ECE/CS 584

Verification of Embedded & Cyber-physical Systems

Lecture 1: Introduction to this course

Prof. Huan Zhang

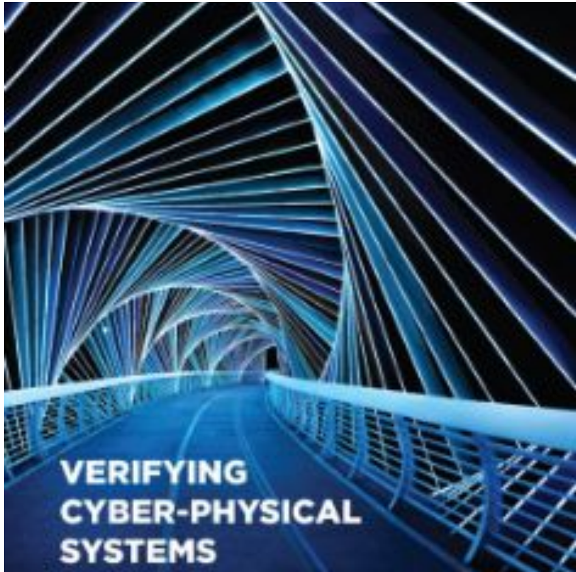
Office: CSL 262

huan@huan-zhang.com

Welcome to Spring 2025
ECE/CS 584

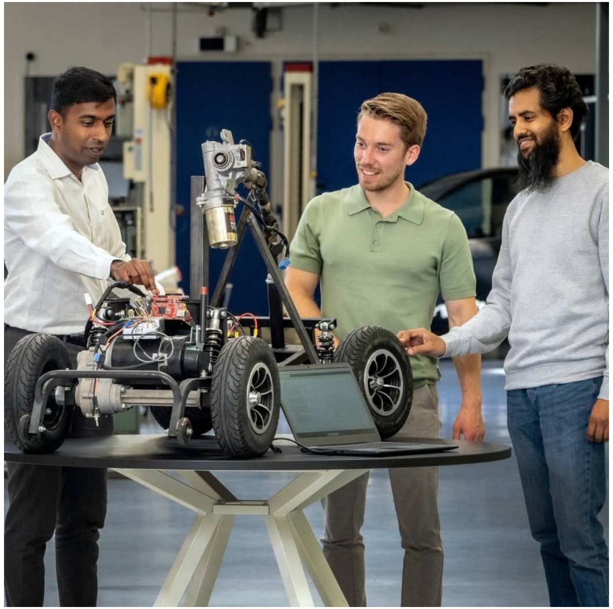
Verification of Embedded & **Cyber-physical** Systems

What is this course about?



What is this course about?

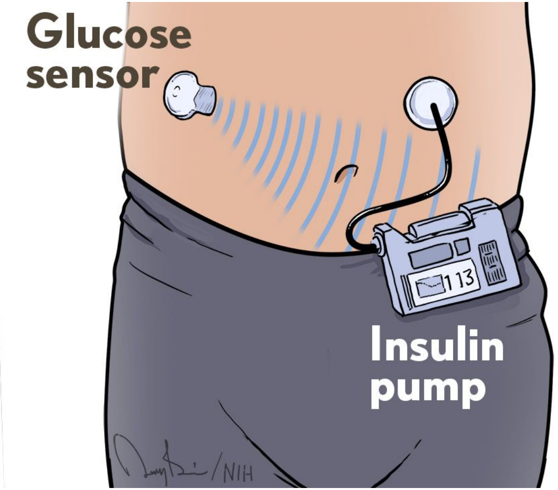
Verification of Embedded & **Cyber-physical** Systems



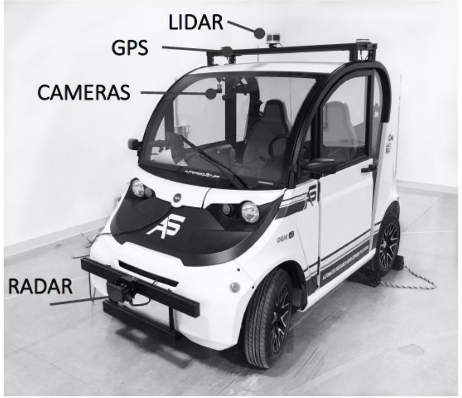
What are cyber-physical systems (CPS)?

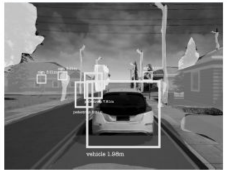
A computer system monitoring or controlling a physical process.

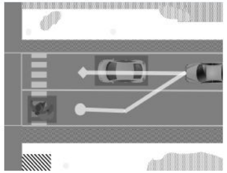
Examples: a drone for package delivery, control system for a smart electric grid, insulin pump for blood glucose control, …

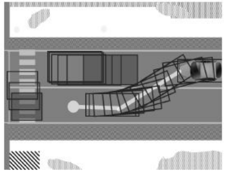


Autonomous vehicle: An example CPS









Sensing

Physics-based models of cameras, LIDAR, radar, GPS, and so on.

