ECE209AS (Fall 2025)

Computational Robotics

Prof. Ankur Mehta ⟨mehtank@ucla.edu⟩

Challenge problems: Particle on a numberline

System definition

Consider a discrete-time, continuous-space dynamic particle with mass $m = 1$ on the numberline, that is, a dynamical system in an operational space of $\mathbb{R}$. The system can have any real-valued position $y \in \mathbb{R}$ and velocity $v \in \mathbb{R}$, and operates under Newton's laws of motion under an applied real-valued input force $f_i \in [-1, 1]$. It may also be influenced by a constant external potential field $\phi(y)$ s.t. $f_\phi(y) = -\frac{d}{dy}\phi(y)$.

(An example of a potential field might be $\phi(y) = 2\cos(y)$ giving external force $f_\phi(y) = 2\sin(y)$ for a net force $f_{net} = f_i + 2\sin(y)$)

Newton's laws of motion in discrete time say that:

$$y[t+1] = y[t] + v[t],$$

$$v[t+1] = v[t] + \frac{1}{m} f_{net}[t].$$

This system has noisy dynamics:

- speed wobble provides an additive zero-mean Gaussian disturbance on the velocity v with $\sigma_d = 0.1v$, i.e. $d \sim N(0, \sigma_d^2)$, and
- there is a probability of crashing, where the velocity gets set to 0 w.p. $p_c \cdot |v|/v_{max}$ (which will be bounded between 0 and $p_c < 1$).

This system also has a noisy sensor, outputting an estimate of the position y with additive zero-mean Gaussian noise with $\sigma_n = 0.5v$, i.e. $n \sim N(0, \sigma_n^2)$.

Week 1: Systems and State

Mathematically formulate this particle-on-a-numberline system using the language of discrete systems, then simulate its behavior. Do so in a way that allows for arbitrary constants, including the potential field ϕ.

You may want to build up to the final system in the following order:

- Pure particle on a numberline dynamics without ϕ, speed wobble, crashes, or sensor
- + noiseless sensor
- + ϕ
- + speed wobble, sensor noise
- + crashes

Week 2: Planning / control on MDPs

Consider a simpler, bounded instance of the particle-on-a-numberline system with limited allowable inputs and potentials:

- The applied force is limited to $f_i \in \{-1, 0, 1\}$.
- The system experiences a constant external force field $\phi(y)$ s.t. the applied force is

$$f_\phi(y) = \text{int}(A\sin(2\pi y/y_{max}))$$

Since the forces are integers, so too will be the states. We will put bounds on the states:

$$y \in \{-y_{max}, -y_{max}+1, ..., 0, ..., y_{max}-1, y_{max}\},$$
$$v \in \{-v_{max}, -v_{max}+1, ..., 0, ..., v_{max}-1, v_{max}\},$$

with $y_{max}, v_{max} \in \mathbb{Z}$. Newton's laws of motion remain the same, with the caveat that the next state is bounded by the state space.

To keep this a finite discrete-space system, the additive speed wobble $d(v)$ on the velocity is instead drawn from a probability distribution:

$$d(v) = \begin{cases} 1 & \text{w.p.} \quad \frac{v}{v_{max}} \cdot p_w/2 \\ 0 & \text{w.p.} \quad 1 - \frac{v}{v_{max}} \cdot p_w \\ -1 & \text{w.p.} \quad \frac{v}{v_{max}} \cdot p_w/2 \end{cases}$$

The probability of crashing (i.e. the velocity immediately goes to 0) is the same as before: $p_c \cdot |v|/v_{max}$.

For simplicity in this problem, we will assume the separation principle and allow our robot to know its exact state (thus ignoring the sensors / observations for now).

Our problem will be derived from getting our particle into a prescribed state.

Formulate then solve several related MDP problems on this dynamical numberline system, finding an optimal policy for a variety of task definitions. Do so in a way that allows for arbitrary environments and tasks to be loaded, giving you a general dynamical MDP solver that you can apply to these problems.

