Department of Engineering Cybernetics
Norwegian University of Science and Technology

Mission Planning for a Ground Autonomous Robot

Objective

The main objective of Computer Assignment 4 is to teach students how to solve real problems involving high-level planning for realistic scenarios with mobile autonomous robots, utilizing techniques studied throughout the course. Other objectives include: (1) modeling a typical oil and gas offshore environment for high-level mission planning using the PDDL language, (2) implementing AI planning techniques, and (3) executing a mission using a Turtlebot 3 in the Robotic Operating System (ROS).

Grading

This assignment accounts for 20% of your overall course grade. Students must submit a report and conduct a demonstration using a physical robot. Both the report and the presentation will be evaluated, emphasizing the importance of ensuring that every module of your system runs properly. Students are to work in groups of 2-3 and may choose their own groups. Each group must submit only one copy of the assignment, and all participants will receive the same score.

Deadline and Delivery Details

The report for the assignment must be submitted by 23:59 on Tuesday, April 23rd, 2024. The contents of the report are flexible, and students are free to choose the template they prefer. The report must include answers to the tasks, plots, and conclusions commenting on the results obtained. Additionally, Python, C++, etc. scripts must be attached.

Optionally: students are allowed to solve the assignment task using alternative approaches they find suitable, and make any assumptions they believe are needed. Thus, you have the freedom to select a different mission planning approach. For example, you could use Q-learning, dynamic programming, deep Q-networks, or any other techniques discussed in the TTK4192 - Mission Planning for Autonomous Systems course by A.M.Lekkas (2024).

Introduction

The use of unmanned ground vehicles (UGVs) for inspection and maintenance (I&M) on offshore oil and gas platforms is currently being considered as an alternative to traditional methods, due to the harsh and remote conditions in these environments, see Fig. 1. Although some advancements have been made in the use of UGVs for I&M, there are still challenges to overcome, such as limited autonomy and the ability of robots to operate in potentially explosive environments.
The main motivation for introducing autonomous unmanned ground vehicles (UGVs) in the oil and gas industry is to improve efficiency and safety. Efficiency is improved by utilizing the UGV as a mobile sensor platform inspecting the facilities at low cost and providing repeatable, geotagged and digital inspection data at high quality and low cost. The improved safety comes from higher frequency inspection of system conditions, for example daily inspection of emerging gas leaks, higher frequency (monthly compared to yearly) inspection and analysis of surface conditions.
To increase the level of autonomy of mobile robots for I&M operations, the use of AI planning has been a promising solution. AI planning, also called automated planning, is a sub-area of artificial intelligence (AI) and refers to the process of using algorithms and computational models to create a sequence of actions that achieve a specific goal or set of goals. The goal of AI planning is to automate decision making and problem solving, by allowing computer systems to create and execute plans in real-world environments [3].

Use case problem for Assignment 4:

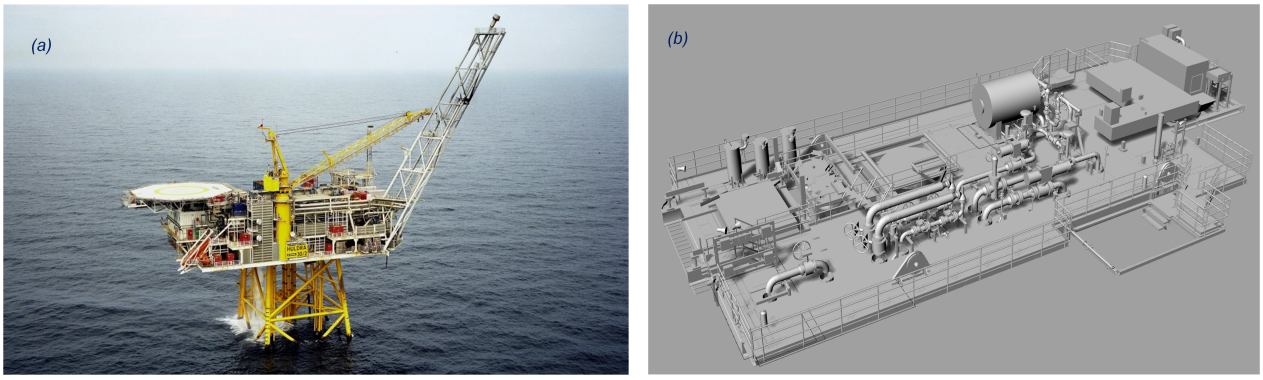

Figure 1: (a) Huldra gas processing plant (courtesy of Equinor), (b) Huldra Upper Part Centered.

This section presents a Use Case that needs to be solved, as shown in Fig. 2. The problem is derived from inspection and maintenance operations in oil and gas processing plants. In summary, it is required to perform an inspection round using a ground robot, capturing images, and carry out specific actions with valves and pipes. To solve this problem, students must use the algorithms learned throughout the course and some concepts that will be given in upcoming lectures. For pseudo-codes and extra information, students may refer to the book by Ghallab et al. [3].

