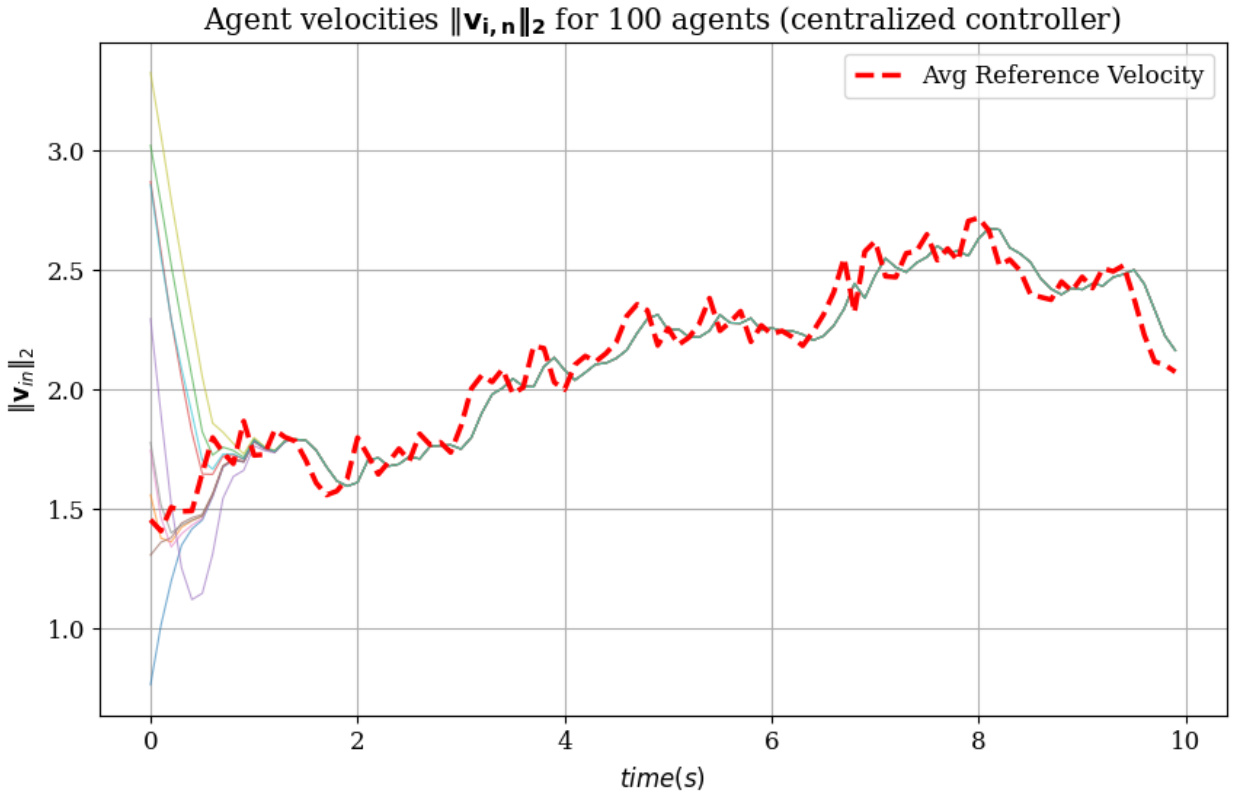
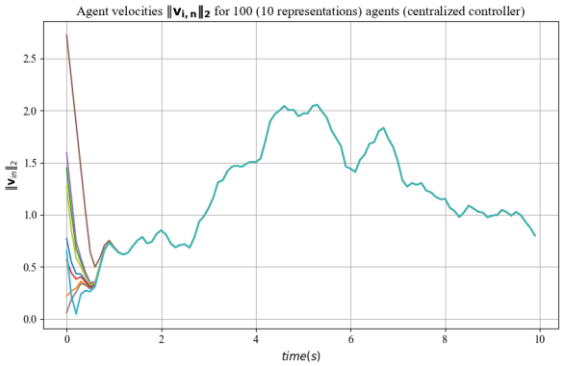
Q2.2

Plot with 10 representative trajectories:

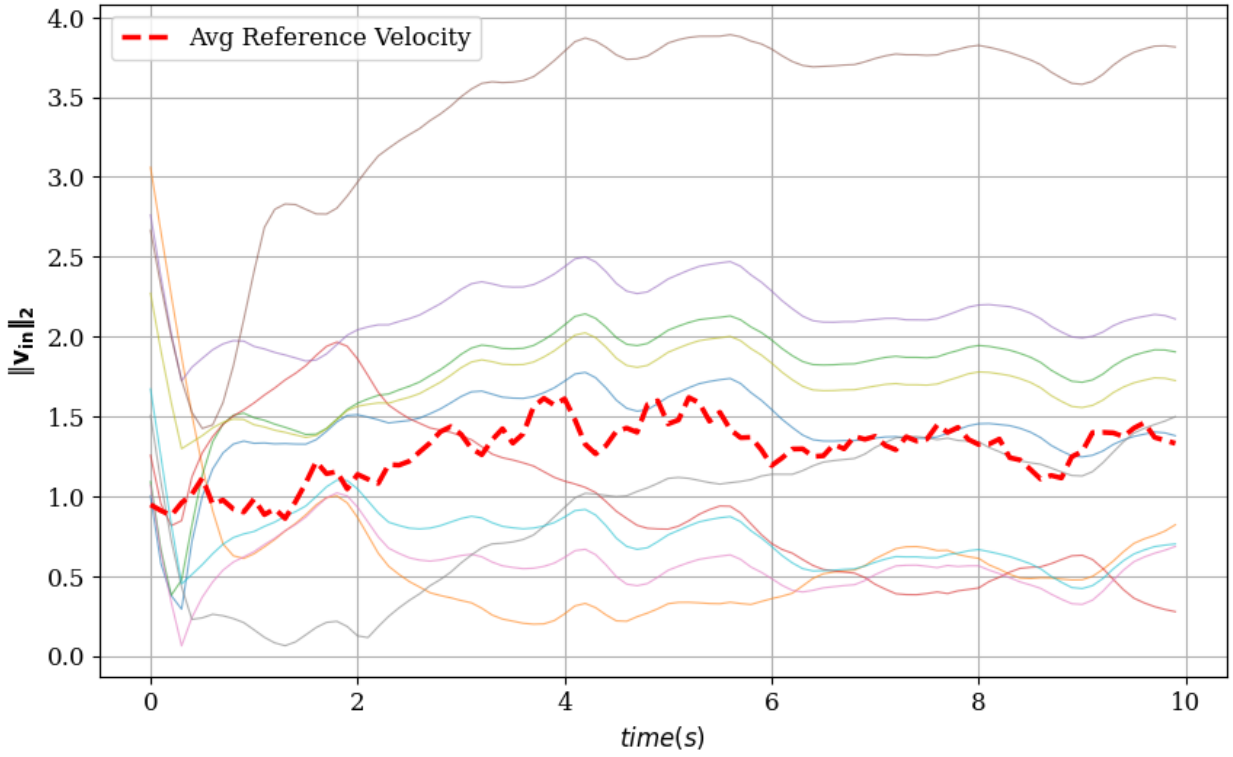


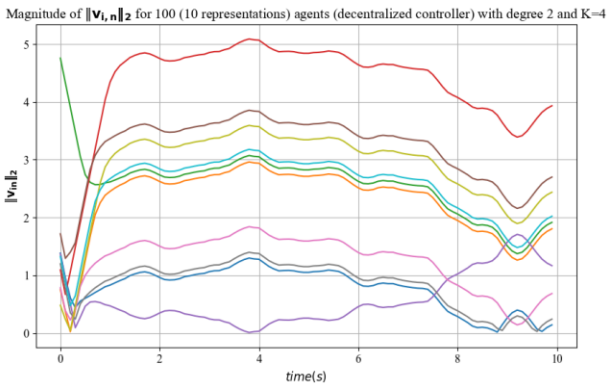


Cost averaged over 100 simulations: **0.02779**

Q2.4

Plot with 10 representative trajectories:





Cost averaged over 100 simulations: **1.3138**

Q3.1

GNN architecture parameters:

Input feature dimension:

- input_dim = 4 (velocity_x, velocity_y, reference_x, reference_y)

Number of layers:

- L = 2

Layer 1 parameters:

- F1 = 64 (number of output features in layer 1)
- K1 = 4 (filter order in layer 1)
- activation = Tanh

Layer 2 parameters:

- F2 = 2 (output control dimension)
- K2 = 1 (filter order in layer 2)
- activation = Tanh (applied after the graph filter; final mapping to control is linear)

