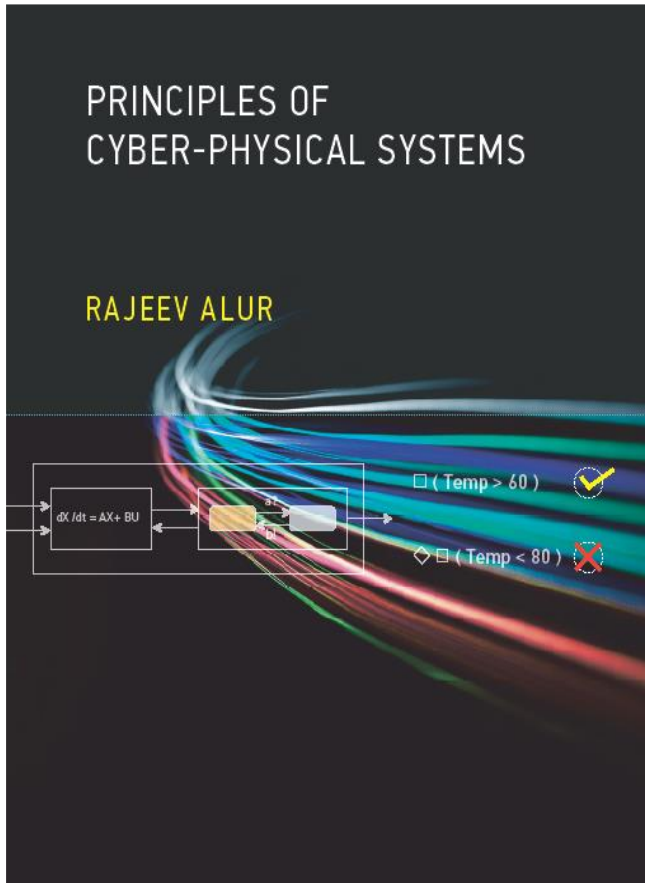
Models and Tools for Cyber-Physical Systems

Chapter 1: Introduction

Instructors: Suman Sourav, Jonathan Julian Huerta y Munive
{sumansourav, jjhymuni}@cs.aau.dk

Slides courtesy of Rajeev Alur
alur@cis.upenn.edu

Models and Tools for Cyber-Physical Systems



Reference book

UPPAAL 5.0.0
Uppsala University and Aalborg University.

Tools

Course Organization and Contents

Session	Activity	Contents	Instructors
1	Lecture	Introduction	Suman
	Exercises 1	Synchronous Model	
2	Lecture		
	Exercises 2	Chapters 1, 2	
3	Lecture	Safety Requirements	Jonathan
	Exercises 3		
4	Lecture	Asynchronous Model	Suman
	Exercises 4		
5	Lecture	Chapters 3, 4	
	Exercises 5		
6	Lecture	Timed Model	Suman
	Exercises 6		
7	Lecture	Real Time Scheduling	
	Exercises 7		
8	Lecture	Chapters 7, 8	
	Exercises 8		

Session	Activity	Contents	Instructors
9	Mini project	Uppaal	Suman
10	Lecture	Dynamical Systems	Jonathan
	Exercises 9		
11	Lecture	Hybrid Systems	
	Exercises 10		
12	Lecture	Chapters 6, 9	
	Exercises 11		
13	Mini project	Python	

This plan is tentative

Join at menti.com | use code 2629 6266

Mentimeter

What do you think a Cyber-Physical System means?

All responses to your question will be shown here

Each response can be up to 200 characters long

Turn on voting to let participants vote for their favorites

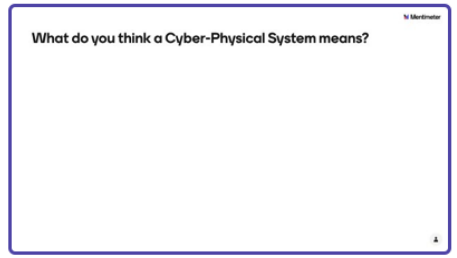
0 / 1

SS

Menti

New presentation

Choose a slide to present



From Desktops to Cyber-Physical Systems

- Traditional computers: Stand-alone devices running software applications (e.g., data processing)

- Traditional controllers: Devices interacting with physical world via sensors and actuators (e.g., thermostat)

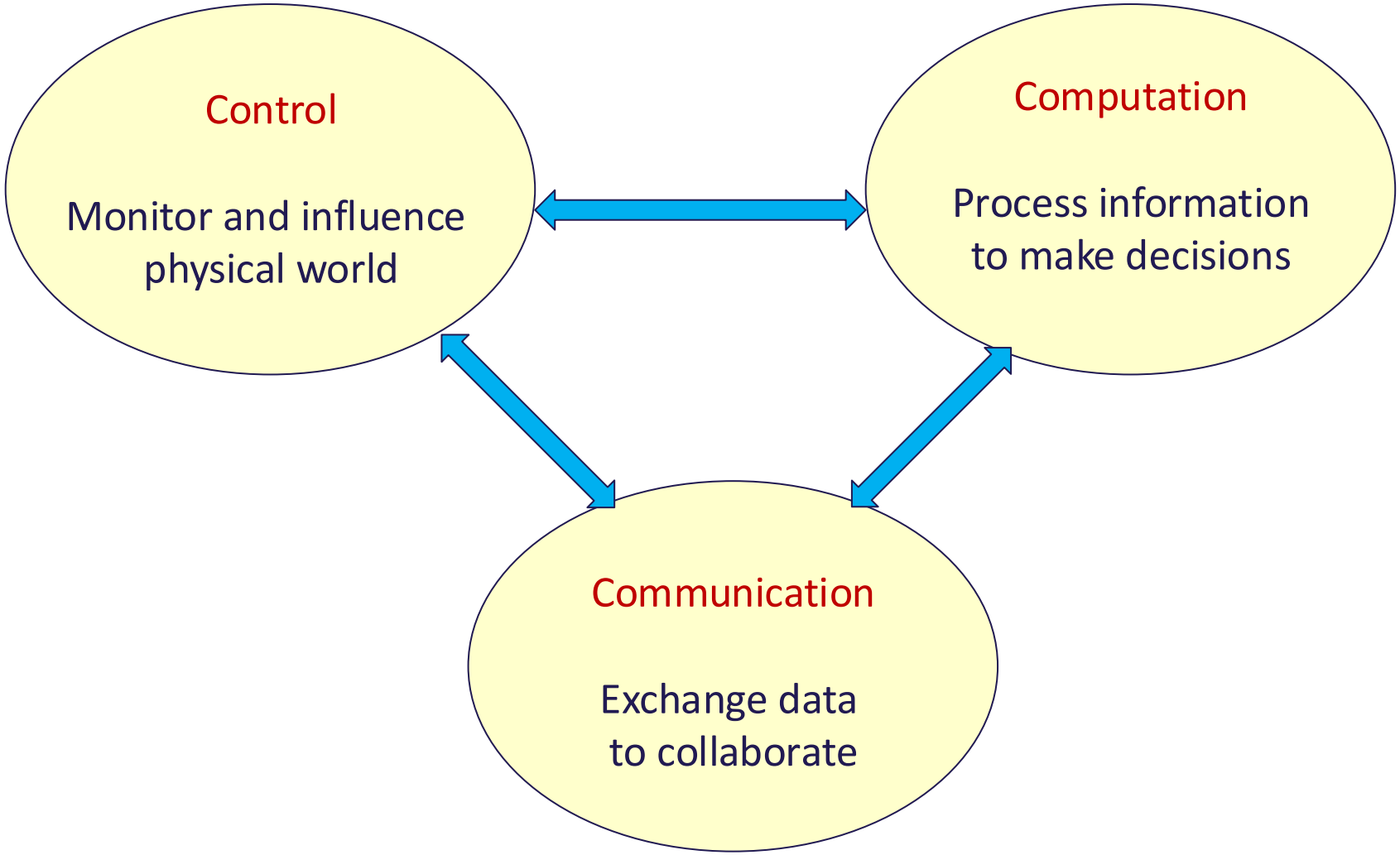
- Embedded systems
  - Special-purpose systems with integrated microcontroller/software
  - Cameras, watches, washing machines…

