Sorbonne Université – Département de Sciences De l'Ingénieur
ROS and experimental robotics
February 2023, 1h30.

You are working in a Robotics company currently developing a new low-cost drone called `mydrone`, endowed with 4 propellers, one camera, one temperature sensor, and one pressure sensor. The drone is controlled from a remote which is only equipped with joysticks and leds indicating the current status of different elements on the drone, like the battery charging level, or warnings about e.g. too far distance between the drone and the pilot.

You are in charge of the altitude estimation of the drone, which must be computed from the data measured onboard during the flight. Luckily, the altitude relates directly to the air pressure and current temperature through the relation

$$P = P_0\, e^{-\frac{\mu g h}{RT}}, \tag{1}$$

where $P_0 = 101325$ Pa is the standard pressure at sea level, $\mu = 0.0289644$ kg/mol is the mean molecular mass of air, $g = 9.80665$ m/s^2 is the gravity, $R = 8.31432$ J/K.mol is the perfect gas constant, P the pressure in Pa, T the temperature[1] in K, and h the altitude in m.

Your colleagues have already developed a ROS node `mydrone` running on the drone, and communicating directly with the hardware on the system, and publishing the sensors values on dedicated topics. For now, the engineers are able to publish (i) the temperature measured by the corresponding sensor in the `/temperature` topic (in $°C$), and the air pressure around the drone in the `/pressure` topic (in Pa).

In addition, the ROS node `mydrone` is also in charge of triggering some LEDs on the remote controller used by the pilot. Your colleagues have so far only implemented the triggering of the `ALT WARN` indicator, which is illuminated in green if the drone flights below 1300m of altitude, or red if ot flights above. This altitude threshold has been determined thanks to the range of the remote controller: if the drone is too high, the drone might not be able to respond to the pilot controls anymore. The change in color of the LED is performed by calling the service `/altitude_warning`, which uses as an input the text string "NORMAL" if the drone altitude is below 1300m, or "WARNING" if it is above.

Your objective is to write a ROS node called `exam`, which must publish an estimation of the drone altitude on a new `/altitude` topic (in m), and trigger if needed the `ALT WARN` indicator on the remote by calling the appropriate service.

Evaluation

At the end of the exam, you have to upload two files to Moodle:

- A PDF report, written with `openoffice` for example (in french or english), where you answer to the questions below and where you can copy and past the outputs of the different commands you use to answer the questions. For example, for the question "6. List all the running topics", you must put in your report :
  - the exact command line you use to answer the question,
  - and the corresponding output you get.

  Obviously, you have to comment all your responses. A simple copy and past without a sentence explaining the outputs is **not** considered as a full response to the questions.
- A `zip` file of your `exam` package that will be used to access your code, but also to actually launch it.

Preparation

1. To begin with, download the file `mydrone.zip` from Moodle. Extract it in the `src` folder of your catkin workspace. Launch `catkin_make` to build your ROS workspace and source you `.bashrc`.
2. Go to the `src` folder of the package `mybot` you just downloaded, and make executable all the python scripts in there with the shell command `chown +x python_file.py`.

[1]Remember that $T[K] = 273.15 + T[°C]$

3. Quickly test that all is OK by running the node `mydrone_node`. Contact your teacher if it is not the case.

1 Exploration of ROS and of the nodes

4. Run the node `mydrone_node` from the package `mydrone`. Then list all the running nodes.

Solution: 1 point. La moitié si pas de commentaire.

```
> rosnode list
/mydrone
/rosout
```

2 nœuds tournent : le nœud attendu et `rosout`, lancé avec le master via la commande `roscore`.

5. List the information about the `mydrone_node` node. Explain each line of the output you get.

Solution: 2 points. 1pt si des lignes ne sont pas expliquées.

```
> rosnode info /mybot_usarray
--------------------------------------------------------------------------------
Node [/mydrone]
Publications:
 * /pressure [std_msgs/Float32]
 * /rosout [rosgraph_msgs/Log]
 * /temperature [std_msgs/Float32]

Subscriptions: None

Services:
 * /altitude_warning
 * /mydrone/get_loggers
 * /mydrone/set_logger_level

contacting node http://127.0.0.1:62496/ ...
Pid: 13996
Connections:
 * topic: /rosout
    * to: /rosout
    * direction: outbound (62498 - 127.0.0.1:62501) [11]
    * transport: TCPROS
```

Je ne vais pas tout lister (mais les étudiants doivent le faire !). La dernière partie n'est souvent pas comprise :

- `contacting node http://127.0.0.1:56353/ ...`: adresse et port où tourne le nœud
- `Pid: 12097`: Process Identification du numéro du processus
- `* direction: outbound (56355 - 127.0.0.1:56358) [10]`: type de connexion, et vers quelle addresse/port (ici, celui du master)
- `* transport: TCPROS`: protocole utilisé par ROS, ici une connection TCPROS.

6. List all the available topics. Explain the role of the `/rosout` topic.

Solution: 1 point. Aucun si /rosout n'est pas expliqué (ou explication fausse).

```
> rostopic list
```

```
/pressure
/rosout
/rosout_agg
/temperature
```

`/rosout` is le topic sur lequel les nœuds peuvent publier des messages de log de manière centralisée.

