JEMARO - July 2024 hackathon instructions

July 5, 2024

1 Task description

During the hackathon, the students will be tasked with programming a robot to perform a "pick and place" task based on computer vision and creating documentation for a developed solution. The task will be performed in teams of 3-4 students on real-world equipment. The overall course of the "pick and place" task should be as follows:

1. detecting the object in the view of the camera,
2. determining the object's position and orientation,
3. moving the end of the manipulator to the object,
4. grabbing the object and picking it up,
5. detecting the object's destination in the view of the camera,
6. determining the destination's position and orientation,
7. moving the end of the manipulator with the object to the destination,
8. letting go of the object and moving the manipulator tip away.

1.1 Manipulation object and destination

The manipulation object is a 3D printed geometry figure - a circle, a square, or an equilateral triangle. The object is fairly light, thin, several centimeters in size, and of intense color. The object's destination will be a larger black tile with a colored indent where the object should be placed. Several different tiles will be prepared, with different indent sizes - from a fairly tight fit to a loose one. The example object and its dedicated tiles are presented in figures 1 and 2.

1.2 Assessment criteria

The results presented at the end of the hackathon will be assessed accordingly:

- system documentation clarity,
- system documentation quality,
- implementation reliability - does the solution work for different object placements,
- fitting accuracy - which hole size does the manipulation target can be placed,
- how many different shapes are handled by the solution.

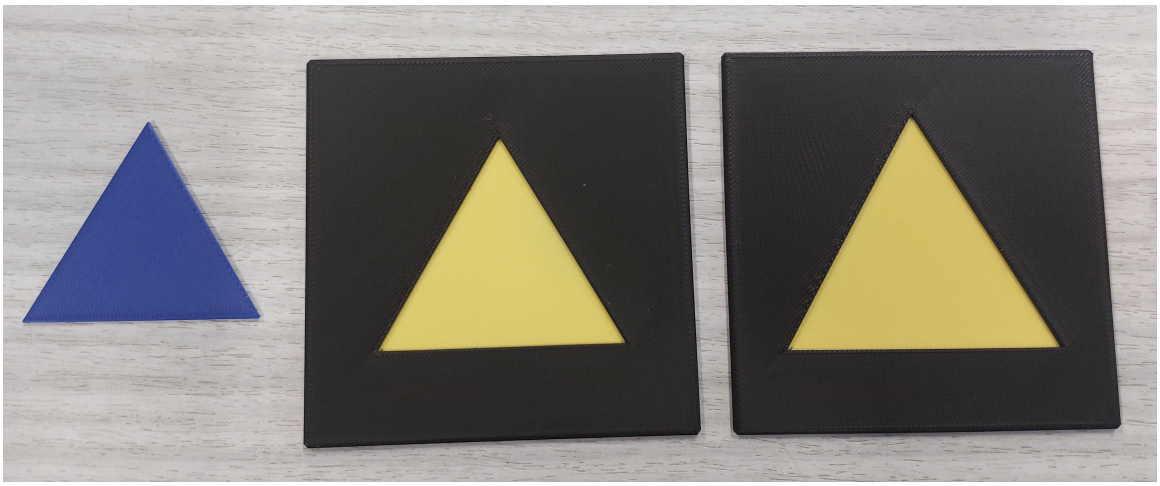
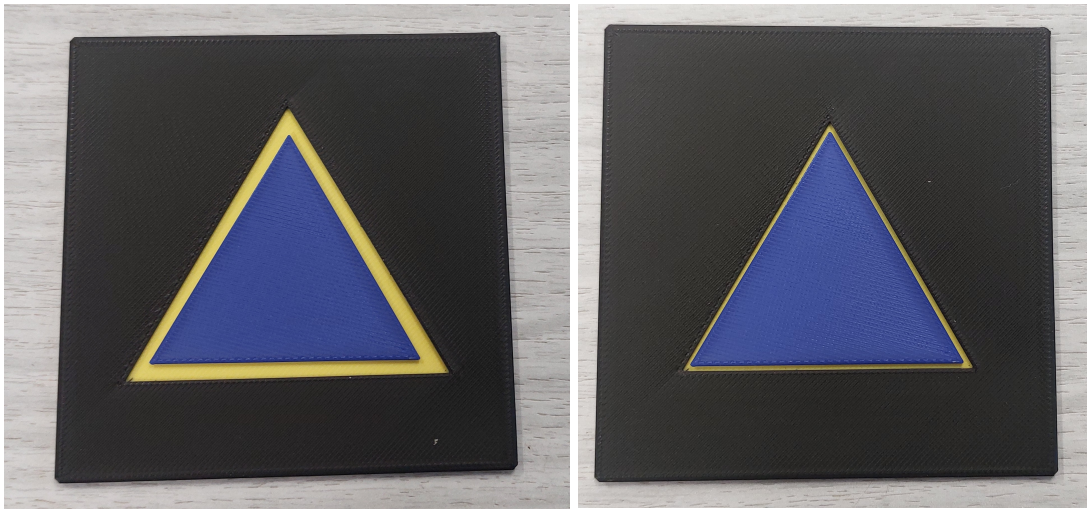