Perception

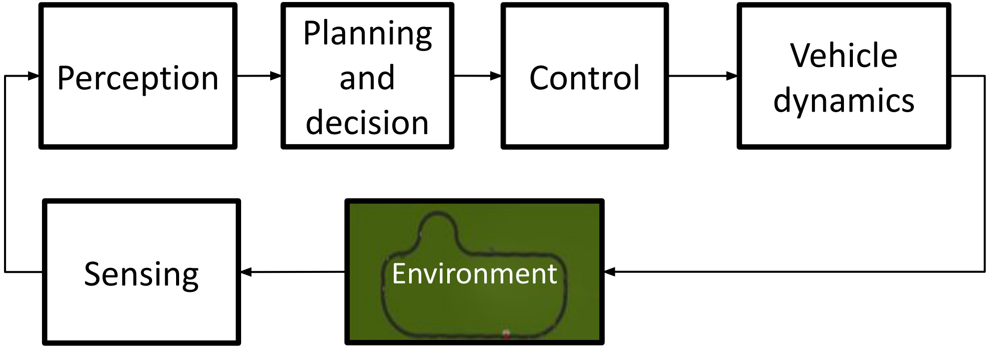
Programs for object tracking, scene understanding, and so on.

Decisions and planning

Programs and multi-agent models of pedestrians, cars, and so on.

Control

Dynamical models of vehicle engine, powertrain, steering, tires, and so on.

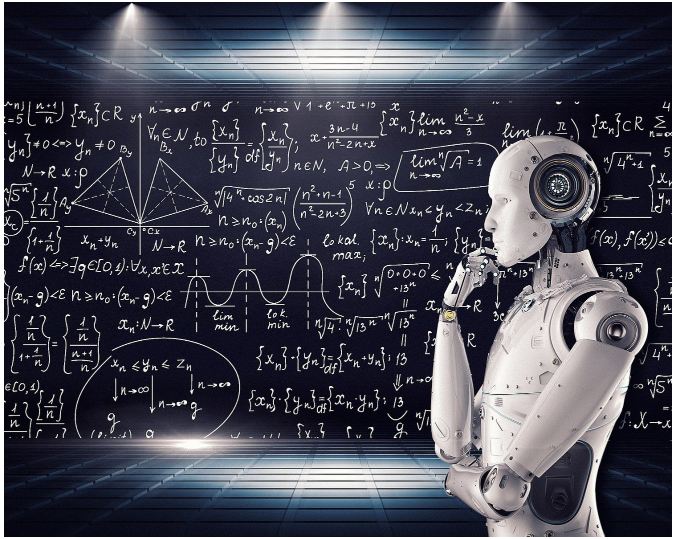


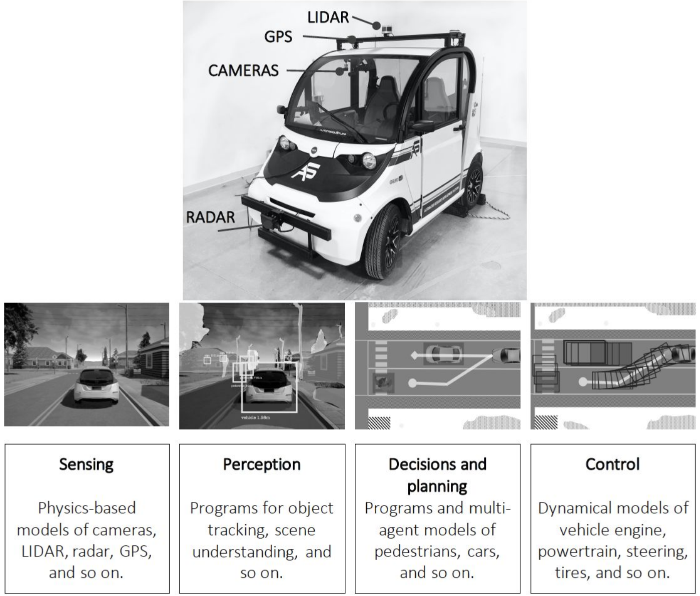
env: Lidar / vision

What is this course about?

Verification of Embedded & Cyber-physical Systems

“Formal Verification”





Sensing

Physics-based models of cameras, LIDAR, radar, GPS, and so on.

Perception

Programs for object tracking, scene understanding, and so on.

Decisions and planning

Programs and multi-agent models of pedestrians, cars, and so on.

Control

Dynamical models of vehicle engine, powertrain, steering, tires, and so on.

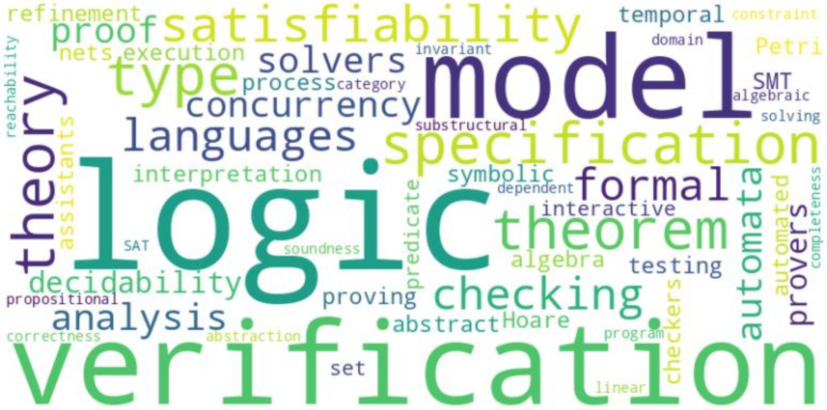
What is **formal verification**?

