Technical University of Cluj-Napoca

Executing picking and placement of an object from and to specific locations using Doosan M1013 cobot

Student name: *Evelyn-Iulia Plesca*

Course: *Robot Control Systems* – Professor: *Dr. Ing. Tassos Natsakis*
Due date: *January 2024*

Introduction

The Doosan Robotics M1013 is a collaborative robot (cobot) designed for a variety of tasks in industrial environments.

It is known for its sleek and ergonomic design. It is built with six-axis flexibility, which allows for smooth and precise movements, mimicking the dexterity of a human arm. This cobot has a payload capacity of 10 kilograms, making it suitable for handling medium-weight objects. This capability is ideal for tasks like packaging, assembly, and material handling. It has a reach of 1.3 meters, enabling it to operate efficiently in both small and medium-sized workspaces.

The M1013 is designed to work alongside human workers safely and is versatile, being used in various industries, including automotive, electronics, food and beverage and others. Its flexibility and accuracy make it suitable for tasks like machine tending, material handling, and quality inspection.

Figure 1: Doosan Robotics M1013 cobot

The following project aims to create an algorithm for the M1013 robot model that can pick and place objects from and to specific locations, which are inputted from the user.

Packages

For this application we used:

- ROS Noetic
- Gazebo 11
- ROS Software for Doosan Robot (found on Github)

In order to simulate our robot, the *dsr_launcher* directory from the Github repository of the Doosan Robot was used, which contained multiple launch options for the robot configurations, facilitating their simulation and control.

The *single_robot_gazebo.launch* launcher was mainly used, which launched the ROS environment specifically configured for a single Doosan robot.

A package was created containing:

- **Manifest files** (*package.xml*) contains metadata about tehe package, including its name, version and dependencies
- **Source Code** found in the scripts folder
- **Build Configuration Files** (*CMakeLists.txt*) which describe how to build the code, including instructions for compiling and linking
- **Launch files** included in order to help us simulate the robot
- **Resource Files** which contain libraries of the Doosan Robot, later used in our code

Implementation

The algorithm somehow resembles what we have done in the second exercise from laboratory 5.

First, we have to initialize the robot and the grippers, after which while rospy is not shutdown the operations of picking and placing an object are executed cyclically.

The user must enter in the terminal the initial and final positions of the object which needs to be manipulated in an x y z a b c format, after which a function is called with these variables.

The Pick and Place function which is called moves to the initial pose, grasping the object, after which it picks it up and moves to the second pose, where it releases the object.

In the final, the gripper moves to the reference base after performing the operation.

```
# ##
# @brief    The implementation of the RCS project, which targets the task
  of
#             picking and placing an object from and to specific locations
  using the
#             Doosan M1013 robotic platform
# @author   Plesca Evelyn-Iulia (plescaevelyn)

