Reinforcement Learning Intro

Reading

- Sutton, Richard S., and Barto, Andrew G. Reinforcement learning: An introduction. MIT press, 2018.
  - http://www.incompleteideas.net/book/the-book-2nd.html
  - Chapter 1
- E.A. Lee and S.A. Seshia, Introduction to Embedded Systems: CPS Approach, Second Edition, MIT Press, 2017
  - https://ptolemy.berkeley.edu/books/leeseshia/releases/LeeSeshia_DigitalV2_2.pdf
  - Chapter 2
- Puterman, Martin L. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.
  - Chapter 1

Overview

- RL is learning what to do, i.e., map situations to actions
  - Typically in the form of maximizing a numerical reward
- The learner is not told what to do
  - Need to explore the space and discover which actions yield the most return
- RL can be used in many settings
  - Control, scheduling of tasks, training language models
- Control is most relevant to this course
  - An alternative to standard control theoretic methods, especially in complex environments, such as image-based control

Comparison with other types of learning

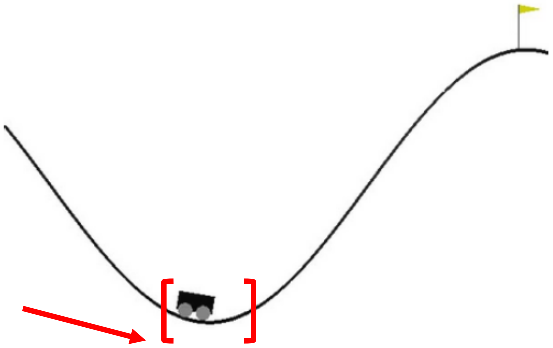
- Different from supervised learning
  - No access to carefully collected labeled data
- Different from unsupervised learning
  - Not trying to learn relationships between unlabeled data
- Similar to unsupervised learning
  - Learning is largely “unsupervised”, agent must explore and learn on its own
- Similar to supervised learning
  - Over time, labeled state-action-reward pairs are collected
- Overall, RL considered a different learning paradigm

Exploration vs Exploitation

- One of the major challenges in RL
- More exploration allows the agent to observe larger parts of the state space and discover higher-reward actions
  - At the expense of more random actions and failures
- More exploitation allows the agent to perform actions that are already known to produce good rewards
  - At the expense of getting stuck in a local minimum
- Decades-old trade-off that does not have an obvious solution
  - Solution is typically task-specific

Example, Mountain Car

- A benchmark reinforcement learning problem
- Learn a controller to get an underpowered car up a hill
  - Need to go up left hill first
  - Small negative reward after each step (smaller for higher inputs)
  - Big positive reward if goal is reached

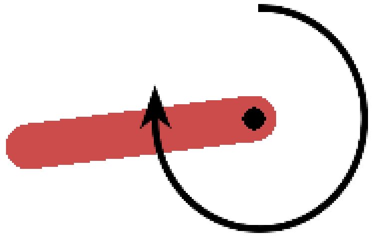


Initial condition chosen randomly from this range

- Learning problem considered “solved” if average reward over 100 random trials is over 90
  - Go up the hill *fast* while conserving energy

Example, Inverted Pendulum

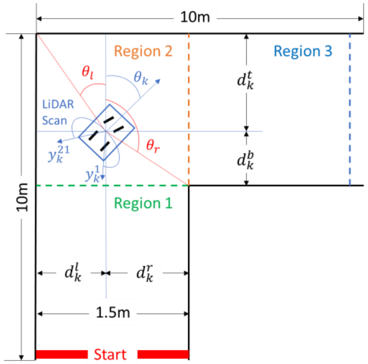
- A benchmark reinforcement learning problem
- Learn a controller to stabilize the pendulum vertically
  - Need to swing it to one side first and then swing the other way
  - Small negative reward after each step (smaller for higher inputs)
  - The longer it takes you to stabilize the pendulum, the lower the reward



- There is no “solved” threshold, but a reward above -200 is generally a good sign

