Instructions for workshop tasks

Task 1 - Getting started with the robot
- Task 1.1 - Connecting to the robot
- Task 1.2 - Blinking with the robot
- Task 1.3 - Get position of the robot
- Task 1.4 - First move
- Task 1.5 - Move the robot between two positions
- Task 1.6 - Draw a square in the air with the robot
- Task 1.7 - Getting started with the tools

Task 2 - Getting started with the vision Set
- Task 2.1 - Workspace preparation
- Task 2.2 - Get image from the robot
- Task 2.3 - Detect object on the workspace
- Task 2.4 - Show image stream from the robot
- Task 2.5 - Implement own detection of the object (advanced)

Task 3 - Pick and Place
- Task 3.1 - Manual Pick and Place
- Task 3.2 - Vision based Pick and Place
- Task 3.3 - Vision based Pick and Place with own detection algorithm

Task 4 - Conveyor belt
- Task 4.1 - Control conveyor belt
- Task 4.2 - Using infrared sensor
- Task 4.3 - Pick and Place from workspace to Conveyor belt
- Task 4.4 - Pick and Place from Conveyor belt workspace to the numbered area

Task 5 - LLM
- Task 5.1 - Basic use of OpenAi
- Task 5.2 - Basic use of Ollama
- Task 5.3 - Prepare for LLM Pick and Place

Task 6 - Pick and Place using LLM
- Task 6.1 - Pick and Place using LLM
- Task 6.2 - Pick and Place using LLM advanced

Task 7 - Be Creative

Task 1 - Getting started with the robot

This task is designed to provide you with the necessary knowledge to move the robot and control its parts, such as the LED ring. To work with the robot using the API, you need to import the members of the pyniryo library. This can be done using the following command:

```
from pyniryo import *
```

This is the easiest way, but of course you can import each member individually, for example:

```
from pyniryo import NiryoRobot
```

Now you can dive into individual tasks.

Task 1.1 - Connecting to the robot

The goal of this task is simple. Connect to the robot using pyniryo API. It consist of three parts. First, create a robot object from NiryoRobot class. As a parameter, you must specify the robot's IP address as a string – i.e. `‘169.254.200.200’`.

```
robot = NiryoRobot('169.254.200.200')
```

In the second part, you have to call robot member method `calibrate_auto()`. The robot will calibrate itself – be careful, robot will move during this phase.

The third part is closing the connection. Use member method of the robot object called `close_connection()`[1].

Task 1.2 - Blinking with the robot

Great, you are able to communicate with the robot. Now try controlling the robot's LED ring. You can use the member method `led_ring_flashing()`[2] for this. You can also use another method from the API, such as `led_ring_breath()` or `led_ring_snake()`. This can be useful in case you want to indicate state of the robot.

Task 1.3 - Get position of the robot

That was easy, wasn't it? Before you try to move the robot, it's good to know how to find its position so that you can manually adjust its position, for example, to observe the workspace.

[1] use dot notation robot.close_connection()
[2] https://niryorobotics.github.io/pyniryo/v1.2.1-1/api/api.html#pyniryo.api.tcp_client.NiryoRobot.led_ring_flashing

The robot object has a member method called `get_pose()`. It returns `PoseObject` that can be used as a parameter when moving the robot.

Now create a script that prints current position of the robot. It can print position in a loop (e.g. every second) or it can react on user input (e.g. hit of the Enter on a keyboard). If you are using loop, the good way to end script is to handle `KeyboardInterrupt` event.

```
try:
    while True:
        # your code
except KeyboardInterrupt:
    # KeyboardInterrupt exception happened. . e.g. close
connection
```

Using this script, you can get the position of the robot during manual (free move) mode.

Task 1.4 - First move

Startup the robot and use script to calibrate it. Then wait 60 second (member method `wait()`[3]). During this minute change the position the robot using `freemove` mode. After 60 seconds, call `move_to_home_pose()`.

Task 1.5 - Move the robot between two positions

The robot has moved to its home position. Its great. Thats great, but also boring. We want to move it to an arbitrary position (within its limits of course). Use the script from Task 1.3 to obtain two positions. Then write a script that uses these two positions and moves the robot between them. Define the positions as `PoseObject`. To move the robot, you must call the move function and pass PoseObject as a parameter. For better results, you can add `wait()` and create a loop (for example, from 1 to 10).

