Reinforcement Learning

Project #3 Report

Class and Section: CPE471.1001

Name: Lucas Pinto

Due Date: March 28, 2025

GitHub Repository: https://github.com/PieFlavr/CPE471-Project-3

World-Agent-Target Design

The environment spans infinitely (to an arbitrary technical limit) across an X and Y axis, but all simulations are cut off to be visible only to a 100×100 section from $(0, 0)$ to $(100, 100)$. All trajectories are carefully fit in there to avoid visibility loss.

Potential Field Controller Algorithm

The artificial potential controller implemented for this projects follows very closely to the Python example laid out, however with the following key distinctions:

- Trajectories are generated via a function customizable with a `start point`, then a `trajectory speed` or `end point`.
- Sine Trajectories are also further customizable as such, given amplitude, phase shift, etc.
- For noise generation, while it mostly follows the example and outline given, `noise mean` is set explicitly to 0 as it doesn't quite make sense to have noise always be positively skewed.
- By default, all trajectories are run in sequence, with .gifs and .pngs of resulting figures generated being shown and saved.
- Additionally, the rather unexplained ~ 1.5 time scaling performed in the example code was NOT followed for the sake of better unit consistency, and the results still closely match the example ones given.

Furthermore, the primary loop is implemented as in the following image…

```
# ============================================================================================
# COMPUTING AGENT TARGET POTENTIAL STATES
# ============================================================================================
"""
# region CONTROLLER_COMPUTATION
# Compute known position and velocity
agent_position[i,:] = agent_position[i-1,:] + np.array([
    agent_velocity[i-1] * np.cos(agent_theta[i-1]) * time,
    agent_velocity[i-1] * np.sin(agent_theta[i-1]) * time
]) + (noise_type * np.array([generate_noise(noise_mean, noise_sigma), generate_noise(noise_mean, noise_sigma)]))
# print("Agent Position: ", agent_position[i,:], "Agent Velocity: ", agent_velocity[i-1], "Agent Theta: ", agent_theta[i-1])

# Compute relative states
relative_position[i,:] = (target_position[i,:] - agent_position[i,:]
                          + (noise_type
                             * np.array([generate_noise(noise_mean, noise_sigma), generate_noise(noise_mean, noise_sigma)]))) # Compute relative position
relative_velocity[i,:] = [(target_velocity * np.cos(target_theta[i])) # Compute relative velocities
                          - (agent_velocity[i-1] * np.cos(agent_theta[i-1])),
                         (target_velocity * np.sin(target_theta[i]))
                          - (agent_velocity[i-1] * np.sin(agent_theta[i-1]))]
relative_theta[i] = np.arctan2(relative_position[i,1], relative_position[i,0]) # Compute relative heading

# Compute error
error[i] = magnitude(relative_position[i,:]) # Compute error

# Compute desired states
desired_agent_velocity = math.sqrt(target_velocity
                                   + (2* lambda_
                                      * magnitude(relative_position[i,:]) * target_velocity
                                        * abs(math.cos(target_theta[i]))
                                   + (lambda_**2) * (magnitude(relative_position[i,:])**2))) # Compute desired velocity magnitude
agent_velocity[i] = min(desired_agent_velocity, max_agent_velocity) # Compute agent velocity
desired_agent_theta = relative_theta[i] + math.asin((target_velocity
                                                     * math.sin(target_theta[i] - agent_theta[i-1]))
                                                    / (agent_velocity[i] if agent_velocity[i] >= target_velocity else target_velocity)) # Compute desired headi
agent_theta[i] = desired_agent_theta # Compute agent heading
# endregion CONTROLLER_COMPUTATION
```

It follows the formulas and equations given closely, although with the choice of specific execution/evaluation order being the `agent position`, `relative position`, `relative velocity`, `relative theta`, `desired agent velocity`, `agent velocity`, `desired agent theta`, and `agent theta`.

The specific order results in some obvious artifacting notably at the very beginning, although this is purely because the agent's position is evaluated before it is allowed to move for the sake of better demonstrating the controller order and effect.

Report Data Parameters

Simulation Parameters

The simulation runs were done with the following hyperparameters…

delta time	lambda	simulation time
$0.05s$	8.5	$1000ms$
noise sigma	noise mean	max agent velocity
0.5	0	50

Of note here, *simulation time* and *delta time* have different units in an attempt to closely follow the example. Additionally, *noise mean* is set to 0 to center noise and avoid obvious noise skew/bias (and to match example results). Furthermore, all parameters were attempted to match closely with the example code's parameters.

Trajectory Parameters

The following simulation runs use the following parameters for generating the trajectory...

LINEAR	*start*	*end*	*noise mean*	*noise sigma*
	$(0, 100)$	$(100, 0)$	see above	see above
SINE	*start*	*trajectory speed*	*amplitude*	*frequency*
	$(10, 40)$	5	15	0.5
	phase shift	*noise mean*	*noise sigma*	
	0.5	see above	see above	

These parameters came as a result of attempting to match the example runs results decently (not perfectly) closely. There are also circular trajectory parameters used during testing, however since those were not specified in the report, they will not be discussed. For further information, please check the source code

RESULTs

The results of the runs are shown below, all images and animations generated are in the `project_data` folder in the submitted .zip file and also the GitHub project repository: https://github.com/PieFlavr/CPE471-Project-3

All the runs proceeded well and gave results matching expectations.

No Noise Trajectories Summary

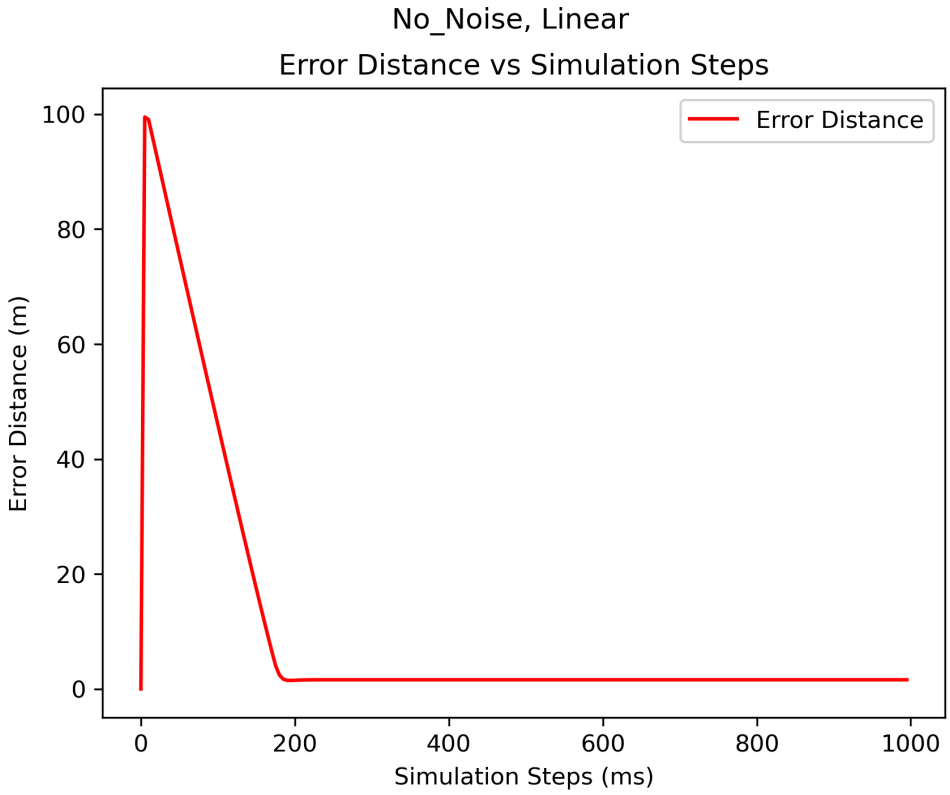
In the no-noise runs, the agent/robot followed the target extremely closely with very little deviation. It was about as expected and matches the example results' patterns. As a matter of fact, there wasn't much trouble getting this to work! The linear and sine trajectories both performed as expected.

