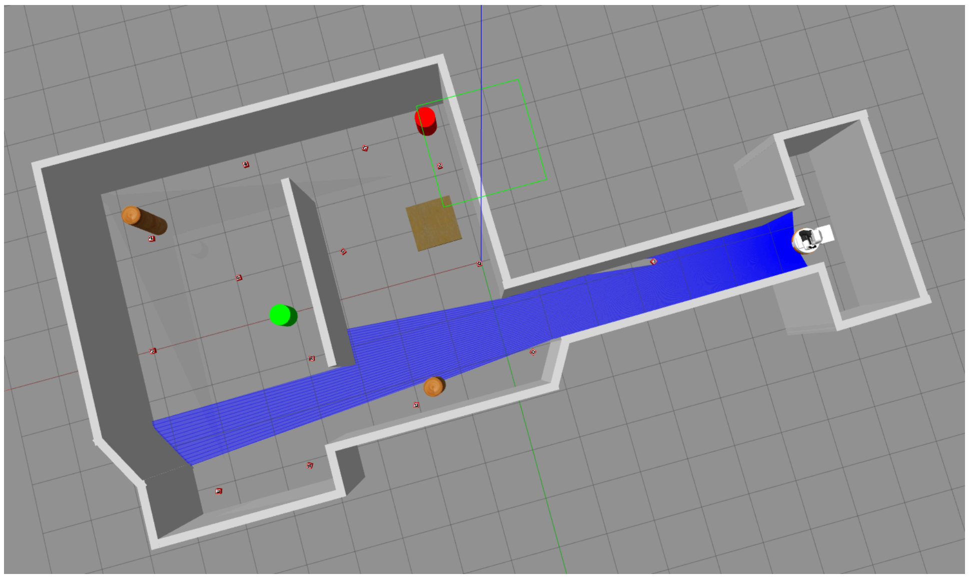
Assignment 1

DEADLINES:

- **Winter Session**: December 23rd 2024, 23.59 CET
- **Summer Session**: June 10th 2025, 23.59 CET
- **Fall Session**: August 22nd 2025, 23.59 CET

Implement a routine that lets Tiago navigate inside the environment and that lets it check over the apriltags placed on red cubes, all around the laboratory, to find the ones having some chosen IDs. Tiago needs to find the selected group of apriltags and return the position of their red cube. The IDs of the apriltags to be found are provided (randomly) by the server `ids_generator_node` on request.
The environment includes two rooms, a narrow space and a small room as initial position. Cubes with apriltags are all over the laboratory (Figure 1).
Suggestion: use the laser range finder and the camera (topic /xtion/…) mounted on the robot for scanning the environment and detect obstacles, walls and the target apriltags.

Figure 1



For the assignment, **you have to create a package** which contains the code to implement.
Your code must be implemented using the following structure:

- Node A requests the apriltags' IDs to server `ids_generator_node` through the service `/apriltags_ids_srv` and sends them to node B,
- Node B executes the navigation task and handles the research of apriltag: when apriltags are detected, you get their pose with respect to the camera

reference frame. Then, using the TF you have to transform the pose from the camera frame to the map frame.
    - Node B sends the final list of cubes' positions to node A,
    - Moreover, during the whole routine, node A implements callbacks to node B's feedback and prints the task's current status in the terminal. The feedback **has to** reflect the robot's current status, e.g., *the robot is moving, the robot is scanning …, the robot has already found Apriltags, the robot has already found Apriltags with IDs …, the detection is finished, etc.*

If you want, you can split node B into multiple nodes, each one handling a part of the workload.
You can also add more custom nodes to the architecture. Use proper names for the nodes.

Extra points

Typically, the ROS Navigation Stack is inefficient in narrow passages. Usually, the programmer has to implement an ad hoc routine to directly process the sensor data (e.g., laser and/or sonar data) and generate velocity commands for the robot, i.e. the programmer creates a **"motion control law"** for the robot. In this assignment, to get extra points, we ask you to implement your control law to navigate through the narrow corridor using the laser data. The routine has to use the laser and send the velocity commands to the topic */cmd_vel* to move the robot without calling the *move_base stack*.

