# EECS 4421 Introduction to Robotics

## Lab 5 Submission

Kasra Cheraghi Seaifabat
kasracheraghi27@gmail.com
Student No.: 218319608

Mahfuz Rahman
mafu@my.yorku.ca
Student No.: 217847518

Please find the code pasted below and the video file attached with the submission.

Structure of the code:

We implemented our image detection logic into the yolo_pose model. This model will interpret hand gestures and publish to a node.

We modified the ros code implementation provided by kinova to control the robot. Using kinova's model for robot control, we subscribed to the node that is published by the yolo_pose model. We then interpret the message and send instructions to the robot accordingly.

# Code for yolo_pose.py :

```
import os
import sys
import rclpy
import cv2
import datetime
import numpy as np
import math
from rclpy.node import Node
from ultralytics import YOLO
from cv_bridge import CvBridge
```

```
from std_srvs.srv import SetBool
from sensor_msgs.msg import Image
from geometry_msgs.msg import Twist
from rclpy.qos import qos_profile_sensor_data
from ultralytics.engine.results import Results, Keypoints
from ament_index_python.packages import get_package_share_directory

import time
from std_msgs.msg import String


# Maximum allowed waiting time during actions (in seconds)
TIMEOUT_DURATION = 20

class YOLO_Pose(Node):
    _BODY_PARTS = ["NOSE", "LEFT_EYE", "RIGHT_EYE", "LEFT_EAR", "RIGHT_EAR", "LEFT_SHOULDER",
"RIGHT_SHOULDER",
                   "LEFT_ELBOW", "RIGHT_ELBOW", "LEFT_WRIST", "RIGHT_WRIST", "LEFT_HIP", "RIGHT_HIP",
"LEFT_KNEE",
                   "RIGHT_KNEE", "LEFT_ANKLE", "RIGHT_ANKLE"]

    def __init__(self):
        super().__init__('pose_node')

        # params
        self._model_file = os.path.join(get_package_share_directory('cpmr_ch12'), 'yolov8n-pose.pt')
        self.declare_parameter("model", self._model_file)
        model = self.get_parameter("model").get_parameter_value().string_value

        self.declare_parameter("device", "cpu")
        self._device = self.get_parameter("device").get_parameter_value().string_value

        self.declare_parameter("threshold", 0.5)
        self._threshold = self.get_parameter("threshold").get_parameter_value().double_value


        self.declare_parameter("camera_topic", "/mycamera/image_raw")
```

```
    self._camera_topic = self.get_parameter("camera_topic").get_parameter_value().string_value



    self._move_flag = False
    self._bridge = CvBridge()
    self._model = YOLO(model)
    self._model.fuse()

    # subs
    self._sub = self.create_subscription(Image, self._camera_topic, self._camera_callback, 1)

    # pubs
    self._publisher = self.create_publisher(String, "/moveRobot", 1)




def parse_keypoints(self, results: Results):
    keypoints_list = []

    for points in results.keypoints:
        if points.conf is None:
            continue

        for kp_id, (p, conf) in enumerate(zip(points.xy[0], points.conf[0])):
            if conf >= self._threshold:
                keypoints_list.append([kp_id, p[0], p[1], conf])

    return keypoints_list



def _camera_callback(self, data):
    #self.get_logger().info(f'{self.get_name()} camera callback')
    img = self._bridge.imgmsg_to_cv2(data)
    results = self._model.predict(
            source = img,
            verbose = False,
```

```
        stream = False,
        conf = self._threshold,
        device = self._device
    )

    if len(results) != 1:
        self.get_logger().info(f'{self.get_name()}  Nothing to see here or too much {len(results)}')
        return

    results = results[0].cpu()
    if len(results.boxes.data) == 0:
        self.get_logger().info(f'{self.get_name()}  boxes are too small')
        return



    left_shoulder = None # 5
    left_wrist = None # 9
    right_shoulder = None # 6
    right_wrist = None  # 10

    if results.keypoints:
        keypoints = self.parse_keypoints(results)
        if len(keypoints) > 0:
            for i in range(len(keypoints)):
                coordinates = [ keypoints[i][1], keypoints[i][2], keypoints[i][3] ]

                #left side
                if(keypoints[i][0] == 5):
                    left_shoulder = coordinates
                elif(keypoints[i][0] == 9):
                    left_wrist = coordinates

                #right side
                elif(keypoints[i][0] == 6):
                    right_shoulder = coordinates
                elif(keypoints[i][0] == 10):
                    right_wrist = coordinates
```

```
        # both hands
        elif(keypoints[i][0] == 9):
            left_wrist = coordinates
        elif(keypoints[i][0] == 10):
            right_wrist = coordinates

    msg = String()

    # Both hands up
    if right_wrist and left_wrist:
        if((right_wrist[1] < right_shoulder[1]) and (left_wrist[1] < left_shoulder[1])):
            self.publish("Both Hands Up")
            msg.data = "5"

    # Left Hand
    elif left_shoulder and left_wrist:
        if(left_wrist[1] < left_shoulder[1]):
            self.publish("Left Hand Up")
            msg.data = "1"
        else:
            self.publish("Left Hand Down")
            msg.data = "2"



    # Right Hand
    elif right_shoulder and right_wrist:
        if(right_wrist[1] < right_shoulder[1]):
            self.publish("Right Hand Up")
            msg.data = "3"
        else:
            self.publish("Right Hand Down")
            msg.data = "4"

    #publish message
    self._publisher.publish(msg)
```

```
            # Visualize results on frame
            annotated_frame = results[0].plot()
            cv2.imshow('Results', annotated_frame)
            cv2.waitKey(1)


    def publish(self, keypoints):
        self.get_logger().info(f' {keypoints}')



def main(args=None):

    # Import the utilities helper module
    rclpy.init(args=args)
    node = YOLO_Pose()
    try:
        rclpy.spin(node)
    except KeyboardInterrupt:
        pass
    rclpy.shutdown()

if __name__ == '__main__':
    main()
```