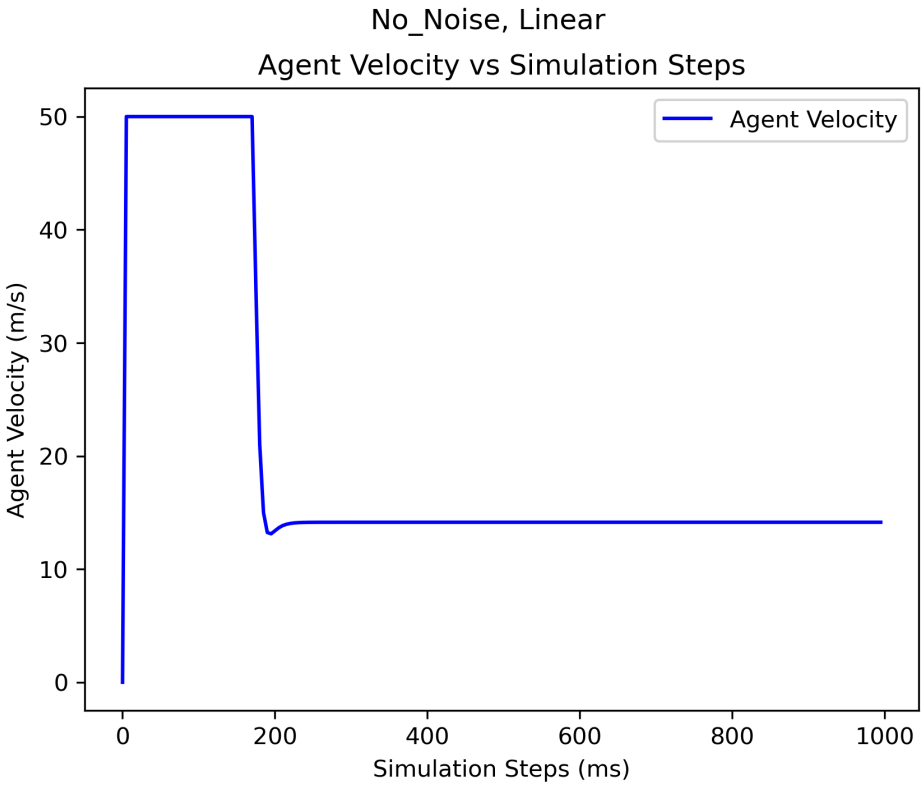
Noisy Trajectories Summary

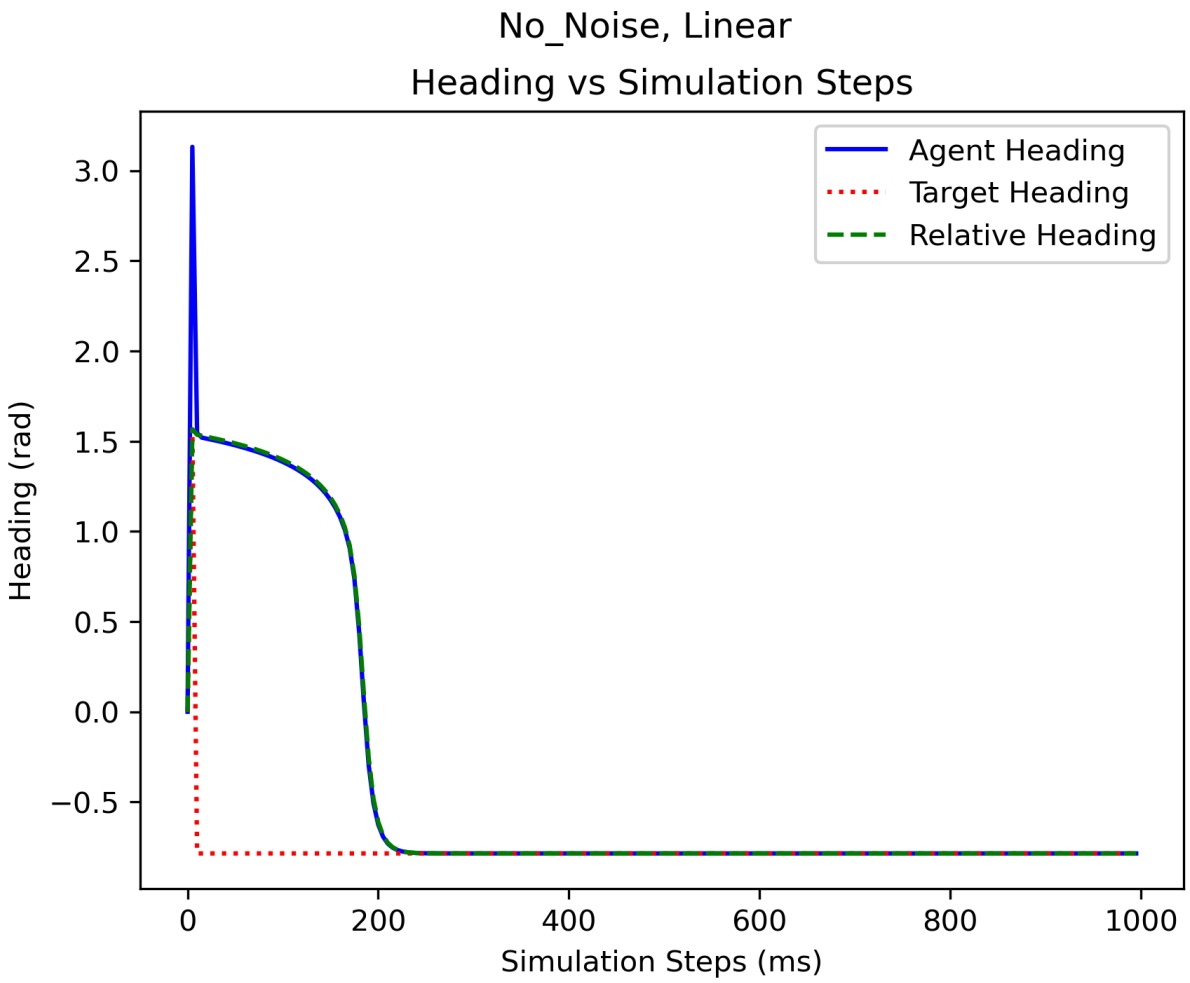
Within the noisy runs, as noise was applied to both the robot and the virtual target, it was quite visibly noisy but the overall following patterns remain either way! It was frankly quite surprising seeing it work so well despite the very clear visible disturbance (of which it was probably a good idea to turn the noise sigma down) but given how well it performed in