“**mathematically rigorous** techniques and tools for the **specification**, **design** and **verification** of software and hardware systems.”

-- NASA

“formal methods are **mathematically rigorous** techniques for the **specification**, **development**, **analysis**, and **verification** of software and hardware systems.”

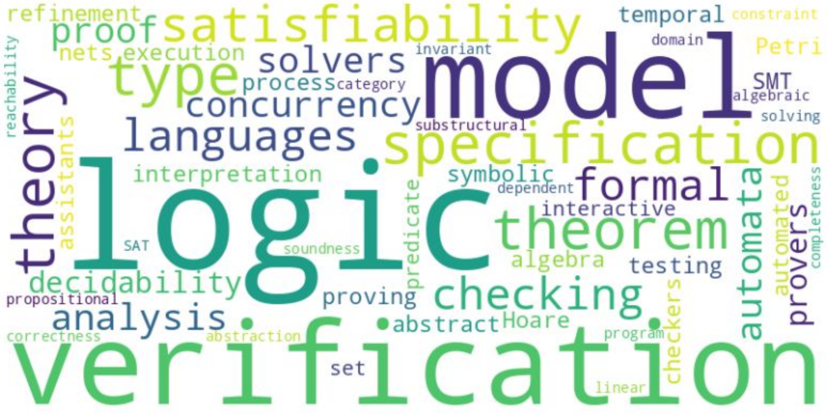
-- Wikipedia



What is **formal verification**?

“**mathematically rigorous** techniques and tools for the **specification**, **design** and **verification** of software and hardware systems.”

-- NASA



This class: deals with verification problems in

cyber-physical systems

(one special case of software + hardware combination)

Formal Verification in Software: an example

Simple programming task: given a 32-bit unsigned integer, calculate how many bits are set to 1 (“population count”)

1000 0000 0**1**00 0000 **1**000 0000 0**1**00 0000 -> 4

00**11** 0000 00**1**0 0000 **1**000 00**1**0 0**1**00 0000 -> 6

Formal Verification in Software: an example

Simple programming task: given a 32-bit unsigned integer, calculate how many bits are set to 1 (“population count”)

Naive implementation

```
int popcount(uint32_t x) {
  int c = 0;
  for (int i = 0; i < 32; i++) {
    c += x & 1;
    x >>= 1;
  }
  return c;
}
```

Example source: Marijn J.H. Heule, “SAT and SMT Solvers in Practice”

Formal Verification in Software: an example

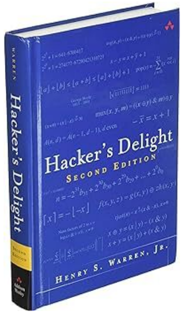
Simple programming task: given a 32-bit unsigned integer, calculate how many bits are set to 1 (“population count”)

Naive implementation

```
int popcount(uint32_t x) {
  int c = 0;
  for (int i = 0; i < 32; i++) {
    c += x & 1;
    x >>= 1;
  }
  return c;
}
```

Clever implementation

```
int popcount (uint32_t x) {
    x = x - ((x >> 1) & 0x55555555);
    x = (x & 0x33333333) + ((x >> 2) & 0x33333333);
    x = ((x + (x >> 4) & 0xf0f0f0f) * 0x1010101) >> 24;
    return x;
}
```



Example source: Marijn J.H. Heule, “SAT and SMT Solvers in Practice”

Formal Verification in Software: an example

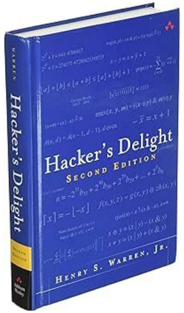
Can we trust this “clever implementation” of the same function?

What would you do to ensure this clever implementation is correct? Brute-force?

Naive implementation **?=** Clever implementation

```
int popcount(uint32_t x) {
  int c = 0;
  for (int i = 0; i < 32; i++) {
    c += x & 1;
    x >>= 1;
  }
  return c;
}
```

```
int popcount (uint32_t x) {
    x = x - ((x >> 1) & 0x55555555);
    x = (x & 0x33333333) + ((x >> 2) & 0x33333333);
    x = ((x + (x >> 4) & 0xf0f0f0f) * 0x1010101) >> 24;
    return x;
}
```



Example source: Marijn J.H. Heule, “SAT and SMT Solvers in Practice”

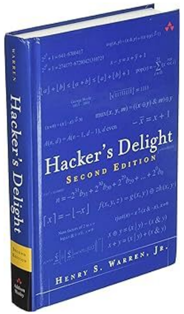
Formal Verification in Software: Specification

