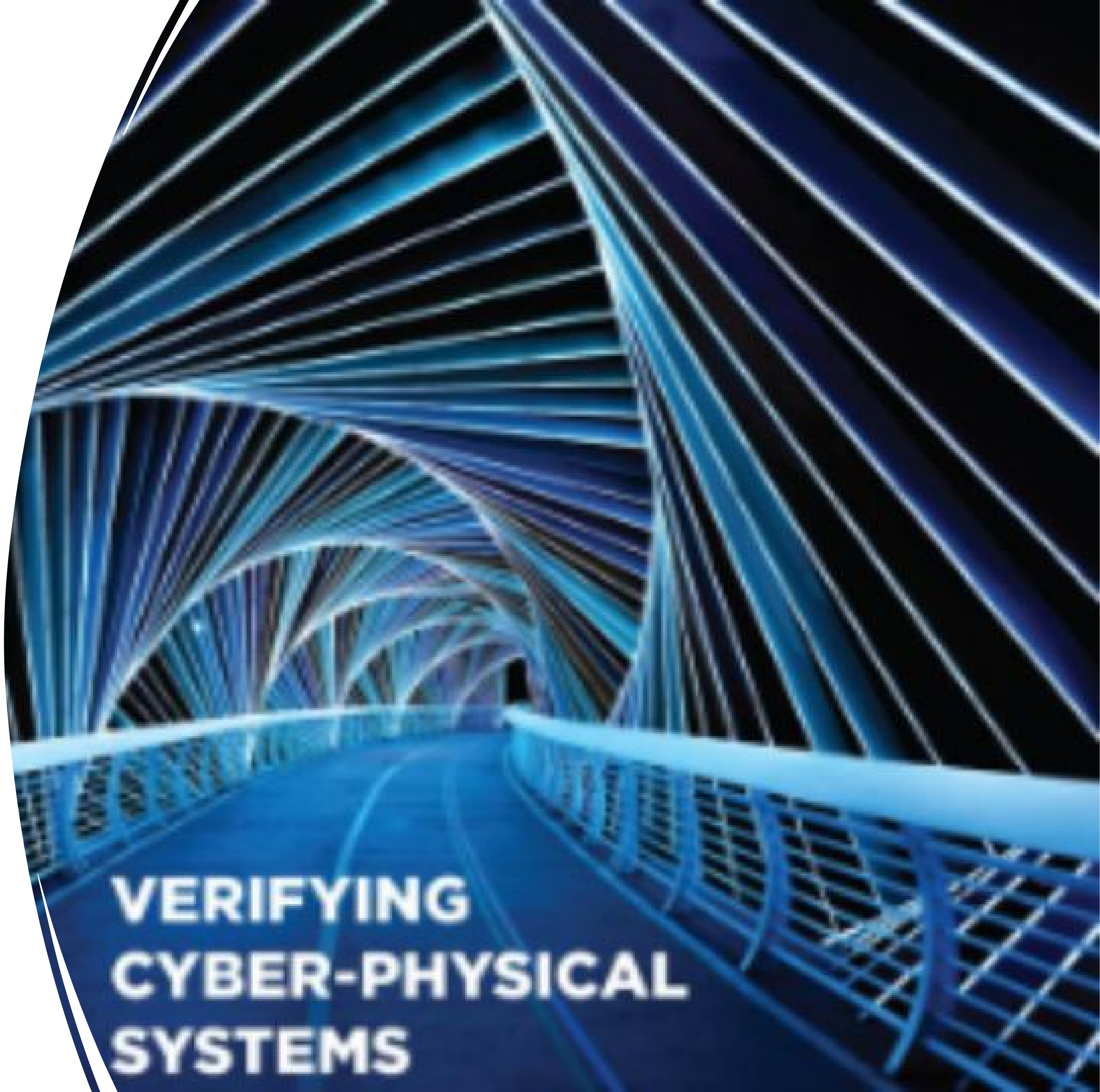
Course Introduction
Verification of embeded & cyberphysical systems
Spring 2024

Huan Zhang
CSL 262
huan@huan-zhang.com

Slides adapted from Prof. Sayan Mitra's slides in Fall 2021

Welcome to Spring 24 edition of ECE/CS 584!



What is this class about?

INTRODUCTION

Verification of embeded & **cyberphysical** systems

What is verification?

Definition. ***Verification*** *is the action of demonstrating or proving some statement to be true by means of evidence. OED*

This class:

some statement = about cyber-physical systems

evidence = mathematical proof

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, …

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

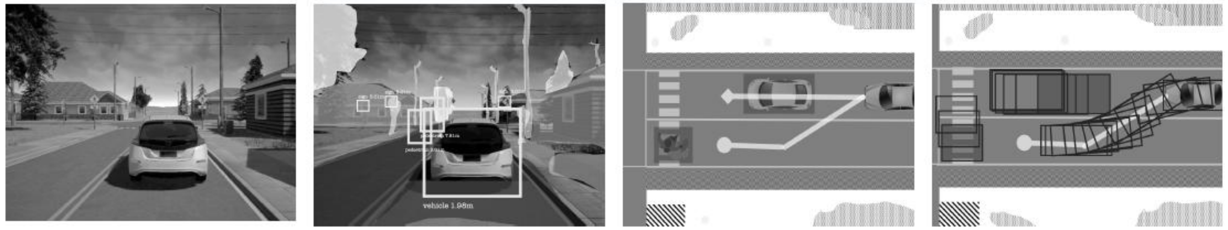
Requirements: Statements about all *behaviors*