Example, F1/10

- (Soon-to-be) A benchmark reinforcement learning problem
- Learn a controller to navigate a hallway environment
  - Get a small positive reward after each step with no crash
  - Get a big negative reward upon crash
  - Over time, learn to avoid walls



- This problem can be solved with standard control techniques but only for known environments with regular shapes

Other Examples

- Chess (and other games)
  - Select a (sequence of) move that leads to victory
- Learning to walk (in simulation)
  - Select joint/muscle actions that lead to stability and movement
- Learn to flip pancakes
- Fold proteins
- Many, many, many more

Elements of RL

- Agent
  - Robot, controller, decision maker who is learning the task
- Environment
  - Agent's environment, e.g., obstacles, other objects, other agents
- Policy
  - A mapping from perceived states (measurements) to actions
  - i.e., a controller
- Reward signal
  - Defines the goal of the RL problem
  - Observe a reward after each action and corresponding state change
  - Easier for some tasks than for others – need to be able to quantify the conceptual goal (e.g., walking, driving safely)

Limitations of RL

Rensselaer

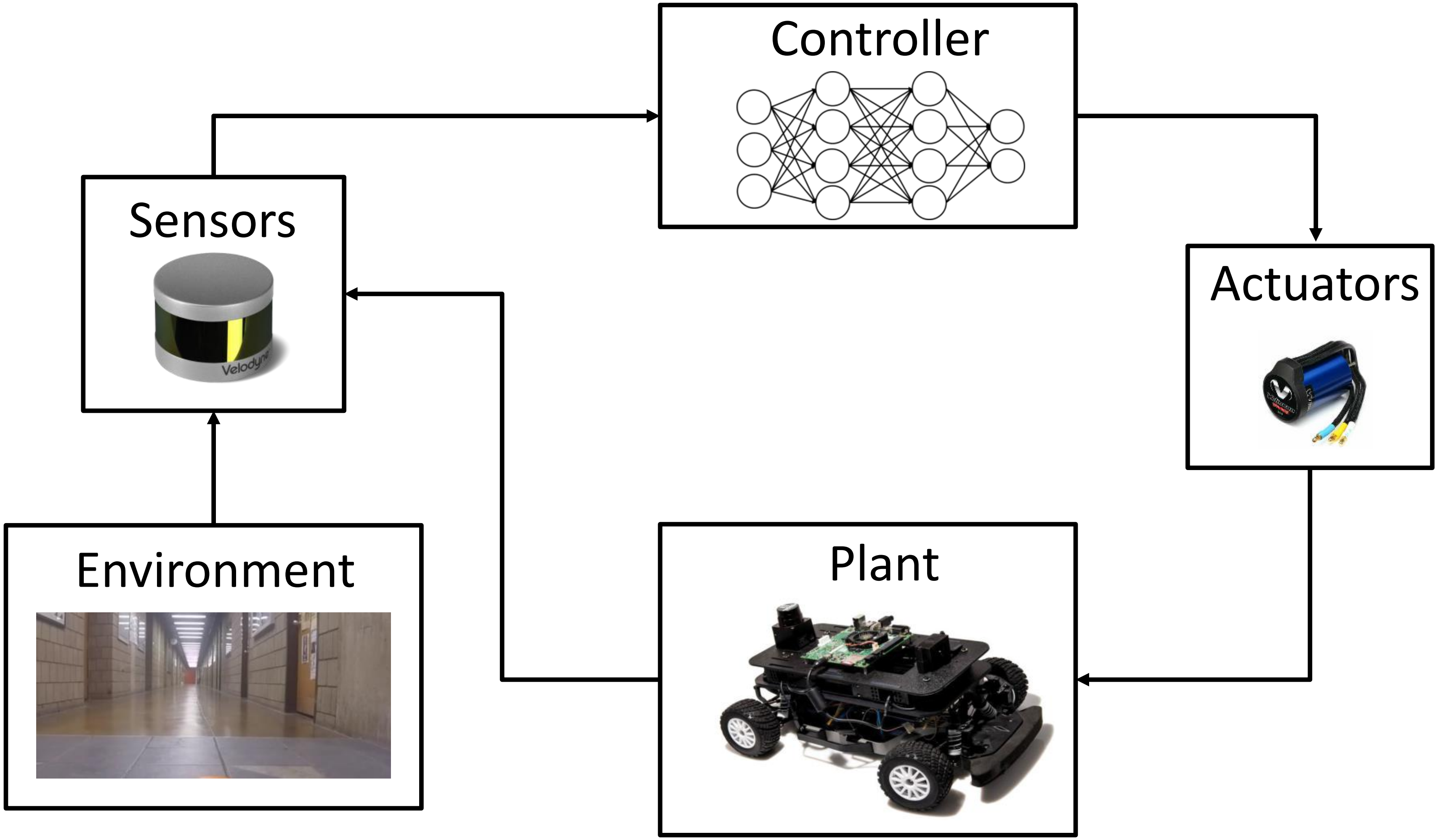
- Measurements need to be sufficient for the agent to maximize reward
  - Some equivalent of "observability" is necessary
    - May be hard to formalize over high-dimensional data
  - If one cannot measure the necessary quantities, then RL unlikely to succeed
- RL is computationally very expensive
  - A lot of iterations necessary and typically no convergence guarantees
  - Often not easy to identify the issue (exploration vs. exploitation, small models, not enough training)

