

# CS:5980

# Foundations of Embedded Systems

## Introduction

*Copyright 2014-26, Rajeev Alur and Cesare Tinelli.*

*Created by Cesare Tinelli at the University of Iowa from notes originally developed by Rajeev Alur at the University of Pennsylvania. These notes are copyrighted materials and may not be used in other course settings outside of the University of Iowa in their current form or modified form without the express written permission of one of the copyright holders. During this course, students are prohibited from selling notes to or being paid for taking notes by any person or commercial firm without the express written permission of one of the copyright holders.*

# From Desktops to Cyber-Physical Systems

**Traditional computers:** Stand-alone device running software applications

- E.g., data processing

**Traditional controllers:** Devices interacting with physical world via sensors and actuators

- E.g., thermostat

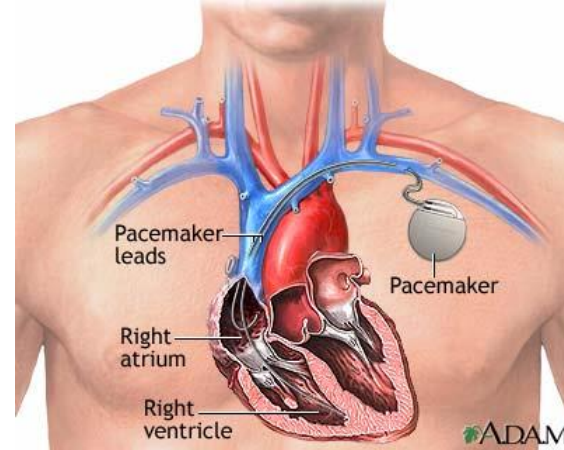
**Embedded** (aka Cyber-physical) **Systems:** Special-purpose system with integrated microcontroller/software

- E.g., cameras, watches, washing machines, ...

# Cyber-Physical Systems



Driverless Cars

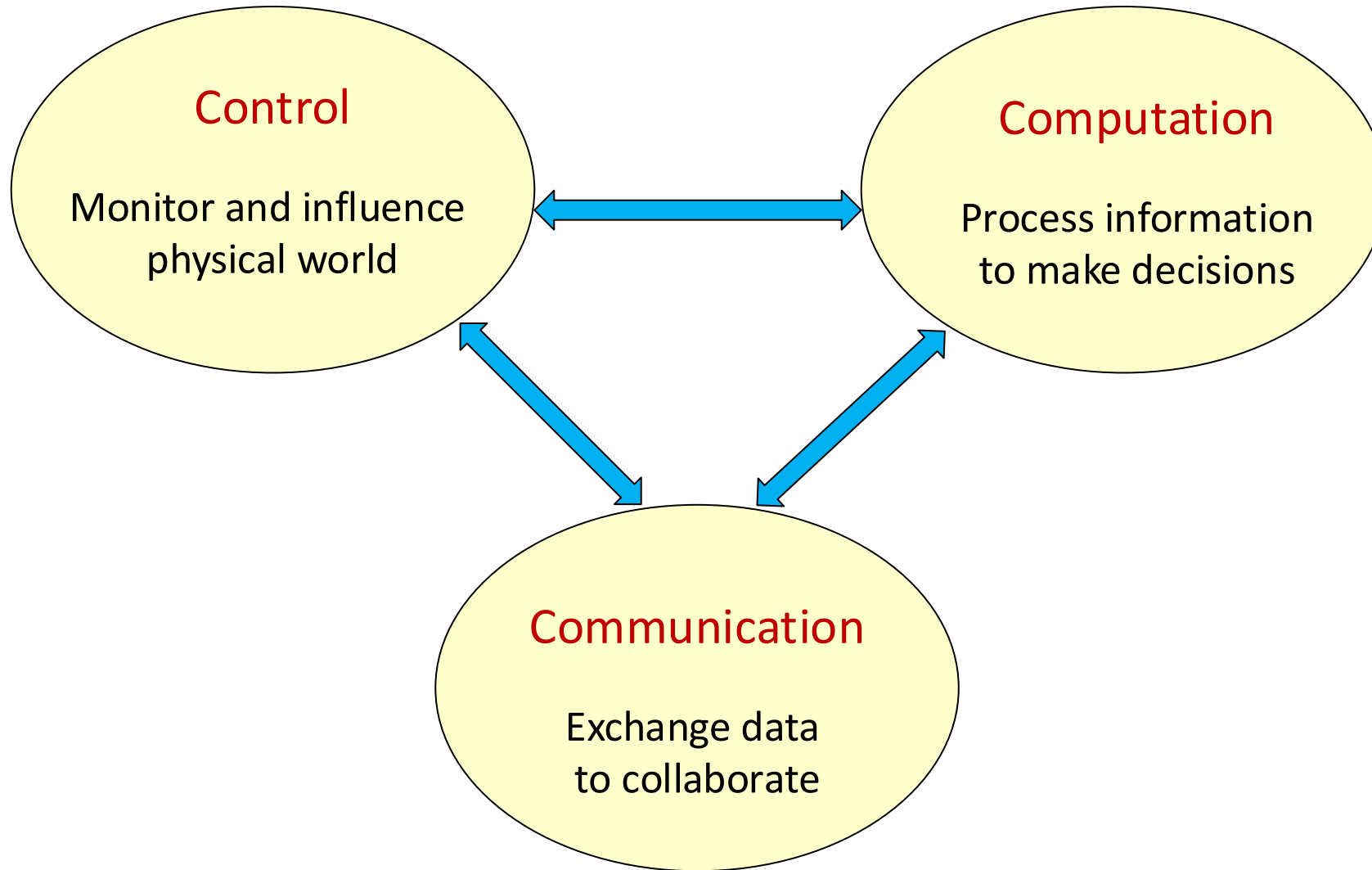


Medical devices



Coordinating robots

# Cyber-Physical Systems



# Design of Cyber-Physical Systems

Systems that integrate control, computation, and communication can do

- cool things and
- useful things

Lots of promise and potential: medicine, transportation, energy, ...

