angus.ai Documentation

Release

Gwennael Gate, Aurelien Moreau

Feb 21, 2018

Contents

This documentation is here to help you integrate our product / API. We are always happy to help, so feel free to contact us at support@angus.ai.

ANGUS.AI
{JSON}

CHAPTER 1

Audience Analytics

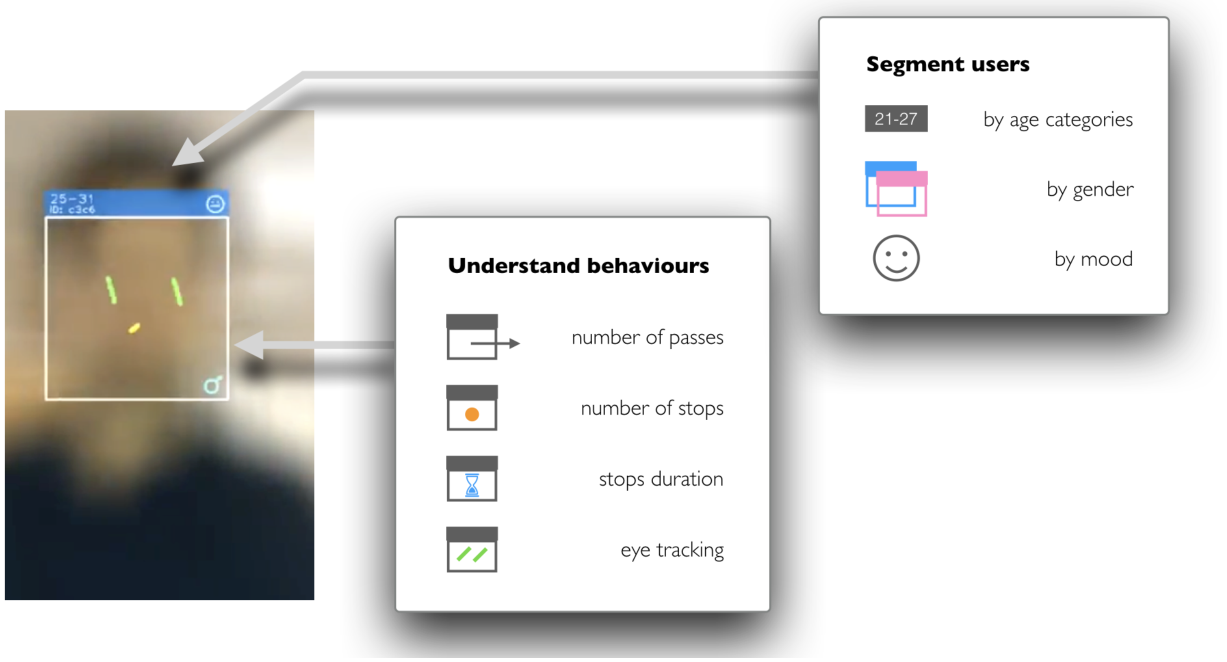
Angus.ai Audience Analytics computes the traffic, interest and demographics metrics of a device (a screen, a kiosk, a specific shelf, etc...).

This data is automatically stored on a secured database and can be visualised in realtime on an online dashboard and/or retrieved programmatically through our API.

This documentation is meant for developers wanting to install, configure and launch Angus.ai audience analytics application on a screen player.

Step 1 - Introduction

What data can be retrieved



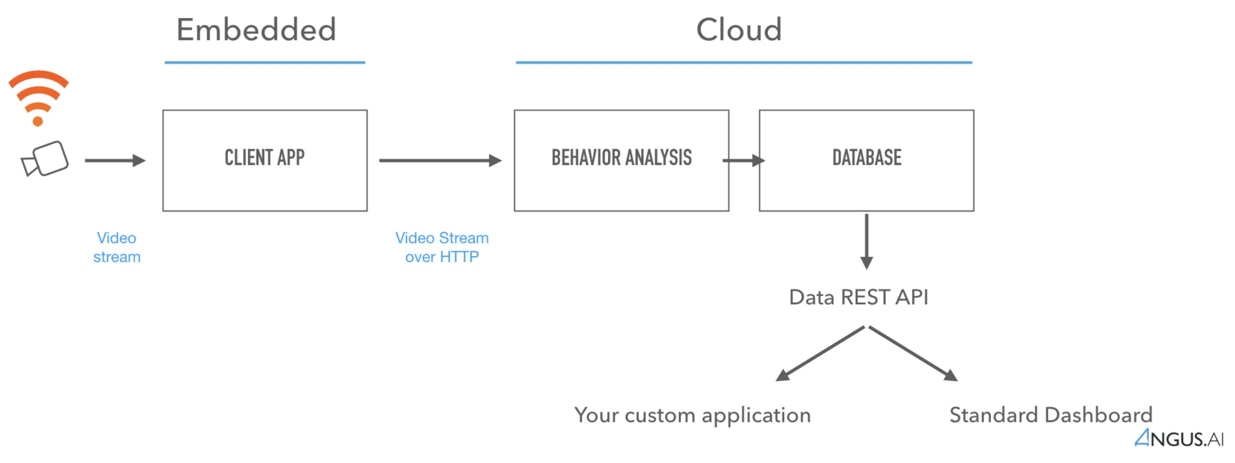
Angus.ai anonymous audience analytics solution computes (from each camera stream) the following metrics:

- The number of people passing by the camera/device,
- The number of people interested in the camera/device
- The time spent stopped in front of the camera/device
- The time spent looking at the camera/device
- The number of people interested by the camera/device, broken down by demographics
  - Age
  - Gender
  - Emotion

For more information about the metrics, see the page dedicated to *the metrics*.

How it works

Angus.ai audience analytics solution is based on a (lightweight) Client / Server architecture as seen on the figure below. All CPU expensive computation are made on our dedicated servers making it possible to run the solution from about any CPU board that can retrieve a camera stream and connect to a server (eg. Raspberry).



Once properly installed and configured, this application will interact with Angus.ai cloud based algorithms to provide audience metrics that can be retrieve through a REST API. This tutorial will show how to do it.

Requirements

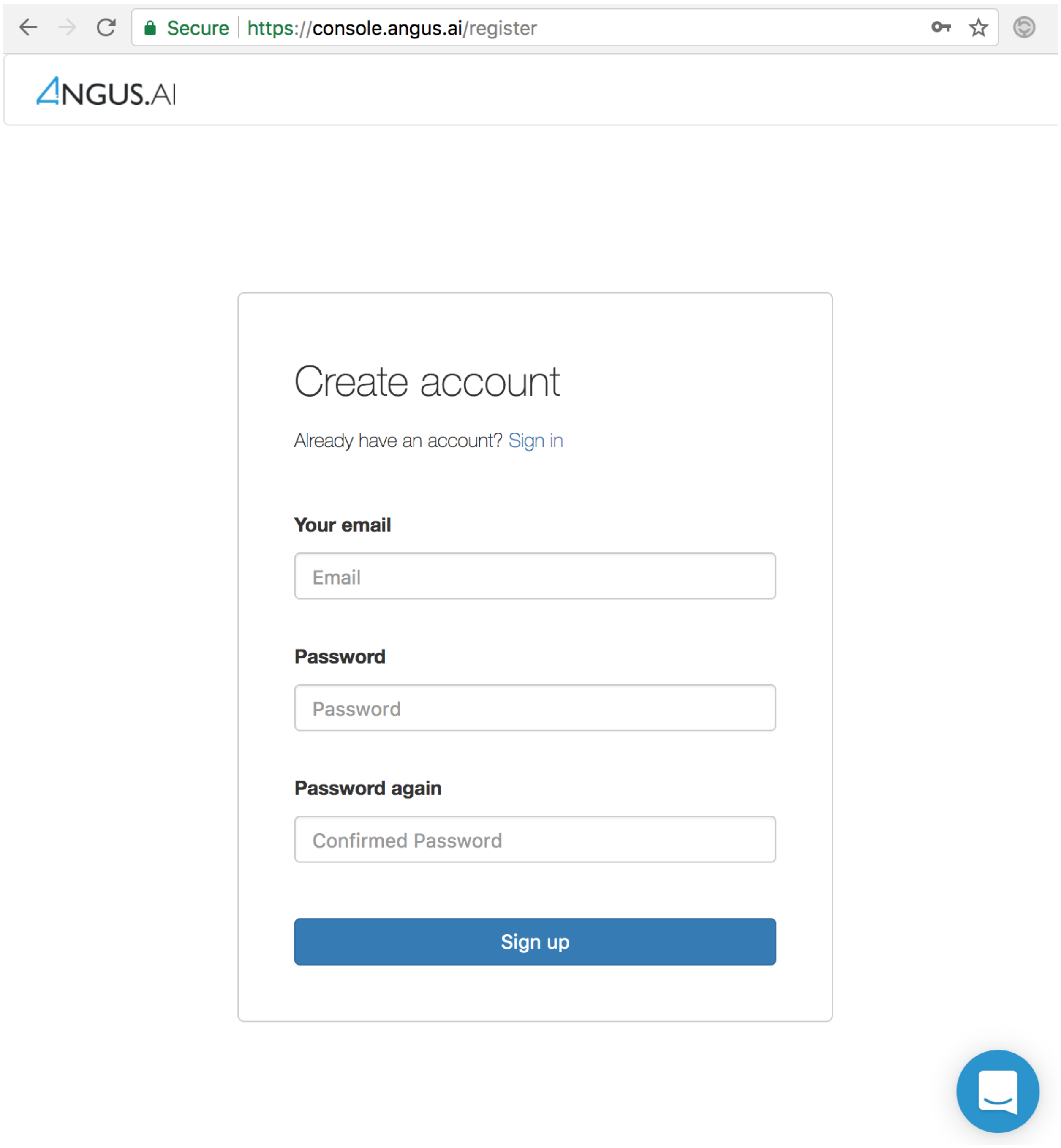
As you go through this tutorial, you will need:

- a computer. Every operating system is ok provided that you can configure a proper Python stack.
- a camera (e.g. webcam) plugged into that computer. USB and IP cameras are supported, although IP cam can be more challenging to interface.
- a working internet connection. An upload bandwidth of about 400ko/sec is advised. If this is a problem, we are able to provide an "hybrid" version of our solution, where part of the CPU expensive computation is done locally, alleviating connection bandwidth requirements. Please contact us at support@angus.ai.

Step 2 - Set up your player

Create an account

To use Angus.ai services, you need to create an account. This can be done very easily by visiting https://console.angus.ai and filling the form shown below.



Create account

Already have an account? Sign in

Your email

Email

Password

Password

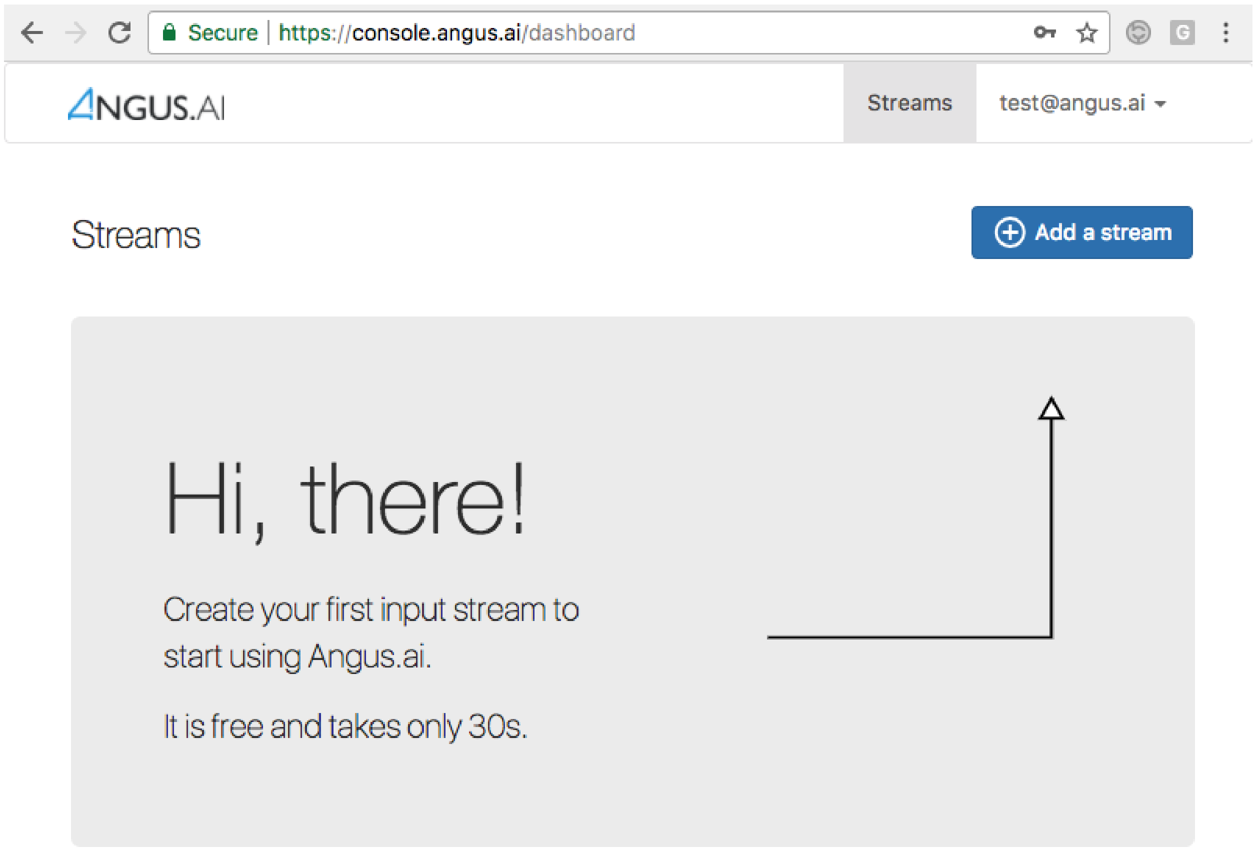
Password again

Confirmed Password

When done, you are ready to create you first camera stream as shown below.

Get credentials for your camera

After creating your personal account on https://console.angus.ai/, you will be asked to create a "stream". This procedure will allow for a private "access_token" and "client_id" keys to be generated for you. This can be done by pressing the "Add a stream" button on the top right hand corner as shown below.

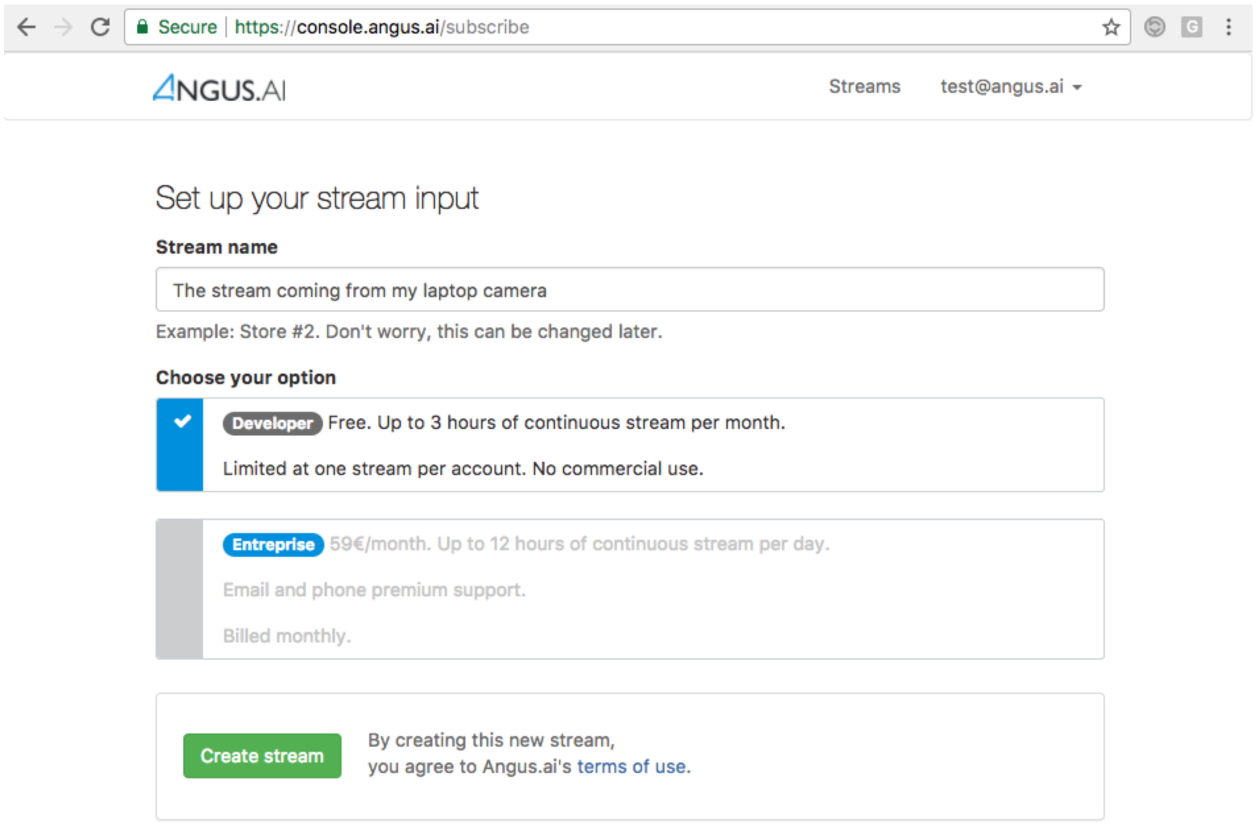


Hi, there!

Create your first input stream to start using Angus.ai.

It is free and takes only 30s.

After clicking, you will be asked to choose between a free developer stream and a paying enterprise stream. Please note that the free developer stream is only for non commercial use and will block after 3 hours of video stream computed every month as seen below.



Set up your stream input

Stream name

The stream coming from my laptop camera

Example: Store #2. Don't worry, this can be changed later.

Choose your option

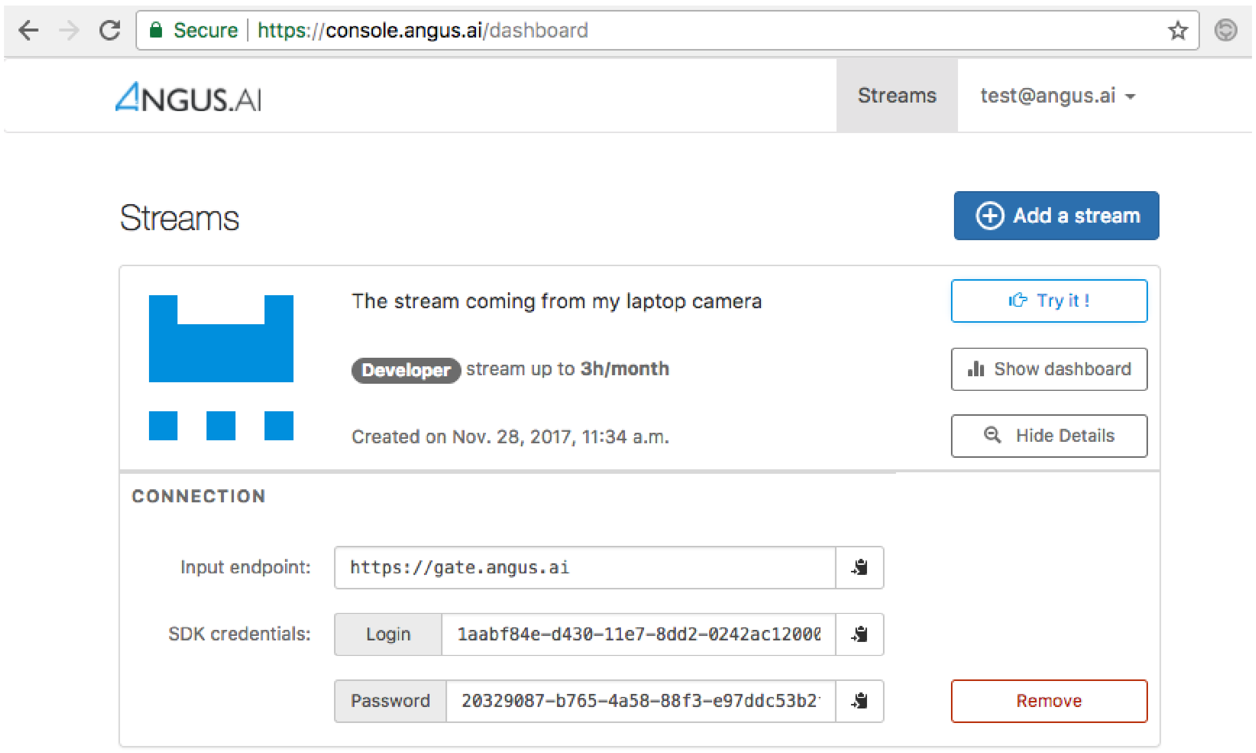
[✓ Developer Free. Up to 3 hours of continuous stream per month. Limited at one stream per account. No commercial use.]

