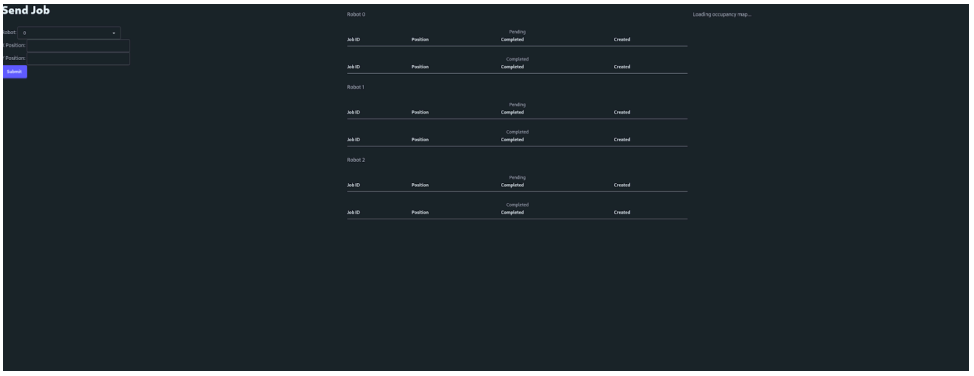
Thibo De Belie, Vince Driesen, Daan Hollands

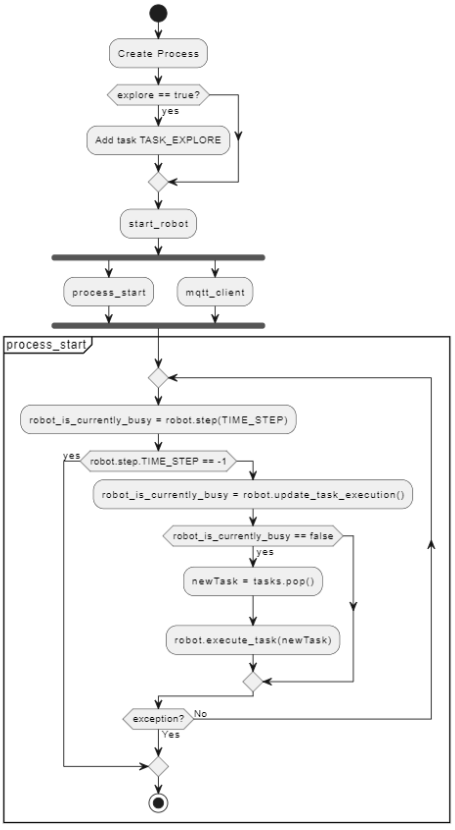
Software engineering for cyber-physical systems

For this project, Python was used to write the code to control the robots that run in Webots. For the dashboard, the SvelteKit framework was used with TypeScript.



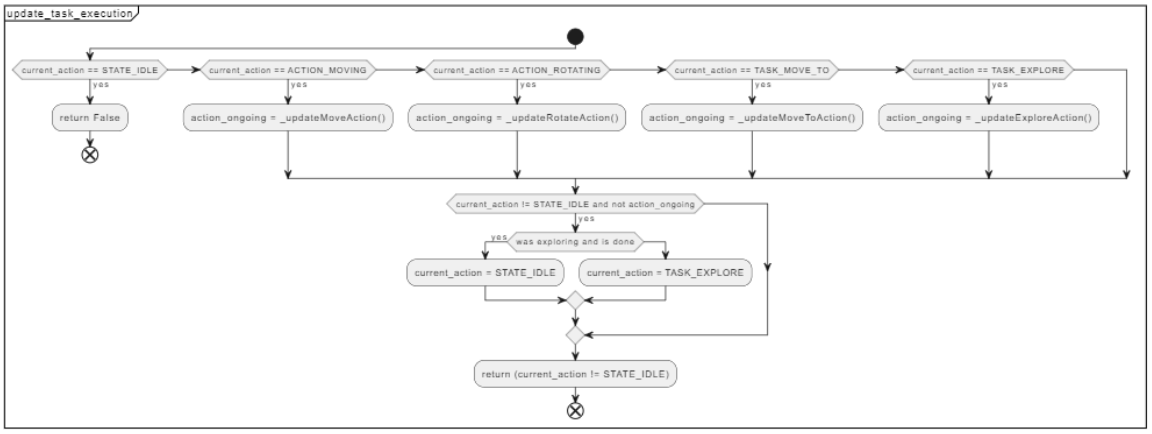
State Machine:

The Plantuml state machine models a robotic process that can handle various tasks that are needed to fulfill the client's wishes that the robots need to be able to do. It begins by creating a process and, if exploration is enabled, adds an exploration task before starting the robot and giving it two different threads: process_start and mqtt_client. The process_start loops repeatedly to check if the robot is busy or available. Task execution is delegated to specific functions based on the task type, each returning whether to stop or continue. The mqtt_client thread changes the tasks array.

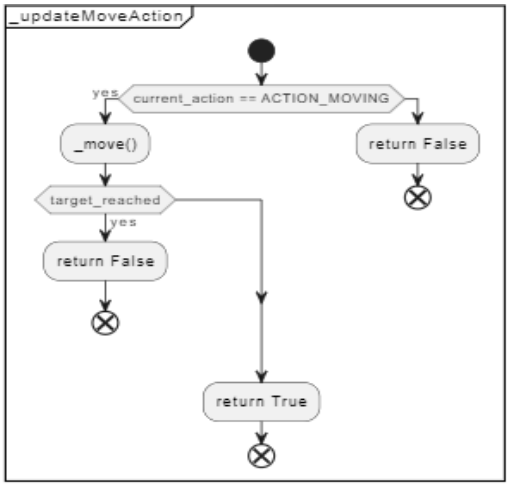


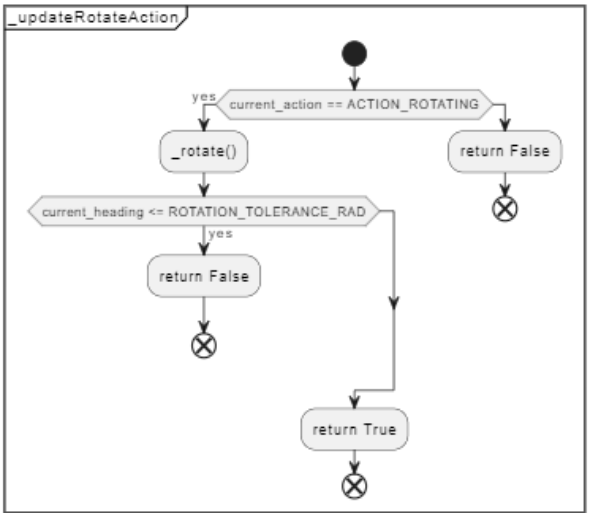
Thibo De Belie, Vince Driesen, Daan Hollands

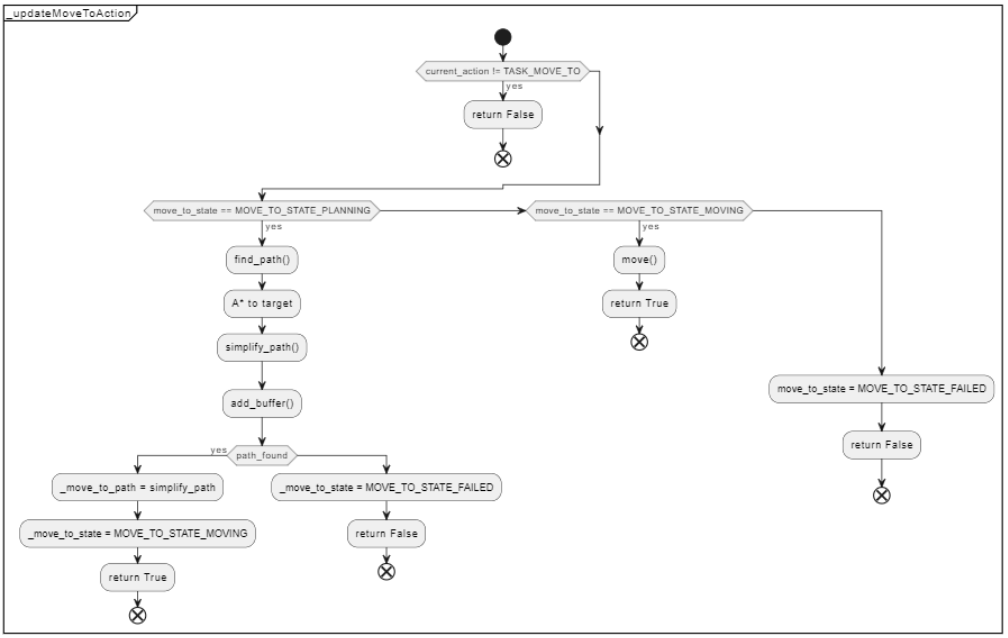
The update_taks_exectuiton component checks the current action and routes it to an update function depending on what the robot is doing at that given moment. These functions are used internally to determine the state and provide feedback to assess progress.