Embedded Systems Everywhere!

Software inside

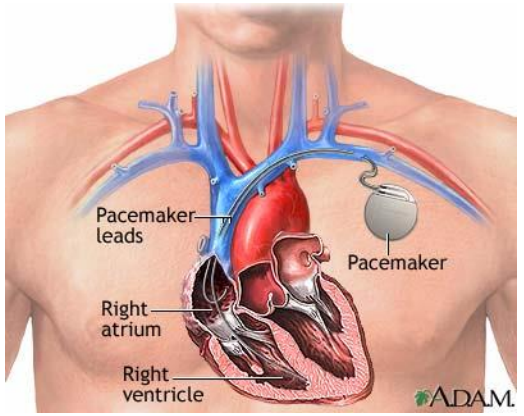
Embedded Systems Everywhere!

Cyber-Physical Systems

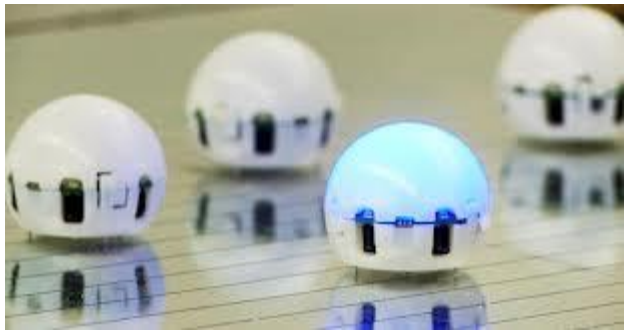


Cyber-Physical Systems

Driverless cars



Medical devices



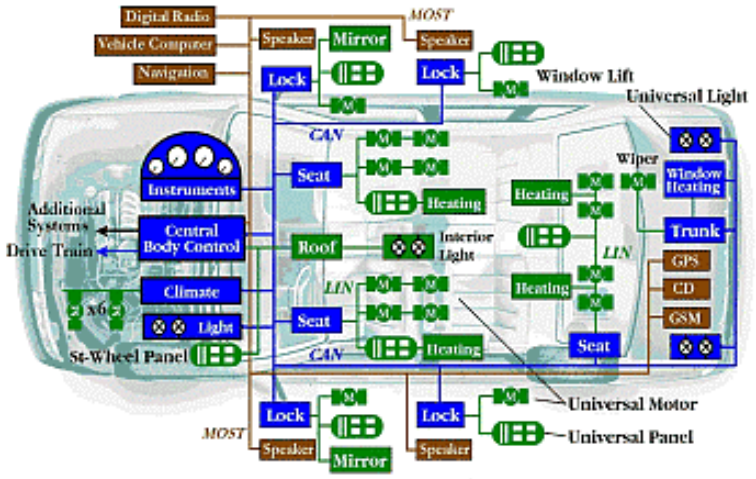
Coordinating robots

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)

2017 Tricentis study: $1.7 trillion in 314 companies

Lack of trust in software as technology barrier

Can we use an autonomous software-controlled round-the-clock monitoring and drug-delivery device?

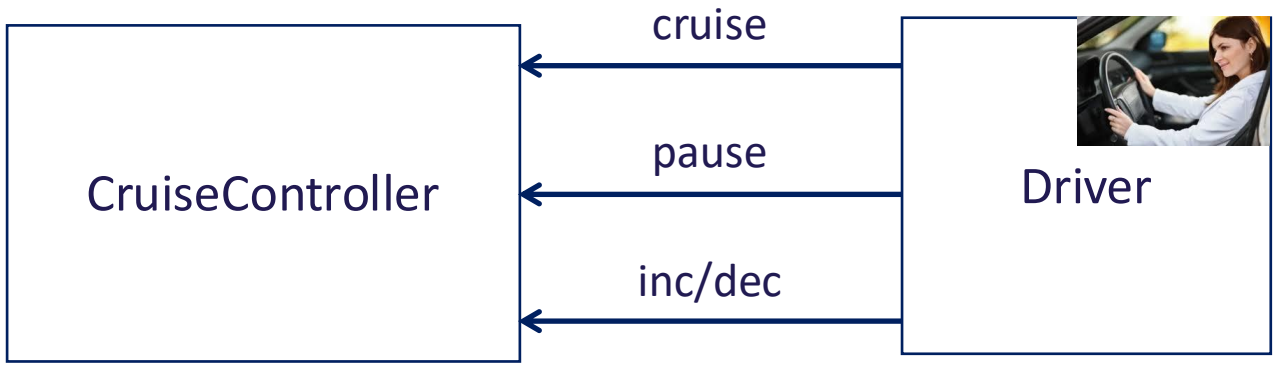
Grand challenge for computer science

Technology for designing reliable cyber-physical systems

Case study: Design a Cruise Controller

The goal of a cruise controller is to automatically adjust the speed of a car so that it matches the speed desired by the driver