One more tip. The arm is sometimes too fast – especially in situations where it moves to a different position than you think you have set (yes, people sometimes make mistakes). To prevent this, we recommend reducing the maximum speed of the arm using the `set_arm_max_velocity()` method, for example to 40% – yes, exactly, use the value 40 as the parameter.

Task 1.6 - Draw a square in the air with the robot

That was easy again, wasn't it? Now it is time to look at PoseObject[4] and also on `JointsPosition`[5]. The first one contains the position of the effector in Cartesian

[3] https://niryorobotics.github.io/pyniryo/v1.2.1-1/api/api.html#pyniryo.api.tcp_client.NiryoRobot.wait
[4] https://niryorobotics.github.io/pyniryo/v1.2.1-1/api/api.html#pyniryo.api.objects.PoseObject
[5] https://niryorobotics.github.io/pyniryo/v1.2.1-1/api/api.html#pyniryo.api.objects.JointsPosition

coordinates. It is useful for setting the arm to a specific position. If we want to move in a specific shape (for example, a square), it may be better to use `JointsPosition`.

For this task, you must first copy the script from Task 1.3 and change `get_pose()` to `get_joints()`. this will give you 6 numbers - each is the position of the joint in radians. In the next step, create a script in which you start at the current position of the joints and then move (by calling move with the `JointsPosition` object) to a new position that will be the same as the old position except for one small change in one joint (positive/negative change) – remember that this is in radians. Using this technique, try to determine which joint is which. For example, to start with, add a value of 0.2 rad to the first item to see where the robot will move.

Finally, create a script in which you try to draw a square by calling the move function with the JointPositions object. Again, this can be done in a loop.

Hint: Start from one defined position and add/subtract small values from the correct joints.

Task 1.7 - Getting started with the tools

This task is straightforward, do exactly same thing as in Task 1.6 but call the robot member methods `grasp_with_tool()` and `release_with_tool()` (alternately). Remember, before using tool it is a good practice to also call `robot.update_tool()`.

Task 2 - Getting started with the vision Set

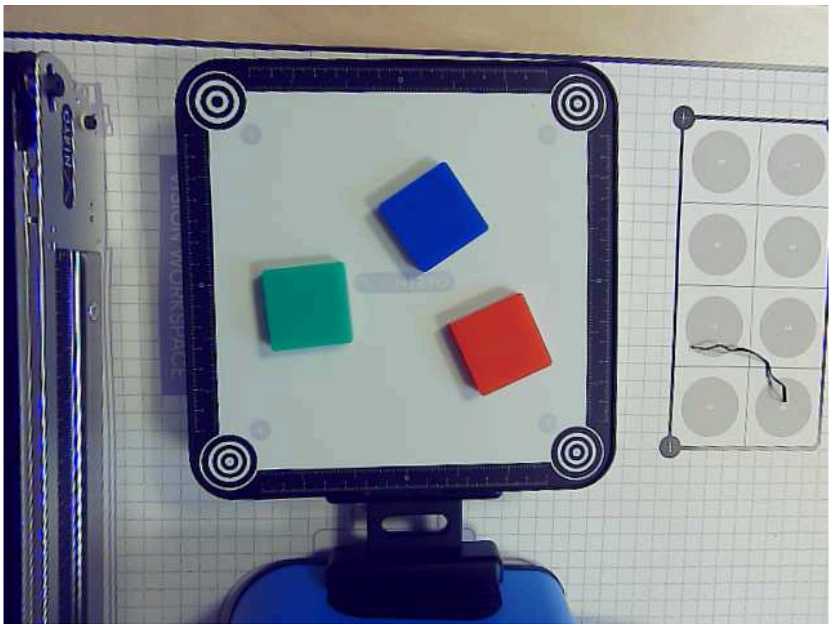
The goal of Task 2 is to familiarize you with the Niryo Vision Set. Its subtasks include preparing the workspace, working with images from the robot, and image processing.

