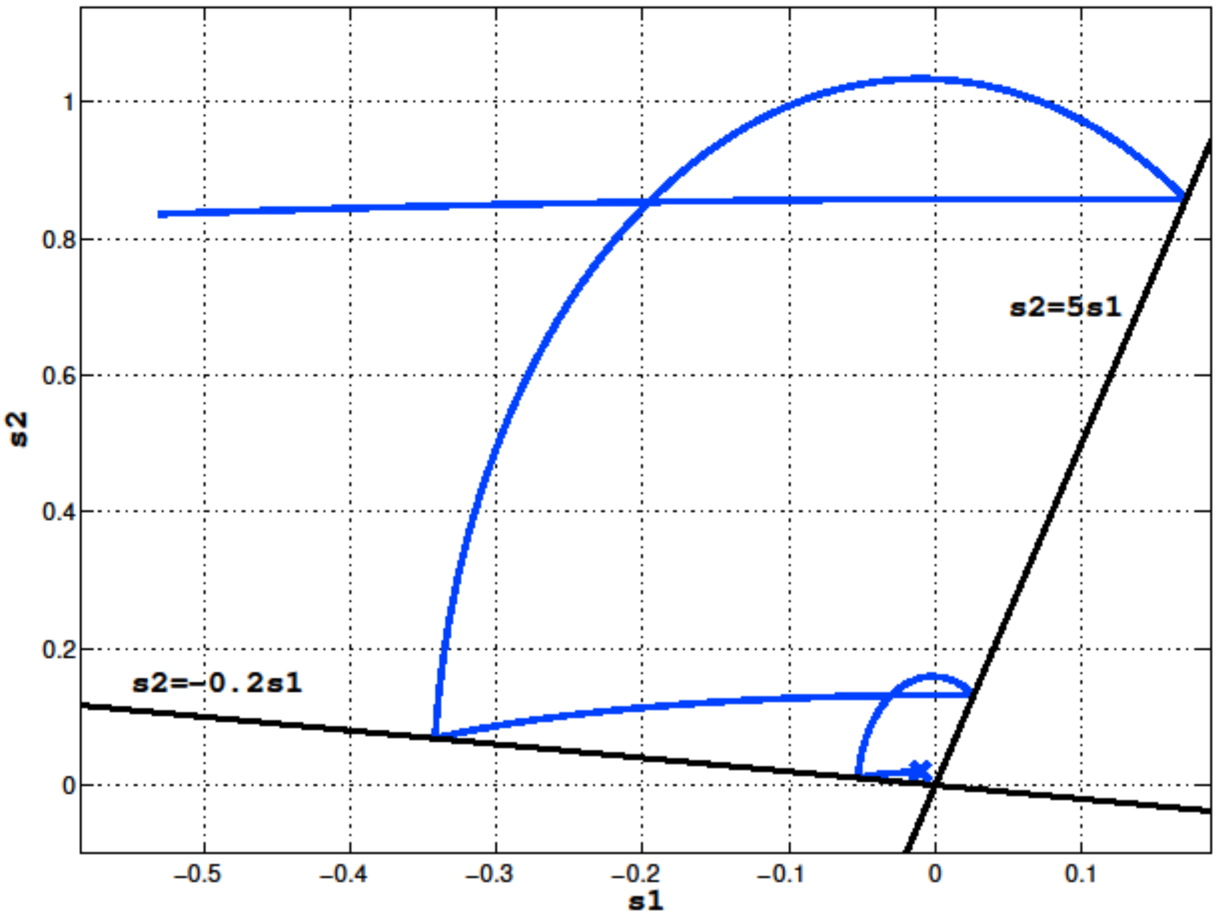


Is the dynamics in mode A stable?

Is the dynamics in mode B stable?

Both modes have stable dynamics, but switching causes instability!

