MKI59: Robotlab practical

Practical 5: Gestures and remote sensing with the Kinect

or: towards practising aerobics classes

December 14, 2017

In order to plot the gestures that we will collect, make sure that you have installed the 2D plotting library *Matplotlib*. Once we have collected a set of gestures, we can pursue gesture recognition or try to mimic the (recognized?) gestures. Consider an aerobics teacher who shows a group of robots some exercises... each robot will be able to use the remote Kinect sensor that we will test today - while receiving the same set of gestures.

1 Introduction

Unil now, we used the so-called egocentric view for building reactive robots. The NAO used its own sensors to try to understand what happens in the environmnt. Today, we will start to employ remote sensors. These may be cameras on the ceiling or other tracking devices, like the Kinect. We will start with setting up an environment through which your robots (actually: your python scripts that control the robot):

- can connect to a Kinect server through sockets
- can receive so-called SkeletonFrames and store them on disk
- can explore the skeleton frames
- can perform simple gesture processing (2D hand gestures)
- *s*ynchronously wait for incoming gestures and react based on the recognized intent

2 Connecting to the Kinect sensor using sockets

You may recall that *money made the World go round*. Well, since the 70's this has changed: sockets make the World go round. We have set up the Mountain (our powerful robotlab PC) such that it can transmit Kinect data to any interested client. Kinect data is represented by SkeletonFrames. To get a frame: (i) connect to the mountain; (ii) read how much data the Mountain will send; (iii) receive the data and (iv) process it.

2.1 Receiving Kinect skeleton frames and json representation

The Microsoft Kinect captures streams of r,g,b and depth images. It contains a host of algorithms to process frames. A powerful tracking algorithm is able to transfer these streams to skeleton data: each sample contains data from 20 joints.

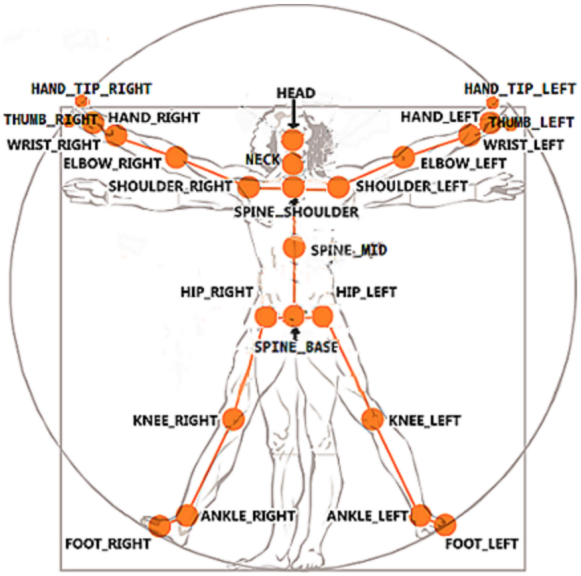

Figure 1: Kinect skeleton model with joints

The Kinect server sends these skeletons in *json* format. *Now, try to connect to the Mountain and receive these frames!*

Code snippet 1: Connect to the Mountain; get skeleton frames

```
import socket
import time
import json #Server sends json skeleton frames

ip = "192.168.1.100" #The ip of the kinect server on the mountain
port = 1337

print "connecting to KinectServer at: "+ip
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect((ip, port))
print "connected, now waiting for SkelFrames"

tstart = time.time()
while True:
        msg    = s.recv(4)     #The 4 bytes represent an integer,
        nbytes = int(msg)      #representing nr of bytes that follow
        t      = time.time()
        skel   = s.recv(nbytes)
        print 1000*(t-tstart) #Time in milliseconds
        print skel
```

Note that you will receive nothing until you (or your peers) are actually generating gesture data by moving in front of the Kinect camera. Your script (*recv(nbytes)*) will simply just wait until bytes are sent. Frames are sampled at 30Hz. Please check if: this sample rate corresponds to what you get and: if samples are received with continuous sample rate (are samples equidistant in time?), and: what happens if you stop gesturing.

Furthermore, please note that *recv(nbytes)* is not guaranteed to really read *nbytes* of data! Use the routine *fullRead(nbytes)* instead. Use a buffer size of 2048 to be safe.

