EXAMINATION PERIOD:	November 2025
MODULE CODE:	5CCS2ITR
TITLE OF EXAMINATION:	Introduction to Robotics – coursework 2
FORMAT OF EXAMINATION:	Online submission
SUBMISSION DEADLINE:	Thursday 18/12/2025 at 4pm

IMPLEMENT THE SPECIFICATION IN ALL SECTIONS

SUBMISSION PROCESS: Your work must be submitted as a zip file containing the full package second_coursework. Do not submit anything else.

Ensure you upload the correct file to the submission folder

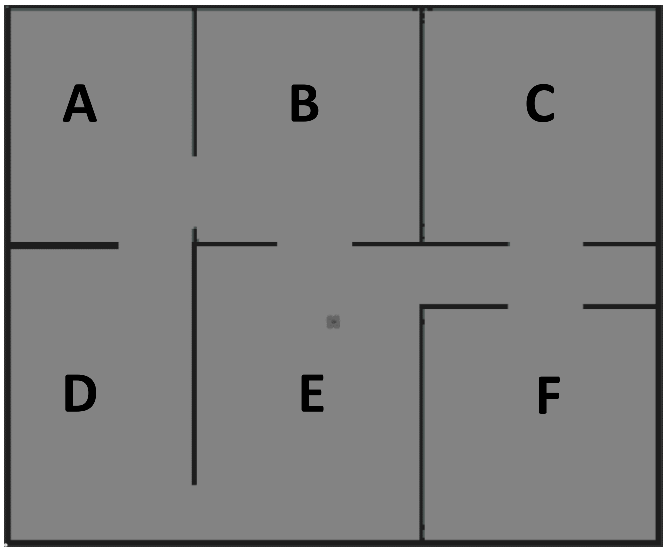
Background Story

Our robot, a TurtleBot3, is a service robot working in a house. The owner is hosting a party, and the robot must ensure the house rules are being followed.

The house rules are:

1- No people should be in the kitchen (room F in the map) at any moment.
2- No food is allowed in the bedroom (room C in the map). Food items are: pizza, sandwich, banana, broccoli.

In addition to checking the rules, the robot may also be asked at any time to find an object. When receiving the request, the robot must stop doing what it is doing and go look for the object anywhere in the house. Objects that can be requested are: pizza, sandwich, banana, broccoli, book, bottle, dog, fork, laptop, remote, toothbrush.



Important note

We will mark based on the observed behaviour of the robot. This means that if we cannot run the nodes because they do not follow the naming conventions specified below, or there are execution errors at launch, the mark will be **0** (or whatever mark has been obtained until the point of error).

Ros Fundamentals

The environment in the provided Stage simulator has the map above.

- Create a package called "second_coursework".
- Create a ROS service of type FindObject.srv (provided), called exactly /find_object. The service is used to communicate to the robot the name of the object to search for, and the service must return immediately true if the robot is not looking for an object yet, and false if the robot is already looking for an object at the time of the request. The search itself must not be implemented in the service. **(2 mark)**

- Create a launch file that runs **your nodes.** This will include all the nodes that **you have developed** and are required to run your code. Your launch file must be called student_nodes.launch, and will be run by the provided second_coursework.launch launchfile. The file second_coursework.launch will include stage, the video player, and any other dependencies you may need. Do not modify the provided launchfile. Do not include anything other than your own nodes into student_nodes.launch. **(1 mark)**

Total of Section: 3 marks

Behaviour

Implement two actions and the code that calls them when needed.

- Your robot must stay still for **10 seconds** at the beginning. This is so that we can fix the localisation before your robot starts moving. Failure to do so may result in a wrong execution.
- The first action must be called **exactly** /check_rules and be of type *CheckRules.action*. While executing this action, the robot continually goes between the kitchen and the bedroom making sure the house rules are followed. Each time the robot sees either a person in the kitchen or a food item in the bedroom it must publish on the action feedback an integer with the number of the broken rule in a field called **exactly** broken_rule (either 1 or 2).
    - Correctly using actionlib **(2 marks)**
    - Correctly encoding behaviour in a SMACH state machine **(5 marks)**
    - Correctly returning action feedback at the right time **(2 marks)**
    - Using YOLO to recognise the required objects **(5 marks)**
    - Stopping the action when a request for object is received **(2 marks)**
- The second action must be called exactly /find_object_action and be of type FindObject.action. The goal of this action is a string with the name of the object to find. When the robot finds the object it must go to the living room (room E) and **say** which room (A, B, C, D, E, F) it found the object in, and the name of the object. You **must** use one of the two TTS methods *covered in class* to make the robot speak.
    - Correctly using actionlib **(2 marks)**
    - Correctly encoding behaviour in a SMACH state machine **(5 marks)**
    - Using YOLO to recognise the required objects **(5 marks)**
    - Correctly using TTS **(2 marks)**
    - Announcing the correct room where the object is **(2 marks)**

Total of Section: 32 marks

Implementation notes

- You **must** use the YOLO version used and taught in class, using the same library/interface as done in the tutorials.

