Overwatch 家庭管理小幫手

R06921058 方浩宇

R06921081 張邵瑀

(1) 問題分析

以遠端運算資源、感應器當作服務提供者，執行遠端居家守望服務，再透過voice kit與google api 的語音辨識系統來完成系統的功能輸入。

(2) 解決方法、流程

　　server:

於遠端電腦藉由kafka當作命令接收與即時資料傳輸載體，於各個  
 topic接收資料，並藉由接收到的指令執行服務。

Server Framework：

由於Iot的架構通常為服務導向，所以我設計的server架構為requset   
 listener Server在有空閒時會不斷的接收todolist 收到agent的  
 request就會去開一條thread執行該任務，所以若是有新的程式更新  
 ，只需要把新的程式寫進services.py 中並在function 底部把function  
 加入 program dictionary中，在主程式中呼叫getProgram就會把  
 services.py中的function以字典的形式傳回server，而在server 中一  
 旦收到相關的request，就會以字典的形式查找，該function並開一  
 條thread執行該項服務，理論上透過python 的特性是可以透過動態  
 重新import 新的package在使得server不需要關閉的情況下提供更新  
 過後的服務，如下圖



　　client:

遠端agent負責傳送命令給server端負責執行運算，並將結果丟到  
 topic讓agent去取得執行結果。  
　　流程：

使用kafka 作為server 載體，因為他的幾個特性可以滿足我們的需  
 求：可擴展性、數據分區、低延遲、處理大量不同消費者的能力。  
 接著處理可以建立雙向溝通的方法，訂定資料傳輸格式，並且確定  
 可以做雙向的串流處理，所以我寫了一個即時雙向語音的溝通程式  
 測試可行性，確保kafka 的Publish/Subscribe機制可以處理python  
 中的資料，而未來任何經由python之程式都可以經由此種program  
 model做開發與溝通並於server上做服務的擴充。

(3) 系統架構 ( 硬 + 軟 )

硬體架構：

1. Agent:

* Raspberry Pi 3
* voice Kit

2. Overwatch(Server):

* Ubuntu 16.04
* webcam
* GPU Nvidia 1070

軟體架構：

* Agent  
  
* Server  
  

(4) 模組及功能( 多用了什麼模組及其他API)

* Voice Kit  
  microphone: 接收語音輸入  
  speaker: 播放結果
* Google Assistant SDK  
  google 語音辨識: 辨認語音輸入  
  google 翻譯: 將YOLO回傳的物品翻譯為中文  
  google 助理: 將英文版語音助理整合入agent
* YOLO  
  darknet: 即時辨識影像中物體
* kafka  
  作為資料傳輸載體
* python  
  wave: 收集聲音資訊  
  pyaudio: 將收到的聲音藉由speaker放出
* Microsoft azure

在雲端儲存暫時的資料並且送信。

(5) 核心函式 ( code )

1.monitorHome:

藉由webcam擷取照片後丟進darknet中辨別影像中物體，再將結果  
 傳回給遠端agent用途分別為

* 冰箱物件偵測
* 家中人員確認

|  |
| --- |
| def monitorhome(prog, sour, dest, action, args, data):  import subprocess as sub  p = sub.Popen(['python',  './darknet.py',  prog, sour, dest, action,  str(args), str(data) ],stdout=sub.PIPE,stderr=sub.PIPE)  output, errors = p.communicate() |

2.talker:

藉由將Local的聲音資料publish到自身id的topic 再由對方去subscrib所以達到多對多的語音通話。

|  |
| --- |
| recoder.py |
| from pyaudio import PyAudio, paInt16  from kafka import KafkaProducer  from kafka.errors import KafkaError  import pickle  import threading  import numpy as np  from datetime import datetime  import time  import sys  class recoder:  def \_\_init\_\_(self):  self.samples = 1024  self.rate = 22000  self.threshold = 1000  self.COUNT\_NUM = 20  self.save\_length = 2  self.time\_count = 4  self.sendingBuffer = []  self.stop = False  self.sour = ''  self.dest = ''  self.topic = 'todolist'  def send(self):  producer = KafkaProducer(bootstrap\_servers=['140.112.41.94:9092'])  time.sleep(1)  while not self.stop:  if len(self.sendingBuffer) > 0:  tosend = self.sendingBuffer.pop(0)  l = ['talker', self.sour, self.dest,  'speaker', (2,1,self.rate), np.array(tosend).tostring()]  p = pickle.dumps(l)  future = producer.send(self.topic, p)  try:  record\_metadata = future.get(timeout=10)  except KafkaError:  # Decide what to do if produce request failed...  log.exception()  pass  def recoder(self, sour, dest, topic):  self.sour = sour  self.dest = dest  self.topic = topic  sender = threading.Thread(target=self.send)  sender.setDaemon(True)  sender.start()  pa = PyAudio()  stream = pa.open(format=paInt16, channels=1,  rate=self.rate, input=True,  frames\_per\_buffer=self.samples)  save\_count = 0  save\_buffer = []  time\_count = self.time\_count  while not self.stop:  time\_count -= 1  try:  string\_audio\_data = stream.read(self.samples)  except:  print('except:string\_audio\_data = stream.read(self.samples)')  stream.close()  stream = pa.open(format=paInt16, channels=1,  rate=self.rate, input=True,  frames\_per\_buffer=self.samples)  audio\_data = np.fromstring(string\_audio\_data, dtype=np.short)  large\_sample\_count = np.sum( audio\_data > self.threshold )  if large\_sample\_count > self.COUNT\_NUM:  save\_count = self.save\_length  else:  save\_count -= 1  if save\_count < 0:  save\_count = 0  save\_buffer.append( string\_audio\_data )  if time\_count==0:  self.sendingBuffer.append(save\_buffer)  save\_buffer = []  time\_count = self.time\_count    sys.stdout.flush()  sender.join() |

|  |
| --- |
| receiver.py |
| from kafka import KafkaConsumer  from kafka.errors import KafkaError  import pyaudio  import pickle  import threading  class receiver():  def \_\_init\_\_(self):  self.consumer = None#KafkaConsumer('remote',bootstrap\_servers=['140.112.41.94:9092'])  self.playBuffer = []  self.fmt = 2  self.channels = 1  self.rate = 22000  self.timeout = 15  self.sour = ''  self.dest = ''  self.topic = 'todolist'  def play(self, sour, dest, topic):  self.sour = sour  self.dest = dest  self.topic = topic  recvthr = threading.Thread(target=self.receive)  recvthr.setDaemon(True)  recvthr.start()    p = pyaudio.PyAudio()  #open stream  stream = p.open(format = p.get\_format\_from\_width(self.fmt),  channels = self.channels,  rate = self.rate,  output = True)  print('start')  lastdata = b''    while self.timeout > 0:  if len(self.playBuffer) > 1:  toplay = self.playBuffer.pop(0)  stream.write(toplay)  lastdata = toplay  elif len(self.playBuffer) == 1:  stream.write(self.playBuffer[0])  self.timeout -= 1  print('end')  self.consumer.close()  stream.stop\_stream()  stream.close()  self.timeout = 0  #close PyAudio  p.terminate()  print('displayer close')  return    def receive(self):  self.consumer = KafkaConsumer(self.topic, bootstrap\_servers=['140.112.41.94:9092'])  for message in self.consumer:  try:  unpkl = pickle.loads(message.value)  if unpkl[:4] == ['talker', self.sour, self.dest, 'speaker']:# and unpkl[1] == and unpkl[2] == and unpkl[3] == 'speaker':  self.playBuffer.append(unpkl[5])  except Exception as e:  print(e)  print('disconection') |

(6) 調教及驗證

首先，先是要制定傳輸資料用的格式，如下：

type: list [str <program>, str <source id>, str <destination id>,  
 str <action>, tuple <argument>, list <data> ]

而每個程式在執行時只接收progam與destination id與自己相同的訊息

接著是handshake機制，在talker中要確認server有傳送action為’response’且data 為’connected’的訊息之後才開始接收對方topic中的聲音資訊並開始傳送agent的聲音資料，而連線會在receiver中檢查timeout的剩餘嘗試次數，若timeout降為0則中斷連線並接收下一輪訊息。