Assignment 1

# Christopher Budd

# 218919068

## Question 1

## A robot on a chessboard Description automatically generated

A robot on a chessboard

Description automatically generated

A white robot on a chess board

Description automatically generated

A robot on a chess board

Description automatically generated

## Question 2

### Pictures

A robot arm on a table

Description automatically generated

A robot on a table

Description automatically generated

A robot arm on a table

Description automatically generated

A robot arm on a table

Description automatically generated

A robot on a table

Description automatically generated

A robot on a table

Description automatically generated

A robot arm on a table

Description automatically generated

A robot on a table

Description automatically generated

A robot on a table

Description automatically generated

### Code

Build\_tower.py

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

# 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 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\_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 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 example\_cartesian\_action\_movement(base, base\_cyclic):

    print("Starting Cartesian action movement ...")

    action = Base\_pb2.Action()

    action.name = "Example Cartesian action movement"

    action.application\_data = ""

    feedback = base\_cyclic.RefreshFeedback()

    cartesian\_pose = action.reach\_pose.target\_pose

    cartesian\_pose.x = feedback.base.tool\_pose\_x          # (meters)

    cartesian\_pose.y = feedback.base.tool\_pose\_y - 0.1    # (meters)

    cartesian\_pose.z = feedback.base.tool\_pose\_z - 0.2    # (meters)

    cartesian\_pose.theta\_x = feedback.base.tool\_pose\_theta\_x # (degrees)

    cartesian\_pose.theta\_y = feedback.base.tool\_pose\_theta\_y # (degrees)

    cartesian\_pose.theta\_z = feedback.base.tool\_pose\_theta\_z # (degrees)

    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("Cartesian movement completed")

    else:

        print("Timeout on action notification wait")

    return finished

def main():

    sys.path.insert(0, os.path.join(os.path.dirname(\_\_file\_\_), ".."))

    import utilities

    # Parse arguments

    args = utilities.parseConnectionArguments()

    # Create connection to the device and get the router

    with utilities.DeviceConnection.createTcpConnection(args) as router:

        # Create required services

        base = BaseClient(router)

        base\_cyclic = BaseCyclicClient(router)

        success = True

        set\_gripper(base, 0.0)

        time.sleep(2)

        success &= example\_move\_to\_home\_position(base)

        #First Block

        success &= example\_angular\_action\_movement(base, [48,-58,78,0,0,0])

        set\_gripper(base, 1.0)

        time.sleep(2)

        success &= example\_angular\_action\_movement(base, [0,0,78,0,0,0])

        # Stack Spot

        success &= example\_angular\_action\_movement(base, [0,-58,78,0,0,0])

        set\_gripper(base, 0.0)

        time.sleep(2)

        # No hit

        success &= example\_angular\_action\_movement(base, [0,0,60,0,0,0])

        # 2 Block

        success &= example\_angular\_action\_movement(base, [-4,-37,126,0,0,0])

        set\_gripper(base, 1.0)

        time.sleep(2)

        success &= example\_angular\_action\_movement(base, [0,0,126,0,0,0])

        # Stack Spot

        success &= example\_angular\_action\_movement(base, [0,-52,83,0,0,0])

        set\_gripper(base, 0.0)

        time.sleep(2)

        # Reset For next

        success &= example\_angular\_action\_movement(base, [0,-40,80,0,0,0])

        # 3 Block

        success &= example\_angular\_action\_movement(base, [-9,-47,98,0,0,30])

        set\_gripper(base, 1.0)

        time.sleep(2)

        # Reset and setup

        success &= example\_angular\_action\_movement(base, [0,-40,80,0,0,0])

        # Last Drop

        success &= example\_angular\_action\_movement(base, [0,-47,84,0,0,0])

        set\_gripper(base, 0.0)

        time.sleep(2)

        return 0 if success else 1

if \_\_name\_\_ == "\_\_main\_\_":

    exit(main())

### How successful were you in this task?

I would say 70% successful as although it does work, it could be better. The tower itself was stable enough but if I had adjusted the arm more, it would have been better.

## Question 3

### Code