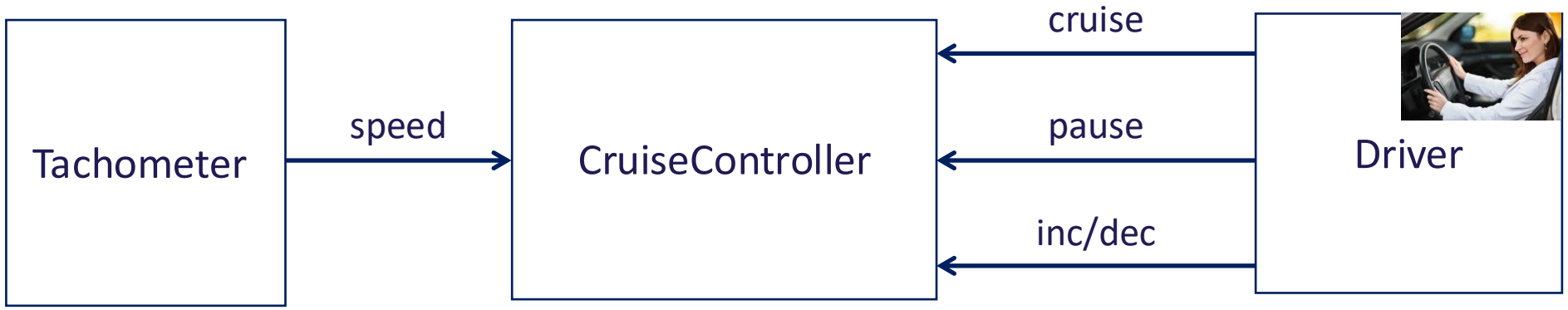
Interfaces for Components: Inputs and Outputs



Driver interacts with the system using 4 buttons:

- Cruise button to turn cruise control on or off
- Pause button to suspend/restart its operation
- Inc and Dec buttons to increment or decrement desired speed

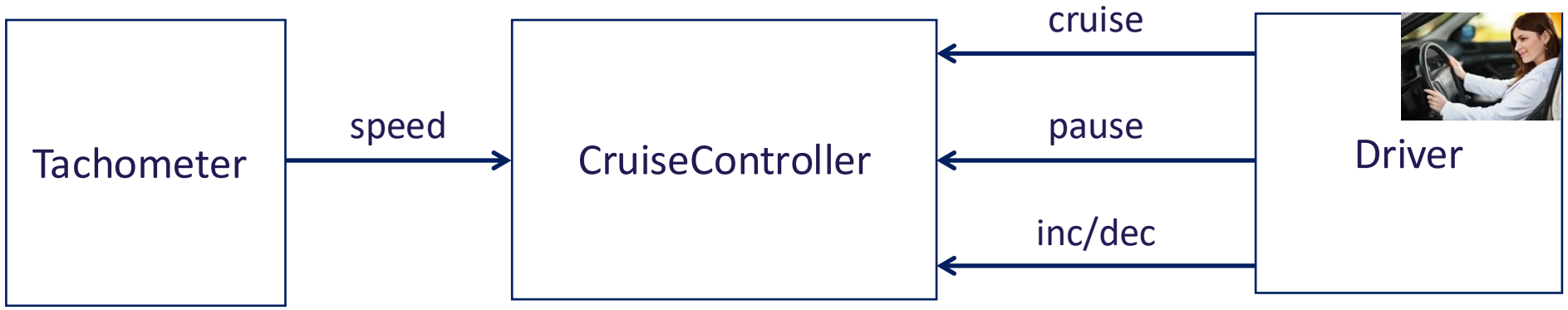
Interfaces for Components: Inputs and Outputs



What other information does the cruise controller need ?

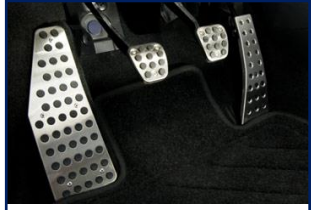
And who supplies it?

Interfaces for Components: Inputs and Outputs



What should be the outputs of the cruise controller?

And who needs these outputs?



