CRUISE CONTROL

⚡Design and Simulation🛠️

01
INTRODUCTION

02
SYSTEM MODELING

03
CONTROL

04
SIMULATION

WHAT IS CRUISE CONTROL SYSTEM?

OBJECTIVE

To automatically maintain a car's speed at selected setpoint

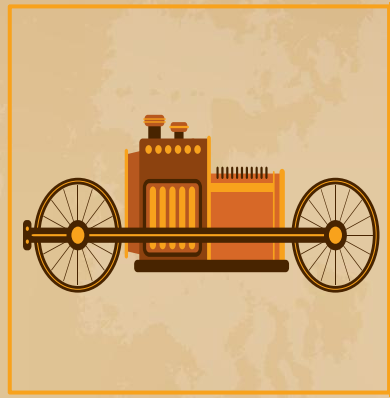
MOTIVATION

Driver Comfort, Fuel Efficiency, & Speed Limit

SCOPE

Design and Simulation of Basic Control System

THE PROBLEM STATEMENT



To build and simulate a basic cruise control system that can automatically adjust the throttle to maintain a target speed, compensating for resistive forces like air drag and friction.

THE SYSTEM

1. We assumed that all resistive forces (like rolling friction and air drag) are linearly proportional to car's velocity (v), modeled as b · v, where b is the linear drag coefficient.

2. The throttle command (u) directly produces a control force equal to its value, so no dynamics in the actuator — just u = applied force.

- $v = x'$
- $a = v'$

ORDINARY DIFFERENTIAL EQUATION

- Applying Newton's Law:
  - $\Sigma F = \text{mass} * \text{acceleration} = m * a(t)$
  - The net force acting on the car:
    - $\Sigma F = F\_input - F\_resistive$
- Using our Assumptions:
  - $m * dv(t)/dt = u(t) - b * v(t)$
- Rearranging to Standard Form:
  - $m * dv(t)/dt + b * v(t) = u(t)$
- This is our Plant Model: A First-Order, Linear, Time-Invariant (LTI) Ordinary Differential Equation describing the car's speed dynamics.

STATE-SPACE MODEL

- Definitions:
  - State vector: $x = [v]$
  - Input vector: $u = [u]$
  - Output vector: $y = [v]$
- State Equation: $\dot{x} = Ax + Bu$
  - $[\ dv/dt\ ] = [-b/m][\ v\ ] + [1/m][\ u\ ]$
- Output Equation: $y = Cx + Du$
  - $[\ y\ ] = [1][\ v\ ] + [0][\ u\ ]$
- System Matrices:
  - $A = [-b/m]$, $B = [1/m]$,
    $C = [1]$, $D = [0]$

TRANSFER FUNCTION MODEL

- Laplace Transform:
  - $m * sV(s) + b * V(s) = U(s)$
- Factoring:
  - $(ms + b) * V(s) = U(s)$
- Transfer Function
  - $G(s) = 1 / (ms + b)$

MODEL PARAMETERS

	SYMBOL	VALUE	UNIT
MASS	m	1000	kg
DAMPING COEFFICIENT	b	50	Ns/m
INPUT FORCE (FOR OPEN-LOOP)	u	500	N

These values are chosen for simulation purposes and they don't represent any specific real car.

OPEN-LOOP ANALYSIS

METHOD

Apply a constant force step input: $u(t) = U$ for $t \geq 0$. (Example: $U = 500$ N)

Assume initial rest condition: $v(0) = 0$.

The ODE:
$m * dv(t)/dt + b * v(t) = U$

ANALYSIS

The velocity response is given by:
$v(t) = (U / b) * [1 - \exp(-t / \tau)]$

$\tau = m / b$ is the system's time constant.

