# 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

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()