Throttle

Force

Tachometer

speed

CruiseController

cruise

pause

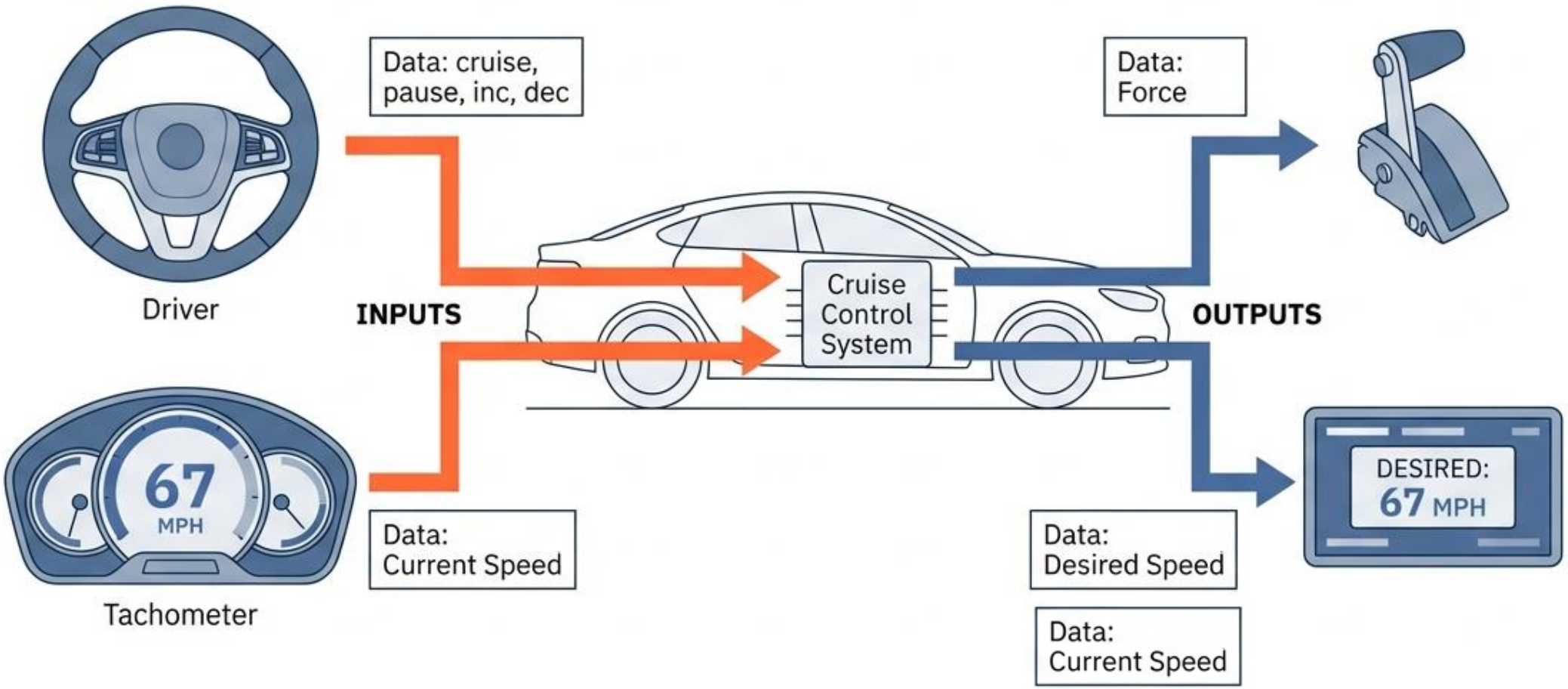
inc/dec

Driver

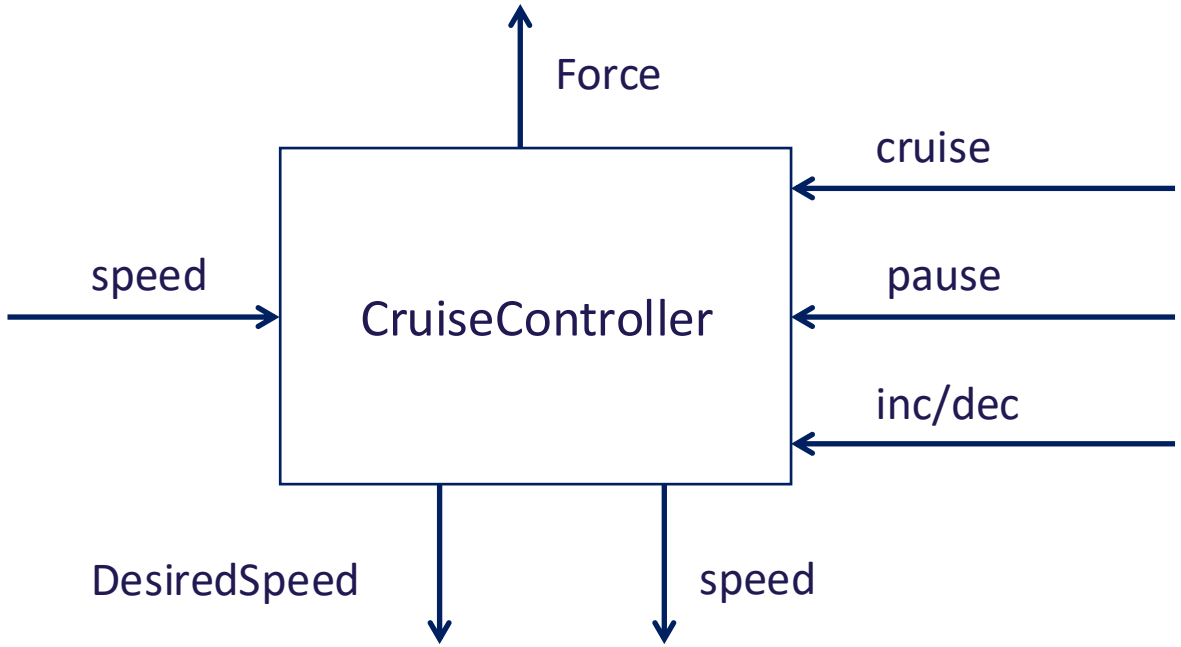
DesiredSpeed

speed

Display

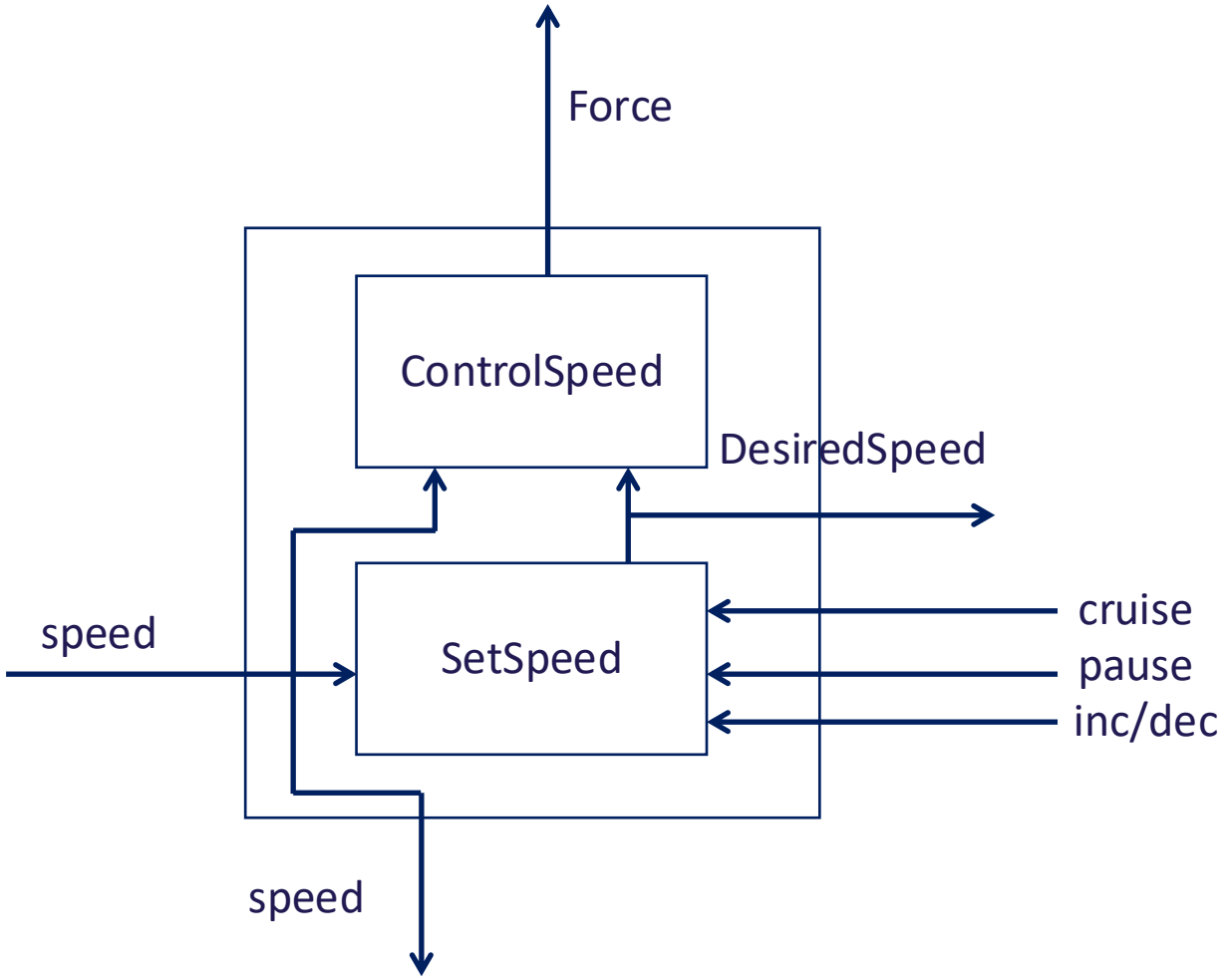

Data: cruise, pause, inc, dec
Data: Force
Driver
INPUTS
Cruise Control System
OUTPUTS
67 MPH
DESIRED: 67 MPH
Data: Current Speed
Data: Desired Speed
Data: Current Speed
Tachometer