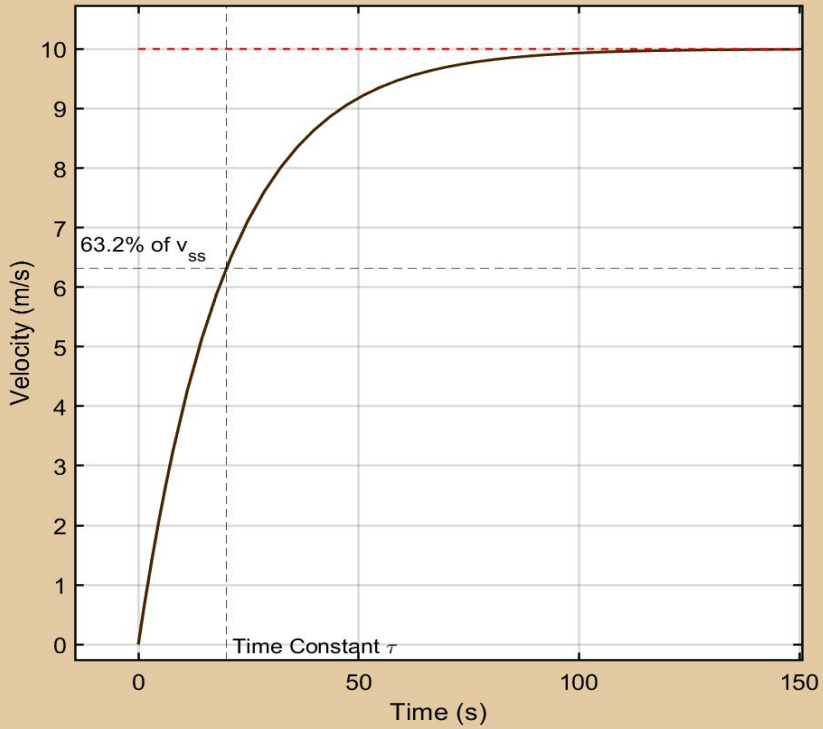
CHARACTERISTICS

Steady-State Velocity:
$v\_ss = U / b = 500/50$
$= \underline{10 \text{ m/s}}$

Time Constant: $\tau = m / b$
$= 1000 /50 = \underline{20 \text{ s}}$

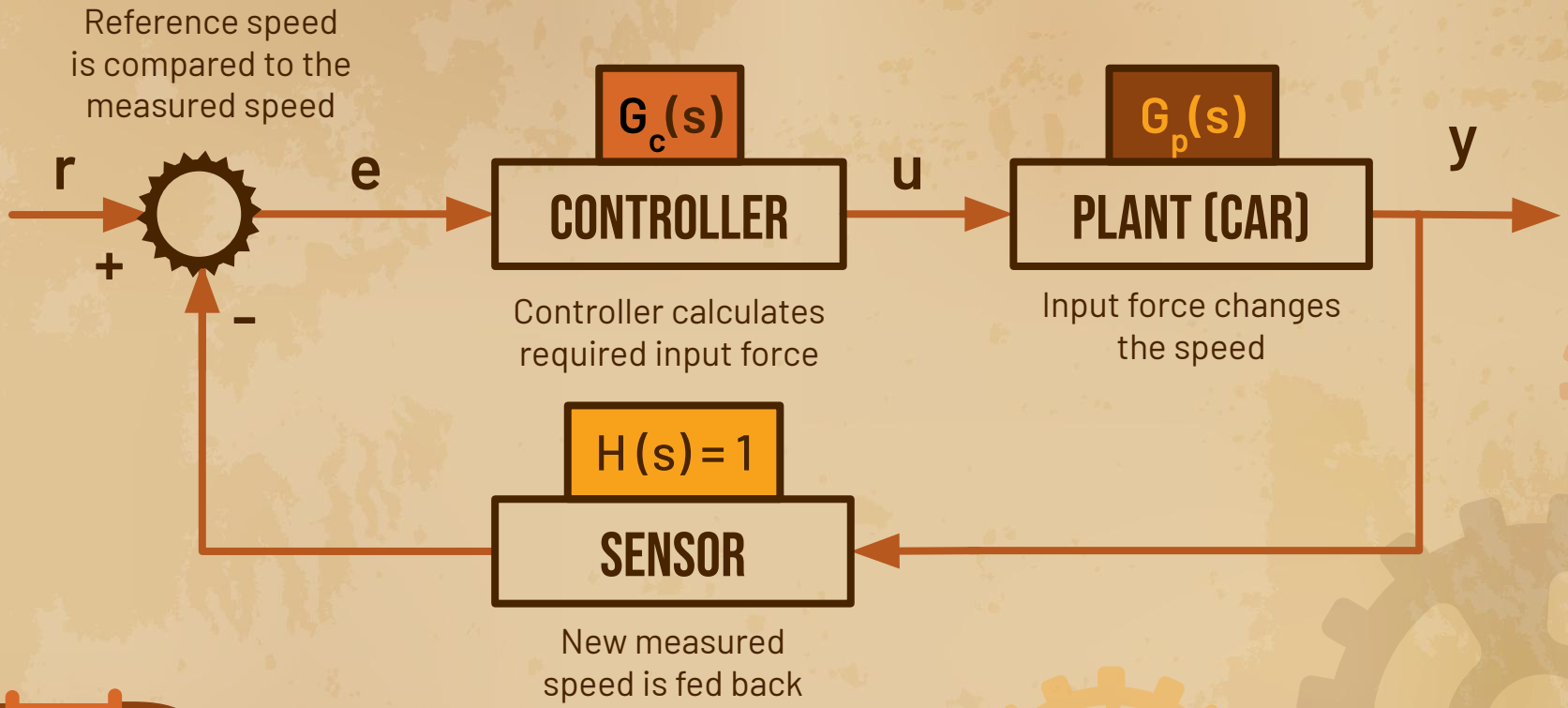
Stability: The open-loop system is inherently stable, approaching v_ss without oscillation.