N.B.: Please, specify in the submission report that you implemented the extra point part of the assignment.

Instruction to start the homework

Local ROS Installation

Who has a local ROS installation (e.g. Ubuntu or Virtual Machine) is required to install the Tiago packages. Please, follow these instructions.

Install the Tiago simulation workspace:

```
>  sudo apt-get update
>  sudo apt-get install git wget ipython3 python3-rosinstall
   ros-noetic-desktop-full python3-catkin-tools python3-rosdep
   python-is-python3 ros-noetic-actionlib-tools
   ros-noetic-moveit-commander
>  sudo apt install -y ros-noetic-four-wheel-steering-msgs
   ros-noetic-urdf-geometry-parser ros-noetic-ddynamic-reconfigure
   ros-noetic-people-msgs ros-noetic-twist-mux ros-noetic-map-server
   ros-noetic-amcl ros-noetic-robot-pose-ekf
   ros-noetic-moveit-planners-ompl
   ros-noetic-moveit-simple-controller-manager ros-noetic-global-planner
   python3-future ros-noetic-teb-local-planner ros-noetic-hokuyo3d
   ros-noetic-urg-node ros-noetic-sick-scan
>  mkdir ~/tiago_public_ws && cd ~/tiago_public_ws
>  wget
   https://raw.githubusercontent.com/pal-robotics/tiago_tutorials/noetic-
   devel/tiago_public-noetic.rosinstall
>  source /opt/ros/noetic/setup.bash
>  rosinstall src /opt/ros/noetic tiago_public-noetic.rosinstall
>  catkin build
```

N.B. At first, you may need to compile the workspace 3-4 times before succeeding.

```
>  source src/setup.bash
```

N.B. If get some errors, try to run `pip install pyyaml==5.4.1`

```
>  rosdep install -y --from-paths src --ignore-src --rosdistro noetic
   --skip-keys "urdf_test omni_drive_controller orocos_kdl pal_filters
   libgazebo9-dev pal_usb_utils speed_limit_node camera_calibration_files
   pal_moveit_plugins pal_startup_msgs pal_local_joint_control
   pal_pcl_points_throttle_and_filter current_limit_controller
   hokuyo_node dynamixel_cpp pal_moveit_capabilities pal_pcl
   dynamic_footprint gravity_compensation_controller pal-orbbec-openni2
   pal_loc_measure pal_map_manager ydlidar_ros_driver"
>  cd ..
```

Create a workspace to develop your packages

```
>  mkdir catkin_ws  && cd catkin_ws
>  source ~/tiago_public_ws/devel/setup.bash
>  mkdir src && cd src && catkin_init_workspace
>  cd .. && catkin build
```

Now you are ready to install the environment for the assignment inside `catkin_ws`

- Clone this repository into the src folder of your workspace. The package contains files necessary for correctly running the whole project.
  ```
  > cd ~/catkin_ws/src
  > git clone https://github.com/AnnaPlt/tiago_iaslab_simulation.git
  ```

- To remap to a new repository it can be helpful remove the old git blob:
  ```
  > rm -rf tiago_iaslab_simulation/.git
  ```

- clone the following repositories in your workspace (src folder).
  - Apriltag: https://github.com/AprilRobotics/apriltag.git
  - Apriltag_ros: https://github.com/AprilRobotics/apriltag_ros.git
  - Gazebo_ros_link_attacher: https://github.com/pal-robotics/gazebo_ros_link_attacher.git
  ```
  > cd ~/catkin_ws/src
  > git clone …
  ```

- Build the packages:
  ```
  > cd ~/catkin_ws
  > catkin build
  > source ~/catkin_ws/devel/setup.bash
  ```

- Start the simulation (cmd console 1):
  ```
  > roslaunch tiago_iaslab_simulation start_simulation.launch
  world_name:=iaslab_assignment1
  ```

- Navigation stack (cmd console 2):
  ```
  > roslaunch tiago_iaslab_simulation navigation.launch
  ```

- AprilTag (cmd console 3):
  ```
  > roslaunch tiago_iaslab_simulation apriltag.launch
  ```

- Apriltags IDs server (cmd console 4):
  ```
  > rosrun tiago_iaslab_simulation apriltag_ids_generator_node
  ```

You **must create your own package** into your group repository, develop your code in it and deliver this one as a whole. Do not submit the packages above, do not edit any file in the packages above.