Compositional Design



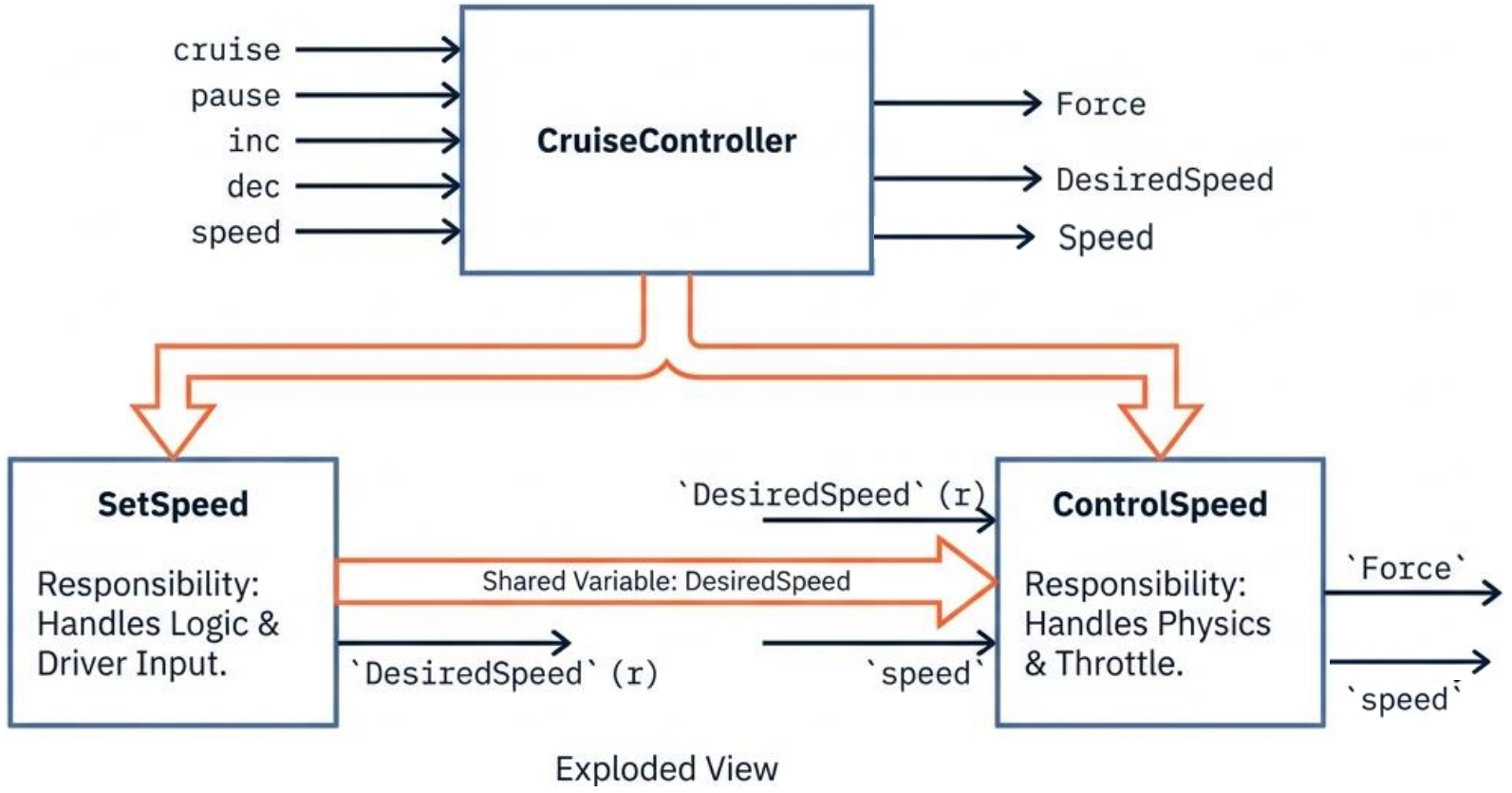
How to break up the computation of the cruise controller into subtasks?

Decomposing the Cruise Controller



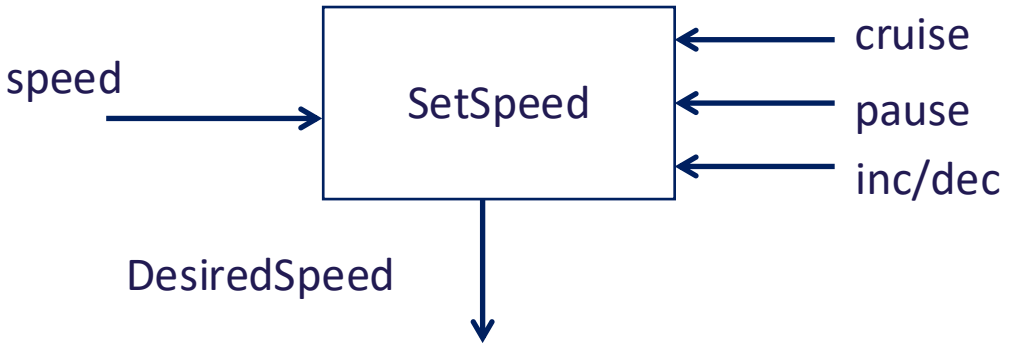
Decomposing the Cruise Controller

Breaking computation into manageable sub-tasks.