7. What kind of message is available on the `/temperature` topic?

Solution: 1 point. La moitié si pas de commentaire.

```
> rosmsg info std_msgs/Float32
float32 data
```

Valeur `data` qui stocke la valeur du message en float.

8. Display a few messages available on the topic `/temperature`. At which rate are they published?

Solution: 1 point. 0.5 si commande bonne mais rate faux.

```
> rostopic echo /temperature
data: 2.8841073513031006
---
data: 2.48634672164917
---
data: 2.0330379009246826
---
> rostopic hz /temperature
subscribed to [/temperature]
average rate: 0.777
min: 1.286s max: 1.286s std dev: 0.00000s window: 2
average rate: 0.777
min: 1.286s max: 1.288s std dev: 0.00066s window: 3
```

Publication à 0.77Hz environ.

2 The exam node

You have to write a new ROS node `exam` which must compute and publish the altitude of the drone.

9. Create a package `exam` in your `catkin` workspace, with dependencies to `rospy`, `std_msgs` and `mydrone` ;

Solution: 1 point. 0 si rien n'est expliqué.
```
catkin_create_pkg exam rospy std_msgs mydrone
```

10. Create the node `exam_node` inside the package `exam` which must subscribe to the two topics `/temperature` and `/pressure`. To begin with, you might only display the temperature and pressure values received on these topics.

Solution: 2 points. 1 point si seulement un topic. Voir la proposition de solution `exam_node.py`. Ici, il faut en fait que les étudiants aient défini 2 subscriber, et les 2 callback qui vont avec, qui peuvent à ce stade ne faire qu'un print du message reçu.

Evaluation

It is important you comment **extensively** your code, even for elementary lines of code. Your node will be assessed by running it (it must obviously run as expected and provide the functionalities listed above), but the quality of the code and its comments will also be taken into account.

11. On the basis on the temperature and pressure values obtained before, compute the altitude in m thanks to Equation (1). The result must then be published on a new topic `/altitude` **with a rate of 10 Hz**.

    Solution: 4 points, on vérifie surtout l'agorithmie et la logique du code. 1.5 points si le publisher est correctement déclaré. Du coup, ici il faut que les étudiants définissent un Publisher, avec une boucle de publication qui doit tourner à 10Hz à l'aide de `rospy.Rate` et `rospy.sleep`. La difficulté/piège ici est que les 2 topics de température et pression ne publient pas à la même vitesse : il faut donc que les étudiants trouvent un moyen pour stocker la solution dans une variable utilisée pour calculer l'altitude avant la publication sur le topic. Dans le code `exam_node.py` j'utilise 2 variables globales pour faire ça, et ce sont ces variables qui sont utilisées dans la boucle de publication. D'autres solutions doivent être possibles, en particulier en utilisant un objet.

The following questions are mostly independant.

12. Write a launchfile in the `exam` package launching both `mydrone_node` and `exam_node` nodes.

    Solution: 1 point. La moitié si pas de commentaires. Rien de spécial ici :

```
<launch>

<!-- Run one mydrone nodes -->
<node pkg="mydrone" name="mydrone" type="mydrone_node.py"/>

<!-- Run one exam node -->
<node pkg="exam" name="exam" type="exam_node.py" output="screen" required="true">

</node>

</launch>
```

13. On the basis on the altitude computed before, trigger the call to the `/altitude_warning` service to change the color of the `ALT WARN` indicator to red or green.

    Solution: 4 points. 2 points si appel au service en boucle. Rien de spécial ici, il faut appeler le service uniquement si on passe dessus ou dessous l'altitude seuil. ATTENTION : il faut appeler le service SEULEMENT à la frontière, et ne pas appeler non-stop le service !

14. The company plans to propose multiple model of the remote controller in the future, so that the current altitude limit of 1300m could be extended.

    (a) To propose a more generic way to deal with these various models of remote, add a `threshold` parameter in the launchfile, and use it in your code to make the `ALT WARN` indicator turn red depending on this parameter value.

    Solution: 1 points. 0 si la variable n'est pas initialisée. Rien de spécial, il faut ajouter dans le launchfile :

```
<!-- Parameters -->
<param name="threshold" value="1300" />
```

    Et ajouter le `getparam` dans le code. Attention, la variable doit être initialisée !

(b) In practice, one would the altitude warning to be triggered only if the drone stays above the `threshold` **more than 5s**. Modify your code to implement such a delay. The `ALT WARN` indicator must still go back to green as soon as the drone flights above the `threshold`.

Solution: **3 points.** Euh ... j'ai eu la flemme de coder ça, mais il suffit de mettre un compteur dans la boucle de publication qui compte combien de fois on a été au dessus de l'altitude seuil. Connaissant la vitesse de la boucle (10ms), il faut donc avoir été 500 fois au dessus pour appeler le service. Bien sûr, le compteur doit être réinitialisé dès qu'on repasse sous le seuil.

Solution: Il reste à évaluer la qualité du code :

- est-ce que le neud se lance et fait ce qu'il doit faire, selon l'avancée de l'étudiant ? **2 point**
- est-ce que le code a été commenté partout et les commentaires sont clairs ? **4 points. La moitié si assez peu de commentaires et même 0 s'il n'y en a clairement pas assez.**
- qualité du code : algorithmiquement, est-ce "bien" codé, en respectant les bonnes pratiques ? **2 points.**