Entreprise 59€/month. Up to 12 hours of continuous stream per day. Email and phone premium support. Billed monthly.

By creating this new stream, you agree to Angus.ai's terms of use.

For an non restricted enterprise stream, you will need to enter a valid credit card number.

Press “Continue” at the bottom of the page and you will soon get the following page. Press “Show Details” and take note of your `client_id` (called Login on the interface) and `access_token` (called Password on the interface) as they will be needed later on.



The stream coming from my laptop camera

Developer stream up to **3h/month**

Created on Nov. 28, 2017, 11:34 a.m.

Input endpoint: `https://gate.angus.ai`

SDK credentials: Login `1aabf84e-d430-11e7-8dd2-0242ac1200`

Password `20329087-b765-4a58-88f3-e97ddc53b2`

The credentials that you have just created will be used to configure the Angus.ai SDK. Your are now ready to proceed to the next step.

Download and configure the SDK

Requirements

- The SDK is Python3 compatible but the documentation code snippets are only Python2 compatible.
- Also, you might want (not mandatory) to create a python virtual environnement with **virtualenv** in order to install the sdk in there.

To do so, please refer to the following virtualenv guide for more information.

Install the SDK

Open a terminal and install the angus python sdk with pip. If you do not use **virtualenv** you may need to be root, administrator or super user depending on your platform (use sudo on linux platform).

```
$ pip install angus-sdk-python
```

Configure your SDK

You must configure your sdk with the keys you received by creating a stream here. These keys are used to authenticate the requests you are about to send.

Your API credentials can be retrieved by clicking on “Show details” on the stream you just created.

In a terminal, type:

```
$ angusme
Please choose your gateway (current: https://gate.angus.ai):
Please copy/paste your client_id: xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx
Please copy/paste your access_token: xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx
```

Fill in the "client_id" prompt with the "login" given on the interface and the "access_token" prompt with the "password" given on the interface.

On **Windows** system, if angusme does not work, please refer to the *FAQ* for more details.

You can check this setup went well by typing the following command and checking that our server sees you:

```
$ angusme -t
Server: https://gate.angus.ai
Status: OK
```

If this command gives you an error, check that you enter the right "client_id" and "acccess_token". You can do this by re-typing "angusme" in a command prompt.

If you need help, contact us here : support@angus.ai !

Download and launch the client application

Our client app is a lightweight, open source Python script.

It performs two basic tasks:

1. retrieve a valid video stream. By default, one of the connected USB camera will be chosen, but you can easily modify the client app to open a different camera and even open a video file.
2. package and send the video stream over https to our computation servers. This part can also be optimized for your needs (image resolution, frame rate, etc...).

If you need help to perform these optimizations, please contact us at support@angus.ai.

Prerequisite

- you have a working webcam plugged into your PC
- you have installed **OpenCV2** and **OpenCV2** python bindings. Please refer to OpenCV documentation to proceed, or check *FAQ* chapter.

On Debian-like platform, **OpenCV2** comes pre-installed, you just need to run

```
$ sudo apt-get install python-opencv
```

Note also that OpenCV2 is not an absolute pre-requisite, the following code sample can easily be adapted to be used with any other way of retrieving successive frames from a video stream.

Client App

Please copy/paste the following code sample in a file and run it.

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

import cv2

import numpy as np
import StringIO
import datetime
import pytz
```

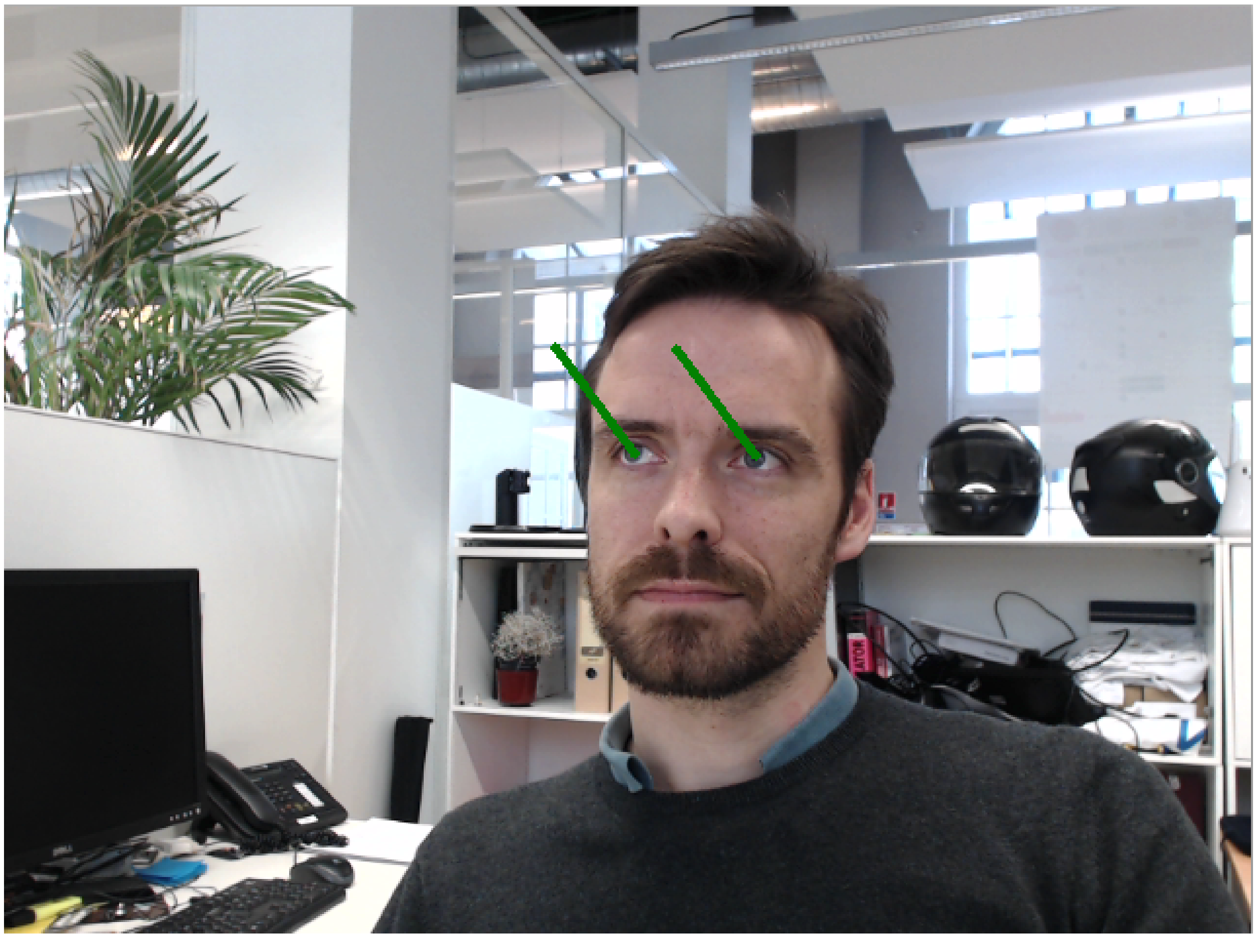
```
from math import cos, sin
import angus.client

def main(stream_index):
    camera = cv2.VideoCapture(stream_index)
    camera.set(cv2.cv.CV_CAP_PROP_FRAME_WIDTH, 640)
    camera.set(cv2.cv.CV_CAP_PROP_FRAME_HEIGHT, 480)
    camera.set(cv2.cv.CV_CAP_PROP_FPS, 10)

    if not camera.isOpened():
        print("Cannot open stream of index {}".format(stream_index))
        exit(1)

    print("Video stream is of resolution {} x {}".format(camera.get(3), camera.
↪get(4)))

    conn = angus.client.connect()
    service = conn.services.get_service("scene_analysis", version=1)
    service.enable_session()

    while camera.isOpened():
        ret, frame = camera.read()

        if not ret:
            break

        gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
        ret, buff = cv2.imencode(".jpg", gray,  [cv2.IMWRITE_JPEG_QUALITY, 80])
        buff = StringIO.StringIO(np.array(buff).tostring())

        t = datetime.datetime.now(pytz.utc)
        job = service.process({"image": buff,
                               "timestamp" : t.isoformat(),
                               "store" : True
        })
        res = job.result

        if "error" in res:
            print(res["error"])
        else:
            # This parses the entities data
            for key, val in res["entities"].iteritems():
                # display only gaze vectors
                # retrieving eyes points
                eyel, eyer = val["face_eye"]
                eyel = tuple(eyel)
                eyer = tuple(eyer)

                # retrieving gaze vectors
                psi = 0
                g_yaw, g_pitch = val["gaze"]
                theta = - g_yaw
                phi = g_pitch

                # Computing projection on screen
                # and drawing vectors on current frame
                length = 150
                xvec = int(length * (sin(phi) * sin(psi) - cos(phi) * sin(theta) *␣
↪cos(psi)))
```

```
                yvec = int(- length * (sin(phi) * cos(psi) - cos(phi) * sin(theta) *␣
↪sin(psi)))
                cv2.line(frame, eyel, (eyel[0] + xvec, eyel[1] + yvec), (0, 140, 0),␣
↪3)

                xvec = int(length * (sin(phi) * sin(psi) - cos(phi) * sin(theta) *␣
↪cos(psi)))
                yvec = int(- length * (sin(phi) * cos(psi) - cos(phi) * sin(theta) *␣
↪sin(psi)))
                cv2.line(frame, eyer, (eyer[0] + xvec, eyer[1] + yvec), (0, 140, 0),␣
↪3)

        cv2.imshow('original', frame)
        if cv2.waitKey(1) & 0xFF == ord('q'):
            break

    service.disable_session()

    camera.release()
    cv2.destroyAllWindows()

if __name__ == '__main__':
    ### Web cam index might be different from 0 on your setup.
    ### To grab a given video file instead of the host computer cam, try:
    ### main("/path/to/myvideo.avi")
    main(0)
```

To run it:

```
$ python yourcopiedfile.py
```

You should see two green vectors showing what your are looking displayed on your screen:



The application displays by default a live view of your stream, with gaze vectors super-imposed. If you need it, it is also possible to display age, gender, emotion, etc... Please refers to the app real-time API here : (*Output API*).

Step 3 - Online Dashboard

The client app you just ran is now feeding a personal and secured database with audience analytics data that you can check by following the steps below.

How to view your dashboard

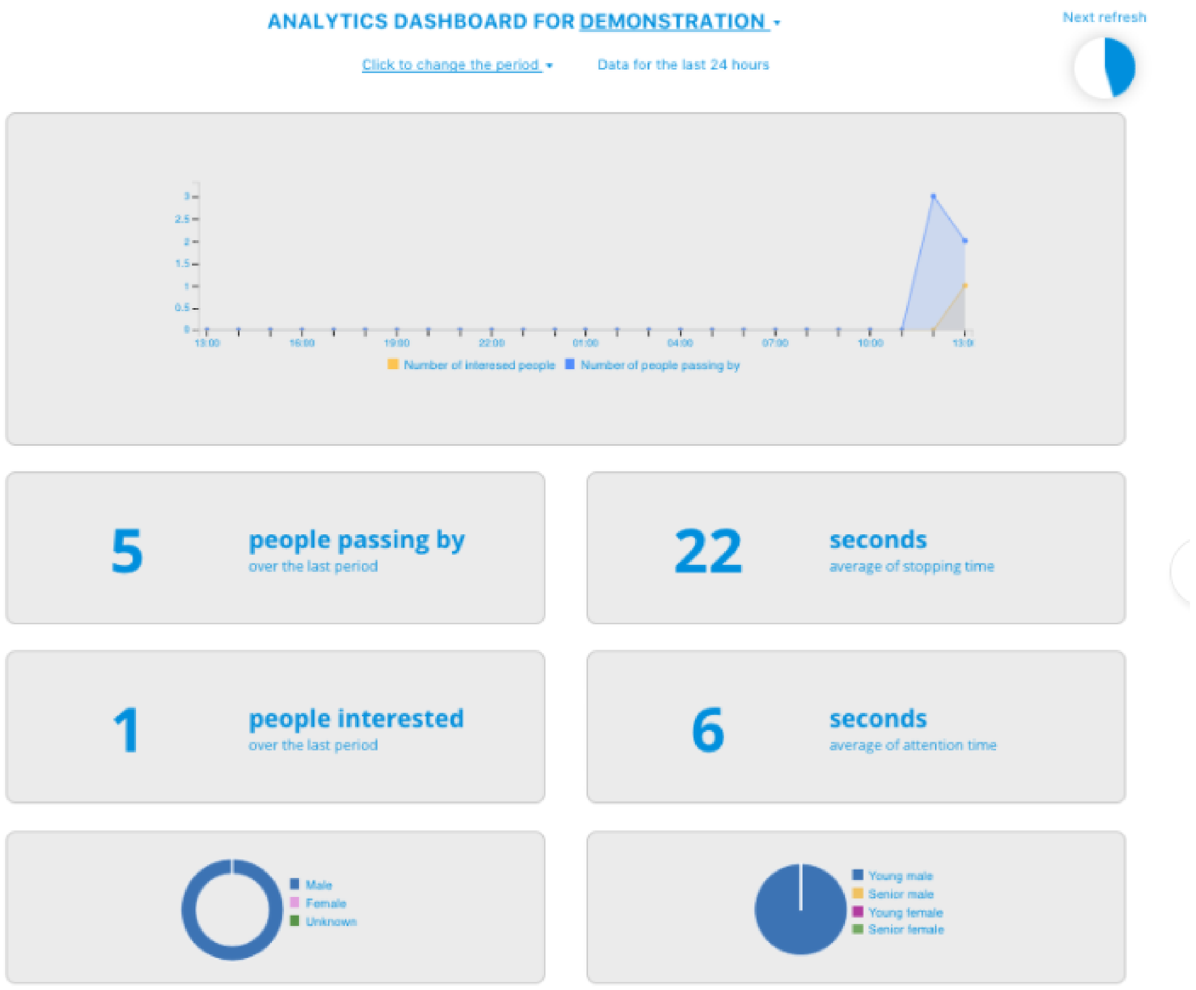
The collected data are meant to be collected programmatically through Angus.ai Data API (see *Retrieve your data*). But for demonstration purposes, we have put together a standard dashboard that allows for a simple visualization over your collected data.

We will use this default dashboard to check that your installation is properly set and that your data are properly stored. But you can also use it for demonstration and even for real world deployment purposes, if it suits your needs.

To view your dashboard:

1. Go back to your personal account here: https://console.angus.ai/

2. Click on the “Show Dashboard” button on the stream you created above.
3. You should see a page showing a dashboard (see example below). If you just launch the client app as explained here (apps), your dashboard might still be empty. Indeed there is about 1min time delay between what happen in front of your camera and the dashboard refreshing for these data. After waiting for the next automatic refresh (see the watch icon on the top right hand corner), your first collected data should appear (as shown on the screenshot below).
4. If your don’t see data appear, please try to get out of the camera field of view and re-enter again.


ANALYTICS DASHBOARD FOR DEMONSTRATION

Click to change the period · Data for the last 24 hours

5 people passing by over the last period

22 seconds average of stopping time

1 people interested over the last period

6 seconds average of attention time

What are these metrics?

People passing by: Count of people who passed (not necessarily stopping or looking) in front of the camera for at least 1 second.

People Interested: Count of people who stopped for at least 3 seconds and looked in the direction of the camera more than 1 second, during the specified time duration.

Average stopping time: Average time a person, among the “interested” people (see above), stay still in front of the camera (in second).

Average attention time: Average time a person, among the “interested” people (see above), spend looking at the camera (in second).

Age Pie Chart: Population segmentation counts of all the “interested” people (see above) for each category.

Gender Chart: The gender repartition of all the “interested” people (see above).

Congratulations, you now have a properly running installation of our audience analytics solution.

If you want to retrieve these data programmatically (for further integration into your own dashboard for example), you have got one more step to go.

Step 4 - Retrieve your Data

Here is a short section to help you get started in retrieving your audience data programmatically. Check our API reference for further details. (*Retrieve your data*)

Getting your JWT Token

You need a JSON Web Token (“JWT”) token in order to securely call the data api endpoint. Your personal JWT is provided by programmatically calling the appropriate endpoint documented below.

Please use your angus.ai in the command line below:

- account `username` (it should be your email address)
- Stream `client_id`
- Stream `access_token`

You can find these credentials on http://console.angus.ai.

Request:

```
curl -X POST -H "Content-Type: application/json" -d '{"username": "aurelien.
↪moreau@angus.ai", "client_id": "xxxxx-xxxx-xxxx-xxxx-xxxxxxxxx", "access_
↪token": "xxxxx-xxxx-xxxx-xxxx-xxxxxxxxx"}' https://console.angus.ai/api-
↪token-authstream/
```

You should get a response as shown below, if this is not the case, contact us.

Response:

```
{
  "token": "eyJhbGciOiJIUzI2NiIsInR5dCI6IkpXVCJ9.
↪eyJ1c2VybmFtZSI6ImF1cmVsaWVuLm1vcmVhdUBhbmd1cy5haSIsIm9yaWdfaWF0IjoxNTA1Mzk4MDM4LCJleHAiOjE1D8
↪K70YXQYMAcdeW7dfscFGxUhenoXXGBAQTiWhNv-9cVc"
}
```

Once you obtained your personal JWT, you can start retrieving your data by calling the API endpoints documented in the *Data API Reference* page.

Example

Here is an example of a request for all entities from 5:45 GMT+2, the 2017, September the 3rd until now, using a time bucket of “one day”.

Request:

```
curl -X GET -H 'Authorization: Bearer eyJhbGciOiJIUzI2NiIsInR5dCI6IkpXVCJ9.
↪eyJ1c2VybmFtZSI6ImF1cmVsaWVuLm1vcmVhdUBhbmd1cy5haSIsIm9yaWdfaWFfta0IjoxNTA1Mzk4MDM4LCJleHAiOjE
↪K70YXQYMAcdeW7dfscFGxUhenoXXGBAQTiWhNv-9cVc' 'https://data.angus.ai/api/1/
↪entities?metrics=satisfaction,gender,category,passing_by,interested&from_
↪date=2017-09-03T05%3A45%3A00%2B0200&time=by_day
```

Response:

