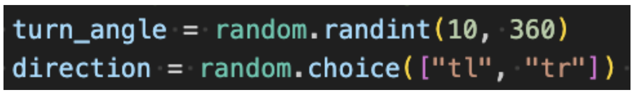
Nicholas Pey, Piotr Bonar, Xavier Sánchez, Carlota Criado

Report Basic Behaviours AAPE

Behaviour 1: Turn

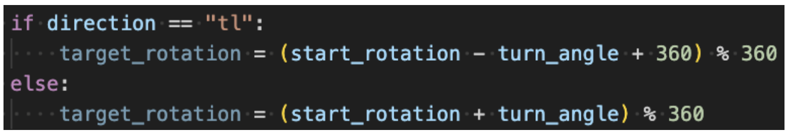
Key Features

1. **Random Turn Parameters**
   The drone randomly selects the degree of rotation between 10 and 360 degrees and randomly chooses the direction (left or right). This randomness simulates a more natural and unpredictable movement pattern



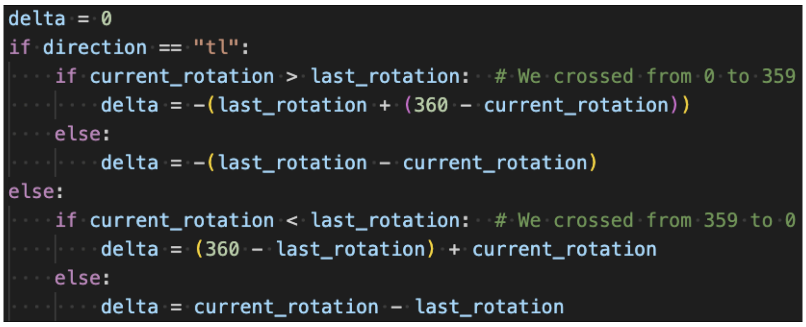
```
turn_angle = random.randint(10, 360)
direction = random.choice(["tl", "tr"])
```

2. **Target Rotation Calculation**
   Once the drone selects a random turn angle and direction, it calculates the target rotation. Modulo arithmetic ensures the rotation stays within the 0-360 range, handling edge cases where the drone might rotate beyond 360 degrees (e.g., from 350° to 10°).



```
if direction == "tl":
    target_rotation = (start_rotation - turn_angle + 360) % 360
else:
    target_rotation = (start_rotation + turn_angle) % 360
```

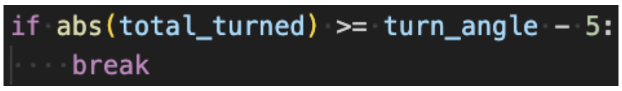
3. **Rotation Monitoring**
   The drone continuously monitors its rotation to track how much it has turned. By comparing the initial rotation with the current one, it tracks the total angle turned. This method provides fine-grained control, allowing the drone to stop once it has reached the desired rotation.



```
delta = 0
if direction == "tl":
    if current_rotation > last_rotation:  # We crossed from 0 to 359
        delta = -(last_rotation + (360 - current_rotation))
    else:
        delta = -(last_rotation - current_rotation)
else:
    if current_rotation < last_rotation:  # We crossed from 359 to 0
        delta = (360 - last_rotation) + current_rotation
    else:
        delta = current_rotation - last_rotation
```

   - The calculation of **delta** is crucial for keeping track of the turn progress. If the rotation crosses the 360° threshold (from 359° to 0° or vice versa), we account for that in the delta calculation.

4. **Completion and Tolerance**
   To account for sensor latency and motor response time, we allow a 5-degree tolerance when completing turns, this prevents overshooting while maintaining sufficient accuracy for navigation. Why 5 degrees? We felt that a 5 degree tolerance balances precision and practicality, tight enough for accurate navigation yet loose enough to prevent delays and jitter from overshooting.

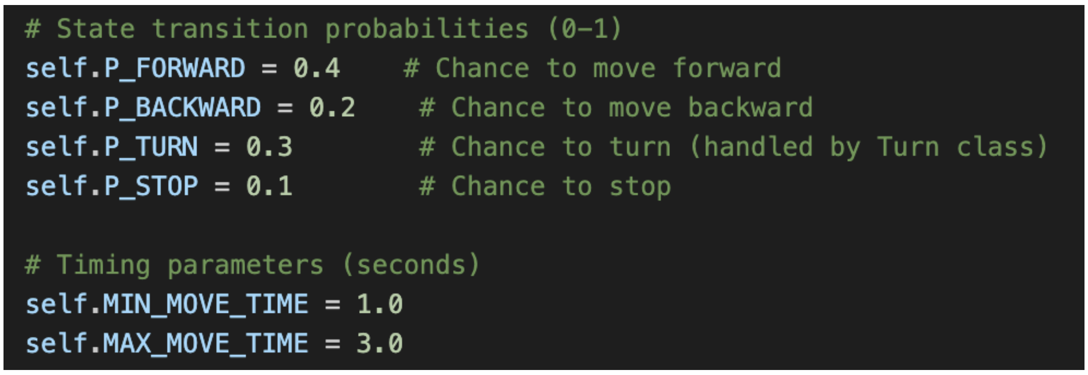


```
if abs(total_turned) >= turn_angle - 5:
    break
```