tracking the target, I did not deem it necessary. The most disturbed plots as a result of noise was actually heading, which makes sense given especially in low-velocity situations smaller disturbances seem more pronounced due to how angle and theta are calculated for the target.

NO NOISE LINEAR TRAJECTORY DATA

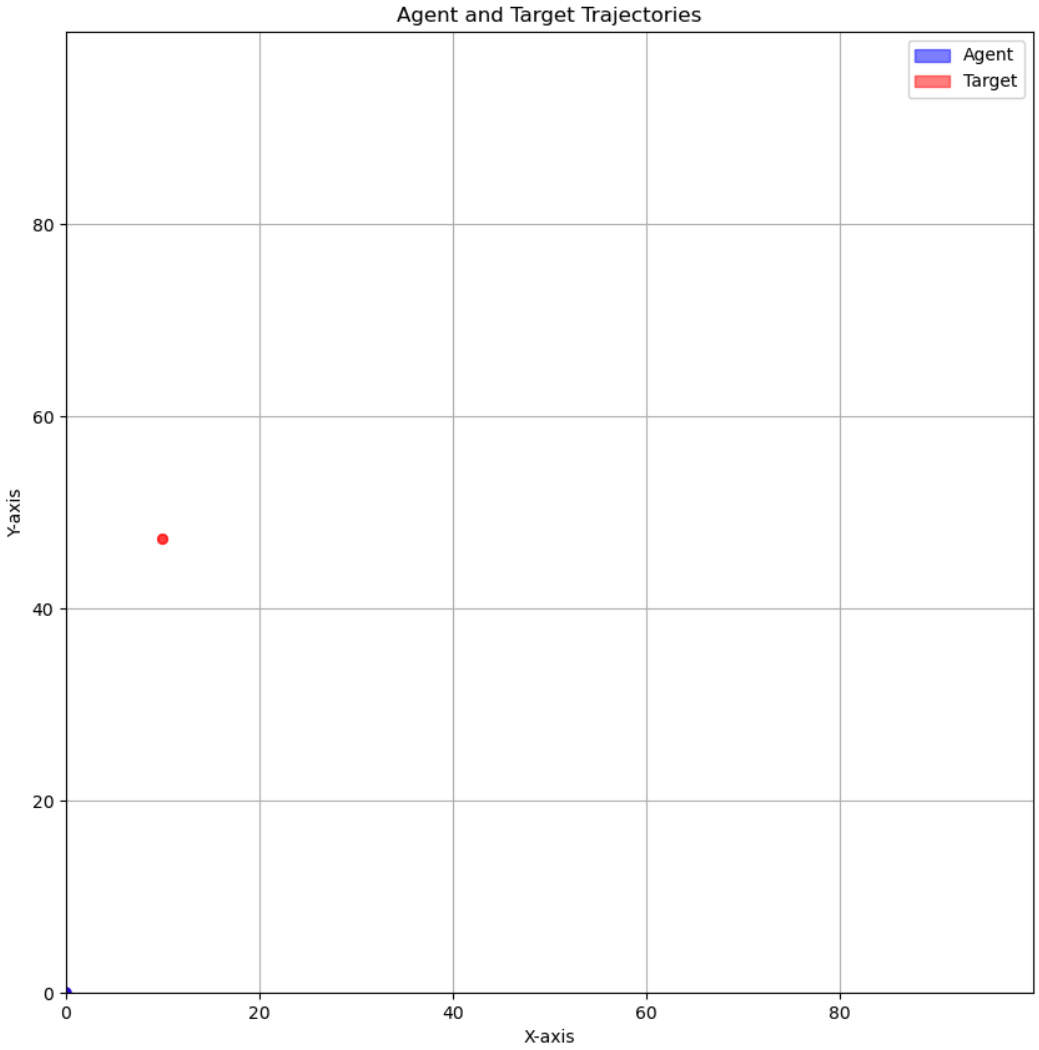

No_Noise, Linear
Error Distance vs Simulation Steps
Error Distance
Error Distance (m)
Simulation Steps (ms)
0
20
40
60
80
100
0
200
400
600
800
1000

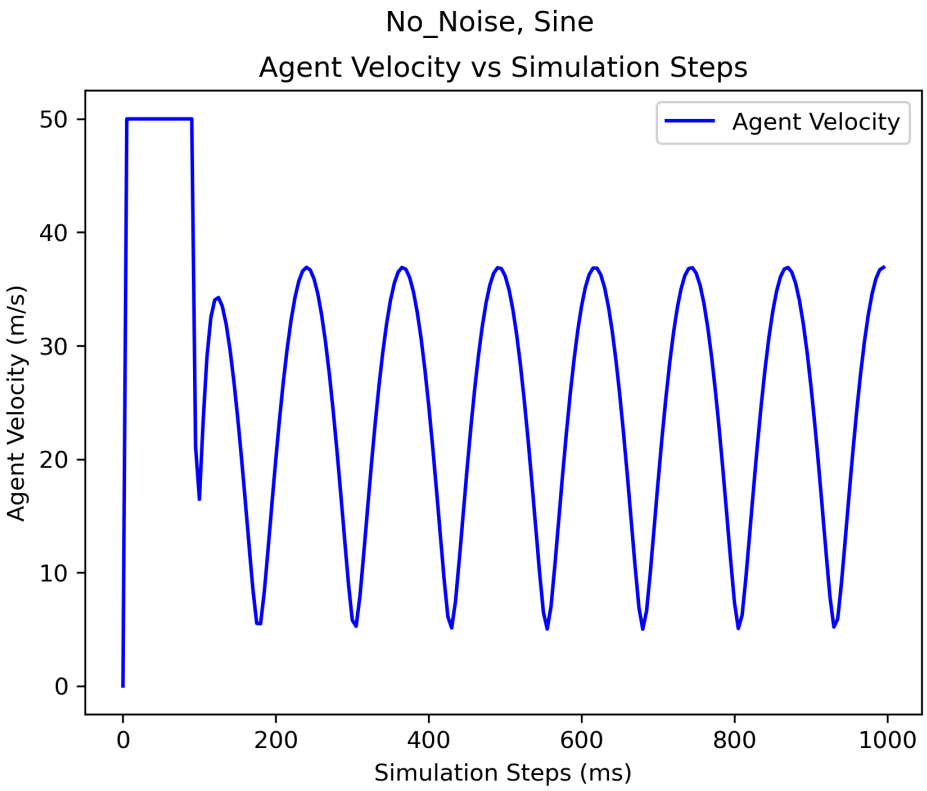

No_Noise, Linear
Agent Velocity vs Simulation Steps
Agent Velocity
Agent Velocity (m/s)
Simulation Steps (ms)
0
10
20
30
40
50
0
200
400
600
800
1000

