3D Printer 多自由度 3D 打印系统控制软件

程

序

源

码

V1.0

```
#! usr/bin/env python
import sys,subprocess
from PyQt5.QtCore import Qt,QTimer,QRunnable,QThreadPool
from PyQt5.QtGui import QPixmap,QColor
from PyQt5.QtWidgets import QApplication,QMainWindow,QPushButton,QLabel,QLineEdit
from PyQt5.QtWidgets import QWidget,QVBoxLayout,QHBoxLayout,QGridLayout, QStackedLayout
from PyQt5.QtWidgets import QTabWidget,QComboBox,QTextEdit,QStatusBar
from heat.Adr import Adr
from move.go_home import Rorigin
from path_tra import PathTra
from tra_exe import TraExe
from read_points import read_points

class MainWindow(QMainWindow):
def __init__(self):
super(QMainWindow,self).__init__()

#set the title ot the window
self.setWindowTitle("Multi-axies Printing---------CIMS/HFUT")
#set the size and the position respect to the screen
#self.resize(1000,150)
self.move(80,730)
self.tabs = QTabWidget()
self.tabs.setTabPosition(QTabWidget.North)
self.tabs.setMovable(True)

#construct the three tab widgets instance
self.spart = Set_Part()
```

```
#take the layout of the three widgets
self.tabs.addTab(self.spart," Settings and Commands ")
self.setCentralWidget(self.tabs)

#create the stutasbar
self.status = QStatusBar()
self.setStatusBar(self.status)

#combox changed siganl and plog
self.spart.prin_tem_cwidget.currentTextChanged.connect(self.set_prin_tem)
self.spart.bed_tem_cwidget.currentTextChanged.connect(self.set_bed_tem)
self.spart.fil_den_cwidget.currentTextChanged.connect(self.set_fil_den)
self.spart.prin_spe_cwidget.currentTextChanged.connect(self.set_prin_spe)
#some variables
#create the ardu
self.port_info = ""
self.adr = Adr("/dev/ttyACM0")
#create file name
self.load_file_name = ""
#initialize the tem
self.setted_prin_tem = '220'
self.setted_bed_tem = '30'
#set the simulate tag
self.sim = True
#create the threadpool instance
self.threadpool = QThreadPool()

#button signal plogged
#check the port button
self.spart.fin_dev.clicked.connect(self.check_port)
#laod file button
```

```
self.spart.load_btn.clicked.connect(self.load_file)
#emergency button
self.spart.emer_btn.clicked.connect(self.press_emer_btn)
#heat command
self.spart.heat_btn.clicked.connect(self.adr.heat_cmd)
#squash motor command
self.spart.squ_btn.clicked.connect(self.adr.motor_cmd)
#execute--pinter+arm
self.spart.exe_btn.clicked.connect(self.exe_cmd)
#connect the fake ur
self.spart.conf_btn.clicked.connect(self.fake_ur)
#connect the real ur
self.spart.conn_btn.clicked.connect(self.real_ur)
#connect the real/fake ur
self.spart.conur_btn.clicked.connect(self.con_ur)
#return to the origin
self.spart.ori_btn.clicked.connect(self.re_ori)
#start the simulation
self.spart.simu_btn.clicked.connect(self.gen_tra)
#start the execute ###
self.spart.exe_btn.clicked.connect(self.tra_exe)

#set the default self.load_file_name
self.load_file_name = "points/commands.txt"

#create Qtimer and set the interval
self.timer1 = QTimer()
self.timer1.setInterval(3000)
#use self.timer.timeout signal to connect the temprature function
self.timer1.timeout.connect(self.read_tem)
self.timer1.start()
```

```
def check_port(self):
#let the port check process run in the threadpool
workerpor = WorkerPor()
#stat the function first
self.port_info = workerpor.run()
length = len(self.port_info)
tol_info = ""
for n in range(1,length):
tol_info = tol_info + self.port_info[n]
self.spart.port_twidget.setPlainText(tol_info)
self.status.showMessage("Find the Arduino Port")

def fake_ur(self):
#change the vqlue of the self.sim to True
self.sim = True
#show the message in the status bar
self.status.showMessage("Select the Fake UR Simulation")

def real_ur(self):
#change the value of the self.sim to false
self.sim = False
#show the message in the status bar
self.status.showMessage("Select the Real UR Robot")

def con_ur(self):
#sim = True -> means open the fake robot --demo.launch
#sim = False -> means connect the real robot --start.launch
#create the WorkerCon instance to add the subprocess into the threadpool
workercon = WorkerCon(sim = self.sim)
self.threadpool.start(workercon)
```

```
#show the messaget in the status bar
self.status.showMessage("Connecting the UR robort")
def gen_tra(self):
#let the path generate process run in the threadpool
workerpath = WorkerPath()
workerpath.set_fn(self.load_file_name)
self.threadpool.start(workerpath)
#show the message in the status bar
self.status.showMessage("Generate the trajectory and stored them")
def tra_exe(self):
#let the execute trajectory process run in the threadpool
sorkerexe = WorkerExe()
self.threadpool.start(workerexe)
#show the message in the status bar
self.status.showMessage("Executue the trajectory")
def re_ori(self):
#create the WorkerRor instance to add the subprocess into the thredpool
wokerror = WorkerRor()
self.threadpool.start(wokerror)
#show the messag in the status bar
self.status.showMessage("Return to the origin")

def exe_cmd(self):
pri_t = self.setted_prin_tem
bed_t = self.setted_bed_tem
self.adr.exe_cmd(pri_t,bed_t)
self.status.showMessage("Execute..")

def set_prin_tem(self,s):
#print(s)
self.setted_prin_tem = s
```

```
self.status.showMessage("Set the print temperature: "+s)

def set_bed_tem(self,s):
#print(s)
self.setted_bed_tem = s
self.status.showMessage("Set the bed temperature: "+s)

def set_fil_den(self,s):
#print(s)
self.setted_fil_den = s
self.status.showMessage("Set the Fill Density: "+s)

def set_prin_spe(self,s):
#print(s)
self.setted_prin_spe = s
self.status.showMessage("Set the Print Speed: "+s)

def load_file(self):
self.load_file_name = self.spart.laod_file_twidget.toPlainText()
self.status.showMessage(self.load_file_name + " has been laod.")

def press_emer_btn(self):
self.status.showMessage("Press the emergency button!!!")

def read_tem(self):
line = self.adr.ser.readline()
#print(line)
if line[0:2]=='PT':
prin_tem = line[2:4]
bed_tem= line[9:11]
print("Printing temperature:"+prin_tem+"\t")
```

```
print("Bed temperature:"+bed_tem+"\n")
self.spart.prin_tem_sensor_twidget.setText(prin_tem+"/"+self.setted_prin_tem)
self.spart.bed_tem_sensor_twidget.setText(bed_tem+"/"+self.setted_bed_tem)

class Set_Part(QWidget):
def __init__(self):
super(QWidget,self).__init__()

self.fin_dev = QPushButton("Check Port")
self.load_btn = QPushButton("Load")
self.conf_btn = QPushButton("Fake UR")
self.conn_btn = QPushButton("Real UR")
self.conur_btn = QPushButton("Connect Robot")
self.ori_btn = QPushButton("Return Origin")
self.simu_btn = QPushButton("Simulate")
self.heat_btn = QPushButton("Heating/Stop")
self.squ_btn = QPushButton("Squashing/Stop")
self.exe_btn = QPushButton("Execute")
self.emer_btn = QPushButton("Emergency!!!")

#heat_btn_cliked_connected
self.heat_btn.setCheckable(True)
#squ_btn_clicked_connected
self.squ_btn.setCheckable(True)

#set the layout of the gui
self.Set_part_layout()

def Set_part_layout(self):
self.pagelayout = QHBoxLayout() #horizontal layout
self.layout1 = QVBoxLayout() #first colume
```

```
self.layout2 = QVBoxLayout() #second colume
self.layout3 = QVBoxLayout() #third colume
self.layout4 = QVBoxLayout() #fourth colume
self.layout5 = QVBoxLayout() #the fifth is fig widget
self.layout6 = QVBoxLayout() #sith colume show the temprature sensor