The updateMoveAction and updateRotateAction handle simple movement and rotation and checking whether the goal has been achieved. The updateMoveToAction performs a path planning (using the A* algorithm)

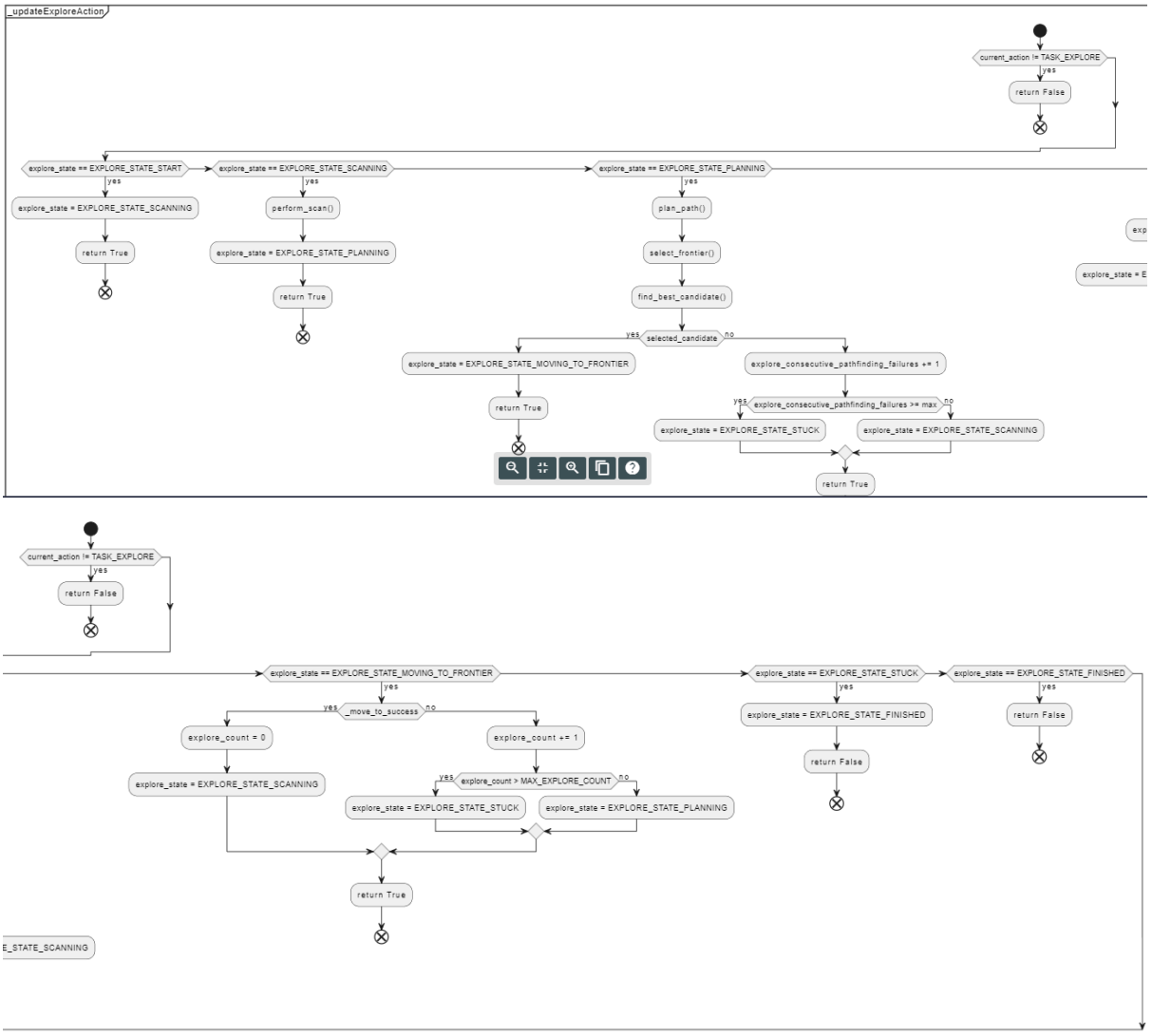






Thibo De Belie, Vince Driesen, Daan Hollands

The exploration routine is defined in updateExploreAction, which includes scanning, path planning, moving to a frontier, and handling failure conditions. If the robot gets stuck or finishes exploring, this function will update the internal state accordingly.



Thibo De Belie, Vince Driesen, Daan Hollands

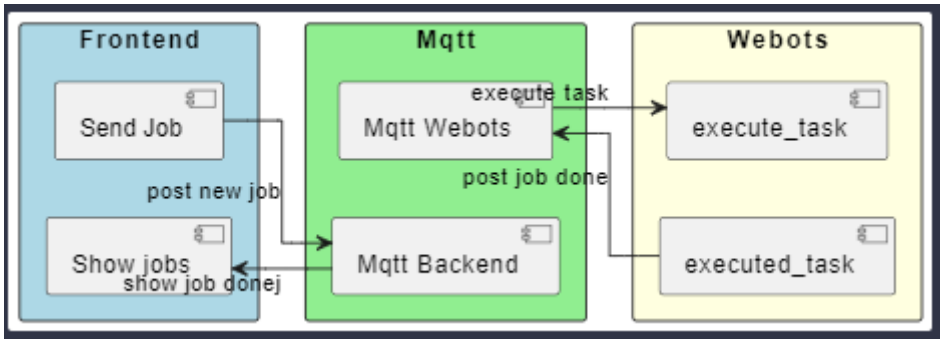
OCCUPANCY MAP

In our solution, we use an occupancy map, which is generated by one robot when all the processes are started. When the discovery is complete, tasks can be sent and robots will be able to move. For the generation of the occupancy map, we use Lidar scanning, and for exploration, we use the implementation of the paper:

> "Mutual information-based exploration on continuous occupancy maps,"
> doi: 10.1109/ICRA.2016.7487635.

This paper gives a way to explore, which is much faster than the normal value iteration process, which is usually implemented.

MQTT



This diagram illustrates a system architecture for job execution using MQTT messaging between the frontend, the MQTT backend, and the Webots simulation environment. The user can submit jobs on the front end and view their status. The MQTT backend will receive and communicate those job requests to the MQTT client for the robots. These will send the execute_taks command to the robots themselves. When the task is completed, the robots will send the executed task to the MQTT client from the robot, translating it to "post job done". This message will allow the front end to update and display the job completion status.