RL vs. Control

Rensselaer

- In most existing settings, standard control is superior to RL
  - Easier to understand, requires (significantly) less computation and easier to adapt/modify
  - Main exception are structured tasks such as games
- The hard problem in modern autonomous systems is perception, not so much control
  - If we know our “state”, then control is easy-ish
- On the other hand, the notion of state may be why it’s so hard to build safe autonomous systems
  - State is an abstraction of the real world, which may be insufficient
  - RL could help in this setting by mapping measurements to controls without explicitly encoding the *state*

Standard Control Loop

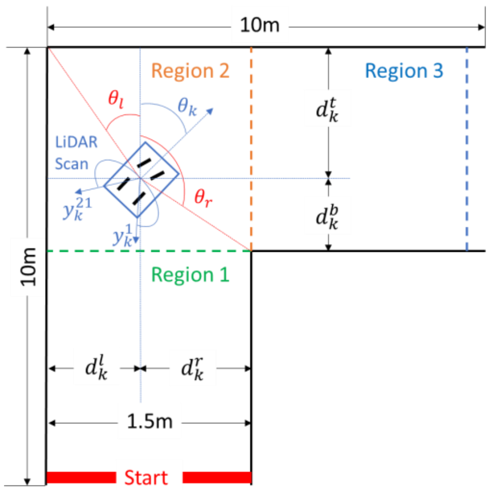


Standard Control Approach

- For simple control tasks, one can build a controller purely based on the error between measurements and a reference
  - PID controller
- For more complex tasks, one needs to model the plant
  - Dozens of modeling frameworks exist
    - Finite state machines, differential equations, hybrid systems, etc.
  - Need to model the measurements as well
  - Given a model, can develop more sophisticated techniques
    - E.g., model predictive control (MPC)
  - Control techniques work fairly well in practice when the model is good

F1/10 Car Simulator

- Developed a model for the F1/10 car as part of my research
- Car navigates a hallway environment while avoiding collisions
  - Has access to LiDAR measurements (laser scan)
- Modeled the car dynamics as well as the LiDAR measurements
- Control inputs are throttle and steering