#add the 1-4 vlayout into the pagelayout
self.pagelayout.addLayout(self.layout1)
self.pagelayout.addLayout(self.layout2)
self.pagelayout.addLayout(self.layout3)
self.pagelayout.addLayout(self.layout4)
self.pagelayout.addLayout(self.layout5)
self.pagelayout.addLayout(self.layout6)

#set suitable margins and spacings
self.pagelayout.setContentsMargins(20,0,20,10)
self.layout1.setContentsMargins(30,0,50,0)
self.layout2.setContentsMargins(0,52,50,0)
self.layout2.setSpacing(20)
self.layout3.setContentsMargins(0,52,50,0)
self.layout3.setSpacing(20)
self.layout4.setContentsMargins(0,40,50,0)
self.layout4.setSpacing(12)
self.layout5.setContentsMargins(0,30,50,0)
self.layout6.setContentsMargins(0,30,30,20)
self.layout6.setSpacing(10)

self.colume1()
self.colume2()
self.colume3()
self.colume4()
```

```
self.colume5()
self.colume6()

self.setLayout(self.pagelayout)

def colume1(self):
#Add two labels--printing temprature--bed temprature-- to show the information
self.prin_tem_widget = QLabel("Printing Temperature:")
self.bed_tem_widget = QLabel("Bed Temperature:")

#create the combobox of the priniting temprature
self.prin_tem_cwidget = QComboBox()
#add the printing temprature range to the comboBox
num1 = []
for i in range(180,250):
num1.append(str(i))
self.prin_tem_cwidget.addItems(num1)
#creating the combobox of the bed temprature
self.bed_tem_cwidget = QComboBox()
#add the printing temprature range to the comboBox
num2 = []
for i in range(30,50):
num2.append(str(i))
self.bed_tem_cwidget.addItems(num2)

#add two labels--Full Density--Printing Speed-- to show the information
self.fil_den_widget = QLabel("Full Density:")
self.prin_spe_widget = QLabel("Print Speed:")

#create the fill density combobox
self.fil_den_cwidget = QComboBox()
```

```
self.fil_den_cwidget.addItems(["30%","50%","70%","90%"])

#create the print speed combobox
self.prin_spe_cwidget = QComboBox()
self.prin_spe_cwidget.addItems(["low","medium","high"])

#add the two groups of information and comboBox into the first clome layout
self.layout1.addWidget(self.prin_tem_widget)
self.layout1.addWidget(self.prin_tem_cwidget)
self.layout1.addWidget(self.bed_tem_widget)
self.layout1.addWidget(self.bed_tem_cwidget)
#layout1.setContentsMargins(10,10,10,10)
self.layout1.addWidget(self.fil_den_widget)
self.layout1.addWidget(self.fil_den_cwidget)
self.layout1.addWidget(self.prin_spe_widget)
self.layout1.addWidget(self.prin_spe_cwidget)
def colume2(self):
#add the lable show the instruction
port_widget = QLabel("Check the Ports:")
#add the textedit to show the device list
self.port_twidget = QTextEdit()
self.port_twidget.setReadOnly(True)
self.port_twidget.setPlaceholderText("This will show the arduino's port")
#add the two groups of information and comboBox into the first clome layout
self.layout2.addWidget(port_widget)
self.layout2.addWidget(self.port_twidget)
self.layout2.addWidget(self.fin_dev)

def colume3(self):
#add the label show the instructions
laod_file_widget = QLabel("Load the trajectory:")
```

```
#add the textedit to input the file path
self.laod_file_twidget = QTextEdit()
self.laod_file_twidget.setPlaceholderText("Please enter the absolute path of the target file..")

#add the the three things to colume3 vlayout
self.layout3.addWidget(laod_file_widget)
self.layout3.addWidget(self.laod_file_twidget)
self.layout3.addWidget(self.load_btn)

def colume4(self):
#get the button in the colume4 layout
#add a sublayout in the top
self.layout_con = QHBoxLayout()
self.layout_con.addWidget(self.conf_btn)
self.layout_con.addWidget(self.conn_btn)
self.layout4.addLayout(self.layout_con)
self.layout4.addWidget(self.conur_btn)
self.layout4.addWidget(self.ori_btn)
self.layout4.addWidget(self.simu_btn)
self.layout4.addWidget(self.heat_btn)
self.layout4.addWidget(self.squ_btn)
self.layout4.addWidget(self.exe_btn)
#set the buttons connect

def colume5(self):
#add the picture.png in the last layout
fig_widget = QLabel("logo")
fig_widget.setPixmap(QPixmap("picture.png"))
#fig_widget.setScaledContents(True)
```

```
self.layout5.addWidget(fig_widget)
self.layout5.addWidget(self.emer_btn)

def colume6(self):
#create the labels to show the information
tem_sensor_widget = QLabel("Temprature sensor")
prin_tem_sensor_widget = QLabel("Print-Tem:")
self.prin_tem_sensor_twidget = QLabel("xx"+"/"+"120")
bed_tem_sensor_widget = QLabel("Bed-Tem:")
self.bed_tem_sensor_twidget = QLabel("xx"+"/"+"40")

#add the itesms into the layout6
self.layout6.addWidget(tem_sensor_widget)
self.layout6.addWidget(prin_tem_sensor_widget)
self.layout6.addWidget(self.prin_tem_sensor_twidget)
self.layout6.addWidget(bed_tem_sensor_widget)
self.layout6.addWidget(self.bed_tem_sensor_twidget)

class WorkerPor(QRunnable):
def __init__(self):
super(QRunnable,self).__init__()
def run(self):
self.info = []
p = subprocess.Popen("python -m serial.tools.list_ports",shell = True, stdout =
subprocess.PIPE,stderr = subprocess.STDOUT)
while p.poll() is None:
self.info.append(p.stdout.readline())
return self.info
class WorkerCon(QRunnable):
def __init__(self,sim):
super(QRunnable,self).__init__()
```

```
self.sim = True
def run(self):
if self.sim:
subprocess.call(["roslaunch ur5_moveit_config demo.launch"], shell = True)
else:
subprocess.call(["roslaunch ur_printing start.launch"],shell = True)

class WorkerRor(QRunnable):
def __init__(self):
super(QRunnable,self).__init__()
def run(self):
subprocess.call(["rosrun ur_printing go_home.py"], shell = True)

class WorkerPath(QRunnable):
def __init__(self):
super(QRunnable,self).__init__()
def set_fn(self,filename):
self.filename = filename
def run(self):
subprocess.call(["rosrun ur_printing path_tra.py "+self.filename],shell = True)

class WorkerExe(QRunnable):
def __init__(self):
super(QRunnable,self).__init__()
def run(self):
subproecss.call(["rosrun ur_printing tra_exe.py"],shell = True)


def main():
app = QApplication(sys.argv)
window = MainWindow()
window.show()
```

```
window.read_tem

app.exec_()

if __name__ == "__main__":
main()



# -*- coding: utf-8 -*-

# Form implementation generated from reading ui file 'ur_printing.ui'
#
# Created by: PyQt5 UI code generator 5.15.3
#
# WARNING: Any manual changes made to this file will be lost when pyuic5 is
# run again. Do not edit this file unless you know what you are doing.

from PyQt5 import QtCore, QtGui, QtWidgets


class Ui_Form(object):
def setupUi(self, Form):
Form.setObjectName("Form")
Form.resize(502, 464)
self.verticalLayout_4 = QtWidgets.QVBoxLayout(Form)
self.verticalLayout_4.setObjectName("verticalLayout_4")
self.horizontalLayout_2 = QtWidgets.QHBoxLayout()
self.horizontalLayout_2.setContentsMargins(20, 45, 20, 87)
self.horizontalLayout_2.setObjectName("horizontalLayout_2")
self.verticalLayout = QtWidgets.QVBoxLayout()
self.verticalLayout.setObjectName("verticalLayout")
```

```
self.horizontalLayout = QtWidgets.QHBoxLayout()
self.horizontalLayout.setObjectName("horizontalLayout")
self.label = QtWidgets.QLabel(Form)
self.label.setObjectName("label")
self.horizontalLayout.addWidget(self.label)
self.label_2 = QtWidgets.QLabel(Form)
self.label_2.setObjectName("label_2")