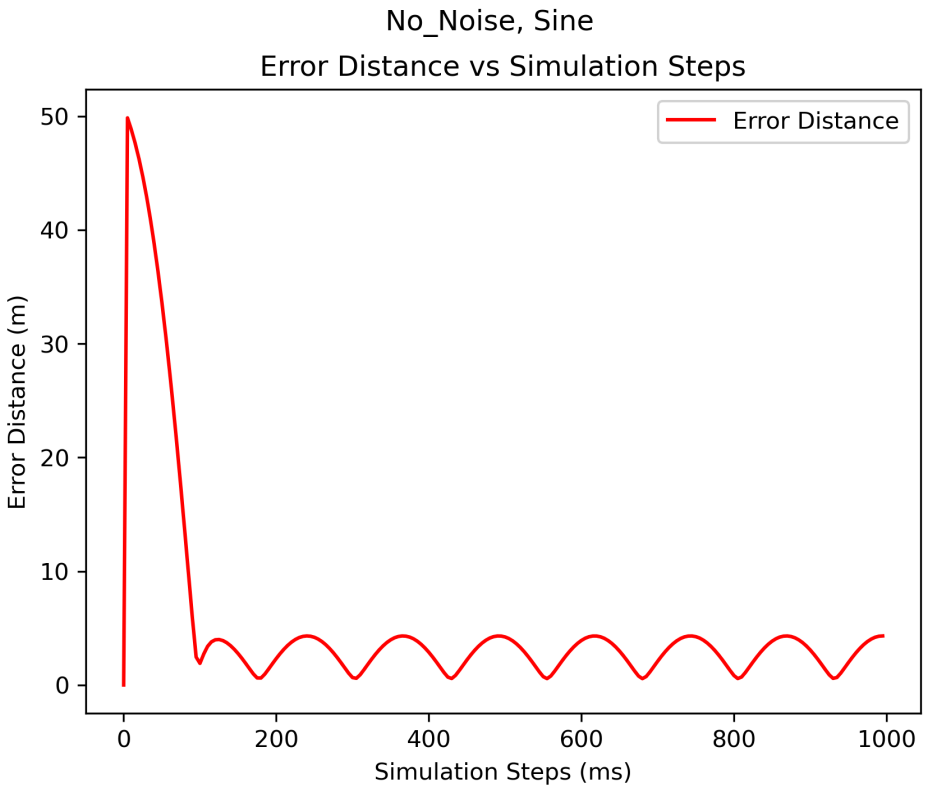

No_Noise, Linear
Heading vs Simulation Steps
Agent Heading
Target Heading
Relative Heading
Heading (rad)
3.0
2.5
2.0
1.5
1.0
0.5
0.0
−0.5
0
200
400
600
800
1000
Simulation Steps (ms)