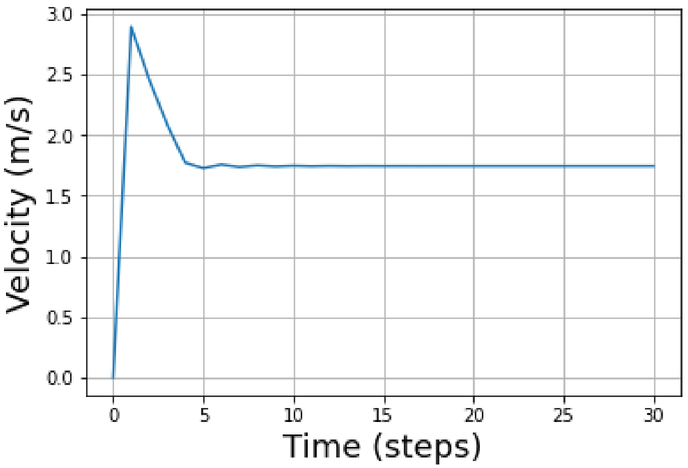


F1/10: control velocity

- Suppose we would like to achieve a target velocity of 2 m/s
- What is a simple approach to achieve that velocity?
  - Try some throttle and observe the error
  - If your velocity is under the target, increase thrust
  - It is enough to know that there is a positive relationship between thrust and velocity
- Attempt 1: apply thrust that is proportionate to the error, i.e., difference between current and target velocity
  - Suppose we observe velocity $v = 1$
  - Error is $e = v_T - v = 1$
  - Apply throttle proportionate to error, e.g.,

$$u = K_p e$$

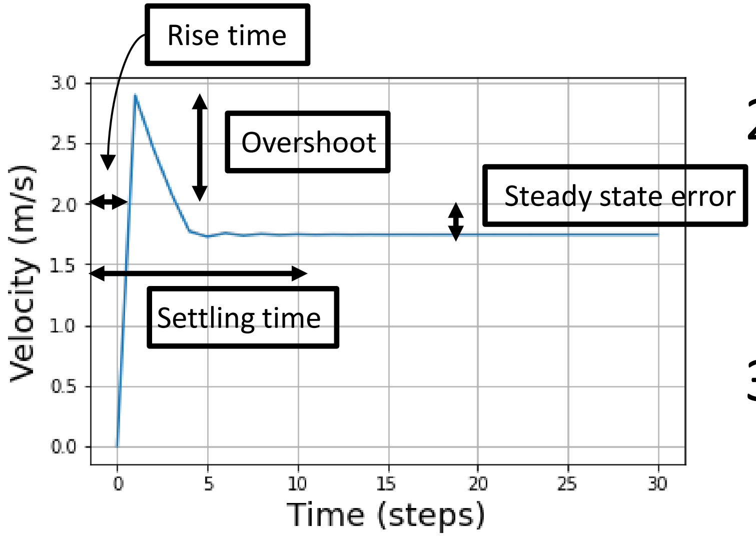
Response of Proportionate Controller



- Step response: How will system output change if at time 0, with $v = 0$, we change reference input to 2?

- Beyond convergence, what are desired characteristics of the response?

Characteristics of the Step Response

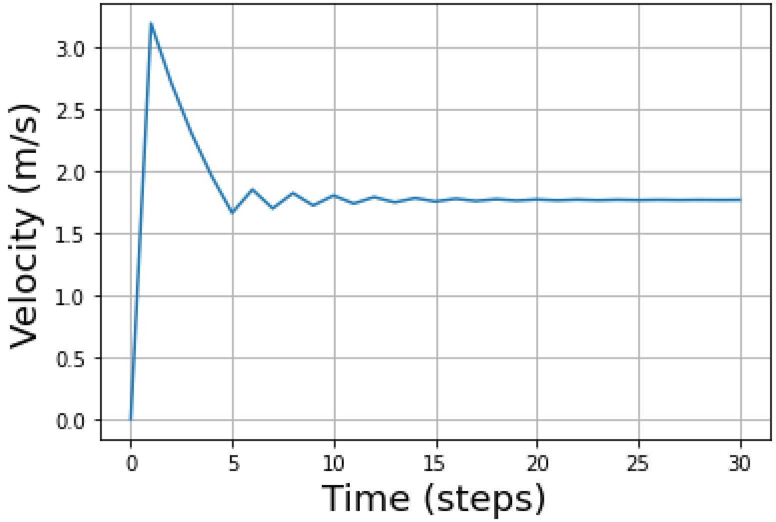


1. Overshoot: Difference between maximum output value and reference value
2. Rise Time: Time at which the output value crosses reference value
3. Settling Time: Time at which output value reaches steady-state value
4. Steady State Error: Difference between steady-state output value and reference

Why is there steady-state error?