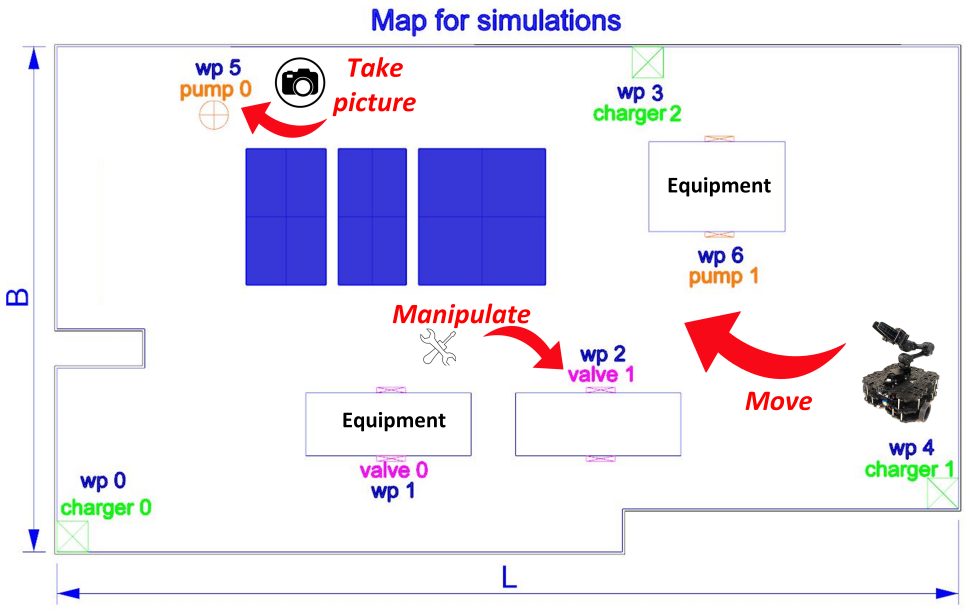

Figure 2: Use case problem for assignment 4

The use case sensors and equipment is composed by:

- 7 waypoints
- 2 valves
- 2 pump
- 3 charge stations

The actions allowed for the robot are:

- move robot between waypoints
- take a picture of pumps
- inspect/manipulate valves

- charge robot's battery

Fig. 3 presents an overview of the modules that compose the system, which will need to be programmed in Assignment 4. Thus, the assignment will cover the modules of AI planning, path-finding, and robot configuration. As the GNC module is outside the scope of this course, this module will already be programmed and delivered in the attached files.

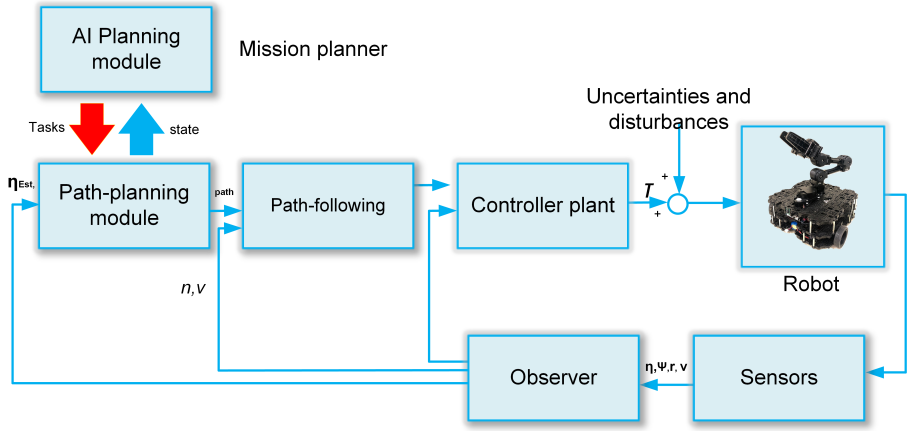

Figure 3: AI planning into a traditional GNC system

Turtlebot3 with OpenManipulator-X

In this assignment, we are going to use the simulator and physical robot of Turtlebot3 and OpenManipulator-X. The robot is the Turtlebot3 waffle-pi (see Fig. 4), developed by the ROBOTIS company [1]. It is a small, affordable, programmable, ROS-based mobile robot designed for use in education, research, and product prototyping. Its main advantages lie in the packages it provides, including 'bring up,' 'navigation,' 'SLAM,' and 'tele-operation.' 'Bring up' initiates the hardware and state publisher, 'navigation' starts nodes necessary for autonomous navigation, 'SLAM' starts nodes required for SLAM operation, and 'tele-operation' allows for manual control over the robot. Additionally, the TurtleBot3 can function as a mobile manipulator capable of manipulating objects by attaching a manipulator such as the OpenManipulator-X.

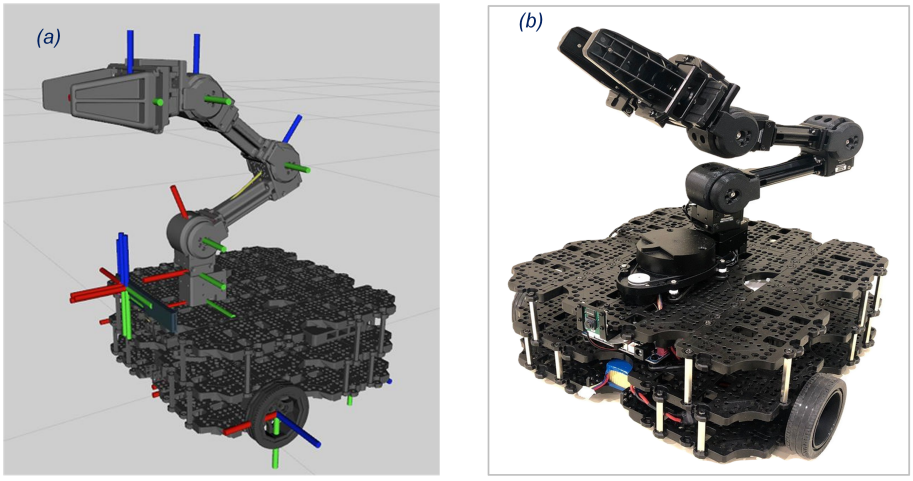

Figure 4: Waffle-pi Turtlebot3 with Open-x manipulator (a) Gazebo simulator; (b) physical robot.

[1] https://emanual.robotis.com/

Part I
Code installation

In order to develop your Mission planning system for Computer Assignment 4, the first step is to download and install a virtual machine containing all the ROS packages and source code for the assignment. Please note that you will need to have previously installed the **VirtualBox 7.0** program and have at least **15GB** of free disk space. Follow the steps below to download and install:

- Download the file named "ttk4192-ubuntu20.04.ova" from NTNU onedrive Link.
- Open the VirtualBox program and go to *file* >> *Import appliance* and then navigate to the location of the downloaded "ttk4192-ubuntu20.0.ova" file, as in Fig. 5.
- When the installation is completed, start the virtual machine "ttk4192-ubuntu20.0; **password: ttk4192**
- Verify if the installation is correct, open a terminal and execute the following commands:

```
$ sudo apt-get update
$ export TURTLEBOT3_MODEL=waffle_pi
$ roslaunch turtlebot3_manipulation_gazebo turtlebot3_manipulation_gazebo.launch
```

  if the installation is correct a gazebo window should open presenting the turtletot3 and manipulator.