- 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

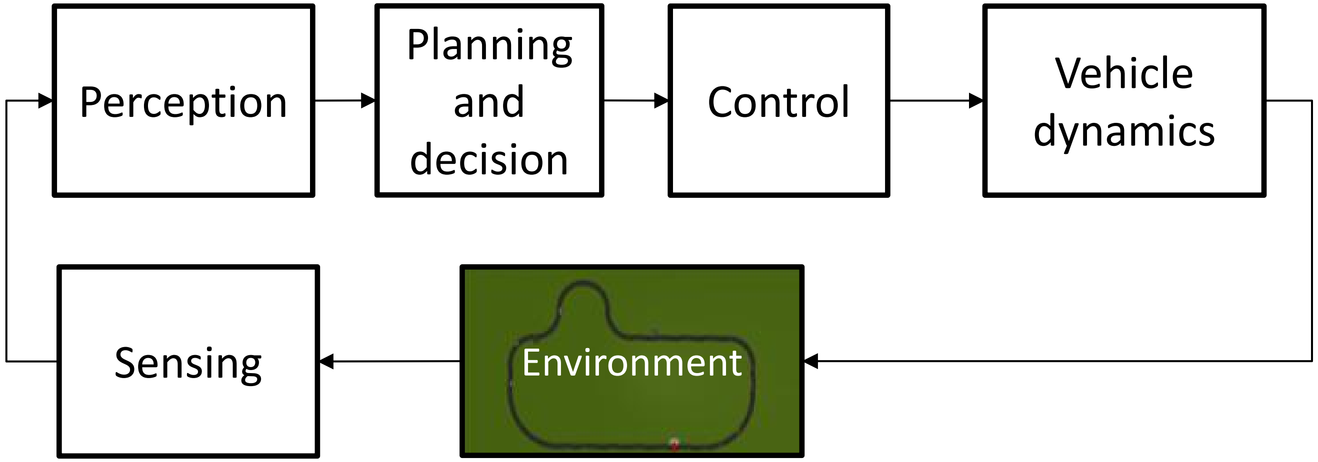
Testing: evaluates requirements on a finite number of behaviors

Verification: aims to prove requirements over all behaviors

Autonomous vehicle: An example CPS



Sensing	Perception	Decisions and planning	Control
Physics-based models of cameras, LIDAR, radar, GPS, and so on.	Programs for object tracking, scene understanding, and so on.	Programs and multi-agent models of pedestrians, cars, and so on.	Dynamical models of vehicle engine, powertrain, steering, tires, and so on.