Stability of Hybrid Systems



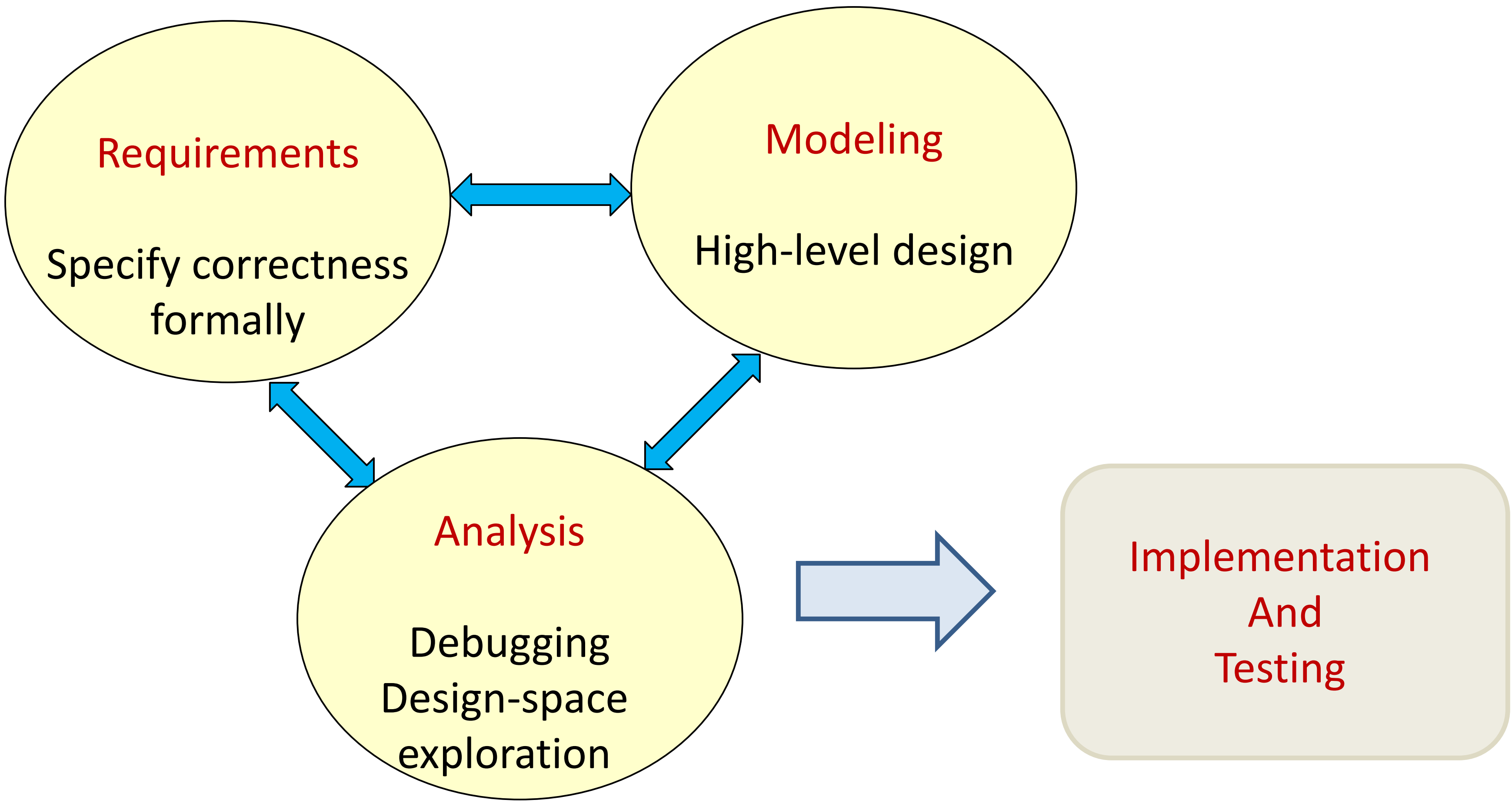
Design and Modeling of Hybrid Systems

- Automated Guided Vehicle
  - Goal: Follow a track as closely as possible
  - Design of mode-switching controller

- Obstacle Avoidance for Robotic System
  - Reach target while avoiding obstacles
  - Augment obstacle estimation via communication

- Multi-hop control network
  - Maintain stability of multiple plants when information flows among sensors, actuators, and computing elements over shared network
  - Design control algorithm in conjunction with scheduling policy for the network