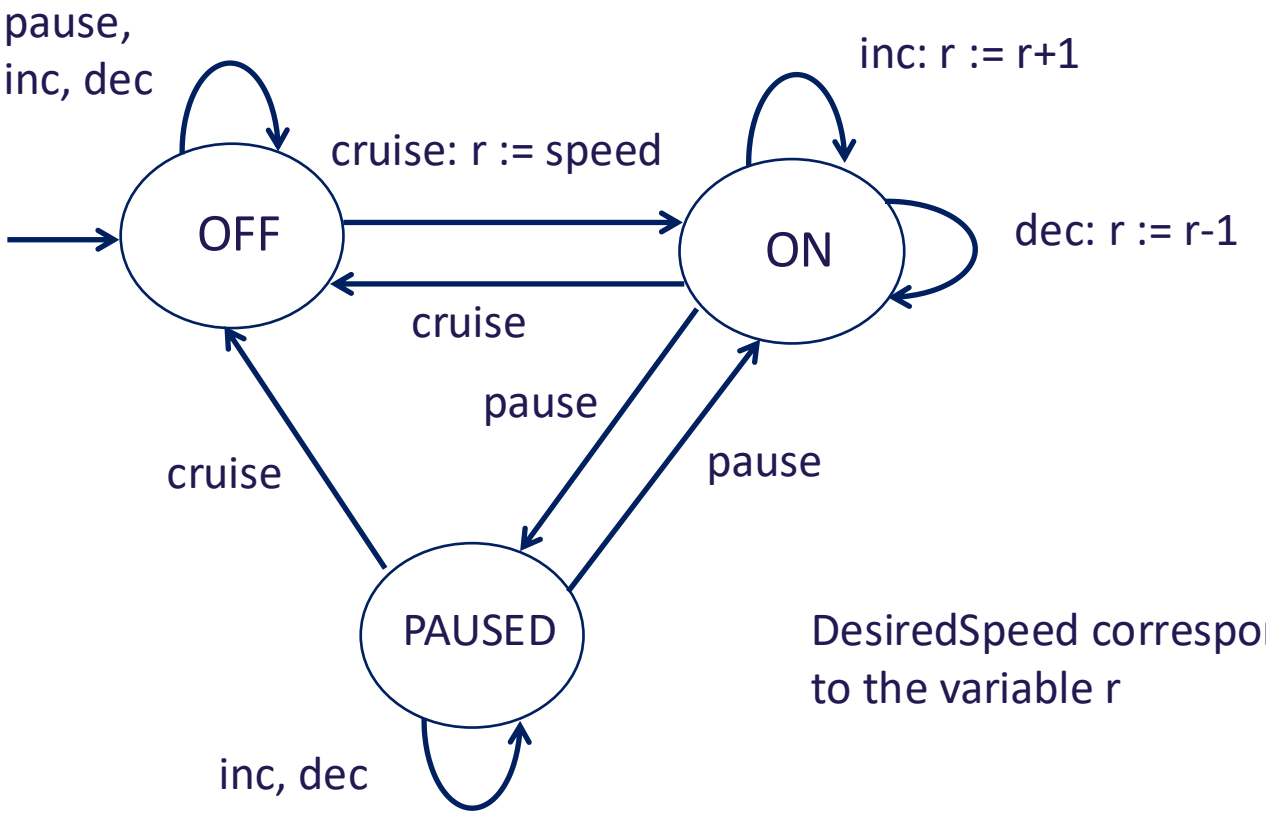
Exploded View

Designing SetSpeed Component

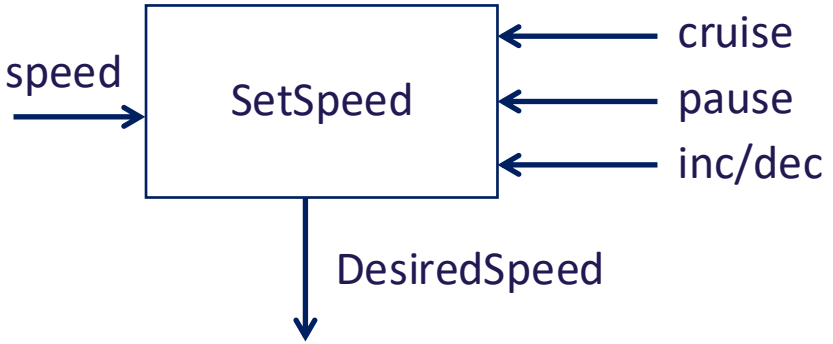


Goal: Compute the desired cruising speed in response to the commands from the driver