Shift operator:

- shift_operator = normalized adjacency matrix (static, does not change during training)

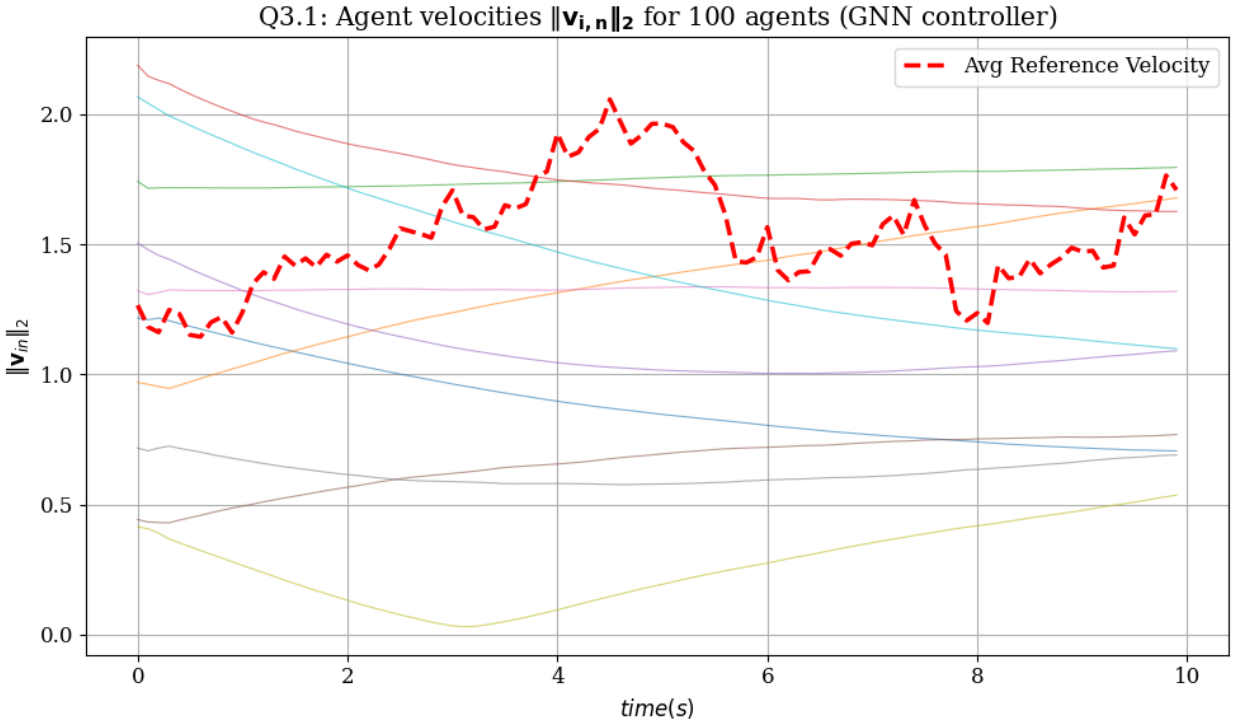
Training settings:

- optimizer = Adam
- num_epochs = 50
- loss = mean squared error between predicted u and centralized u
- mobility = False (training performed with fixed/static graphs)

Controller used:

- GraphNNs.archit (the GNN trained on static graph data)