OPEN-LOOP STEP RESPONSE



- Only specific v at a given u
- Cannot adapt to changes caused by ‘b’ to maintain constant ‘v’
- Feedback is needed to automatically adjust ‘u’ to maintain constant ‘v’

THE BLOCK DIAGRAM



PI CONTROLLER

- Controller Output: The PI controller calculates the required input force $u(t)$ using the error $e(t)$:
  - $u(t) = Kp*e(t) + Ki*\int [0 \text{ to } t]\, e(\tau)\, d\tau$
- Input to Vehicle Model: This calculated force $u(t)$ becomes the input to our ODE:
  - $m*dv(t)/dt + b*v(t) = u(t)$

    $m*dv(t)/dt + b*v(t) = Kp*(v\_r - v(t)) + Ki*\int [0 \text{ to } t](v\_r - v(\tau))\, d\tau$
- PI Controller:
  - Proportional Action: $Kp*e(t)$: Provides input force adjustment based on the current speed difference caused by Resistive forces. But leaves a steady-state error.
  - Integral Action: $Ki*\int e(\tau)\, d\tau$: Adjusts the input force to eliminate the speed error, compensating for the resistive force $b*v$ to achieve the reference speed v_r. Achieves zero steady-state error.

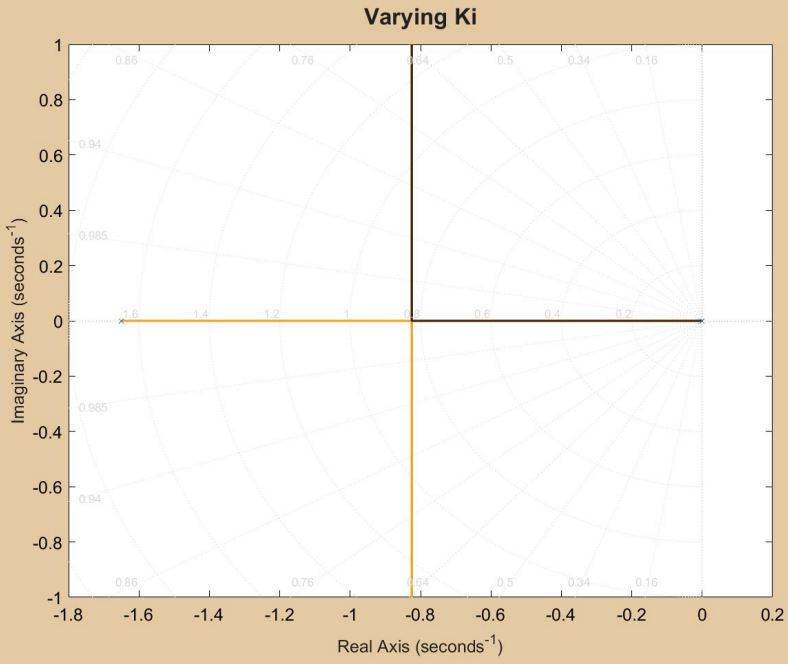
CLOSED-LOOP ODE