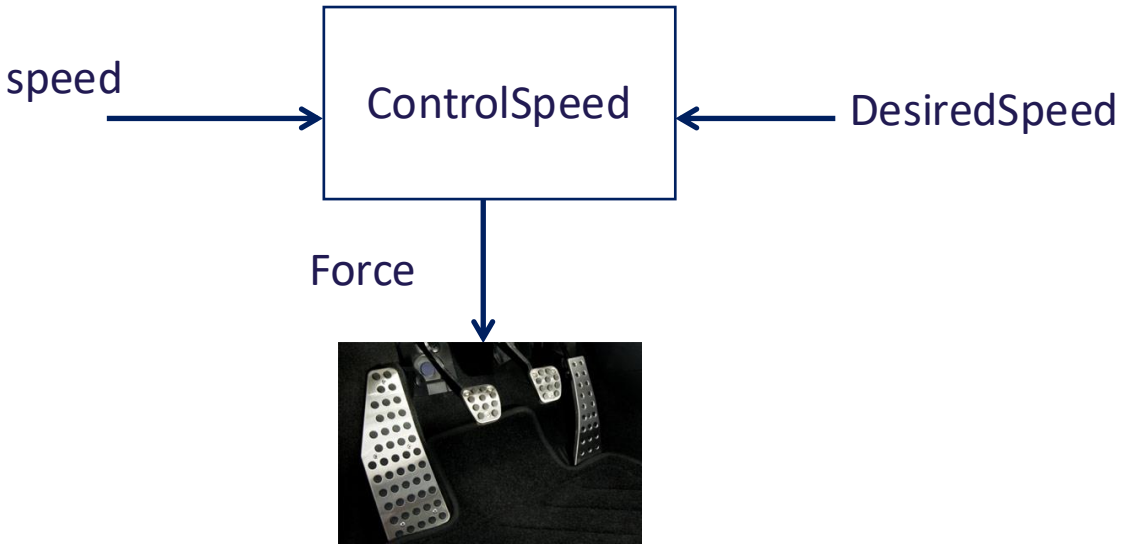
Designing SetSpeed: State Machine



DesiredSpeed corresponds to the variable r

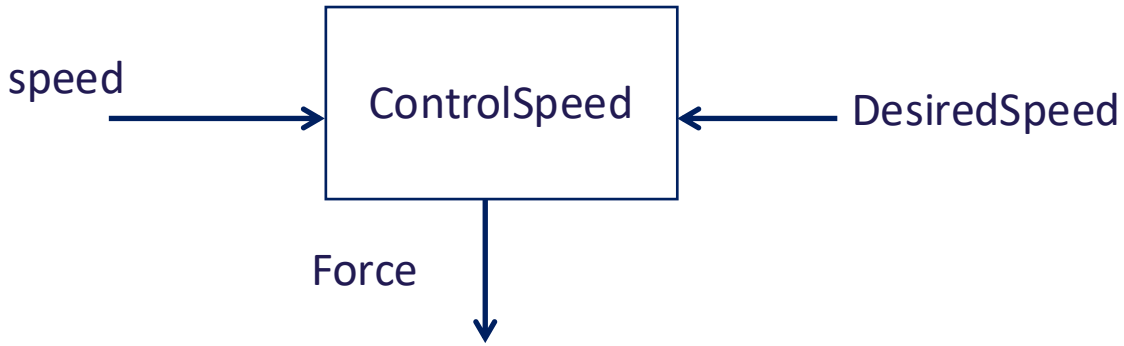


Designing ControlSpeed Component



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

Capturing Requirements



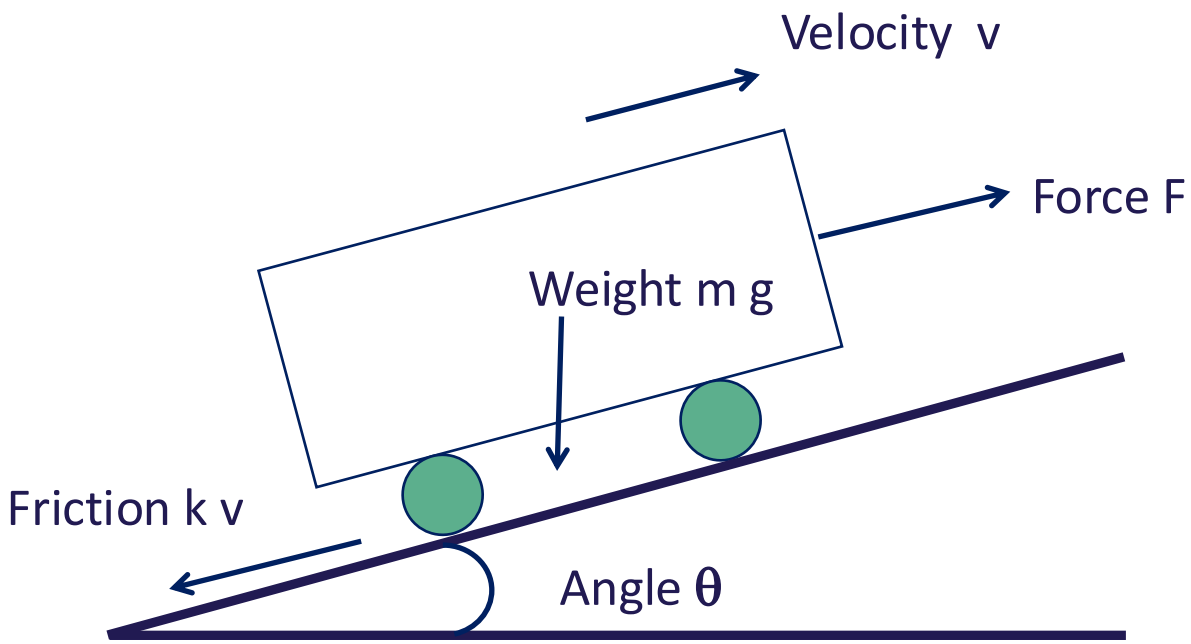
Requirements: Mathematically precise description of what a system is supposed to do.

Writing requirements is key to ensuring reliability of systems

Requirement 1: Actual speed eventually converges to desired speed

Requirement 2: Speed of the car stays “stable”