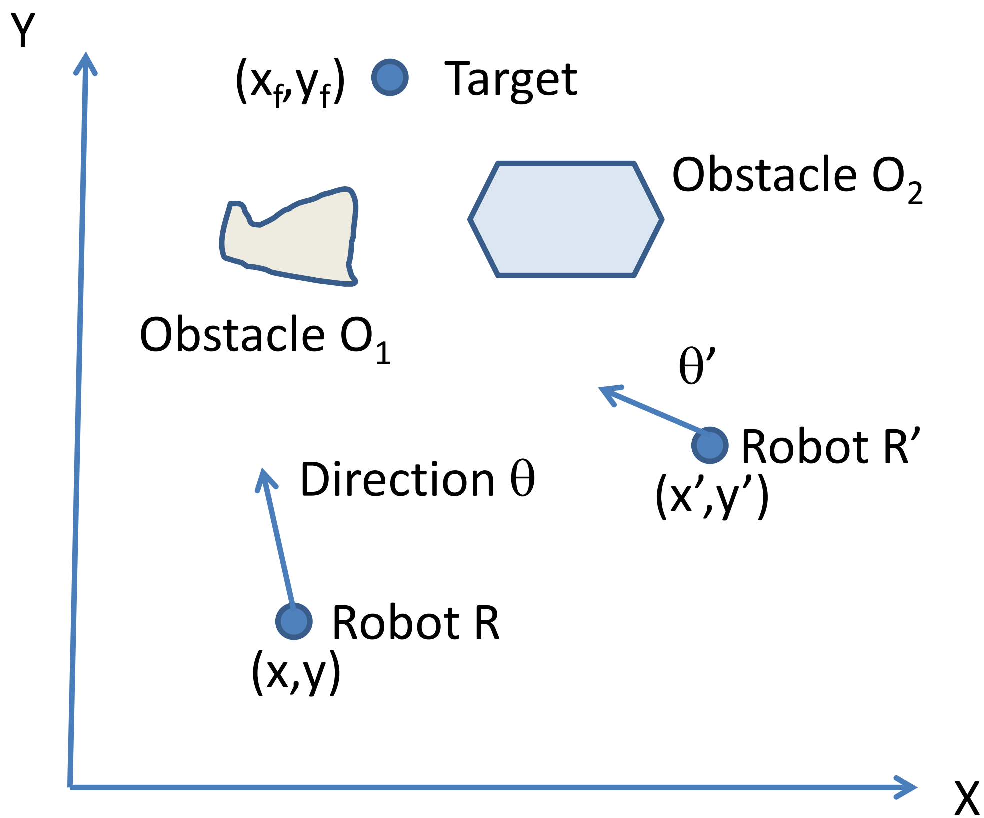
Model-Based Design and Analysis



Multi-Robot Coordination

- Autonomous mobile robots in a room
- Goal of each robot:
  - Reach a target at a known location
  - Avoid obstacles (positions of obstacles not known in advance)
  - Minimize distance travelled
- Cameras and vision processing algorithms allow each robot to estimate obstacle positions
  - Estimates are only approximate, and depend on relative position of obstacles with respect to a robot's position
  - How often should robot update these estimates ?
- Each robot can communicate with others using wireless links
  - How often and what information?
  - How does communication help ?
- High-level motion control (path planning)
  - Decide on speed and direction

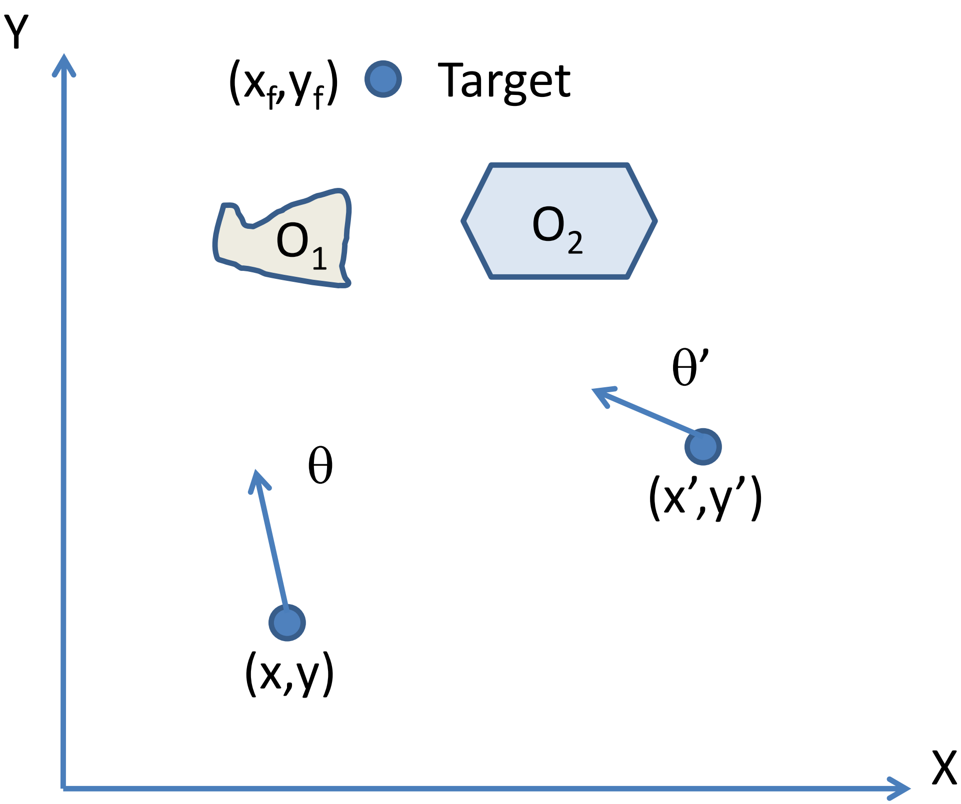
Path Planning with Obstacle Avoidance



Assumptions:

- Two dimensional world
- Point Robots
- Fixed speed v

Path Planning with Obstacle Avoidance



Performance: Reduce distance travelled!

State variables: (x,y); (x',y')

Initialization:

$(x,y) := (x_0, y_0)$;

$(x',y') := (x'_0, y'_0)$

Dynamics:

$dx = v \cos \theta$, $dy = v \sin \theta$;

$dx' = v \cos \theta'$, $dy' = v \sin \theta'$

Safety requirement:

(x,y) is not in O_1 U O_2 &

(x',y') is not in O_1 U O_2

Liveness requirement:

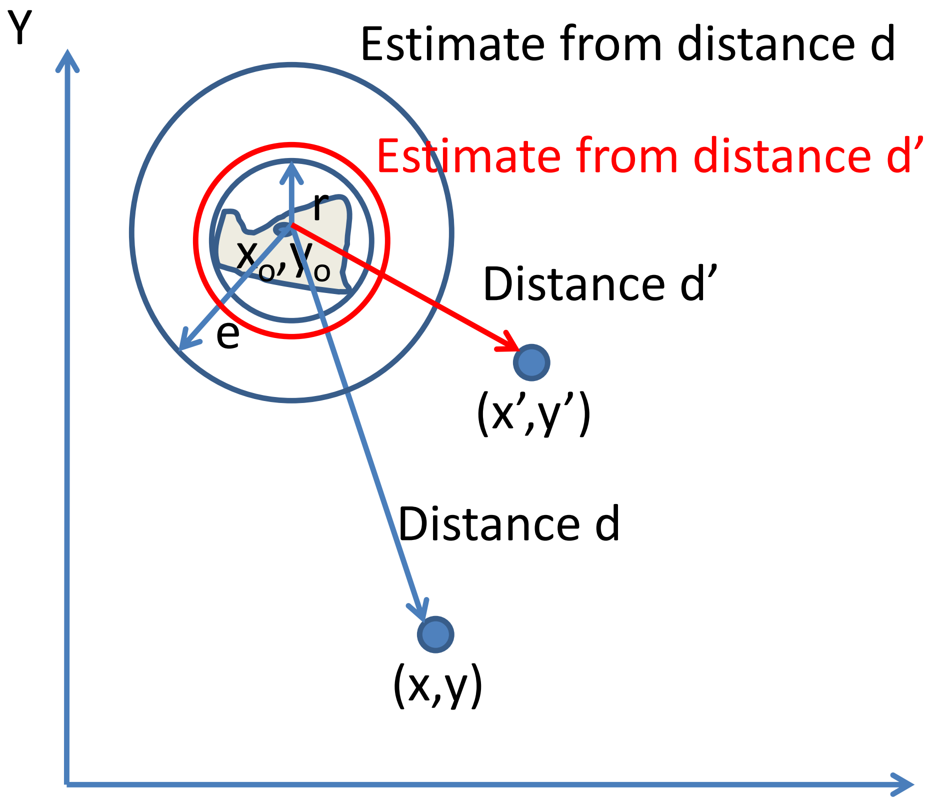
Eventually $(x,y) = (x_f, y_f)$ &

Eventually $(x',y') = (x_f, y_f)$

Modeling Obstacles

- For modeling and analysis, in context of motion planning, need to simplify obstacle shapes and complexity of image processing algorithms
  - Simplicity and abstraction: Key to modeling
- Assume each obstacle/estimate is a circle
  - Can be described by coordinates of center and radius
  - Assumption: Real obstacle is always contained in estimated circle
  - Alternative: ellipses (more accurate)
- Consider an obstacle with center (x_o,y_o) and radius r
  - Radius of smallest circle that envelopes the actual obstacle
- Estimate of the obstacle as computed by a robot using image processing algorithms of a robot
  - A circle with center (x_o,y_o) and radius $e > r$
  - Closer is the robot to the obstacle, better is the estimate
  - Decreases with distance of robot from obstacle, and converges to r