So, what's the main challenge?

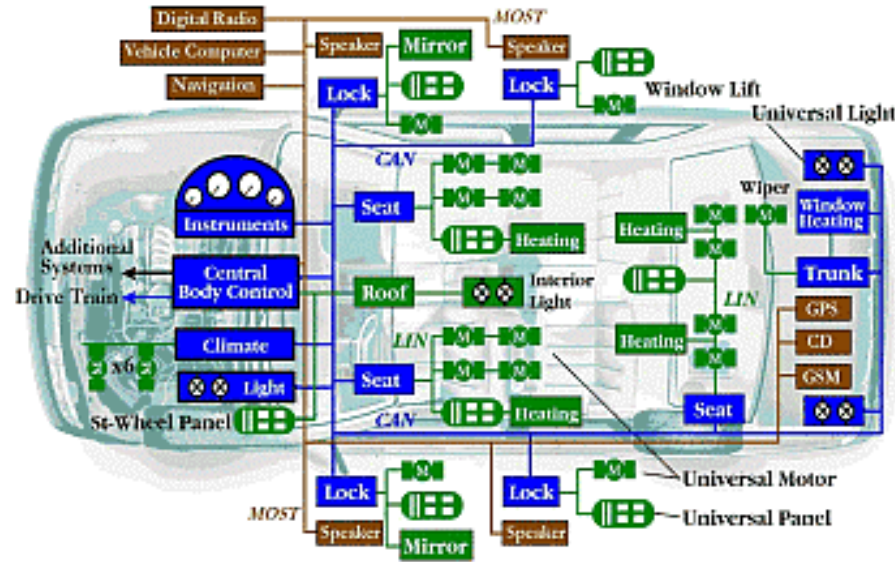
# Ariane 5 Explosion



“It took the European Space Agency 10 years and \$7 billion to produce Ariane 5. All it took to explode that rocket less than a minute into its maiden voyage last June, scattering fiery rubble across the mangrove swamps of French Guiana, was a small computer program trying to stuff a 64-bit number into a 16-bit space.”

A bug and a crash, J. Gleick, New York Times, Dec 1996

# Prius Brake Problems Blamed on Software Glitches



“Toyota officials described the problem as a "disconnect" in the vehicle's complex anti-lock brake system (ABS) that causes less than a one-second lag. With the delay, a vehicle going 60 mph will have traveled nearly another 90 feet before the brakes begin to take hold.”

CNN Feb 4, 2010

# Software: The Achilles' Heel

Software everywhere means bugs everywhere

2002 study by NIST:

Software bugs cost US economy \$60 billion annually (0.6% of GDP)

Lack of trust in software as technology barrier

Would you use an autonomous software-controlled round-the-clock monitoring and drug-delivery device?

# Software: The Achilles' Heel

Software everywhere means bugs everywhere

2002 study by NIST:

Software bugs cost US economy \$60 billion a

Lack of trust in software as technology barrier

Would you use an autonomous software-con



delivery device?

A grand challenge for computer science:

Technology for designing **reliable** cyber-physical systems

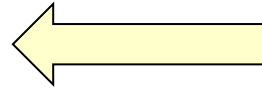
# Designing a Cruise Controller



What's the goal of a cruise controller?

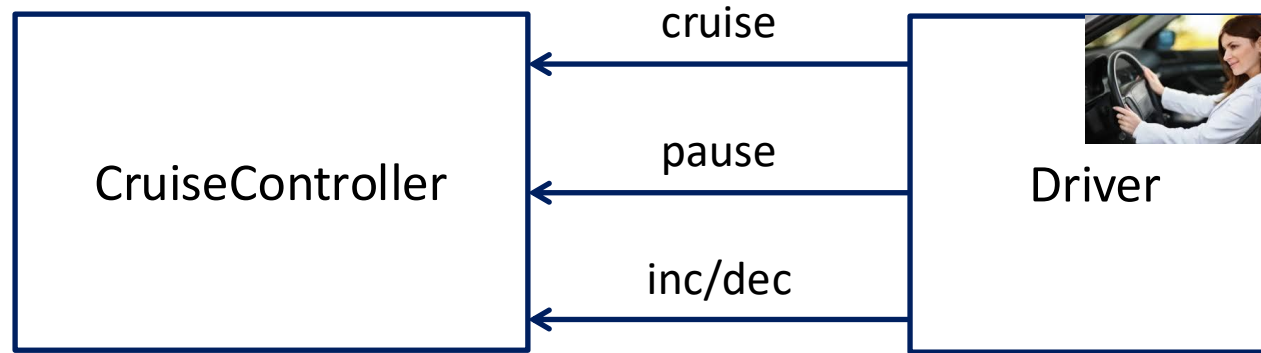
Automatically adjust the speed of the car so that it matches the speed desired by the driver

# Block Diagrams of High-Level Design



How does this component interact with the rest of the world ?