You may want to consider problems in the following order, using both value- and policy- iteration:

- Define a reward and compute an optimal value and policy to get the particle to rest at the origin in the shortest amount of time. Demonstrate trajectories under this optimal policy under a representative variety of p_w, p_c, A. Make sure y_{max}, v_{max} are sufficiently large to see the relevant effects.
- Add in a fuel cost—i.e. a negative (discounted) reward of value c during each transition in which $a \neq 0$. How do the optimal policy and value, with resulting trajectories, change as the relative urgency of achieving the goal γ is varied under different sized hills A?
- How would we formulate and solve a problem in which the particle must intercept a target moving at constant velocity (as opposed to coming to rest at the origin)? Why do we need to modify our solution framework (i.e. policy or value iteration) for this problem?

Week 3: Discretization and function approximation

Consider the continuous particle-on-a-numberline problem, i.e. without the constraints of Week 2. Again assume the separation principle, ignoring the output due to full state knowledge. Use approximate MDP solving techniques to derive and evaluate a continuous space controller / planner to drive this particle to a desired state, namely at rest at the origin.

Formulate then solve an approximate MDP problem on this continuous space dynamical numberline system, finding a continuous space policy for a variety of task definitions.

Try to do so in a way that allows for arbitrary environments and tasks to be loaded, giving you a general approximate dynamical MDP solver that you can apply to these problems.

You may want to consider problems in the following order:

- Discretize the state and action spaces using grid decomposition (with the resolution being a configurable hyperparameter), and compute an optimal policy on the discretized problem.
- Using a nearest-neighbor (NN), 0-step lookahead process: derive, simulate, and evaluate the resulting continuous space policies for varying grid resolutions.

- Extend the above to use just bilinear interpolation over 4-NN, just 1-step lookahead, and both; again deriving, simulating, and evaluating the resulting continuous space policies for varying grid resolutions.
- Draw some conclusions about the effects of the hyperparameters of the discretization approach on the relationship between the computational cost and the resulting continuous space behavior.

Week 4: Graph-search based motion planning

We will build a planner to drive our continuous particle-on-a-numberline system to rest at the origin. However, in doing so, you may find it helpful to establish bounds on the allowable state $x = (y, v)$, and start by assuming no noise terms. Again assume the separation principle, ignoring the output due to full state knowledge.

This is a non-holonomic system (why?) so we will need to modify what "straight line motion" between two states would mean. Define that to mean that two points in state space can be connected if there exists a constant input f_i that drives the initial state to the final state (over some upper-bounded duration of time).

Develop a probabilistic roadmap (PRM) based approach to planning, solving and simulating your planner on a variety of problem configurations.

You may find it helpful to approach the problem in the following steps:

- Build an algorithm to compute whether two points in the state space are connected by our version of "straight line motion".
- Use that algorithm to build a probabilistic roadmap (PRM) graph on the space. Demonstrate the evolution of the graph, and empirically observe how many datapoints are needed to get a prescribed density of connected points within our graph.
- Use the generated graph to compute a path plan to drive the system from an arbitrary initial state to rest at the origin under a variety of amplitudes $A : \phi(y) = A\cos(y)$.
- Compare the execution of motion plans under various non-zero noise conditions.

Week 5: Linear quadratic regulators

Consider a reduced version of the continuous particle-on-a-numberline system with the following simplifications:

- no applied field $\phi(y) = 0$, thus making it a linear system,
- only additive white gaussian noise (AWGN) on the velocity $d[t] \sim N(0, \sigma_d^2), \sigma_d \in \mathbb{R}$ (no other noise terms), and
- no explicit bounds on the input force f_i.

As before, we will assume the separation principle and consider full state knowledge, so we can ignore the output sensors for now.

Thus, our system model looks like:

$$y[t+1] = y[t] + v[t],$$
$$v[t+1] = v[t] + f_i[t] + d[t].$$

We will try to drive this system to a desired goal state y_d, v_d.

Mathematically formulate this as an LQR control problem on a linear dynamical system, then simulate its behavior under varying cost terms.

You may want to build up to the final system in the following order:

- Drive the system to rest at the origin, i.e. $y_d = v_d = 0$. How does changing the relative values of Q, R impact the behavior?
- If you knew that your initial state was bounded by $|y[0]| \leq y_{max}, v[0] = 0$, predict (analytically) how to set Q, R to ensure bounds on your input $|f_i[t]| < 1$ for all t. Verify this in simulation.

Week 6: Bayesian filtering and POMDPs

We can now consider the full, continuous particle-on-a-numberline system, with a noisy sensor and unbounded state space.