self.horizontalLayout.addWidget(self.label_2)
self.verticalLayout.addLayout(self.horizontalLayout)
self.textBrowser = QtWidgets.QTextBrowser(Form)
self.textBrowser.setObjectName("textBrowser")
self.verticalLayout.addWidget(self.textBrowser)
self.horizontalLayout_2.addLayout(self.verticalLayout)
spacerItem = QtWidgets.QSpacerItem(40, 20, QtWidgets.QSizePolicy.Expanding, QtWidgets.QSizePolicy.Minimum)
self.horizontalLayout_2.addItem(spacerItem)
self.verticalLayout_2 = QtWidgets.QVBoxLayout()
self.verticalLayout_2.setObjectName("verticalLayout_2")
self.pushButton = QtWidgets.QPushButton(Form)
self.pushButton.setObjectName("pushButton")
self.verticalLayout_2.addWidget(self.pushButton)
self.pushButton_2 = QtWidgets.QPushButton(Form)
self.pushButton_2.setObjectName("pushButton_2")
self.verticalLayout_2.addWidget(self.pushButton_2)
self.pushButton_3 = QtWidgets.QPushButton(Form)
self.pushButton_3.setObjectName("pushButton_3")
self.verticalLayout_2.addWidget(self.pushButton_3)
self.horizontalLayout_2.addLayout(self.verticalLayout_2)
self.verticalLayout_4.addLayout(self.horizontalLayout_2)
self.horizontalLayout_3 = QtWidgets.QHBoxLayout()
self.horizontalLayout_3.setContentsMargins(20, 20, 20, 20)
```

```
self.horizontalLayout_3.setObjectName("horizontalLayout_3")
self.horizontalLayout_4 = QtWidgets.QHBoxLayout()
self.horizontalLayout_4.setObjectName("horizontalLayout_4")
self.horizontalLayout_5 = QtWidgets.QHBoxLayout()
self.horizontalLayout_5.setObjectName("horizontalLayout_5")
self.openGLWidget = QtWidgets.QOpenGLWidget(Form)
self.openGLWidget.setEnabled(True)
self.openGLWidget.setObjectName("openGLWidget")
self.horizontalLayout_5.addWidget(self.openGLWidget)
self.horizontalLayout_4.addLayout(self.horizontalLayout_5)
self.horizontalLayout_3.addLayout(self.horizontalLayout_4)
self.verticalLayout_3 = QtWidgets.QVBoxLayout()
self.verticalLayout_3.setContentsMargins(134, -1, -1, -1)
self.verticalLayout_3.setObjectName("verticalLayout_3")
self.pushButton_4 = QtWidgets.QPushButton(Form)
self.pushButton_4.setObjectName("pushButton_4")
self.verticalLayout_3.addWidget(self.pushButton_4)
self.pushButton_5 = QtWidgets.QPushButton(Form)
self.pushButton_5.setObjectName("pushButton_5")
self.verticalLayout_3.addWidget(self.pushButton_5)
self.pushButton_6 = QtWidgets.QPushButton(Form)
self.pushButton_6.setObjectName("pushButton_6")
self.verticalLayout_3.addWidget(self.pushButton_6)
self.pushButton_7 = QtWidgets.QPushButton(Form)
self.pushButton_7.setObjectName("pushButton_7")
self.verticalLayout_3.addWidget(self.pushButton_7)
self.horizontalLayout_3.addLayout(self.verticalLayout_3)
self.verticalLayout_4.addLayout(self.horizontalLayout_3)
self.progressBar = QtWidgets.QProgressBar(Form)
self.progressBar.setProperty("value", 24)
self.progressBar.setObjectName("progressBar")
```

```
self.verticalLayout_4.addWidget(self.progressBar)

self.retranslateUi(Form)
QtCore.QMetaObject.connectSlotsByName(Form)

def retranslateUi(self, Form):
_translate = QtCore.QCoreApplication.translate
Form.setWindowTitle(_translate("Form", "HFUT-6DOF-Printing"))
self.label.setText(_translate("Form", "喷头温度"))
self.label_2.setText(_translate("Form", "热床温度"))
self.pushButton.setText(_translate("Form", "加热"))
self.pushButton_2.setText(_translate("Form", "电机转动"))
self.pushButton_3.setText(_translate("Form", "电机停止"))
self.pushButton_4.setText(_translate("Form", "路径规划"))
self.pushButton_5.setText(_translate("Form", "回零操作"))
self.pushButton_6.setText(_translate("Form", "开始打印"))
self.pushButton_7.setText(_translate("Form", "中断急停"))



#! /usr/bin/env python

#this file is used to figure out the problem of
#Robot arm forward kinematics
#here we need the class and functions defined in matrices.py file
#We can use matrices by using this file

from matrices import Matrix,mult
from math import sin,cos,pi

class Transfrom():
def __init__(self):
```

```
#initialize the 6 Trasform matrices firstly
self.T0_1 = Matrix(4,4)
self.T1_2 = Matrix(4,4)
self.T2_3 = Matrix(4,4)
self.T3_4 = Matrix(4,4)
self.T4_5 = Matrix(4,4)
self.T5_6 = Matrix(4,4)
#create the D-H patam of 6DOF arm, initialize the default value to 0
#arf parameter
self.arf0 = 0
self.arf1 = 0
self.arf2 = 0
self.arf3 = 0
self.arf4 = 0
self.arf5 = 0
#a parameter
self.a0 = 0
self.a1 = 0
self.a2 = 0
self.a3 = 0
self.a4 = 0
self.a5 = 0
#d parameter
self.d1 = 0
self.d2 = 0
self.d3 = 0
self.d4 = 0
self.d5 = 0
self.d6 = 0
#theta parameter
self.theta1 = 0
```

```
self.theta2 = 0
self.theta3 = 0
self.theta4 = 0
self.theta5 = 0
self.theta6 = 0

def set_ur_dh_param(self):
#here we use the UR robot, input its parameter
#assign the d paramenter, remember the default value is 0
self.d1 = 89.459
self.d4 = 109.15
self.d5 = 94.56
self.d6 = 82.3
#assign the a value
self.a1 = -425
self.a2 = -392.25
#assign the arf
self.arf0 = pi/2
self.arf3 = pi/2
self.arf4 = -pi/2
def get_t(self):
#use the dh tablet to create the corresponding matrices
self.T0_1 = get_dh_matrix(self.arf0,self.a0,self.d1,self.theta1)
self.T1_2 = get_dh_matrix(self.arf1,self.a1,self.d2,self.theta2)
self.T2_3 = get_dh_matrix(self.arf2,self.a2,self.d3,self.theta3)
self.T3_4 = get_dh_matrix(self.arf3,self.a3,self.d4,self.theta4)
self.T4_5 = get_dh_matrix(self.arf4,self.a4,self.d5,self.theta5)
self.T5_6 = get_dh_matrix(self.arf5,self.a5,self.d6,self.theta6)
def get_T06(self):
#caluculate the total Transform matrix from 0-6:
#the sequence is very important
```

```
self.T4_6 = mult(self.T4_5,self.T5_6)
#pri_t(temp1)
self.T3_6 = mult(self.T3_4,self.T4_6)
#pri_t(temp2)
self.T2_6 = mult(self.T2_3,self.T3_6)
#pri_t(temp3)
self.T1_6 = mult(self.T1_2,self.T2_6)
#pri_t(temp4)
self.T0_6 = mult(self.T0_1,self.T1_6)
#pri_t(temp5)
def show_Ts(self):
print("T0_1:")
pri_t(self.T0_1)
print("T1_2:")
pri_t(self.T1_2)
print("T2_3:")
pri_t(self.T2_3)
print("T3_4:")
pri_t(self.T3_4)
print("T4_5:")
pri_t(self.T4_5)
print("T5_6:")
pri_t(self.T5_6)

def update_joints(self,joint_v):
#this method ued to update the joint parameters:theta1-6
self.theta1 = joint_v[0]
self.theta2 = joint_v[1]
self.theta3 = joint_v[2]
self.theta4 = joint_v[3]
self.theta5 = joint_v[4]
```

```
self.theta6 = joint_v[5]

def joint2end(self,joint_v):
#set the attribute according to the sequence
self.set_ur_dh_param()
self.update_joints(joint_v)
self.get_t()
self.get_T06()
#print(self.T0_6.val)
return self.T0_6.val

def get_dh_matrix(arf,a,d,theta):
#--(arfi-1)--(ai-1)--(di)--(thetai)
#The first row of matrix
t11 = cos(theta)
t12 = -sin(theta)
t13 = 0
t14 = a
t_row1 = [t11,t12,t13,t14]

#The second row of matrix
t21 = sin(theta)*cos(arf)
t22 = cos(theta)*cos(arf)
t23 = -sin(arf)
t24 = -sin(arf)*d
t_row2 = [t21,t22,t23,t24]

#The third row of matrix
t31 = sin(theta)*sin(arf)
t32 = cos(theta)*sin(arf)
t33 = cos(arf)
```

```
t34 = cos(arf)*d
t_row3 = [t31,t32,t33,t34]

#The fourth row of matrix
t41 = 0
t42 = 0
t43 = 0
t44 = 1
t_row4 = [t41,t42,t43,t44]

#combine the rows together
tt = [t_row1,t_row2,t_row3,t_row4]

#create the class Matrix
#assign the value attribute and return the substance
A = Matrix(4,4)
A.val = tt
return A

def pri_t(A):
print(A.val[0])
print(A.val[1])
print(A.val[2])
print(A.val[3])

if __name__ == "__main__":
#create the Transform to use the method