# Interfaces for Components: Inputs and Outputs



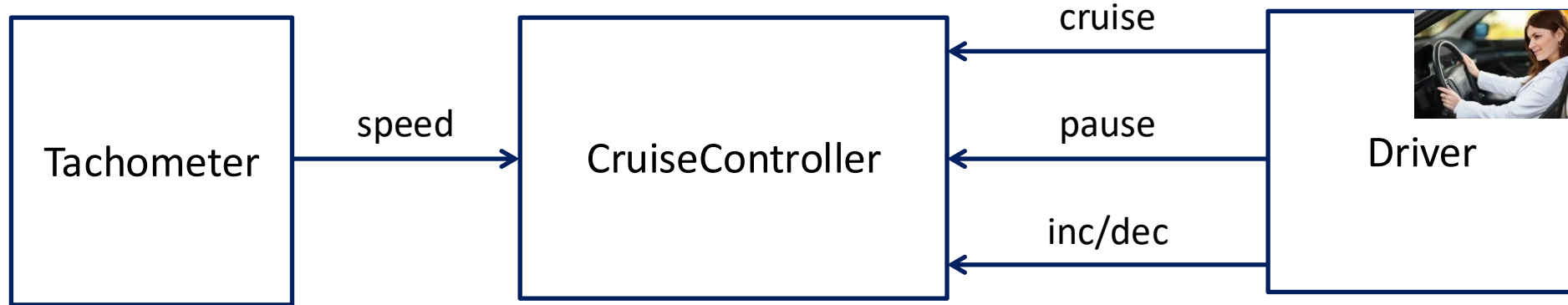
Driver interacts with the system using 4 buttons:

**Cruise** button to turn cruise control on or off

**Pause** button to suspend/resume its operation

**Inc** and **Dec** buttons to increment or decrement desired speed

# Interfaces for Components: Inputs and Outputs

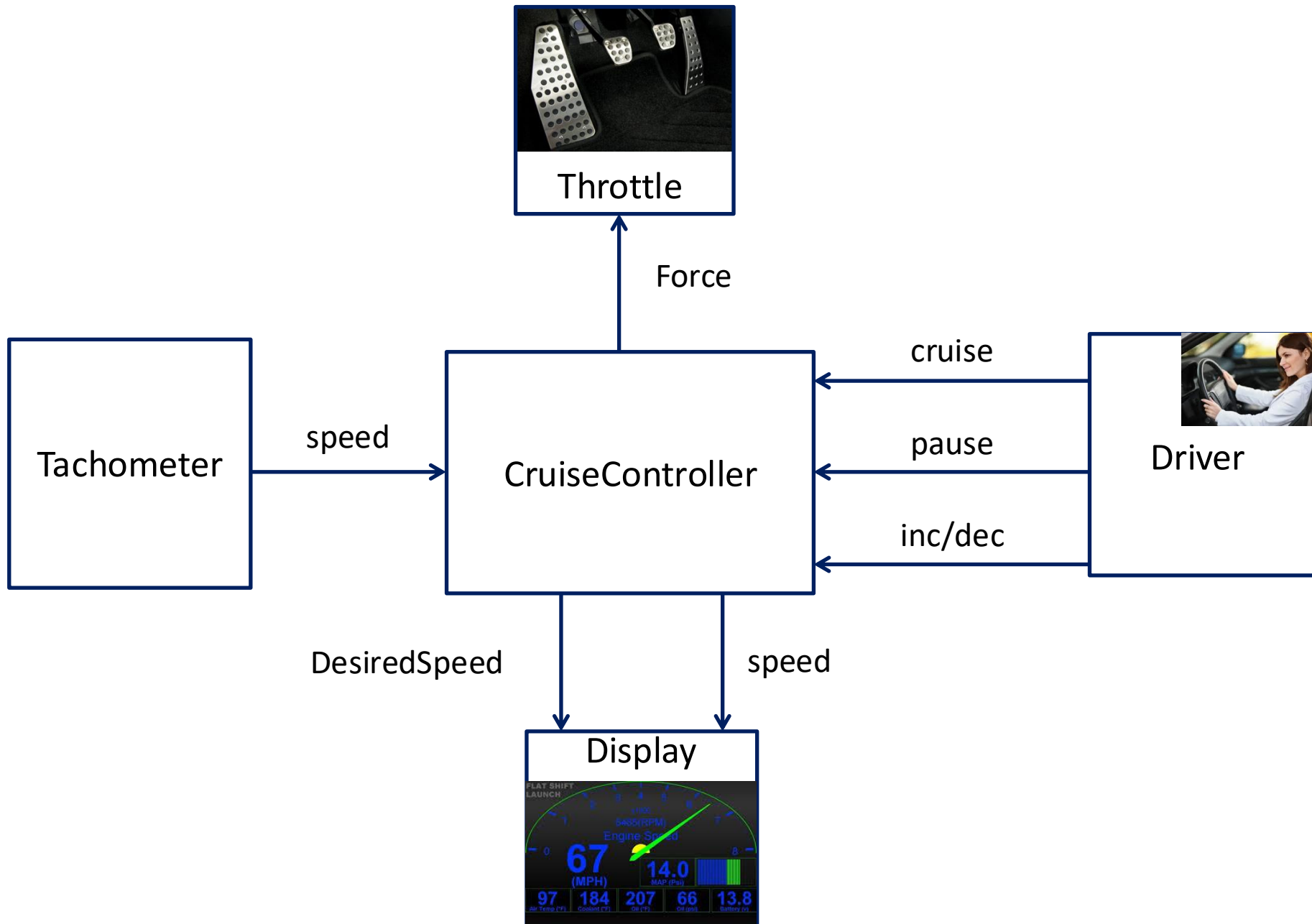


What other information does the cruise controller need?