Figure 1: From the left: sample manipulation object, tight tile, looser tile



(a) Loose indent (b) Tight indent

Figure 2: Object placed inside the tile's indent

2 Logging onto the computer

Each computer has been prepared for the hackathon. To access the prepared space, one should choose the Ubuntu 22.04 operating system when turning on the machine. The credentials for the prepared account are the following:

- **User:** student-en
- **Password:** student

3 Connecting hardware

To work with the supplied hardware, both the robot and the camera must be connected to the computer's USB ports. The proper sequence of connecting the hardware is:

1. Connect the robot to the power supply with the power cord.
2. Connect the robot to any of the computer's blue USB ports using provided USB B - USB A cable.
3. Turn on the robot using the large button with the power symbol placed on the top of the robot's platform.
    - The robot may perform some movement when turned on. Don't try to stop it or otherwise interfere with this movement.
4. Connect the camera to any of the computer's blue USB ports using provided USB C - USB A cable.

Important Connecting the camera before the robot may cause communication problems with the latter.

4 Prepared software

Every computer has software for both the robot and the camera installed. Participants **don't** have to build them themselves. Every opened terminal should automatically source all of the basic ROS2 libraries along with these responsible for hardware control:

- https://github.com/GroupOfRobots/magician_ros2
- https://github.com/IntelRealSense/realsense-ros.git

5 Starting the robot

1. In the first terminal, launch the robot (wait for the "Dobot Magician control stack has been launched correctly" message, don't close the terminal while working with the robot):

```
ros2 launch dobot_bringup dobot_magician_control_system.launch.py
```

2. In the second terminal, launch the camera (don't close the terminal while working with the camera):

```
ros2 launch realsense2_camera rs_launch.py
```

3. In the third terminal, home the robot:

```
ros2 service call /dobot_homing_service dobot_msgs/srv/
    ExecuteHomingProcedure
```

4. (Optional) If the robot won't home properly and makes strange noises, it's possible that its joints are out of the allowed movement range. To fix this issue, press and hold the padlock button near the end of the robot's arm and move the arm to a more acceptable position. Then return to step 3.

6 Published topics

- /joint_states (sensor_msgs/msg/JointState) - angles values in the manipulator's joints
- /dobot_TCP (geometry_msgs/msg/PoseStamped) - position of the coordinate frame associated with the end of the last robot link (orientation given as a quaternion)
- /dobot_pose_raw (std_msgs/msg/Float64MultiArray) - position of the coordinate frame associated with the end of the last robot link (raw orientation received from Dobot, expressed in degrees)
- /camera/camera/color/image_raw (sensor_msgs/msg/Image) - image from realsense camera
- /camera/camera/color/camera_info (sensor_msgs/msg/CameraInfo) - properties from realsense camera (e.g. image size, distortion matrix)

7 Motion

The motion of the manipulator is handled using the ROS 2 action. For the manipulator to move to the desired position, the motion target must be sent to the action server. The following describes the structure of the message sent to the action server (part of dobot_msgs/action/PointToPoint):