trans = Transfrom()
trans.joint2end([0.1,0.1,0.1,0.1,0.1,0.1])
#pri_t(trans.T0_6)
```

```
#! /usr/bin/env python

def jaco_end(j,w):
#assign the w vlaue to w1-6
w1 = w[0]
w2 = w[1]
w3 = w[2]
w4 = w[3]
w5 = w[4]
w6 = w[5]
#assign the jacobian matrix
#the first row
j11 = j[0][0]
j12 = j[0][1]
j13 = j[0][2]
j14 = j[0][3]
j15 = j[0][4]
j16 = j[0][5]
#the second row
j21 = j[1][0]
j22 = j[1][1]
j23 = j[1][2]
j24 = j[1][3]
j25 = j[1][4]
j26 = j[1][5]
#the third row
j31 = j[2][0]
j32 = j[2][1]
j33 = j[2][2]
```

```
j34 = j[2][3]
j35 = j[2][4]
j36 = j[2][5]
#the fourth row
j41 = j[3][0]
j42 = j[3][1]
j43 = j[3][2]
j44 = j[3][3]
j45 = j[3][4]
j46 = j[3][5]
#the fifth row
j51 = j[4][0]
j52 = j[4][1]
j53 = j[4][2]
j54 = j[4][3]
j55 = j[4][4]
j56 = j[4][5]
#the sixth row
j61 = j[5][0]
j62 = j[5][1]
j63 = j[5][2]
j64 = j[5][3]
j65 = j[5][4]
j66 = j[5][5]

#caculate the result
vx = j11*w1+j12*w2+j13*w3+j14*w4+j15*w5+j16*w6
vy = j21*w1+j22*w2+j23*w3+j24*w4+j25*w5+j26*w6
vz = j31*w1+j32*w2+j33*w3+j34*w4+j35*w5+j36*w6
wx = j41*w1+j42*w2+j43*w3+j44*w4+j45*w5+j46*w6
wy = j51*w1+j52*w2+j53*w3+j54*w4+j55*w5+j56*w6
```

```
wz = j61*w1+j62*w2+j63*w3+j64*w4+j65*w5+j66*w6
#return the p list
p =[vx,vy,vz,wx,wy,wz]
return p

#! /usr/bin/env python

#this is the classed and functions used to deal with the matrices

class Matrix():
def __init__(self,row,col):
#defualt number of row is one
self.row = row
#default number of colume is one
self.col = col
#default elements is all zeros
self.val = zeros(self.row,self.col)
def size(self):
#this method return the row and col number of the matrix class
return self.row,self.col
def set(self,n,m,val):
#this is going to set the concrete value
temp = self.val.pop(n)
sub = []
for i in range(0,self.col):
t_v = temp[i]
if (i==m):
t_v = val
sub.append(t_v)
#print(sub)
self.val.insert(n,sub)
```

```
def zeros(n,m):
#this function is to create a dimantional list (whose elements is all zero
row_zero = []
for i in range(0,m):
row_zero.append(0)
tol_zero = []
for k in range(0,n):
tol_zero.append(row_zero)
return tol_zero

def mult(A,B):
C = Matrix(A.col,B.row)
if(A.col == B.row):
#the former matrix's colume number
#must be equal to the latter matrix's row number
num_i = A.row
num_j = B.col
#temp value of the mulitiply
T_val = []
TT_val = []
for i in range(0,num_i):
#catch the corresponding row and colume
list_1 = A.val[i]
T_val = []
for j in range(0,num_j):
list_2 = []
for k in range(0,B.row):
#get the colume of B
t_v = B.val[k][j]
list_2.append(t_v)
#use the defined function to do list multiplication
```

```
temp = list_mult_list(list_1,list_2)
#add the element
T_val.append(temp)
#combine the rows acquired above together
TT_val.append(T_val)
#create a new Matrix substance
C = Matrix(num_i,num_j)
#assign the resulte to the matrix's value attribute
C.val = TT_val
return C
else:
print("The colume and row are not suitable!")

def list_mult_list(list1,list2):
res = 0
if(len(list1)==len(list2)):
#do the multiply with the two lists
num = len(list1)
#print(list1)
#print(list2)
for i in range(0,num):
#create the temp variable
#multiply the element one by one
temp = list1[i]*list2[i]
res = res+temp
else:
print("The length of list is not suitable!")
return res

if __name__ =="__main__":
#used to check
```

```
A = Matrix(3,3)
A.val = [[1,2,3],[4,5,6],[7,8,9]]
B = Matrix(3,3)
B.val = [[1,3,1],[2,5,6],[1,8,9]]
C = mult(A,B)
print(C.val)

#! /usr/bin/env python
import os
import rospy,sys
import moveit_commander
from moveit_commander import MoveGroupCommander
from geometry_msgs.msg import Pose
from copy import deepcopy
from read_points import read_points

#x_s,y_s = read_points()

class PathTra:
def __init__(self,filename):
#filename define
self.filename = filename
#initializing the API of move_group
moveit_commander.roscpp_initialize(sys.argv)
#node initializeation
rospy.init_node('moveit_cartesian_demo')
#choose to use cartersian
cartersian = rospy.get_param('~cartesian',True)
#initializeation the manipulator group in ur5
self.arm = MoveGroupCommander('manipulator')
#allow the replann
```

```
self.arm.allow_replanning(True)
#set the referance frame
self.arm.set_pose_reference_frame('base_link')
#set the tolerance
self.arm.set_goal_position_tolerance(0.002)
self.arm.set_goal_orientation_tolerance(0.01)
#get the name of the end_effector
end_effector_link = self.arm.get_end_effector_link()
#get the current position and orientation
self.start_pose = self.arm.get_current_pose(end_effector_link).pose

#add the attributes x_s,y_s
self.x_s,self.y_s = read_points(self.filename)
self.num = 0

def plan_tra(self):
#initializing the waypoints
waypoints=[]
#add the start pose
waypoints.append(self.start_pose)
self.num = len(self.x_s)
#add the end_effector pose
for n in range(1,2):
for i in range(self.num-1,0,-1):
wpose = deepcopy(self.start_pose)
wpose.position.x += self.x_s[i]
wpose.position.y += self.y_s[i]
wpose.position.z += -0.0012*(n-1)
#print x_s[i]
#print y_s[i]
waypoints.append(wpose)
```

```
for i in range(0,self.num):
wpose = deepcopy(self.start_pose)
wpose.position.x += self.x_s[i]
wpose.position.y += self.y_s[i]
wpose.position.z += -0.0012*(n-1)-0.0005*1.2
#print x_s[i]
#print y_s[i]
waypoints.append(wpose)
#some parameters
fraction = 0.0
maxtries = 200
attempts = 0
#current_state
self.arm.set_start_state_to_current_state()
#try to plan a path passing all the waypionts
while fraction < 1.0 and attempts <maxtries:
(plan, fraction) = self.arm.compute_cartesian_path(waypoints,0.0008,0.0,True)
#eff-step = 0.00005 is the best
#recod the times
attempts += 1
if fraction == 1.0:
rospy.loginfo("Path computed successfully,Store the trajectory")
self.tra = plan.joint_trajectory
#get the number of the points
self.num = len(self.tra.points)

#use the method store_stra to write the trajectory file
self.store_tra()
rospy.loginfo("trajectory's storation complete.")
else:
rospy.loginfo("Failes")
```

```
def store_tra(self):
#set the file name------"trajectory.txt"
filename = "trajectory.txt"
#open the file in the 'write' way
with open(filename,'w') as f_obj:
for i in range(0,self.num):
#write in the points in the planed trajectory
#write the index
f_obj.write(str(i))
f_obj.write("\n")
#write the positions
f_obj.write(str(self.tra.points[i].positions))
f_obj.write("\n")
#write the velocities
f_obj.write(str(self.tra.points[i].velocities))
f_obj.write("\n")
#write the acceleraties
f_obj.write(str(self.tra.points[i].accelerations))
f_obj.write("\n")
#write the time_from_start
f_obj.write(str(self.tra.points[i].time_from_start))
f_obj.write("\n")
#exit
moveit_commander.roscpp_shutdown()
moveit_commander.os._exit(0)

if __name__ =="__main__":
filename = sys.argv[1]
print(filename)
pathtra = PathTra(filename)
pathtra.plan_tra()
```

```
pathtra.store_tra()

#! /usr/bin/env python

import rospy
from std_msgs.msg import String
from std_msgs.msg import Float64MultiArray

class PlatPub():
def __init__(self):
#create the publisher
pub = rospy.Publisher('plat_states',Float64MultiArray,queue_size=5)
self.msg = Float64MultiArray()
#set the defalut platform_states equal to this
self.msg.data = [0.0,0.0,0.0,0.0,0.0,0.0]
#initialize the node