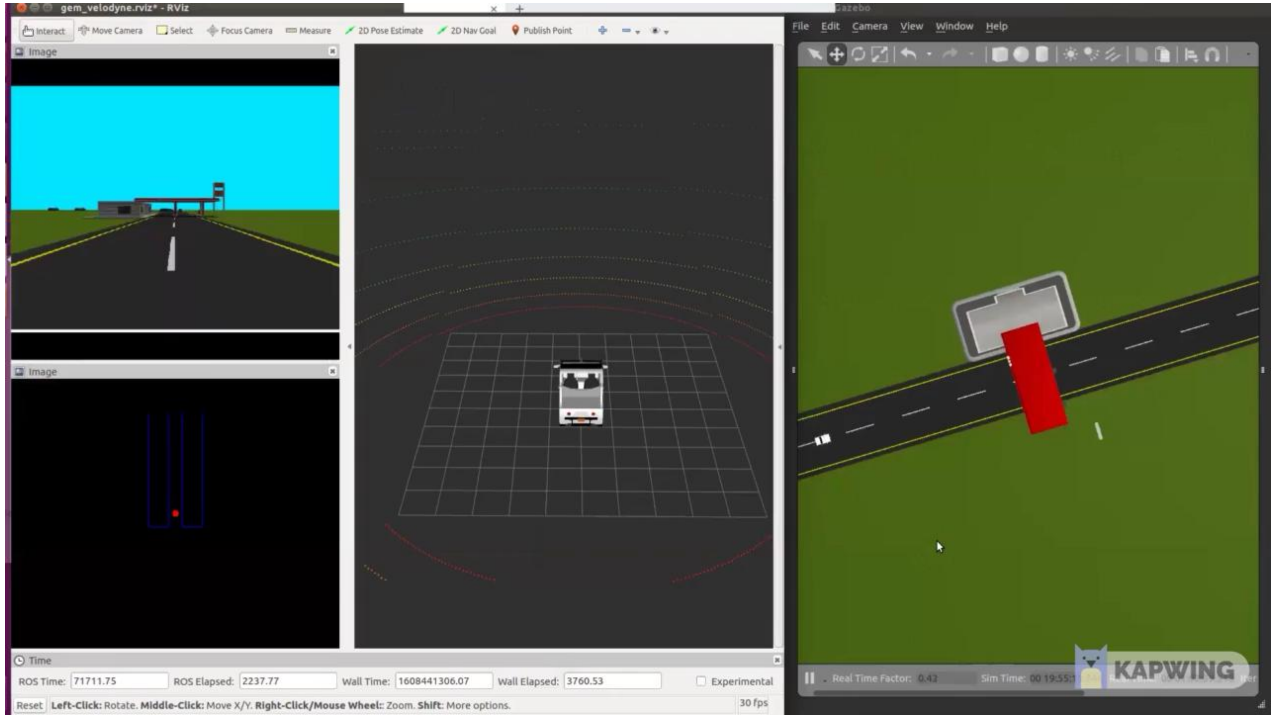
env: Lidar / vision

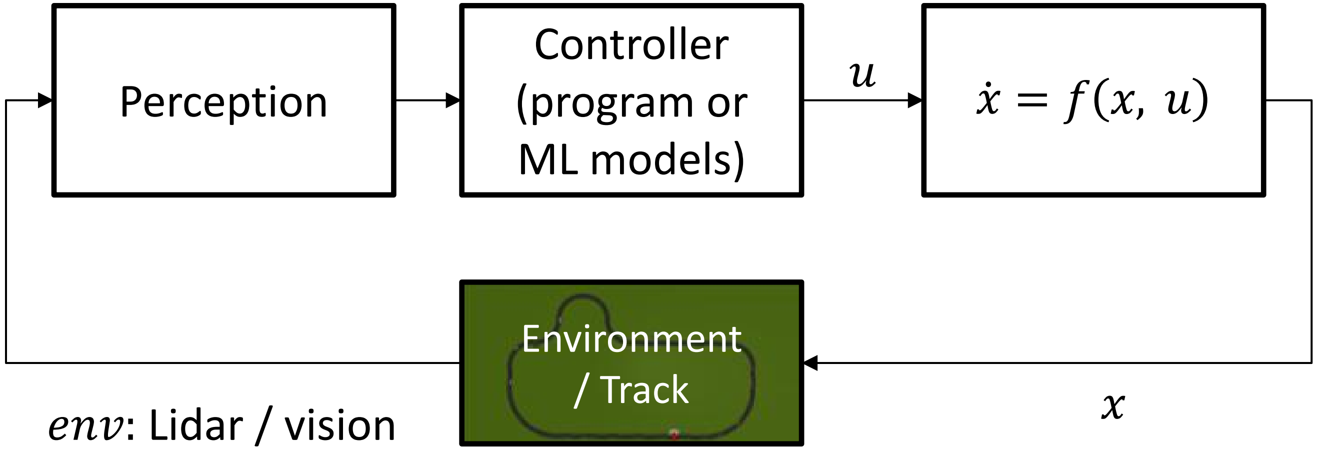
Open problem

Simulated race car following a track with Lidar-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?