Create a particle filter on this system, then use that state estimator to solve a noisy planning problem.

You may want to build up to the final system in the following order:

- Use your simulator to build a particle filter, then displaying the evolution of belief state over time as the system executes a series of actions from both known and unknown initial states. Be sure to explore various system constants.
- Couple this state estimator with a previous planner, using the mean of the belief state as the state planner. Characterize its performance.
- Apply the planner to each particle from your filter, then combine the resulting set of prescribed actions to a single action to take. Evaluate the resulting behavior, and compare to your previous approach.

Week 7: Kalman filtering and SLAM

Consider the full continuous particle-on-a-numberline system, but without crashes (i.e. $p_c = 0$).

Formulate and implement a Kalman filter on this problem, and run through some examples.

You may find it helpful to approach the problem in the following steps:

- Consider only simple zero-mean additive white Gaussian noise (AWGN): a disturbance (process noise) on the velocity v with constant variance q_v^2 and a noisy measurement of the position y with (observation noise) variance r_y^2.
  - What are the steady-state (limit as $t \to \infty$) a-priori and a-posteriori state estimate uncertainties ($\Sigma_{t|t-1}, \Sigma_{t|t}$ respectively)?
  - What happens as we add uncorrelated process noise on the position with variance q_y^2? What if we add a velocity sensor with associated uncorrelated observation noise with variance r_v^2?
- Consider the question of sensor selection – what is the tradeoff between a sensor that has uncorrelated observation noise between y and v:
$$R_a = \begin{pmatrix} r_y^2 & 0 \\ 0 & r_v^2 \end{pmatrix}$$
vs a sensor that has smaller but correlated observation noise:
$$R_b = \begin{pmatrix} r_y^2 & r_{yv} \\ r_{yv} & r_v^2 \end{pmatrix},$$
with smaller on-diagonal terms, but nonzero off-diagonal terms?
- What happens if we instead reintroduce the velocity-dependent (multiplicative) sensor noise and speed wobble?

Week 8: Reinforcement Learning

Assume that you don't know P (i.e. the system dynamics) or R (i.e. the desired setpoint) explicitly, and must use your simulator to extract data samples. Use a Q-learning approach to learn a policy on this continuous-space system. Assume the separation principle at first, ignoring the sensor measurements in favor of full state knowledge.

Formulate, implement, and evaluate a Q-learning approach to the dynamical particle-on-a-numberline system

You may want to approach this problem along the following steps:

- Ensure your simulator works, and can return a next state and reward associated with any given initial state and action (from the associated infinite continuous spaces), under appropriate probability distributions.
- Choose a feature basis on the Q value function, possibly driven by an understanding of the underlying physics, or otherwise using a general function decomposition on the space.
- Given a dataset D of sampled transitions and rewards, compute the following as applied to D:
  - the loss function $\mathcal{L}$ that maps a parameterization θ to a real number $\mathcal{L}(\theta) \in \mathbb{R}$, and
  - the gradient of that loss function $\vec{\nabla}_\theta \mathcal{L}$ that maps a parameterization θ onto a vector in θ-space.

  Use your simulator to generate (random) datasets of transitions and rewards, and test your loss and gradient functions on those datasets.

- Implement Q-learning with experience replay, using an ϵ-greedy strategy to select your actions. Build up to the following improvements:
  - Compute the gradient of the loss on a random subset $\bar{D} \Subset D$ of your dataset.
  - Split your Q function into a slowly-updating $\hat{Q}$ in addition to your current function approximation Q_θ.
- Experimentally observe the effects of the various hyperparameters, such as:
  - the number of transitions in a trajectory before resetting and moving to a new episode
  - the exploration vs. exploitation rate ϵ
  - your gradient descent step size
  - the size of $\bar{D}$ compared to D
  - the number of gradient descent C before updating $\hat{Q}$ from Q_θ
- What if instead of full state feedback, you only had the observation / output along with the reward after each transition? Re-run the above using the output instead of the state everywhere in your analysis. You'll want to consider the following points:
  - The output doesn't capture the state completely, so not all of your original basis functions will apply.
  - Similarly, you may want to include additional basis functions on past outputs as well.
    - (As an aside, this is the line of reasoning that leads to recurrent learning.)