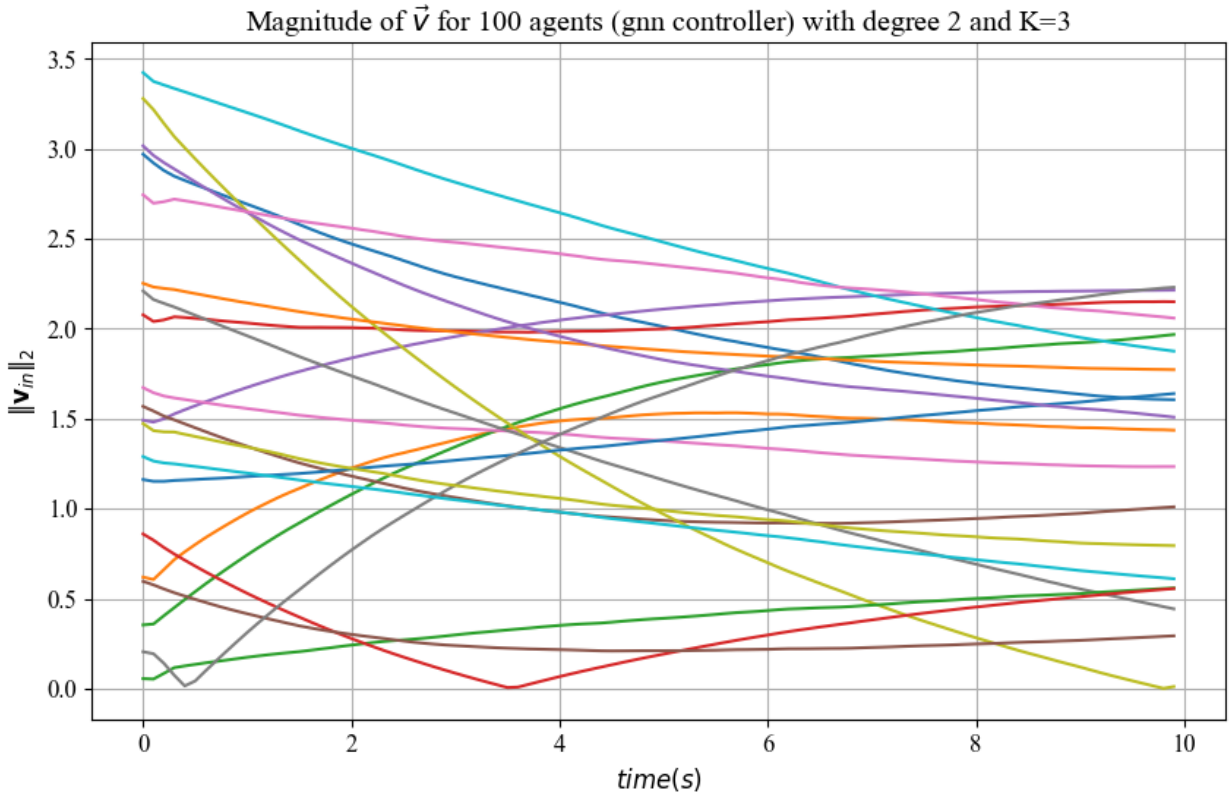
Plot with 10 representative trajectories:



Cost averaged over 100 simulations: 0.859

Q4.1

Plot with 10 representative trajectories:



Cost averaged over 100 simulations: 0.9847

Q4.2

Although the GNN in Section 3 was trained on a fixed communication graph, it still performs surprisingly well when the agents move and the graph changes over time. This robustness arises from the fundamental properties of GNN that message passing is a strictly local operation, and the learned update rules are permutation-equivariant, meaning they do not depend on any particular node ordering or global topology. As long as each agent continues to receive information from a neighborhood with similar statistical structure, the same learned filters can be applied on the time-varying graph without modification. In addition, the GNN layers behave like Lipschitz graph filters, which are known to be stable to small perturbations of the underlying graph. Because the agents' positions evolve gradually, the resulting graph variations are also gradual, and the GNN generalizes smoothly to these changes. As a result, a model trained on a static adjacency matrix can still produce meaningful and coordinated control actions when deployed in a dynamic, mobility-driven environment.

Q4.3

GNN architecture parameters:

Input feature dimension:

- input_dim = 4 (velocity_x, velocity_y, reference_x, reference_y)

Number of layers:

- L = 2

Layer 1 parameters:

- F1 = 64
- K1 = 4
- activation = Tanh

Layer 2 parameters:

- F2 = 2
- K2 = 1
- activation = Tanh (applied after the graph filter; final mapping to control is linear)

Shift operator:

- shift_operator = normalized adjacency
  (built by agent_communication_pos, normalized by largest eigenvalue, updated at every timestep under mobility)

Training settings:

- loss = MSE between predicted u and centralized u
- optimizer = Adam
- num_epochs = 50
- mobility = True (training performed with dynamic graphs instead of a fixed graph)

