## *Basic Behaviours Documentation - Group 4*

Alejandra Reinares Guerreros (1665499)
Marino Oliveros Blanco (1668563)
Andreu Gascón Marzo (1670919)
Pere Mayol Carbonell (1669503)

---

### Turn

The drone's behaviour is designed to autonomously select a random turn (both the degree and the direction) and execute that turn. Once the turn is completed, it immediately selects a new turn. The agent prints the chosen degree and direction in the console for verification.

The behaviour operates in two states:

- ➔ **SELECTING**: When a random turn is chosen
- ➔ **TURNING**: When the agent executes the selected turn until the target rotation is reached

This clear separation allows for a straightforward flow: choose a turn, execute it, and then select another turn once the current one is complete.

During the SELECTING state, the random turn selection takes place. First, we implement the degree selection, where a random integer between 10 and 360 is chosen to determine the turn's magnitude. Then, a random choice between "left" and "right" defines the turn's direction. The selected values are printed to the console for easy verification.

Based on the current rotation (self.i_state.rotation["y"]), the target rotation is computed by adding (for right) or subtracting (for left) the chosen degree. Modulo arithmetic (using %360) is used to correctly handle cases where the rotation value exceeds 360 or drops below 0, ensuring proper wraparound.

The behaviour sends the corresponding turn command ("tl" for left and "tr" for right) to initiate the turn. Then, the state changes to TURNING.

During the TURNING state, the current rotation is periodically checked to determine when the target is reached. A tolerance is included in the condition to account for any minor discrepancies: If the absolute difference between the current and target rotation is less than 3, the turn is considered complete. The asyncio.sleep(0.1) prevents constant looping without delay. Upon reaching the target (or if close enough), a stop command ("nt") is sent, and the behaviour resets to the SELECTING state.

The implementation leverages asynchronous programming (via asyncio) to ensure non-blocking execution. Proper cancellation handling (asyncio.CancelledError) is included, so if the task is cancelled, it cleanly stops the turn operation and resets the state.

## RandomRoam

The RandomRoam class makes the drone explore its environment in a random, autonomous way by alternating between moving forward, turning, and stopping. These are the main parts of the class:
States: The drone has four possible states that control its behaviour.

- ➔ Deciding (0): State where the drone chooses the next action (move forward, turn, stop)
- ➔ Moving (1): The drone moves forward for a random amount of time or until it detects an obstacle.
- ➔ Turning (2): The drone makes a random turn to the left or right and stops once it reaches the target rotation.
- ➔ Stopped (3): The drone stops for a short but random duration before deciding the next action

1. **DECIDING** state: When the drone is in the DECIDING state, it randomly selects what to do next.
   - ➔ 60% chance to move forward for 2 to 5 seconds.
   - ➔ 30% chance to turn left or right between 30 and 180 degrees.
   - ➔ 10% chance to stop for 1 to 3 seconds.
2. **MOVING** state: The drone moves forward while checking for obstacles. If it detects one within 0.5 units, it stops and returns to the DECIDING state.
3. **TURNING** state: The drone turns in the chosen direction until it reaches the target rotation, then stops and returns to the **DECIDING** state.
4. **STOPPED** state: The drone stays still for a short time and then switches back to **DECIDING**.

---

## Avoid

The Avoid makes the drone avoid obstacles on its way and adjust its position and orientation to continue moving. It alternates between moving and, when an obstacle is encountered, avoiding so no contact is made with obstacles.
Initially, the drone tries to start moving forward, including exceptions regarding cancellations in case an error happens, so the code is able to handle it, stop the drone and move again to the moving state. Inside the while loop, the first thing to do is to get the sensor's information, including the rays hit, their distances and angles.
During the MOVING state, we create a list of tuples where items are structured as (angle, distance) for each ray using the sensor_hits variable we just created. We then iterate through that list of tuples using enumerate and, if the distance for any of the sensors is smaller than the safe distance and larger than 0, we add that information to a list and set a boolean for obstacles as True.
If continuing, the boolean is set as True, meaning at least one obstacle is found within the safe distance: firstly, we send an asynchronous command indicating the drone to stop its movement. Afterwards, we

use info[0], so the angle of the detected objects, and add them to a list depending on which side they are. Then, we have two lists, containing info for obstacles for those on the right and on the left. Therefore, we compute the average distance to the objects, to know which one has more space. Lastly, for this part, we check for which side the average distance is, and then we send an asynchronous command to take an action moving to the less crowded side. We save the initial rotation before we start moving and change states. We get the initial rotation for future computations.

During the AVOIDING state, we use the saved variable and, by getting the current rotation, compute the difference.

To handle the wraparound problem, since rotations work from 0 to 360, we have to find a way of computing the rotation difference and get a workable result.

We make sure the turn is larger than 45 degrees, we stop rotating and send an asynchronous command for the robot to stop turning, make sure the rays in the centre and both sides are clear. If they are, send another message for the drone to move forward and set the state again as MOVING.

By using these two states, the continuous check of the sensors, the checks during the MOVING state and the differentiation regarding the two directions and the computations to ensure we direct the drone to the open space and then on the AVOIDING state, where we make sure we handle the wraparound problem and ensure the obstacle is no longer on the agent's way whilst handling errors, allows the class to work perfectly and for the drone to avoid obstacles perfectly in a continuous and efficient way.

---