## Code for kinova_gen3_node.py:

```
#
# This is an absolutely minimal ros2 wrapper around some demo code for the Kinova arm.
# Absolutely no apologies for what is happening here. Including the terrible hack in the
# utilities code included. This could all be made much prettier, etc.
#
```

```
from kinova_gen3_interfaces.srv import Status, SetGripper, GetGripper, SetJoints, GetJoints, GetTool, SetTool
import rclpy
from rclpy.node import Node

import sys
import os
import time
import threading

from kortex_api.autogen.client_stubs.BaseClientRpc import BaseClient
from kortex_api.autogen.client_stubs.BaseCyclicClientRpc import BaseCyclicClient
from kortex_api.autogen.messages import Base_pb2, BaseCyclic_pb2, Common_pb2

from kinova_gen3.utilities import parseConnectionArguments, DeviceConnection

from std_msgs.msg import String


# Maximum allowed waiting time during actions (in seconds)
TIMEOUT_DURATION = 20

# Create closure to set an event after an END or an ABORT
def check_for_end_or_abort(e):
    """Return a closure checking for END or ABORT notifications

    Arguments:
    e -- event to signal when the action is completed
        (will be set when an END or ABORT occurs)
    """
    def check(notification, e = e):
        print("EVENT : " + \
              Base_pb2.ActionEvent.Name(notification.action_event))
        if notification.action_event == Base_pb2.ACTION_END \
        or notification.action_event == Base_pb2.ACTION_ABORT:
            e.set()
    return check
```

```python
def example_move_to_home_position(base):
    # Make sure the arm is in Single Level Servoing mode
    base_servo_mode = Base_pb2.ServoingModeInformation()
    base_servo_mode.servoing_mode = Base_pb2.SINGLE_LEVEL_SERVOING
    base.SetServoingMode(base_servo_mode)

    # Move arm to ready position
    print("Moving the arm to a safe position")
    action_type = Base_pb2.RequestedActionType()
    action_type.action_type = Base_pb2.REACH_JOINT_ANGLES
    action_list = base.ReadAllActions(action_type)
    action_handle = None
    for action in action_list.action_list:
        if action.name == "Home":
            action_handle = action.handle

    if action_handle == None:
        print("Can't reach safe position. Exiting")
        return False

    e = threading.Event()
    notification_handle = base.OnNotificationActionTopic(
        check_for_end_or_abort(e),
        Base_pb2.NotificationOptions()
    )

    base.ExecuteActionFromReference(action_handle)
    finished = e.wait(TIMEOUT_DURATION)
    base.Unsubscribe(notification_handle)

    if finished:
        print("Safe position reached")
    else:
        print("Timeout on action notification wait")
    return finished

def set_gripper(base, position):
```

```
    gripper_command = Base_pb2.GripperCommand()
    finger = gripper_command.gripper.finger.add()

    # Close the gripper with position increments
    print("Performing gripper test in position...")
    gripper_command.mode = Base_pb2.GRIPPER_POSITION
    finger.value = position
    print(f"Going to position {position}")
    base.SendGripperCommand(gripper_command)


def get_gripper(base):
    gripper_request = Base_pb2.GripperRequest()
    gripper_request.mode = Base_pb2.GRIPPER_POSITION
    gripper_measure = base.GetMeasuredGripperMovement(gripper_request)
    if len (gripper_measure.finger):
        print(f"Current position is : {gripper_measure.finger[0].value}")
        return gripper_measure.finger[0].value
    return None

def example_angular_action_movement(base, angles=[0.0, 0.0, 0.0, 0.0, 0.0, 0.0]):

    print("Starting angular action movement ...")