- Implement each action as a SMACH state machine with different states and correct use of userdata.
    - Structuring the code in different State classes (ideally in different files and importing them into the main node) **(1 mark)**.
    - Using SMACH states such as SimpleActionState or CBState **(2 marks)**.
    - Using SMACH userdata appropriately **(2 marks)**.

Total of Section: 5 marks

Grand Total: 40 marks

Important: Launch and semi-automatic marking

We will run **all** the submissions using the launch file provided. Therefore, you must make sure that the launch file works and runs your nodes, ideally starting from a clear workspace (with nothing else in it). If the launch file fails to run the nodes, we will deem the submission as "failed to run".

The launch file requires a parameter video_files, which can be appended at the end of the launch command as: "`video_files:=<PATH>`" (where you need to replace <PATH> with the path of the video files in your computer (see next section). We will run all the components with the same seed for reproducibility (see below).

Only code that runs in the default container is allowed, thus no external libraries or anything not used during the module is allowed.

The automatic marker will **stop** your code if either of the following conditions are met:

- The robot stays still (without moving at all, nor rotating in place) **for more than 30 seconds**.
- Your code runs for more than **15 minutes**.

Note that your robot must stay still for 10 seconds at the beginning of the execution. The first 30 seconds will not be considered for the timeout, but the robot should start moving between second 10 and second 30 of execution.

Do not modify any of the provided action files. Doing so may create errors when running your code, as it would change the API of the automatic marker. Do not modify the provided launch file either, as we will run our own (which includes everything included in the one submitted). Therefore, if you add node outside the student_nodes.launch, we will not run them.

Video player node and test videos

To facilitate the development of the coursework without access to the real robots or a webcam, we provide video/image files and a video node that will reproduce different videos when the robot is in certain locations.

The video player behaviour will be as follows:

- When the robot enters the kitchen (room F), images of the objects, people, or empty corridors will be played at a random order.
- When the robot enters the bedroom (room C), images of objects or food will be played at random order.
- When entering any other room, any object, food, or person will be played at random order.

The videos will show the relevant object for at least 5 seconds.

To run the node, you can put it in your workspace, and run it as `rosrun second_coursework itr_cw_CS_2546 --video_folder <path_to_video_folder>` (if that does not work, you can just run "`python3 itr_cw_CS_2546 --video_folder <path_to_video_folder>`"). You need to replace "path_to_video_folder" with the path to the folder where you stored the downloaded videos. For instance, if your videos are in /home/kcl/videos, you will run the node as "`rosrun second_coursework itr_cw2425CS --video_folder /home/kcl/videos`"

In order to facilitate testing, you can add an option "--seed" to make the random generator fixed, thus removing the random component (as every run will produce the same results).
Example: "`rosrun second_coursework itr_cw_CS_2546 --video_folder <path_to_video_folder> --seed` **x**", where **X** is a number of your choice (if that does not work, you can just run "`python3 itr_cw2425CS --video_folder <PATH> –seed X`").

You can also use the option "--test". This will force all rules to be broken, and will only show a laptop in room B.

Example command setting up forced test:

```
rosrun second_coursework itr_cw_CS_2546 --video_folder /home/kcl/videos  --test
```

Execution testing scripts

In order to test that your execution works in the automatic marker environment, we provide two scripts that simulate what the automatic marker does. The scripts are student_aa_run_tester.sh and student_aa_run_tester_client-node.sh. The first one creates the workspace and downloads the files that we provide (in case you modified them). The second one publishes to the topic to start the behaviour.

Some important notes:

- These scripts are **NOT** meant to be used for testing while developing, just to test what you submitted. Do not use it to compile the workspace every time!
- The test node just sends one object request, chosen at random. This won't be the same one we test the environment with. This is just to show that your code reacts to the service call.
- The fact that these scripts work in your computer does not mean it will work in the automatic marker (but chances are higher). However, if it does not work with these scripts, then it will for sure not work in the automatic marker and the full mark will be 0.

To use the student_aa_run_tester.sh script (workspace builder):

- Download the zip file you submitted on KEATS and the script below.
- Start the itr container.
- Go to the directory where your zip file is.
- Run the script as follows:

```
bash student_aa_run_tester.sh <FILE NAME>.zip <VIDEO PATH> <YOLO_COCO_DATA_PATH>
```

Where **<FILE NAME>** is the name of your file (i.e., second_coursework.zip), **<VIDEO PATH>** is the **full** path to the folder where the videos are located, and **<YOLO_COCO_DATA_PATH>** is the path in your computer where the **coco.data** file is located.