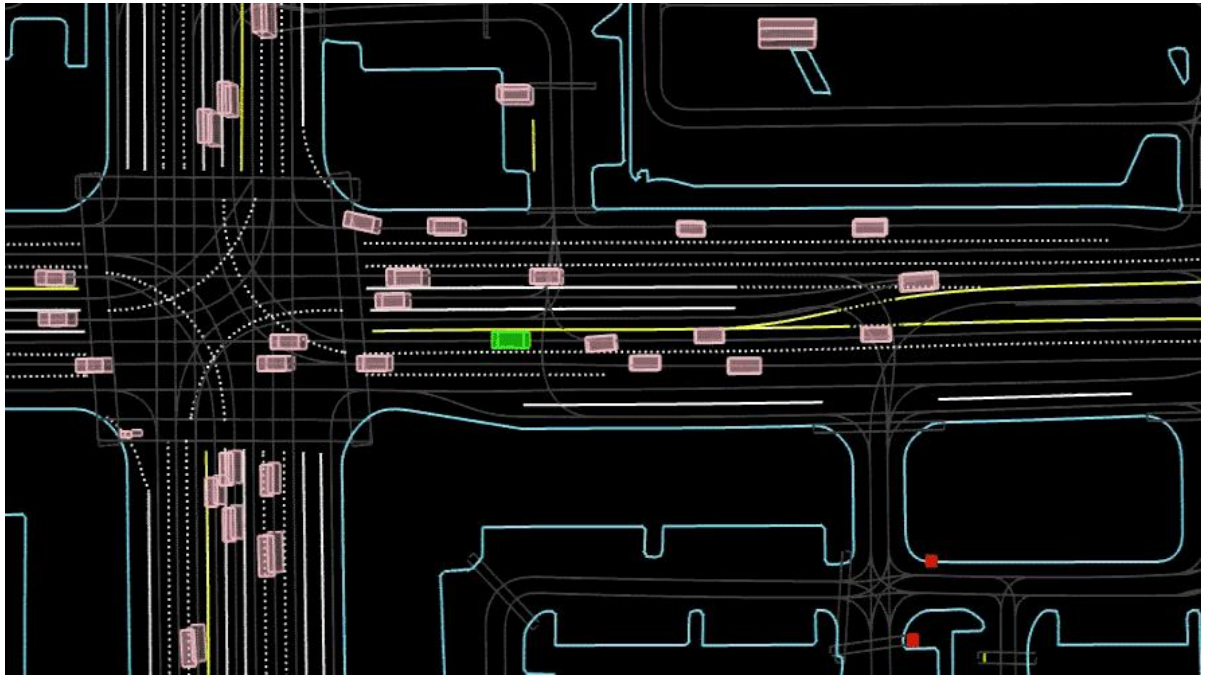


Open problem

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

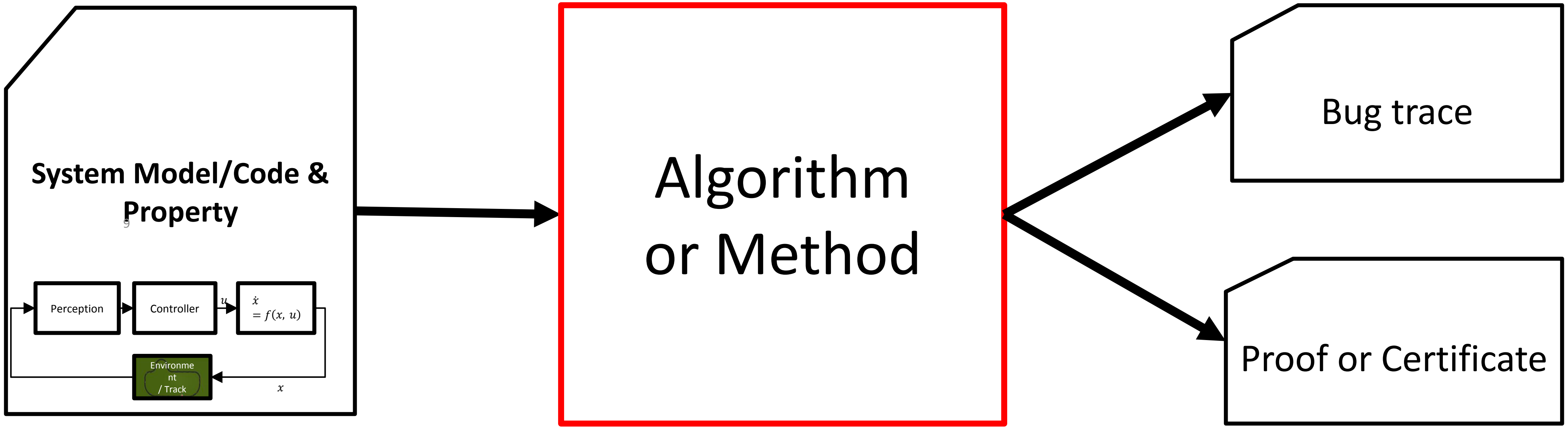
Problem: For a given track and initial conditions check that the *trajectory* of the car does not collide and stays in lane.

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?
Even more challenging with multiple agents



https://github.com/waymo-research/waymax

The verification problem



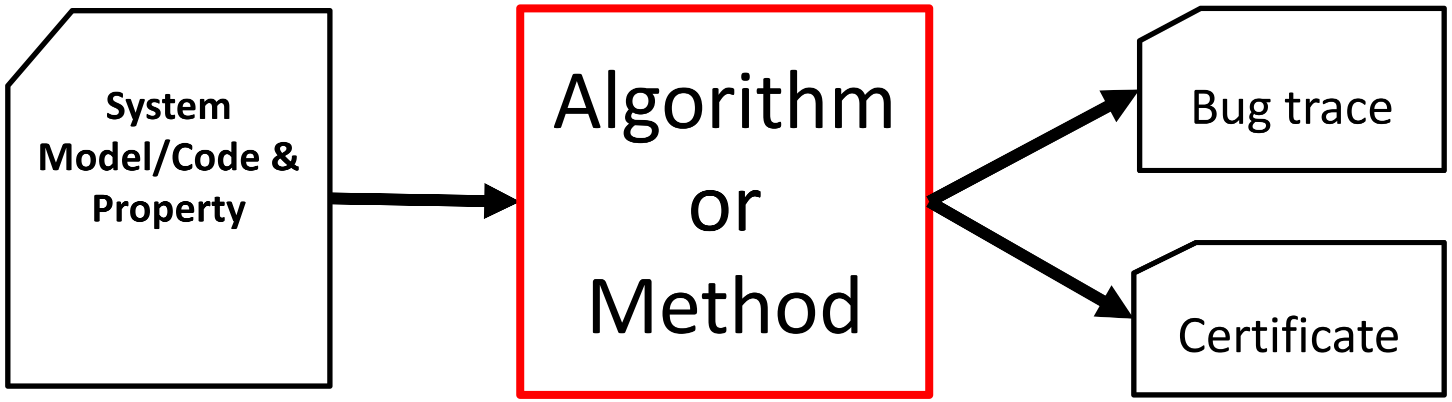
Verification. *The action of demonstrating or proving to be true by means of evidence; formal assertion of truth. (OED)*

Program verification

System. A subroutine
`sort(int a[])`
for returning a sorted array of integers in some programming language, e.g. C

A model M for execution of programs in C

Requirement. Output of
`sort(int a[])`
is the sorted version of the input array `a[]`



counterexample. A particular input array `a` and initialization of **`sort`** that produces wrong output

A mathematical proof that establishes that **`sort`**`(int a[])` works for all inputs in the given model M of C

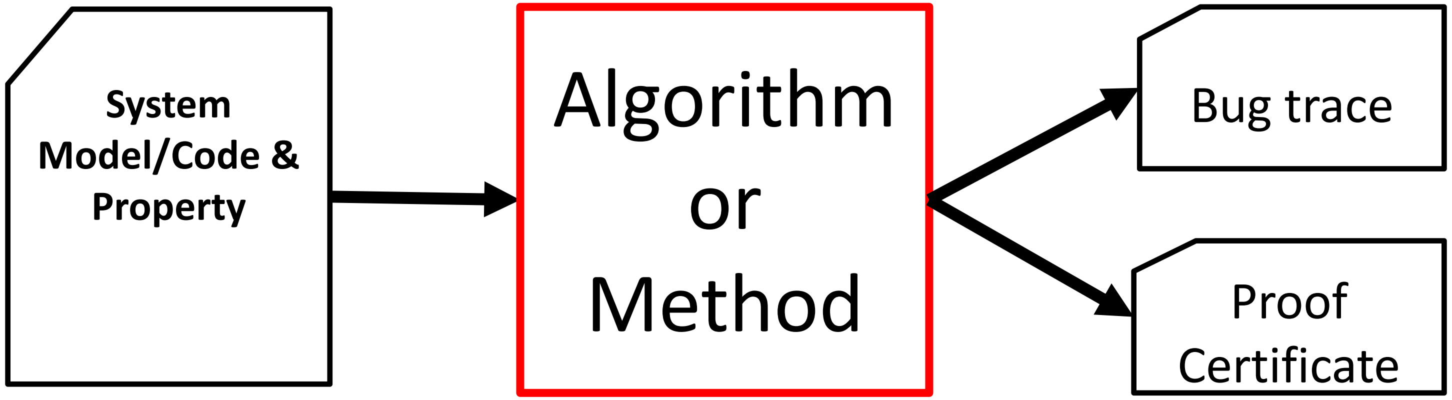
Verifying compiler.
Checks that **`sort`** meets the requirement