Formal verification aims to prove that **for all possible inputs,** the results of the two functions are formally the same (mathematically, the same integer is returned)

Naive implementation **==** Clever implementation

```
int popcount(uint32_t x) {
  int c = 0;
  for (int i = 0; i < 32; i++) {
    c += x & 1;
    x >>= 1;
  }
  return c;
}
```

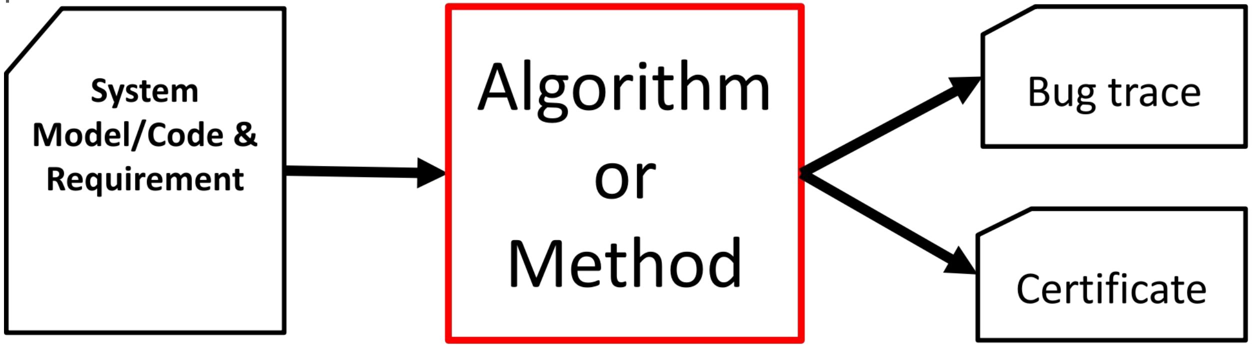
```
int popcount (uint32_t x) {
    x = x - ((x >> 1) & 0x55555555);
    x = (x & 0x33333333) + ((x >> 2) & 0x33333333);
    x = ((x + (x >> 4) & 0xf0f0f0f) * 0x1010101) >> 24;
    return x;
}
```



Example source: Marijn J.H. Heule, “SAT and SMT Solvers in Practice”

The verification problem: model + requirement + algorithm

Model: boolean logic



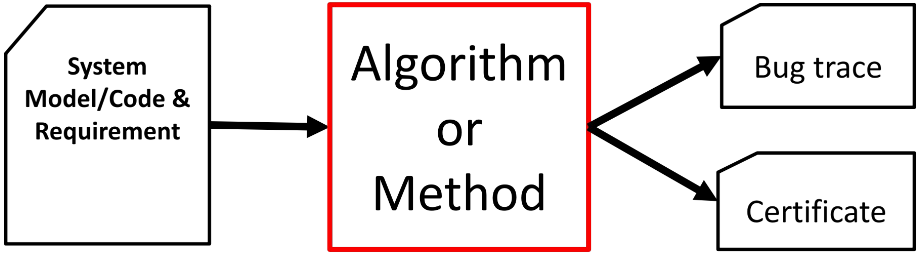
Counterexamples

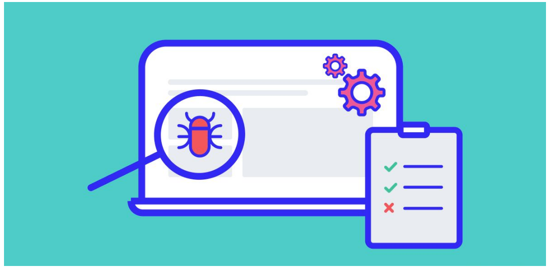
Proofs!

Requirement: check if the two functions return the same integer

Verification algorithm to solve boolean satisfiability (DPLL, CDCL)

Verification vs Testing





Testing: evaluates requirements on a finite number of behaviors

Verification: aims to prove requirements over all behaviors

Formal Verification in Software: an example

Formal verification aims to prove that **for all possible inputs,** the results of the two functions are the same

How is that achieved? Possible without enumeration?

Convert the problem in to logic and use a satisfiability (SAT) solver

Example source: Marijn J.H. Heule, “SAT and SMT Solvers in Practice”

SMTLIB formulation

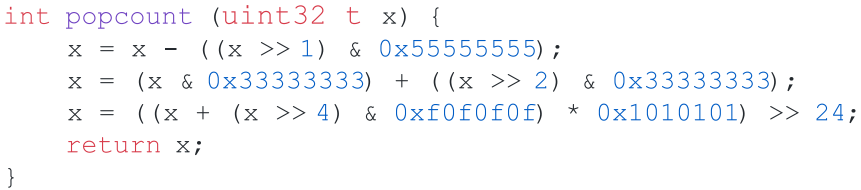
Implement the naive version

```c
int popcount(uint32_t x) {
  int c = 0;
  for (int i = 0; i < 32; i++) {
    c += x & 1;
    x >>= 1;
  }
  return c;
}
```