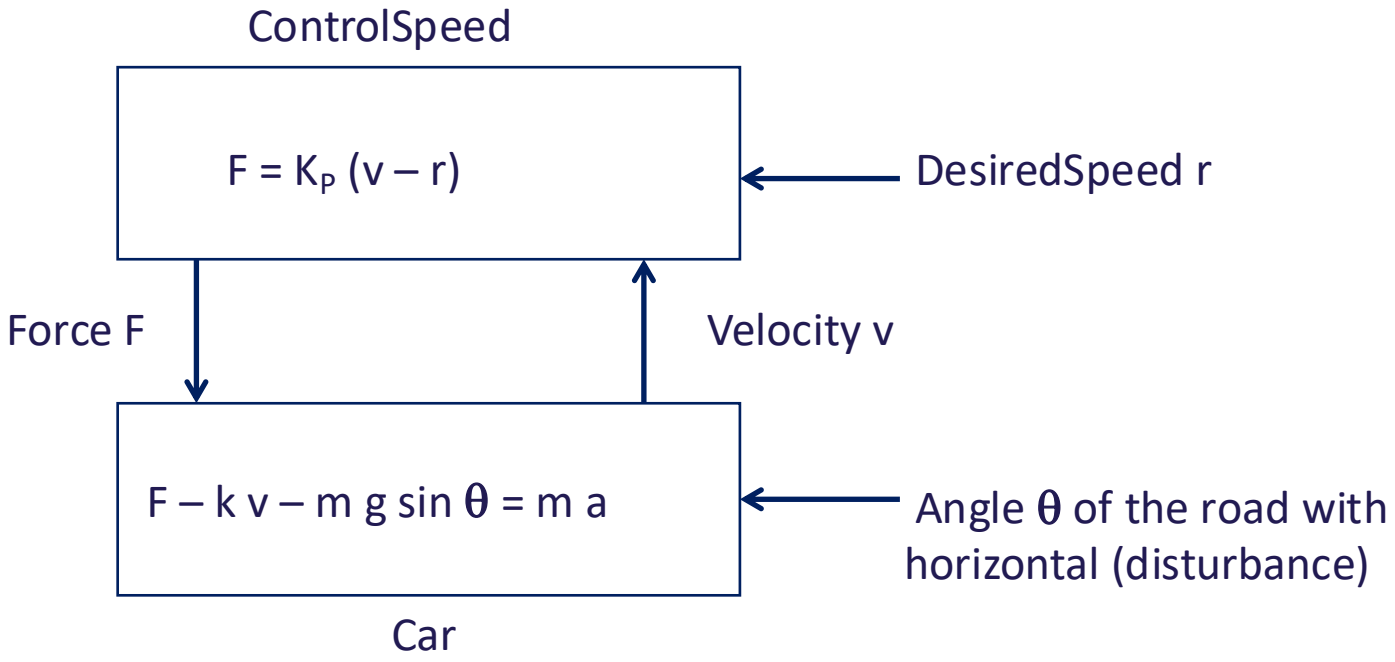
A Bit of Physics: Modeling a Car



Newton's law of motion gives

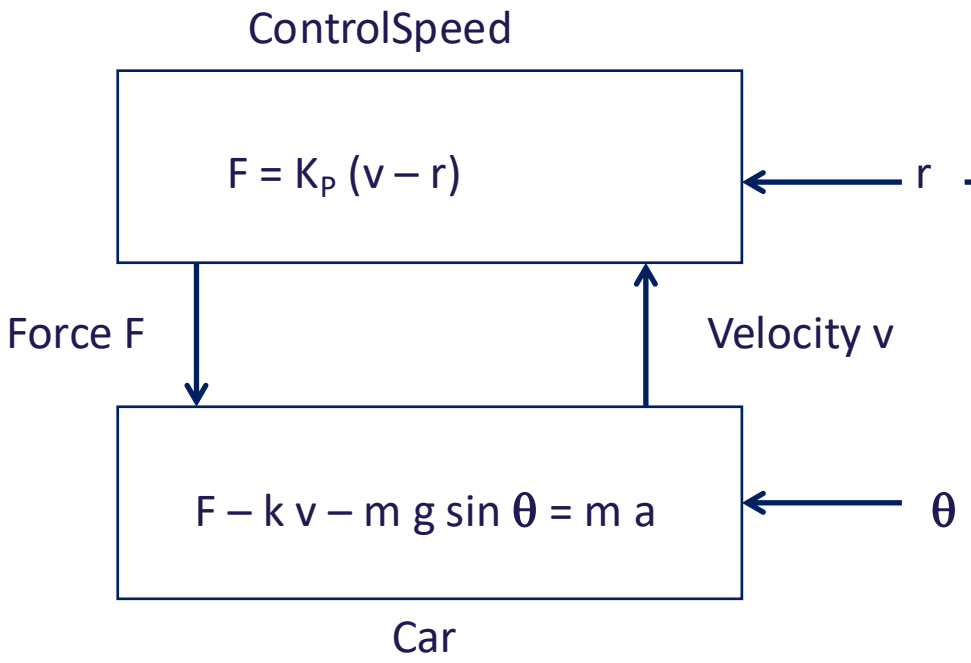
$$F - k\,v - m\,g\,\sin\theta = m\,a$$

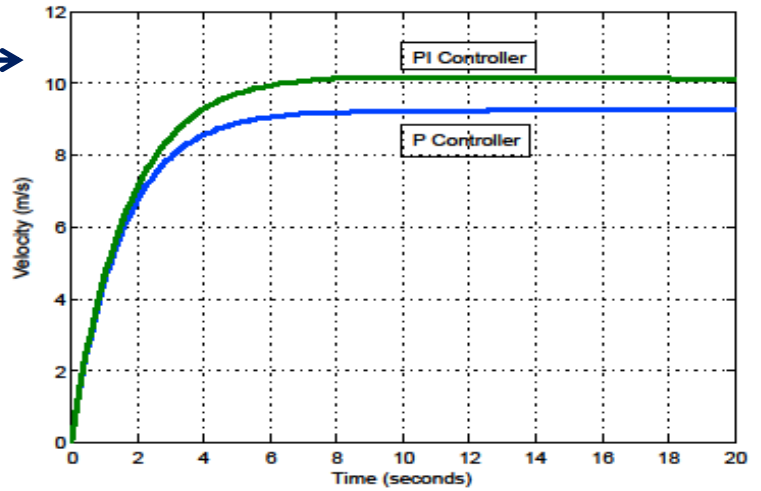
ControlSpeed Component



Control Theory: Mathematical techniques to compute force (F) as a function of velocity (v) and desired speed (r)

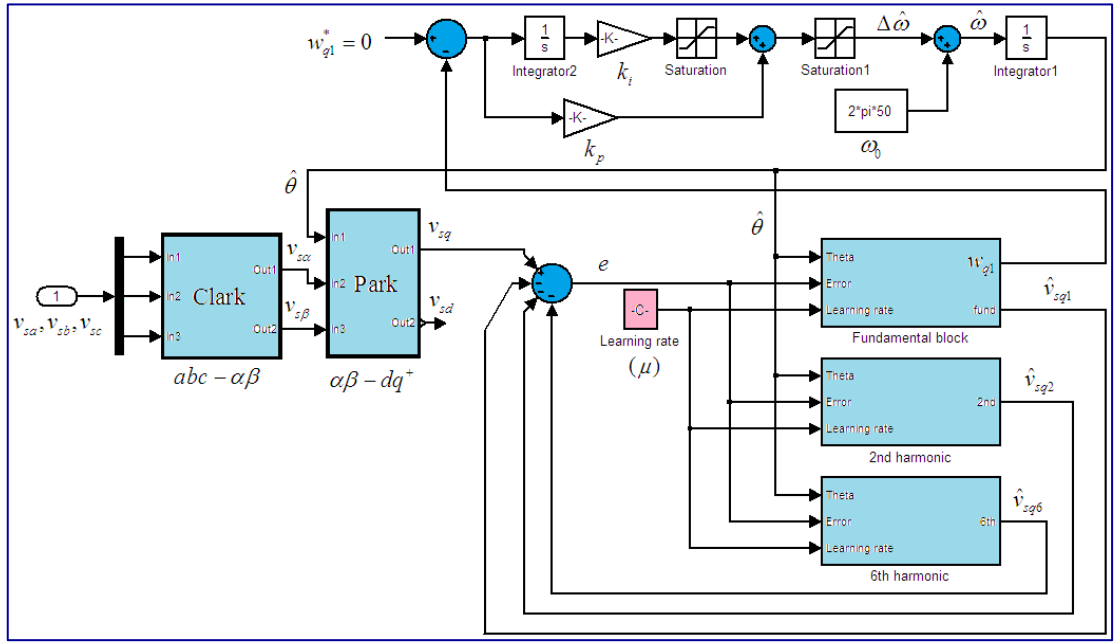
Does our controller work?





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

Model-based design != Coding

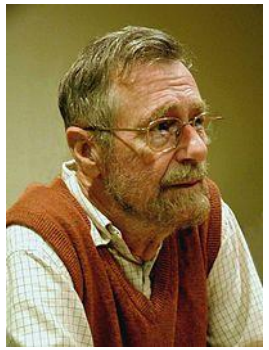



```python
#make sure that we are allowed to recurse many times
import sys
sys.setrecursionlimit(20000)

def probOfStreak(numCoins, minHeads, headProb, saved=None):
    #Computes the probability (i.e. S[n,k]) of getting a run of minHeads
    #(i.e. K) or more heads in a row out of numCoins
    #(i.e. N) independent coin tosses where the probability of getting a
    #heads each time is headProb (i.e. p) and the
    #probability of a tails is 1-headProb (i.e. q).
    #We will be using the recursion:
    #S[N,K] = p^K + sum_{j=1,K} p^(j-1) (1-p) S[N-j,K]
    #As well as the base cases   S[0,k] = 0  and  S[N,K] = 0 for K>N

    #if it's our first call, allocate a hash table to store saved values
    if saved == None: saved = {}

    #get a unique identifier for the value that they want to compute
    ID = (numCoins, minHeads, headProb)

    #if it has been computed before, just return the precomputed value.
    #there is no point in wasting time computing the same thing again.
    if ID in saved: return saved[ID]
    else: #if it's never been computed before
        #handle the base case where we have no coins or where we have
        #more heads we are looking for than we have coins
        if minHeads > numCoins or numCoins <= 0:
            result = 0
        else:
            #use our recursive relationship to compute S[n,k] by
            #breaking it into a sum of terms involving S[n-j,k] for 1<=j<=k
            result = headProb**minHeads  #S[n,k] = p^k + ...
            #S[n,k] = ... + sum_{j=1,k} p^(j-1) (1-p) S[n-j,k]
            for firstTail in xrange(1, minHeads+1):
                pr = probOfStreak(numCoins-firstTail, minHeads, headProb, saved)
                result += (headProb**(firstTail-1))*(1-headProb)*pr
        #save the resulting value so that we can use it later, if need be
        saved[ID] = result

        #return the computed value
        return result
```

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

Not only a model of the designed system, 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)
  - Asynchronous models (Chapter 4)
  - Timed models (Chapter 7)
  - Continuous-time dynamical systems (Chapter 5)
  - 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: 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