```
{
    "entities": {
        "2017-09-03T00:00:00+00:00": {
            "category": {
                "senior_female": 0,
                "senior_male": 0,
                "young_female": 0,
                "young_male": 0
            },
            "gender": {
                "?": 0,
                "female": 0,
                "male": 0
            },
            "interested": {
                "value": 0
            },
            "passing_by": {
                "value": 0
            },
        },
        "2017-09-04T00:00:00+00:00": {
            "category": {
                "senior_female": 0,
                "senior_male": 0,
                "young_female": 0,
                "young_male": 8
            },
            "gender": {
                "?": 0,
                "female": 0,
                "male": 10
            },
            "interested": {
                "value": 10
            },
            "passing_by": {
                "value": 18
            },
        },
        "2017-09-05T00:00:00+00:00": {
            "category": {
                "senior_female": 0,
                "senior_male": 0,
                "young_female": 4,
                "young_male": 52
            },
            "gender": {
                "?": 0,
                "female": 4,
                "male": 56
            },
            "interested": {
                "value": 60
            },
            "passing_by": {
                "value": 152
            },
        },
        "2017-09-06T00:00:00+00:00": {
            "category": {
                "senior_female": 0,
                "senior_male": 0,
                "young_female": 0,
                "young_male": 3
            },
            "gender": {
                "?": 0,
                "female": 0,
                "male": 4
            },
            "interested": {
                "value": 4
            },
            "passing_by": {
                "value": 20
            },
        },
        ...
        ...
        ...
        "2017-09-13T00:00:00+00:00": {
            "category": {
                "senior_female": 0,
                "senior_male": 0,
                "young_female": 0,
                "young_male": 0
            },
            "gender": {
                "?": 0,
                "female": 0,
                "male": 0
            },
            "interested": {
                "value": 0
            },
            "passing_by": {
                "value": 0
            },
        },
        "2017-09-14T00:00:00+00:00": {
            "category": {
                "senior_female": 0,
                "senior_male": 0,
                "young_female": 0,
                "young_male": 43
            },
            "gender": {
                "?": 1,
                "female": 0,
                "male": 59
            },
            "interested": {
                "value": 60
            },
            "passing_by": {
                "value": 153
            },
        }
    },
    "from_date": "2017-09-03T05:45:00+02:00",
    "total_results": 12,
    "nb_of_pages": 1,
    "next_page": "",
    "page": 1,
    "time": "by_day",
    "to_date": "2017-09-14T16:53:11+02:00"
}
```

What next?

You have a running installation of Angus.ai audience analytics solution. Congratulations!

- When time comes, you can plug more cameras by creating additional stream as shown here (create-stream).
- If you need to deploy your system in a situation where internet bandwidth is a problem, or for any issues please contact Angus.ai team at: support@angus.ai, and if possible, please specify your operating system, python version, as well as the error backtrace if any. Thanks!

CHAPTER 2

Full API Reference

Main API Reference

Introduction and RESTful architecture

Angus.ai provides a RESTful API for its services. That implies:

- Use of HTTP protocol
- A standard protocol for encryption: ssl (HTTPs)
- A resource oriented programmation style
- A common resource representation: JSON
- A linked resources approach

In the rest of this documentation, we will use command line curl to interact with angus.ai gateway and present each of these features, one by one.

Encryption and Authentication

All requests to an angus.ai gateway needs to be done through Basic Authentication and https protocol (http over ssl).

As a user, you need to signup first at https://console.angus.ai/register to get your credentials.

These credentials are an equivalent of a login/password but for a device.

If you do not have your credentials yet, you can use the following ones for this tutorial:

- client id: 7f5933d2-cd7c-11e4-9fe6-490467a5e114
- access token: db19c01e-18e5-4fc2-8b81-7b3d1f44533b

To check your credentials you can make a simple **GET** request on service list resource https://gate.angus.ai/services (we will see the content of this resource in *Service directory*). Curl accepts the option `-u` that computes the value for the `Authorization` HTTP header in order to conform to Basic Authentication protocol.

```
$ curl -u 7f5933d2-cd7c-11e4-9fe6-490467a5e114:db19c01e-18e5-4fc2-8b81-7b3d1f44533b \
  -s -o /dev/null -w "%{http_code}" \
  https://gate.angus.ai/services

200
```

You just made your first call to angus.ai and got the response code `200`. All communications were encrypted (because we use https protocol) and you were authenticated thanks to your credentials.

Resources

Angus.ai provides a "resource oriented" API. Each image, piece of sound, document and other assets are represented as a resource with at least one URL. Currently, most angus.ai resources only have a JSON representation.

This means that when you get a resource (with **GET**) from angus.ai, only the value `application/json` is available for the HTTP header `Accept`. The response body will be a JSON object.

You can have a look at the body of a response by, for example, using the previous curl command and removing the extra options:

```
$ curl -u 7f5933d2-cd7c-11e4-9fe6-490467a5e114:db19c01e-18e5-4fc2-8b81-7b3d1f44533b \
  https://gate.angus.ai/services

{
  "services": {
     "dummy": {
        "url": "/services/dummy"
     },

    (...)

     "face_detection": {
        "url": "/services/face_detection"}
     }
}
```

This response body is a JSON object, its content is not important right now, we will describe it in the next sections.

Service directory

We chose to follow the HATEOAS constraints by linking resources via URLs provided dynamically instead of providing an a priori description of all resources with their URLs.

But you must have an entry point to start the navigation. The entry point for angus.ai services is https://gate.angus.ai/services. This resource describes a service directory. By requesting it, you get back a list of available services provided by angus.ai.

```
$ curl -u 7f5933d2-cd7c-11e4-9fe6-490467a5e114:db19c01e-18e5-4fc2-8b81-7b3d1f44533b \
  https://gate.angus.ai/services

{
 "services": {
     "face_expression_estimation": {
        "url": "/services/face_expression_estimation"
     },
     "dummy": {
        "url": "/services/dummy"
     },
     "gaze_analysis": {
        "url": "/services/gaze_analysis"
     },
     "motion_detection": {
        "url": "/services/motion_detection"
     },
     "age_and_gender_estimation": {
        "url": "/services/age_and_gender_estimation"
     },
     "sound_localization": {
        "url": "/services/sound_localization"
     },
     "face_detection": {
        "url": "/services/face_detection"
     }
  }
}
```

This request reveals for example a service named `dummy`. A service is a resource too, so let's `get` it:

```
$ curl -u 7f5933d2-cd7c-11e4-9fe6-490467a5e114:db19c01e-18e5-4fc2-8b81-7b3d1f44533b \
  https://gate.angus.ai/services/dummy

{
  "versions": {
     "1": {"url": "/services/dummy/1"}
  }
}
```

The response shows that there is only one version of the dummy service. Let's continue and `get` the new given url:

```
$ curl -u 7f5933d2-cd7c-11e4-9fe6-490467a5e114:db19c01e-18e5-4fc2-8b81-7b3d1f44533b \
  https://gate.angus.ai/services/dummy/1

{
  "url": "https://gate.angus.ai/services/dummy/1",
  "version": 1,
  "description": "\nA simple dummy service. You can send {\"echo\": \"Hello world\"}␣
↪to get back the\nmessage \"Hello world\" as result. Moreover, the dummy service␣
↪enables statefull\nfeatures",
  "jobs": "https://gate.angus.ai/services/dummy/1/jobs",
}
```

We started at the entry endpoint of service directory and finaly got an endpoint on a "jobs" resource.

In the next section we will see how to use this resource to request new compute to angus.ai.

Jobs (compute)

The previous "jobs" resource is a collection of job resources.

As a user, you can create a new job by using a **POST** request on it.

To make a valid request you must comply with these constraints:

- the body of the request must be a JSON message whose format matches the documentation of the service

- the `Content-Type` header of the request must be set to `application/json`
- you must specify the synchronous or asynchronous type of request you wish to make. Please see *Asynchronous call* for more details

The new curl command is as follows:

```
$ curl -u 7f5933d2-cd7c-11e4-9fe6-490467a5e114:db19c01e-18e5-4fc2-8b81-7b3d1f44533b \
  -H "Content-Type: application/json" \
  -d '{ "echo": "Hello world!", "async": false}' \
  https://gate.angus.ai/services/dummy/1/jobs

{
  "url": "https://gate.angus.ai/services/dummy/1/jobs/db77e78e-0dd8-11e5-a743-
↪19d95545b6ca",
  "status": 201,
  "echo": "Hello world!"
}
```

The response contains an absolute url on the resource (the job), its status (201 : **CREATED**), and its result as a synchronous job has been requested.

Note that an new url is provided to get back later on the job (accessing its result in an async way for example).

```
$ curl -u 7f5933d2-cd7c-11e4-9fe6-490467a5e114:db19c01e-18e5-4fc2-8b81-7b3d1f44533b \
  https://gate.angus.ai/services/dummy/1/jobs/db77e78e-0dd8-11e5-a743-19d95545b6ca

{
  "url": "https://gate.angus.ai/services/dummy/1/jobs/db77e78e-0dd8-11e5-a743-
↪19d95545b6ca",
  "status": 201,
  "echo": "Hello world!"
}
```

Asynchronous call

All job requests are asynchronous by default if no `async` parameter is set.

```
$ curl -u 7f5933d2-cd7c-11e4-9fe6-490467a5e114:db19c01e-18e5-4fc2-8b81-7b3d1f44533b \
  -H "Content-Type: application/json" \
  -d '{ "echo": "Hello world!"}' \
  https://gate.angus.ai/services/dummy/1/jobs

{
  "url": "https://gate.angus.ai/services/dummy/1/jobs/db77e78e-0dd8-11e5-a743-
↪19d95545b6ca",
  "status": 202,
}
```

The response status is `202` for HTTP status code **ACCEPTED**, and the replied url allows to get back to the result in the future.

```
$ curl -u 7f5933d2-cd7c-11e4-9fe6-490467a5e114:db19c01e-18e5-4fc2-8b81-7b3d1f44533b \
  https://gate.angus.ai/services/dummy/1/jobs/db77e78e-0dd8-11e5-a743-19d95545b6ca

{
  "url": "https://gate.angus.ai/services/dummy/1/jobs/db77e78e-0dd8-11e5-a743-
↪19d95545b6ca",
  "status": 200,
  "echo": "Hello world!"
}
```

If you want a synchronous job with the result, you must specify `async` as `false`.

```
$ curl -u 7f5933d2-cd7c-11e4-9fe6-490467a5e114:db19c01e-18e5-4fc2-8b81-7b3d1f44533b \
  -H "Content-Type: application/json" \
  -d '{ "echo": "Hello world!", "async": false}' \
  https://gate.angus.ai/services/dummy/1/jobs

{
  "url": "https://gate.angus.ai/services/dummy/1/jobs/db77e78e-0dd8-11e5-a743-
↪19d95545b6ca",
  "status": 201,
  "echo": "Hello world!"
}
```

Binary attachment

Most requests to Angus.ai will need you to attach binary files for sound, images, videos or other raw data from various sensors. Angus.ai provides two ways to upload them:

- attached in the request
- or by referring to a previously created resource

Make a request with an attached binary file

You need to create a multipart request to send binary file to angus.ai as follows:

- the name and type of the binary part are specified with: `attachment://<name_of_the_resource>`
- the JSON body part is prefixed with `meta`
- the JSON body part refers to the attachement `attachment://<name_of_the_resource`

For example, the service `face_detection` must be provided an image as input. You can upload it as an attachment as follows:

```
$ curl -u 7f5933d2-cd7c-11e4-9fe6-490467a5e114:db19c01e-18e5-4fc2-8b81-7b3d1f44533b  \
  -F "attachment://bar=@macgyver.jpg;type=image/jpg" \
  -F 'meta={"async" : false, "image": "attachment://bar"};type=application/json' \
  https://gate.angus.ai/services/face_detection/1/jobs

{
  "url": "https://gate.angus.ai/services/face_detection/1/jobs/1944556c-baf8-11e5-
↪85c3-0242ac110001",
  "status": 201,
  "input_size": [480, 640],
  "nb_faces": 1,
  "faces": [{"roi": [262, 76, 127, 127], "roi_confidence": 0.8440000414848328}]
}
```

Create a binary resource

Angus.ai provides a “blob storage” to upload a binary resource once and use it later for one or more services. This service is available at https://gate.angus.ai/blobs.

Binaries need to be sent as an attachement to the request (as shown above), made on the “blob storage” resource. The JSON body part needs to contain a key `content` whose value matches the attached file.

```
$ curl -u 7f5933d2-cd7c-11e4-9fe6-490467a5e114:db19c01e-18e5-4fc2-8b81-7b3d1f44533b \
  -F "attachment://bar=@macgyver.jpg;type=image/jpg" \
  -F 'meta={"async": false, "content": "attachment://bar"};type=application/json' \
  https://gate.angus.ai/blobs

{
  "status": 201,
  "url": "https://gate.angus.ai/blobs/a5bca2da-baf6-11e5-ad97-0242ac110001"
}
```

The response contains the url of the new blob resource created. You can now use this (binary) resource it in all angus.ai services by referring to it in your requests:

```
$ curl -u 7f5933d2-cd7c-11e4-9fe6-490467a5e114:db19c01e-18e5-4fc2-8b81-7b3d1f44533b \
  -F 'meta={"async": false, "image": "https://gate.angus.ai/blobs/a5bca2da-baf6-11e5-
↪ad97-0242ac110001"};type=application/json' \
  https://gate.angus.ai/services/face_detection/1/jobs

{
  "url": "http://localhost/services/face_detection/1/jobs/1944556c-baf8-11e5-85c3-
↪0242ac110001",
  "status": 201,
  "input_size": [480, 640],
  "nb_faces": 1,
  "faces": [{"roi": [262, 76, 127, 127], "roi_confidence": 0.8440000414848328}]
}
```

Session / State

Despite angus.ai API aiming at RESTful and hence stateless services, some services can currently and optionally be made statefull.

In that case, the state is kept by the client and attached with each request in a `state` JSON parameter. For the statefull services, states are currently represented as `session_id` generated on the client side.

In followed example, we generate a uuid session id with the `uuidgen` linux tool and we loop 4 times over the same image that contains a face and send it to the face detection service.

```
$ export SESSION=`uuidgen`
> for i in `seq 1 4`; do
>   curl -su 7f5933d2-cd7c-11e4-9fe6-490467a5e114:db19c01e-18e5-4fc2-8b81-
↪7b3d1f44533b \
>        -F "attachment://bar=@macgyver.jpg;type=image/jpg" \
>        -F 'meta={"async" : false, "image": "attachment://bar", "state": { "session_
↪id": "'$SESSION'"}};type=application/json' \
>        https://gate.angus.ai/services/face_detection/1/jobs | python -m json.tool |␣
↪grep "nb_faces"
> done;
```

```
"nb_faces": 0
"nb_faces": 0
"nb_faces": 0
"nb_faces": 1
```

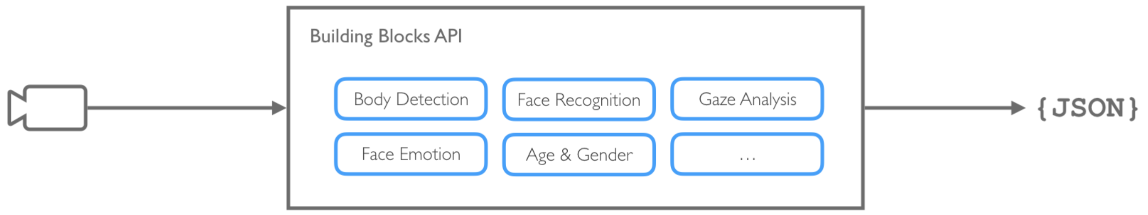
When a session is requested, the service try to track faces in sucessive images but returns no result at first time. Then, we can notice, the three first calls have 0 face result but the fourth one (for the same image) find a face. That validates the session id parameter is taken into account.

Building Blocks

Tutorial

Step 1 - Introduction

This documentation is meant at developers wanting to use Angus.ai building blocks API services.



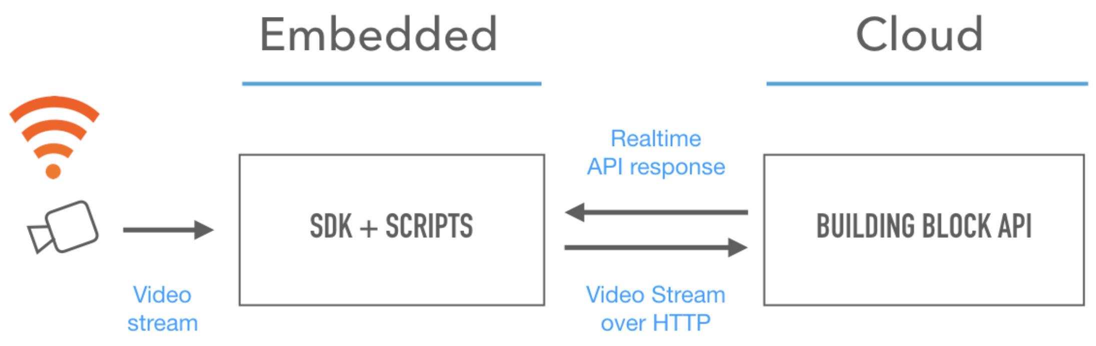
What the difference with other AI building blocks providers?

Angus.ai is focus 100% on turning existing 2D cameras into a new generation of monitoring and alerting tools, as a consequence these building blocks are optimized to work:

- on video streams
- in realtime
- and with low resolution 2D cameras

How it works

Angus.ai audience analytics solution is based on a (lightweight) Client / Server architecture as seen on the figure below. All CPU expensive computation are made on our dedicated servers making it possible to run the solution from about any CPU board that can retrieve a camera stream and connect to a server (eg. Raspberry).



List of the available building blocks

Blocks

Scene Analysis

This is Angus.ai main API that is meant to help you leverage your video streams by extracting:

- how many people are visible?
- who is looking at what?
- are people only passing by or do they stop?
- do they look happy?
- what are their age and gender?
- etc...

Besides the code samples provided on this page, you can find a first way to use the API on GitHub here

Getting Started

Using the Python SDK:

```
# -*- coding: utf-8 -*-
import angus.client
from pprint import pprint

conn = angus.client.connect()
service = conn.services.get_service("scene_analysis", version=1)

service.enable_session()

while True:
    job = service.process({"image": open("./image.jpg", 'rb'),
                           "timestamp" : "2016-10-26T16:21:01.136287+00:00",
                           "camera_position": "ceiling",
                           "sensitivity": {
                                            "appearance": 0.7,
                                            "disappearance": 0.7,
                                            "age_estimated": 0.4,
                                            "gender_estimated": 0.5,
                                            "focus_locked": 0.9,
                                            "emotion_detected": 0.4,
                                            "direction_estimated" : 0.8
                                          }
                          })
    pprint(job.result)


service.disable_session()
```

Input

The API takes a stream of 2d still images as input, of format "jpg" or "png", without constraints on resolution. However, 640p x 480p tends to be a good trade-off for both precision/recall and latencies.

Note also that the bigger the resolution, the longer the API will take to process and give a result. The function `process()` takes a dictionary as input formatted as follows:

```
{
  "image" : binary file,
  "timestamp" : "2016-10-26T16:21:01.136287+00:00",
  "camera_position" : "ceiling" or "facing",
  "sensitivity": {
                    "appearance": 0.7,
                    "disappearance": 0.7,
                    "age_estimated": 0.4,
                    "gender_estimated": 0.5,
                    "focus_locked": 0.9,
                    "emotion_detected": 0.4,
                    "direction_estimated" : 0.8
                 },
}
```

- `image`: a python `File Object` returned for example by `open()` or a `StringIO` buffer.
- `timestamp`: a string formated using the iso 8601 UTC date format.
- `camera_position`: a preset is a list of parameters set in advance. This list of parameters is used to calibrate the API based on the camera position.
- `sensitivity`: an optional dictionary that sets the sensitivity (between 0 and 1) of the system regarding each events. For instance, If you feel that the events "appearance" is triggered too often, you can decrease its value.
- `store`: store process results (not video data) for analytics dashboard (Beta)

Here is the list of the different presets that are available :

- `ceiling`: this preset has to be used if the camera is a ceiling camera or if it is placed at ceiling height.
- `facing`: this preset has to be used if the camera is placed at human height.

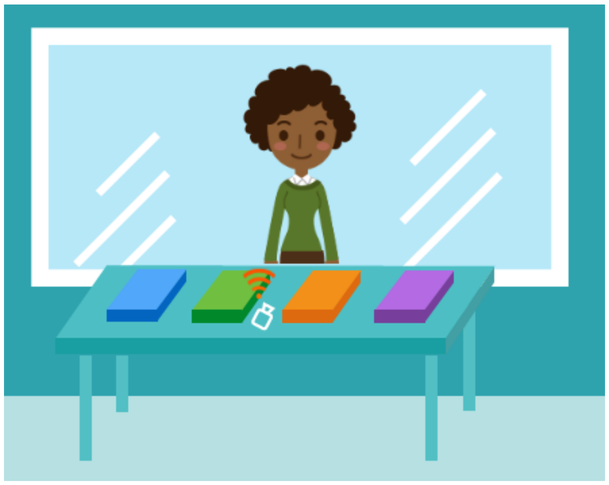


Fig. 2.1: The “facing” preset should be used in this situation

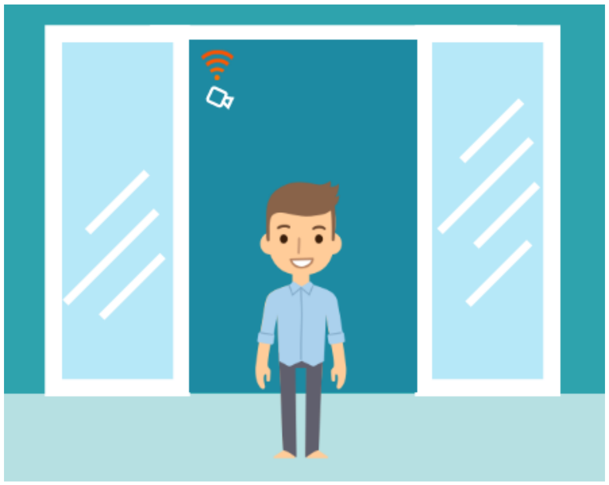


Fig. 2.2: The “ceiling” preset should be used in this situation

Output API

Events will be pushed to your client following that format. Note that if nothing happened, the events list will be empty, but the timestamp will still be updated.