```

```
# Importing needed libraries
import rospy
import os
import threading, time
import sys
sys.dont_write_bytecode = True
sys.path.append('/home/plescaevelyn/catkin_ws/src/y4-IA/RCS/project/scripts')
sys.path.append( os.path.abspath(os.path.join(os.path.dirname(__file__),
"../../../../doosan-robot/common/imp")) ) # get import path : DSR_ROBOT.py

# defining a single robot
ROBOT_ID     = "dsr01"
ROBOT_MODEL  = "m1013"
import DR_init
DR_init.__dsr__id = ROBOT_ID
DR_init.__dsr__model = ROBOT_MODEL
from DSR_ROBOT import *

# Global variables for ROS service proxies and publisher
pub_stop = None
srv_robotiq_2f_open = None
srv_robotiq_2f_move = None

# Define functions for opening and closing the gripper
def gripper_grasp(width):
    global srv_robotiq_2f_move

    # close the gripper with a width of 0.1 - 0.8 units
    try:
        # Attempt to call the service
        srv_robotiq_2f_move(width)
    except rospy.ServiceException as e:
        rospy.logerr("Service call failed: %s", e)

def gripper_release():
    global srv_robotiq_2f_open

    # open the gripper
    try:
        # Attempt to call the service
        srv_robotiq_2f_open()
    except rospy.ServiceException as e:
        rospy.logerr("Service call failed: %s", e)

def SET_ROBOT(id, model):
    ROBOT_ID = id; ROBOT_MODEL= model

def pick_and_place(initial_pose, final_pose):
    # Move to the initial pose
    movej(initial_pose, vel=60, acc=30)
```

```

    # Close the gripper to pick up the object
    gripper_grasp(0.4)

    # Wait for the gripper to close
    time.sleep(1)

    # Relative motion for picking up the object
    movel(x1, velx, accx, 2, 0, MOVE_REFERENCE_BASE, MOVE_MODE_RELATIVE)

    # Move to the final pose
    movej(final_pose, vel=60, acc=30)

    # Relative motion for placing the object
    movel(x1, velx, accx, 2, 0, MOVE_REFERENCE_BASE, MOVE_MODE_RELATIVE)

    # Open the gripper to place the object
    gripper_release()

    # Wait for the gripper to open
    time.sleep(1)

    # Relative motion after placing the object
    movel(x2, velx, accx, 2, 0, MOVE_REFERENCE_BASE, MOVE_MODE_RELATIVE)

def shutdown():
    print("shutdown time!")
    print("shutdown time!")
    print("shutdown time!")

    pub_stop.publish(stop_mode=1) #STOP_TYPE_QUICK)
    return 0

# convert list to Float64MultiArray
def _ros_listToFloat64MultiArray(list_src):
    _res = []
    for i in list_src:
        item = Float64MultiArray()
        item.data = i
        _res.append(item)
    #print(_res)
    #print(len(_res))
    return _res

if __name__ == "__main__":
    # set target robot
    my_robot_id    = "dsr01"
    my_robot_model = "m1013"
    SET_ROBOT(my_robot_id, my_robot_model)

```

```python
    rospy.init_node('pick_and_place_simple_py')
    rospy.on_shutdown(shutdown)

    # Check if the gripper open service is available
    rospy.wait_for_service('/' + ROBOT_ID + ROBOT_MODEL +
     ↪  '/gripper/robotiq_2f_open', timeout=None)
    # Check if the gripper move service is available
    rospy.wait_for_service('/' + ROBOT_ID + ROBOT_MODEL + '/gripper/g',
     ↪  timeout=None)

    pub_stop = rospy.Publisher('/'+ROBOT_ID+ROBOT_MODEL+'/stop', RobotStop,
     ↪  queue_size=10)

    srv_robotiq_2f_open = rospy.ServiceProxy('/' + ROBOT_ID + ROBOT_MODEL +
     ↪  '/gripper/robotiq_2f_open', Robotiq2FOpen)
    srv_robotiq_2f_move = rospy.ServiceProxy('/' + ROBOT_ID + ROBOT_MODEL +
     ↪  '/gripper/g', Robotiq2FMove)

    while not rospy.is_shutdown():
            # position and rotation from coordinates are needed
        print("Enter the initial position (format: x y z a b c): ")
        initial_pose = list(map(float, input().split()))
        print("Enter the final position (format: x y z a b c): ")
        final_pose = list(map(float, input().split()))

        # Call the pick_and_place function with the provided positions
        pick_and_place(initial_pose, final_pose)

    print('good bye!')
```

Limitations

One of the limitations of this application is the mechanical precision, because the inherent mechanical tolerances and backlash in the robot's joints and gears can affect precision.

Another limitation might be the grasping of the object using the gripper, since it takes values between 0 and 0.8 and in the application, the width was hard coded to a value of 0.4, therefore it doesn't work with any object. There would be further improvements needed for more precision, such as including a sensor that senses the width of the object that needs to be handled. Another problem might occur if the object is too heavy to grasp as well.

Conclusions

The presented project manages to create a simple form of object picking and placement despite all its limitations, constituting a good base for any application of this type.

As future improvements, enhancements and refinements to the current system could lead to a more reliable and precise robot. One of these could be using sensors to see the width of the object in order to control the gripper more efficiently.

In conclusion, this project has been a good introductory path towards unlocking the full potential of the Doosan M1013 cobot, implementing a basic task: pick and place an object from and to specific locations, which helped sediment the knowledge acquired during this semester of Robot Operating System and Robot Control System.

References

I have used resources from the following pages:

1. ROS Tutorials
2. Doosan Robot Github Repository
3. Robot User Manual
4. Gripper User Manual
5. Doosan Robot ROS Workshop