Task 2.1 - Workspace preparation

To use vision set, it is necessary to create workspace in the robot's coordinate system. To do so, follow the oficial manual https://docs.niryo.com/accessories/vision-set/setup/ and video https://www.youtube.com/watch?v=S8BIA9ycF18. Once you create and select a workspace name, it will be available through the pyniryo API. If you physically move the workspace, you will need to recreate it.

Task 2.2 - Get image from the robot

Now you have a workspace for the Vision set. Using the code from Task 1.3 get position of the robot for the situation where the robot is observing the workspace – in this position, all 4 markers are visible.



In the next step use the member methods `get_img_compressed()` and `uncompress_image()`[6]. The result is a numpy array that can be processed using OpenCV, for example.

The goal of this task is to create a script that captures and saves an image containing the created workspace. We recommend saving multiple images with different configurations of objects in the scene. This will be useful for future tasks.

Task 2.3 - Detect object on the workspace

This task is supposed to introduce you to the object detection using built-in methods. The pyniryo API includes the following methods that are worth trying out

- `detect_object()`[7]
  - The method detects an object based on its shape and color (it is therefore not possible to select a specific red square if there is more than one red square in the image). It returns detection status (bool), a list containing relative position of an object, shape and color.
- `get_target_pose_from_rel()`[8]
  - Based on the relative position (not pixels directly) of the object, it is able to compute target position of the object for robot tool in the 3D coordinate system.
- `get_target_pose_from_cam()`[9]
  - Combines the previous methods. Based on the defined shape and color, it returns the target position in the 3D coordinate system.

[6] To use `uncompress_image`, it must be imported from `pyniryo.vision` (u can use same approach as in the case on pyniryo)
[7] https://niryorobotics.github.io/pyniryo/v1.2.1-1/api/api.html#pyniryo.api.tcp_client.NiryoRobot.detect_object
[8] https://niryorobotics.github.io/pyniryo/v1.2.1-1/api/api.html#pyniryo.api.tcp_client.NiryoRobot.get_target_pose_from_rel
[9] https://niryorobotics.github.io/pyniryo/v1.2.1-1/api/api.html#pyniryo.api.tcp_client.NiryoRobot.get_target_pose_from_cam

- `move_to_object()`[10]
  - Same as previous method, but move the robot to the detected object.

All methods except `detect_object()` have a parameter called `height_offset`. For your initial attempts, we strongly recommend using a positive value (moves the arm higher than the calculated position - specified in meters) to ensure that the arm does not collide with any objects or the surface of the workspace.

You can try object detection on saved images. In contrast, movement toward an object must be based on "live" data. Therefore, don't forget to add arm movement to the observation position and loading of the current image before each detection to the script.

The goal is to create a script that detects the selected object and moves the arm over it—the combination of methods is up to you.

Task 2.4 - Show image stream from the robot

In this task you are supposed to show the stream from the robot camera. It can be done using OpenCV `imshow()` method or a method in pyniryo, such as `show_img()`[11].

There is one more thing left to do. It should be possible to extract the part of the image that contains only the workspace. This can be done using two methods from the `pyniryo.vision` library. The first is `undistort_image()`[12] (It requires the camera's intrinsic parameters as arguments, which can be obtained using the `get_camera_intrinsics()`[13] method)

The second method is `extract_img_workspace()`[14]. For the vision set, workspace ratio parameter should be set to 1.0.

Task 2.5 - Implement own detection of the object (advanced)

The problem with the methods in the previous task is that sometimes they do not work well. In particular, sometimes they do not recognize the shape—for example, the target of the method is a red square, and the robot moves to a red circle. This usually happens when there is no object of the desired type in the image.

The goal is to do exactly the same as in the previous task, but you must detect the object and calculate its relative position (x, y, yaw) yourself..

First, extract the image workspace. Then perform detection—either using pyniryo methods or `opencv/skimage` libraries. You should detect the center of the object in pixels and its rotation in radians.

[10] https://niryorobotics.github.io/pyniryo/v1.2.1-1/api/api.html#pyniryo.api.tcp_client.NiryoRobot.move_to_object
[11] https://niryorobotics.github.io/pyniryo/v1.2.1-1/api/vision.html#pyniryo.vision.image_functions.show_img
[12] https://niryorobotics.github.io/pyniryo/v1.2.1-1/api/vision.html#pyniryo.vision.image_functions.undistort_image
[13] https://niryorobotics.github.io/pyniryo/v1.2.1-1/api/api.html#pyniryo.api.tcp_client.NiryoRobot.get_camera_intrinsics
[14] https://niryorobotics.github.io/pyniryo/v1.2.1-1/api/vision.html#pyniryo.vision.image_functions.extract_img_workspace

After detection, you have to call `relative_pos_from_pixels()`[15] to obtain relative position of the object, which can be used in pyniryo methods.