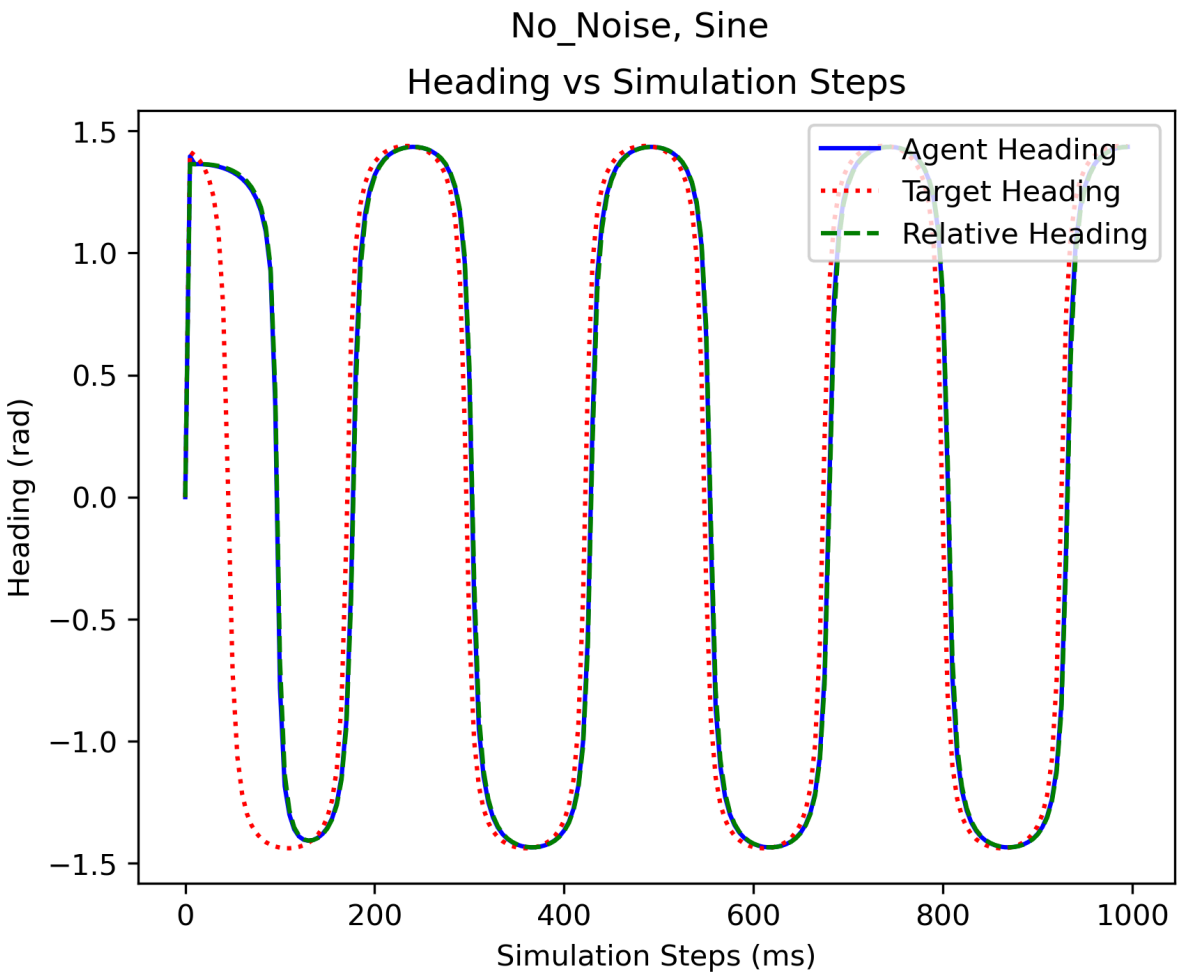
NO NOISE SINE TRAJECTORY DATA

No_Noise, Sine



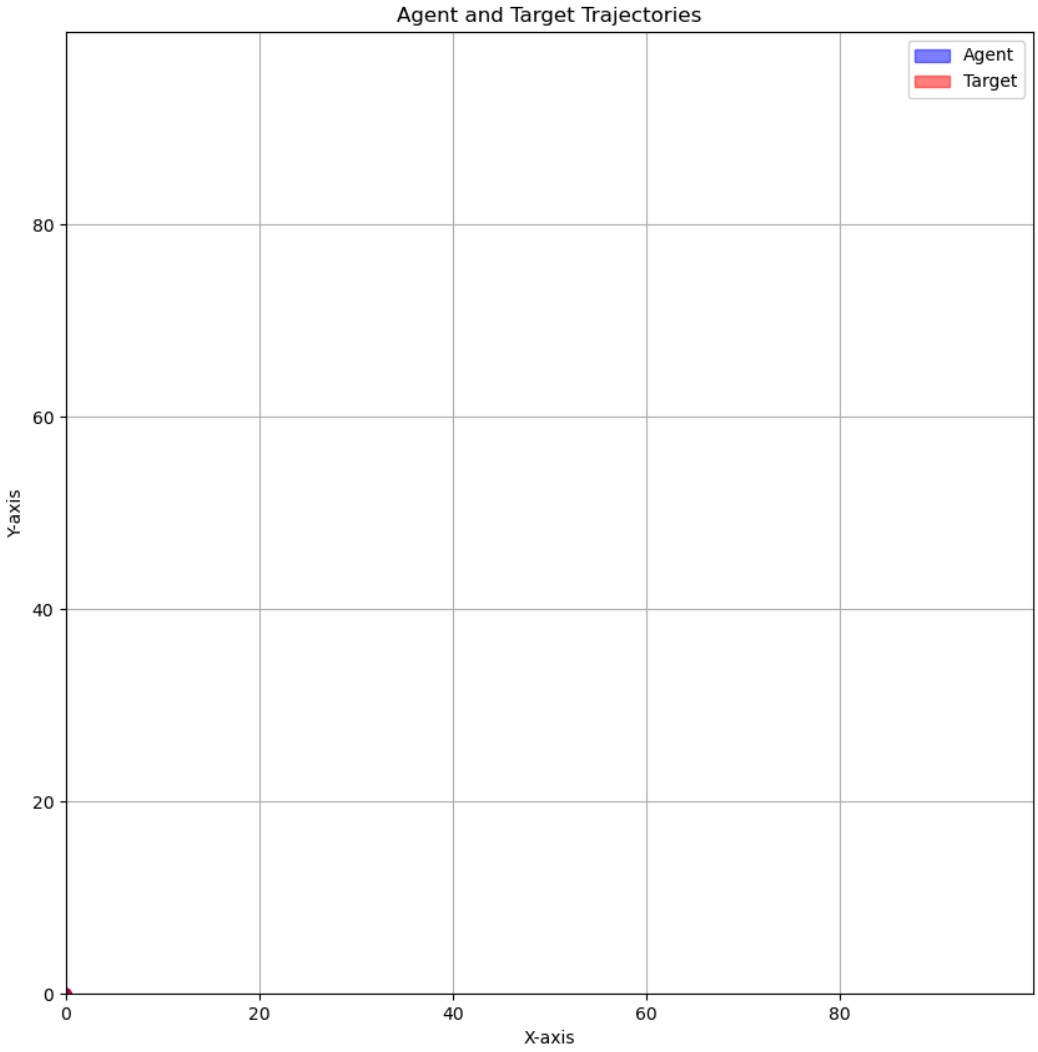

No_Noise, Sine
Agent Velocity vs Simulation Steps
Agent Velocity
Agent Velocity (m/s)
Simulation Steps (ms)
0
10
20
30
40
50
0
200
400
600
800
1000

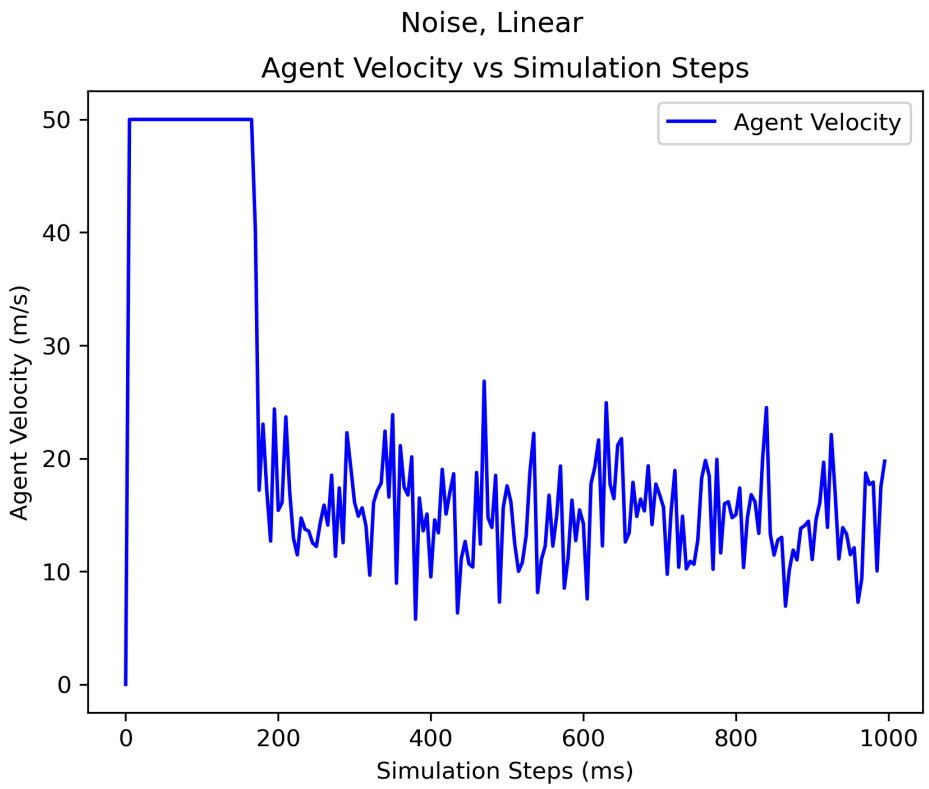

No_Noise, Sine
Error Distance vs Simulation Steps
Error Distance
Error Distance (m)
Simulation Steps (ms)
0
10
20
30
40
50
0
200
400
600
800
1000

