Homework 1

Yunfan Gao

September 9, 2020

1. TCP, known as the tool center point, is the location on the end effector or tool of a robot manipulator whose position and orientation define the coordinates of the controlled object. [1].
2. MoveL Ptarget, v2000, fine, tool1\Wobj:=wobj1 [2];
3. The cylinder diagram with red markers are added to the original figure.

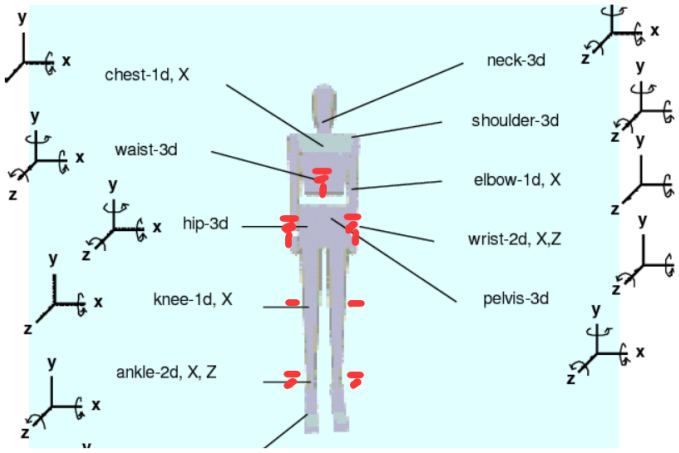


Figure 1: The controlled degrees of freedom of the human model. There are 18 body segments and a total of 36 controlled degrees of freedom [3]

(a) 3 at the waist
(b) 3 at the hip
(c) 1 at the knee
(d) 2 at the ankle, to be consistent with cited source. There are also some sources with different model considering the ankle as Ball-and-socket joint (3 DOF).