If you want to use pinyrio image processing methods, feel free to explore `pinyrio.vision` image processing[16] methods.

Try to solve a situation where you want a square, but you only have a circle available. Can you do better than the built-in methods?

Hint: Use thresholding[17] on color and contours or `skimage.measure` – `regionprops()`[18] – on shapes (circularity property).

Note: Remember that an RGB image is a 3D numpy matrix—three layers, one for each color. You can take OpenCV-Python tutorials[19]

Task 3 - Pick and Place

You should now be able to detect objects and get its target pose. Now its time to add methods for pick and plac.

Task 3.1 - Manual Pick and Place

First, you can try picking up and placing an object manually. Use the code from Task 1.3 to obtain a position of the object you want to pick up (open robot gripper and set the robot's position using free move). Then obtain the position for placing the object. Finally, call the `pick_and_place()`[20] method to move the object.

Task 3.2 - Vision based Pick and Place

Now do the same thing, but use the code from Task 2.3. You can use the `vision_pick()`[21] method. This is the easiest way, but it uses a built-in algorithm, so sometimes make mistakes.

To place the object to the place position, you have to call `place()` method. If you want to go to place location using specified trajectory, you can use the `move()` method and then the `release_with_tool()` method – i.e., do it by yourself instead of using the bult-in `place()` method.

[15] https://niryorobotics.github.io/pyniryo/v1.2.1-1/api/vision.html#pyniryo.vision.image_functions.relative_pos_from_pixels
[16] https://niryorobotics.github.io/pyniryo/v1.2.1-1/api/vision.html#pure-image-processing
[17] https://docs.opencv.org/4.x/d7/d4d/tutorial_py_thresholding.html
[18] https://scikit-image.org/docs/0.25.x/auto_examples/segmentation/plot_regionprops.html
[19] https://docs.opencv.org/4.x/d6/d00/tutorial_py_root.html
[20] https://niryorobotics.github.io/pyniryo/v1.2.1-1/api/api.html#pyniryo.api.tcp_client.NiryoRobot.pick_and_place
[21] https://niryorobotics.github.io/pyniryo/v1.2.1-1/api/api.html#pyniryo.api.tcp_client.NiryoRobot.vision_pick

Task 3.3 - Vision based Pick and Place with own detection algorithm

In this task, you are supposed to do same as in Task 3.2, but use your code from Task 2.5.

If you have completed this task, you are able to pick up and place objects using the Niryo robot.

Task 4 - Conveyor belt

To make it even more fun, we will learn how to work with a conveyor belt.

Task 4.1 - Control conveyor belt

The first thing todo is to control the conveyor belt itself – i.e., to start it running in one direction or the other.

To do this, it is necessary to tell pinyrio that you want to work with conveyor belt by calling `set_conveyor()`[22] method. It should be also unset after use – method `unset_conveyor()`[23]. Between these commands, you can use method `run_conveyor()`[24] with `conveyor_id`, `speed` and its `direction` as parameters.

As output, prepare a script that starts the belt in one direction for 10 seconds, then changes direction for another 10 seconds. Finally, the belt must be stopped (`stop_conveyor()`).

Task 4.2 - Using infrared sensor

Infrared sensor can be used to stop conveyor belt when something appears on the belt. It is connected to digital pin defined by `PinID.DI5`. Its state can be read using member method `digital_read()`[25]. And the state can be either `PinState.LOW` or `PinState.HIGH` (something is on a belt).

Now prepare a script that stops the conveyor belt when an object appears in front of the robot.

Task 4.3 - Pick and Place from workspace to Conveyor belt

Now, just combine code from Tasks 3.2 or 3.3 with 4.2. Write a script that performs the entire process, from selection using the camera, through placement on the conveyor belt, to starting the belt until the object reaches the infrared sensor.