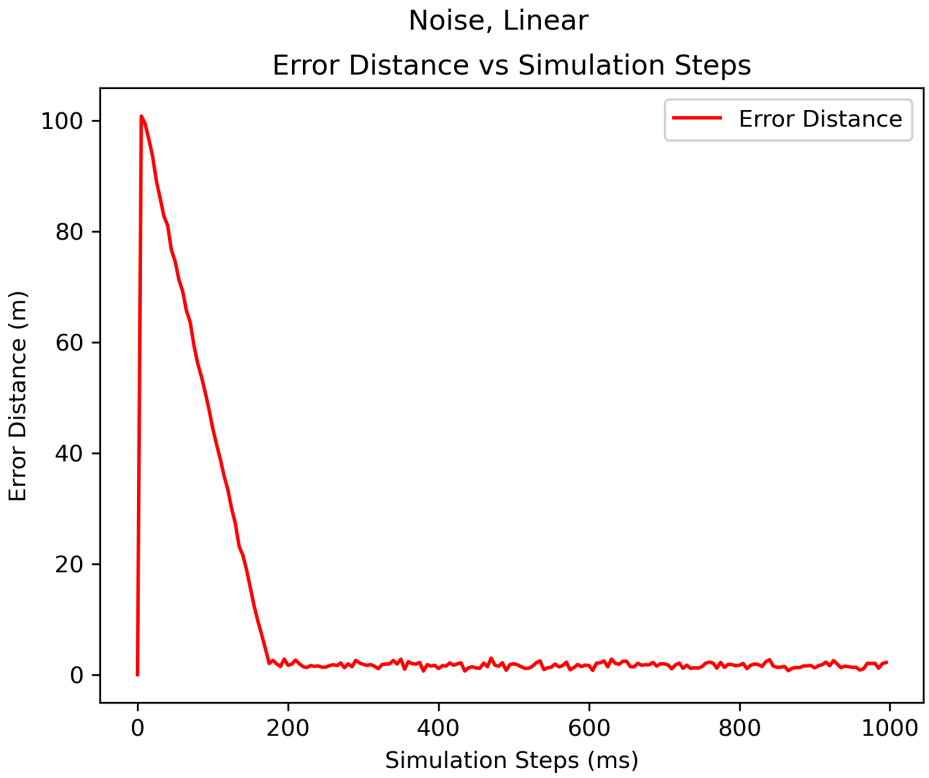

No_Noise, Sine
Heading vs Simulation Steps
Agent Heading
Target Heading
Relative Heading
Heading (rad)
Simulation Steps (ms)
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
0
200
400
600
800
1000