Movement of the differential-drive robot

```
k_rot = 3
k_lin = 10.0

if abs(angle_needed_rel) > ROTATION_TOLERANCE_RAD * 2:
    v_left = -math.copysign(
        self.velocity_norm * self.max_speed, angle_needed_rel
    )
    v_right = math.copysign(
        self.velocity_norm * self.max_speed, angle_needed_rel
    )
else:
    forward_speed = min(
        self.velocity_norm * self.max_speed, k_lin * distance_to_waypoint
    )
    turn_speed = k_rot * angle_needed_rel

    v_left = forward_speed - turn_speed
    v_right = forward_speed + turn_speed

    max_wheel_speed = self.velocity_norm * self.max_speed
    v_left = max(-max_wheel_speed, min(max_wheel_speed, v_left))
    v_right = max(-max_wheel_speed, min(max_wheel_speed, v_right))

self.left_motor.setVelocity(v_left)
self.right_motor.setVelocity(v_right)
```

The code that controls the movement of the robots is implemented using two proportional controllers, this is done with two values: k_rot for rotation and k_linear for linear movement. With these two values, we can implement a basic feedback control system. Whenever the robot is closer to the target distance, the velocity of the wheels will linearly decrease until the robot reaches its target. This factory of decrease is done with k_linear. This is the same for rotation. This will result in much better results, and not over-rotating or too far movement, and the robot will almost exactly stop at the target.