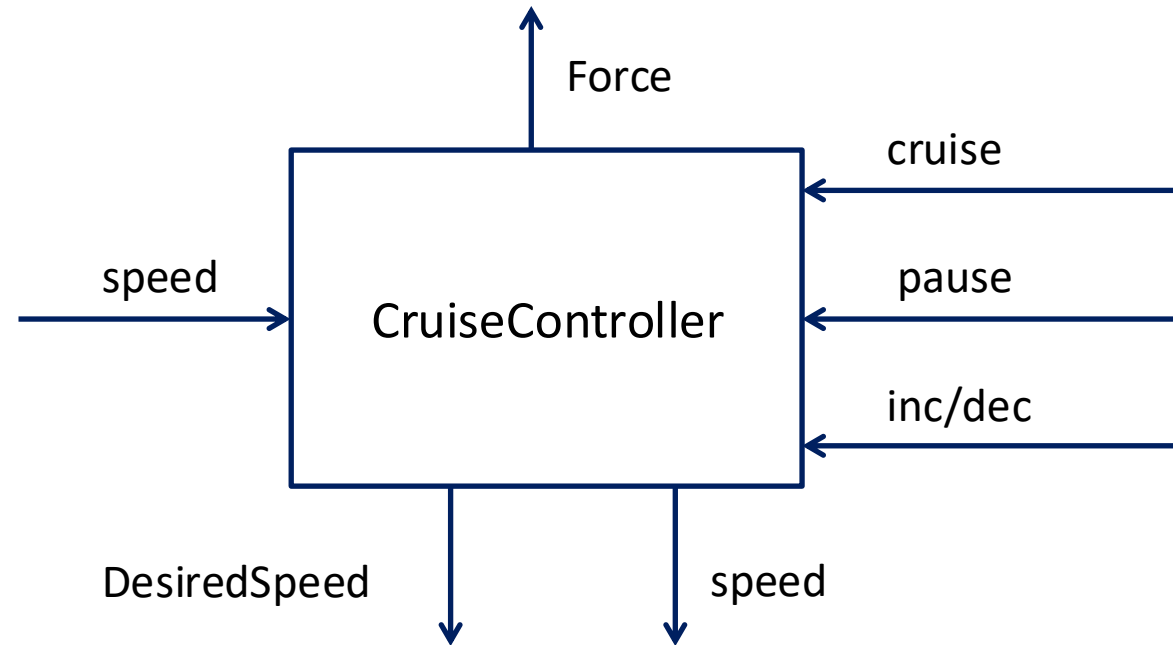
And who supplies it?

What should be the outputs of the cruise controller?

And who needs these outputs?