```
{
  "timestamp" : "2016-10-26T16:21:01.136287+00:00",
  "events" : [
              {
                "entity_id" : "16fd2706-8baf-433b-82eb-8c7fada847da",
                "entity_type" : "human",
                "type" : "age_estimated",
                "confidence" : 0.96,
                "key" : "age"
              }
            ],
  "entities" : {"16fd2706-8baf-433b-82eb-8c7fada847da":
                          {
                           "face_roi": [339, 264, 232, 232],
                           "face_roi_confidence": 0.71,
                           "full_body_roi": [59, 14, 791, 1798],
                           "full_body_roi_confidence": 0.71,

                           "age": 25,
                           "age_confidence": 0.34,

                           "gender": "male",
                           "gender_confidence": 0.99,

                           "emotion_anger": 0.04,
                           "emotion_surprise": 0.06,
                           "emotion_sadness": 0.14,
                           "emotion_neutral": 0.53,
                           "emotion_happiness": 0.21,
                           "emotion_smiling_degree": 0.42,
                           "emotion_confidence": 0.37,

                           "face_eye": [[414, 346], [499, 339]],
                           "face_mouth": [456, 401],
                           "face_nose": [456, 401],
                           "face_confidence": 0.37,

                           "gaze": [0.02, 0.14],
                           "gaze_confidence": 0.37,

                           "head": [-0.1751, -0.0544, -0.0564]
                           "head_confidence": 0.3765,

                           "direction": "unknown"
                          }
            }
}
```

- `timestamp`: a string formated using the iso 8601 UTC date format.
- `entity_id` : id of the human related to the event.
- `entity_type` : type of the entity, only “human” is currently supported

- `type` : type of the event, a list of event types can be found below.
- `confidence` : a value between 0 and 1 which reflects the probability that the event has really occurred in the scene.
- `key` : a string which indicates which value has been updated in the attached entities list.
- `face_roi` : contains `[pt.x, pt.y, width, height]` where pt is the upper left point of the rectangle outlining the detected face.
- `face_roi_confidence` : an estimate of the probability that a real face is indeed located at the given `roi`.
- `full_body_roi` : contains `[pt.x, pt.y, width, height]` where pt is the upper left point of the rectangle outlining the detected human body.
- `full_body_roi_confidence` : an estimate of the probability that a real human body is indeed located at the given `roi`.
- `age` : an age estimate (in years) of the person outlined by `roi`.
- `age_confidence` : an estimate of the probability that the outlined person is indeed of age `age`.
- `gender` : an estimation of the gender of the person outlined by `roi`. Value is either `"male"` or `"female"`.
- `gender_confidence` : an estimate of the probability that the outlined person is indeed of gender `gender`.
- `emotion_neutral`, `emotion_happiness`, `emotion_surprise`, `emotion_anger`, `emotion_sadness` : a float in `[0, 1]` measuring the intensity of the corresponding face expression.
- `face_eye`, `face_mouth`, `face_nose` : the coordinate of the detected eyes, nose and mouth in pixels.
- `head` : head pose orientation (yaw, pitch and roll) in radian
- `gaze` : gaze orientation (yaw, pitch) in radian
- `direction` : an indication of the average direction of the person. Value is either `"unknown"`, `"up"`, `"right"`, `"left"` or `"down"`.

The list of the possible events :

- `"appearance"` : a new human has just been detected.
- `"disappearance"` : a known human has just disappeared.
- `"age_estimated"` : the age of the corresponding human has just been estimated, (expect 1 or 2 events of this type for each human)
- `"gender_estimated"` : gender estimation of the corresponding human. (expect 1 or 2 events of this type for each human)
- `"focus_locked"` : if a human look in a specific direction for a significant time, this event is triggered with the pitch and yaw of the gaze registered in the data.
- `"emotion_detected"` : if a remarkable emotion peak is detected, the event is triggered with the related emotion type registered in the data.
- `"direction_estimated"` : if the human stays enough time in order to determine his average direction.

Code Sample

requirements: opencv2, opencv2 python bindings

This code sample retrieves the stream of a web cam and display in a GUI the result of the `scene_analysis` service.

```
# -*- coding: utf-8 -*-
import StringIO

import angus.client
import cv2
import numpy as np
import datetime
import pytz

def main(stream_index):
    camera = cv2.VideoCapture(stream_index)
    camera.set(cv2.cv.CV_CAP_PROP_FRAME_WIDTH, 640)
    camera.set(cv2.cv.CV_CAP_PROP_FRAME_HEIGHT, 480)
    camera.set(cv2.cv.CV_CAP_PROP_FPS, 10)

    if not camera.isOpened():
        print("Cannot open stream of index {}".format(stream_index))
        exit(1)

    print("Input stream is of resolution: {} x {}".format(camera.get(3), camera.
↪get(4)))

    conn = angus.client.connect()
    service = conn.services.get_service("scene_analysis", version=1)
    service.enable_session()

    while camera.isOpened():
        ret, frame = camera.read()
        if not ret:
            break

        gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
        ret, buff = cv2.imencode(".jpg", gray, [cv2.IMWRITE_JPEG_QUALITY, 80])
        buff = StringIO.StringIO(np.array(buff).tostring())

        t = datetime.datetime.now(pytz.utc)
        job = service.process({"image": buff,
                               "timestamp" : t.isoformat(),
                               "camera_position": "facing",
                               "sensitivity": {
                                   "appearance": 0.7,
                                   "disappearance": 0.7,
                                   "age_estimated": 0.4,
                                   "gender_estimated": 0.5,
                                   "focus_locked": 0.9,
                                   "emotion_detected": 0.4,
                                   "direction_estimated": 0.8
                               },
        })
        res = job.result

        if "error" in res:
            print(res["error"])
        else:
            # This parses the events
            if "events" in res:
                for event in res["events"]:
                    value = res["entities"][event["entity_id"]][event["key"]]
                    print("{}| {}, {}".format(event["type"],
                                              event["key"],
                                              value))

            # This parses the entities data
            for key, val in res["entities"].iteritems():
                x, y, dx, dy = map(int, val["face_roi"])
                cv2.rectangle(frame, (x, y), (x+dx, y+dy), (0, 255, 0), 2)

        cv2.imshow("original", frame)
        if cv2.waitKey(1) & 0xFF == 27:
            break

    service.disable_session()

    camera.release()
    cv2.destroyAllWindows()

if __name__ == "__main__":
    ### Web cam index might be different from 0 on your setup.
    ### To grab a given video file instead of the host computer cam, try:
    ### main("/path/to/myvideo.avi")
    main(0)
```

Upper Body Detection

Do I see a human? How many? Where? As opposed to the Face Detection service, this service is able to detect a human even if his/her face is not visible.

Getting Started

Using Angus python SDK:

```
# -*- coding: utf-8 -*-
import angus.client
from pprint import pprint

conn = angus.client.connect()
service = conn.services.get_service('upper_body_detection', version=1)
job = service.process({'image': open('./macgyver.jpg', 'rb')})
pprint(job.result)
```

Input

The API takes a stream of 2d still images as input, of format `jpg` or `png`, without constraints on resolution.

Note however that the bigger the resolution, the longer the API will take to process and give a result.

The function `process()` takes a dictionary as input formatted as follows:

```
{'image' : file}
```

- `image`: a python `File Object` as returned for example by `open()` or a `StringIO` buffer.

Output

Events will be pushed to your client following that format:

```
{
  "input_size" : [480, 640],
  "nb_upper_bodies" : 2,
  "upper_bodies" : [
                {
                  "upper_body_roi" : [345, 223, 34, 54],
                  "upper_body_roi_confidence" : 0.89
                },
                {
                  "upper_body_roi" : [35, 323, 45, 34],
                  "upper_body_roi_confidence" : 0.56
                }
             ]
}
```

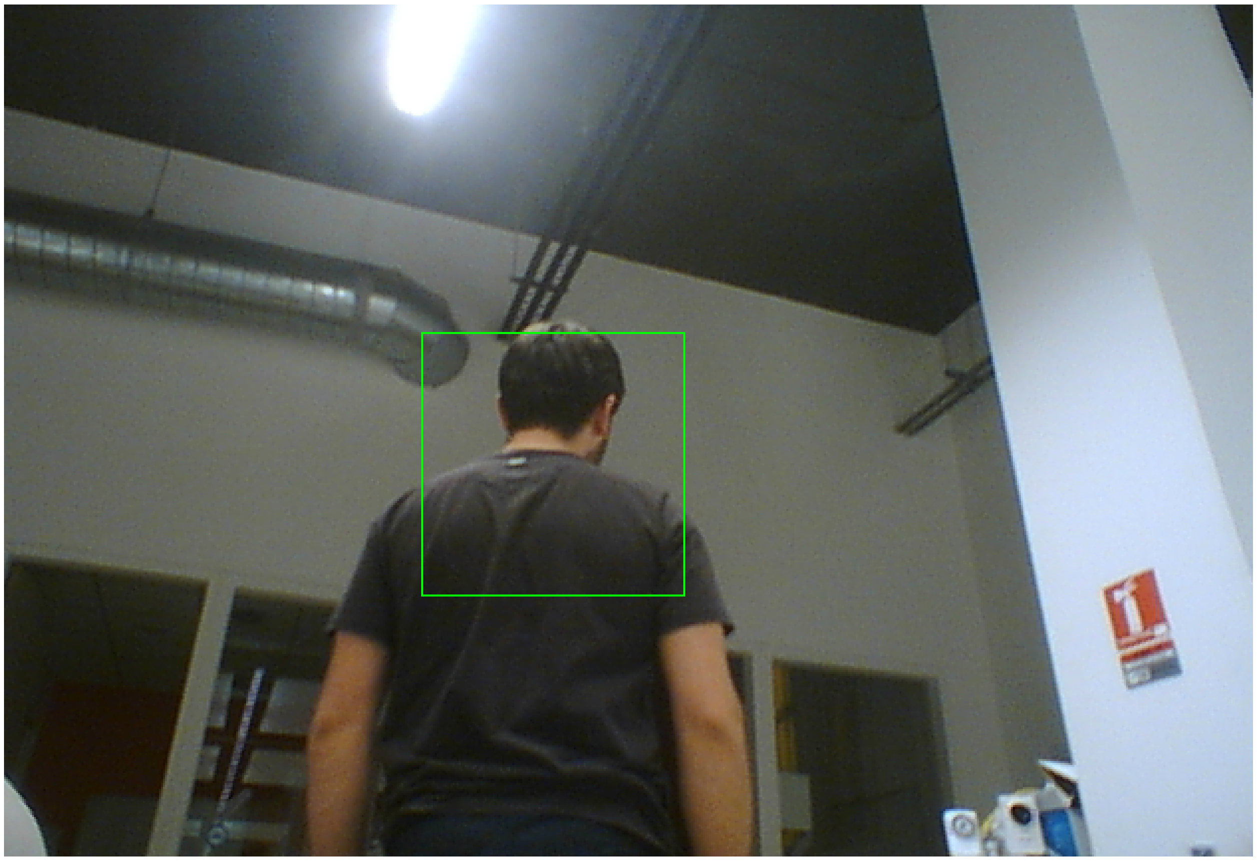
- `input_size` : width and height of the input image in pixels (to be used as reference to `roi` output.
- `nb_upper_bodies` : number of upper bodies detected in the given image
- `upper_body_roi` : contains `[pt.x, pt.y, width, height]` where pt is the upper left point of the rectangle outlining the detected upper body.
- `upper_body_roi_confidence` : an estimate of the probability that a real human upper body is indeed located at the given `upper_body_roi`.

Code Sample

requirements: opencv2, opencv2 python bindings

This code sample retrieves the stream of a web cam and display in a GUI the result of the `upper_body_detection` service.

```
# -*- coding: utf-8 -*-
import StringIO
import cv2
import numpy as np
from pprint import pprint
import angus.client

def main(stream_index):
    camera = cv2.VideoCapture(stream_index)
    camera.set(cv2.cv.CV_CAP_PROP_FRAME_WIDTH, 640)
    camera.set(cv2.cv.CV_CAP_PROP_FRAME_HEIGHT, 480)
    camera.set(cv2.cv.CV_CAP_PROP_FPS, 10)

    if not camera.isOpened():
        print("Cannot open stream of index {}".format(stream_index))
        exit(1)

    print("Input stream is of resolution: {} x {}".format(camera.get(3), camera.
↪get(4)))

    conn = angus.client.connect()
    service = conn.services.get_service("upper_body_detection", version=1)
    service.enable_session()

    while camera.isOpened():
        ret, frame = camera.read()
        if not ret:
            break

        gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
        ret, buff = cv2.imencode(".jpg", gray, [cv2.IMWRITE_JPEG_QUALITY, 80])

        buff = StringIO.StringIO(np.array(buff).tostring())

        job = service.process({"image": buff})
        res = job.result
        pprint(res)

        for body in res['upper_bodies']:
            x, y, dx, dy = body['upper_body_roi']
            cv2.rectangle(frame, (x, y), (x+dx, y+dy), (0,255,0))

        cv2.imshow('original', frame)
        if cv2.waitKey(1) & 0xFF == ord('q'):
            break

    service.disable_session()

    camera.release()
    cv2.destroyAllWindows()

if __name__ == '__main__':
    ### Web cam index might be different from 0 on your setup.
    ### To grab a given video file instead of the host computer cam, try:
    ### main("/path/to/myvideo.avi")
    main(0)
```



Age and Gender Estimation

How old are people in front of my object? Are they male or female?

Getting Started

Using the Python SDK:

```
# -*- coding: utf-8 -*-
import angus.client
from pprint import pprint

conn = angus.client.connect()
service = conn.services.get_service('age_and_gender_estimation', version=1)
job = service.process({'image': open('./macgyver.jpg', 'rb')})

pprint(job.result)
```

Input

The API takes a stream of 2d still images as input, of format `jpg` or `png`, without constraints on resolution.

Note however that the bigger the resolution, the longer the API will take to process and give a result.

The function `process()` takes a dictionary as input formatted as follows:

```
{'image' : file}
```

- `image`: a python `File Object` as returned for example by `open()` or a `StringIO` buffer.

Output

Events will be pushed to your client following that format:

```
{
  "input_size" : [480, 640],
  "nb_faces" : 1,
  "faces" : [
              {
                "roi" : [345, 223, 34, 54],
                "roi_confidence" : 0.89,
                "age" : 32,
                "age_confidence" :0.87,
                "gender" : "male",
                "gender_confidence" : 0.95
              }
            ]
}
```

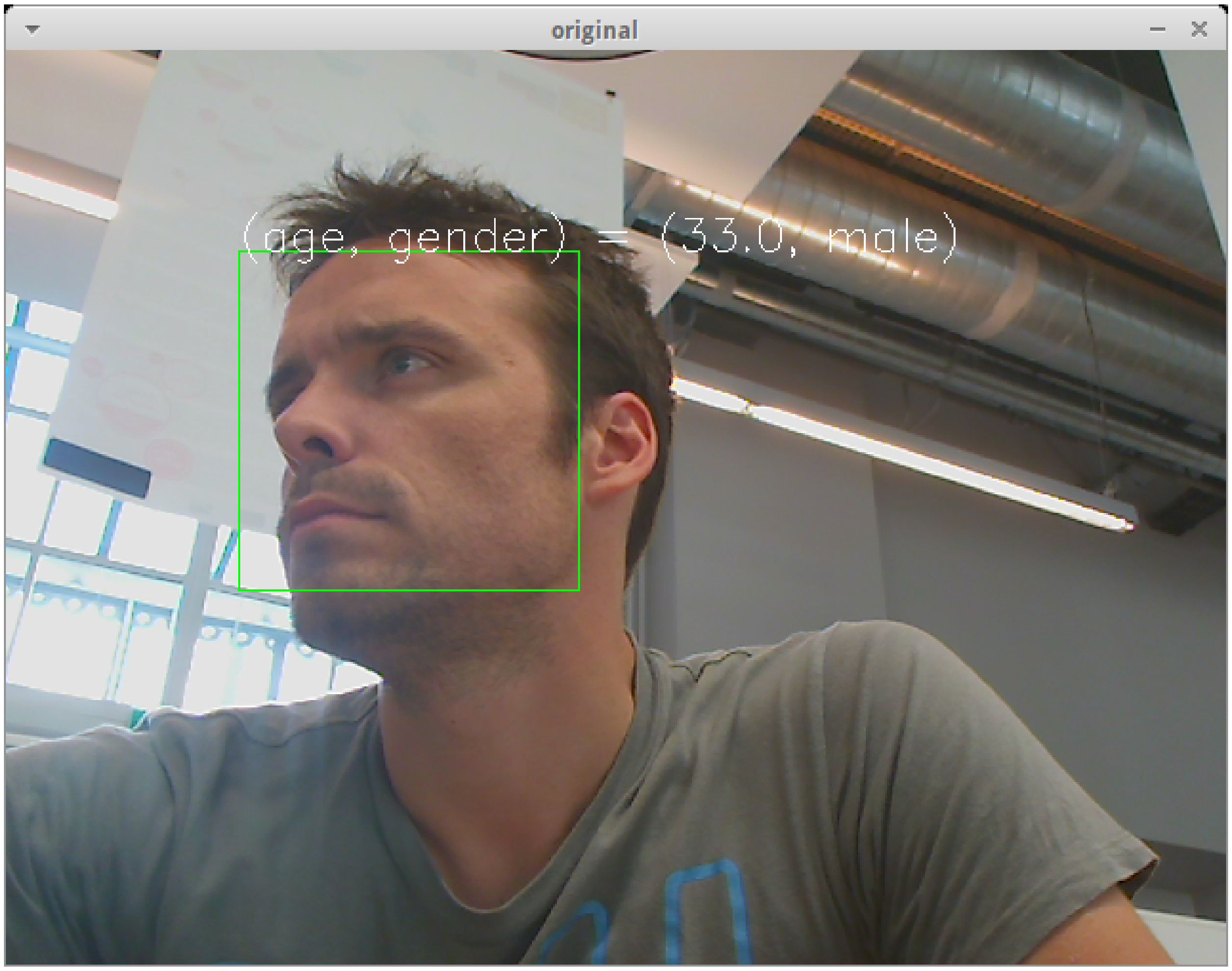
- `input_size` : width and height of the input image in pixels (to be used as reference to `roi` output.
- `nb_faces` : number of faces detected in the given image
- `roi` : contains `[pt.x, pt.y, width, height]` where pt is the upper left point of the rectangle outlining the detected face.
- `roi_confidence` : an estimate of the probability that a real face is indeed located at the given `roi`.
- `age` : an age estimate (in years) of the person outlined by `roi`.
- `age_confidence` : an estimate of the probability that the outlined person is indeed of age `age`.
- `gender` : an estimation of the gender of the person outlined by `roi`. Value is either `"male"` or `"female"`.
- `gender_confidence` : an estimate of the probability that the outlined person is indeed of gender `gender`.

Code Sample

requirements: opencv2, opencv2 python bindings

This code sample retrieves the stream of a web cam and display in a GUI the result of the `age_and_gender_estimation` service.