rospy.init_node('plat_states_pulisher',anonymous = True)
#set the publish rate
rate = rospy.Rate(5)
#publishe the data
while not rospy.is_shutdown():
pub.publish(self.msg)
rate.sleep()

def update(self,instant_value):
#update the platform_states by this method
self.plat_states = instant_value
#print some notion
print("The plat_states get is:")
print(self.plat_states)
#assign the valuet to Float64MultiArray() instance
```

```
self.msg.data = self.plat_states

if __name__ == "__main__":
try:
platpub = PlatPub()
except rospy.ROSInterruptException:
pass

#!usr/bin/env python
#This is the code used to load the commands.txt file
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import numpy as np

def read_points(filename):
#define the filename
#filename = "points/points.txt"
#open the file as a file object
with open(filename) as f_obj:

#read the file line by line
lines = f_obj.readlines()

#get the lines number and set a index
num_lines = len(lines)
#define the whole data 3-dimontional array piles
piles = np.array([])

# define points_ox,points_oy,points_oz
points_ox = []
points_oy = []
```

```
points_oz = []
points_ov = []
#define the thetas list
thetas = []


for line in lines[1:num_lines]:
#try to write the code to settle the data
if line[0]=='L':
theta = float(line[16:21])
thetas.append(-theta)
pile = [[theta],points_ox,points_oy,points_oz,points_ov]
#add this pile to the total piles data
piles = np.append(piles,pile)

#clear the x,y,z values
points_ox = []
points_oy = []
points_oz = []
points_ov = []
#clear the theta values
#thetas = []

else:
x = float(line[3:11])
y = float(line[15:23])
z = float(line[27:35])
v = float(line[39:47])

print(x)
print(y)
print(z)
```

```
print(v)
# negative !!
points_ox.append(x)
points_oy.append(y)
points_oz.append(z)
points_ov.append(v)

#actually read the other parameters is the same operation
return piles

if __name__ == '__main__':
data = read_points("points/path.txt")

print(data)

fig = plt.figure()
ax = Axes3D(fig) # Create a figure containing a single axes.
ax.set_title('Path of the profile') #set the title
ax.set_xlabel('x position /mm') #set the x label
ax.set_ylabel('y position /mm') #set the y label
ax.set_zlabel('z position /mm') #set the z label
ax.set_zticks([0,0.005,0.1])
levels = len(data)/5
for i in range(0,levels):
ax.plot3D(data[5*i+1],data[5*i+2],data[5*i+3])
plt.axis('equal')
plt.show()

#!/usr/bin/env python
# -*- coding: utf-8 -*-
import sys
```

```
import rospy
import numpy as np
from ur_kin_py.kin import Kinematics as kin
from trajectory_msgs.msg import JointTrajectory,JointTrajectoryPoint
from read_points import read_points
from read_points_2 import read_points
from funs import cal_distance
import matplotlib.pyplot as plt

#Set the ur kinematics calculator
ur5_kin = kin('ur5')

#Load the point data into here
data = read_points("points/whatever_2.txt")
levels = len(data)/5

#intialize the parameters of the arm's motion
tra_pos = []
tra_pos_time = []
time = 0
ori_x = 0.00
ori_y = 0.27
ori_z = 0.55
#ori_z = 0.5560
save = True
val = 0
#tra_pos_time = [0.004/velocity]

for n in range(0,levels,1):
#make a motion in a lever
#assigin the values into the temp varibles
```

```
index = 5*n
theta_deg = data[index][0]
x_s = data[index+1]
y_s = data[index+2]
z_s = data[index+3]
v_s = data[index+4]*5

#print(x_s)
#make rotation first
#make this angle negative
theta_rad = np.deg2rad(theta_deg)
cos_theta = np.cos(theta_rad)
sin_theta = np.sin(theta_rad)
R = np.matrix([[-sin_theta,0.,cos_theta,ori_x+x_s[0]],
[0.,-1.,0.,ori_y+y_s[0]],
[cos_theta,0.,sin_theta,ori_z+z_s[0]],
[0.,0.,0.,1.]])
#Use the inverse function slove the IK calculation
joint_angles = ur5_kin.inverse(R)
if (len(tra_pos) > 0):
save,val = cal_distance(joint_angles,tra_pos[-1],1.2)
print(val)
tra_pos.append(joint_angles)
time = time + 0.5
tra_pos_time.append(time)

num = len(x_s)
#print(num)
#Initialize the parameter distance and time
distance = 0
time = time + 0.5
```

```
tra_pos_time.append(time)
#Use velocity to calculate corresponding start_time from point_0 to point_n
for i in range(1,num):
distance = np.sqrt((x_s[i]-x_s[i-1])**2+(y_s[i]-y_s[i-1])**2)
if distance/(0.5*(v_s[i]+v_s[i-1])) < 0.0:
print(distance/(0.5*(v_s[i]+v_s[i-1])))
print("\n")
time = time + distance/(0.5*(v_s[i]+v_s[i-1]))
tra_pos_time.append(time)


for i in range(0,num):
delta_x = x_s[i]
delta_y = y_s[i]
delta_z = z_s[i]
R = np.matrix([[-sin_theta,0.,cos_theta,ori_x+delta_x],
[0.,-1.,0.,ori_y+delta_y],
[cos_theta,0.,sin_theta,ori_z+delta_z],
[0.,0.,0.,1.]])
#Use the inverse function slove the IK calculation
joint_angles = ur5_kin.inverse(R)
save,val = cal_distance(joint_angles,tra_pos[-1],1.2)
print(val)
tra_pos.append(joint_angles)

#The trajectory position and time_form_start param have got

print(tra_pos)
num = len(tra_pos_time)
print(tra_pos_time)
print(len(tra_pos))
print(len(tra_pos_time))
```

```
for i in range(1,num):
if not (tra_pos_time[i]>tra_pos_time[i-1]):
print("%d: %f"%(i-1,tra_pos_time[i-1]))
print("%d: %f"%(i,tra_pos_time[i]))
#print("\n")
#n = range(0,len(tra_pos_time))
#plt.plot(n,tra_pos_time)
#plt.show()


def perform_trajectory():
rospy.init_node('arm_trajectory_publisher')
control_name = '/arm_controller/command'
trajectory_publisher = rospy.Publisher(control_name,JointTrajectory,queue_size=20)
arm_joints =
['shoulder_pan_joint','shoulder_lift_joint','elbow_joint','wrist_1_joint','wrist_2_joint','wrist_3_
joint']
arm_trajectory = JointTrajectory()
arm_trajectory.joint_names = arm_joints
num_1= len(tra_pos_time)
for n in range(0,num_1):
point = JointTrajectoryPoint()
point.positions = tra_pos[n]
point.velocities = [0.0 for i in arm_joints]
point.accelerations = [0.0 for i in arm_joints]
point.time_from_start = rospy.Duration(tra_pos_time[n])
arm_trajectory.points.append(point)

rospy.sleep(1)
trajectory_publisher.publish(arm_trajectory)

if __name__=="__main__":
```

```
perform_trajectory()



#! usr/bin/env python
import sys,rospy,subprocess
import moveit_commander
from moveit_commander import MoveGroupCommander
from geometry_msgs.msg import Pose
from sensor_msgs.msg import JointState
from std_msgs.msg import Float64MultiArray
from copy import deepcopy
from PyQt5.QtCore import Qt,QTimer,QRunnable,QThreadPool
from PyQt5.QtGui import QPixmap,QColor
from PyQt5.QtWidgets import QApplication,QMainWindow,QPushButton,QLabel
from PyQt5.QtWidgets import
QWidget,QVBoxLayout,QHBoxLayout,QGridLayout,QStackedLayout
from PyQt5.QtWidgets import QTabWidget,QComboBox,QTextEdit,QStatusBar
from jaco_end import jaco_end

class MainWindow(QMainWindow):
def __init__(self):
super(QMainWindow,self).__init__()

#set the title ot the window
self.setWindowTitle("Multi-axies Printing Speed Monitor---------CIMS/HFUT")

#set the size and the position respect to the screen
self.resize(150,350)
self.move(0,0)
#set the QtabWidget
self.tabs = QTabWidget()
```

```
self.tabs.setTabPosition(QTabWidget.North)
self.tabs.setMovable(True)
#create the instance of the calss
self.apart = Arm_Part()
self.ppart = Plat_Part()

#take the layout of the three widgets
self.tabs.addTab(self.apart," Robot Arm ")
self.tabs.addTab(self.ppart," Muti-axies Platform ")