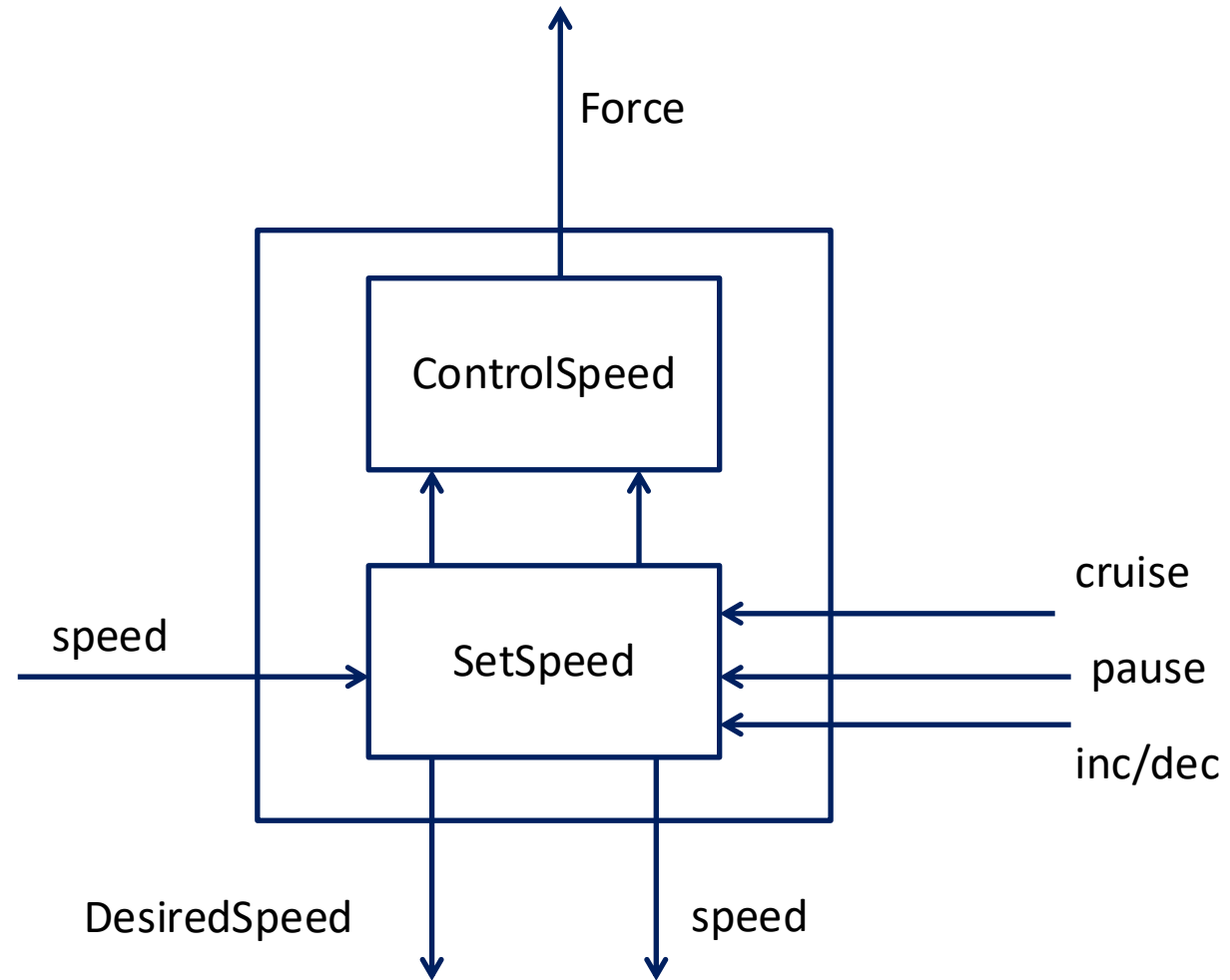


# Compositional Design

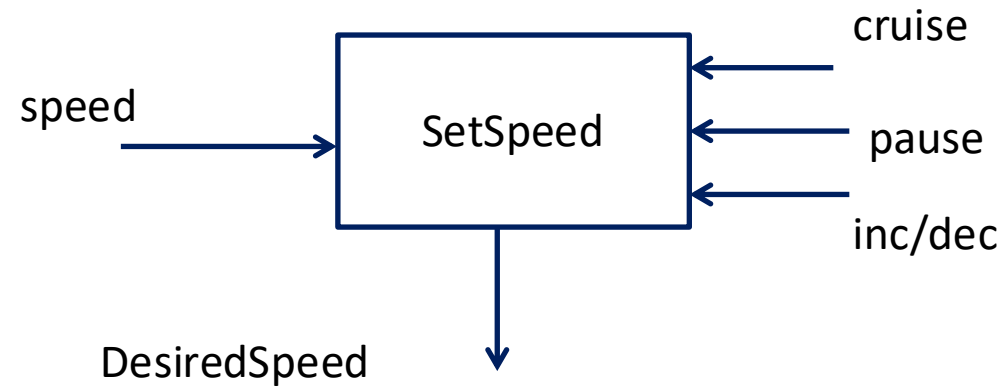


How to break up the computation of the cruise controller into subtasks?