```python
# -*- coding: utf-8 -*-
import cv2
import numpy as np
import StringIO
import angus.client

def main(stream_index):
    camera = cv2.VideoCapture(stream_index)
    camera.set(cv2.cv.CV_CAP_PROP_FRAME_WIDTH, 640)
    camera.set(cv2.cv.CV_CAP_PROP_FRAME_HEIGHT, 480)
    camera.set(cv2.cv.CV_CAP_PROP_FPS, 10)
    if not camera.isOpened():
        print("Cannot open stream of index {}".format(stream_index))
        exit(1)

    print("Video stream is of resolution {} x {}".format(camera.get(3), camera.
↪get(4)))

    conn = angus.client.connect()
    service = conn.services.get_service("age_and_gender_estimation", version=1)
    service.enable_session()

    while camera.isOpened():
        ret, frame = camera.read()

        if not ret:
            break

        gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
        ret, buff = cv2.imencode(".jpg", gray,  [cv2.IMWRITE_JPEG_QUALITY, 80])
        buff = StringIO.StringIO(np.array(buff).tostring())

        job = service.process({"image": buff})
        res = job.result

        for face in res['faces']:
            x, y, dx, dy = face['roi']
            age = face['age']
            gender = face['gender']

            cv2.rectangle(frame, (x, y), (x+dx, y+dy), (0,255,0))
            cv2.putText(frame, "(age, gender) = ({:.1f}, {})".format(age, gender),
                        (x, y), cv2.FONT_HERSHEY_SIMPLEX,
                        0.8, (255, 255, 255))

        cv2.imshow('original', frame)
        if cv2.waitKey(1) & 0xFF == ord('q'):
            break

    service.disable_session()

    camera.release()
    cv2.destroyAllWindows()

if __name__ == '__main__':
    ### Web cam index might be different from 0 on your setup.
    ### To grab a given video file instead of the host computer cam, try:
    ### main("/path/to/myvideo.avi")
    main(0)
```



Face Expression Estimation

Are people in front looking happy or surprised?

Getting Started

Using Angus Python SDK:

```python
# -*- coding: utf-8 -*-
from pprint import pprint
import angus.client

conn = angus.client.connect()
service = conn.services.get_service('face_expression_estimation', version=1)
job = service.process({'image': open('./macgyver.jpg', 'rb')})
pprint(job.result)
```

Input

The API takes a stream of 2d still images as input, of format `jpg` or `png`, without constraints on resolution.

Note however that the bigger the resolution, the longer the API will take to process and give a result.

The function `process()` takes a dictionary as input formatted as follows:

```
{'image' : file}
```

- `image`: a python `File Object` as returned for example by `open()` or a `StringIO` buffer.

Output

Events will be pushed to your client following that format:

```
{
  "input_size" : [480, 640],
  "nb_faces" : 1,
  "faces" : [
              {
                "roi" : [345, 223, 34, 54],
                "roi_confidence" : 0.89,
                "neutral" : 0.1,
                "happiness" : 0.2,
                "surprise" : 0.7,
                "anger" : 0.01,
                "sadness" : 0.1,
              }
            ]
}
```

- `input_size` : width and height of the input image in pixels (to be used as reference to `roi` output.
- `nb_faces` : number of faces detected in the given image
- `roi` : contains `[pt.x, pt.y, width, height]` where pt is the upper left point of the rectangle outlining the detected face.
- `roi_confidence` : an estimate of the probability that a real face is indeed located at the given `roi`.
- `neutral`, `happiness`, `surprise`, `anger`, `sadness` : a float in `[0, 1]` measuring the intensity of the corresponding face expression.

Code Sample

requirements: opencv2, opencv2 python bindings

This code sample retrieves the stream of a web cam and display in a GUI the result of the `face_expression_estimation` service.

```
# -*- coding: utf-8 -*-
import StringIO
import cv2
import numpy as np
import angus.client

def main(stream_index):
    camera = cv2.VideoCapture(stream_index)
    camera.set(cv2.cv.CV_CAP_PROP_FRAME_WIDTH, 640);
    camera.set(cv2.cv.CV_CAP_PROP_FRAME_HEIGHT, 480);
    camera.set(cv2.cv.CV_CAP_PROP_FPS, 10)
    if not camera.isOpened():
        print("Cannot open stream of index {}".format(stream_index))
        exit(1)

    print("Input stream is of resolution: {} x {}".format(camera.get(3), camera.
↪get(4)))

    conn = angus.client.connect()
    service = conn.services.get_service('face_expression_estimation', 1)
    service.enable_session()

    while camera.isOpened():
        ret, frame = camera.read()
        if not ret:
            break

        ### angus.ai computer vision services require gray images right now.
        gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
        ret, buff = cv2.imencode(".jpg", gray, [cv2.IMWRITE_JPEG_QUALITY, 80])
        buff = StringIO.StringIO(np.array(buff).tostring())

        job = service.process({"image": buff})
        res = job.result

        for face in res['faces']:
            x, y, dx, dy = face['roi']
            cv2.rectangle(frame, (x, y), (x+dx, y+dy), (0,255,0))

            ### Sorting of the 5 expressions measures
            ### to display the most likely on the screen
            exps = [(face[exp], exp) for exp in
                    ['sadness', 'happiness', 'neutral', 'surprise', 'anger']]
            exps.sort()
            max_exp = exps[-1]

            cv2.putText(frame, str(max_exp[1]), (x, y),
                        cv2.FONT_HERSHEY_SIMPLEX, 1, (255, 255, 255))

        cv2.imshow('original', frame)

        if cv2.waitKey(1) & 0xFF == ord('q'):
            break


    ### Disabling session on the server
    service.disable_session()

    camera.release()
    cv2.destroyAllWindows()

if __name__ == '__main__':
    ### Web cam index might be different from 0 on your setup.
    ### To grab a given video file instead of the host computer cam, try:
    ### main("/path/to/myvideo.avi")
    main(0)
```

Face Detection

Do I see human faces? How many? Where?

Getting Started

Using Angus python SDK:

```
# -*- coding: utf-8 -*-
import angus.client
from pprint import pprint

conn = angus.client.connect()
service = conn.services.get_service('face_detection', version=1)
job = service.process({'image': open('./macgyver.jpg', 'rb')})
pprint(job.result)
```

Input

The API takes a stream of 2d still images as input, of format `jpg` or `png`, without constraints on resolution.

Note however that the bigger the resolution, the longer the API will take to process and give a result.

The function `process()` takes a dictionary as input formatted as follows:

```
{'image' : file}
```

- `image`: a python `File Object` as returned for example by `open()` or a `StringIO` buffer.

Output

Events will be pushed to your client following that format:

```
{
  "input_size" : [480, 640],
  "nb_faces" : 2,
  "faces" : [
              {
                "roi" : [345, 223, 34, 54],
                "roi_confidence" : 0.89
              },
              {
                "roi" : [35, 323, 45, 34],
                "roi_confidence" : 0.56
              }
            ]
}
```

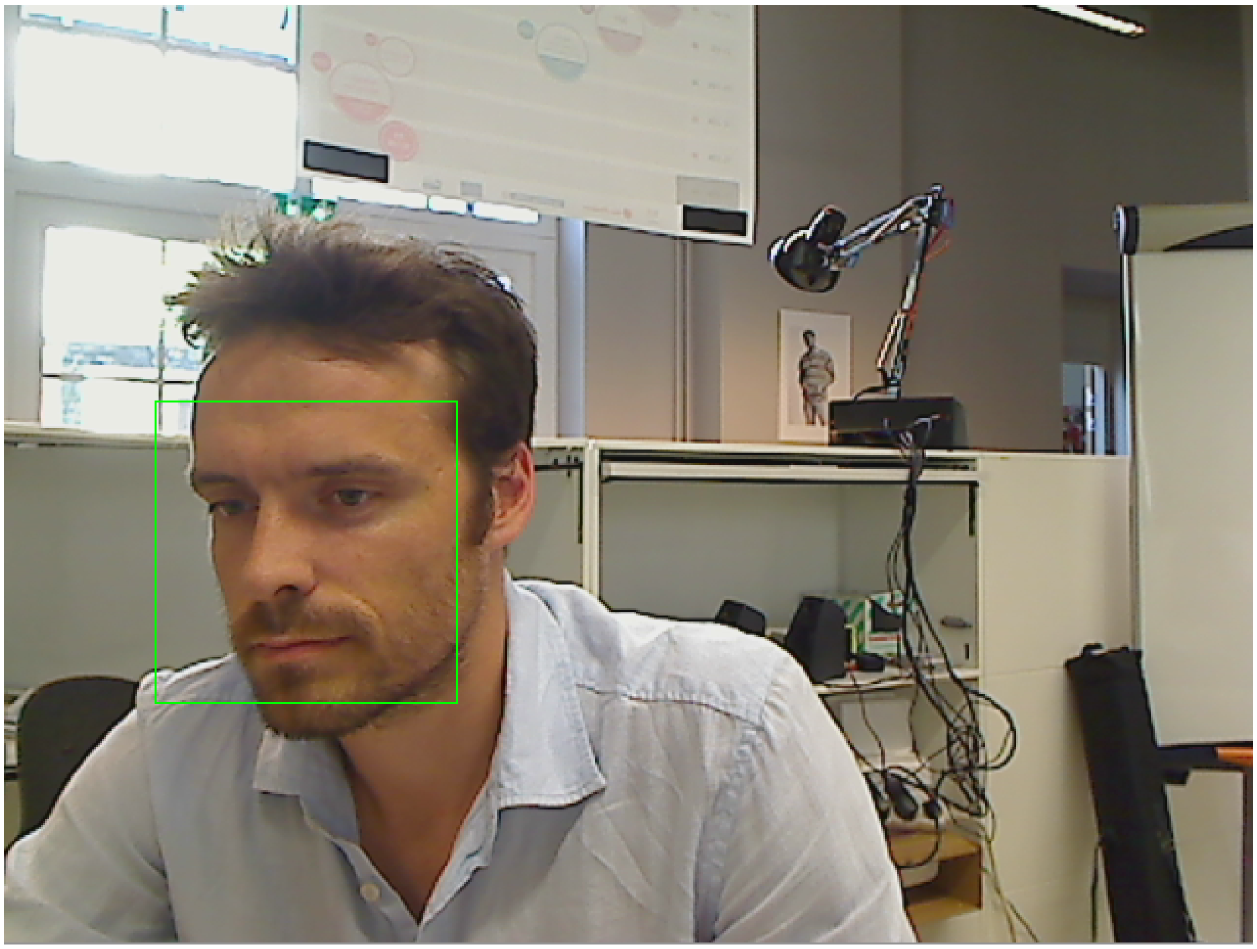
- `input_size` : width and height of the input image in pixels (to be used as reference to `roi` output.
- `nb_faces` : number of faces detected in the given image
- `roi` : contains `[pt.x, pt.y, width, height]` where pt is the upper left point of the rectangle outlining the detected face.
- `roi_confidence` : an estimate of the probability that a real face is indeed located at the given `roi`.

Code Sample

requirements: opencv2, opencv2 python bindings

This code sample retrieves the stream of a web cam and display in a GUI the result of the `face_detection` service.

```
# -*- coding: utf-8 -*-
import cv2
import numpy as np
import StringIO

import angus.client

def main(stream_index):
    camera = cv2.VideoCapture(stream_index)
    camera.set(cv2.cv.CV_CAP_PROP_FRAME_WIDTH, 640)
    camera.set(cv2.cv.CV_CAP_PROP_FRAME_HEIGHT, 480)
    camera.set(cv2.cv.CV_CAP_PROP_FPS, 10)

    if not camera.isOpened():
        print("Cannot open stream of index {}".format(stream_index))
        exit(1)

    print("Video stream is of resolution {} x {}".format(camera.get(3), camera.
↪get(4)))

    conn = angus.client.connect()
    service = conn.services.get_service("age_and_gender_estimation", version=1)
    service.enable_session()

    while camera.isOpened():
        ret, frame = camera.read()

        if not ret:
            break

        gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
        ret, buff = cv2.imencode(".jpg", gray,  [cv2.IMWRITE_JPEG_QUALITY, 80])
        buff = StringIO.StringIO(np.array(buff).tostring())

        job = service.process({"image": buff})
        res = job.result

        for face in res['faces']:
            x, y, dx, dy = face['roi']
            cv2.rectangle(frame, (x, y), (x+dx, y+dy), (0,255,0))

        cv2.imshow('original', frame)
        if cv2.waitKey(1) & 0xFF == ord('q'):
            break

    service.disable_session()

    camera.release()
    cv2.destroyAllWindows()

if __name__ == '__main__':
    ### Web cam index might be different from 0 on your setup.
    ### To grab a given video file instead of the host computer cam, try:
    ### main("/path/to/myvideo.avi")
    main(0)
```



Face Recognition

This service spots for a specified set of people in images or videos.

To be able to recognize people, this service needs to be first provided with a few pictures of each person's face.

How to prepare face samples?

Here are a few tips to make sure you get the most of Angus `face_recognition` service:

- make sure the resolution of these samples is high enough.
- make sure these samples show a unique face only, in order to avoid any ambiguity.
- the service will perform better if you provide more than 1 sample for each person, with different face expressions.

For example, the code sample shown below makes use of the following face sample (only 1 sample per people to recognize is used in that case).

Getting Started

Using the Angus python SDK:

```
# -*- coding: utf-8 -*-
import angus.client
from pprint import pprint

conn = angus.client.connect()
service = conn.services.get_service('face_recognition', version=1)

PATH = "/path/to/your/face/samples/"

w1_s1 = conn.blobs.create(open(PATH + "jamel/1.jpeg", 'rb'))
w1_s2 = conn.blobs.create(open(PATH + "jamel/2.jpg", 'rb'))
w1_s3 = conn.blobs.create(open(PATH + "jamel/3.jpg", 'rb'))
w1_s4 = conn.blobs.create(open(PATH + "jamel/4.jpg", 'rb'))

w2_s1 = conn.blobs.create(open(PATH + "melissa/1.jpg", 'rb'))
w2_s2 = conn.blobs.create(open(PATH + "melissa/2.jpg", 'rb'))
w2_s3 = conn.blobs.create(open(PATH + "melissa/3.jpg", 'rb'))
w2_s4 = conn.blobs.create(open(PATH + "melissa/4.jpg", 'rb'))

album = {'jamel': [w1_s1, w1_s2, w1_s3, w1_s4], 'melissa': [w2_s1, w2_s2, w2_s3, w2_
↪s4]}

job = service.process({'image': open(PATH + "melissa/5.jpg", 'rb'), "album" : album})
pprint(job.result)
```

Input

The API captures a stream of 2D still images as input, under `jpg` or `png` format, without any constraint of resolution.

Note however that the bigger the resolution, the longer the API takes to process and give a result.

The function `process()` takes a dictionary as input formatted as follows:

```
{
 'image' : file,
 'album' : {"people1": [sample_1, sample_2], "people2" : [sample_1, sample_2]}
}
```

- `image`: a python `File Object` as returned for example by `open()` or a `StringIO` buffer.
- `album` : a dictionary containing samples of the faces that need to be spotted. Samples need first to be provided to the service using the function `blobs.create()` as per the example above. The more samples the better, although 1 sample per people is enough.

Output

Events will be pushed to your client following that format:

```
{
  "input_size" : [480, 640],
  "nb_faces" : 1,
  "faces" : [
              {
                "roi" : [345, 223, 34, 54],
                "roi_confidence" : 0.89,
                "names" : [
                            {
                              "key" : "jamel",
                              "confidence" : 0.75
                            },
                            {
                              "key" : "melissa",
                              "confidence" : 0.10
                            }
                          ]
              }
            ]
}
```

- `input_size` : width and height of the input image in pixels (to be used as reference to `roi` output.
- `nb_faces` : number of faces detected in the given image
- `roi` : Region Of Interest containing `[pt.x, pt.y, width, height]`, where pt is the upper left point of the rectangle outlining the detected face.
- `roi_confidence` : probability that a real face is indeed located at the given `roi`.
- `key` : they key identifying a given group of samples (as specified in the `album` input).
- `confidence` : probability that the corresponding people was spotted in the image / video stream.

Code Sample

requirements: opencv2, opencv2 python bindings

This code sample captures the stream of a web cam and displays the result of the `face_recognition` service in a GUI.