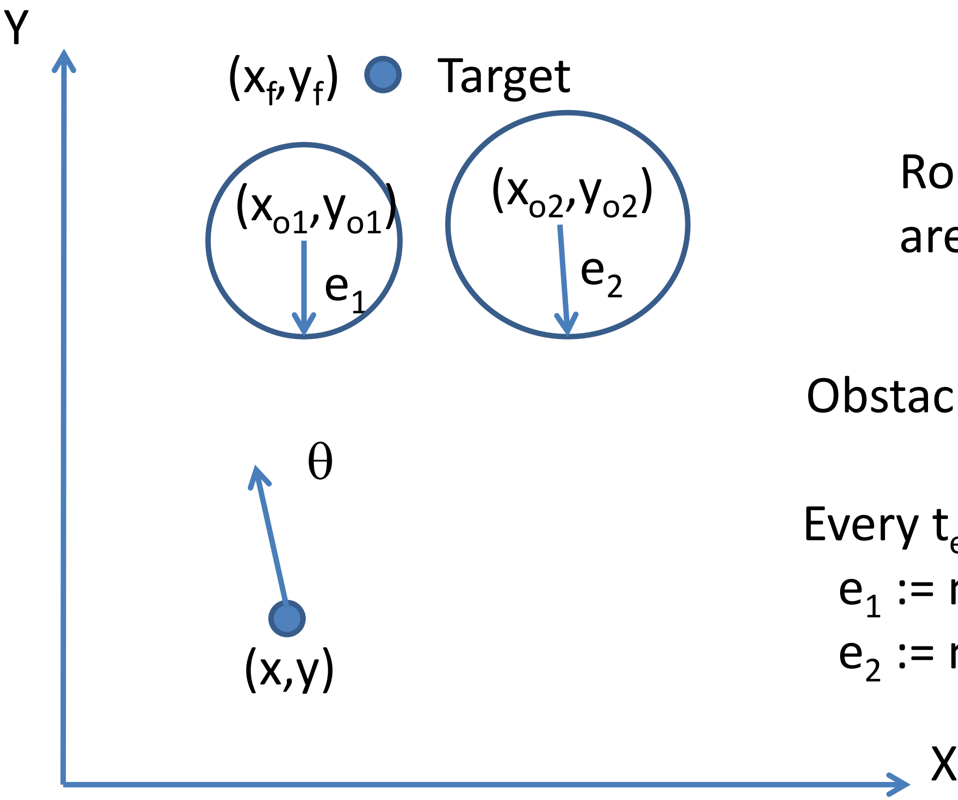
Obstacle Estimation



Estimated radius $e = r + a(d-r)$

$0 < a < 1$ is a constant

Rule for Obstacle Estimation



Robot R maintains radii e_1 and e_2 that are estimates of the obstacles

Obstacle estimation is computationally expensive

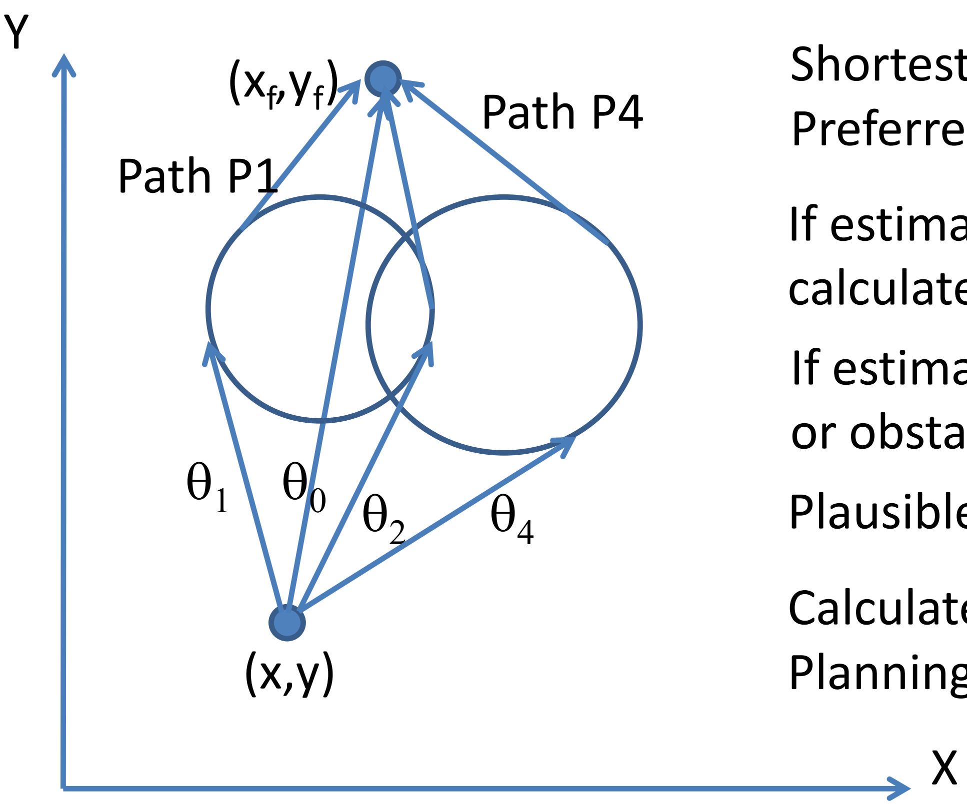
Every t_e seconds, robot executes discrete update:

$e_1 := \min(\ e_1,\ r_1 + a(dist((x,y),\ (x_{o1},y_{o1})) - r_1);$

$e_2 := \min(\ e_2,\ r_2 + a(dist((x,y),\ (x_{o2},y_{o2})) - r_2);$

Computation of robot R' is symmetric

Path Planning



Shortest path: Straight line to target
Preferred direction θ_0

If estimate of obstacle 1 intersects straight path, calculate two paths that are tangents to obstacle

If estimate of obstacle 2 intersects straight path, or obstacle 1, calculate tangent paths