    action = Base_pb2.Action()
    action.name = "Example angular action movement"
    action.application_data = ""

    actuator_count = base.GetActuatorCount()

    # Place arm straight up
    print(actuator_count.count)
    if actuator_count.count != len(angles):
        print(f"bad lengths {actuator_count.count} {len(angles)}")
    for joint_id in range(actuator_count.count):
        joint_angle = action.reach_joint_angles.joint_angles.joint_angles.add()
        joint_angle.joint_identifier = joint_id
        joint_angle.value = angles[joint_id]
```

```
    e = threading.Event()
    notification_handle = base.OnNotificationActionTopic(
        check_for_end_or_abort(e),
        Base_pb2.NotificationOptions()
    )

    print("Executing action")
    base.ExecuteAction(action)

    print("Waiting for movement to finish ...")
    finished = e.wait(TIMEOUT_DURATION)
    base.Unsubscribe(notification_handle)

    if finished:
        print("Angular movement completed")
    else:
        print("Timeout on action notification wait")
    return finished

def get_angular_state(base_cyclic):
    feedback = base_cyclic.RefreshFeedback()
    actuators = feedback.actuators
    v = []
    for j in actuators:
        v.append(j.position)
    return v

def example_cartesian_action_movement(base, x, y, z, theta_x, theta_y, theta_z):

    print("Starting Cartesian action movement ...")
    action = Base_pb2.Action()
    action.name = "Example Cartesian action movement"
    action.application_data = ""

    cartesian_pose = action.reach_pose.target_pose
    cartesian_pose.x = x
```

```
    cartesian_pose.y = y
    cartesian_pose.z = z
    cartesian_pose.theta_x = theta_x
    cartesian_pose.theta_y = theta_y
    cartesian_pose.theta_z = theta_z

    e = threading.Event()
    notification_handle = base.OnNotificationActionTopic(
        check_for_end_or_abort(e),
        Base_pb2.NotificationOptions()
    )

    print("Executing action")
    base.ExecuteAction(action)

    print("Waiting for movement to finish ...")
    finished = e.wait(TIMEOUT_DURATION)
    base.Unsubscribe(notification_handle)

    return finished

def get_tool_state(base_cyclic):
    feedback = base_cyclic.RefreshFeedback()
    base = feedback.base

    return  base.tool_pose_x, base.tool_pose_y, base.tool_pose_z, base.tool_pose_theta_x, base.tool_pose_theta_y,
base.tool_pose_theta_z


class Kinova_Gen3_Interface(Node):

    def __init__(self):
        super().__init__('kinova_gen3_interface')
        self.get_logger().info(f'{self.get_name()} created')

        self.create_service(Status, "home", self._handle_home)
        self.create_service(GetGripper, "get_gripper", self._handle_get_gripper)
```

```
        self.create_service(SetGripper, "set_gripper", self._handle_set_gripper)
        self.create_service(SetJoints, "set_joints", self._handle_set_joints)
        self.create_service(GetJoints, "get_joints", self._handle_get_joints)
        self.create_service(SetTool, "set_tool", self._handle_set_tool)
        self.create_service(GetTool, "get_tool", self._handle_get_tool)

        self._base = None
        self._base_cyclic = None

        args = parseConnectionArguments()
        with DeviceConnection.createTcpConnection(args) as router:
            self._router = router
            self._base = BaseClient(self._router)
            self._base_cyclic = BaseCyclicClient(self._router)

        self._subscriber_keypoints_1 = self.create_subscription(String, "/moveRobot", self.moveRobot ,1)


    def moveRobot(self, msg):
        msg = msg.data

        self.get_logger().info(f'{msg}')