self.setCentralWidget(self.tabs)

#create the stutasbar
self.status = QStatusBar()
self.setStatusBar(self.status)
#set the siganl and the plog
self.apart.mon_btn.clicked.connect(self.change_act)
self.ppart.mon_btn.clicked.connect(self.change_act)
self.apart.pic_btn.clicked.connect(self.draw_graph1)
self.ppart.picv_btn.clicked.connect(self.draw_graph2)
self.ppart.picw_btn.clicked.connect(self.draw_graph3)
#set the graph button
#j_w is just a shell command
#plat_v other need a publisher and a shell command

#set the threadpool
self.threadpool = QThreadPool()
#create the active tag
self.active = False

#set some instance
self.s_monitor = SpeMon()
```

```
#create Qtimer and set the interval
self.timer1 = QTimer()
self.timer1.setInterval(10)
#use self.timer.timeout signal to connect the temprature function
self.timer1.timeout.connect(self.update_para)
self.timer1.start()

def draw_graph1(self):
#use the workerdrawgraph first
workerd_1 = WorkerDrawGraph1()
self.threadpool.start(workerd_1)
print("Draw the first graph")
self.status.showMessage("Draw the graph one")

def draw_graph2(self):
workerd_2 =WorkerDrawGraph2()
self.threadpool.start(workerd_2)
print("Draw the second graph")
self.status.showMessage("Draw the graph two")

def draw_graph3(self):
workerd_3 = WorkerDrwaGraph3()
self.threadpool.start(workerd_3)
print("Draw the third frapg")
self.status.showMessage("Draw the graph 3")

def change_act(self,en):
if en:
self.active = True
else:
```

```
self.active = False
def update_para(self):
if self.active:
#get the prarameter needed
self.w_vel()
self.s_monitor.get_date(self.w_v)
#use the parameter to set the Qlabel
#set the joint position 1-6
self.apart.joint_1p.setText(str(self.s_monitor.j_v[0]))
self.apart.joint_2p.setText(str(self.s_monitor.j_v[1]))
self.apart.joint_3p.setText(str(self.s_monitor.j_v[2]))
self.apart.joint_4p.setText(str(self.s_monitor.j_v[3]))
self.apart.joint_5p.setText(str(self.s_monitor.j_v[4]))
self.apart.joint_6p.setText(str(self.s_monitor.j_v[5]))
#set the joint angular velocity 1-6
self.apart.joint_1v.setText(str(self.w_v[0]))
self.apart.joint_2v.setText(str(self.w_v[1]))
self.apart.joint_3v.setText(str(self.w_v[2]))
self.apart.joint_4v.setText(str(self.w_v[3]))
self.apart.joint_5v.setText(str(self.w_v[4]))
self.apart.joint_6v.setText(str(self.w_v[5]))

#set the endeffector velocity
self.ppart.plat_vx.setText(str(self.s_monitor.e_v[0]))
self.ppart.plat_vy.setText(str(self.s_monitor.e_v[1]))
self.ppart.plat_vz.setText(str(self.s_monitor.e_v[2]))
self.ppart.plat_wx.setText(str(self.s_monitor.e_v[3]))
self.ppart.plat_wy.setText(str(self.s_monitor.e_v[4]))
self.ppart.plat_wz.setText(str(self.s_monitor.e_v[5]))
else:
pass
```

```
def w_vel(self):
#let the port check process run in the threadpool
workerang = WorkerAng()
#stat the function first
self.pr_info = workerang.run()
print("The info get is")
print(self.pr_info)
self.w_v = [0,0,0,0,0,0]
#select the valuable information of w
w = self.pr_info[0]
print(w)
if len(w) > 60:
#the w is not none
self.w_v = self.strl2floatl(w)
else:
#the w is none
self.w_v = [0,0,0,0,0,0]
self.status.showMessage("Get the angular velocity")

def strl2floatl(self,str):
kat = []
length = len(str)
for i in range(1,length):
if str[i] == ',':
kat.append(i)
#find the str number
s_num1 = str[1:(kat[0]-1)]
s_num2 = str[(kat[0]+1):(kat[1]-1)]
s_num3 = str[(kat[1]+1):(kat[2]-1)]
s_num4 = str[(kat[2]+1):(kat[3]-1)]
s_num5 = str[(kat[3]+1):(kat[4]-1)]
```

```
s_num6 = str[(kat[4]+1):(length-2)]
#convert the string to float
num1 = float(s_num1)
num2 = float(s_num2)
num3 = float(s_num3)
num4 = float(s_num4)
num5 = float(s_num5)
num6 = float(s_num6)
#comine the six float into a list and retuen
num = [num1,num2,num3,num4,num5,num6]

return num

class SpeMon():
def __init__(self):
#initialization the node
rospy.init_node('speed_monitor',anonymous = True)
#use the arm group manipulator
self.arm = MoveGroupCommander('manipulator')
#create the publisher
self.pub = rospy.Publisher('plat_states',Float64MultiArray,queue_size=5)
self.msg = Float64MultiArray()
def get_date(self,w_value):
self.j_v = self.arm.get_current_joint_values()
self.c_p = self.arm.get_current_pose()
self.position = self.c_p.pose.position
self.orientation = self.c_p.pose.orientation
self.jacobian = self.arm.get_jacobian_matrix(self.j_v)
#here we need the jacobian and the angular value
#so this need to get the information of the w vector
self.e_v = jaco_end(self.jacobian,w_value)
```

```
#update the platform_states by this method
self.msg.data = self.e_v
#print some notion
print("The plat_states get is:")
print(self.msg.data)
#assign the valuet to Float64MultiArray() instance
self.pub.publish(self.msg)

print("Jacobian")
print(self.jacobian)
print("joint_value")
print(self.j_v)
print("Position")
print(self.position)
print("Orientation")
print(self.orientation)
print("Endeffector linear velocity")
print(self.e_v[0:3])
print("Endeffector angular velocity")
print(self.e_v[3:6])
class Arm_Part(QWidget):
def __init__(self):
super(QWidget,self).__init__()
self.layout1 = QGridLayout()
#se the contantsmargin
self.layout1.setContentsMargins(30,20,20,20)
#create two buttons and the plogged function
self.mon_btn = QPushButton(" Monitor ")
self.pic_btn = QPushButton(" Graph ")
#set the monitor button to be true
self.mon_btn.setCheckable(True)
```

```
#use the grid layout
self.Grid_1()
self.setLayout(self.layout1)

def Grid_1(self):
#create the information labels
#headline labels
joint_name = QLabel("Joint name")
joint_pos = QLabel("Position")
joint_vel = QLabel("Velocity")

#joint1-6 name labels
self.joint_1n = QLabel("elbow_joint")
self.joint_2n = QLabel("shoulder_lift_joint")
self.joint_3n = QLabel("shoulder_pan_joint")
self.joint_4n = QLabel("wrist_1_joint")
self.joint_5n = QLabel("wrist_2_joint")
self.joint_6n = QLabel("wrist_3_joint")

#joint1-6 positions labels
self.joint_1p = QLabel("joint_1p")
self.joint_2p = QLabel("joint_2p")
self.joint_3p = QLabel("joint_3p")
self.joint_4p = QLabel("joint_4p")
self.joint_5p = QLabel("joint_5p")
self.joint_6p = QLabel("joint_6p")

#joint1-6 velocities labels
self.joint_1v = QLabel("joint_1v")
self.joint_2v = QLabel("joint_2v")
self.joint_3v = QLabel("joint_3v")
```

```
self.joint_4v = QLabel("joint_4v")
self.joint_5v = QLabel("joint_5v")
self.joint_6v = QLabel("joint_6v")

#add the labels into the grid_1 layout
self.layout1.addWidget(joint_name,0,0)
self.layout1.addWidget(joint_pos,1,0)
self.layout1.addWidget(joint_vel,2,0)
#add two button here

#add the joint1-6 names labels into grid_1 layout
self.layout1.addWidget(self.joint_1n,0,1)
self.layout1.addWidget(self.joint_2n,0,2)
self.layout1.addWidget(self.joint_3n,0,3)
self.layout1.addWidget(self.joint_4n,0,4)
self.layout1.addWidget(self.joint_5n,0,5)
self.layout1.addWidget(self.joint_6n,0,6)
#add the joint1-6 positions labels into grid_1 layout
self.layout1.addWidget(self.joint_1p,1,1)
self.layout1.addWidget(self.joint_2p,1,2)
self.layout1.addWidget(self.joint_3p,1,3)
self.layout1.addWidget(self.joint_4p,1,4)
self.layout1.addWidget(self.joint_5p,1,5)
self.layout1.addWidget(self.joint_6p,1,6)

#add the joint1-6 velocities labels into grid_1 layout