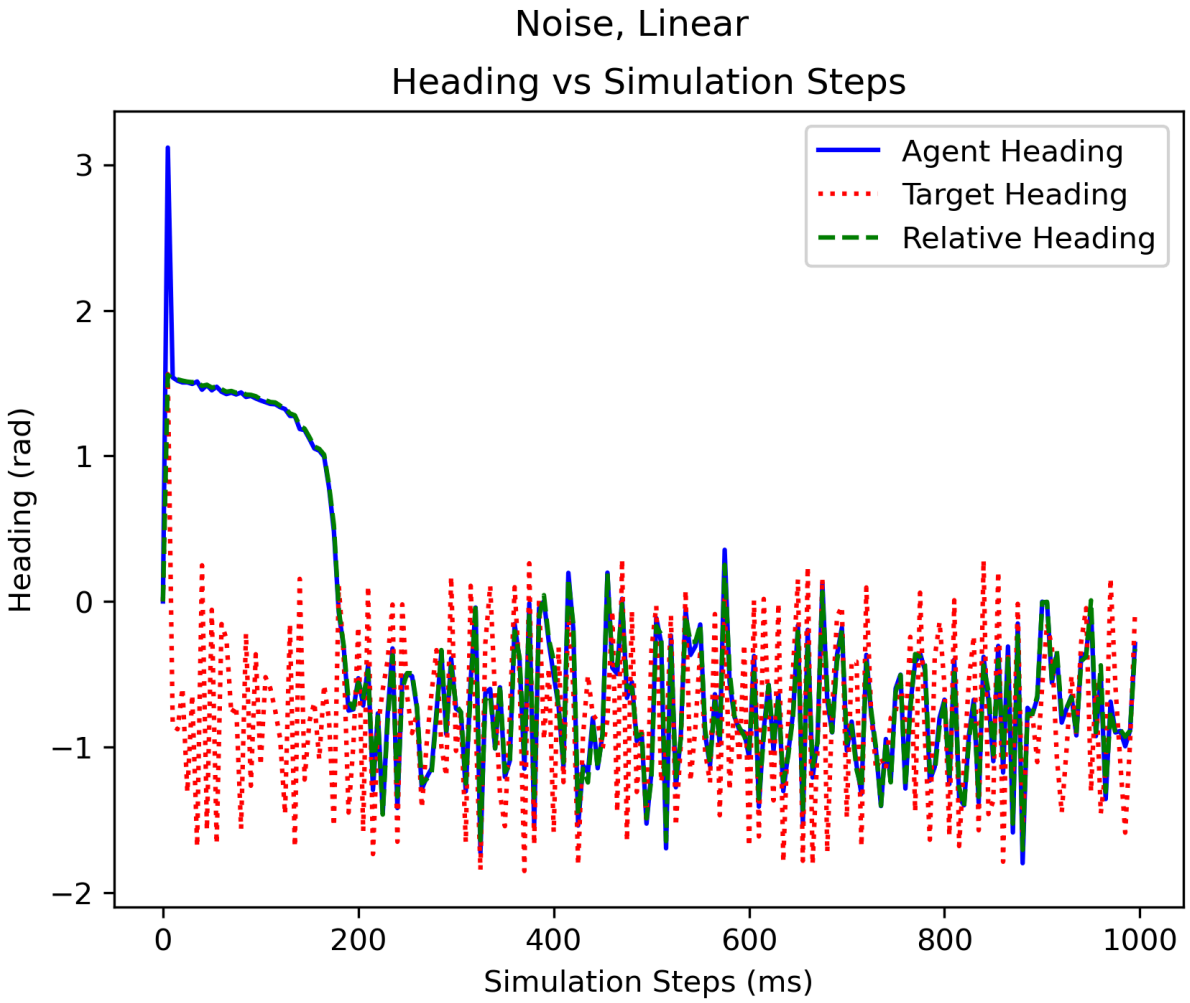
NOISY LINEAR TRAJECTORY DATA

Noise, Linear



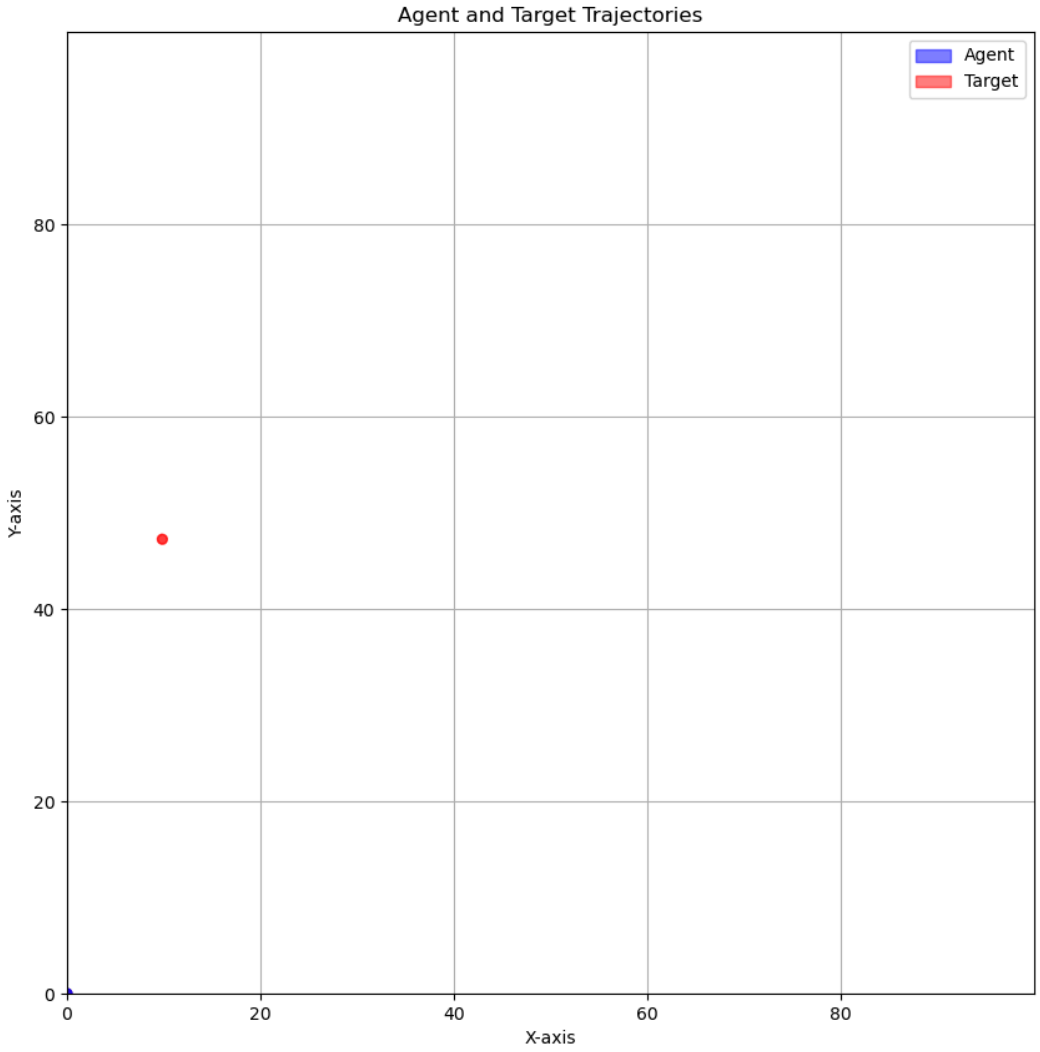

Noise, Linear
Agent Velocity vs Simulation Steps
Agent Velocity
Agent Velocity (m/s)
Simulation Steps (ms)
0
10
20
30
40
50
0
200
400
600
800
1000

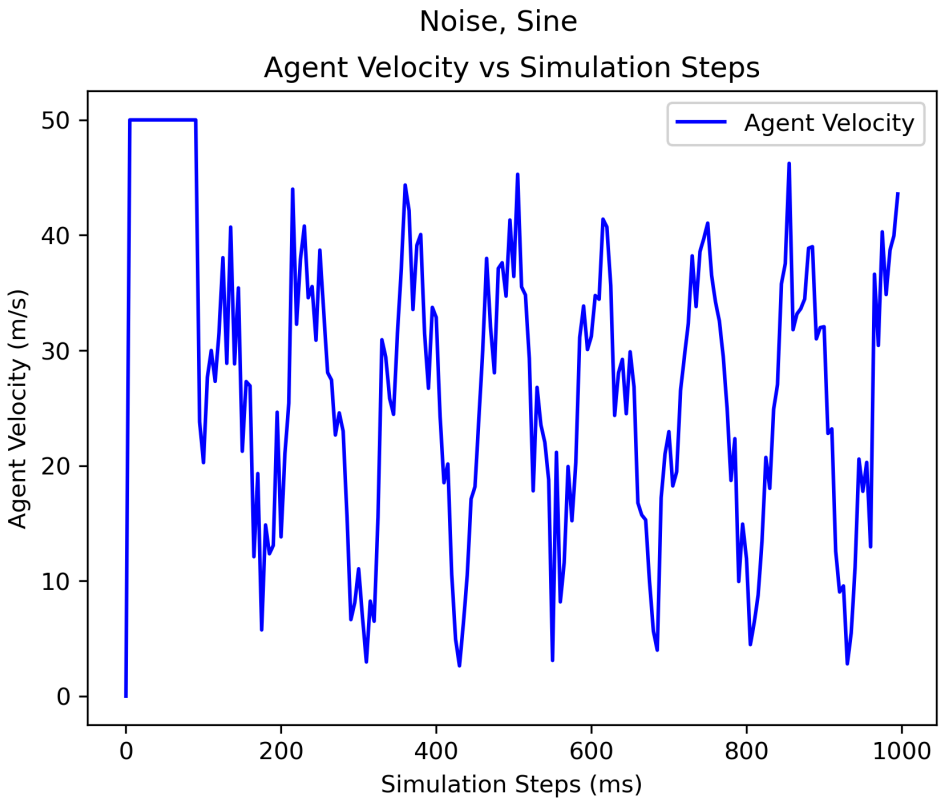

Noise, Linear
Error Distance vs Simulation Steps
Error Distance
Error Distance (m)
Simulation Steps (ms)
0
20
40
60
80
100
0
200
400
600
800
1000