A *cyber-physical* example

System. A program/system for *lane keeping control* for vehicles

Model/assumptions for executing such programs including the effects on the physical vehicle

Requirement. The vehicle does not go outside the lane boundaries



counterexample. A particular environment situation (lane geometry, sensor failure, computer configuration) that makes the vehicle go outside lanes

A mathematical proof that establishes that for all *allowed* inputs and environments the vehicle stays with the lane

Verification tool

When can we build such a tool? How expensive is it? How well is it going to work? Under what assumptions?

Algorithm or Method

Our goals in this course

Write programs (tools) that prove correctness

- *Understand fundamental limits of creating such tools*
- *Learn models of CPS at different levels of abstractions*
- *Gain research experience*

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 u

Commercialization: Coverity, Galois, SRI, and others

"Things like even software verification, this has been the Holy Grail of computer science for many decades but now in some very key areas, for example, driver verification we're building tools that can do actual proof about the software and how it works in order to guarantee the reliability." **Bill Gates, April 18, 2002. Keynote address at WinHec 2002**

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

Intellectual successes

Turing Awards:

Lamport (2013): Verification of distributed and concurrent systems

Clarke, Sifakis & Emerson (2006): Model checking

Pnueli (1996): Temporal logic

Lampson (1992): Distributed system

Milner (1991): Logic for Computable Functions, Meta Language

Hoare (1980): Hoare logic, program verification

Rabin & Scott (1976): Finite Automata

Dijkstra (1972): Structured programming, algorithms, distributed systems

Intellectual successes

ACM Doctoral Dissertation Award: Chuchu Fan (2020) alumni of this class, now a professor at MIT

Covers and connects some of the brightest ideas in CS and control

Vibrant community: CAV, TACAS, PLDI (programming languages),

HSCC, EMSoft, ICCPS (hybrid and cyber-physical systems)

Robotics, automatic control (IROS, ICRA, RSS, CDC, ACC, ...)

AI and machine learning (NeurIPS, ICML, ICLR, ...)

Faculty and research positions: Alumni of this course are professors at Vanderbilt, UNC Chapel Hill, MIT, Kansas, Stoney Brook, and researchers at Waymo, Toyota, Boeing

Can you name a few challenges for CPS verification?

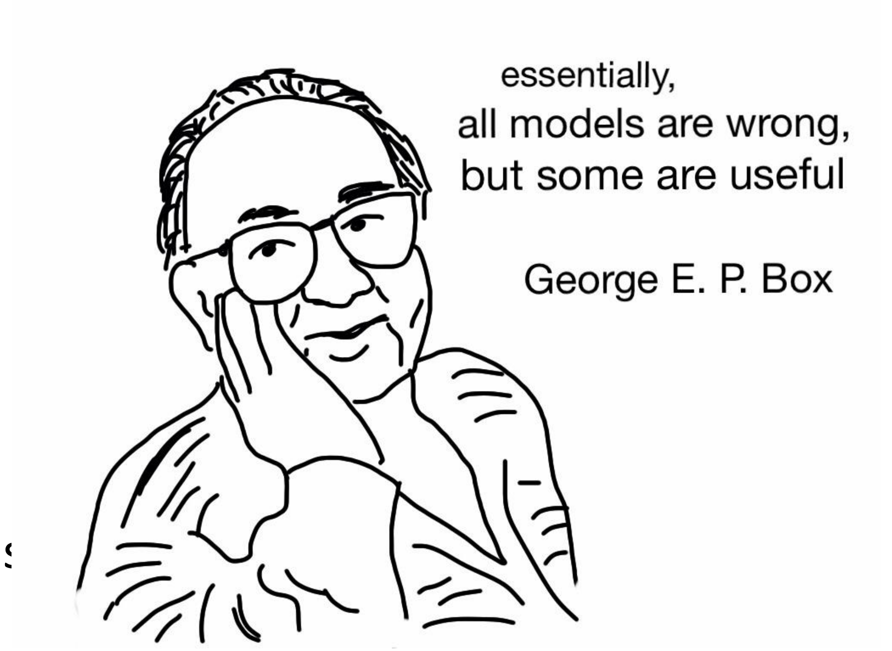
Challenge 1: Models

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
- Modeling languages, tools
- Modeling machine learning, deep neural networks