Behaviour 2 : RandomRoam

Key Features

1. **Probabilistic State Transitions and Movement Timing Control**



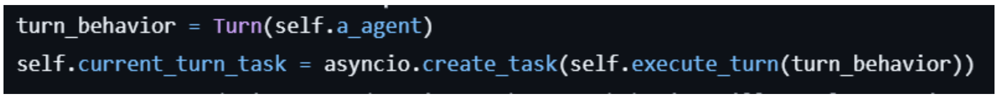
```
# State transition probabilities (0-1)
self.P_FORWARD = 0.4    # Chance to move forward
self.P_BACKWARD = 0.2    # Chance to move backward
self.P_TURN = 0.3        # Chance to turn (handled by Turn class)
self.P_STOP = 0.1        # Chance to stop

# Timing parameters (seconds)
self.MIN_MOVE_TIME = 1.0
self.MAX_MOVE_TIME = 3.0
```

Defined in __init__, these are the predefined probabilities that can be changed accordingly to how the user wants, along with the movement states which all have configurable duration parameters

2. **Precise Turn Handling**

As mentioned earlier, we reuse the "Turn" behavior from the previous class. When the agent decides to turn, the Turn class is invoked, and the agent doesn't need to define its own turn logic. We are taking advantage of the modularity of the "Turn" class, ensuring that we don't need to re-implement the logic for turning.

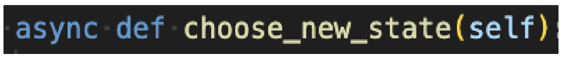


```
turn_behavior = Turn(self.a_agent)
self.current_turn_task = asyncio.create_task(self.execute_turn(turn_behavior))
```

(Note: When deciding to turn and completing the action, it will wait for 2 seconds before the next state as defined in the 'Turn' class).

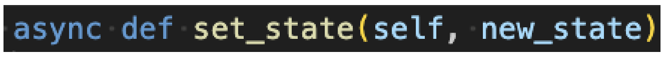
3. **State Management**

State transitions are handled smoothly using a helper function set_state(). This function ensures that before transitioning to a new state, the drone finishes its current action, such as completing a turn or stopping movement.



```
async def choose_new_state(self):
```

This function uses the weighted probabilities and selects a state, before returning the selected state to set_state().



```
async def set_state(self, new_state)
```

This function allows the agent to finish the current action before starting the next one, which ensures smooth transitions between states, preventing the agent from doing multiple actions at once.

Technical Considerations

State Management: There is around 100 ms of delay between state changes
Turn Precision: There is a +-5 degree accuracy, which matches the Turn Class behaviour tolerance (explained above in Turn Class)
Movement Continuity: Always completes current action before transitioning to next action / state

Behaviour 3 : Avoid

Overview

The Avoid class ensures safe movement by:

a. Continuously monitoring proximity sensors
b. Executing **30 degree turn increments** when obstacles are detected (Note: The degree of turn is an arbitrary number which can be changed under the configuration variables)
c. Verifying clearance before resuming forward motion

Key Features

1. **Configuration Variables**

```
self.TURN_ANGLE = 30  # degrees per turn
self.MIN_DISTANCE = 2  # meters to object
self.REQUIRED_HITS = 1   # min sensors detecting obstacle
```

These parameters allow the agent to be fine-tuned for different environments. The TURN_ANGLE defines how much the agent turns at a time, and MIN_DISTANCE controls the sensitivity of the obstacle detection.

2. **Three-State Finite State Machine**
The Avoid behavior uses a three-state finite state machine: MOVING, TURNING, and CHECKING. The agent moves forward by default, turns when it detects obstacles, and checks if the path is clear after turning. This logic ensures that the drone can adapt to its surroundings by checking sensors continuously. It switches between states to react to obstacles dynamically.

3. **Sensor Processing**
The agent checks all raycast sensors every 50ms (`await asyncio.sleep(0.05)`).

```
def count_obstacles(self, hits, distances):
    """Count sensors detecting obstacles within min distance"""
    return sum(
        1 for h, d in zip(hits, distances)
        if h == 1 and d is not None and d < self.MIN_DISTANCE
    )
```

a. **h == 1**: Checks if a sensor ray hit an obstacle (from rc_sensor.sensor_rays[Sensors.RayCastSensor.HIT]).
b. **d < self.MIN_DISTANCE**: Validates if the obstacle is within the 2-meter threshold (from rc_sensor.sensor_rays[Sensors.RayCastSensor.DISTANCE]).
c. **d is not None**: Safeguards against invalid distance readings.

Turning Logic: Since the sensor has 5 sensor rays, we can utilize them to decide a 'smart' way to turn. If any of the first two rays detects an object, the agent will turn to the right and, if any of the others detects an object, it will turn left. This helps the agent navigate the environment more efficiently and smoothly.

Possible Improvements

1. **Obstacle Memory -** This allows the agent to remember where it met an obstacle, and avoid moving towards the same obstacle again
2. **Dynamic Turn Angles - Turn[]**