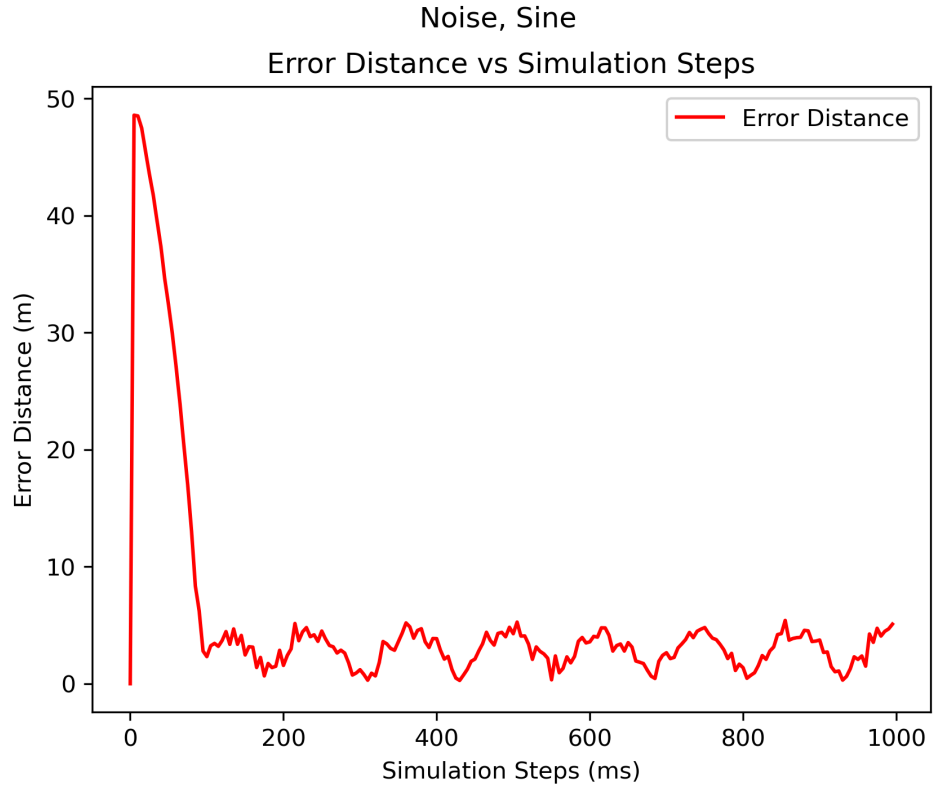

Noise, Linear
Heading vs Simulation Steps
Agent Heading
Target Heading
Relative Heading
Heading (rad)
3
2
1
0
−1
−2
0
200
400
600
800
1000
Simulation Steps (ms)

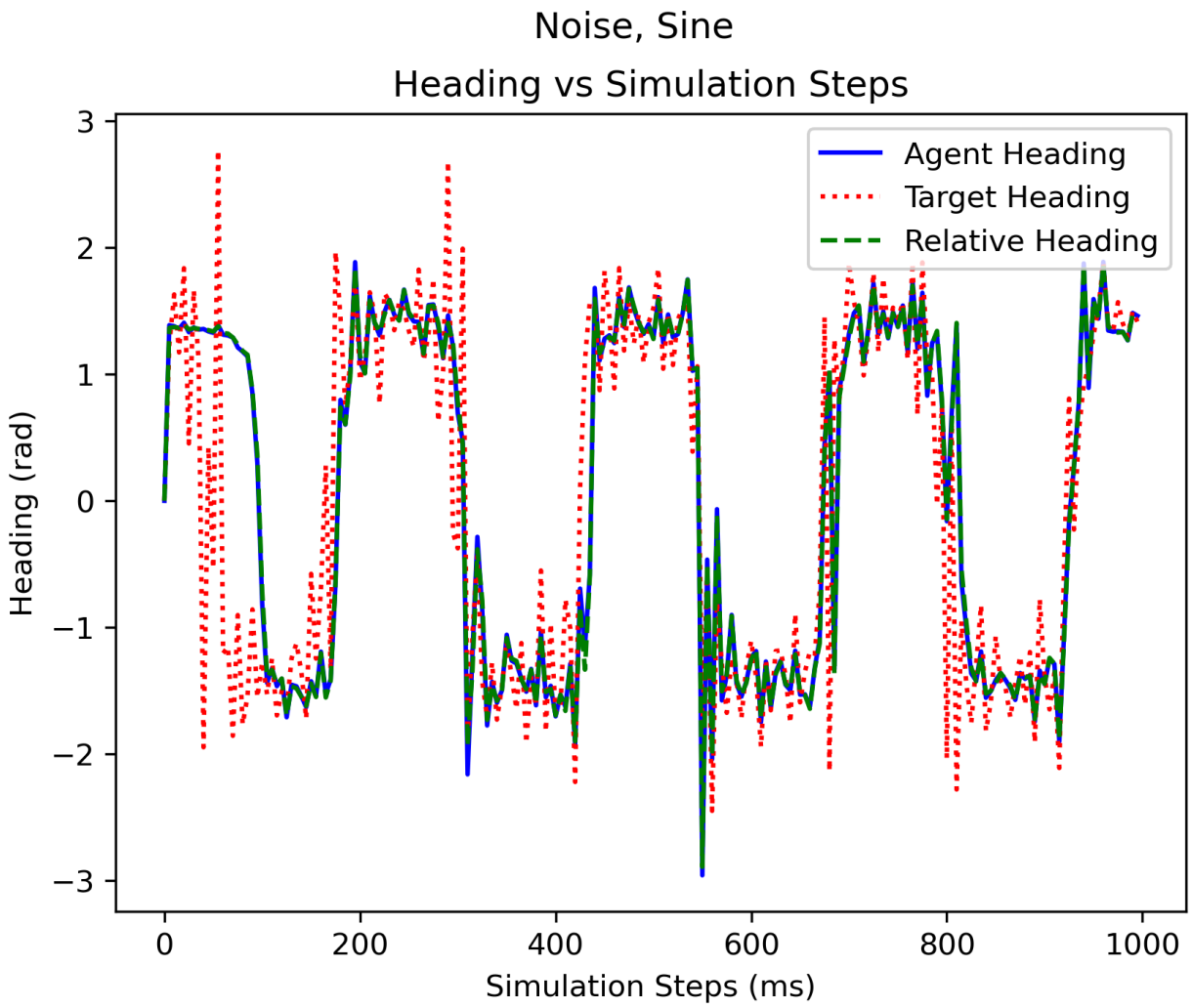
NOISY SINE TRAJECTORY DATA

Noise, Sine




Noise, Sine
Agent Velocity vs Simulation Steps
Agent Velocity
Agent Velocity (m/s)
Simulation Steps (ms)
0
10
20
30
40
50
0
200
400
600
800
1000


Noise, Sine
Error Distance vs Simulation Steps
Error Distance
Error Distance (m)
Simulation Steps (ms)
0
10
20
30
40
50
0
200
400
600
800
1000


Noise, Sine
Heading vs Simulation Steps
Agent Heading
Target Heading
Relative Heading
Heading (rad)
Simulation Steps (ms)
3
2
1
0
−1
−2
−3
0
200
400
600
800
1000