Plausible paths: P1 and P4

Calculate which one is shorter:
Planning algorithm returns either θ_1 or θ_4

Path Planning

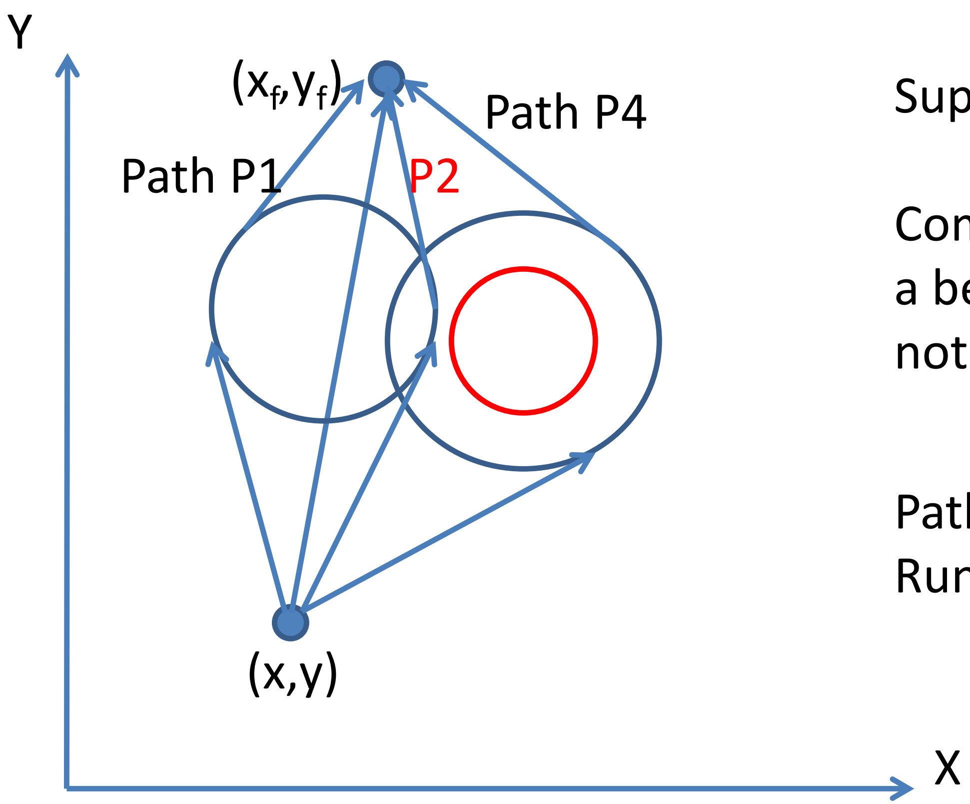
- Function plan with inputs
  - Current position of robot
  - Target position
  - Obstacle 1 position (center and radius estimate)
  - Obstacle 2 position (center and radius estimate)
- Output: Direction for motion
  - Best possible path to target while avoiding obstacles assuming estimates are correct
- Function plan written in C code (can be embedded in model)
- Does it help to execute planning algorithm again as robot moves ?
  - Yes! Estimates may improve suggesting shorter paths
  - Invoke planning algorithm every t_p seconds

Communication

- Each robot has its own estimate of each obstacle
- Robot 2's estimates may be better than Robot 1's own estimates
- Strategy: Every t_c seconds, send your own estimates to the other robot, and receive estimates from the other
- If your own estimates are e_1 and e_2, and receive estimates e'_1 and e'_2, execute

  $e_1 := \min(e_1, e'_1)$; $e_2 := \min(e_2, e'_2)$

Effect of Coordination

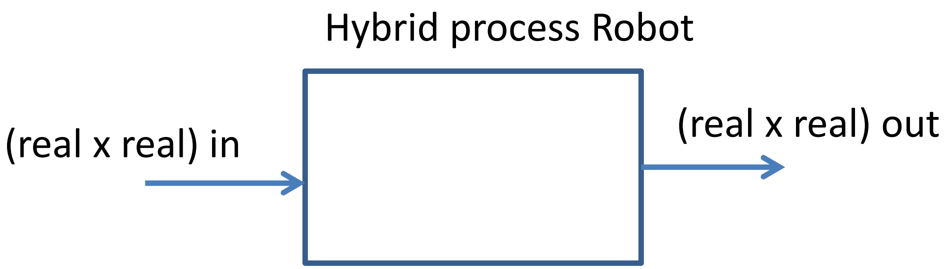


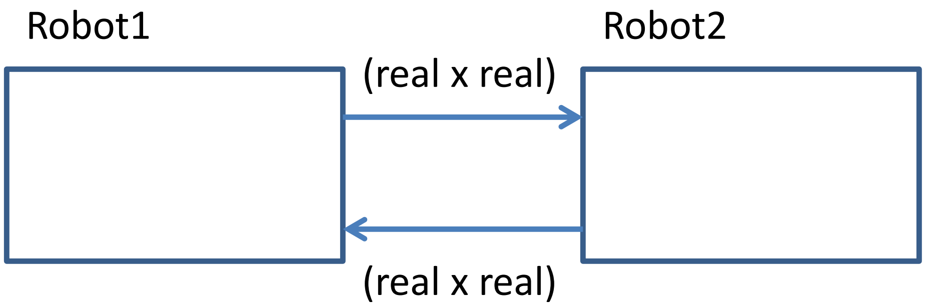
Suppose Path P1 was preferred

Communication with Robot 2 gives a better estimate of obstacle 2, but not for obstacle 1