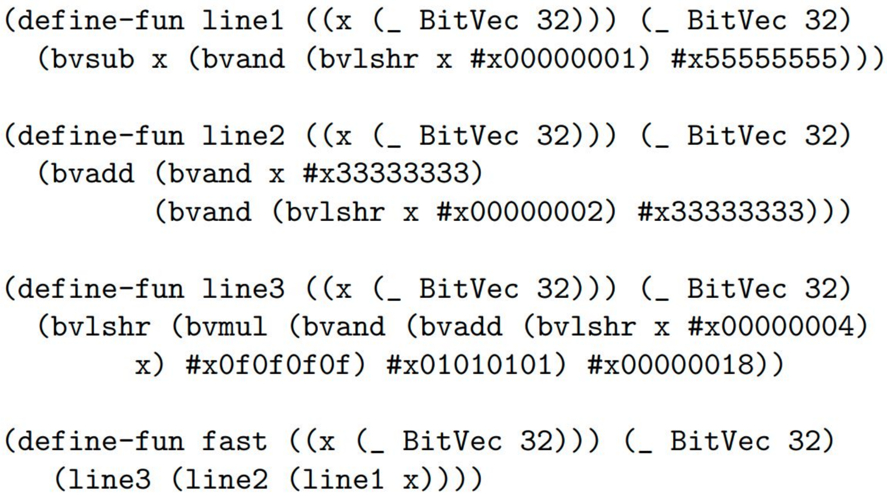
```
(define-fun slow ((x (_ BitVec 32))) (_ BitVec 32)
 (bvadd
   (ite (= #b1 ((_ extract  0  0) x)) #x00000001 #x00000000)
   (ite (= #b1 ((_ extract  1  1) x)) #x00000001 #x00000000)
   (ite (= #b1 ((_ extract  2  2) x)) #x00000001 #x00000000)
                          ...
   (ite (= #b1 ((_ extract 30 30) x)) #x00000001 #x00000000)
   (ite (= #b1 ((_ extract 31 31) x)) #x00000001 #x00000000)))
```

SMTLIB formulation

Implement the faster version



```c
int popcount (uint32 t x) {
    x = x - ((x >> 1) & 0x55555555);
    x = (x & 0x33333333) + ((x >> 2) & 0x33333333);
    x = ((x + (x >> 4) & 0xf0f0f0f) * 0x1010101) >> 24;
    return x;
}
```



```
(define-fun line1 ((x (_ BitVec 32))) (_ BitVec 32)
  (bvsub x (bvand (bvlshr x #x00000001) #x55555555)))

(define-fun line2 ((x (_ BitVec 32))) (_ BitVec 32)
  (bvadd (bvand x #x33333333)
         (bvand (bvlshr x #x00000002) #x33333333)))

(define-fun line3 ((x (_ BitVec 32))) (_ BitVec 32)
  (bvlshr (bvmul (bvand (bvadd (bvlshr x #x00000004)
         x) #x0f0f0f0f) #x01010101) #x00000018))

(define-fun fast ((x (_ BitVec 32))) (_ BitVec 32)
   (line3 (line2 (line1 x))))
```

SMTLIB formulation (full program, can be verified using Z3)

```
(set-logic QF_BV)
```

“Quantifier free bit vectors”

```
(declare-const x (_ BitVec 32))
```

Define inputs

```
(define-fun fast ((x (_ BitVec 32))) (_ BitVec 32)
    ...
```

Implement the clever version

```
(define-fun slow ((x (_ BitVec 32))) (_ BitVec 32)
    ...
```

Implement the naive version

```
(assert (not (= (fast x) (slow x))))
```

Assert the outputs are not the same

```
(check-sat) ; expect UNSAT
```

Ask the verifier to check this claim

```
(exit)
```

UNSAT means the assertion cannot be satisfied for any input x => two functions equal

Successes of Verification

Hardware verification now standard in EDA tools from Synopsys, Cadence, etc.

SLAM tool from MSR routinely used for verification of Device Drivers at Microsoft:

AMAZON AWS developers write proofs using CBMC and other Automated reasoning tools

Google runs static analysis tools on their entire codebase

Airbus: verified C code on safety-critical software for various plane series, including the A380

Formal modeling and analysis is becoming part of certification process for avionics (e.g., ASTREE); DO-333 supplement of DO-178C identifies aspects of airworthiness certification that pertains to of software using *formal methods*

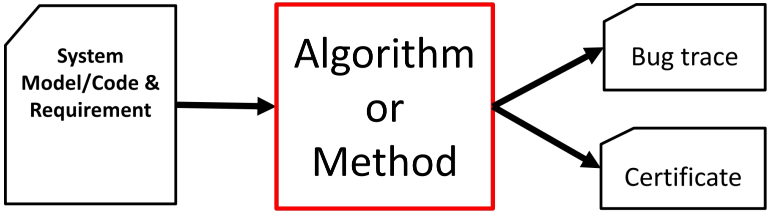
Commercialization: Coverity, Galois, SRI, and others

Check out https://github.com/ligurio/practical-fm

How about verification + cyber-physical systems (CPS)?

CPS: A computer system monitoring or controlling a physical process.