Code snippet 2: Keep reading data from socket until nbytes read

```
def fullRead (nbytes):
        chunks = []
        bytes_recd = 0
        while bytes_recd < nbytes:
            chunk = s.recv(min(nbytes - bytes_recd, BUFFER_SIZE))
            if chunk == '':
                raise RuntimeError("Socket connection to KinectServer broken")
            chunks.append(chunk)
            bytes_recd = bytes_recd + len(chunk)
        return ''.join(chunks)
```

For now, this is fine. However, please note that if the Kinect server breaks down, an error is raised which makes your script exit. Later on, you may want to keep your scripts alive by catching these events (and try to reconnect or continue doing other stuff). See how you can catch socket errors using *s.error*.

2.2 Accessing joints from skeleton data

We will have to parse the json data to extract joint coordinates. We will focus on hand gestures. If you want to use other body parts, take a look at https://msdn.microsoft.com/en-us/library/microsoft.kinect.jointtype.aspx

Code snippet 3: Accessing joints from skeleton data

```
import json

LHAND = 7
RHAND = 11

# assuming we have read a skeleton as illustrated above

jskel = json.loads(skel)
lx = jskel[LHAND]['Position']['X']
ly = jskel[LHAND]['Position']['Y']
lz = jskel[LHAND]['Position']['Z']
rx = jskel[RHAND]['Position']['X']
ry = jskel[RHAND]['Position']['Y']
rz = jskel[RHAND]['Position']['Z']

print '(x,y,z) for the left hand is: (',lx,ly,lz,')'
```

2.3 Gesture segmentation: begin and end of gesture

The Kinect server will start sending skeleton data the moment some subject is making gestures in front of the camera (first, a skeleton has to be detected). *Try this out. Does your script know when to stop receiving data? No it doesn't, it will keep waiting for incoming skeleton frames forever.* To solve this problem of end-point detection, we will use a powerful socket mechanism called *select*. The select routine allows you to check, for example, if some input is ready to be received on a particular socket. It also allows you to wait for incoming data until a certain timeout has passed. *Determine a proper timeout value after which it is probable that a gesture has finished. Also discuss other ways to determine that a gesture has finished.*

The reason that we're spending some time on exploring these issues is that we want to use gestures to communicate with the NAO. A gesture should have a clear beginning and end-point. Now, one way to control this is to use a period during which no skeleton data is received (cfr. end-point detection in speech: when a user is silent, an utterance is finished). Here: when the user moves out of the field of vision of the Kinect, the skeleton stream is stopped. You can detect this using *select* (see this link or this other link for more details). We will use select to determine wether there is data to read, and otherwise we will wait:

Code snippet 4: Example of using select()

```
import select

if __name__ == "__main__":
    #In case of disconnects, do this in the main loop to reconnect everytime
    print "Connecting to KinectServer at: "+self.ip
    s = sock.socket(sock.AF_INET, sock.SOCK_STREAM)
    s.connect((ip, port))
    print "Connected, now waiting for SkelFrames"

    timeout = .5 # seconds

    lx = []
    ly = []
    rx = []
    ry = []

    while True:
            ready_read,ready_write,serror = select.select([s],[],[],timeout)
            if ready_read:
                gesture = recordGestures(lx, ly, rx, ry)
                    if gesture is not None
                        print gesture
                        time.sleep(timeout)
                    else:
                        time.sleep(timeout)
        else:
                # nothing to read and not receiving anything
                time.sleep(timeout)
```

Then, if there is data to read, we will record a gesture by recording the skeleton frames and checking whether a gesture is performed:

Code snippet 5: Method for recording a gesture

```
def recordGestures(lx, ly, rx, ry):
    """
    Record a gesture
    """
    msg = s.recv(4)
    nbytes = int(msg)
    skel = fullRead(nbytes)
    jskel = json.loads(skel)
    #Append the next skeleton frame
    lx.append(skel[LHAND]["Position"]["X"])
    ly.append(skel[LHAND]["Position"]["Y"])
    rx.append(skel[RHAND]["Position"]["X"])
    ry.append(skel[RHAND]["Position"]["Y"])
    detected = detectGestures(lx,ly,rx,ry)
    return detected
```