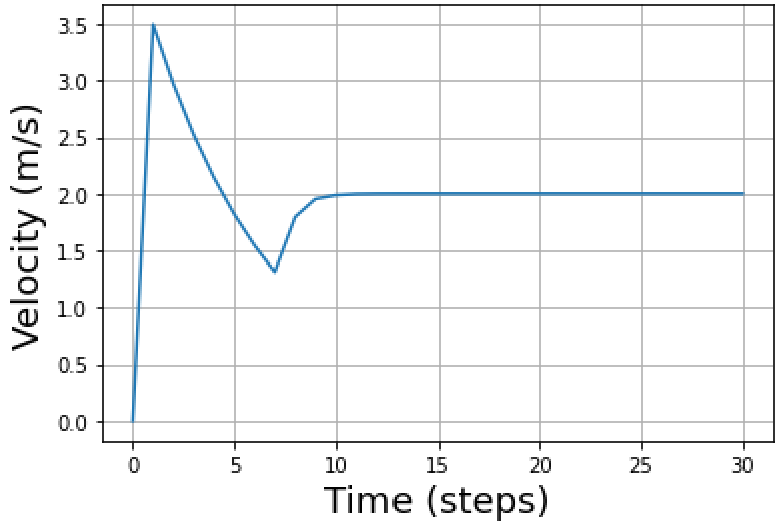
Eventually error becomes small enough so that a proportional controller can't remove it

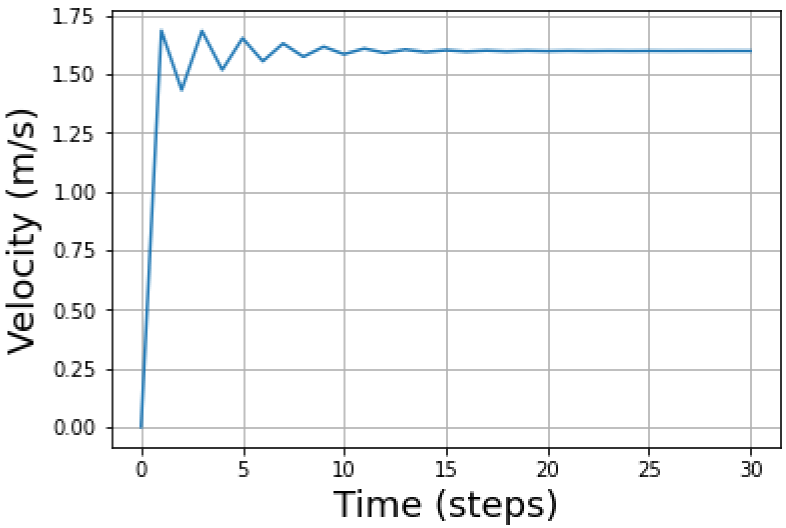
Improving the Step Response



- Performance of the P-controller depends on the value of the proportional gain constant K_P
- What happens if we increase it?
- Rise time decreases, but overshoot increases
- Steady-state error remains!
- How do we get rid of steady-state error?

