EECS 294-98: Formal Methods for Engineering Education

Sanjit A. Seshia

EECS, UC Berkeley

Engineering Education (recent past)

- On-campus experience
- Instructor lectures / discusses in class
- Students listen / discuss
  - Some email / forum interaction
- Homework exercises/exams: paper/electronic hand-ins, programming assignments, …
  - Grading mostly manual, some automated testing
- "Hardware" Labs: must be on campus, limited capacity

Engineering Education (today)

- Mostly on-campus experience
  - Some recorded video
- Instructor lectures / discusses in class
- Students listen / discuss
- Online discussion forums (e.g. piazza)
- Homework exercises/exams: paper/electronic hand-ins, programming assignments, …
  - Grading mostly manual, some automated testing
- “Hardware” Labs: must be on campus, limited capacity

Engineering Education
(near future)

- On-campus + MOOCs + blended models
  - Substantial online content (not just video)
- Instructor may lecture / discuss in class
  - Also online
- Students listen / discuss in class + online
  - Online discussion forums standard
- Homeworks/exams: substantially electronic / online
  - Lot more automated grading, peer grading, crowd-grading, …
- “Hardware” Labs: on campus, limited capacity?
  - Effective Virtual Lab Environments

Some Technical Problems in the "New Wave" of Engg. Education

- Automatic Grading
  - Verification / Effective Testing: synthesizing test cases, input stimuli, equivalence checking, etc.
  - Generating feedback
  - Assigning partial / extra credit
- Automatic Exercise Generation
  - Problem generation (to deter cheating, allow for self-paced / customized learning, etc.)
  - Solution generation (supports the above)
- Virtual Laboratory Environments
  - Realistic lab setting (solution that works in virtual lab should work in real lab!)
  - Auto-grading, lab customization, etc.

Formal Methods

Mathematical / algorithmic techniques used to model, verify, design, and maintain computational systems.

Formal Verification

Algorithmic techniques to prove/disprove that a **system** satisfies a given **property** in a specified operating **environment**

Automatic Grading (one part)

Algorithmic techniques to prove/disprove that a student solution satisfies correctness criteria in a specified operating conditions

Formal Specification

Mathematical techniques
for MODELING systems.

Mathematical statement of
Problem to be solved.

Formal Synthesis

Algorithmic techniques
for correct-by-construction SYNTHESIS.

Automatic synthesis of problems/solutions.

Debugging

Algorithmic techniques for providing feedback when a design fails or is non-optimal.

Automatic generation of feedback for student solutions (correct / incorrect).

(the other part of automatic grading)

Algorithmic Formal Methods

- Formal specification
  - Propositional, first-order logic
  - Temporal logic(s)
- Boolean satisfiability (SAT) solving
- Satisfiability modulo theories (SMT) solving
- Automatic theorem proving
- Model checking
- Program verification and synthesis based on the above

Point to Ponder

- Is Auto-Grading exactly the same problem as Formal Verification for “industry designs”?

Three Sample Efforts

- Auto-grading for Python programming course at MIT
- Auto-grading for Automata Theory course at U.Penn.
- Exercise generation / Auto-grading for Embedded Systems course/lab at Berkeley

MIT project: Correction Model for Programming Assignments

Array Index: v[a] -> v[{a+1, a-1}]

Increment: v++ -> { ++v, v--, --v }

Conditional: a op b -> a ops b
where ops = { <, >, <=, >=, ==, != }

Initialization: v=n -> v={n+1, n-1, 0}

Return Value: return v -> return ?v

Key Idea: Finding minimal set of such changes that convert a buggy solution into a correct one is phrased as a synthesis problem.

PLDI 2013: Rishabh Singh, Sumit Gulwani, Armando Solar-Lezama

Feedback Synthesis: Array Reverse

```
using System;
public class Program {
  public static int[] Puzzle(int[] a) {

    int [] b= new int[a.Length];
    for (int i = 0; i < a.Length; i++)
      {
      b[a.Length-i]=a[i-1];
    }
    return b;

  }
}
```

i = 1

i <= a.Length

```
using System;
public class Program {
  public static int[] Puzzle(int[] a) {
    int front, back, temp;
    front = 0;
    back = a.Length-1;
    temp = a[back];
    while (front > back)
    { a[back] = a[front];
      a[front] = temp;
      ++back;
      ++front;
      temp = a[back];
    }
    return a;
  }
}
```