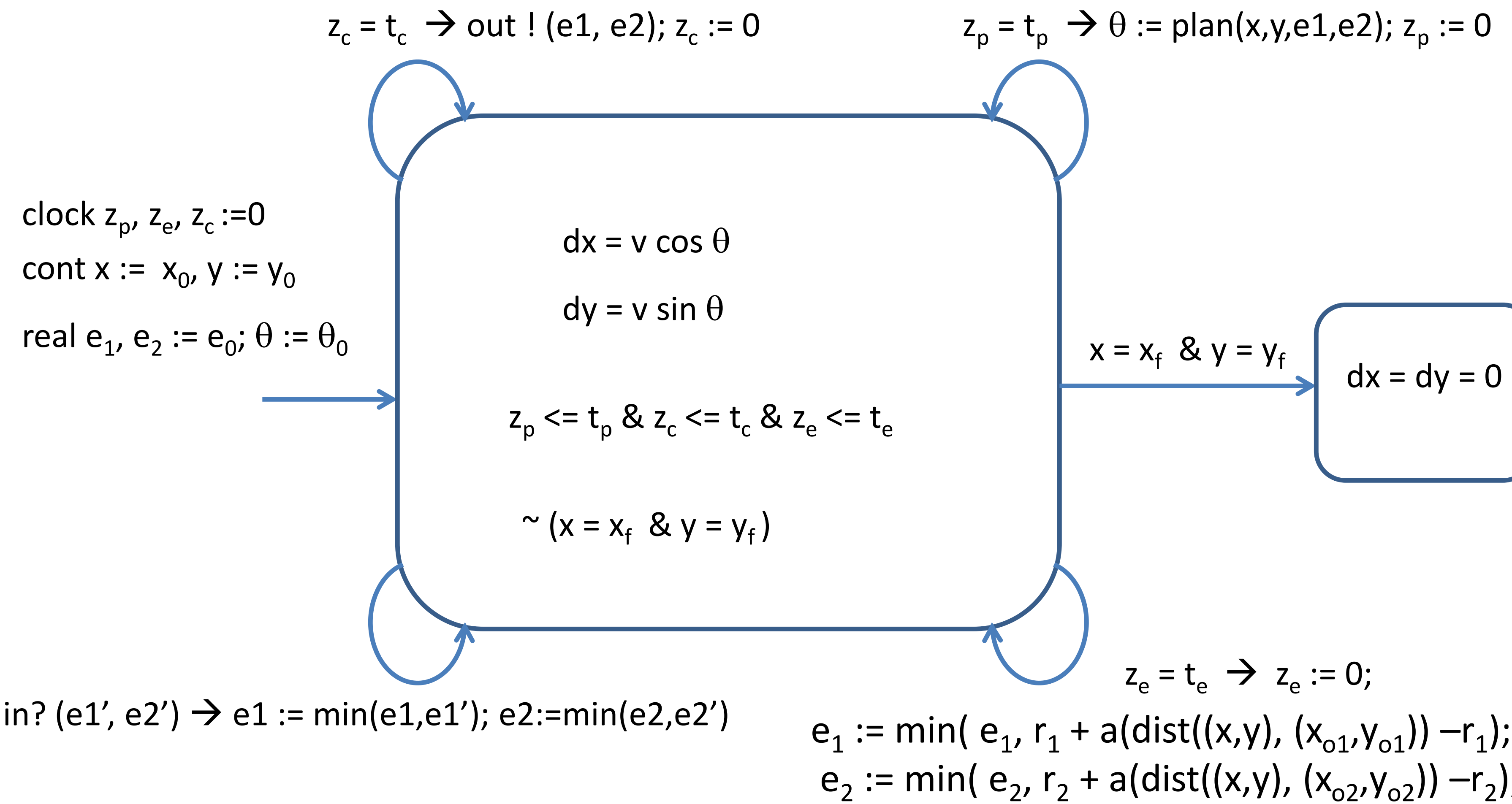
Path P2 is now viable.
Running planner again could choose path P2

System of Robots





Robot Model



$z_c = t_c \rightarrow$ out ! (e1, e2); $z_c := 0$
$z_p = t_p \rightarrow \theta :=$ plan(x,y,e1,e2); $z_p := 0$
clock z_p, z_e, z_c :=0
cont $x := x_0$, $y := y_0$
real $e_1, e_2 := e_0$; $\theta := \theta_0$
$dx = v \cos \theta$
$dy = v \sin \theta$
$z_p \le t_p$ & $z_c \le t_c$ & $z_e \le t_e$
~ ($x = x_f$ & $y = y_f$)
$x = x_f$ & $y = y_f$
$dx = dy = 0$
in? (e1', e2') $\rightarrow$ e1 := min(e1,e1'); e2:=min(e2,e2')
$z_e = t_e \rightarrow z_e := 0$;
$e_1 := \min(e_1, r_1 + a(\mathrm{dist}((x,y), (x_{o1},y_{o1})) - r_1)$;
$e_2 := \min(e_2, r_2 + a(\mathrm{dist}((x,y), (x_{o2},y_{o2})) - r_2)$;