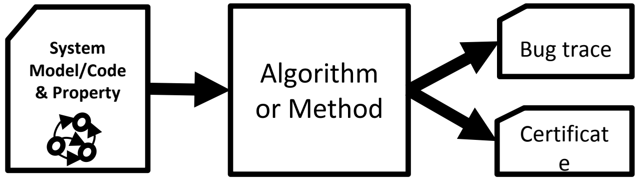


https://tribalsimplicity.com/2014/07/28/george-box-models-wrong-useful/

Challenge 2: Scalability

Verification of hybrid automaton is *undecidable*

- impossible to construct an algorithm that always leads to a correct yes-or-no answer

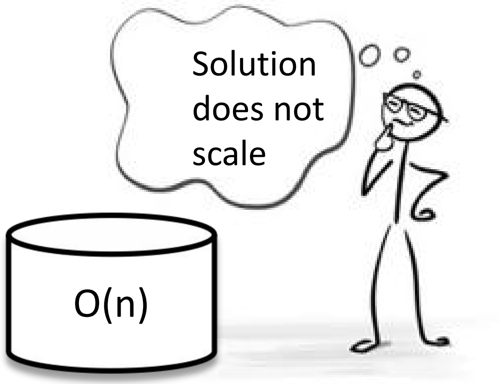


Approximate and bounded time versions of the problem can be solved algorithmically

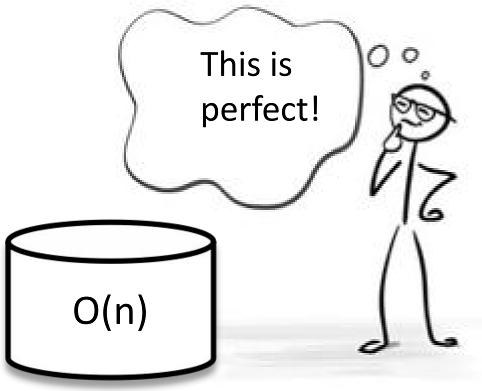
Often the algorithms do not *scale* with the size of the model, number of agents, time horizon, etc.

Trilemma: **Scalability** of analysis vs **Expressivity** of model vs **Precision** of analysis

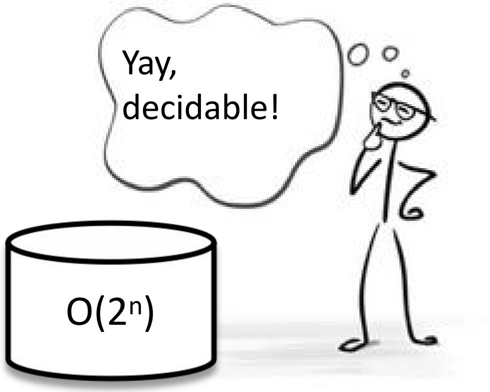
Perspectives on scalability



data scientist

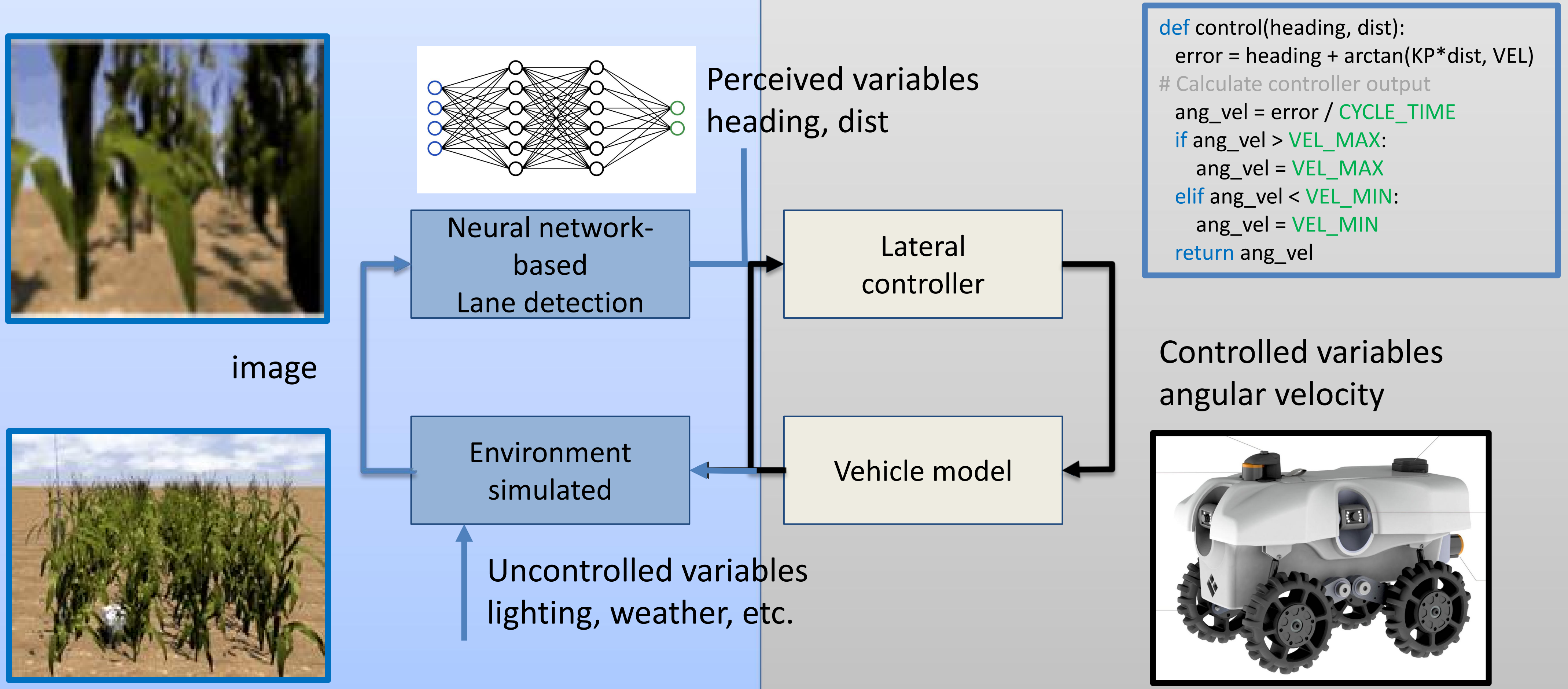


algorithmist



verification engineer

Challenge 3: Perception & Machine Learning



New, underspecified, empirical

Well-understood

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
- Model anything
- Foundational connections between computer science and control theory
- Learn powerful algorithms and tools
- Jumpstart research