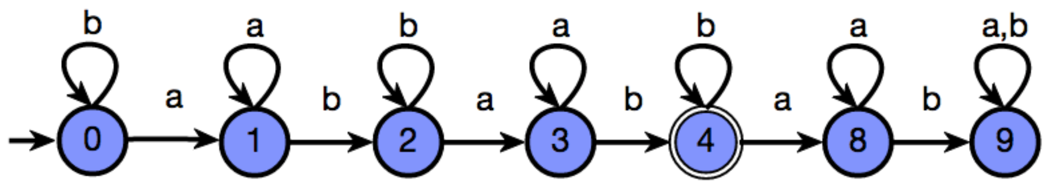
front <= back

--back

http://sketch1.csail.mit.edu/python-autofeedback/

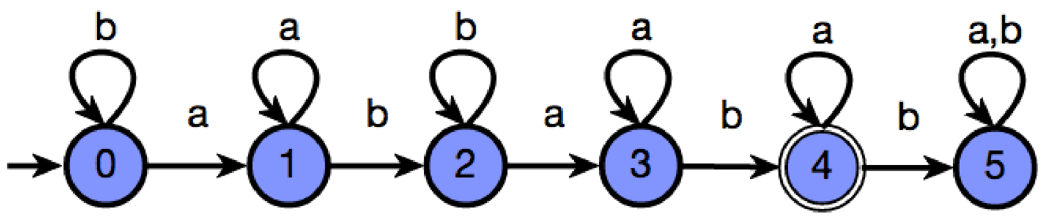
Feedback Synthesis: Finite State Automata

Draw a DFA that accepts: { *s* | '*ab*' appears in *s* exactly *2* times }



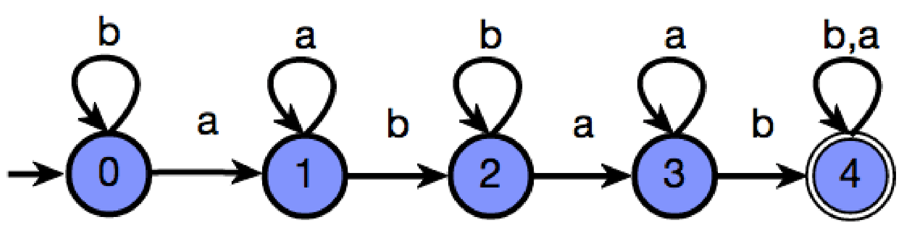
Attempt 1

Grade: 9/10
Feedback: One more state should be made final
Based on Edit-distance to a correct solution



Attempt 2

Grade: 6/10
Feedback: The DFA is incorrect on the string '*ababb*'
Based on Counterexample



Attempt 3

Grade: 5/10
Feedback: The DFA accepts {*s* | '*ab*' appears in *s* at least *2* times}
Based on Edit-distance to the problem description

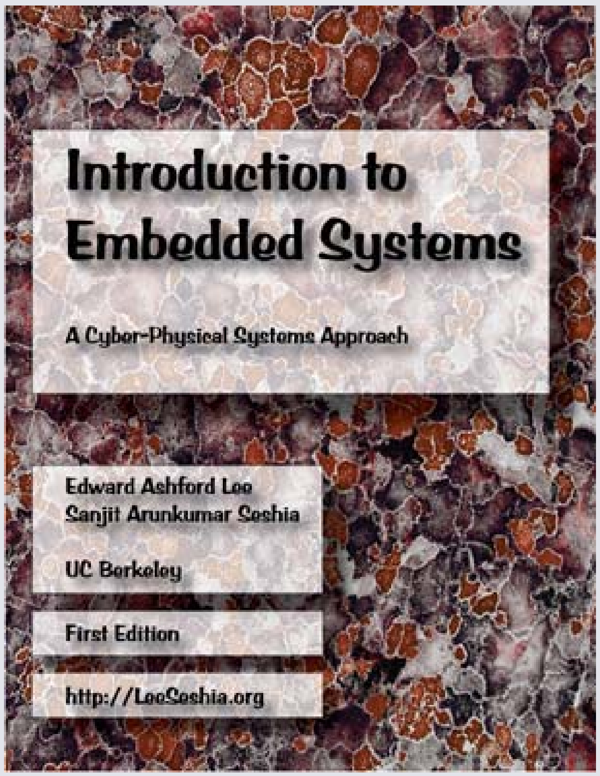
IJCAI 2013: Alur, Loris d'Antoni, Gulwani, Dileep Kini, Viswanathan