Analysis

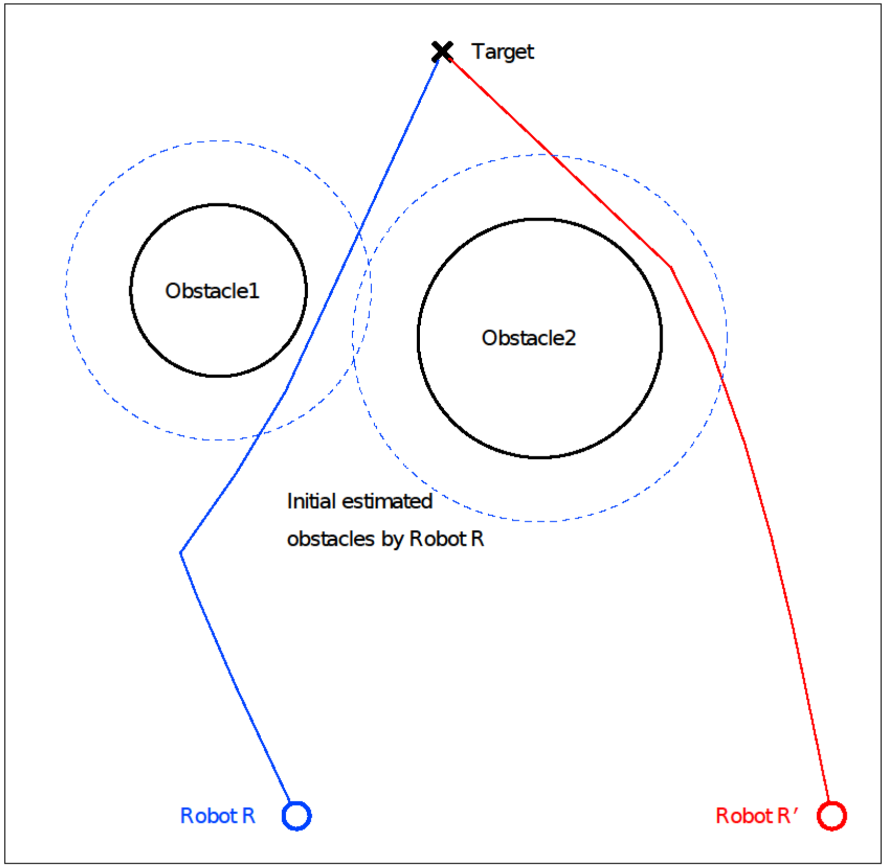
- Key system parameters
  - How often should a robot communicate?
  - How often should a robot execute planning algorithm
  - How often should a robot execute image processing algorithm to update obstacle estimates?

- Design-space exploration: Choose values of t_c, t_p, t_e
  - Reduce distance travelled, but also account for costs of communication/computation

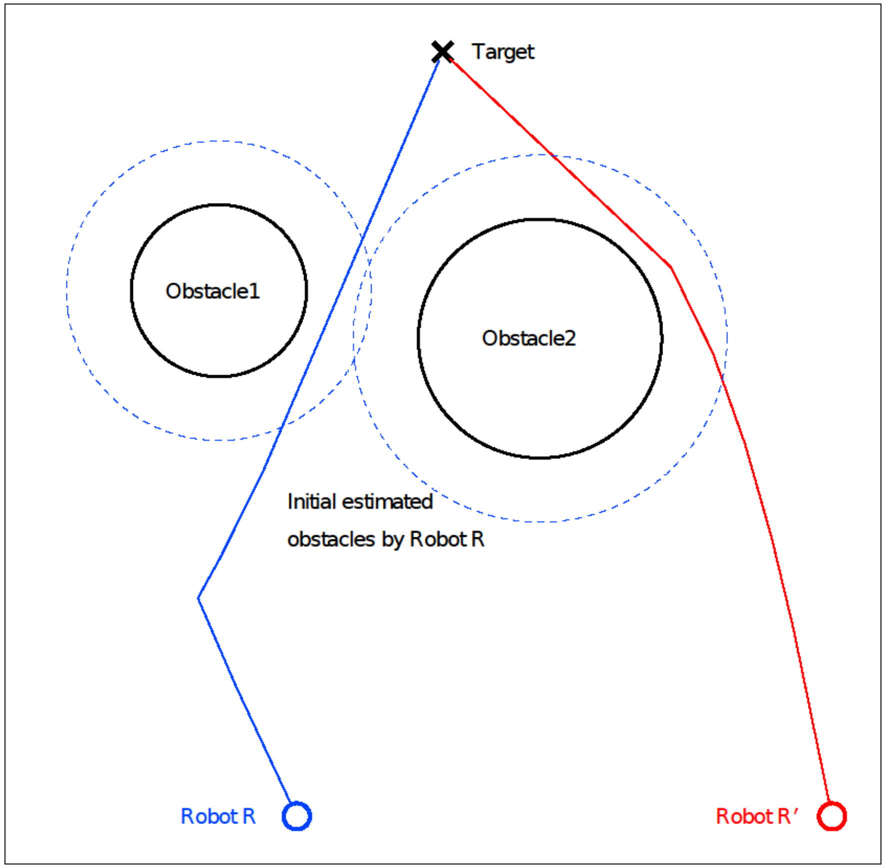
- Symbolic analysis beyond the scope of current tools, so need to run multiple simulations

Illustrative Execution: No Communication

Distance travelled by R = 8.81



Illustrative Execution: No Communication



Distance travelled by R = 8.64

Hybrid Systems Wrap-up

- Integrated modeling of control, communication, and computation
- See section 9.2.3 for modeling of multi-hop control networks
- Not covered: Linear Hybrid Automata (section 9.3)
  - Restricted continuous dynamics and discrete tests/updates
  - Symbolic reachability analysis by representing regions as polyhedra