Note: The virtual image was configured for 8 cores CPU and 4 GB memory, however, you can change these parameters in the VirtualBox configuration setting.

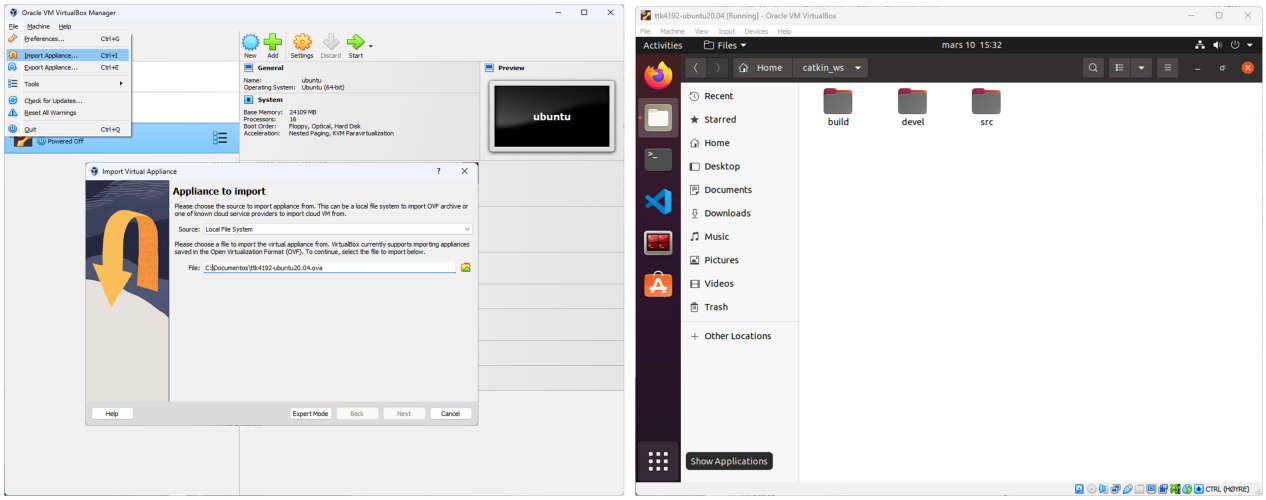

Figure 5: Virtual machine for computer assignment 4

Part II
AI planning

In this first part the objective is to develop a high-level AI mission planner and compute a plan that solves the inspection round problem in Fig. 2. Students are free to choose any AI planner of their preference, i.e. STRIPS, HTN, Graphplan, Temporal planner, etc., most of the algorithms can be found in Ghallab et al. [3] book.
This year, students are encouraged to use the STP temporal planner to solve the mission planning problem. The STP temporal planner offers relevant advantages in comparison with other classical methods, as it allows for the computation of a more realistic representation of actions. This planner is already **pre-installed** in the virtual machine 'ttk4192-ubuntu20.04' and can be found in *catkin_ws/src/AI-planning/temporal-planning*. In case you wish to install from the source files, the installation steps can be found at: https://github.com/aig-upf/temporal-planning.

Temporal STP planner: The *simultaneous temporal planner* (STP) relies on a transformation process that converts temporal planning into classical planning problem, and constructs a temporal plan by identifying a sequence of classical actions which not only resolve the problem but also adhere to a specified set of temporal constraints. The main novelty

of STP, as it was presented in [2], is its capability to address problems that necessitate simultaneous events, for example, the temporal actions have to be organized in way that allows two or more of their effects occur concurrently. To facilitate this, STP breaks down each event into three distinct phases: the first phase in which temporal actions are scheduled to the end, the second phase handles the occurrence of simultaneous effects, and the third phase focuses on temporal actions are scheduled to the start. Thus, a temporal planning problem is a tuple:

$$P =< F, A, I, G > \tag{1}$$

where the fluent set F, initial state I and goal condition G are defined as for classical planning. The action set A consists of temporal or durative actions $a \in A$ composed of:

$$\begin{cases} \text{d(a):} & \text{duration of } a \\ \text{pre(a):} & \text{preconditions of } a \\ \text{add(a):} & \text{add effects} \\ \text{del(a):} & \text{delete effects} \end{cases} \tag{2}$$

Although a has a duration, its effects apply instantaneously at the start and end of a, respectively. The preconditions $pre(a)$ is also checked instantaneously. Comprehensive details on the temporal logic equations, lemmas, and proofs for the STP planner are available in [2, 4].

1 PDDL Domain

In this task, the students must create a PDDL domain based on the use case presented in Fig. 2. The domain must contain variables, actions, and conditions. For more details about PDDL domains see https://planning.wiki/ref/pddl/domain.

1.a *Task:* Build the PDDL domain based on the Use Case, consider the allowed actions and agents.

Hint: In the virtual machine "ttk4192-ubuntu20.04", in the folder *catkin_ws/src/AI-planning*, students will found an example of the PDDL domain, which shoudl extend with more actions. The PDDL domain should have a similar structure as it follows:

```
(define (domain mission-planning-mass)
(:requirements :typing :durative-actions)
  (:types
     ship - vehicle
     port - location
     cont team_rep team_refuel - subject
     route)
  (:predicates
    (at ?physical_obj1 - subject ?location1
    - location)
    (available ?vehicle1 - vehicle)
    (busy ?vehicle1 - vehicle)
    (loaded ?subject1 - subject ?vehicle1 - vehicle)
    (connects ?route1 - route ?location1 - location
    ?location2 - location)
    (in_port ?location1 - location ?port1 - port)
    (route_available ?route1 - route)
    )

(:functions
           (distance ?O - location ?L - location)
           (route-length ?O - route)
        (speed ?V - vehicle)
            )
  (:durative-action move-ship
       :parameters ( ?V - vehicle ?O - location ?Port -
       port ?L - location ?Port1 - port ?R - route)
       :duration (= ?duration
                 (/ (route-length ?R) (speed ?V)))
       :condition (and
            (at start (at ?V ?O))
                    (at start (in_port ?O ?Port))
                    (at start (in_port ?L ?Port1))
```

```
                    (at start (connects
                    ?R ?Port ?Port1))
       )
       :effect (and
          (at start (not (at ?V ?O)))
                  (at end (at ?V ?L))
        ) )
```

2 PDDL problem

In this task, the students must create a PDDL problem based on the use case presented in Fig. 2.