Primary Motivation: Embedded Systems course

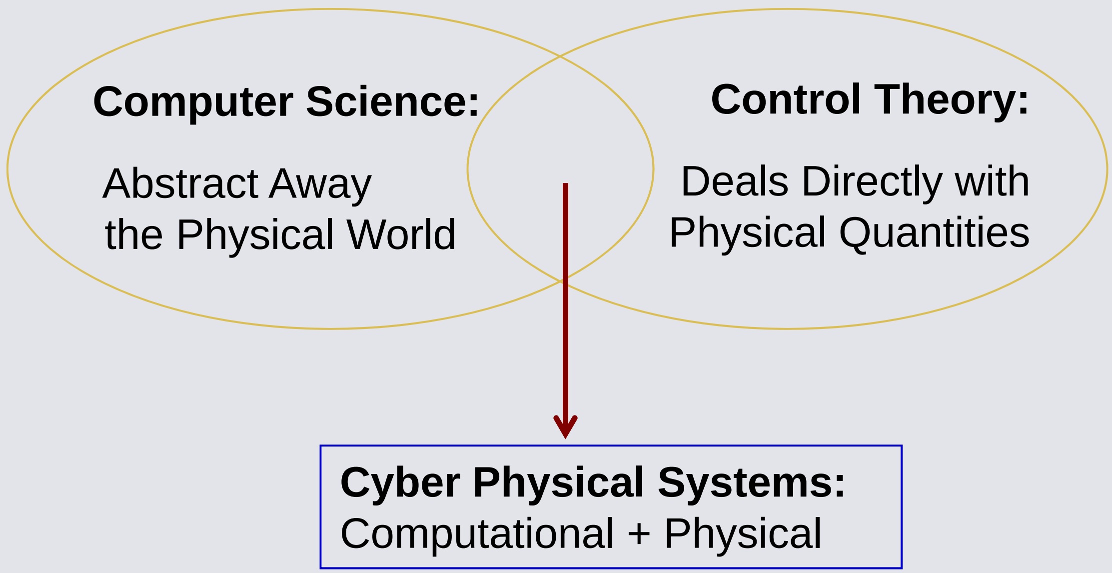
- Context: Introduction to Embedded Systems
  - Undergraduate capstone design course at Berkeley
- Automating Exercise Generation
  - Generating problems and solutions
- Virtual Laboratory
  - Taking the lab experience online
- Representative of “Lab-based” Courses
  - Robotics, circuits, control systems, …

EECS 149: Introduction to Embedded Systems
UC Berkeley

This course introduces the *modeling, design and analysis* of *computational systems that interact with physical processes*.



http://leeseshia.org/



On-campus course gets somewhat diverse enrollment (EE/CS, ME, CE, …)

Traditional View of Embedded Systems

- Tend to be special-purpose.

Embedded computer with a special-purpose processor

Traditional View of Embedded Systems

- Interfacing to sensors and actuators.

Cypres II automatic activation device:

1. embedded microcontroller
2. barometric pressure sensor
3. pyrotechnic cutter actuator

It deploys a reserve parachute for a skydiver who is unaware or unconscious of altitude.

Traditional View of Embedded Systems

- Subject to resource constraints: memory, time, energy,…

Our View of Embedded Systems

We hold that embedded systems should be:

- characterized by **interactions with the physical world**, not resource constraints

- introduced through **formal modeling, design and analysis**, not ad-hoc engineering practices

Motivating Example of a Cyber-Physical System
(see Ch 1 in book)

STARMAC quadrotor aircraft (Tomlin, et al.)

- Introductory Video: http://www.youtube.com/watch?v=rJ9r2orcaYo
- Back-Flip Manuever: http://www.youtube.com/watch?v=iD3QgGpzzIM

Modeling:
- Flight dynamics (ch2)
- Modes of operation (ch3)
- Transitions between modes (ch4)
- Composition of behaviors (ch5)
- Multi-vehicle interaction (ch6)

Design:
- Processors (ch7)
- Memory system (ch8)
- Sensor interfacing (ch9)
- Concurrent software (ch10)
- Real-time scheduling (ch11)