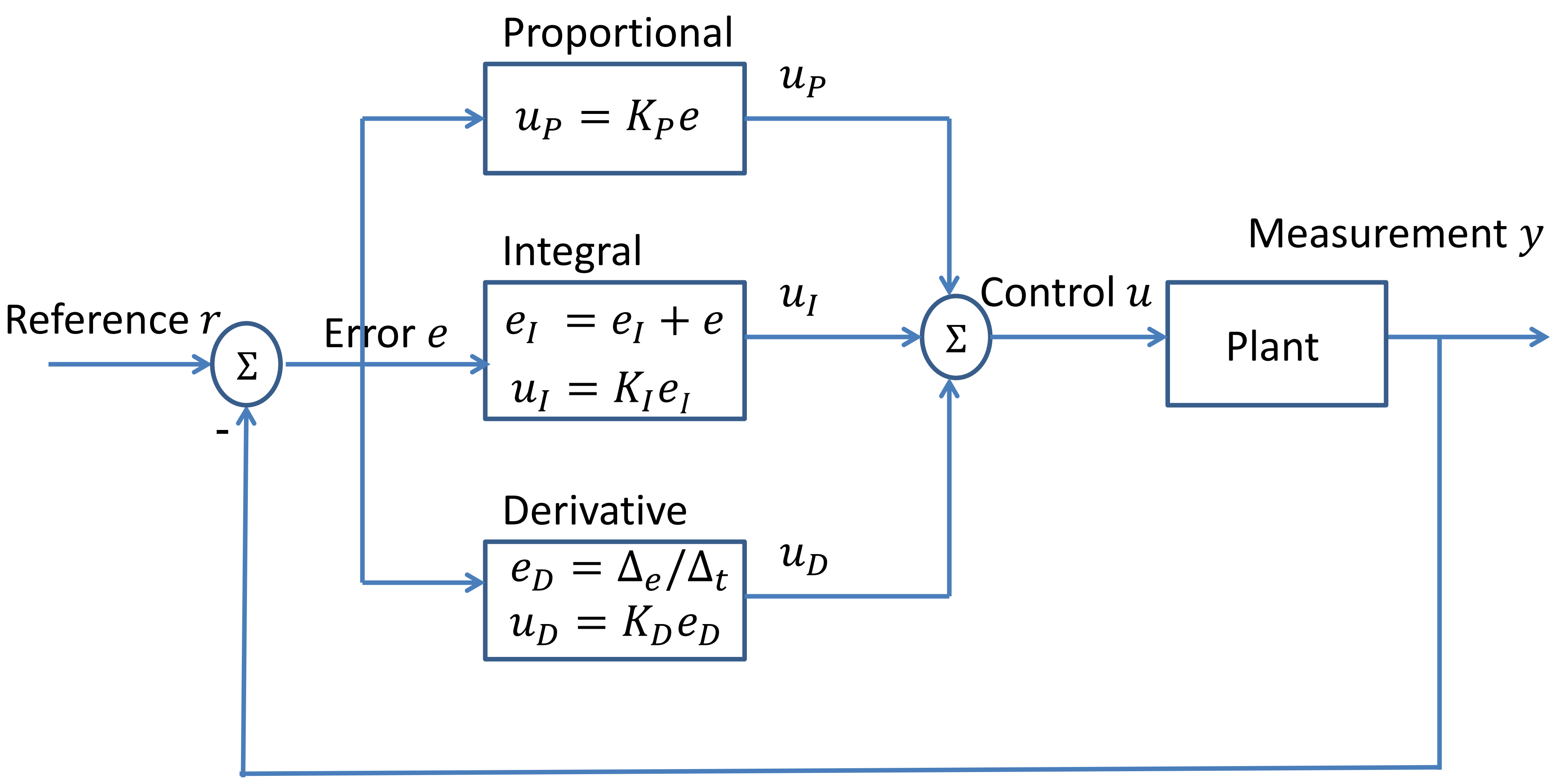
Adding up errors over time





- PI Controller: add up errors over time and adjust throttle accordingly
  - Even if steady-state error is very small, it will eventually accumulate and be corrected
  - Overshoot, rise time, settling time increase (why?)
- PD controller: adding derivative term to proportional controller gets rid of overshoot
  - Steady state error remains