[22] https://niryorobotics.github.io/pyniryo/v1.2.1-1/api/api.html#pyniryo.api.tcp_client.NiryoRobot.set_conveyor
[23] https://niryorobotics.github.io/pyniryo/v1.2.1-1/api/api.html#pyniryo.api.tcp_client.NiryoRobot.unset_conveyor
[24] https://niryorobotics.github.io/pyniryo/v1.2.1-1/api/api.html#pyniryo.api.tcp_client.NiryoRobot.run_conveyor
[25] https://niryorobotics.github.io/pyniryo/v1.2.1-1/api/api.html#pyniryo.api.tcp_client.NiryoRobot.digital_read

Task 4.4 - Pick and Place from Conveyor belt workspace to the numbered area

This task will be more tricky. The robot set contains 4 markers that can be added to conveyor belt. The goal is to create a second workspace above the conveyor belt and then pick up and place the object from belt into the numbered area.

Challenge: Can you pick and place more objects (which come in front of the sensor) in a row and place them to the numbered area?

Task 5 - LLM

This goal of Task 5 is to introduce you to work with an API of Large Language Model Services. In particular OpenAi – paid – and Ollama, where the server we will be using is located at the Department of Cybernetics (and the New Technologies for the Information Society Research Center) of the Faculty of Applied Sciences at the University of West Bohemia.

Task 5.1 - Basic use of OpenAi

The OpenAI API is a common way to work with LLM using Python. It is based on the OpenAI library. Unlike ChatGPT, it is not free. It is paid for according to a token-based pricing policy. To use the API, you must obtain an API key and then use it in your code:

```
from openai import OpenAI

client = OpenAI(pi_key=openai_api_key)
```

After that command you can send prompts. We recommend reading the documentation, especially about structured output[26] which will be very useful for this task. In summary, you have to create model of the structured output as a child of BaseModel from pydantic library. Then you have to call `client.chat.completions.parse` method with desired model such as `gpt-4o`.

In this task try to use saved images and ask OpenAI[27] if it can select, for example, a red square. The output should be the text answer itself, followed by the shape and color in a format that pyniryo can process.

Hint: To send image to OpenAi it has to be encoded in Base64 format. Use base64 library for this.

```
def encode_image(image_path):
```

[26] https://platform.openai.com/docs/guides/structured-outputs

[27] Remember that when using OpenAI, you pay for each token. Therefore, try out the task, but don't call it unnecessarily many times. And above all, **never call the prompt in a loop**.

```
    with open(image_path, "rb") as image_file:
        return base64.b64encode(image_file.read()).decode('utf-8')
```

Task 5.2 - Basic use of Ollama

The goal is to do the same but with Ollama. Look at the project page ib GitHub[28]. You'll find lots of examples there, including structured output. Login details for Ollama will be handed out during the summer school.

Task 5.3 - Prepare for LLM Pick and Place

This task is focused on preparing LLM part for Pick and Place. The goal is to write a python library (set of functions) that provides LLM – i.e. It can be easily imported and called upon to perform a specific function for a specific task. It should include Color and Shape structures, a Response structure, and API calls to OpenAi or Ollama.

Task 6 - Pick and Place using LLM

Congratulations, you have completed all the sub-sections that helped you learn how to work with the robot, its accessories, and LLM. In this task, you will connect all the parts together.

Task 6.1 - Pick and Place using LLM

The goal is to write a script that sends a text prompt to the LLM and, based on its result, picks up and places the desired object from the image processing workspace onto the conveyor belt.

Task 6.2 - Pick and Place using LLM advanced

An advanced version of the previous task. The goal is to create a prompt that asks the LLM to generate a list of objects (for example, all red objects in the workspace) and then perform pick and place of all listed objects into a numbered area.

Task 7 - Be Creative

Ideas:

- Add an Automatic Speech Recognition to prompt LLM in a more comfortable way.
- Create a GUI app that will work as a chat bot for controlling of the robot.
- Use text to speech - yes robot can speak just look over here https://niryorobotics.github.io/pyniryo/v1.2.1-1/api/api.html#pyniryo.api.tcp_client.NiryoRobot.say
- Simulate quality control process.
- Shape or color based sorting machine?

[28] https://github.com/ollama/ollama-python