        # Left Hand Up
        if(msg == "1"):
            self.get_logger().info(f'{msg} Left Hand Up')
            example_angular_action_movement(self._base, [-90, -30, 130, 90, 20, 10])
            time.sleep(2)

        # Left Hand Down
        elif(msg == "2"):
            self.get_logger().info(f'{msg} Left Hand Down')
            example_angular_action_movement(self._base, [90, -30, 130, 90, 20, 10])
            time.sleep(2)

        # Right Hand Up
        elif(msg == "3"):
```

```
            self.get_logger().info(f'{msg} Right Hand Up')
            example_angular_action_movement(self._base, [0, -20, 150, 90, 0, 10])
            time.sleep(2)

        # Right Hand Down
        elif(msg == "4"):
            self.get_logger().info(f'{msg} Right Hand Down')
            example_angular_action_movement(self._base, [0, -40, 100, 90, 30, 10])
            time.sleep(2)

        # Both Hands Up
        elif(msg == "5"):
            self.get_logger().info(f'{msg} Both Hands Up')
            example_angular_action_movement(self._base, [0, -30, 120, 90, 30, 10])
            time.sleep(2)


        return


    def _handle_home(self, request, response):
        """Move to home"""
        self.get_logger().info(f'{self.get_name()} moving to home')

        response.status = example_move_to_home_position(self._base)
        return response

    def _handle_get_gripper(self, request, response):
        """Get gripper value"""
        self.get_logger().info(f'{self.get_name()} Getting gripper value')

        response.value = get_gripper(self._base)
        return response

    def _handle_set_gripper(self, request, response):
        """Set gripper value"""
        self.get_logger().info(f'{self.get_name()} Setting gripper value')
```

```
        set_gripper(self._base, request.value)
        response.status = True
        return response

    def _handle_set_joints(self, request, response):
        """Set joint values"""
        self.get_logger().info(f'{self.get_name()} Setting joint values')
        if len(request.joints) != 6:
            self.get_logger().info(f'{self.get_name()} Must specify exactly six joint angles')
            response.status = False
            return response

        response.status = example_angular_action_movement(self._base, angles=request.joints)
        return response

    def _handle_get_joints(self, request, response):
        """Get joint values"""
        self.get_logger().info(f'{self.get_name()} Getting joint values')
        response.joints = get_angular_state(self._base_cyclic)
        return response


    def _handle_set_tool(self, request, response):
        """Set tool values"""
        self.get_logger().info(f'{self.get_name()} Setting tool values')

        response.status = example_cartesian_action_movement(self._base, request.x, request.y, request.z,
request.theta_x, request.theta_y, request.theta_z)
        return response

    def _handle_get_tool(self, request, response):
        """Get tool values"""
        self.get_logger().info(f'{self.get_name()} Getting tool values')
        x, y, z, theta_x, theta_y, theta_z = get_tool_state(self._base_cyclic)
        response.x = x
        response.y = y
```

```
        response.z = z
        response.theta_x = theta_x
        response.theta_y = theta_y
        response.theta_z = theta_z
        return response

def main(args=None):
    rclpy.init(args=args)
    try:
        node = Kinova_Gen3_Interface()
        rclpy.spin(node)
    except KeyboardInterrupt:
        pass
    rclpy.shutdown()

if __name__ == '__main__':
    main()
```