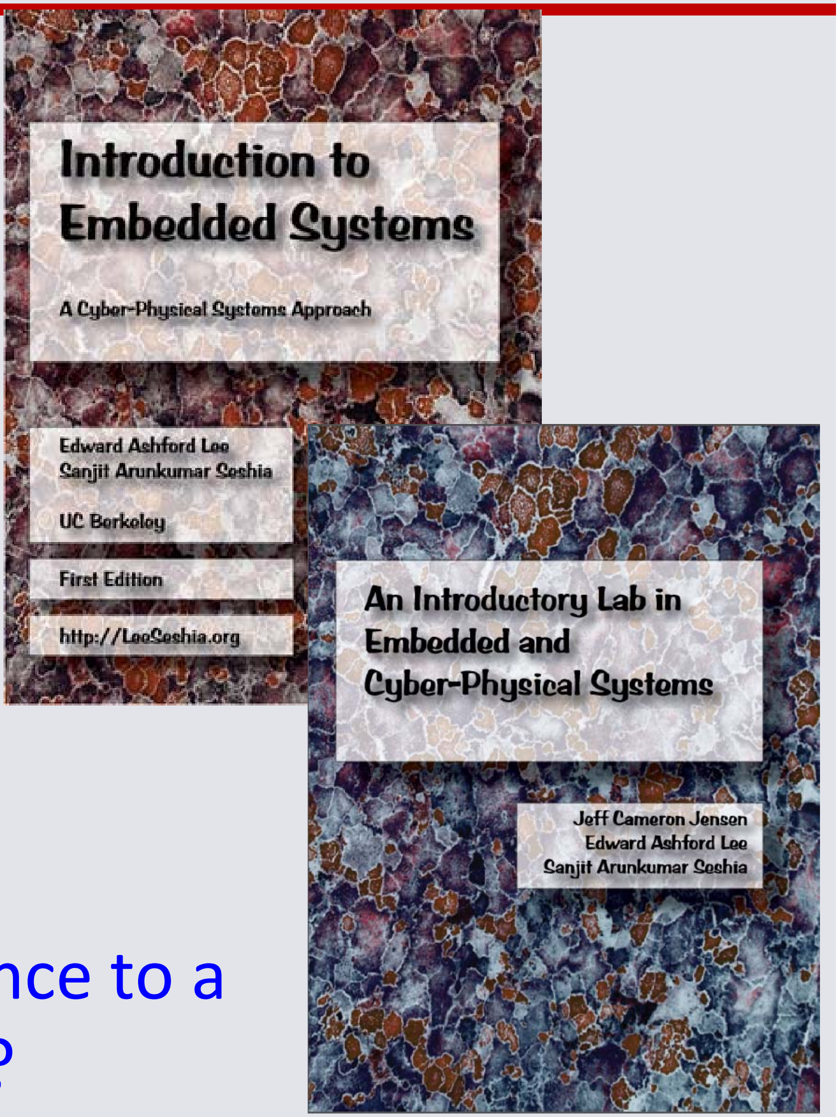
Analysis:
- Specifying safe behavior (ch12)
- Achieving safe behavior (ch13)
- Verifying safe behavior (ch14)
- Guaranteeing timeliness (ch15)

Also in the course: Security and Networking

The Core Learning Experience: Exercises and Labs

- Textbook Exercises:
  - High-level modeling with FSMs, ODEs, temporal logic, etc.
  - Programming in various languages (C, LabVIEW, etc.)
  - Algorithm design and analysis (scheduling, verification, etc.)
- Laboratory (6 weeks)
- Capstone design project (12 weeks)

➢ How to extend this experience to a MOOC version of EECS 149?