- Plant Equation: $m*dv(t)/dt + b*v(t) = Kp*(v\_r - v(t)) + Ki * \int [0 \text{ to } t](v\_r - v(\tau))\, d\tau$
  - Rearranging the Equation:
    $m*dv/dt + b*v = Kp*v\_r - Kp*v + Ki * \int v\_r\, d\tau - Ki * \int v\, d\tau$
- Differentiate w.r.t time (t):
  - $d/dt[\ m*dv/dt + b*v\ ] = d/dt[\ Kp*v\_r - Kp*v + Ki * \int v\_r\, d\tau - Ki * \int v\, d\tau\ ]$
    $m*d^2v/dt^2 + b*dv/dt = 0 - Kp * dv/dt + Ki * v\_r - Ki*v$
  - Rearranging the Equation into ODE:
    $m * d^2v/dt^2 + (b + Kp) * dv/dt + Ki*v = Ki * v\_r$
- Characteristic Equation: $m * s^2 + (b + Kp) * s + Ki = 0$
- Since $m>0$: $(b+Kp) > 0$ and $Ki > 0$ for system to be stable (Routh-Hurwitz Criterion)
- Therefore: $Ki > 0$ and $Kp > -b$

STEADY-STATE ERROR ANALYSIS

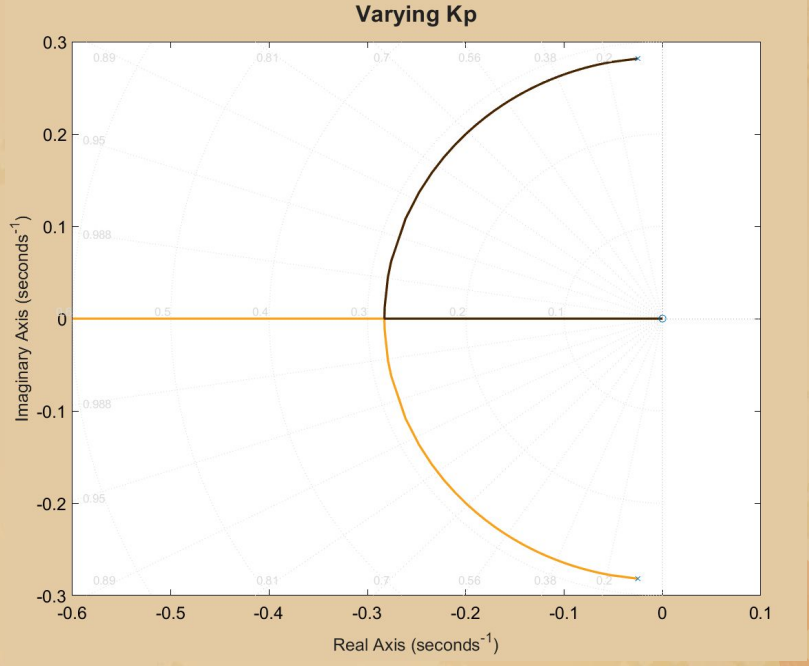
- Steady-State Conditions:
  - $v(t) \rightarrow v\_ss$
  - $dv/dt \rightarrow 0$
  - $d^2v/dt^2 \rightarrow 0$
- Applying to Closed-Loop ODE:
  - $m * (0) + (b + Kp) * (0) + Ki * v\_ss = Ki * v\_r$
    $Ki * v\_ss = Ki * v\_r$
- Provided $Ki \neq 0$, we can divide by Ki:
  - $v\_ss = v\_r$
- Therefore, PI controller (with $Ki > 0$) guarantees zero steady-state error for a constant reference velocity v_r.