2.a *Task:* Build the PDDL problem based on the use case and the presented elements.

Hint: In the virtual machine "ttk4192-ubuntu20.04", in the folder *catkin_ws/src/AI-planning*, students will found an example of the PDDL problem, which can extend. The PDDL domain should have a similar structure as it follows:

```
(define (problem robplan) (:domain robplan)
(:objects

turtlebot0 - robot
camera0 - camera
gas_sensor0 - gas_sensor
ultra_sensor0 - ultra_sensor
robo_arm0 - robo_arm
battery_station0 battery_station1 battery_station2 - battery_station

valve0  - valve
pipe0  - pipe
pump0  - pump
waypoint0 waypoint1 - waypoint
d01 d02 d03 d04 d05 d06 d07 - route

)
(:init
(= (speed turtlebot0) 1)

(connects d01 waypoint0 waypoint1)
(connects d02 waypoint0 waypoint2)
(connects d03 waypoint0 waypoint3)
(connects d0b0 waypoint0 battery_station0)
(connects d0b1 waypoint0 battery_station1)

(at valve0 waypoint4)
(at pump0 waypoint2)
(at charger0 waypoint6)
(at pipe0 waypoint0)

(at gas_ind0 waypoint0)
(at gas_ind1 waypoint1)

(= (route-length d01)   2.78   )
(= (route-length d02)   9.36   )
(= (route-length d03)   6.63   )

(available turtlebot0  )
(available camera0  )
(no_seals_check valve0  )
(no_seals_check valve1  )
(at turtlebot0 waypoint6  )
(no_photo valve1  )
(no_photo valve2  )

)
(:goal (and
(at turtlebot0 waypoint0)
```

```
))
(:metric minimize (total-time))
)
```

3 Compute a plan

In this section, the students are requested to compute a plan for the use case presented in 2 using the AI planner that students have chosen.

3.a *Task:* Using your chosen planner and the PDDL domain, solve the following mission planning problem presented in Table 1. Present the calculated plan in a table, similar to the Table 2. It is important to mention that your PDDL domain can be slightly different depending in which AI planner you had chosen.

Hint: What is asked in this task is to run the temporal planner and solve the problem, to run the pre-installed STP planner, execute the below command in a new terminal:

```
$ source /home/ntnu-itk/catkin_ws/src/AI-planning/bin/activate
$ cd /home/ntnu-itk/catkin_ws/src/AI-planning/
$ python2.7 /home/ntnu-itk/catkin_ws/src/AI-planning/temporal-planning/bin/plan.py stp-2
/home/ntnu-itk/catkin_ws/src/AI-planning/temporal-planning/domains/ttk4192/domain/domain.pddl
/home/ntnu-itk/catkin_ws/src/AI-planning/temporal-planning/domains/ttk4192/problem/problem.pddl
```

Note that: last three lines must be written as a single command, separated by spaces. In case of successfully calculation the message will be: *Solution found.*, and the computed plan can be found in temporal-planning folder and is named as:*tmp_sas_plan.1*.

Table 1: Initial state and goals.

Case	Initial state		Goal
1	(at turtlebot waypoint 0)	→	(at turtlebot waypoint 3)
	(no check valve0)	→	(check valve0)
	(no check valve1)	→	(check valve1)
	(no check pump0)	→	(check pump0)
	(no check pump1)	→	(check pump1)
	((speed turtlebot) 0.18)		-

Table 2: Example of calculated plan.

n.	Action
1:	move turtlebot0 waypoint0 waypoint1 d01
2:	picture eo turtlebot0 waypoint1 valve0
3:	move turtlebot0 waypoint1 waypoint3 d13
4:	picture eo turtlebot0 waypoint3 valve2
5:	move turtlebot0 waypoint3 waypoint2 d32
6:	picture eo turtlebot0 waypoint2 valve1
7:	charge battery waypoint0 battery0
8:	move turtlebot0 waypoint0 waypoint5 d05
9:	picture ir turtlebot0 waypoint5 pump0
10:	move turtlebot0 waypoint5 waypoint6 d56
11:	picture ir turtlebot0 waypoint6 pump1
12:	move turtlebot0 waypoint6 waypoint7 d67

Part III
Developing GNC system for Turtlebot3 in ROS

In this part of the assignment, the students will install the simulator of the Turtlebot3 + OpenManipulator-X and program a mission planning system in ROS. The mission planning system is composed by the following modules, AI planning, path-finding, guidance and control (GNC) and robot simulator. The guidance and control modules are already programmed in the script called *mission_planning_ttk4192.py*. Note that, the system must be programmed in Ubuntu 20.04, ROS Noetic and python 3.8.10, check the *VirtualBox, Ubuntu and ROS Noetic Installation guide* for additional details.