The MOOC Challenge: Bringing Automation to Two Course Components

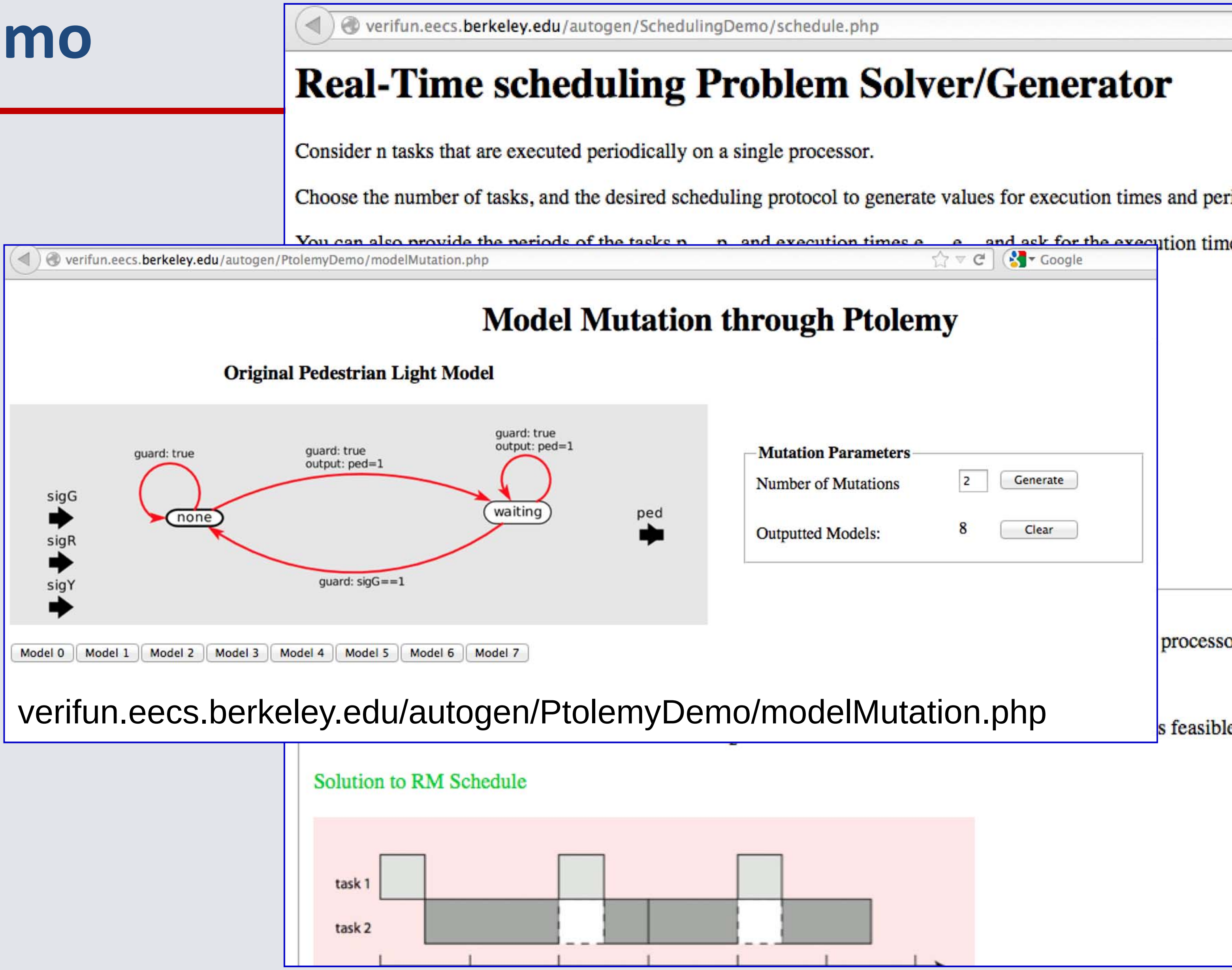
- "Theory" Component: Homeworks and Exams
  - Generating new problems (e.g., "similar difficulty" to existing problems, test range of concepts)
  - Generating solutions
  - Grading / Tutoring

  → *Exercise Creation & Grading Toolkit*

- Lab Component: Structured Labs and Open-Ended Projects
  - Mix of software and hardware design
  - Challenge: how do we give students in an online course offering a similar design experience?

  → *Lab Creation & Grading Toolkit*

demo

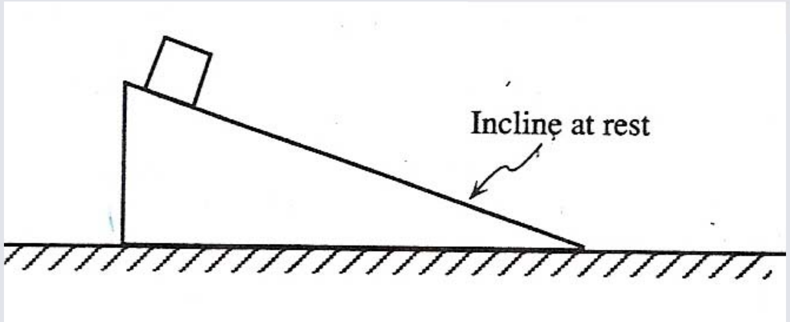


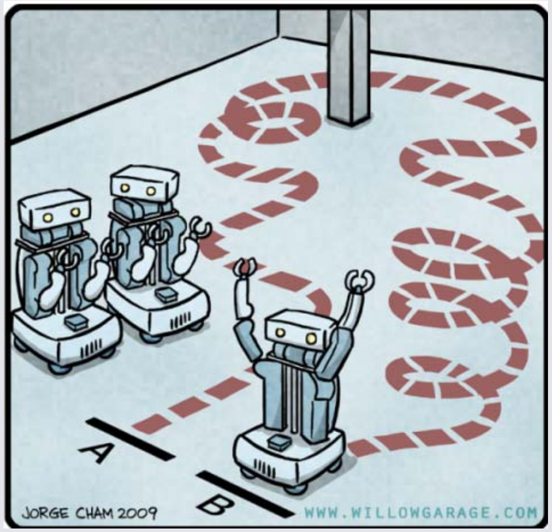
verifun.eecs.berkeley.edu/autogen/PtolemyDemo/modelMutation.php