ROOT LOCUS



Kp= 1600

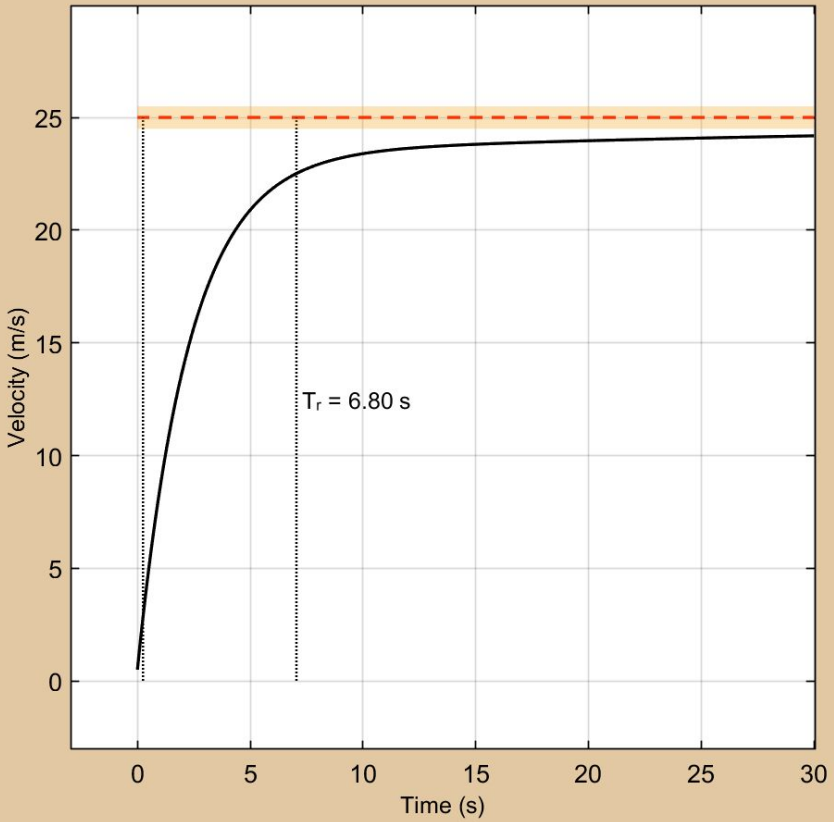
Breakaway: -0.825



Ki=80

Breakaway: -0.283

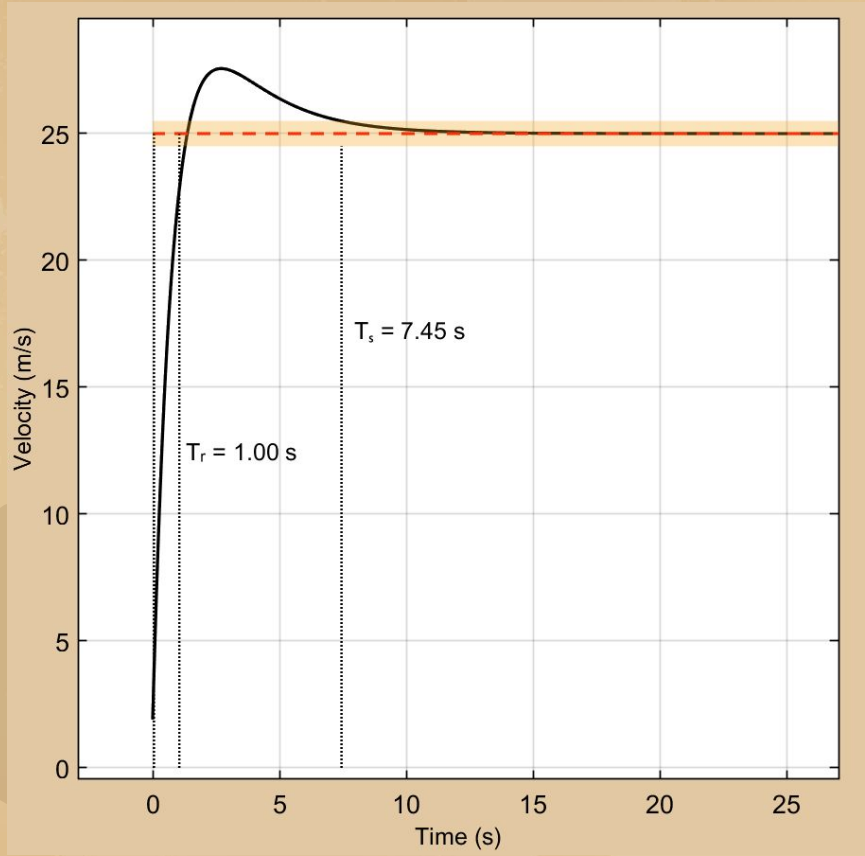
SIMULATIONS



Kp= 400, Ki=10

- Slower Rise (Tr = 6.8 s)
- Extremely slow approach to v_r
- Zero Overshoot
- Overdamped