```
# -*- coding: utf-8 -*-
import StringIO
import cv2
import numpy as np

import angus.client


def main(stream_index):
    camera = cv2.VideoCapture(stream_index)
    camera.set(cv2.cv.CV_CAP_PROP_FRAME_WIDTH, 640)
    camera.set(cv2.cv.CV_CAP_PROP_FRAME_HEIGHT, 480)
    camera.set(cv2.cv.CV_CAP_PROP_FPS, 10)

    if not camera.isOpened():
        print("Cannot open stream of index {}".format(stream_index))
        exit(1)

    print("Input stream is of resolution: {} x {}".format(camera.get(3), camera.
↪get(4)))
```

```
    conn = angus.client.connect()
    service = conn.services.get_service("face_recognition", version=1)

    ### Choose here the appropriate pictures.
    ### Pictures given as samples for the album should only contain 1 visible face.
    ### You can provide the API with more than 1 photo for a given person.
    w1_s1 = conn.blobs.create(open("./images/gwenn.jpg", 'rb'))
    w2_s1 = conn.blobs.create(open("./images/aurelien.jpg", 'rb'))
    w3_s1 = conn.blobs.create(open("./images/sylvain.jpg", 'rb'))

    album = {'gwenn': [w1_s1], 'aurelien': [w2_s1], 'sylvain': [w3_s1]}

    service.enable_session({"album" : album})

    while camera.isOpened():
        ret, frame = camera.read()
        if not ret:
            break

        gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
        ret, buff = cv2.imencode(".jpg", frame, [cv2.IMWRITE_JPEG_QUALITY, 80])
        buff = StringIO.StringIO(np.array(buff).tostring())

        job = service.process({"image": buff})
        res = job.result

        for face in res['faces']:
            x, y, dx, dy = face['roi']
            cv2.rectangle(frame, (x, y), (x+dx, y+dy), (0,255,0))

            if len(face['names']) > 0:
                name = face['names'][0]['key']
                cv2.putText(frame, "Name = {}".format(name), (x, y),
                            cv2.FONT_HERSHEY_SIMPLEX, 1, (255, 255, 255))

            cv2.imshow('original', frame)
            if cv2.waitKey(1) & 0xFF == ord('q'):
                break

    service.disable_session()

    camera.release()
    cv2.destroyAllWindows()

if __name__ == '__main__':
    ### Web cam index might be different from 0 on your setup.
    ### To grab a given video file instead of the host computer cam, try:
    ### main("/path/to/myvideo.avi")
    main(0)
```

Gaze Analysis

What are people in front of my object looking at?

Getting Started

Using the Python SDK:

```python
# -*- coding: utf-8 -*-
import angus.client
from pprint import pprint

conn = angus.client.connect()
service = conn.services.get_service('gaze_analysis', version=1)
job = service.process({'image': open('./macgyver.jpg', 'rb')})

pprint(job.result)
```

Input

The API takes a stream of 2d still images as input, of format `jpg` or `png`, without constraints on resolution.

Note however that the bigger the resolution, the longer the API will take to process and give a result.

The function `process()` takes a dictionary as input formatted as follows:

```
{'image' : file}
```

- `image`: a python `File Object` as returned for example by `open()` or a `StringIO` buffer.

Output

Events will be pushed to your client following that format:

```
{
  "input_size" : [480, 640],
  "nb_faces" : 1,
  "faces" : [
              {
                "roi" : [250, 142, 232, 232],
                "roi_confidence" : 0.89,
                "eye_left" : [123, 253],
                "eye_right" : [345, 253],
                "nose" : [200, 320],
                "head_yaw" : 0.03,
                "head_pitch"    : 0.23,
                "head_roll"  : 0.14,
                "gaze_yaw"    : 0.05,
                "gaze_pitch"  : 0.12
              }
            ]
}
```

- `input_size` : width and height of the input image in pixels (to be used as reference to `roi` output.
- `nb_faces` : number of faces detected in the given image
- `roi` : contains `[pt.x, pt.y, width, height]` where pt is the upper left point of the rectangle outlining the detected face.
- `roi_confidence` : an estimate of the probability that a real face is indeed located at the given `roi`.

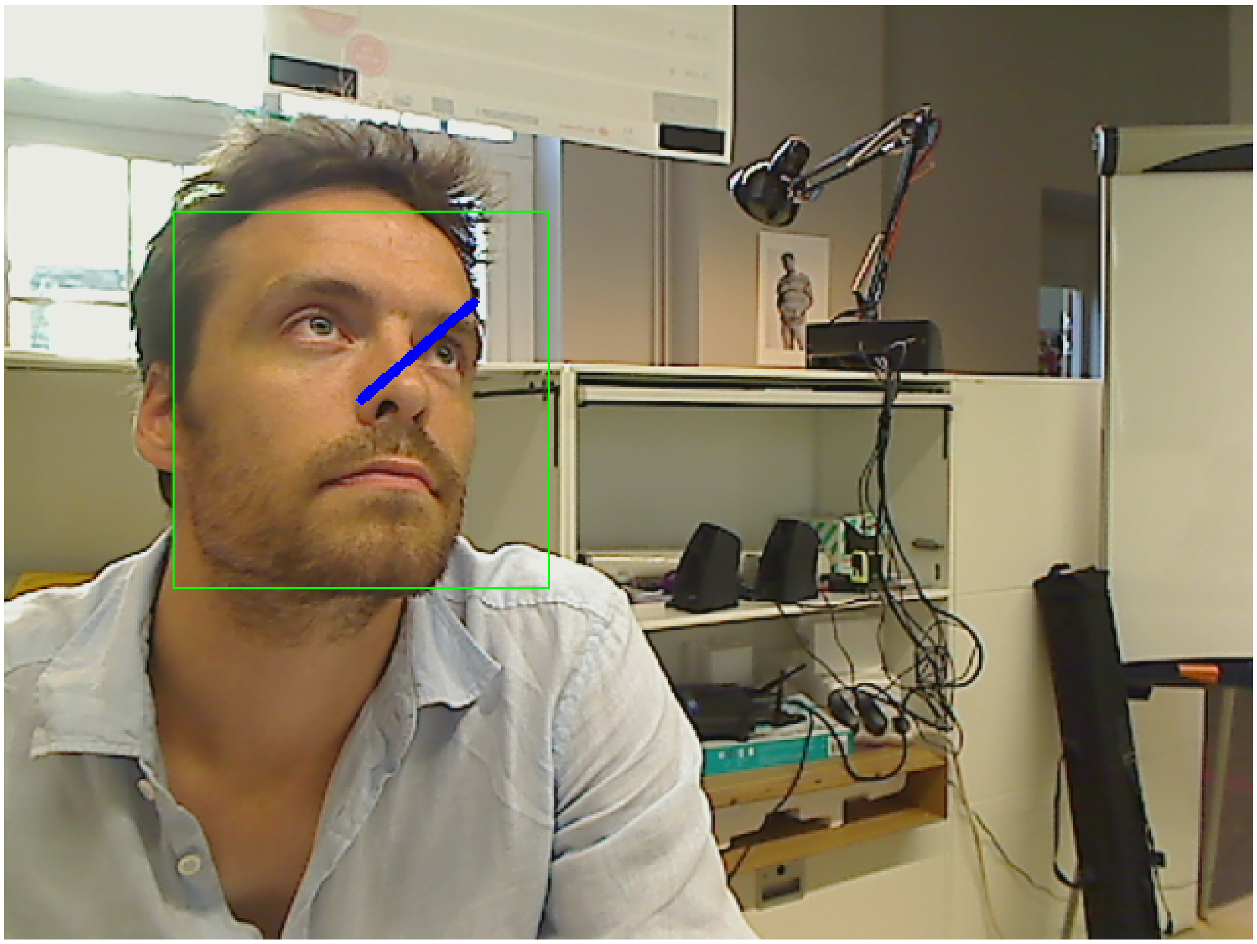
- `head_yaw`, `head_pitch`, `head_roll` : head pose orientation in radian.
- `gaze_yaw`, `gaze_pitch` : gaze (eyes) orientation in radian.
- `eye_left`, `eye_right`, `nose` : the coordinate of the eyes and noze in the given image.

Code Sample

requirements: opencv2, opencv2 python bindings

This code sample retrieves the stream of a web cam and display in a GUI the result of the `face_detection` service.

```
# -*- coding: utf-8 -*-
import StringIO
from math import cos, sin
import cv2
import numpy as np
import angus.client

def main(stream_index):
    camera = cv2.VideoCapture(0)
    camera.set(cv2.cv.CV_CAP_PROP_FRAME_WIDTH, 640);
    camera.set(cv2.cv.CV_CAP_PROP_FRAME_HEIGHT, 480);
    camera.set(cv2.cv.CV_CAP_PROP_FPS, 10)

    if not camera.isOpened():
        print("Cannot open stream of index {}".format(stream_index))
        exit(1)

    print("Input stream is of resolution: {} x {}".format(camera.get(3), camera.
↪get(4)))

    conn = angus.client.connect()
    service = conn.services.get_service('gaze_analysis', 1)
    service.enable_session()

    while camera.isOpened():
        ret, frame = camera.read()
        if not ret:
            break

        ### angus.ai computer vision services require gray images right now.
        gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
        ret, buff = cv2.imencode(".jpg", gray, [cv2.IMWRITE_JPEG_QUALITY, 80])
        buff = StringIO.StringIO(np.array(buff).tostring())

        job = service.process({"image": buff})
        res = job.result

        for face in res['faces']:
            x, y, dx, dy = map(int, face['roi'])

            nose = face['nose']
            nose = (nose[0], nose[1])

            eyel = face['eye_left']
            eyel = (eyel[0], eyel[1])
            eyer = face['eye_right']
            eyer = (eyer[0], eyer[1])

            psi = face['head_roll']
            theta = - face['head_yaw']
            phi = face['head_pitch']

            ### head orientation
            length = 150
            xvec = int(length*(sin(phi)*sin(psi) - cos(phi)*sin(theta)*cos(psi)))
            yvec = int(- length*(sin(phi)*cos(psi) - cos(phi)*sin(theta)*sin(psi)))
            cv2.line(frame, nose, (nose[0]+xvec, nose[1]+yvec), (0, 140, 255), 3)

            psi = 0
            theta = - face['gaze_yaw']
            phi = face['gaze_pitch']

            ### gaze orientation
            length = 150
            xvec = int(length*(sin(phi)*sin(psi) - cos(phi)*sin(theta)*cos(psi)))
            yvec = int(- length*(sin(phi)*cos(psi) - cos(phi)*sin(theta)*sin(psi)))
            cv2.line(frame, eyel, (eyel[0]+xvec, eyel[1]+yvec), (0, 140, 0), 3)

            xvec = int(length*(sin(phi)*sin(psi) - cos(phi)*sin(theta)*cos(psi)))
            yvec = int(- length*(sin(phi)*cos(psi) - cos(phi)*sin(theta)*sin(psi)))
            cv2.line(frame, eyer, (eyer[0]+xvec, eyer[1]+yvec), (0, 140, 0), 3)


        cv2.imshow('original', frame)

        if cv2.waitKey(1) & 0xFF == ord('q'):
            break

    ### Disabling session on the server
    service.disable_session()

    camera.release()
    cv2.destroyAllWindows()

if __name__ == '__main__':
    ### Web cam index might be different from 0 on your setup.
    ### To grab a given video file instead of the host computer cam, try:
    ### main("/path/to/myvideo.avi")
    main(0)
```



Motion Detection

Is there anything moving in front of my object? Where exactly?

Getting Started

Using Angus python SDK:

```
# -*- coding: utf-8 -*-
import angus.client
from pprint import pprint

conn = angus.client.connect()
service = conn.services.get_service('motion_detection', version=1)

service.enable_session()

for i in range(200):
    job = service.process({'image': open('./photo-{}.jpg'.format(i), 'rb')})
    pprint(job.result)

service.disable_session()
```

Input

The API takes a stream of 2d still images as input, of format `jpg` or `png`, without constraints on resolution.

Note however that the bigger the resolution, the longer the API will take to process and give a result.

The function `process()` takes a dictionary as input formatted as follows:

```
{'image' : file}
```

- `image`: a python `File Object` as returned for example by `open()` or a `StringIO` buffer.

Output

Events will be pushed to your client following that format:

```
{

  "input_size" : [480, 640],
  "nb_targets": 1
  "targets":
            [
              {
                "mean_position" : [34, 54],
                "mean_velocity" : [5, 10],
                "confidence" : 45
              }
            ]
}
```

- `input_size` : width and height of the input image in pixels.
- `mean_position` : `[pt.x, pt.y]` where `pt` is the center of gravity of the moving pixels.
- `mean_velocity` : `[v.x, v.y]` where `v` is the average velocity of the moving pixels.
- `confidence` : in `[0,1]` measures how significant the motion is (a function of the number of keypoints moving in the same direction).

Code Sample

requirements: opencv2, opencv2 python bindings

This code sample retrieves the stream of a web cam and display in a GUI the result of the `motion_detection` service.

```
# -*- coding: utf-8 -*-
import StringIO
import cv2
import numpy as np
import angus.client

def main(stream_index):
    camera = cv2.VideoCapture(stream_index)
    camera.set(cv2.cv.CV_CAP_PROP_FRAME_WIDTH, 640)
    camera.set(cv2.cv.CV_CAP_PROP_FRAME_HEIGHT, 480)
    camera.set(cv2.cv.CV_CAP_PROP_FPS, 10)
    if not camera.isOpened():
        print("Cannot open stream of index {}".format(stream_index))
        exit(1)

    print("Input stream is of resolution: {} x {}".format(camera.get(3), camera.
↪get(4)))

    conn = angus.client.connect()
    service = conn.services.get_service("motion_detection", 1)

    service.enable_session()

    while camera.isOpened():
        ret, frame = camera.read()
        if not ret:
            break

        gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
        ret, buff = cv2.imencode(".jpg", gray, [cv2.IMWRITE_JPEG_QUALITY, 80])
        buff = StringIO.StringIO(np.array(buff).tostring())

        job = service.process({"image": buff})
        res = job.result

        for target in res['targets']:
            x, y = map(int, target['mean_position'])
            vx, vy = map(int, target['mean_velocity'])

            cv2.circle(frame, (x, y), 5, (255,255,255))
            cv2.line(frame, (x, y), (x + vx, y + vy), (255,255,255))

        cv2.imshow('original', frame)
        if cv2.waitKey(1) & 0xFF == ord('q'):
            break

    service.disable_session()

    camera.release()
    cv2.destroyAllWindows()

if __name__ == '__main__':
    ### Web cam index might be different from 0 on your setup.
    ### To grab a given video file instead of the host computer cam, try:
    ### main("/path/to/myvideo.avi")
    main(0)
```

Text To Speech

This service generates a sound file (".wav") from a any given text in the following languages:

- English (US)
- English (GB)
- German
- Spanish (ES)
- French
- Italian

Getting Started

Using Angus python SDK:

```
# -*- coding: utf-8 -*-
import angus.client

conn = angus.client.connect()
service = conn.services.get_service('text_to_speech', version=1)
```

```
job = service.process({'text': "Hi guys, how are you today?", 'lang' : "en-US"})

### The output wav file is available as compressed (zlib), base64 string.
sound = job.result["sound"]
```

Input

The function `process()` takes a dictionary formatted as follows:

```
{'text' : "Hello guys",
 'lang' : "en-US"}
```

- `text`: a string containing the text to be synthesized.
- `lang`: the code of the language to be used for synthesis. Languages currently available are:
  - English (US) : `en-US`
  - English (GB) : `en-GB`
  - German : `de-DE`
  - Spanish (ES) : `es-ES`
  - French : `fr-FR`
  - Italian : `it-IT`

Output

Events will be pushed to your client following the below format:

```
{
  "status" : 201,
  "sound" : "'eJzsvHdUFNm6N1yhEzQ ... jzf//+T/jj/A8b0r/9"
}
```

- `status`: the http status code of the request.
- `sound` : contains the synthesized sound file (.wav) as a compressed (zlib), base64 string. Please refer to the code sample below of how to decode it in Python.

Code Sample

This code sample uses Angus `text_to_speech` service to synthesize “hi guys, how are you today?”.

```
# -*- coding: utf-8 -*-
import angus.client
import base64
import zlib
import subprocess

def decode_output(sound, filename):
        sound = base64.b64decode(sound)
        sound = zlib.decompress(sound)
        with open(filename, "wb") as f:
            f.write(sound)

conn = angus.client.connect()
service = conn.services.get_service('text_to_speech', version=1)

job = service.process({'text': "Hi guys, how are you today?", 'lang' : "en-US"})

### The output wav file is available as compressed (zlib), base64 string.
### Here, the string is decoded and played back by Linux "aplay".
decode_output(job.result["sound"], "output.wav")
subprocess.call(["/usr/bin/aplay", "./output.wav"])
```

Sound Detection

Is there any noticeable sound?

Getting Started

Using the Angus python SDK:

```
# -*- coding: utf-8 -*-
import angus.client
from pprint import pprint

conn = angus.client.connect()
service = conn.services.get_service('sound_detection', version=1)
job = service.process({'sound': open("./sound.wav", 'rb'), 'sensitivity':0.7})

pprint(job.result)
```

Input

```
{'sound' : file,
 'sensitivity' : 0.3}
```

- `sound` : a python `File Object` as returned for example by `open()` or a `StringIO` buffer describing a wav file with the following format : `PCM 16bit, Mono`, without constraints on sample rate.
- `sensitivity` : modifies the ability of the algorithms to detect quiet sounds. `[0, 1]`. The higher the value is, the better the algorithm will detect quiet sounds, but the more it will be sensitive to background noise.

Output

Events will be pushed to your client following that format:

```
{
  "input_size" : 8192,
  "nb_events" : 2,
  "events" : [
                {
                  "index" : 3454,
                  "type"    : 'sound_on'
                },
                {
                  "index" : 6544,
                  "type"    : 'sound_off'
                }
              ]
}
```

- `input_size` : number of frame given as input (eg. in a stereo file, 1 frame = 1 left sample + 1 right sample).
- `nb_events` : number of events detected in the given input buffer.
- `index` : the frame index in the given input buffer where the event has been detected.
- `type` : `sound_on` if the beginning of a sound is detected. `sound_off` if the end of a sound is detected. Note that an event of type `sound_on` is always followed by an event of type `sound_off`.

Code Sample

requirements: PyAudio

This code sample retrieve the audio stream of a web cam and display the result of the `sound_detection` service.

```
# -*- coding: utf-8 -*-
import Queue
import StringIO
import wave
import time
import sys
from pprint import pprint
import pyaudio
import numpy as np
import angus.client

CHUNK = 8192
PYAUDIO_FORMAT = pyaudio.paInt16
NUMPY_FORMAT = np.int16
TARGET_RATE = 16000
TARGET_CHANNELS = 1

def list_inputs():
    p = pyaudio.PyAudio()
    for i in range(p.get_device_count()):
        info = p.get_device_info_by_index(i)
        if info['maxInputChannels'] > 0:
            print("Device index={} name={}".format(info['index'], info['name']))

def prepare(in_data, channels, rate):
    # Extract first channel
    in_data = np.fromstring(in_data, dtype=NUMPY_FORMAT)
    in_data = np.reshape(in_data, (CHUNK, channels))
    in_data = in_data[:,0]

    # Re-sample if needed only for mono stream
    srcx = np.arange(0, in_data.size, 1)
    tgtx = np.arange(0, in_data.size, float(rate) / float(TARGET_RATE))
    in_data = np.interp(tgtx, srcx, in_data).astype(NUMPY_FORMAT)
    return in_data.tostring()

def main(stream_index):
    p = pyaudio.PyAudio()

    # Device configuration
    conf = p.get_device_info_by_index(stream_index)
    channels = int(conf['maxInputChannels'])
    if channels < TARGET_CHANNELS:
        raise RuntimeException("Bad device, no input channel")

    rate = int(conf['defaultSampleRate'])
    if rate < TARGET_RATE:
        raise RuntimeException("Bad device, sample rate is too low")

    # Angus
    conn = angus.client.connect()
    service = conn.services.get_service('sound_detection', version=1)
    service.enable_session()

    # Record Process
    stream_queue = Queue.Queue()
    def chunk_callback(in_data, frame_count, time_info, status):
        in_data = prepare(in_data, channels, rate)
        stream_queue.put(in_data)
        return (in_data, pyaudio.paContinue)
    stream = p.open(format=PYAUDIO_FORMAT,
                    channels=channels,
                    rate=rate,
                    input=True,
                    frames_per_buffer=CHUNK,
                    input_device_index=stream_index,
                    stream_callback=chunk_callback)
    stream.start_stream()

    # Get data and send to Angus.ai
    while True:
        nb_buffer_available = stream_queue.qsize()
        if nb_buffer_available > 0:
            print("nb buffer available = {}".format(nb_buffer_available))

        if nb_buffer_available == 0:
            time.sleep(0.01)
            continue

        data = stream_queue.get()

        buff = StringIO.StringIO()

        wf = wave.open(buff, 'wb')
        wf.setnchannels(TARGET_CHANNELS)
        wf.setsampwidth(p.get_sample_size(PYAUDIO_FORMAT))
        wf.setframerate(TARGET_RATE)
        wf.writeframes(data)
        wf.close()

        job = service.process(
                {'sound': StringIO.StringIO(buff.getvalue()), 'sensitivity': 0.7})
            pprint(job.result)

    stream.stop_stream()
    stream.close()
    p.terminate()

if __name__ == "__main__":
    if len(sys.argv) < 2:
        list_inputs()
        INDEX = raw_input("Please select a device number:")
    else:
        INDEX = sys.argv[1]
    try:
        main(int(INDEX))
    except ValueError:
        print("Not a valid index")
        exit(1)
```

Voice Detection (Beta)

This service takes an audio stream as an input and tries to discriminate what is human voice and what is not. If detecting noise in general, and not specifically human voice, use *Sound Detection* instead.

Getting Started

Using the Angus python SDK:

```
# -*- coding: utf-8 -*-
import angus.client.cloud

conn = angus.client.connect()

service = conn.services.get_service('voice_detection', version=1)

job = service.process({'sound': open("./sound.wav", 'rb'), 'sensitivity':0.7})

print job.result
```

Input

```
{'sound' : file,
 'sensitivity' : 0.3}
```

- `sound` : a python `File Object` as returned for example by `open()` or a `StringIO` buffer describing a wav file with the following format : `PCM 16bit, Mono`, without constraints on sample rate.
- `sensitivity` : modifies the ability of the algorithms to detect quiet voices. `[0, 1]`. The higher the value is, the better the algorithm will detect quiet voices, but the more it will be sensitive to background noise.

Output

Events will be pushed to your client following that format:

```
{
  "voice_activity" : "SILENCE"
}
```

- `voice_activity` : this field takes 4 different values: `SILENCE` when no voice is detected, `VOICE` when voice is detected, `ON` when a transition occurs between `SILENCE` and `VOICE`, and `OFF` when a transition occurs between `VOICE` and `SILENCE`.

Code Sample

requirements: PyAudio

This code sample retrieve the audio stream of a web cam and display the result of the `voice_detection` service.

