AI in Autonomous Maritime Navigation

Hamza 22p9222 C

February 12, 2025

1 Agent Perspective

1.1 AI Agent Classification and Implementation

Type: Model-based utility agent.
Justification: I used a model-based utility agent because model-based knows the environment will and utility makes smart decisions. With the combination of these two, it will be easy to take a better decision during any hazard.
Perception Mechanisms: GPS, radar, sonar, weather APIs.

Listing 1: Code

```

intitally normal values
 position  = position of ship
 obstacles = no obstacles
 weather  = safe

 getting  preception from sensor like stilite,radar,wetherforcast

 After getting preception make a decision based on current value
 if weather == strom  than slow down
 else if obstacles == in front obstacles than change the position
 else go normaly as were before


class AutonomousShip:
    def __init__(self):
        self.position = (0, 0)
        self.obstacles = []
        self.weather = "calm"

    def perceive(self, sensors):
        self.position = sensors['gps']
        self.obstacles = sensors['radar']
        self.weather = sensors['weather']

```

```python
    def decide_action(self):
        if self.weather == "storm":
            return "slow_down"
        elif self.obstacles:
            return "adjust_course"
        else:
            return "maintain_speed"
```

1.2 Data Integration and Decision-Making

Ans: so i will gathering data from statilite , whether forcasting using apis or using senor of ship I will prioritize prioritize first obstacles than second wether and thired fuel

Listing 2: code

```python
def prioritize_navigation(gps, radar, sonar, weather):
    alerts = []
    if radar.distance_to_nearest < 500:
        alerts.append("collision_risk_high")
    if weather.wind_speed > 30:
        alerts.append("storm_warning")
    if sonar.submerged_objects:
        alerts.append("underwater_hazard")

    if "collision_risk_high" in alerts:
        return "emergency_maneuver"
    elif "storm_warning" in alerts:
        return "reroute"
    else:
        return "proceed_normal"
```

1.3 Single vs. Multi-Agent Systems

Ans: Multi-agent system for coordination, due to which ships will talk to each other to get information from each other, such as the problem condition of rotue, etc.

A real-world example would be to inform another ship that certain route is closed fro example, as we know, perious year a canal get blocked because of a ship incident, due to which a lot of other ship got stuck and route got blocked If we inform each other, it will take safe route to avoid accident

Listing 3: code

```python
import requests

class ShipAgent:
    def __init__(self, id):
        self.id = id
        self.position = (0, 0)
```

```python

    def send_position(self, port_url):
        data = {'ship_id': self.id, 'position': self.position}
        response = requests.post(port_url, json=data)
        return response.status_code
```

2 Environment Perspective

2.1 Classifying the Oceanic Environment

Classification: Partially observable
Because Ocean is too large, it will not get a full observable as we will obserb those area which are required to us

Listing 4: Ocean environment simulation

```python
class OceanEnvironment:
    def __init__(self):
        self.visibility = 10
        self.weather = "calm"
        self.obstacle_density = 0.1

    def update_conditions(self):
        self.visibility = max(0, self.visibility + random.randint(-2, 2))
        self.weather = random.choice(["calm", "storm", "fog"])
        self.obstacle_density = random.uniform(0, 1)
```

2.2 Adapting to Unpredictable Conditions

Strategy: The model will make decisions based on try-and-error and also using probability to make better decisions.

Listing 5: Adaptive navigation algorithm

```python
def adaptive_navigation(current_position, obstacles, weather):
    SAFE_DISTANCE = 1000
    if weather == "storm":
        speed = 10
    else:
        speed = 20

    for obstacle in obstacles:
        if distance(current_position, obstacle.position) < SAFE_DISTANCE:
            return calculate_detour(current_position, obstacle)
    return current_position
```

2.3 Decision-Making Without Standardized Rules

Strategy: Traffic density analysis + COLREGs compliance.

Listing 6: Safety-based routing algorithm

```
def safest_route(traffic_density, weather, fuel):
    safety_score = 0.7 * (1 - traffic_density) + 0.3 * (1 - weather.severity)
    fuel_score = 0.5 * fuel.efficiency

    if safety_score > 0.8:
        return "primary_route"
    elif safety_score > 0.5:
        return "alternate_route"
    else:
        return "emergency_anchorage"
```