4 Installation of Turtlebot3 waffle-pi with Manipulator

4.a *Task:* Install the packages for the turtlebot3 model waffle-pi + OpenManipulator-X.
Hint: Follow the instruction at Robotis e-manual at section of Manipulation, be aware of selection the correct ROS distribution (Noetic), and copy commands for **Remote PC** only, or run the below commands:

Open a new terminal window (CTRL+ALT+T) and execute the following commands:

```
$ cd ~/catkin_ws/src
$ git clone https://github.com/ROBOTIS-GIT/turtlebot3_manipulation.git
$ git clone https://github.com/ROBOTIS-GIT/turtlebot3_manipulation_simulations.git
$ git clone https://github.com/ROBOTIS-GIT/open_manipulator_dependencies.git
$ sudo apt install ros-noetic-ros-control* ros-noetic-control* ros-noetic-moveit*
$ cd ~/catkin_ws && catkin_make
```

To test that the package is installed correctly, you can start a Gazebo simulation as seen in Fig. 9, executing with the following commands:

```
$ roscore
$ export TURTLEBOT3_MODEL=waffle_pi
$ roslaunch turtlebot3_manipulation_gazebo turtlebot3_manipulation_gazebo.launch
$ roslaunch turtlebot3_manipulation_moveit_config move_group.launch
$ roslaunch turtlebot3_manipulation_gui turtlebot3_manipulation_gui.launch
```

Note: In order to be able to simulate the manipulator with gazebo you need to active the ”play” [>] simulation button in Gazebo, in the bottom left.

4.b *Task:* Add some figures to your report with the OpenManipulator-X in the ’home pose’ and moving the ’gripper’ in Gazebo.

5 Creating a Map in Gazebo

In this task, students are requested to build a map (*.world*) in ROS Gazebo. The dimensions of the map are presented in Fig. 6. The height of walls is approximately 0.5 meters, and windows should be added to allow visibility of the robot during mission execution. Please assume the missing dimensions.

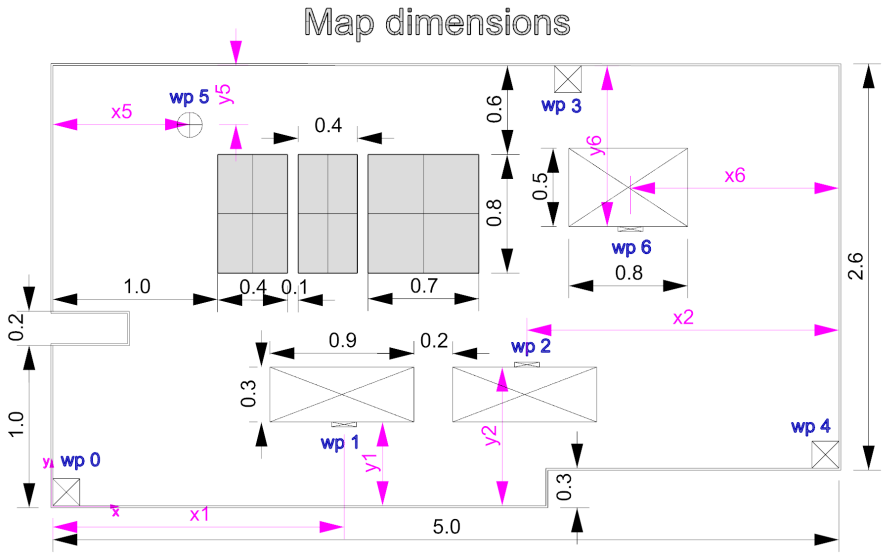

Figure 6: Map dimension to build an environment in ROS gazebo

where the location of waypoints are denoted with coordinate $W_{Pi} = [x_i, y_i]$ where, i is the number of waypoint. Note that the origin of coordinates $x - y$ is located at the bottom left of the map.
The location of waypoints are presented in Table 3. This location must be changed by each group in order to have different map for each group of student, thus add or subtract a value between $\pm 0.1m$. Use common sense to do not cross the map limits.

5.a *Task:* Present in a table the position of your waypoints for simulations (add or subtract a value between $\pm 0.1m$ to values in Table 3).

Table 3: Location of waypoints.

waypoint (wp)	x_i	y_i
0	0.2	0.2
1	1.85	0.5
2	3	0.91
3	3.25	2.6
4	4.8	0.4
5	0.87	2.4
6	3.65	1.75

5.b *Task:* Build a map in Gazebo for map in Fig. 6 and save it as "turtlebot3_ttk4192.world" at */catkin_ws/src/turtlebot3_simulations/turtlebot3_gazebo/worlds*. Note that working in gazebo constructor environment could be a bit challenging, and some bugs could appear, so you will need to do it carefully. Comment any problem that you had found and how you solve it in your report.

5.c *Task:* Create a *turtlebot3_ttk4192.launch* file to be used in the Mission planning system, save it at */catkin_ws/src/turtlebot3_simulations/turtlebot3_gazebo/launch*.
Hint: The content of the file is presented in below and also in "assignment4_ttk4192.zip" folder.

```
<launch>
  <arg name="model" default="$(env TURTLEBOT3_MODEL)"
  doc="model type [burger, waffle, waffle_pi]"/>
  <arg name="x_pos" default="2.0"/>
  <arg name="y_pos" default="2.45"/>
  <arg name="z_pos" default="0.0"/>

  <include file="$(find gazebo_ros)/launch/empty_world.launch">
    <arg name="world_name"
    value="$(find turtlebot3_gazebo)/worlds/turtlebot3_ttk4192.world"/>
    <arg name="paused" value="false"/>
    <arg name="use_sim_time" value="true"/>
    <arg name="gui" value="true"/>
    <arg name="headless" value="false"/>
    <arg name="debug" value="false"/>
  </include>

  <param name="robot_description" command="$(find xacro)/xacro --inorder
  $(find turtlebot3_description)/urdf/turtlebot3_$(arg model).urdf.xacro" />

  <node pkg="gazebo_ros" type="spawn_model" name="spawn_urdf"
  args="-urdf -model turtlebot3_$(arg model) -x $(arg x_pos)
  -y $(arg y_pos) -z $(arg z_pos) -param robot_description" />
</launch>
```

5.d *Task:* Launch the gazebo environment that you just created. Add also the robot. Open a new terminal and run the tool *rqt*, to see the topic and services actives. **Hint:** To open the map that you just created, run the following commands.

```
$ cd catkin_ws && catkin_make
$ export TURTLEBOT3_MODEL=waffle_pi
$ source devel/setup.bash
$ roslaunch turtlebot3_gazebo turtlebot3_ttk4192.launch
$ rqt
```

The resulting map, should be something like show in Fig. 9.