Examples: a drone for package delivery, control system for a smart electric grid, insulin pump for blood glucose control, …



Models: deal with the physical world

- ○ Boolean logic is likely not enough
- ○ Must describe the physics!

Requirements: statements about all possible behaviors

- ○ Drone visits waypoints while avoiding collisions
- ○ Under all nominal conditions the vehicle stays within the lanes
- ○ Insulin pump maintains blood glucose level to within the prescribed range

Algorithms: handle complex real-world scenarios

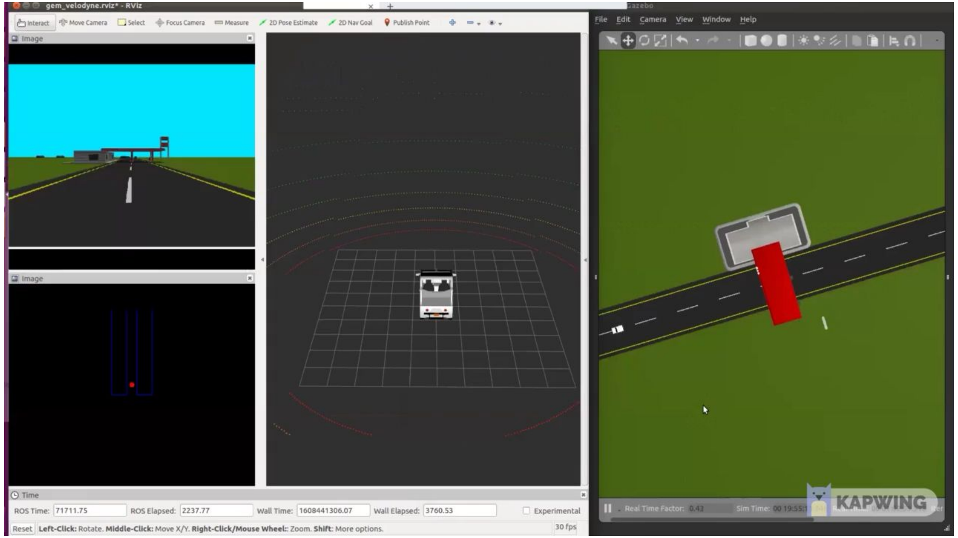
- ○ The number of possible behaviors of such systems is usually *uncountably infinite*

Open problem

Simulated race car following a track with Lidar/vision-based perception and control.

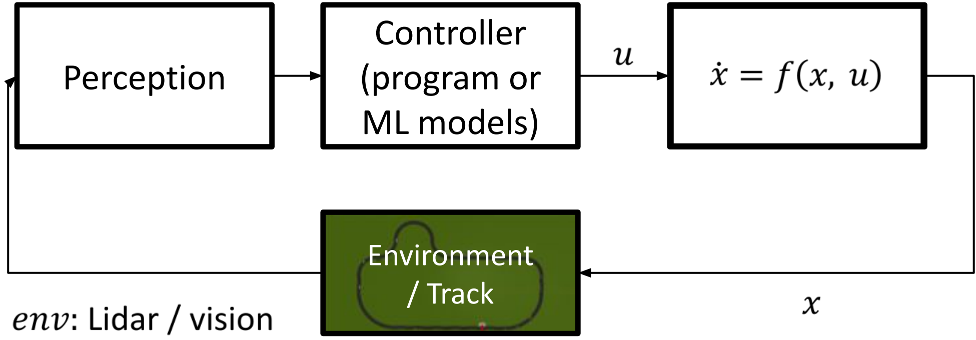
Problem: For a given track and initial conditions check that the *trajectory* of the car does not collide and stays in lane.

Can we check *efficiently*?
Can we *generalize* to *similar* tracks?
What should we assume about perception, accuracy of the vehicle model?
What should we assume about the execution of the controller?



ROS Time: 71711.75 ROS Elapsed: 2237.77 Wall Time: 1608441306.07 Wall Elapsed: 3760.53

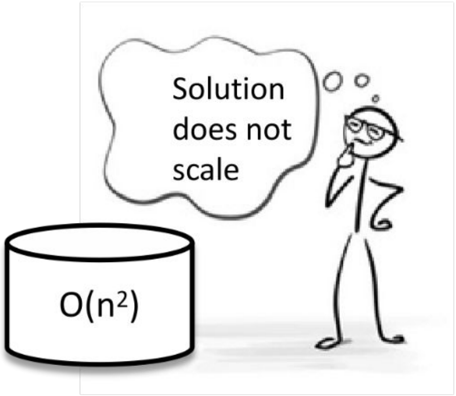
Left-Click: Rotate. **Middle-Click:** Move X/Y. **Right-Click/Mouse Wheel:**: Zoom. **Shift**: More options.