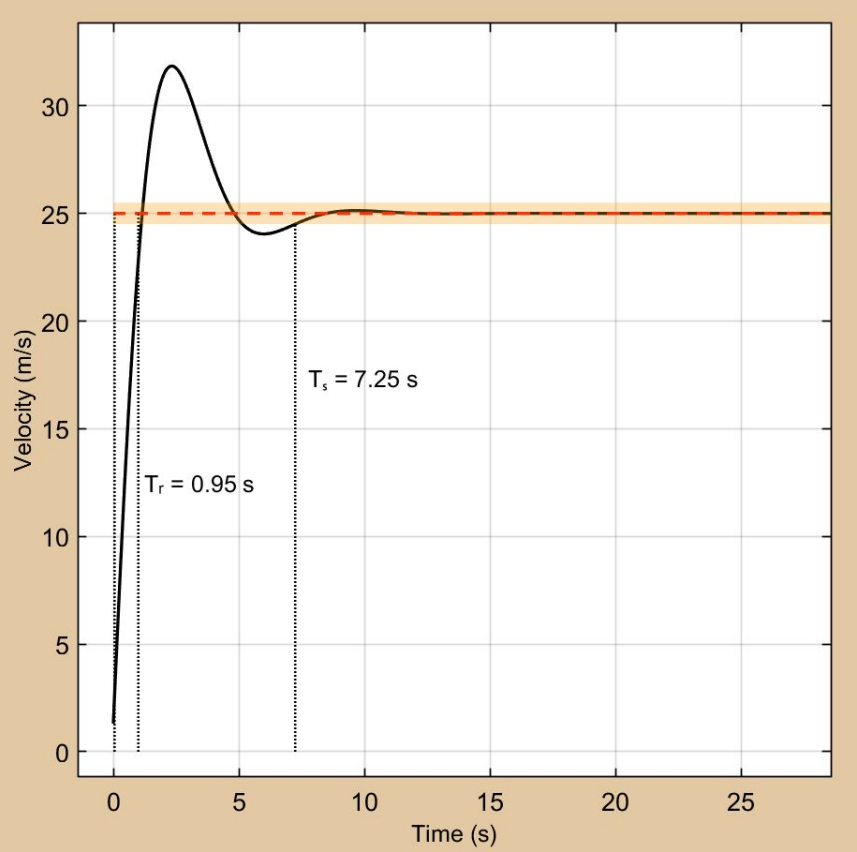
SIMULATIONS



Kp= 1500, Ki=500

- Quick Rise (Tr = 1.0 s)
- Slow approach to v_r (Ts = 7.45 s)
- Small Overshoot

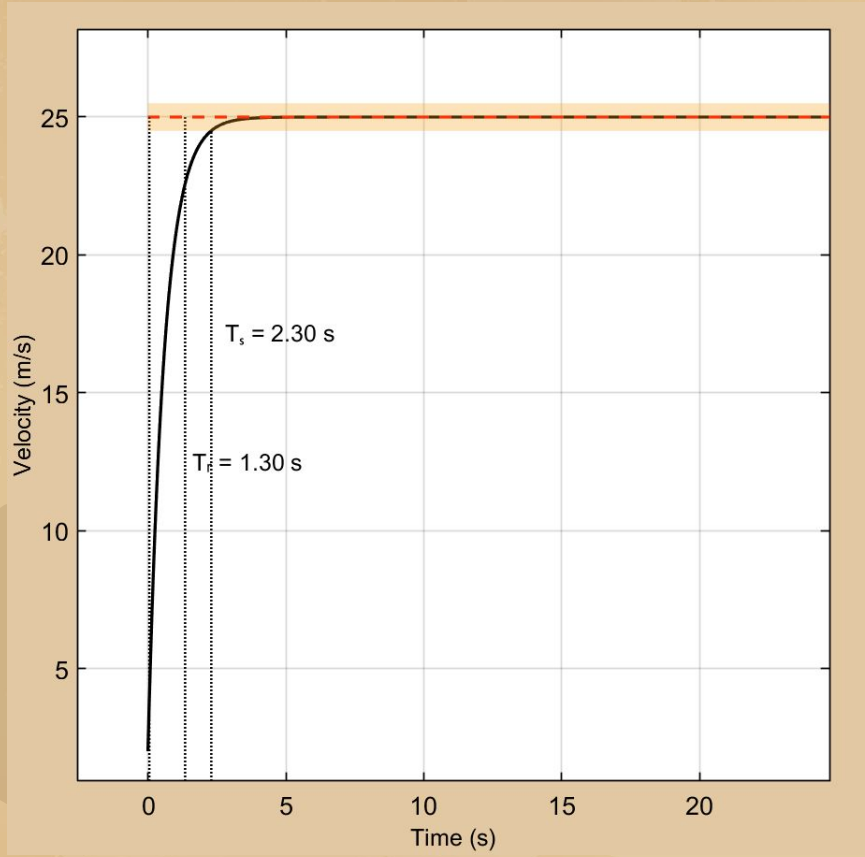
SIMULATIONS



Kp= 1000, Ki=1000

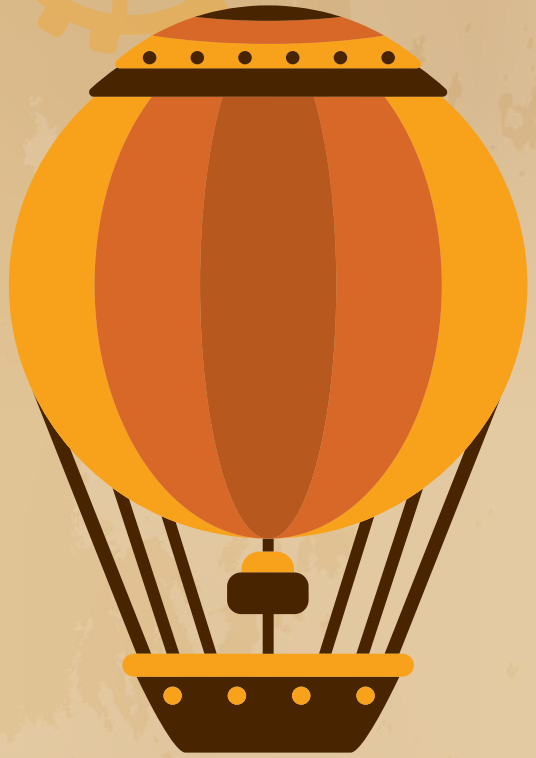
- Quick Rise (Tr = 0.95 s)
- Slow approach to v_r (Ts = 7.25 s)
- Huge Overshoot
- Underdamped

SIMULATIONS



Kp= 1600, Ki=80

- Quick Rise (Tr = 1.3 s)
- Reaches v_r quickly (Ts = 2.3 s)
- Zero Overshoot
- No oscillations



THANK YOU