# Decomposing the Cruise Controller

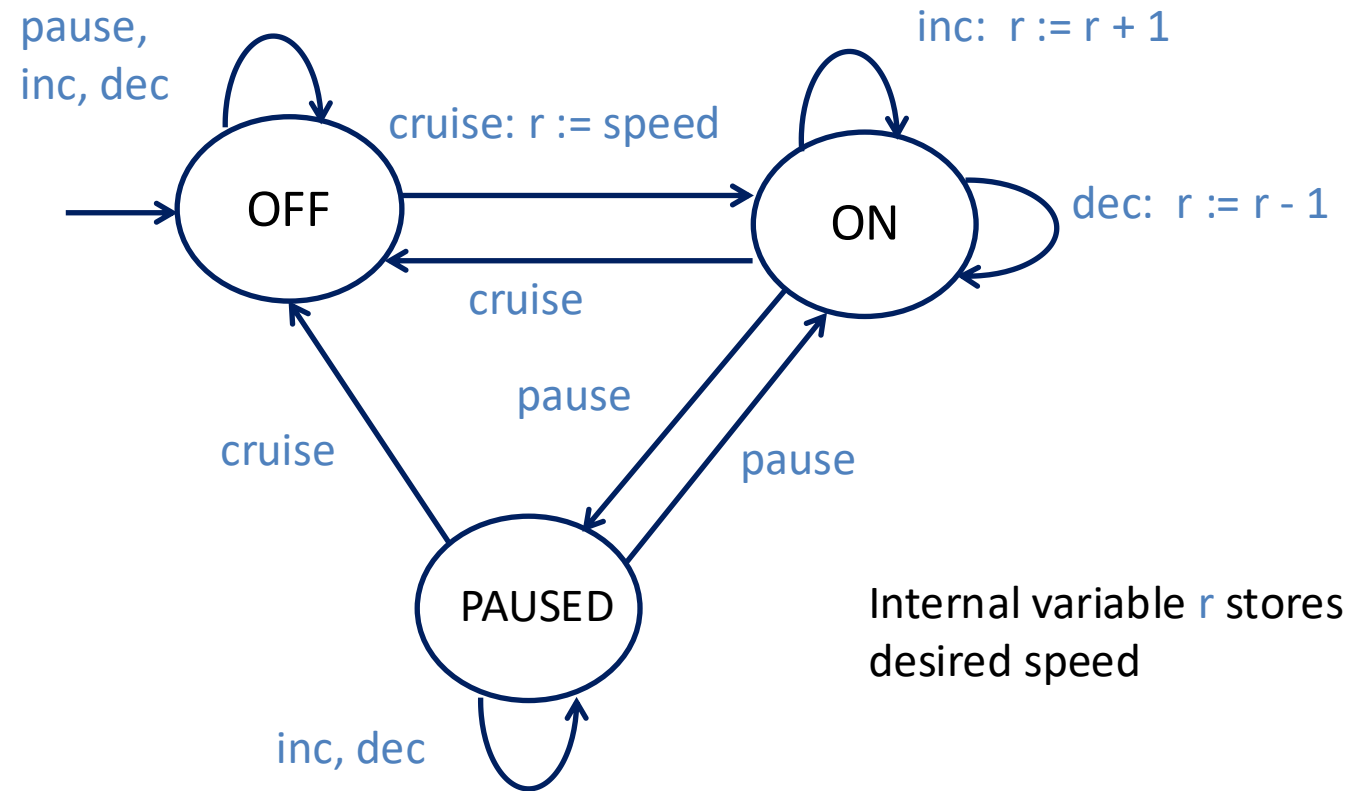


# Designing SetSpeed Component

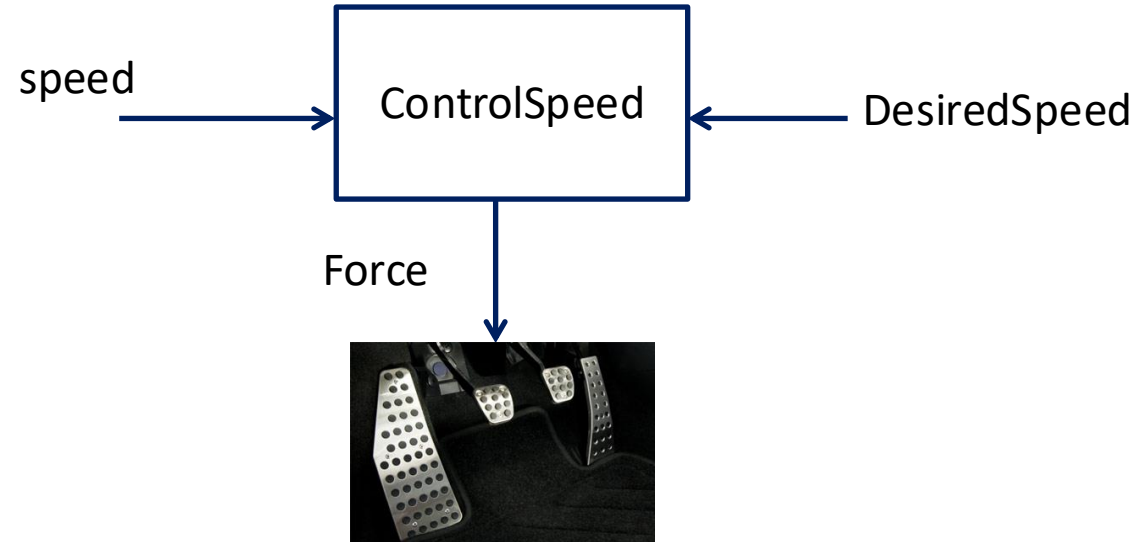


**Goal:** Compute the desired cruising speed in response to the commands from the driver

# Designing SetSpeed: State Machines

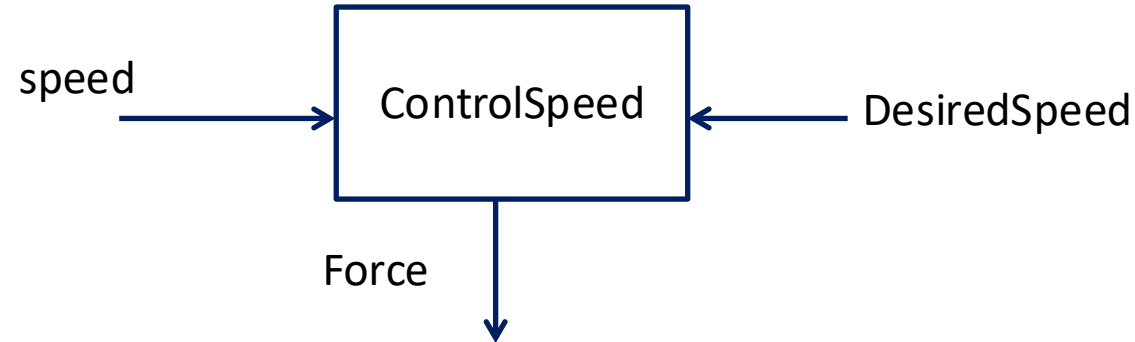


# Designing ControlSpeed Component



**Goal:** Determine the force to be applied to throttle so that speed becomes equal to DesiredSpeed

# Capturing Requirements



**Requirements:** mathematically precise description of what a system is supposed to do.

Writing requirements is key to ensuring reliability of systems

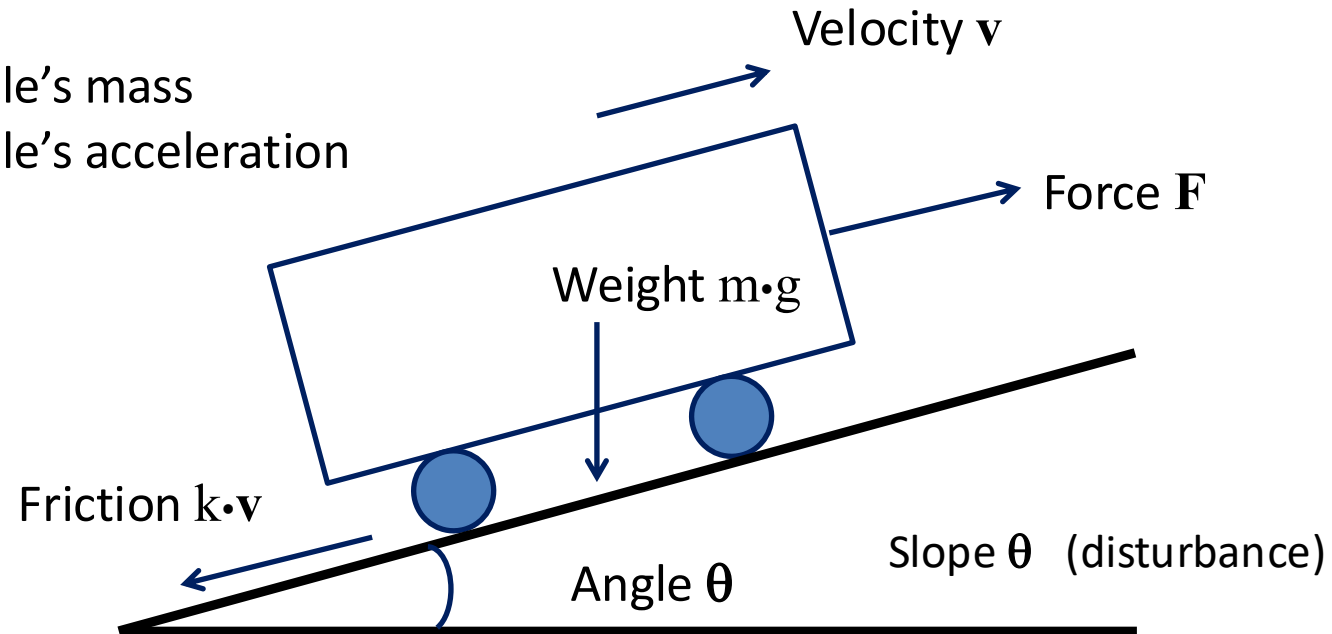
**Requirement 1:** Actual speed (velocity) eventually **converges** to desired speed