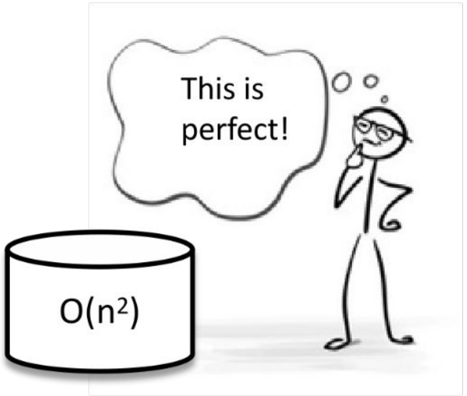
Verification problems are very challenging: **scalability**

Essentially we are seeking for formal mathematical proofs

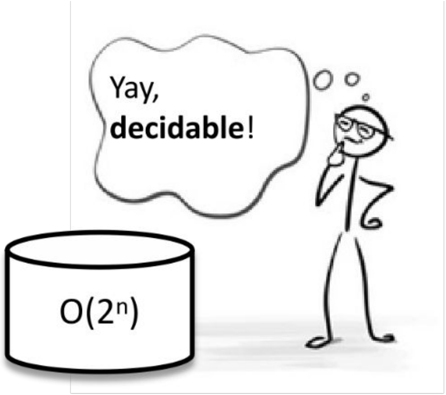
Proof usually done by a systematic algorithm



data scientist



algorithmist



verification engineer

Image from Prof. Sayan Mitra

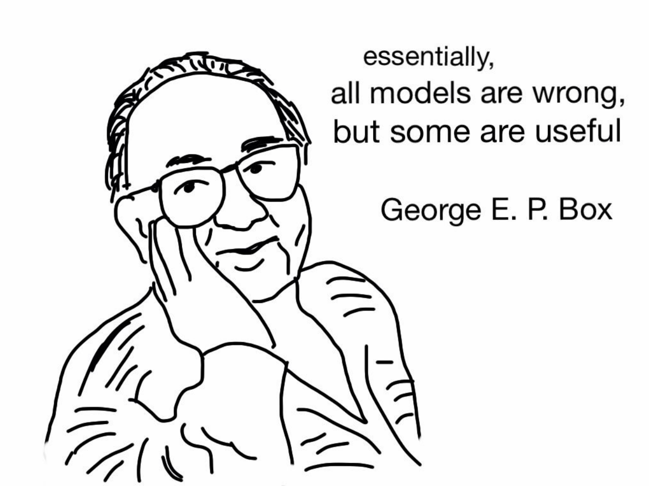
Verification problems are very challenging: **modeling**

To prove anything, first we have to start with **assumptions**

Assumptions are captured in the ***models*** (of cyberphysical systems)

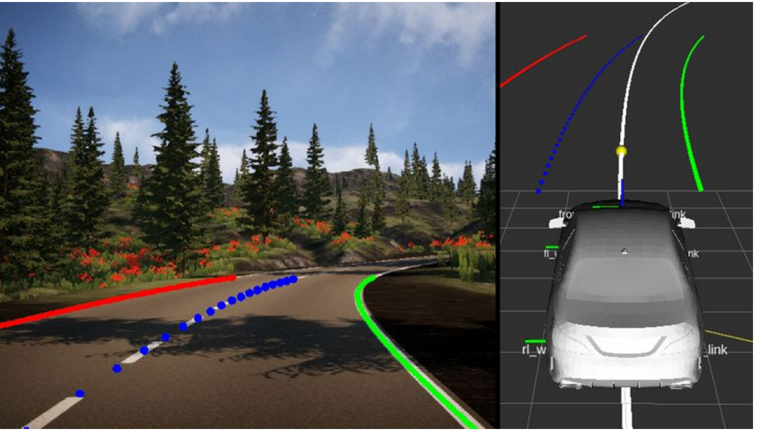
1/3 of this class is about models

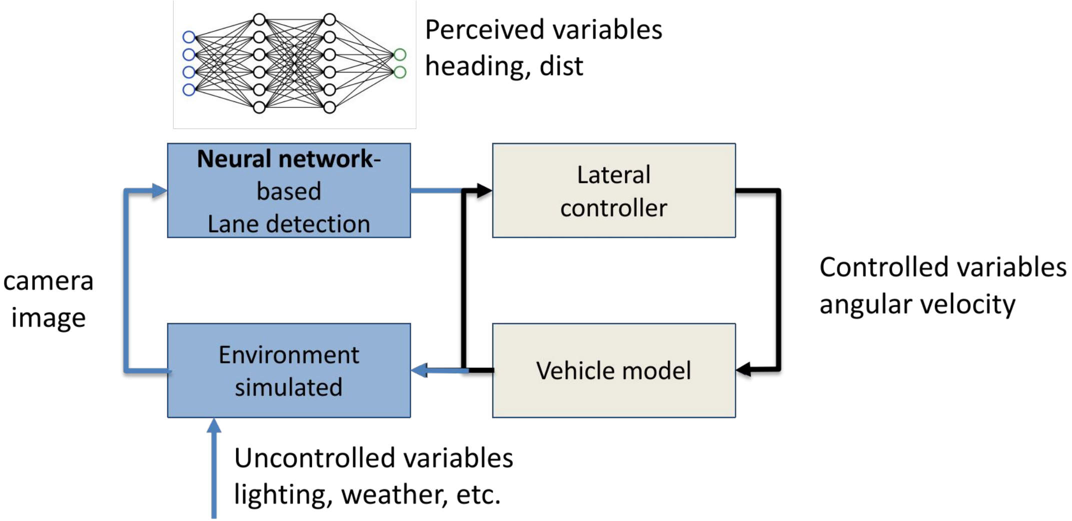
- Programs, state machines, or differential equations, block diagrams
- Discrete or continuous time, state or both -- hybrid
- Deterministic or nondeterministic or probabilistic
- Composition and interfaces, abstraction
- Deal with **machine learning**, **deep neural networks**