6 Path-finding algorithm

In this section the students are requested to develop a path-finding algorithm. This algorithm could, for instance, be A*, hybrid A*, RRT, or another algorithm of your choice.

6.a *Task:* Program a path-finding algorithm and add it to the "mission_planning_system.py" script.

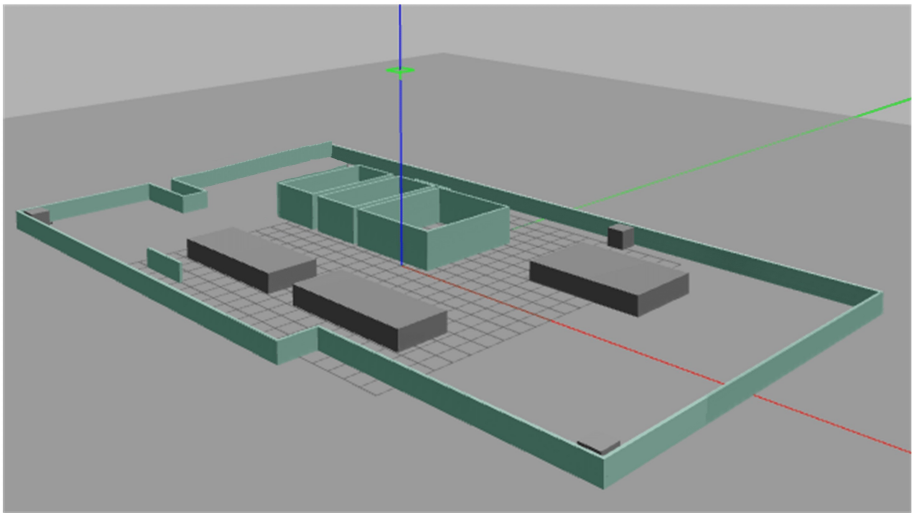

Figure 7: Turtlebot and maze simulated in Gazebo

6.b *Task:* Compute the robot trajectory between way-points 1-2; 2-3 and 3-4. Presents plots of the robot trajectory. **Hint:** Similar to what you did in assignment 1 for hybrid A* algorithm, you will need to create a map for path-finding, then change the robot positions. Assume the robot length *L=0.5*, The plots must be similar to Fig. 8.

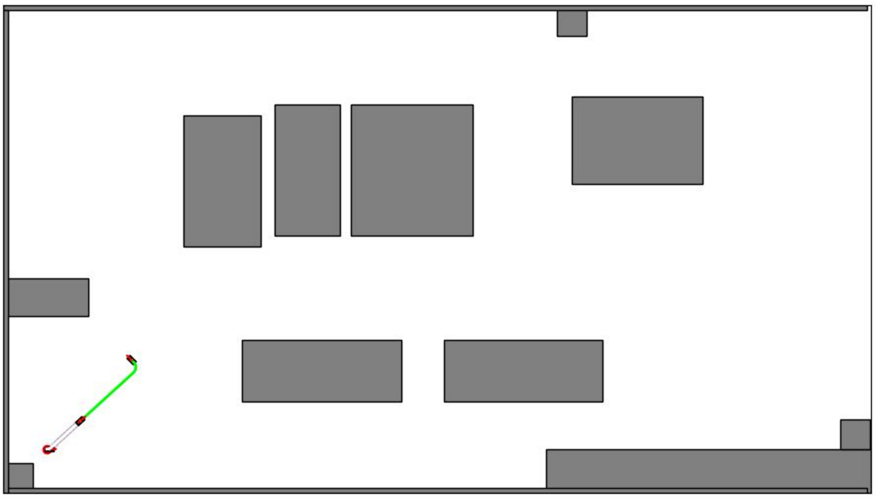

Figure 8: Pathfinding algorithm outputs

7 GNC module

In attachment of the project description, there is a script called *mission_planner_ttk4192.py* in this script at line 69, the function called ”Robot GNC module” is programmed. Follow the below instructions to install and run the script.

- Download and extract the file ”assignment4_ttk4192.zip” file in a folder.
- Copy the folder to your ROS installation, at location catkin_ws/src/.
- Install the VS code in your PC (in case you don’t have it yet).
- Install python 3.8 or higher (in case you don’t have it yet).
- Install python packages (in case you have not installed yet): matplotlib.pyplot as plt, numpy.
- Open the VS code program.
- Open the ”assignment4_ttk4192” folder with ”VS code” program (Menu - open Folder - choose: assignment4_ttk4192 - ok)
- in VS code: Open the file ”mission_planner_ttk4192.py” - Select the python 3.8 as interpreter in VS code (it is in the right-down).
- run the mission_planner_ttk4192.py script (button in the right-top)
- If the installation was corrected, the message output of the program should be ””********** TTK4192 - Assignment 4 **********””.

7.a *Task:* Once the system is installed, test the GNC module for a simple case moving the robot between two way-points i,e. Wp0-Wp1.
Hint: In a new terminal, run the below code to open the map ROS. It is important that python file "mission_planner_ttk4192.py" is defined as an executable file, to do that, run the command: "chmod +x mission_planner_ttk4192.py", or, navigate to the file and right-click in the file and **thick the option** "Allow executing file as program" at the bottom.

```
$ cd catkin_ws
$ source devel/setup.bash
$ rosrun assigment4_ttk4192 mission_planner_ttk4192.py
```

8 Programming robot actions

In this section the students are requested to program the actions of the Robot during mission execution. Actions for example to, move between waypoints, take a picture or *manipulate an object*.

8.a *Task:* Open the file named "mission_planner_ttk4192.py", and in line 426 "Turtlebot 3 actions" program the mission planning actions.
Hint: In the file, you can find some function pre-defined and other empty functions.