self.layout1.addWidget(self.joint_1v,2,1)
self.layout1.addWidget(self.joint_2v,2,2)
self.layout1.addWidget(self.joint_3v,2,3)
self.layout1.addWidget(self.joint_4v,2,4)
self.layout1.addWidget(self.joint_5v,2,5)
```

```
self.layout1.addWidget(self.joint_6v,2,6)
#add the two button into the row3
self.layout1.addWidget(self.mon_btn,3,2)
self.layout1.addWidget(self.pic_btn,3,5)

#def modify the joint1-6 pos function
#def modify the joint1-6 vel function
#def modify the joint1-6 acc function

class Plat_Part(QWidget):
def __init__(self):
super(QWidget,self).__init__()
#create the layout
self.pagelayout = QVBoxLayout()

self.layout1 = QHBoxLayout()
self.layout2 = QHBoxLayout()
self.pagelayout.addLayout(self.layout1)
self.pagelayout.addLayout(self.layout2)
#stretch the widgets, create the ratio
self.pagelayout.setStretch(0,1)
self.pagelayout.setStretch(1,7)

#create two buttons and the plogged function
self.mon_btn = QPushButton(" Monitor ")
self.picv_btn = QPushButton(" Graph V ")
self.picw_btn = QPushButton(" Graph W")
#set the monitor button to be true
self.mon_btn.setCheckable(True)

#add the labels
```

```
self.Lay_1()
self.Lay_2()
self.setLayout(self.pagelayout)

def Lay_1(self):
plat_n = QLabel("6 DOF Platform")
self.layout1.addWidget(plat_n)
self.layout1.setContentsMargins(500,0,500,0)

def Lay_2(self):
#create the sublevel layout
layout2_1 = QVBoxLayout()
layout2_2 = QGridLayout()
layout3 = QVBoxLayout()
self.layout2.addLayout(layout2_1)
self.layout2.addLayout(layout2_2)
self.layout2.setStretch(0,1)
self.layout2.setStretch(1,4)

#add the button widget
layout3.addWidget(self.picv_btn)
layout3.addWidget(self.picw_btn)

#add the labels in the left colume
plat_pos_n = QLabel("Catersian \nPosition\n")
plat_vel_n = QLabel("\n\nCatersian \nVelocity\n\n\n\n")
layout2_1.addWidget(plat_pos_n)
layout2_1.addWidget(plat_vel_n)

#create the labels and add them into the gridlayout
#the title of velocity of the platform
```

```
plat_vx_n = QLabel("Vx")
plat_vy_n = QLabel("Vy")
plat_vz_n = QLabel("Vz")
plat_vt_n = QLabel("V")
layout2_2.addWidget(plat_vx_n,0,0)
layout2_2.addWidget(plat_vy_n,0,1)
layout2_2.addWidget(plat_vz_n,0,2)
layout2_2.addWidget(plat_vt_n,0,3)

#the velocity of the platform
self.plat_vx = QLabel("Vx_value")
self.plat_vy = QLabel("Vy_value")
self.plat_vz = QLabel("Vz_value")
self.plat_vt = QLabel("V_value")
layout2_2.addWidget(self.plat_vx,1,0)
layout2_2.addWidget(self.plat_vy,1,1)
layout2_2.addWidget(self.plat_vz,1,2)
layout2_2.addWidget(self.plat_vt,1,3)

#the value of angular velocity of the platform
plat_wx_n = QLabel("wx")
plat_wy_n = QLabel("wy")
plat_wz_n = QLabel("Wz")
layout2_2.addWidget(plat_wx_n,2,0)
layout2_2.addWidget(plat_wy_n,2,1)
layout2_2.addWidget(plat_wz_n,2,2)
layout2_2.addWidget(self.mon_btn,2,3)

#the angular velocity of the platform
self.plat_wx = QLabel("Wx_value")
self.plat_wy = QLabel("Wy_value")
```

```
self.plat_wz = QLabel("Wz_value")
layout2_2.addWidget(self.plat_wx,3,0)
layout2_2.addWidget(self.plat_wy,3,1)
layout2_2.addWidget(self.plat_wz,3,2)
layout2_2.addLayout(layout3,3,3)

class WorkerAng(QRunnable):
def __init__(self):
super(QRunnable,self).__init__()
def run(self):
self.info = []
p = subprocess.Popen("python w_mon.py",shell = True, stdout = subprocess.PIPE,stderr =
subprocess.STDOUT)
while p.poll() is None:
self.info.append(p.stdout.readline())
return self.info

class WorkerDrawGraph1(QRunnable):
def __init__(self):
super(QRunnable,self).__init__()
#run the rqt_plot to draw the graph of pos/vel[0]~[5]
def run(self):
subprocess.Popen("rqt_plot
/joint_states/position[0]:position[1]:position[2]:position[3]:position[4]:position[5]",shell =
True)

class WorkerPubMsg(QRunnable):
def __init__(self):
super(QRunnable,self).__init__()
#def publish(self):
```

```
class WorkerDrawGraph2(QRunnable):
def __init__(self):
super(QRunnable,self).__init__()
#run the rqt_plot to draw the graph of vx-vy-vz
def run(self):
subprocess.Popen("rqt_plot /plat_states/data[0]:data[1]:data[2]",shell = True)

class WorkerDrwaGraph3(QRunnable):
def __init__(self):
super(QRunnable,self).__init__()
#run the rqt_plot to draw the graph of wx-wy-wz
def run(self):
subprocess.Popen("rqt_plot /plat_states/data[3]:data[4]:data[5]",shell = True)

if __name__ == "__main__":
app = QApplication(sys.argv)
window = MainWindow()
window.show()
app.exec_()

#! /usr/bin/env python

import sys,rospy
import actionlib

from control_msgs.msg import FollowJointTrajectoryAction
from control_msgs.msg import FollowJointTrajectoryGoal
from trajectory_msgs.msg import JointTrajectory,JointTrajectoryPoint

class TraExe:
def __init__(self):
```

```
#Initialize the node
rospy.init_node("tra_exe")
#Connect the manipulator controller action server
rospy.loginfo("Waiting for the controller...")
#Set the actionlib client 'fake_endeffector_controller'
self.arm_client =
actionlib.SimpleActionClient('fake_endeffector_controller',FollowJointTrajectoryAction)
self.arm_client.wait_for_server()
rospy.loginfo("...connected")
#create the trajectory
self.tra = JointTrajectory()
#create the arm_name
self.arm_name = ['shoulder_pan_joint','shoulder_lift_joint','elbow_joint',
'wrist_1_joint','wrist_2_joint','wrist_3_joint']
self.tra.joint_names = self.arm_name

def load_tra(self,points_num):
filename = "trajectory.txt"
with open(filename) as f_obj:
for n in range(0,points_num):
self.l_index = f_obj.readline()
self.l_pos = f_obj.readline()
self.l_vel = f_obj.readline()
self.l_acc = f_obj.readline()
self.l_tim = f_obj.readline()
'''
print(self.l_index)
print(self.l_pos)
print(self.l_vel)
print(self.l_acc)
print(self.l_tim)
```

```
'''
pos = self.strl2floatl(self.l_pos)
vel = self.strl2floatl(self.l_vel)
acc = self.strl2floatl(self.l_acc)
tim = float(self.l_tim)

tra_point = JointTrajectoryPoint()
tra_point.positions = pos
tra.point.velocities = vel
tra.point.acceleraties = acc
tra.point.time_from_start = tim

self.tra.points.append(tra_point)

'''
av_useful = (len(self.l_vel) >= 100)
print(len(self.l_pos))
print(len(self.l_vel))
if av_useful :

else:
'''
def strl2floatl(self,str):
kat = []
length = len(str)
for i in range(1,length):
if str[i] == ',':
kat.append(i)
#find the str number
s_num1 = str[1:(kat[0]-1)]
s_num2 = str[(kat[0]+1):(kat[1]-1)]
```

```
s_num3 = str[(kat[1]+1):(kat[2]-1)]
s_num4 = str[(kat[2]+1):(kat[3]-1)]
s_num5 = str[(kat[3]+1):(kat[4]-1)]
s_num6 = str[(kat[4]+1):(length-2)]
#convert the string to float
num1 = float(s_num1)
num2 = float(s_num2)
num3 = float(s_num3)
num4 = float(s_num4)
num5 = float(s_num5)
num6 = float(s_num6)
#comine the six float into a list and retuen
num = [num1,num2,num3,num4,num5,num6]

return num
def go(self):
rospy.logo("Moving the arm to goal position")
#create an empty trajectory goal
self.arm_goal = FollowJointTrajectoryGoal()
#Set the trajectory component to the goal trajectory created above
self.arm_goal.trajectory = self.tra
#Specify the zero tolerance for the execution time
self.arm_goal.goal_time_tolerance = rospy.Duration(0.0)
#Send the goal to the action server
self.arm_client.send_goal(self.arm_goal)

if __name__ == "__main__":
try:
TraExe()
except rospy.ROSInterruptException:
pass
```