**Requirement 2:** After convergence, velocity remains **stable**

## A bit of Physics: Modeling a car

$m$  = vehicle's mass

$a$  = vehicle's acceleration

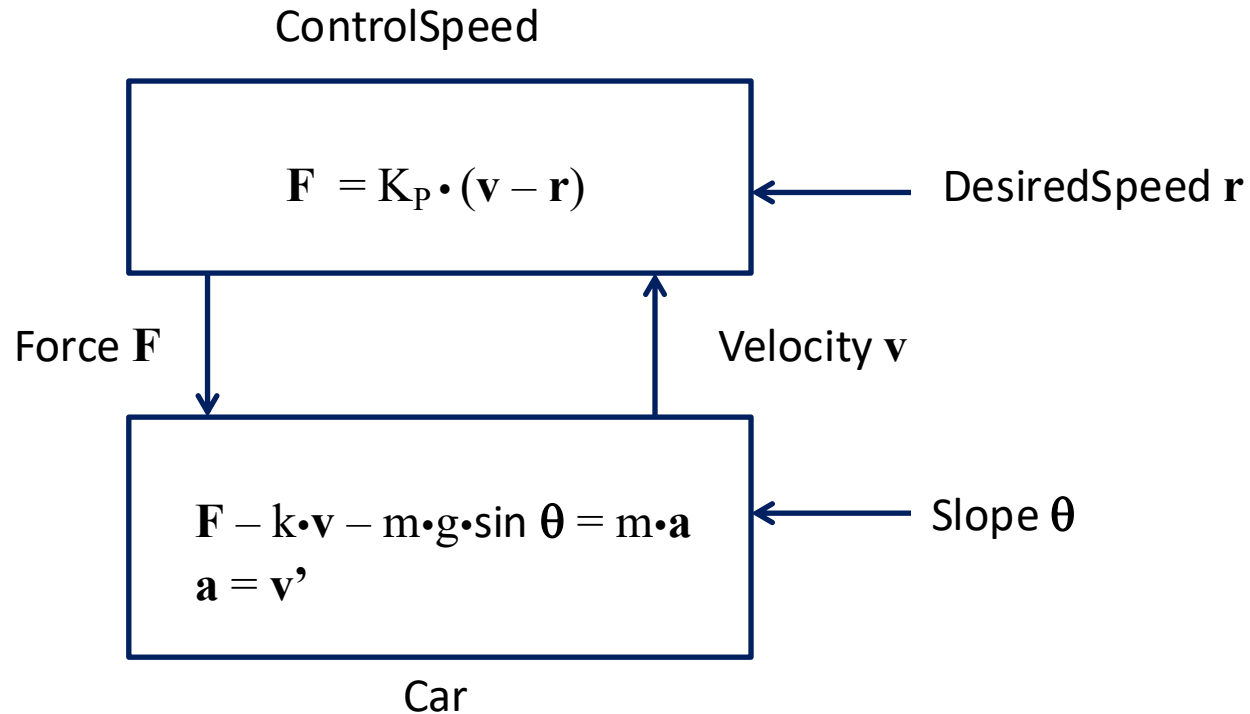


Newton's law of motion gives

$$F - k \cdot v - m \cdot g \cdot \sin \theta = m \cdot a$$

$$a = v'$$

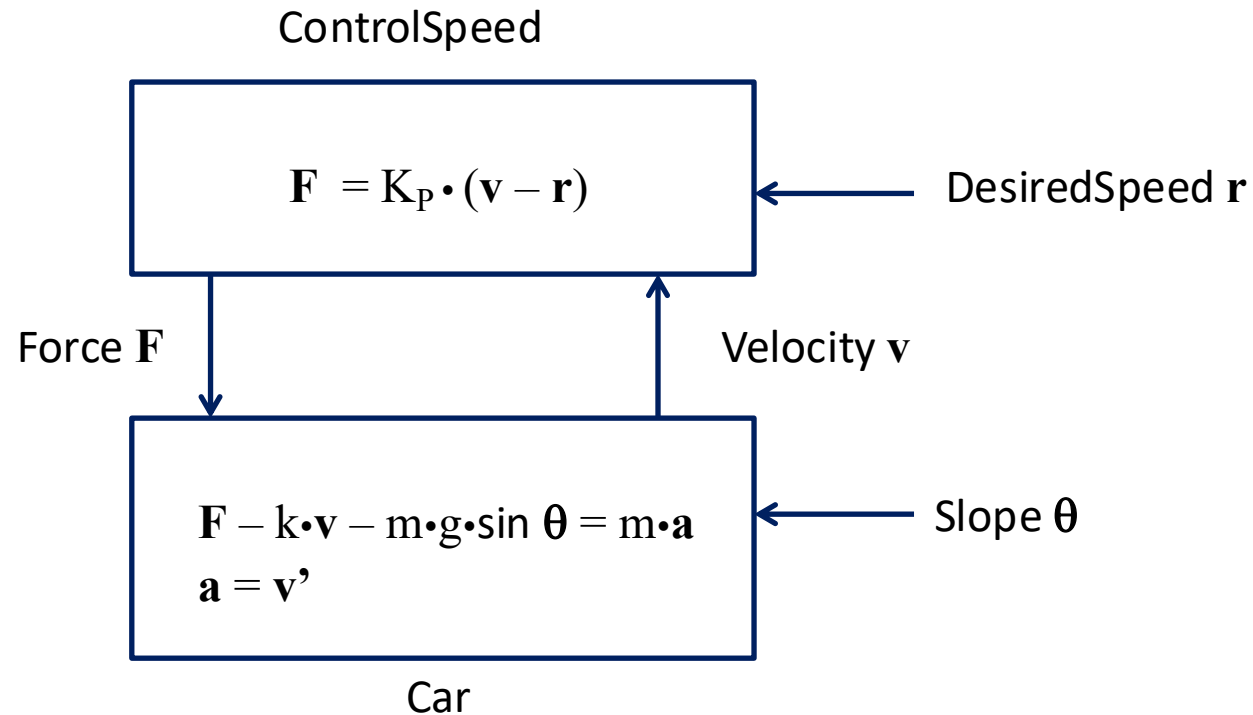
# ControlSpeed Component



## Control Theory:

Mathematical techniques to compute force ( $\mathbf{F}$ ) as a function of velocity ( $\mathbf{v}$ ) and desired speed ( $\mathbf{r}$ )