total $= 3 + 2 \cdot (3 + 1 + 2) = 15$

4. Operating System: Windows10

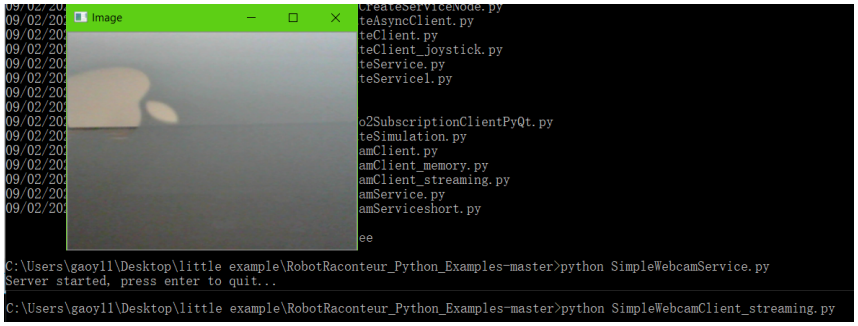


Figure 2: Robot Raconteur webcam streaming verification from Windows 10

Listing 1: Webcam Service example code [4]

```
#Simple example Robot Raconteur webcam service

#Note: This example is intended to demonstrate Robot
    ↪ Raconteur
#and is designed to be simple rather than optimal.

import time
import RobotRaconteur as RR
#Convenience shorthand to the default node.
#RRN is equivalent to RR.RobotRaconteurNode.s
RRN=RR.RobotRaconteurNode.s
import threading
import numpy
import traceback
import cv2
import platform
import sys
import argparse

#Class that implements a single webcam
class Webcam_impl(object):
    #Init the camera being passed the camera number and the
        ↪ camera name
    def __init__(self,cameraid,cameraname):
        self._lock=threading.RLock()
        self._framestream=None
        self._framestream_endpoints=dict()
        self._framestream_endpoints_lock=threading.RLock()
        self._streaming=False
```

```
        self._cameraname=cameraname

        #Create buffers for memory members
        self._buffer=numpy.array([],dtype="u1")
        self._multidimbuffer=numpy.array([],dtype="u1")

        #Initialize the camera
        with self._lock:
            if platform.system() == "Windows":
                self._capture=cv2.VideoCapture(cameraid + cv2.
                    ↪ CAP_DSHOW)
            else:
                self._capture=cv2.VideoCapture(cameraid)
            self._capture.set(3,320)
            self._capture.set(4,240)

    #Return the camera name
    @property
    def Name(self):
        return self._cameraname

    #Capture a frame and return a WebcamImage structure to
        ↪ the client
    def CaptureFrame(self):
        with self._lock:
            image=RRN.NewStructure("experimental.createwebcam2
                ↪ .WebcamImage")
            ret, frame=self._capture.read()
            if not ret:
                raise Exception("Could␣not␣read␣from␣webcam")
            image.width=frame.shape[1]
            image.height=frame.shape[0]
            image.step=frame.shape[1]*3
            image.data=frame.reshape(frame.size, order='C')

            return image

    #Start the thread that captures images and sends them
        ↪ through connected
    #FrameStream pipes
    def StartStreaming(self):
        if (self._streaming):
            raise Exception("Already␣streaming")
        self._streaming=True
        t=threading.Thread(target=self.frame_threadfunc)
        t.start()
```

```
#Stop the streaming thread
def StopStreaming(self):
    if (not self._streaming):
        raise Exception("Not streaming")
    self._streaming=False

#FrameStream pipe member property getter and setter
@property
def FrameStream(self):
    return self._framestream
@FrameStream.setter
def FrameStream(self,value):
    self._framestream=value
    #Create the PipeBroadcaster and set backlog to 3 so
        ↪ packets
    #will be dropped if the transport is overloaded
    self._framestream_broadcaster=RR.PipeBroadcaster(
        ↪ value,3)



#Function that will send a frame at ideally 4 fps,
    ↪ although in reality it
#will be lower because Python is quite slow. This is for
#demonstration only...
def frame_threadfunc(self):
    #Loop as long as we are streaming
    while(self._streaming):
        #Capture a frame
        try:
            frame=self.CaptureFrame()
        except:
            #TODO: notify the client that streaming has
                ↪ failed
            self._streaming=False
            return
        #Send the new frame to the broadcaster. Use
            ↪ AsyncSendPacket
        #and a blank handler. We really don't care when
            ↪ the send finishes
        #since we are using the "backlog" flow control in
            ↪ the broadcaster.
        self._framestream_broadcaster.AsyncSendPacket(
            ↪ frame,lambda: None)
```

```
            #Put in a 100 ms delay
            time.sleep(.1)

    #Captures a frame and places the data in the memory
        ↪ buffers
    def CaptureFrameToBuffer(self):
        with self._lock:
            #Capture and image and place it into the buffer
            image=self.CaptureFrame()

            self._buffer=image.data
            self._multidimbuffer=numpy.concatenate((image.data
                ↪ [2::3].reshape((image.height,image.width,1)
                ↪ ),image.data[1::3].reshape((image.height,
                ↪ image.width,1)),image.data[0::3].reshape((
                ↪ image.height,image.width,1))),axis=2)

            #Create and populate the size structure and return
                ↪  it
            size=RRN.NewStructure("experimental.createwebcam2.
                ↪ WebcamImage_size")
            size.height=image.height
            size.width=image.width
            size.step=image.step
            return size

    #Return the memories. It would be better to reuse the
        ↪ memory objects,
    #but for simplicity return new instances when called
    @property
    def buffer(self):
        return RR.ArrayMemory(self._buffer)

    @property
    def multidimbuffer(self):
        return RR.MultiDimArrayMemory(self._multidimbuffer)


    #Shutdown the Webcam
    def Shutdown(self):
        self._streaming=False
        del(self._capture)


#A root class that provides access to multiple cameras
class WebcamHost_impl(object):
```

```
    def __init__(self,camera_names):
        cams=dict()
        for i in camera_names:
            ind,name=i
            cam=Webcam_impl(ind,name)
            cams[ind]=cam

        self._cams=cams


    #Returns a map (dict in Python) of the camera names
    @property
    def WebcamNames(self):
        o=dict()
        for ind in self._cams.keys():
            name=self._cams[ind].Name
            o[ind]=name
        return o

    #objref function to return Webcam objects
    def get_Webcams(self,ind):
        #The index for the object may come as a string, so
            ↪ convert to int
        #before using. This is only necessary in Python
        int_ind=int(ind)

        #Return the object and the Robot Raconteur type of
            ↪ the object
        return self._cams[int_ind], "experimental.
            ↪ createwebcam2.Webcam"

    #Shutdown all the webcams
    def Shutdown(self):
        for cam in self._cams.values():
            cam.Shutdown()



def main():

    #Accept the names of the webcams and the nodename from
        ↪ command line

    parser = argparse.ArgumentParser(description="Example␣
        ↪ Robot␣Raconteur␣webcam␣service")
    parser.add_argument("--camera-names",type=str,help="List␣
```

