Basic ROS2 Tasks

The following tasks are 1) introduction to working with Linux (Ubuntu) and ROS2; 2) creating ros2 nodes that integrate hardware and make data available to other parts of the system via topics. Please, do not hesitate to ask early if something is unclear about the tasks.

1) **Install System, ROS and Basic Tutorials**
   a) **Install natively Ubuntu 20 or 22** (dual boot or solo linux, not virtual machine – for your own good)– IF you have such an Ubuntu20 or 22 computer set up already, you can skip this step -> make sure your system is up to date (sudo apt update and sudo apt upgrade).
   b) get used to the basic linux shell commands: ps, ls, cd, env | grep <word>,....
   c) check your .bashrc (add the following alias: alias ..='cd ..' and test it)
   d) **Install ROS2** – note *Ubuntu20 needs ROS2 foxy* and *Ubuntu22 needs ROS2 humble* (they are all very similar and it does not matter which duo you use). To install ROS2 follow this guide (this is the foxy guide, switch to humble if you have ubuntu 22, deb installation is fine): https://docs.ros.org/en/foxy/Installation/Ubuntu-Install-Debians.html
   e) check if your ROS2 installation is fine by opening two terminals (1) ros2 run demo_nodes_py talker; (2) ros2 run demo_nodes_py listener
   f) get acquainted with the basic ROS2 commands: type ros2 and see all commands. Most important: ros2 topic (list,echo), ros2 run <modulename> <programname>
   g) Do the basic ros2 tutorials (note sometimes the same task is done in py and cpp, chose one, you do not need to do both)
      i) CLI TOOLS ("what is what") https://docs.ros.org/en/foxy/Tutorials/Beginner-CLI-Tools.html
      ii) Client LIBRARY – build and compile nodes (modules) https://docs.ros.org/en/foxy/Tutorials/Beginner-Client-Libraries.html
2) **Your own Talker/Listener:**
   a) Create a new ros2 workspace and a new ros2 package
   b) Write a simple program (cpp, py,...) that reads data from your laptop hardware, at a rate R1. Easy: temperature or cpu load;
   c) Write a ros node (ros publisher) around that program to publish this data at rate R2 on a new topic called /myHWtopic (the publisher node may expose this topic)
   d) Depending on R1 and R2, you might have too much, exactly right or too little data. Think about these cases and how to solve them (you will only have to solve your case, e.g., R1 > R2 (too much data)).
   e) Write another ros node (ros subscriber) that listens to everything that is posted on /myHWtopic, reads it and prints it to stdout (prints it to shell).
   f) finish your ros package and make it available system wide, so we can use ros2 run <package> <program>
   g) run your publisher and subscriber and see that it works (two terminals)
      i) (1) publisher node that sends the HW data your program retrieved and publishes it on /myHWtopic
      ii) (2) subscriber node that reads /myHWtopic and prints the data (stdout)

Submission: Source code, screen recording for 2) (zoom)

Advanced Tasks for AI Robotics:
Choose one task among the following options, python will suffice (you are free to use cpp):

3. LLM-Integrated Robotic Decision-Making
Build a ros2 node that receives an image from a rbg cam or an image from a folder (if you do not have a webcam use your phone and do it offline) as well as user input (text based – stdin). It queries the LLM (gpt4) and uses the response to decide a series of robot actions (which it publishes as a plan on a topic called /robotPlan). So for example: Prompt: Check this image if there is a door and then, if open, move to it. With the sought response: if there is no door or it is closed: “cannot do it” and if there is a door: “can do it” + what actions are required (action options: turn left, turn right, move forward). I.e. if the door is on the left of the image we want to turn left and then move forward (plan: turn left, move forward, stop close to door). You are free to improve my simple example with different objectives (or objects) , more actions or generally a better task – and you of course need to improve the prompt. You can work with .json for handling data or with anything else. This task requires linux and a camera (webcam). You might need some free API key from a random VLM.
This task checks ros2, API handling, multi-modal data and prompt engineering, parsing, mock integration of robot actions and real integration of Hardware

4. CV – Object Detection + Robot Control / Action
Build a ros2 node that receives a live image from a rgb cam. Use CV2 or Yolo to create a simple pipeline that can detect a human and calculate a bounding box and center point. Then based on the center points offset recommend an action to a robot. At least the robot should be able to turn left and right. And, you should somehow tell it by how much to turn. You can do so offline (i.e. math/geometry) or online via minimizing the error (detection_centerpoint vs. image center). The detection and recommended action are to be published on a ros topic (/robotAction). Bonus: Make your program recognize you and make decisions based on that. I.e. if it knows a person it turns towards the person and if not it turns away. This bonus will require to integrate a local network – there are several that can run (slowly and badly) on CPU. If you do not have a webcam or a way to get a live image this task will be hard to do – you can in principle create a series of pictures in which you are at different positions in the frame and get recommended actions. This task requires linux and a camera (webcam). – yolo might require a GPU to run locally.
This task checks ros2, practical openCV / vision pipelines, mock integration of robot actions and real integration of Hardware

5. Text Based Automated RL - LLM Loop
Make the LLM of your choice create a basic text based simulation environment (in .py) for RL. I recommend using a small grid world (a description follows) - but you are free to also use block stacking or similar. Make the LLM implement an RL algorithm. This can be tabular value iteration, also a small network is fine – but what's important here is rapid convergence (so a small simple problem) and that it was produced by the LLM (and it is able to extend/tune it). This first step can be done manually via prompting.
Then in the second step, create a pipeline around an API call to that LLM. Create a prompt that feeds it information about the environment, the training results of the agent and that requires a response that tunes that environment and the learning algorithm (add skills to the agent, add/remove/re-weight rewards, reward tuning, tuning hyper-parameters, changing the agent type, changing the algorithm, improving the algorithm). Note, that the second step's goal is not to produce a functioning improving agent, but to prove that you can create such a pipeline (actually making this work may take considerable time (or you get lucky – thus it is NOT expected). So, the goal is for the LLM to automatically change the RL environment and your pipeline to restart training (switching between the LLM tuning and RL training).

Recommended Environment description:
Simple planar grid world (e.g. 6x6 to 10x10 tiles). A randomized start state and one randomized goal state, several randomized trap states. An initial agent that has four move actions (up down left right). If it reaches the goal state it gets +2pts, if it falls in a trap on its way it gets -1pt but can continue. Per move action it gets -0.05pts.
A nice idea for the LLM loop would be to realize that the agent may need a sense action – which upon activation costs -0.1pts and prevents it from moving into a trap state (i.e., the agent may learn if and when to sense). Note, that tuning the exact costs and rewards can be left to the LLM (I just give an example). And again, the goal is not necessarily to produce a learning agent, but a loop that changes the setup via LLM and restarts. Note that you need to decide how to handle data from the RL process s.t. the LLM can make decisions on an informed prompt.

This task is ROS independent. It checks LLM usage, prompting, API calls and software engineering skills – e.g. creating pipelines and handling data. On some level it checks a basic understanding of RL.