```
# -*- coding: utf-8 -*-
import Queue
import StringIO
import wave
import time
import angus.client
import pyaudio
import sys
import numpy as np


CHUNK = 8192
PYAUDIO_FORMAT = pyaudio.paInt16
NUMPY_FORMAT = np.int16

def list_inputs():
    p = pyaudio.PyAudio()
    for i in range(p.get_device_count()):
        info = p.get_device_info_by_index(i)
        if info['maxInputChannels'] > 0:
            print("Device index={} name={}".format(info['index'], info['name']))

def prepare(in_data, channels, rate):
    # Extract first channel
    in_data = np.fromstring(in_data, dtype=NUMPY_FORMAT)
    in_data = np.reshape(in_data, (CHUNK, channels))
    in_data = in_data[:,0]

    # Down sample if needed
    srcx = np.arange(0, in_data.size, 1)
    tgtx = np.arange(0, in_data.size, float(rate) / float(16000))

    in_data = np.interp(tgtx, srcx, in_data).astype(NUMPY_FORMAT)
    return in_data


def main(stream_index):

    p = pyaudio.PyAudio()
    # Device configuration
    conf = p.get_device_info_by_index(stream_index)
    channels = int(conf['maxInputChannels'])
    if channels < 1:
        raise RuntimeException("Bad device, no input channel")

    rate = int(conf['defaultSampleRate'])
    if rate < 16000:
        raise RuntimeException("Bad device, sample rate is too low")


    # Angus
    conn = angus.client.connect()
    service = conn.services.get_service('voice_detection', version=1)
    service.enable_session()

    # Record Process
    stream_queue = Queue.Queue()
    def chunk_callback(in_data, frame_count, time_info, status):
        in_data = prepare(in_data, channels, rate)
        stream_queue.put(in_data.tostring())
        return (in_data, pyaudio.paContinue)

    stream = p.open(format=PYAUDIO_FORMAT,
                    channels=channels,
                    rate=rate,
                    input=True,
                    frames_per_buffer=CHUNK,
                    input_device_index=stream_index,
                    stream_callback=chunk_callback)
    stream.start_stream()

    # Get data and send to Angus.ai
    while True:
        nb_buffer_available = stream_queue.qsize()

        if nb_buffer_available == 0:
            time.sleep(0.01)
            continue

        data = stream_queue.get()
        buff = StringIO.StringIO()

        wf = wave.open(buff, 'wb')
        wf.setnchannels(1)
        wf.setsampwidth(p.get_sample_size(PYAUDIO_FORMAT))
        wf.setframerate(16000)
        wf.writeframes(data)
        wf.close()

        job = service.process(
            {'sound': StringIO.StringIO(buff.getvalue()), 'sensitivity': 0.2})

        res = job.result["voice_activity"]

        if res == "VOICE":
            print "\033[A                                            \033[A"
            print "***************************"
            print "*****   VOICE !!!!   ******"
            print "***************************"


    stream.stop_stream()
    stream.close()
    p.terminate()


if __name__ == "__main__":
    if len(sys.argv) < 2:
        list_inputs()
        index = raw_input("Please select a device number:")
    else:
        index = sys.argv[1]

    try:
        index = int(index)
        main(index)
    except ValueError:
        print("Not a valid index")
        exit(1)
```

Sound Localization (Beta)

Where is the sound coming from?

Getting Started

Using the Angus python SDK:

```
# -*- coding: utf-8 -*-
import angus.client
from pprint import pprint

conn = angus.client.connect()
service = conn.services.get_service('sound_localization', version=1)
job = service.process({'sound': open("./sound.wav", 'rb'), 'baseline' : 0.7,
↪'sensitivity:0.5'})

pprint(job.result)
```

Input

```
{'sound' : file,
 'baseline' : 0.7,
 'sensitivity' : 0.3}
```

- `sound` : a python `File Object` as returned for example by `open()` or a `StringIO` buffer describing a wav file with the following format: `PCM 16bit, 48kHz, Stereo`.
- `baseline` : distance between the 2 microphones of the array in `meters`.

- `sensitivity` : modifies the ability of the algorithms to locate quiet sounds. `[0, 1]`. The higher the value is, the better the algorithm will locate quiet sounds, but the more it will be sensitive to background noise.

Output

Events will be pushed to your client following that format:

```
{
  "input_size" : 8192,
  "nb_sources" : 1,
  "sources" : [
                {
                  "index" : 345,
                  "yaw" : 0.156,
                  "confidence" : 0.53,
                }
              ]
}
```

- `input_size` : number of frame given as input (in a stereo file, 1 frame = 1 left sample + 1 right sample).
- `nb_sources` : number of sound sources located.
- `yaw` : angle of the sound source in radian as shown below:
- `confidence` : an estimate of the probability that a real sound source is indeed located at the given `yaw`.

Code Sample

This sample assumes that you have a sound card able to record in stereo.

requirements: PyAudio

This code sample retrieve the audio stream of a recording device and display the result of the `sound_localization` service.

```
# -*- coding: utf-8 -*-
import Queue
import StringIO
import wave
import time
import sys
from pprint import pprint
import pyaudio
import numpy as np
import angus.client

CHUNK = 8192
PYAUDIO_FORMAT = pyaudio.paInt16
NUMPY_FORMAT = np.int16
TARGET_RATE = 48000
TARGET_CHANNELS = 2

def list_inputs():
    p = pyaudio.PyAudio()
    for i in range(p.get_device_count()):
        info = p.get_device_info_by_index(i)
        if info['maxInputChannels'] > 0:
            print("Device index={} name={}".format(info['index'], info['name']))

def prepare(in_data, channels, rate):
    # Extract first channel
    in_data = np.fromstring(in_data, dtype=NUMPY_FORMAT)
    in_data = np.reshape(in_data, (CHUNK, channels))

    # Re-sample if needed
    srcx = np.arange(0, CHUNK, 1)
    tgtx = np.arange(0, CHUNK, float(rate) / float(TARGET_RATE))

    print ((in_data[:,0]).size)

    left = np.interp(tgtx, srcx, in_data[:,0]).astype(NUMPY_FORMAT)
    right = np.interp(tgtx, srcx, in_data[:,1]).astype(NUMPY_FORMAT)

    print left.size
    print CHUNK

    c = np.empty((left.size + right.size), dtype=NUMPY_FORMAT)
    c[0::2] = left
    c[1::2] = right
    return c.tostring()

def main(stream_index):
    p = pyaudio.PyAudio()

    # Device configuration
    conf = p.get_device_info_by_index(stream_index)
    channels = int(conf['maxInputChannels'])
    if channels < TARGET_CHANNELS:
        raise RuntimeException("Bad device, no input channel")

    rate = int(conf['defaultSampleRate'])

    # Angus
    conn = angus.client.connect()
    service = conn.services.get_service('sound_localization', version=1)
    service.enable_session()

    # Record Process
    stream_queue = Queue.Queue()
    def chunk_callback(in_data, frame_count, time_info, status):
        in_data = prepare(in_data, channels, rate)
        stream_queue.put(in_data)
        return (in_data, pyaudio.paContinue)
    stream = p.open(format=PYAUDIO_FORMAT,
                    channels=channels,
                    rate=rate,
                    input=True,
                    frames_per_buffer=CHUNK,
                    input_device_index=stream_index,
                    stream_callback=chunk_callback)
    stream.start_stream()

    while True:
        nb_buffer_available = stream_queue.qsize()
        if nb_buffer_available > 0:
            print("nb buffer available = {}".format(nb_buffer_available))

        if nb_buffer_available == 0:
            time.sleep(0.01)
            continue

        data = stream_queue.get()

        buff = StringIO.StringIO()

        wf = wave.open(buff, 'wb')
        wf.setnchannels(TARGET_CHANNELS)
        wf.setsampwidth(p.get_sample_size(PYAUDIO_FORMAT))
        wf.setframerate(TARGET_RATE)
        wf.writeframes(data)
        wf.close()

        job = service.process(
            {'sound': StringIO.StringIO(buff.getvalue()), 'baseline': 0.14,
↪'sensitivity': 0.7})
        pprint(job.result['sources'])


    stream.stop_stream()
    stream.close()
    p.terminate()

if __name__ == "__main__":
    if len(sys.argv) < 2:
        list_inputs()
        INDEX = raw_input("Please select a device number:")
    else:
        INDEX = sys.argv[1]
    try:
        main(int(INDEX))
    except ValueError:
        print("Not a valid index")
        exit(1)
```

Qrcode decoder

Do I see a qrcode ? What is the content ?

Getting Started

You can use this qrcode for example:

Using Angus python SDK:

```
# -*- coding: utf-8 -*-
import angus.client
from pprint import pprint

conn = angus.client.connect()
service = conn.services.get_service('qrcode_decoder', version=1)
job = service.process({'image': open('./qrcode.jpg', 'rb')})

pprint(job.result)
```

Input

The API takes a stream of 2d still images as input, of format `jpg` or `png`, without constraints on resolution.

Note however that the bigger the resolution, the longer the API will take to process and give a result.

The function `process()` takes a dictionary as input formatted as follows:

```
{'image' : file}
```

- `image`: a python `File Object` as returned for example by `open()` or a `StringIO` buffer.

Output

Events will be pushed to your client following that format:

```
{
  "type": "QRCODE",
  "data": "http://www.angus.ai"
}
```

- `type` : qrcode data type
- `data` : content

Code Sample

requirements: opencv2, opencv2 python bindings

This code sample retrieves the stream of a webcam and print on standard output the qrcode content data.

```
# -*- coding: utf-8 -*-
import StringIO
import cv2
import numpy as np
from pprint import pprint
import angus.client

def main(stream_index):
    camera = cv2.VideoCapture(stream_index)
    camera.set(cv2.cv.CV_CAP_PROP_FRAME_WIDTH, 640)
    camera.set(cv2.cv.CV_CAP_PROP_FRAME_HEIGHT, 480)
    camera.set(cv2.cv.CV_CAP_PROP_FPS, 10)

    if not camera.isOpened():
        print("Cannot open stream of index {}".format(stream_index))
        exit(1)

    print("Input stream is of resolution: {} x {}".format(camera.get(3), camera.
↪get(4)))

    conn = angus.client.connect()
    service = conn.services.get_service("qrcode_decoder", version=1)
    service.enable_session()

    while camera.isOpened():
        ret, frame = camera.read()
        if not ret:
            break

        gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
        ret, buff = cv2.imencode(".jpg", gray, [cv2.IMWRITE_JPEG_QUALITY, 80])
        buff = StringIO.StringIO(np.array(buff).tostring())

        job = service.process({"image": buff})

        if "data" in job.result:
            pprint(job.result["data"])
        cv2.imshow('original', frame)
        if cv2.waitKey(1) & 0xFF == ord('q'):
            break

    service.disable_session()

    camera.release()
    cv2.destroyAllWindows()

if __name__ == '__main__':
    ### Web cam index might be different from 0 on your setup.
    ### To grab a given video file instead of the host computer cam, try:
    ### main("/path/to/myvideo.avi")
    main(0)
```

Dummy

Is my configuration correct ? I want to make my first call to angus.ai cloud.

Getting Started

Using Angus python SDK:

```
# -*- coding: utf-8 -*-
import angus.client
from pprint import pprint

conn = angus.client.connect()
service = conn.services.get_service('dummy', version=1)
job = service.process({'echo': 'Hello world'})

pprint(job.result)
```

Input

The API takes a optional string as parameter and return a result equals to these string.

The function `process()` takes a dictionary as input formatted as follows:

```
{'echo' : 'Hello world!'}
```

- `echo`: a python string or unicode object.

Output

The service just return the input parameter if defined or the string `"echo"` if no parameter.