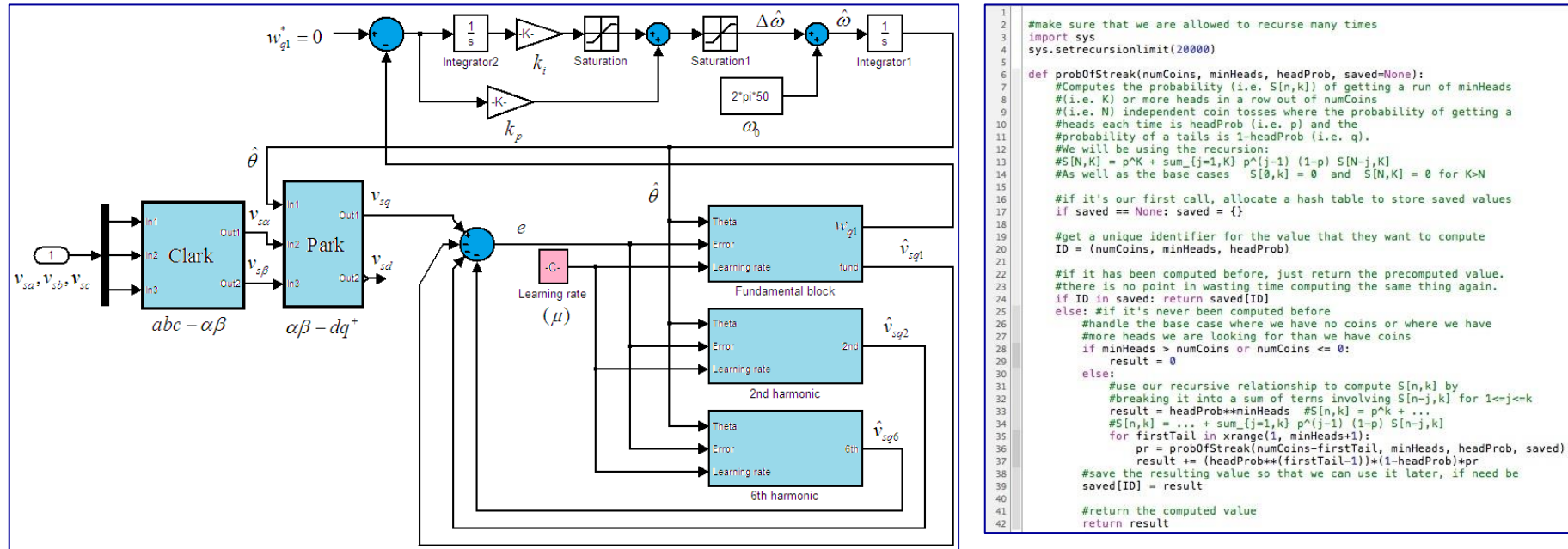
## How well does our controller work?



### Verification Tools:

Allow you to check if system model indeed works as expected, that is, satisfies requirements

# Model-based design != Coding



Design using high-level block diagrams and state machines gets **automatically** compiled to low-level code!

*It models not only system being designed, but also of its environment*

# Verification != Simulation/Testing



“Program testing can be used to show the presence of bugs, but never their absence!”

Edsger W. Dijkstra

# Formal Verification



**Goal:** Establish that model satisfies requirements under all possible scenarios

**First challenge:** Need formal definitions of **model** and **requirement** to make the problem mathematically precise

**Second challenge:** Need verification techniques and tools

# Course Topics

**Goal:** Introduction to principles of design, specification, analysis and implementation of CPS

## Disciplines:

- Model-based design
- Concurrency theory
- Distributed algorithms
- Formal specification
- Verification techniques and tools
- Control theory
- Real-time systems
- Hybrid systems

Emphasis on mathematical concepts

# Theme 1: Formal Models

Mathematical abstractions to describe system designs

## Modeling formalisms

- Synchronous models (Chapter 2 of textbook)
- Asynchronous models (Chapter 4)
- Continuous-time dynamical systems (Chapter 5)
- Timed models (Chapter 7)
- Hybrid systems (Chapter 9)

## Modeling concepts

- Syntax vs. semantics
- Composition
- Input/output interfaces
- Nondeterminism, fairness, ...

## Theme 2: Specification and Analysis

Formal techniques to ensure correctness at design time

### Requirements

- Safety (invariants, monitors)
- Liveness (temporal logic, automata over infinite sequences)
- Stability
- Schedulability

### Analysis techniques

- Deductive based on inductive invariants and ranking functions
- Enumerative and symbolic search for state-space exploration
- Model checking
- Linear-algebra-based analysis of dynamical systems
- Verification of timed and hybrid systems

# Theme 3: Model-based Design

Design and analysis of illustrative computing problems

## Design methodology

- Structured modeling (bottom-up, top-down)
- Requirements-based design and design-space exploration

## Case studies

- Distributed coordination (mutual exclusion, consensus, leader election)
- Communication (reliable transmission, synchronization)
- Control design (PID, cruise controller)
- CPS (pacemaker, obstacle avoidance for robots, multi-hop control network)