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import rospy,sys
import moveit_commander

from control_msgs.msg import GripperCommand

from nav_msgs.msg import Path
from geometry_msgs.msg import Quaternion, PoseStamped




#initializing the API of move_group
moveit_commander.roscpp_initialize(sys.argv)
#initializing the ROS node
rospy.init_node('endeffector_visulization')
path_pub = rospy.Publisher('show_path', Path ,queue_size = 5)
#initializing the arm group in the target ur
arm = moveit_commander.MoveGroupCommander('manipulator')
rate = rospy.Rate(100)
#substance the path
path = Path()

current_time = rospy.Time.now()

path.header.stamp = current_time
path.header.frame_id = "/world"

while not rospy.is_shutdown():
```

```
this_pose_stamped = PoseStamped()

#get end_effectror_link_pose
pose = arm.get_current_pose()
print pose

#print end_effector_link
#print end_effector_link.position.y
#print end_effector_link.position.z
#print end_effector_link.orientation.x
#print end_effector_link.orientation.y
#print end_effector_link.orientation.z
#print end_effector_link.orientation.w
path.poses.append(pose)
path_pub.publish(path)
rate.sleep()



#!user//bin/python
import serial
import time

class Adr:
def __init__(self,port_name):
#turn on the serial and set initialization
#connected the serial port,set the baudrate
self.ser = serial.Serial(port_name,9600)
print("Set the serial port: "+ port_name +"\nSet the BaudRate: 9600")

def heat_cmd(self,enable):
#enable/disenble the heat process
```

```
if enable:
self.ser.write('HS\n'.encode()) #HS---heat start
print("Heat: S---start")
else:
self.ser.write('HO\n'.encode()) #HO---heat over
print("Heat: O---over")

def tem_read(self):
#read the message transported by the serial
line = self.ser.readline()
#print(line)
if line[0:2]=='PT':
prin_tem = line[2:4]
bed_tem= line[9:11]
print("Printing temprature:"+prin_tem+"\t")
print("Bed temprature:"+bed_tem+"\n")
#return the line read just now
if(prin_tem and bed_tem):
return prin_tem,bed_tem

def motor_cmd(self,enable):
#control the motor to squash the material
if enable:
self.ser.write("CS\n".encode()) #CS----command strat
print("Command: S---start")
else:
self.ser.write("CO\n".encode()) #CT----command over
print("Command: O----over")

def exe_cmd(self,setted_bed_tem,setted_prin_tem):
#send the execute commad to motor
```

```
self.motor_cmd(True)
self.set_prin_tem(setted_prin_tem)
self.set_bed_tem(setted_bed_tem)

def set_prin_tem(self,prin_tem):
#write the command tp set ptin_tem
self.ser.write(("TP"+prin_tem+"\n").encode()) #TP---set prin_tem
print("Set prin_tem :"+prin_tem)

def set_bed_tem(self,bed_tem):
#write the command to set the bed_tem
self.ser.write(("TB"+bed_tem+"\n").encode()) #TB---set bed_tem
print("Set bed_tem :"+bed_tem)
def tem_close(self):
#turn off the serial
self.ser.close()

if __name__ == "__main__":
adr = Adr("/dev/ttyUSB0")
while True:
time.sleep(0.25)
```