N.B. depending on your GPU setup, you may face some problems in the laser simulation. We tried some workarounds, but they are not always effective. Anyway, you can use VLAB to correctly simulate laser sensors.

VLAB

In VLAB, the tiago environment is already installed inside the container.

- As usual, please follow the instructions to start the container:
  ```
  > start_tiago
  > source /opt/ros/noetic/setup.bash
  > source /tiago_public_ws/devel/setup.bash
  ```

- Create a new workspace (you can also use one already created):
  ```
  > mkdir -p ~/catkin_ws/src
  > cd ~/catkin_ws
  > catkin build
  > source ~/catkin_ws/devel/setup.bash
  ```

- Download the package with the simulated environment into the workspace:
  ```
  > cd ~/catkin_ws/src
  > git clone https://github.com/AnnaPlt/tiago_iaslab_simulation.git
  ```

  You are not supposed to edit this package. Just download and use it.

- To remap to a new repository it can be helpful remove the old git blob:

```
> rm -rf tiago_iaslab_simulation/.git
```

- Clone the following repository: https://github.com/pal-robotics/gazebo_ros_link_attacher.git

```
> cd ~/catkin_ws/src
> git clone
https://github.com/pal-robotics/gazebo_ros_link_attacher.git
```

- Build the package:

-
  ```
  > cd ~/catkin_ws
  > catkin build
  > source ~/catkin_ws/devel/setup.bash
  ```

- Start the simulation (cmd console 1) and wait until the start up is completed:

```
> roslaunch tiago_iaslab_simulation start_simulation.launch
world_name:=iaslab_assignment1
```

- Navigation stack (cmd console 2):

```
> roslaunch tiago_iaslab_simulation navigation.launch
```

- AprilTag (cmd console 3):

```
> roslaunch tiago_iaslab_simulation apriltag.launch
```

- Apriltags IDs server cmd console 4):

```
> rosrun tiago_iaslab_simulation apriltag_ids_generator_node
```

N.B. Sometimes the simulation does not properly initialize. If you experience some issues, try to kill and restart.

How to submit your solution

To submit your solution, you **must** do the following:

1. Setup your group repository

a. Create your **private** group repository on Bitbucket named as "ir2425_group_XX", where XX must be substituted with your group number:

```
>   cd ~/catkin_ws/src/ir2425_group_XX
>   git init
>   touch README.md
>   git add README.md
>   git commit -m "first commit"
>   git push
```

If you are using local installation, please be sure to upload in the repository only the packages you are required to submit, for example calling:

```
> echo -e 'tiago_iaslab_simulation/' >> .gitignore
```

b. Includes all the group members and the 3 tutors in the collaborators of the repository. Here our BitBucket profiles:
   - anna.polato.2@phd.unipd.it
   - alberto.bacchin.1@phd.unipd.it
   - albertogottardi7@gmail.com

c. Public repositories or repositories we cannot access **will not be considered for correction**.

2. Start coding

d. **Push your code (with good comments)** into your group's git repository. Code explainability will be part of the evaluation criteria.

e. We encourage you to commit the code as you write it and not in one block

f. **Add a README.md file** which contains the necessary commands to correctly execute and test your code (e.g. *roslaunch* and *rosrun* commands). Make sure the instructions are working properly. **The first lines of the README.md must contains the following informations**:

```
GROUP XX
Member1's name, email
Member2's name, email
Member3's name, email (if any)
```

g. We suggest you to test your code step by step in a new clear workspace, cloning the code from your repository in order to be sure that your code is executable by third parties.

h. **Make sure your code is compiling before submitting**. Not compiling codes will be penalized in the final grade.

3. Write a Report

Alongside code, the report is part of the examination. The report is a written document in PDF format that explains your solutions. It must be **max 3 pages** (longer reports will be penalized in scoring).