You will determine yourself what gestures you want to detect. You will have to play with the x, y and z coordinates that are recorded and determine when a gesture is detected. To start, try accepting any movement in front of the camera when the user stops moving; this means their x, y and z coordinates do not change (very much):

Code snippet 6: Detecting a gesture

```
def detectGestures(lx, ly, rx, ry):
    """
    Detect which gesture is being performed
    """
    gesture = None
    if len(lx) > 10: #After recording at least 10 data points
        #If both the X and Y distance of both the left and right hand are less
        #than the threshold, the gesture has ended
        LXstop =
        LYstop =
        RXstop =
        RYstop =
        if LXstop and LYstop and RXstop and RYstop:
            gesture = "stop"
    return gesture
```

The *select()* mechanism allows for asynchronous communication: we don't have to wait endlessly for data to arrive using synchronous *recv()* calls. Alternatively, we could use threads to combine, e.g. robot behaviour or other sensor processing while reading Kinect data.

3 Plotting gestures

We assume that you have captured a gesture and stored all gesture data in arrays *(lx,ly)* and (for the right hand) *rx,ry)*, as illustrated in Code snippets 3, 4, 5 and 6. We will use *Matplotlib* for plotting the gestures.

Code snippet 7: Extracting hand trajectories and plotting the results

```
import matplotlib.pyplot as plt
```

```
#Don't take all the data, but from 1/3 of the data up to 2/3 of the data
begin =int(m.floor(len(rx)/3))
end = int((2*m.floor(len(rx)/3)))
#Plot the detected gesture
plt.plot(lx[begin:end] ,ly[begin:end] ,'ro-')
plt.plot(lx[begin], ly[begin],'bo',ms =10)
plt.plot(lx[end],ly[end],'o',ms=10,c='black')
plt.plot(rx[begin:end] ,ry[begin:end] ,'g+-')
plt.plot(rx[begin], ry[begin],'bo',ms =10)
plt.plot(rx[end],ry[end],'o',ms=10,c='black')
plt.show()
```

We've plotted a two-handed circular gesture using this script, resulting in Figure 2 below. Values on x and y axes are in meters. Starting points of the gesture are in color blue, end points in black.

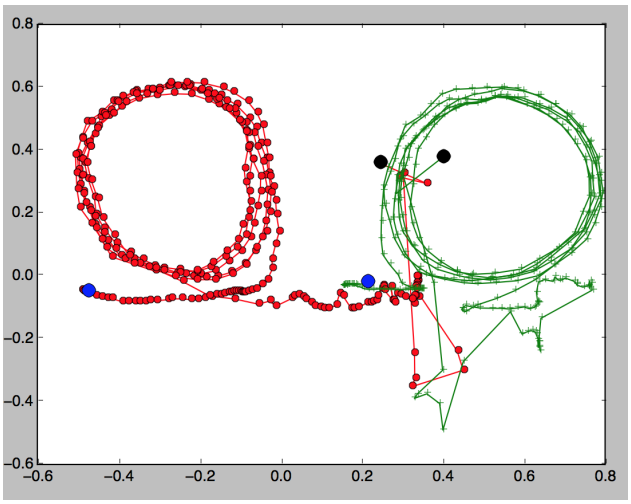

Figure 2: Two- handed circular 2D gestures

Please note that gesture segmentation is a delicate process: starting points seem OK, but end-points are detected too late. Still, two circles are clearly distinguishable.

4 Assignment

For this assignment, you will develop a simple application that uses a NAO robot as a sign-language interpreter. To this end, select 5-6 simple (and visually distinctive!) gestures from a Dutch/English sign language (google image/video has plenty of examples). Then, record these gestures using Kinnect. You may want to repeat the gestures a few times to see the similarities and emerging patterns. Plot the results and think how you can recognize these gestures without using any classifier. Feel free to extract any knowledge about the hand trajectories (i.e., some gestures use the right hand, and not left etc.). Finally, put these parts together (gesture recording, recognition) and make the robot say the recognized gesture (e.g., hello, Goodbye) or something else in case the robot fails to recognize the gesture or the performed gesture doesn't belong to the robot's dictionary of gestures (e.g., I dont understand). Upload the *.zip file that contains the figure with selected gestures, plots of hand trajectories and your code.