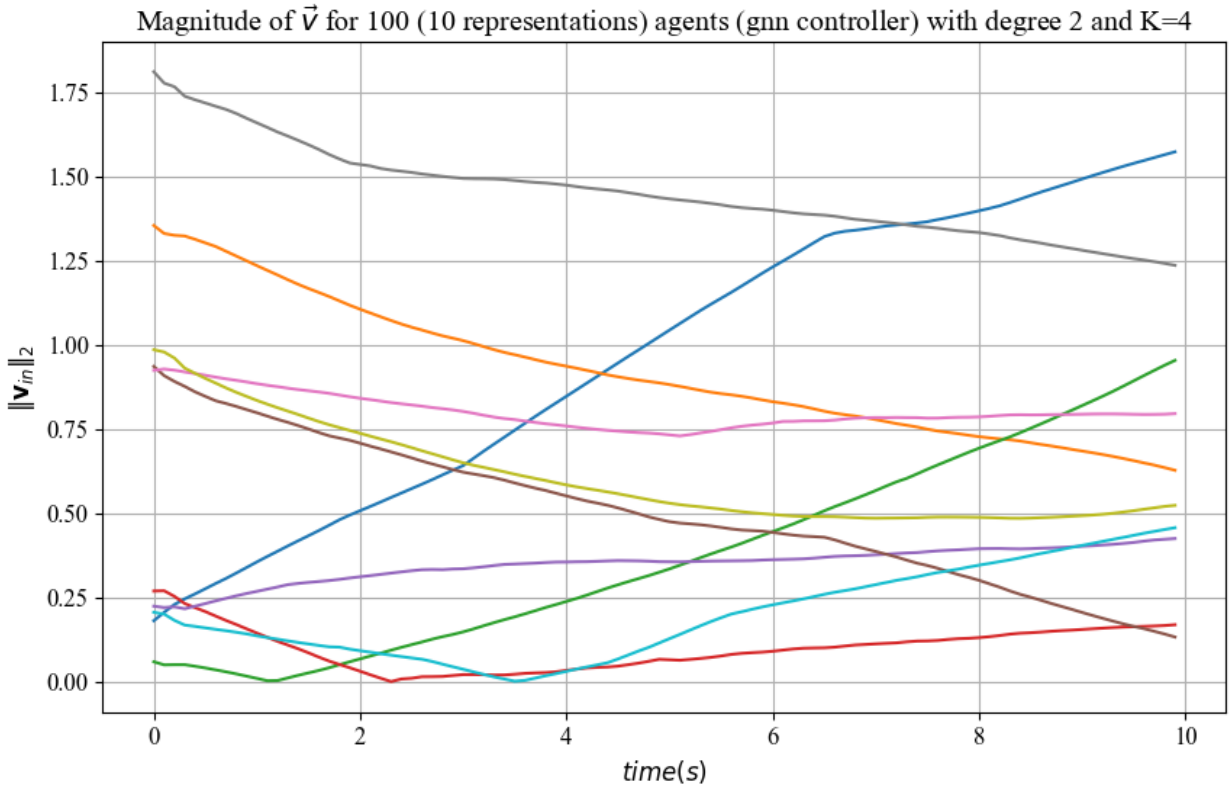
Controller used for mobility rollouts:

- GraphNNs.archit (the GNN trained on mobility data)

Rollout function:

- computeTrajectory_pos
  returns: (ref_vel, est_vel, est_vels, biased_ref_vels, accels)

Plot with 10 representative trajectories:



Cost averaged over 100 simulations: 0.7027

Q5.1

The quantity $\mathbf{q}_{i,n} = \sum_{j \in \mathcal{N}_i}(\mathbf{p}_{i,n} - \mathbf{p}_{j,n})$measures the "relative mass" of node i's neighborhood, and it is essential that this feature be expressed relative to the position of node $i$rather than in absolute coordinates. Using relative positions ensures that the GNN's input is invariant to global translations and independent of how nodes are indexed or labeled. In other words, if we permute the agent labels or shift the entire formation in space, the set of relative vectors $\mathbf{p}_{i,n} - \mathbf{p}_{j,n}$does not change, and therefore the GNN computes exactly the same embeddings. This is precisely what guarantees permutation equivariance, a fundamental requirement for graph neural networks: the output for each node must depend only on its local neighborhood structure, not on arbitrary global coordinates or indexing. If absolute positions were used instead, the GNN's behavior would change under relabelings or translations, breaking equivariance and preventing the model from generalizing to different formations or agent identities.