The first lines of the report must be like:

```
GROUP XX
Member1's name, email
Member2's name, email
Member3's name, email (if any)
LINK TO THE REPOSITORY
LINK TO ADDITIONAL MATERIAL
```

The report is expected to provide a high-level overview of the solutions you implemented to solve the home. In particular, we expect to find:

- **Organization of the package**: which nodes you implemented and their main functions (do not just repeat our instructions, go a bit in depth explaining what you have implemented); introduction to custom messages/services/actions you implemented (if any); etc…
- **The algorithm and strategies you adopted**: high level description of the algorithms and solution you implemented (it could be in written form, schemas, ecc…). For example: "We implemented a banana detection algorithm using the RGB-D camera images. We converted RGB images into HSV representations and we applied a threshold on the H channel around 60±10 to segment only yellow pixels. We then compute the bounding box enclosing the yellow pixels. The four corners of the bounding boxes are then mapped into the robot space using the camera matrix and the depth map of the scene etc...."
- **Logics and problem solving**: try to organize the report not as a "shopping list". Try to explain the logic behind your design choices. For example, it is a good idea to start from a problem you encountered (e.g. "to solve the problem of banana detection, we decided to use RGB-D camera images…"). Explain and justify your choices (e.g. "We decided to pass to HSV space because it is straightforward to isolate a specific hue").

You can use images, schemas and diagrams in your report.
If you use external materials, put the citation to the sources.

4. Submission

- **You are allowed to push the code in the repo until the submission on Moodle**. Later commits will not be considered for the evaluation.
- **Prepare a brief PDF report (see the section "Write a Report")** that explains your solution. The report is supposed to explain the high-level ideas, strategies or algorithms you implemented to solve the assignment. **The first lines of the report must be like**:

  ```
  GROUP XX
  Member1's name, email
  Member2's name, email
  Member3's name, email (if any)
  LINK TO THE REPOSITORY
  LINK TO ADDITIONAL MATERIAL
  ```
- **Prepare a short video** (e.g., screen recording) that shows the execution of your software by the robot. This is necessary in case we cannot easily run your code.
- **Report and video must not be placed in the repository**. We suggest you create a Google Drive folder, upload the video or any additional material and share with us the link to the folder inside the report.

We will open a submission area in Moodle. You will be able to attach the report to your submission (be sure that it contains all the needed information such as a link to the repo, link to the video, additional info, etc.). **The name of the submission must be as follows: GroupXX_A1**. Please submit only once per group, one member can submit for everybody.

5. Evaluation Criteria

- **Report Quality**: For detailed guidelines, refer to Section 3, *"Write a Report*." Additionally, we will assess the quality and effectiveness of your solutions. For example, flexible and generalizable approaches will be rewarded over hard-coded solutions, and efficient and optimized solutions will be rewarded over brute-force methods.
- **Code Quality**:
  - The code is well-organized (correct usage of classes, functions, etc). Note that an over-usage of classes and functions is not good.
  - The code should be documented with comments and linked to what is written in the report (explainability).
  - The code makes a nice usage of ROS tools like messages, services, actions, launch files, etc…
  - Usage of proper names for variables, functions, nodes, etc…
  - Code compiles without errors.
- **Video Quality**: video should be explicative of the task and the solution you implemented. You should not show the robot executing the task only, but also any additional information that may be useful (e.g., terminals, rviz, etc…) to understand what is going on. Try to valorize your own solutions in the video. Video should be recorded at a proper resolution.

Assignments are evaluated with a score **up to 30 points**. On the top of this, bonus (additional points) or malus (penalization points) may be applied:

- **Bonus** may be applied when the work implements particularly smart solutions.
- **Bonus** (up to 3 points) may be applied if you correctly implement the "Extra points" problem.
- **Malus** (-3 points) is applied for who delivers the homework in Summer or Fall sessions.
- **Malus** may be applied when the work does not follow the guidelines (e.g., too long reports, not compiling code, wrong instructions to start up the project, etc.…) or implements too naive solutions.