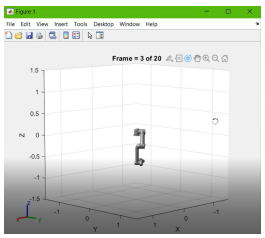
```
    ↪ of␣camera␣names␣separated␣with␣commas")
parser.add_argument("--nodename",type=str,default="
    ↪ experimental.createwebcam2.WebcamHost",help="The␣
    ↪ NodeName␣to␣use")
parser.add_argument("--tcp-port",type=int,default=2355,
    ↪ help="The␣listen␣TCP␣port")
parser.add_argument("--wait-signal",action='store_const',
    ↪ const=True,default=False)
args, _ = parser.parse_known_args()

#Initialize the webcam host root object
camera_names=[(0,"Left"),(1,"Right")]
if args.camera_names is not None:
    camera_names_split=list(filter(None,args.camera_names
        ↪ .split(',')))
    assert(len(camera_names_split) > 0)
    camera_names = [(i,camera_names_split[i]) for i in
        ↪ range(len(camera_names_split))]


obj=WebcamHost_impl(camera_names)

with RR.ServerNodeSetup(args.nodename,args.tcp_port,argv=
    ↪ sys.argv):

    RRN.RegisterServiceTypeFromFile("experimental.
        ↪ createwebcam2")
    RRN.RegisterService("Webcam","experimental.
        ↪ createwebcam2.WebcamHost",obj)

    c1=obj.get_Webcams(0)[0]
    c1.CaptureFrameToBuffer()

    if args.wait_signal:
        #Wait for shutdown signal if running in service
            ↪ mode
        print("Press␣Ctrl-C␣to␣quit...")
        import signal
        signal.sigwait([signal.SIGTERM,signal.SIGINT])
    else:
        #Wait for the user to shutdown the service
        if (sys.version_info > (3, 0)):
            input("Server␣started,␣press␣enter␣to␣quit..."
                ↪ )
        else:
            raw_input("Server␣started,␣press␣enter␣to␣quit
```

```
                    ↪ ...")

        #Shutdown
        obj.Shutdown()


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

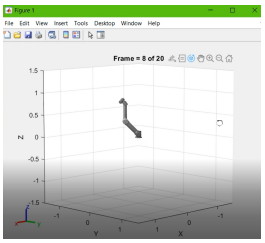
Listing 2: Webcam Client Streaming code [4]

```
#Simple example Robot Raconteur webcam client
#This program will show a live streamed image from
#the webcams. Because Python is a slow language
#the framerate is low...

from RobotRaconteur.Client import *

import time
import numpy
import cv2
import sys

#Function to take the data structure returned from the
    ↪ Webcam service
#and convert it to an OpenCV array
def WebcamImageToMat(image):
    frame2=image.data.reshape([image.height, image.width, 3],
        ↪  order='C')
    return frame2

current_frame=None


def main():

    url='rr+tcp://localhost:2355?service=Webcam'
    if (len(sys.argv)>=2):
        url=sys.argv[1]

    #Startup, connect, and pull out the camera from the
        ↪ objref
    c_host=RRN.ConnectService(url)

    c=c_host.get_Webcams(0)
```

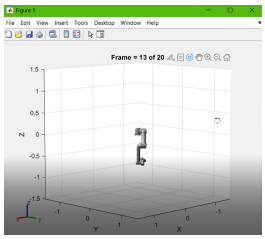
```
    #Connect the pipe FrameStream to get the PipeEndpoint p
    p=c.FrameStream.Connect(-1)

    #Set the callback for when a new pipe packet is received
        ↪ to the
    #new_frame function
    p.PacketReceivedEvent+=new_frame
    try:
        c.StartStreaming()
    except: pass

    cv2.namedWindow("Image")

    while True:
        #Just loop resetting the frame
        #This is not ideal but good enough for demonstration

        if (not current_frame is None):
            cv2.imshow("Image",current_frame)
        if cv2.waitKey(50)!=-1:
            break
    cv2.destroyAllWindows()

    p.Close()
    c.StopStreaming()

#This function is called when a new pipe packet arrives
def new_frame(pipe_ep):
    global current_frame

    #Loop to get the newest frame
    while (pipe_ep.Available > 0):
        #Receive the packet
        image=pipe_ep.ReceivePacket()
        #Convert the packet to an image and set the global
            ↪ variable
        current_frame=WebcamImageToMat(image)

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



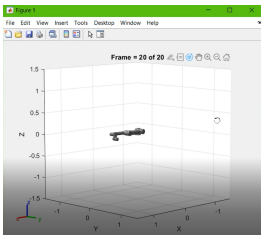
(a) 3rd frame



(b) 8th frame



(c) 13th frame



(d) last frame

Figure 3: plots of simulation at different points of the movement

5. See Figure 3 for simulation results, and matlab code in Listing 3

Listing 3: matlab code

```
close all;

%%% Example animation of jogging robot joints
robotname="universalUR5";
robot = loadrobot(robotname,"DataFormat","column","Gravity",[0 0
    ↪ -9.81]);

%get current configuration
homeJointPosition=robot.homeConfiguration;
%set joint configuration
targetJointPosition = [1 1 1 1 1 1]';

%joint state sinc interpolation
t = linspace(0,2,20);
v = pi/4*sin(2*pi*t);

yPD = v.*targetJointPosition+homeJointPosition

%display animation
exampleHelperRigidBodyTreeAnimation(robot,yPD',1)
```

References

[1] E. McGraw-Hill Dictionary of Scientific Technical Terms, "tool-center point." https://encyclopedia2.thefreedictionary.com/tool-center+point.

[2] A. Robotics, "Technical reference manual: Rapid instructions, functions and data types," 2014.

[3] Shih-Kai Chung and J. K. Hahn, "Animation of human walking in virtual environments," in *Proceedings Computer Animation 1999*, May 1999, pp. 4–15.

[4] J. Wason, "Robotraconteur$_{p}ython_{e}xamples,''$ $https$: $//github.com/robotraconteur/RobotRaconteur_{P}ython_{E}xamples.git$.