- motion_type:
  - 1 -> joint interpolated motion, target expressed in Cartesian coordinates
  - 2 -> linear motion, target expressed in Cartesian coordinates
  - 4 -> joint interpolated motion, target expressed in joint coordinates
  - 5 -> linear motion, target expressed in joint coordinates
- target_pose - desired position expressed in Cartesian coordinates [mm] or in joint coordinates [degrees]
- velocity_ratio (default 1.0)
- acceleration_ratio (default 1.0)

It is advised to use motion_type equal to 1. It should also be noted, that the target position expressed in Cartesian coordinates is **not** the position of the suction cup, but the position of the coordinate frame associated with the end of the last robot link (for visual representation one may check the "info" tab at robot's Control panel or check robot's visualization in Rviz - see "Other useful tools" section). An example of a command that allows you to send a goal to an action server can be found below (adding –feedback flag will cause the terminal to display the current position of the robot while it is moving):

```
ros2 action send_goal /PTP_action  dobot_msgs/action/PointToPoint "{
    motion_type: 1, target_pose: [200.0, 0.0, 100.0, 0.0], velocity_ratio:
    0.5, acceleration_ratio: 0.3}" --feedback
```

If you want to cancel the goal, run the following command:

```
ros2 service call /PTP_action/_action/cancel_goal action_msgs/srv/CancelGoal
```

8 Suction cup

The robot's end effector is a suction cup that can be turned on and off using the proper ros service.

```
ros2 service call /dobot_suction_cup_service dobot_msgs/srv/
    SuctionCupControl "{enable_suction: true}"
```

9 Other usefull tools

- Rviz visualization and transformation frames (\tf topic, please note that the linear transformations between the camera, suction cup, and coordinate frame associated with the end of the last robot link are not fully accurate and those who need to use them are advised to tune these values themselves):

  ```
  ros2 launch dobot_description display.launch.py DOF:=4 tool:=suction_cup
       use_camera:=true
  ```

- Control panel in rqt:

  ```
  rqt -s dobot_control_panel
  ```

- Robot diagnostic in rqt:

  ```
  rqt -s rqt_robot_monitor
  ```

10 Shape detection

The participants are free to use any library they want for shape detection.

For those who don't have a preferred library, we have prepared a simple Python one, in the form of the *shapes.py* file. There are three functions in this file, that may be of use to the participants and are summarized below. The example usage of these functions can also be found inside the file itself.

- shapes = detectShapes(image, colors, threshold = 40) - detects shapes of chosen colors on the provided image, it requires 2-3 arguments and outputs 1 variable:
  - image - image loaded into Python program (for example using cv2.imread(fname) function from openCV library)
  - colors - dictionary of colors of the detected objects, where each key is the color name and value is a color presented as a list in BGR format

  - threshold (optional) - changing this value increases or decreases the sensitivity of color detection
  - shapes - dictionary consisting of detected shapes with information of their position in the provided image, points in the contour, color, and number of corners, center, orientation, and radius

- shapes = localizeShapes(detected_shapes, refRadius, cameraInfo) - localizes shapes in the camera frame using information about the camera and detected object, it requires 3 arguments and outputs 1 variable:
  - detected_shapes - dictionary received from detectShapess function
  - refRadius - radius of the object we are trying to detect (largest distance from center to any point in contour)
  - cameraInfo - python dictionary containing a single key 'k' with value in the form of the list of 9 numbers representing the distortion matrix of the camera
  - shapes - dictionary similar to the one given as input, but each shape has its position in the camera frame

- out_image = drawShapes(image, shapes, colors) - draws detected shapes on provided image
  - image - original image
  - shapes - dictionary received from localizeShapes function
  - colors - dictionary of colors
  - out_image - image with shapes drawn on it

11 Other sources of documentation

If participants of the hackathon, despite the information above, have trouble using the provided software and require additional information, they are free to utilize the extensive documentation at the README of https://github.com/GroupOfRobots/magician_ros2 repository or ask the supervising personnel for help.