Q5.2

GNN architecture parameters:

Input feature dimension:

- input_dim = 6 (velocity_x, velocity_y, biased_reference_x, biased_reference_y, relative_mass_x, relative_mass_y)

Number of layers:

- L = 1

Layer 1 parameters:

- F1 = 64 (number of output features in layer 1)
- K1 = 4 (filter order in layer 1)
- activation = Tanh

Output/Readout parameters:

- dimReadout = [2] (MLP readout mapping 64 features → 2 control dimensions)
- output_dim = 2 (control acceleration u_x, u_y)
- activation = None (linear output)

Edge features:

- dimEdgeFeatures = 1

Shift operator:

- shift_operator = adjacency matrix (time-varying due to agent mobility)

Training settings:

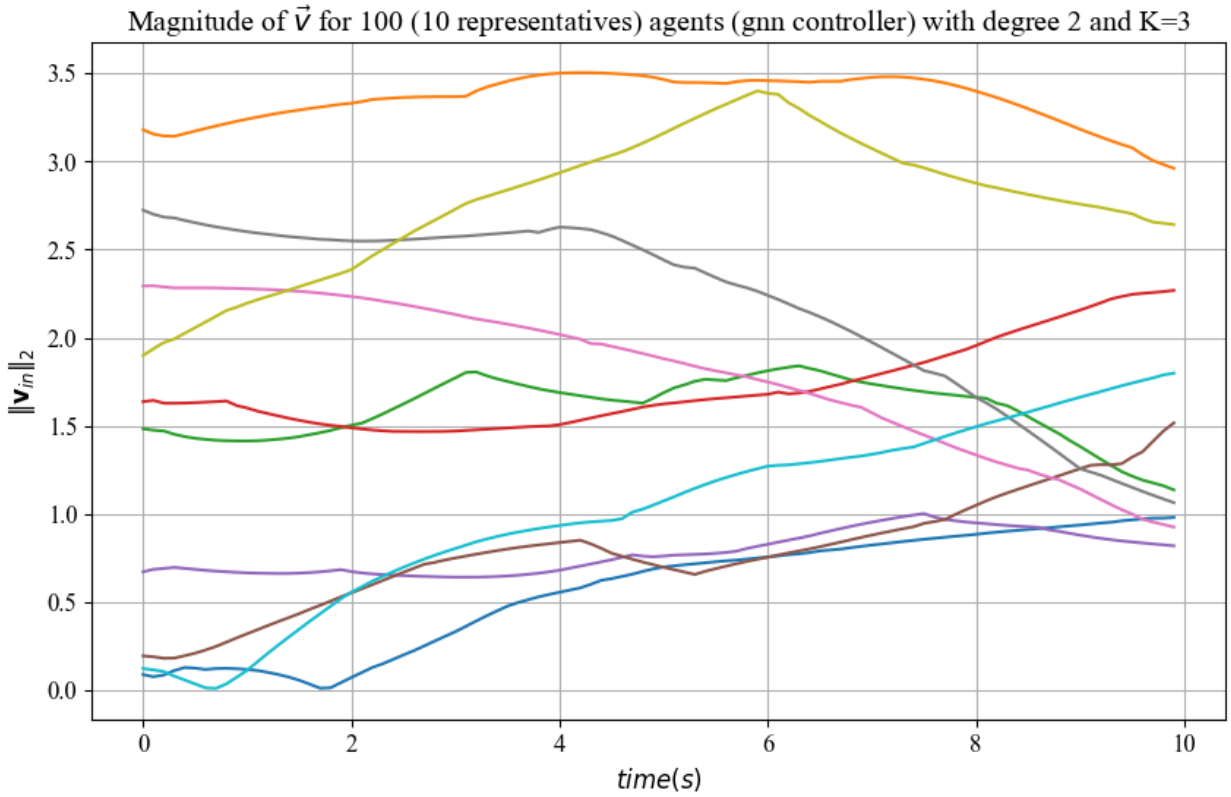
- optimizer = Adam
- num_epochs = 50
- device = 'cuda:0' if available, else 'cpu'
- bias = True
- loss = mean squared error between predicted control and centralized optimal control (from equation 30)

Additional features specific to Q5.2:

- Collision avoidance potential function U(pin, pjn) from equation (28)

- Reference distance $d0 = 2$ m
- Collision avoidance weight $\gamma = 10$ m
- Collision cost weight $\mu = 1$

Plot with 10 representative trajectories.



Cost averaged over 100 simulations: 0.8523