8.b *Task:* Program a task related to pick an object using the OpenManipulator-X attached to the turtlebot.

8.c *Task:* Using the actions that you had programmed, perform the action, "move_robot_waypoint0_waypoint1" and "take_picture" with the robot's camera.

9 Assembly AI planner and GNC

In this section the students are requested to assembly the AI planning and GNC modules together. Note that, since this is an initial development, not reactive planning is considered. We will only place the AI planning algorithm into the *mission_planner_ttk4192.py* script: *Program here your AI planner*.

Hint: You need to program a function which calls the *STP temporal planner* and execute it, then you will need to read the plan, and parse, to later send the commands to the robot, to be executed.

9.a *Task:* Insert your AI planning code into the main script *mission_planner_ttk4192.py* and make sure the inputs and outputs are consistent.

10 Mission execution

Once you had integrated all the modules of the mission planning system, Solve the mission planning problem presented in Table 4 and execute the mission in ROS Gazebo.

Table 4: Initial state and goal for mission execution.

Case	Initial state		Goal
1	(at turtlebot waypoint 2)	→	(at turtlebot waypoint 4)
	(no check valve0)	→	(check valve0)
	(no check valve1)	→	(check valve1)
	(no check pump0)	→	(check pump0)
	(no check pump1)	→	(check pump1)
	((speed turtlebot) 0.18)		-

10.a *Task:* Perform the mission execution and provide a link to drive where a video record of the full-mission is recorded.
Hint: An snapshot or a video of the mission executing in presented in Fig. 9.

Part IV
Experiments with a physical robot

In addition to the simulation results, the students will need to perform a demonstration with the turtlebot3 in a physical environment. The physical robot and environment for tests will be provided by the instructors, similar as in Fig. 10, and the students only will

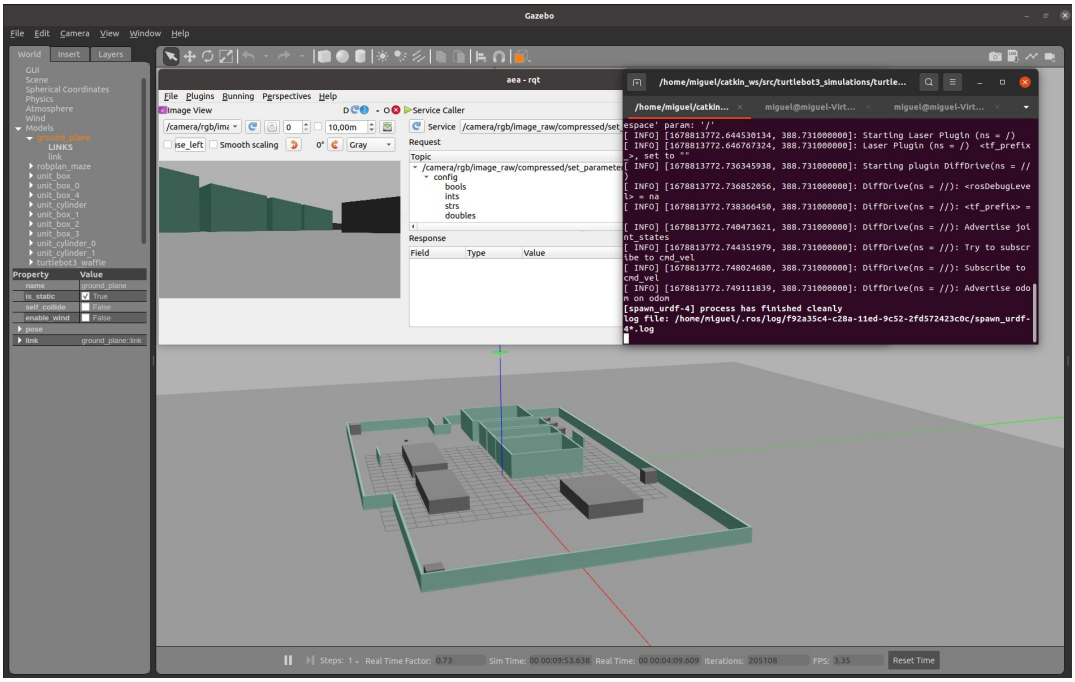

Figure 9: Snapshot of ROS environment during mission execution

need to copy their codes and run the demonstration. During the demonstration the instructors will ask some questions related to your system.
Note that: Updated information or changes will be provided closer to the date of the demonstration experiments.

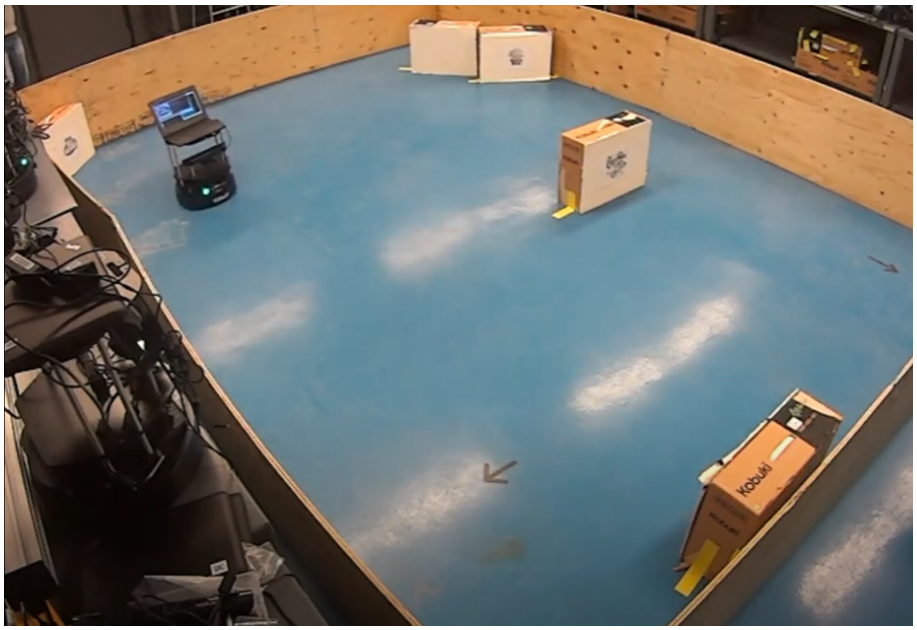

Figure 10: Physical map for experiments (reference image).