Invariant, barrier certificates, ranking functions, stability, self-stabilization, convergence, transition system

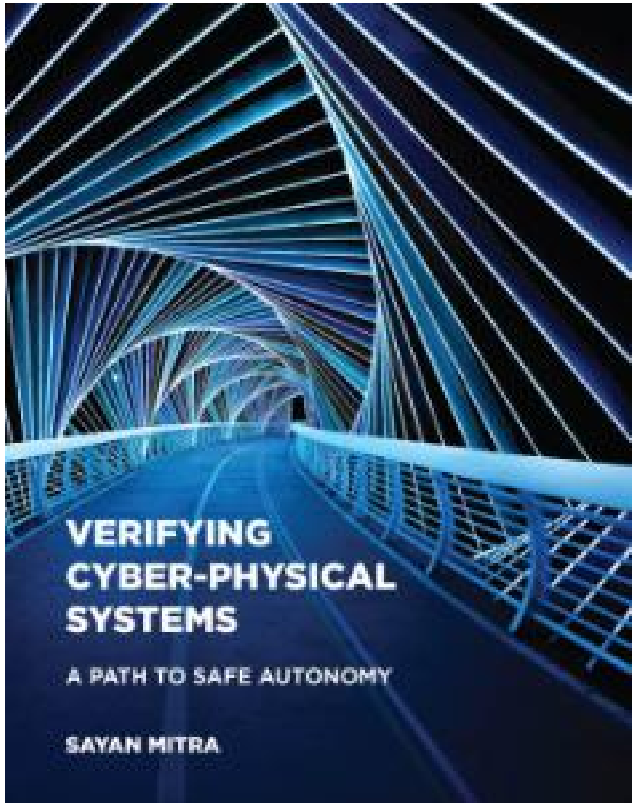
Programs, state machines, or differential equations, discrete or continuous state or both, Hybrid, switched, Deterministic or nondeterministic or both, composition, interfaces, abstraction, modeling languages, tools

satisfiability modulo theory, semantics, temporal logics, theorem provers, SAF solvers, ranking functions, data-driven verification, HYLAA, C2E2, SpaceEx, Flow*, Z3, …

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

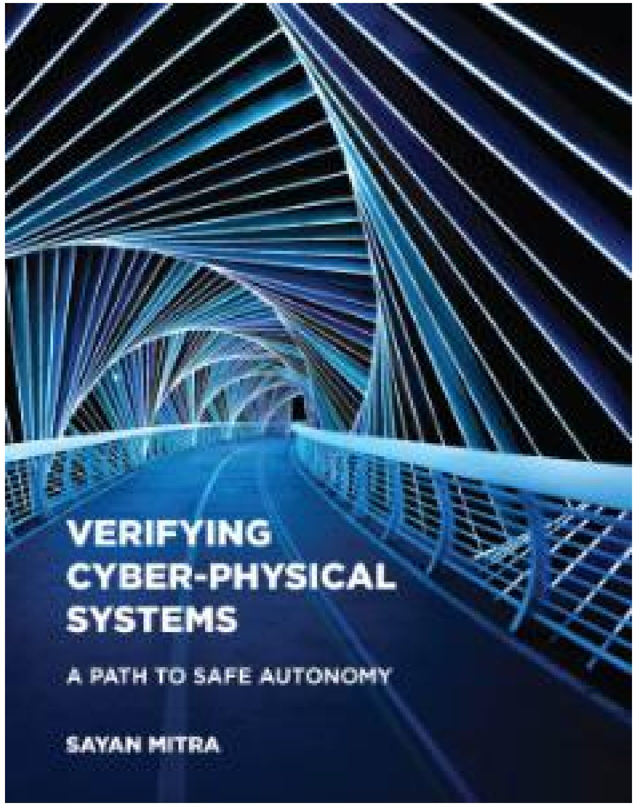
Course Logistics

- Course website: https://publish.illinois.edu/ece584-spring2024/
- Canvas: homework submission & announcement
- Lectures TR 11:00 – 12:20
- Textbook (by Prof. Sayan Mitra)
- Slides will be posted on website after lecture
- Please do the reading assignment before each lecture