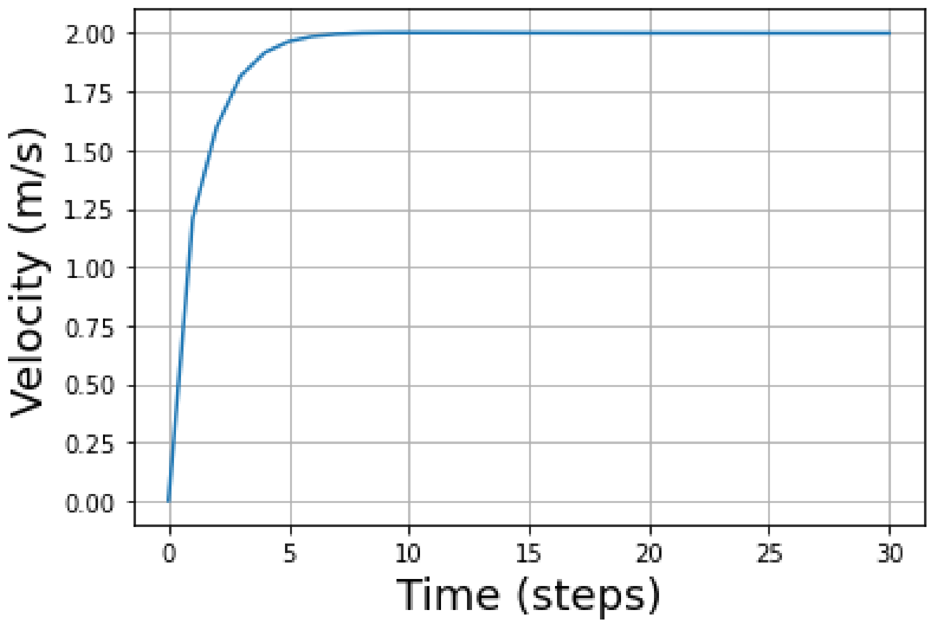
PID Controller



PID Controller

- If $e(t)$ is the error signal, then the output $u(t)$ of the PID controller is the sum of 3 terms:
  - Proportional term: $K_P e(t)$, K_P is called proportional gain (response to current error)
  - Integral term: $K_I \int_0^1 e(t)dt$, K_I is integral gain (response to error accumulated so far)
  - Derivative term: $K_D \dot{e}$, K_D is derivative gain (response to current rate of change of error)
- Special cases of controllers: P, PD, PI
  - You rarely need all 3

PID Controller for F1/10 Car Velocity



- Excellent performance on all metrics
  - $K_P = 18, K_D = 0.2, K_I = 4$
- Small rise time, settling time, negligible steady state error, no overshoot

Designing PID Controllers

- What are the effects of changing the gain constants K_P, K_D, K_{I}?
- Broad co-relationships well understood
  - A PI controller is sufficient for many tasks
  - Derivative term increases variance so people often avoid it
  - It is not uncommon to have a “stack” of controllers, operating at different rates (long- and short-term)
- Control toolboxes allow automatic tuning of parameters
- PID controllers seem to work well even when the actual system differs significantly from the plant model
  - Computation of control output depends only on the measured error, and not on the model!

Deficiencies of PID controller

- When is the PID controller not sufficient?
  - For example, can you solve Mountain Car?
  - No, because you need to get farther from the goal first
- PID controller is only good when the error provides enough information
  - Sometimes, you need to plan ahead
  - Need to know how your control affects the plant
  - Need to know the dynamics of the plant!
- For more sophisticated control, we need to model the plant
  - Same goes for RL – need to have a good model in order to learn a sophisticated strategy

A Simple Dynamics Model

- Suppose a car is moving in a straight line at $v\ m/s$
- How much will the car have travelled after $T\ s$?

$$vT\ m$$

- Suppose the car's position at time 0 is p_0 and at time T is p_T

$$p_T = p_0 + vT$$

- Suppose every T seconds velocity jumps up by $a\ m/s$
- How do we adapt the model (for discrete times when velocity is changed)?

$$p_{kT} = p_{(k-1)T} + v_{(k-1)T}T$$
$$v_{kT} = v_{(k-1)T} + a$$

  – where $k = 1,2, \ldots$