11 Set-up for experiments

11.a **Code preparation:** In order to be ready for experiments, the students must have their codes at least one day before the experiments.

11.b **Robot setup:** The UGV considered for the simulations is the Turtlebot3 model waffle_pi, Fig. 4, equipped with OpenManipulator-X.

11.c **Map for experiments:** The map will be a small scenario with static obstacles in a room located at Department of Engineering Cybernetics - NTNU, resembling the Gazebo environment shown in Fig. 9.

12 Experiments

The students will be asked to perform a simple mission, such as moving the robot to *valve-1* and taking a picture. The robot must be able to do this autonomously. Additional points will be given to students who execute a mission that includes manipulation. Finally, in real life, the world is not always static, provide any ideas for a reactive implementation.

13 Mapping the physical map

In order to perform the experiments with the turtlebot3 in the lab. First, you will need collect information about the map, for that you will need to create a ".yaml" map.

13.a **file "map.yaml":** Turn on the turtlebot and lab computer and execute the below commands, to start collection information for the ".yaml" map.
First, bring-up the turtlebot, in a terminal of the lab PC.

```
$ roscore
$ ssh ubuntu@192.168.50.239
$ password: turtlebot
$ export TURTLEBOT3_MODEL=waffle_pi
$ roslaunch turtlebot3_bringup turtlebot3_robot.launch
```

In another tab of the lab pc:

```
$ export TURTLEBOT3_MODEL=waffle_pi
$ roslaunch turtlebot3_slam turtlebot3_slam.launch
$ Use the joystic and move the robot around the Lab
$ rosrun map_server map_saver -f ~/map_ttk4192
```

Later, test the map with the Rviz command, move between waypoints "2D navigation" tool.

```
$ export TURTLEBOT3_MODEL=waffle_pi
$ roslaunch turtlebot3_navigation turtlebot3_navigation.launch map_file:=$HOME/map_ttk4192.yaml
```

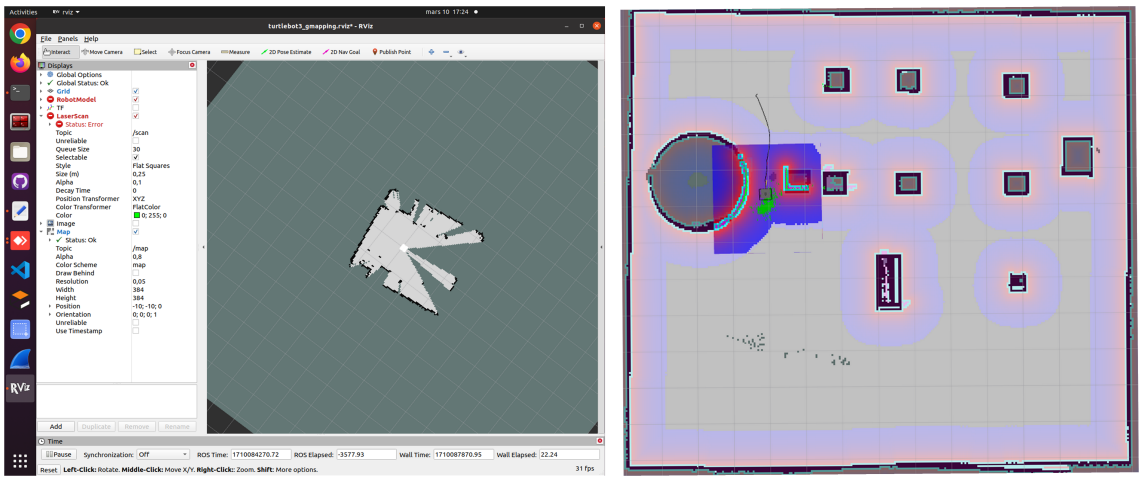

Figure 11: SLAM module turtlebot3 (reference imagen).

14 Replacing Hybrid A* with Turtlebot 3 SLAM

The physical demonstration of mission planning with turtlebot3, will use the SLAM module of the robot. Therefore, you will need to go replace the path-planning module in your code with the SLAM commands.

15 Perform the mission planning for turtlebot3

Similar as you did with the simulation in gazebo, execute your code in order to accomplish the mission. The advice is start with a simple mission such moving between way points, and from them scale.
Later, test the map with the Rviz command, move between waypoints ”2D navigation” tool.

```
$ export TURTLEBOT3_MODEL=waffle_pi
$ rosrun Assigment4_ttk4192 assigment4_ttk4192.py
```

Part V
Additional Remarks

The structure of the assignment is flexible, so students have the option of using any other approaches they consider appropriate and making the assumptions they deem necessary to solve the use case problem. Thus, you are free to choose a different high-level planning problem instead of the one proposed in section 2. For instance, you may explore techniques such as Q-learning, deep Q-networks, or any other method covered in TTK4192 - Mission Planning for Autonomous Systems [1] course.

References

[1] Lekkas A.M. *Lecture notes on Mission Planning for Autonomous Systems*. NTNU, 2024.

[2] Daniel Furelos Blanco et al. “Forward-search temporal planning with simultaneous events”. In: *13th Workshop on Constraint Satisfaction Techniques for Planning and Scheduling, Jun 24-29; Delft, the Netherlands.* 2018.

[3] Malik Ghallab, Dana Nau, and Paolo Traverso. *Automated Planning: theory and practice*. Elsevier, 2004.

[4] Miguel A Hinostroza and AM Lekkas. “Temporal mission planning for autonomous ships: Design and integration with guidance, navigation and control”. In: *Ocean Engineering* 297 (2024), p. 117104.