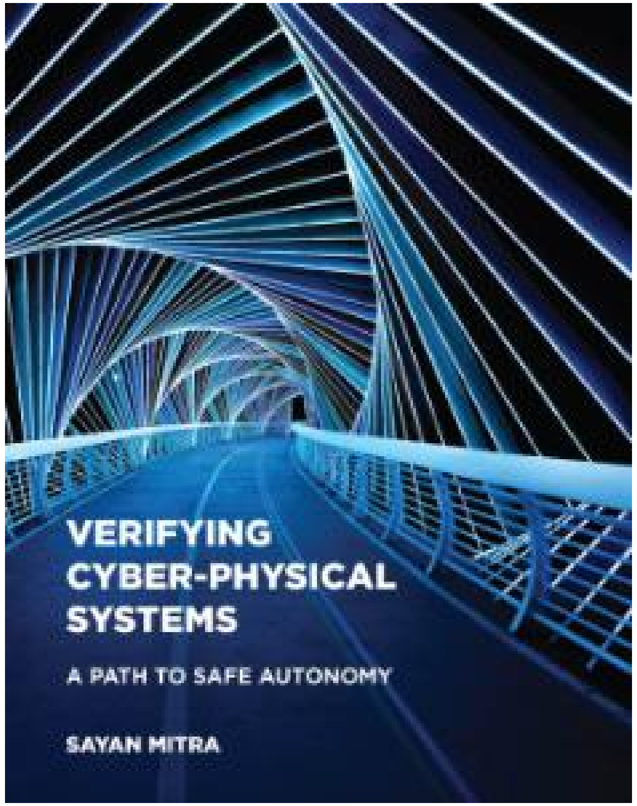
Homework

- Homeworks: 5 sets. Analysis and some coding
  - Due 11:59 pm on the due date (hard deadline, no exceptions)
  - Late policy: 20% grade reduction per day; no late homework will be accepted >=5 days after the due date.
- Submit your homework on Canvas:
  - https://canvas.illinois.edu/courses/44138
  - HW1 will be released on 01/23 and due on 02/09



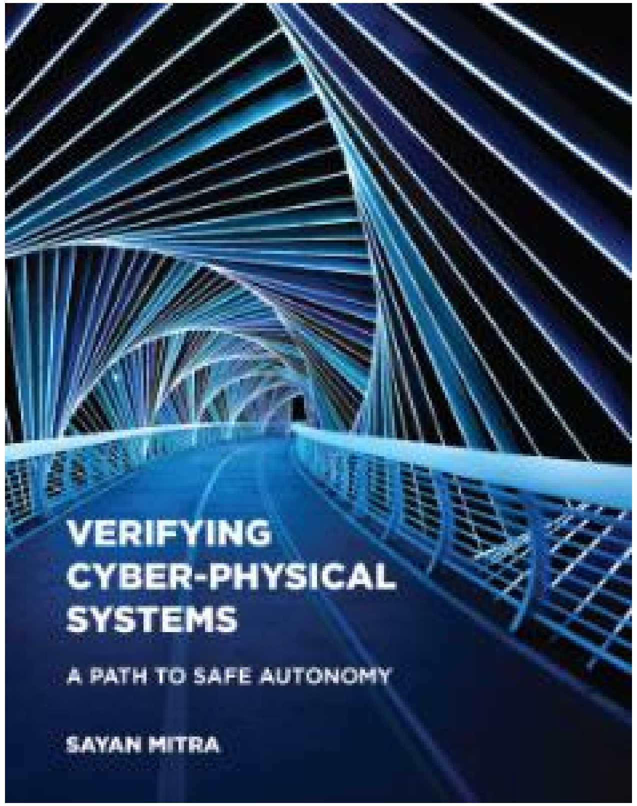
Project

- Proposal due 2/23
- Work individually or in a team of 2 (if you want to form a larger team, please talk to me)
- Mid-semester project review: after Spring break
- Final presentation: last week of instruction
- Final report: due in finals week



Office Hours

- By appointment. 2 hours available per week:
  - In-person: https://calendly.com/huan-lye/584-spring24-inperson
  - Virtual: https://calendly.com/huan-lye/ece-cs-584-office-hours-virtual



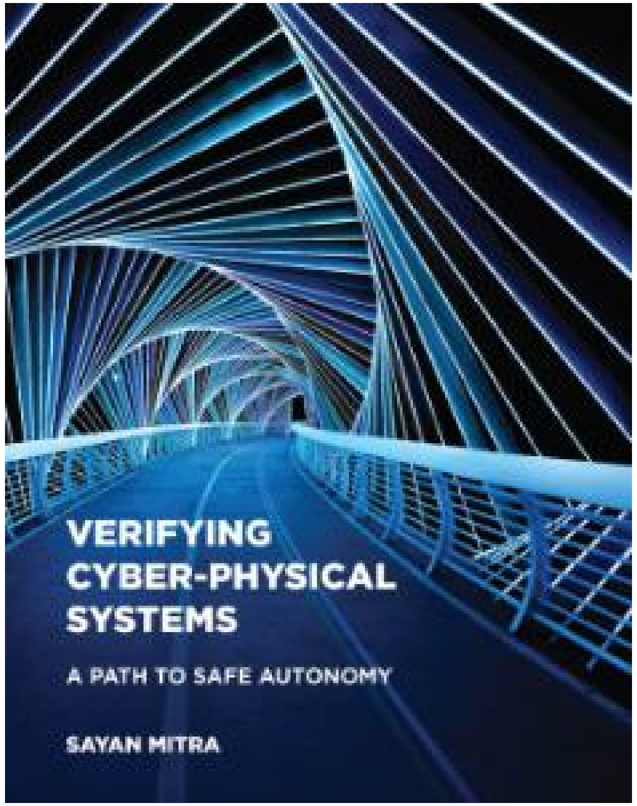
Grading

Homework 50%

Project 45%

Participation 5%

No exams. Spend time on making a successful project!



Think about course project!

https://publish.illinois.edu/ece584-spring2024/project-jumpstart/