https://tribalsimplicity.com/2014/07/28/george-box-models-wrong-useful/

Verification problems are challenging: system complexity





Perceived variables
heading, dist

Neural network-
based
Lane detection

Lateral
controller

camera
image

Controlled variables
angular velocity

Environment
simulated

Vehicle model

Uncontrolled variables
lighting, weather, etc.

Learning objectives

- Introduction to key concepts in **formal methods** and **cyberphysical systems**; exposure to some of the most influential ideas in CS and control theory
- Different types of **models**
- Foundational connections between computer science and control theory
- Learn powerful algorithms and tools
- Jumpstart research

Invariant, barrier certificates, stability, self-stabilization, convergence

Programs, state machines, or differential equations, discrete or continuous state or both, Hybrid, switched, deterministic or nondeterministic or both, composition, abstraction

satisfiability modulo theory, semantics, temporal logics, theorem provers, SAF solvers, ranking functions, neural network verification

semester-long project, feedback, presentation, hardware, software, and data resources

Course Format

- Lectures about **formal methods** and **cyber-physical systems**
  - We will spent a few weeks on verifying machine learning & AI, since these are popular in CPS nowadays
- Four sets of homework - covering basic knowledge and some coding experiences
- **Class project** - very important component of the class (proposal, midterm feedback, final presentations)
- No exams

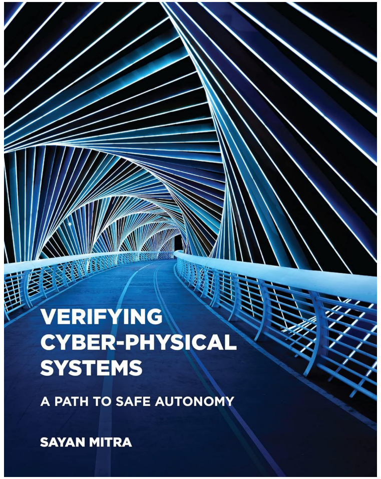
Course Logistics

Course website: https://publish.illinois.edu/ece584-spring2025/

Canvas: homework submission & announcement
https://canvas.illinois.edu/courses/56512

Lectures TR 11:00 am – 12:20 pm

Textbook (by Prof. Sayan Mitra):
Verifying Cyber-Physical Systems: A Path to Safe Autonomy



Slides will be posted on website after lecture on the course schedule page:
https://publish.illinois.edu/ece584-spring2025/course-schedule/

Reading assignment before each class (check course schedule)

Grading

- Homework (4 sets): 40%
  - Problems will involve some coding and pencil paper proofs
- Project: 55%
  - Proposal: 10%
  - Midterm presentation: 10%
  - Final presentation (EOS): 15%
  - Report (EOS), code and documentation: 20%
- Participation: 5%
  - Provide feedback on peer projects
  - Class participation, new problem suggestions and solutions

Homework

Homeworks: 4 sets. Analysis and some coding

Due **11:59 pm** on the due date (**hard deadline**, **no exceptions** unless with university approved absence letters)

Late policy: 20% grade reduction per day; no late homework will be accepted >=5 days after the due date.

Submit your homework on **Canvas**

First homework will be release next week (likely on Friday/weekend)

Typically you are given 2+ weeks for each set of homework

TA: Keyi Shen (keyis2 at illinois.edu)
Contact the TA for homework submission related issues and re-grading request.

Office Hours

By appointment. 1.5 hours available per week

TA Office Hours: Tuesdays 3:30 – 4:30 pm (location TBA or Zoom). Please book your appointment 24 hours in advance:

In person: https://cal.com/keyi-shen-jc0apb/ece-cs-584-ta-office-hours-in-person

Zoom: https://cal.com/keyi-shen-jc0apb/ece-cs-584-ta-office-hour-zoom

Instructor Office Hours: Thursdays 3:30 – 4 pm (CSL 262; Zoom upon request).

Book your appointment 24 hours in advance: https://cal.com/huanzhang/zhang-in-person-hour

After booking, you should receive a confirmation email (if not, please check your spam folder).

Additional office hours are **available upon request**.

Class Project

Proposal due 3/3

Work in **a team of 2** (if you want to form a larger team, please talk to me)

Work solo is o

Mid-semester project presentation and feedback: first week after Spring break

Final presentation: last week of instruction

Final report: due in finals week

Starting thinking about course projects!

I listed some ideas in a document (mostly relevant to neural network verification directions, which will be cover in mid~late Feburary)

https://tinyurl.com/ece584-project-ideas