verifun.eecs.berkeley.edu/autogen/SchedulingDemo/schedule.php

iRobot Hill Climb

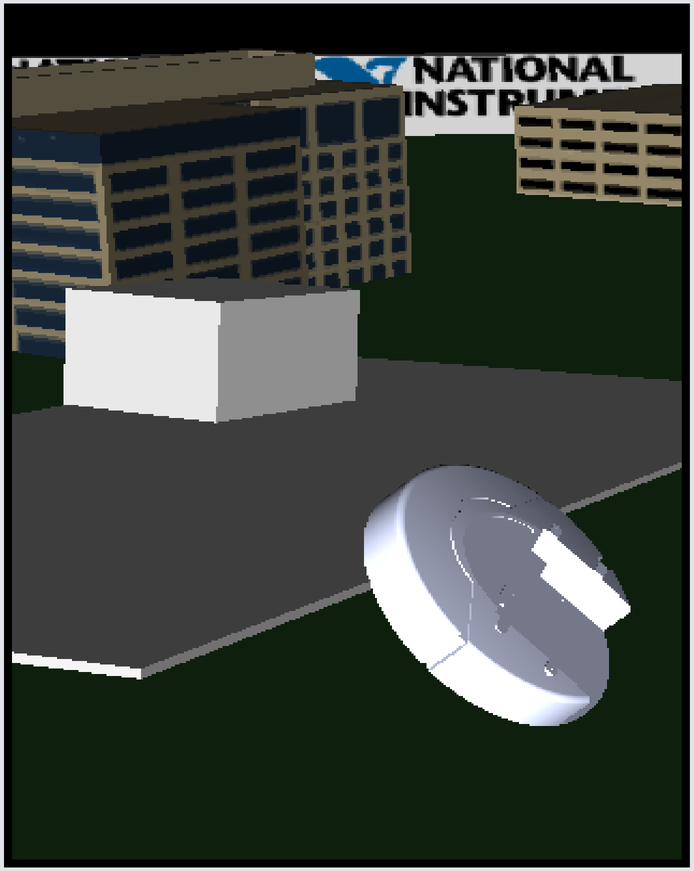
Modify the iRobot Create to autonomously navigate to the top of an incline, avoiding cliffs and obstacles along the way.





Lab: Robot Hill Climb → Virtual World

- Students design Statecharts controller to climb a hill
- ODE simulation of robot, software, & environment
- Statechart deploys to real robot without modification



Simulation of a not-yet-perfect

Videos / Demos

Automatic Grading of Lab Exercises: The How

- Specify desired properties (e.g. end goal for hill climb, waypoints)
  - Temporal logic / monitor automata
- Run-time Verification
  - Check whether solution is correct / wrong
- Feedback generation
  - Leverage simulation/property-based error localization methods [Li & Seshia, DAC'10, RV '12]
- Use of Simulation-based Falsification tools
  - Tools such as S-Taliro, Breach [Donze et al., HSCC '13], …
- Other considerations: Local vs Global, Detecting cheating, etc.

Class Introductions

Please introduce yourselves

-- state name and research interests/areas and what you'd like to get out of this class

(Programming Systems, Computer Security, CAD, Embedded Systems, Synthetic Biology, Control Theory, etc.)

Course Logistics

- Check out the webpage:

  www.eecs.berkeley.edu/~sseshia/fmee

- Tentative class schedule is up
  - Suggested project topics will be posted by next week

Grading

- Paper discussions / class participation (30%)
  - Most of the course
  - Each student signs up to present a paper and the rest of us help dissect it
- **Project (70%)**
  - Project proposal due mid Feb. (date TBA)
  - Culminates in final presentation + tool (+ written paper)
  - *~50% of past projects in 219C led to conference papers!*
  - *Looking forward to actual usable tools from this course!*

Misc.

- Office hours: MW 1:30 – 2:30, and by appointment
- Pre-requisites: basic knowledge of algorithms, data structures, mathematics
  - Berkeley EECS lower-division courses should cover it