```
{'echo': 'Hello world!'}
```

- `echo` : the copy of the input string or `"echo"` if no defined.

Requirements

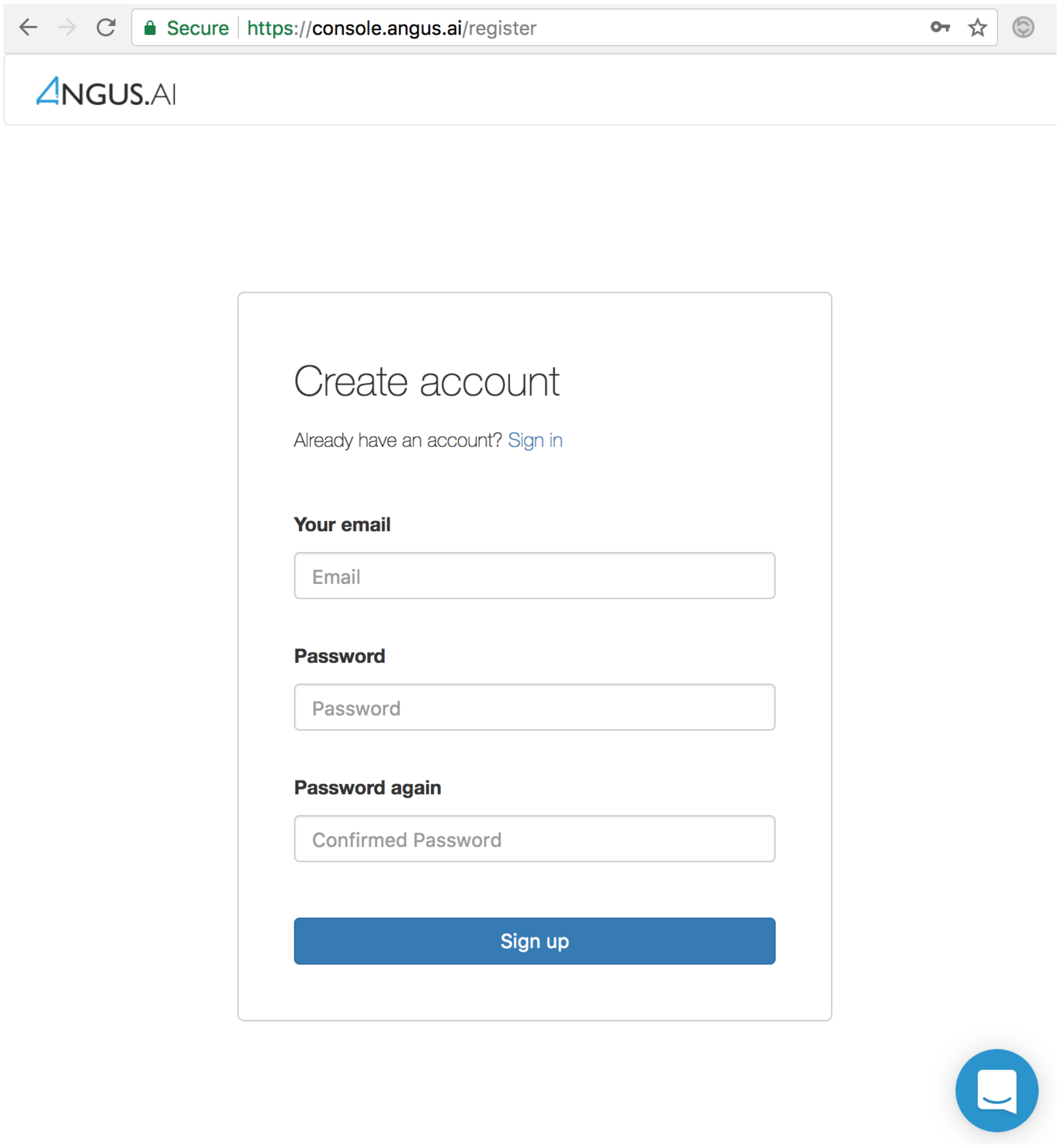
As you go through this tutorial, you will need:

- a computer. Every operating system is ok provided that you can configure a Python or Java stack.
- a camera (e.g. webcam) plugged into that computer. USB and IP cameras are supported, although IP cam can be more challenging to interface. If you need help doing so please contact us at support@angus.ai.
- a working internet connection.

Step 2 - Install our SDK

Create an account

To use Angus.ai services, you need to create an account. This can be done very easily by visiting https://console.angus.ai and filling the form shown below.



Create account

Already have an account? Sign in

Your email

Email

Password

Password

Password again

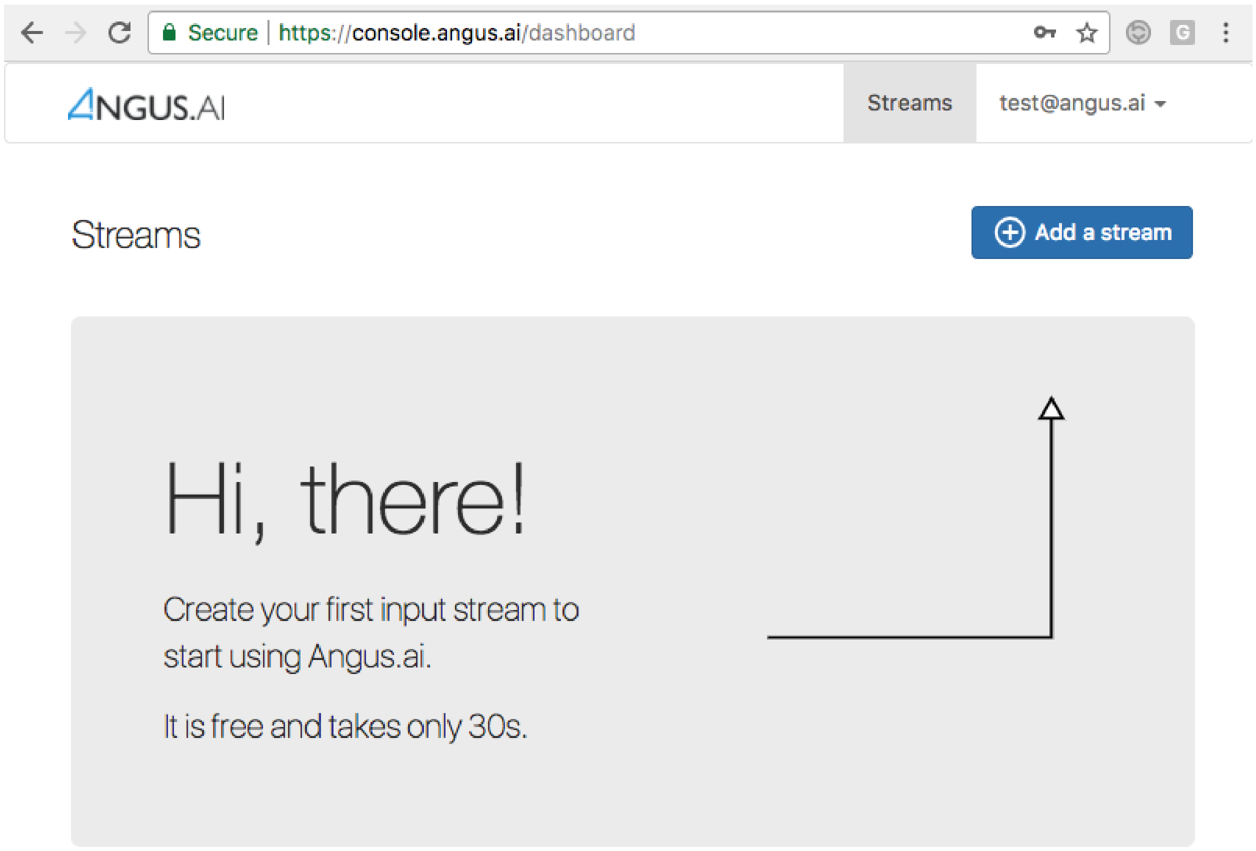
Confirmed Password

Sign up

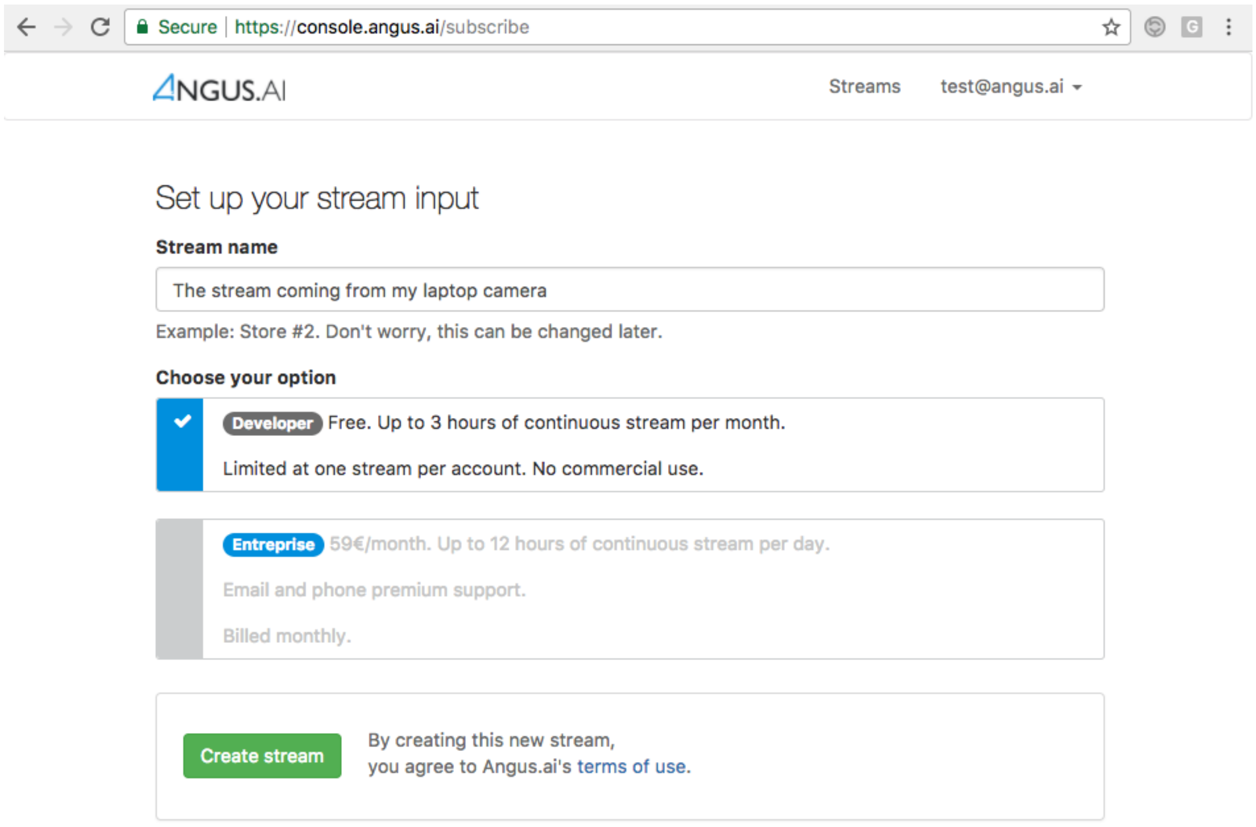
When done, you are ready to create you first camera stream as shown below.

Get credentials for your camera

After creating your personal account on https://console.angus.ai/, you will be asked to create a "stream". This procedure will allow for a private "access_token" and "client_id" keys to be generated for you. This can be done by pressing the "Add a stream" button on the top right hand corner as shown below.


Secure | https://console.angus.ai/dashboard
ANGUS.AI
Streams
test@angus.ai
Streams
Add a stream
Hi, there!
Create your first input stream to start using Angus.ai.
It is free and takes only 30s.

After clicking, you will be asked to choose between a free developer stream and a paying enterprise stream. Please note that the free developer stream is only for non commercial use and will block after 3 hours of video stream computed every month as seen below.



https://console.angus.ai/subscribe

ANGUS.AI Streams test@angus.ai

Set up your stream input

Stream name

The stream coming from my laptop camera

Example: Store #2. Don't worry, this can be changed later.

Choose your option

✓ Developer Free. Up to 3 hours of continuous stream per month.

Limited at one stream per account. No commercial use.

Entreprise 59€/month. Up to 12 hours of continuous stream per day.

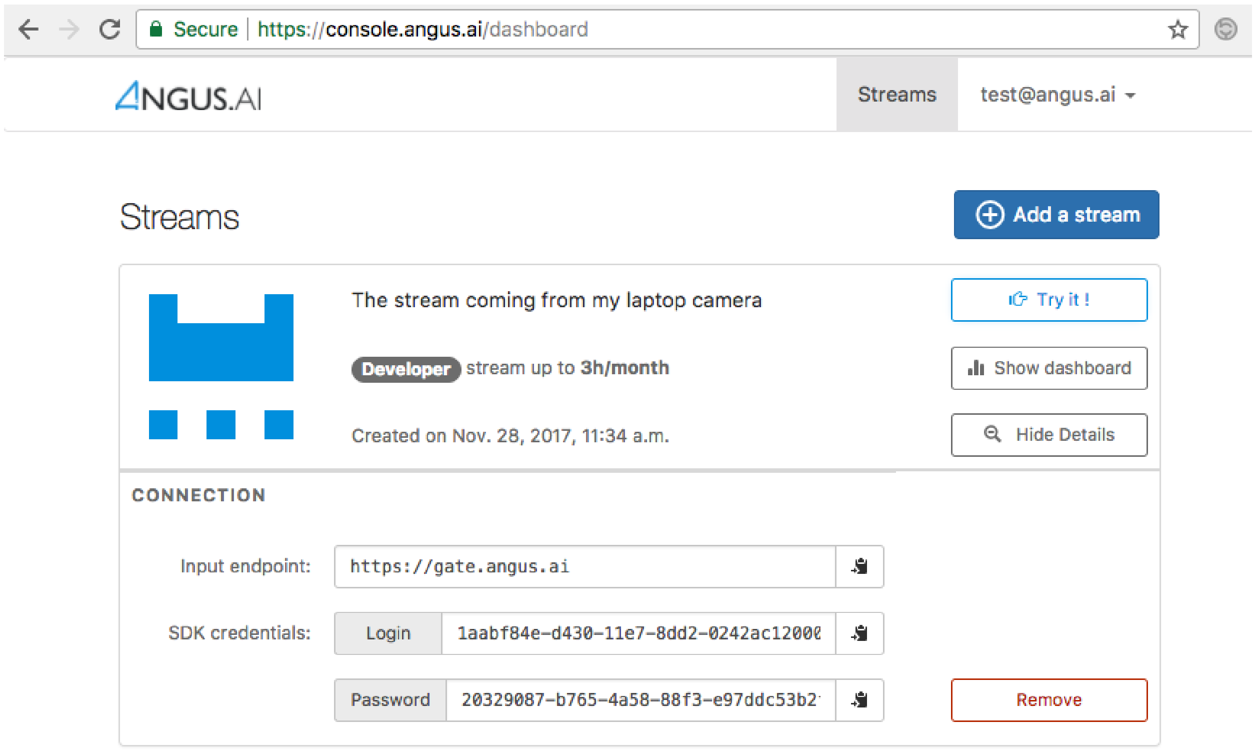
Email and phone premium support.

Billed monthly.

By creating this new stream, you agree to Angus.ai's terms of use.

For an non restricted enterprise stream, you will need to enter a valid credit card number.

Press "Continue" at the bottom of the page and you will soon get the following page. Press "Show Details" and take note of your `client_id` (called Login on the interface) and `access_token` (called Password on the interface) as they will be needed later on.



Streams

The stream coming from my laptop camera

Developer stream up to **3h/month**

Created on Nov. 28, 2017, 11:34 a.m.

CONNECTION

Input endpoint: https://gate.angus.ai

SDK credentials: Login 1aabf84e-d430-11e7-8dd2-0242ac12000

Password 20329087-b765-4a58-88f3-e97ddc53b2

The credentials that you have just created will be used to configure the Angus.ai SDK. Your are now ready to proceed to the next step.

Download and configure the SDK

Requirements

- The SDK is Python3 compatible but the documentation code snippets are only Python2 compatible.
- Also, you might want (not mandatory) to create a python virtual environnement with **virtualenv** in order to install the sdk in there.

To do so, please refer to the following virtualenv guide for more information.

Install the SDK

Open a terminal and install the angus python sdk with pip. If you do not use **virtualenv** you may need to be root, administrator or super user depending on your platform (use sudo on linux platform).

```
$ pip install angus-sdk-python
```

Configure your SDK

You must configure your sdk with the keys you received by creating a stream here. These keys are used to authenticate the requests you are about to send.

Your API credentials can be retrieved by clicking on “Show details” on the stream you just created.

In a terminal, type:

```
$ angusme
Please choose your gateway (current: https://gate.angus.ai):
Please copy/paste your client_id: xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx
Please copy/paste your access_token: xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx
```

Fill in the "client_id" prompt with the "login" given on the interface and the "access_token" prompt with the "password" given on the interface.

On **Windows** system, if angusme does not work, please refer to the *FAQ* for more details.

You can check this setup went well by typing the following command and checking that our server sees you:

```
$ angusme -t
Server: https://gate.angus.ai
Status: OK
```

If this command gives you an error, check that you enter the right "client_id" and "acccess_token". You can do this by re-typing "angusme" in a command prompt.

If you need help, contact us here : support@angus.ai !

Step 3 - Pick your building block

What next?

Congratulations! You went through all the steps to use our building blocks.

- When time comes, you can plug more cameras by creating additional stream as shown here (create-stream).
- If you need to deploy your system in a situation where internet bandwidth is a problem, please contact us at support@angus.ai.

For any issues please contact Angus.ai team at: support@angus.ai, and if possible, please specify your operating system, Python version, as well as the error backtrace if any. Thanks!

Python SDK

Our SDK are here to help you call Angus.ai http API easily, without drafting the appropriate HTTP request yourself. Installing and configuring one of our SDKs is needed to run:

- the audience analytics client applications shown here (apps)
- and/or the building blocks code samples documented here (*Building Blocks*)

Don't want to use Python?

If the SDK in the language of your choice is not provided here, you can:

- contact us at support@angus.ai.
- or use our http API directly by referring to our full API reference (*Main API Reference*)

Requirements

- The SDK is Python3 compatible but the documentation code snippets are only Python2 compatible.
- Also, you might want (not mandatory) to create a python virtual environnement with **virtualenv** in order to install the sdk in there.

To do so, please refer to the following virtualenv guide for more information.

Install the SDK

Open a terminal and install the angus python sdk with pip. If you do not use **virtualenv** you may need to be root, administrator or super user depending on your platform (use sudo on linux platform).

```
$ pip install angus-sdk-python
```

Configure your SDK

You must configure your sdk with the keys you received by creating a stream here. These keys are used to authenticate the requests you are about to send.

Your API credentials can be retrieved by clicking on “Show details” on the stream you just created.

In a terminal, type:

```
$ angusme
Please choose your gateway (current: https://gate.angus.ai):
Please copy/paste your client_id: xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx
Please copy/paste your access_token: xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx
```

Fill in the “client_id” prompt with the “login” given on the interface and the “access_token” prompt with the “password” given on the interface.

On **Windows** system, if angusme does not work, please refer to the *FAQ* for more details.

You can check this setup went well by typing the following command and checking that our server sees you:

```
$ angusme -t
Server: https://gate.angus.ai
Status: OK
```

If this command gives you an error, check that you enter the right “client_id” and “acccess_token”. You can do this by re-typing “angusme” in a command prompt.

If you need help, contact us here : support@angus.ai !

Access your sensor stream

Angus.ai API is specifically designed to process a video stream. This section will show you a way to access the stream of a webcam plugged to your computer by using OpenCV2.

Note that the following code sample can be adapted to process a video file instead.

Note also that OpenCV2 is not an absolute pre-requisite, the following code sample can easily be adapted to be used with any other way of retrieving successive frames from a video stream. If you need assistance, please contact us at support@angus.ai

Prerequisite

- you have a working webcam plugged into your PC
- you have installed **OpenCV2** and **OpenCV2** python bindings. Please refer to OpenCV documentation to proceed, or check *FAQ* chapter.

On Debian-like platform, **OpenCV2** comes pre-installed, you just need to run

```
$ sudo apt-get install python-opencv
```

Then copy this code snippet in a file and run it.

```
# -*- coding: utf-8 -*-
import cv2

def main(stream_index):
    camera = cv2.VideoCapture(stream_index)
    camera.set(cv2.cv.CV_CAP_PROP_FRAME_WIDTH, 640)
    camera.set(cv2.cv.CV_CAP_PROP_FRAME_HEIGHT, 480)
    camera.set(cv2.cv.CV_CAP_PROP_FPS, 10)

    if not camera.isOpened():
        print("Cannot open stream of index {}".format(stream_index))
        exit(1)

    print("Video stream is of resolution {} x {}".format(camera.get(3), camera.
↪get(4)))

    while camera.isOpened():
        ret, frame = camera.read()
        if not ret:
            break

        cv2.imshow('original', frame)
        if cv2.waitKey(1) & 0xFF == ord('q'):
            break

    camera.release()
    cv2.destroyAllWindows()


if __name__ == '__main__':
    ### Web cam index might be different from 0 on your setup.
    ### To grab a given video file instead of the host computer cam, try:
    ### main("/path/to/myvideo.avi")
    main(0)
```

```
$ python yourcopiedfile.py
```

Check that your web cam video stream is correctly displayed on your screen.

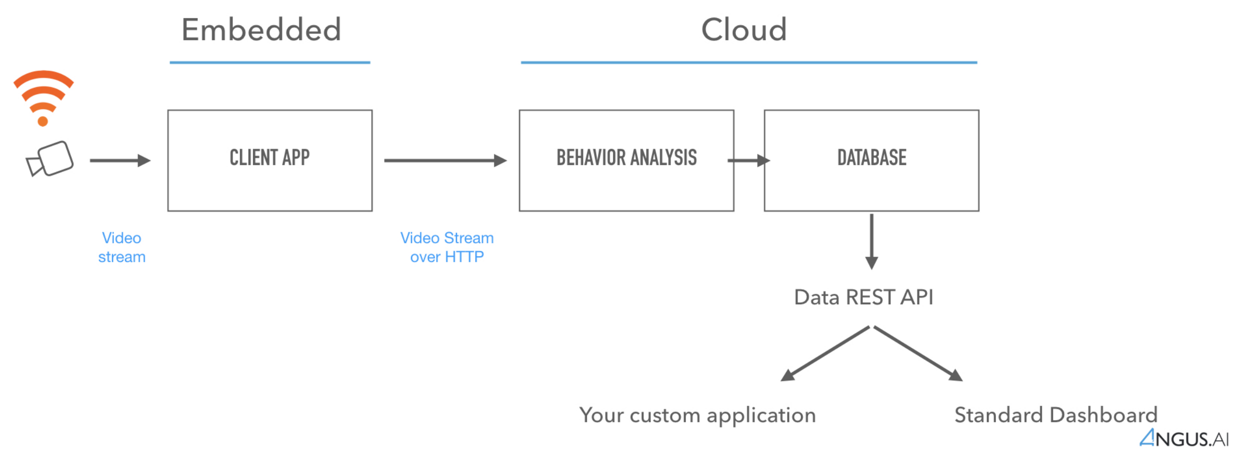
You are setup to start using Angus.ai services:

- our plug and play audience analytics solution here (*Audience Analytics*).
- one of our building blocks here (*Building Blocks*).

Retrieve your data

API Authentication

This documentation is aimed at developers wanting to retrieve programmatically the data computed by Angus.ai audience analytics solution through our Data REST API (see diagram below).



Prerequisite

This procedure requires that you already have a properly configured audience analytics client application running on your device. If this is not the case, please follow our step by step instruction here: (*Audience Analytics*).

API Authentication

Info

You need a JSON Web Token ("JWT") token in order to securely call the data api endpoint. Your personal JWT is provided by programmatically calling the appropriate endpoint documented below.

Endpoint and parameters

To retrieve a JWT token, you have to make a `POST` request to:

`https://console.angus.ai/api-token-authstream`

- **Description**: retrieve a JWT token associated to
- **Authentication**: none
- **Parameters**:
  - `username`: your console login (email)
  - `client_id`: the client_id associated with your stream
  - `access_token`: the access_token associated with your stream
- **Response Code**: 200 OK
- **Response**: JSON

Example

Request:

```
$ curl -X POST -H "Content-Type: application/json" -d '{"username":
↪"aurelien.moreau@angus.ai", "client_id":      "3bd15f50-c69f-11e5-ae3c-
↪0242ad110002", "access_token": "543eb007-1bfe-89d7-b092-e127a78fe91c"}'   ␣
↪https://console.angus.ai/api-token-authstream/
```

Response:

```
{
  "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
↪eyJ1c2VybmFtZSI6ImF1cmVsaWVuLm1vcmVhdUBhbmd1cy5haSIsIm9yaWdfaWF0IjoxNTA1Mzk4MDM4LCJleHAiOjE1D8
↪K70YXQYMAcdeW7dfscFGxUhenoXXGBAQTiWhNv-9cVc"
}
```

Once provided, you will need to put this token as a HTTP header `Authorization: Bearer [YOURJWTTOKEN]` (see the Python example in *Retrieving the data*) in every HTTP requests you make.

Retrieving the data

Once you obtained your personal JWT, you can start retrieving your data by calling the endpoint documented in the *Data API Reference* page.

Python example

For this example, you will need to install `requests` and `pytz` modules

```
import requests
import pytz
import datetime
import json


def get_token():
  data = {
    "username": "test@example.com",
    "client_id": "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx",
    "access_token": "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx",
  }

  req = requests.post('https://console.angus.ai/api-token-authstream/', json=data)
  req.raise_for_status()
  req = req.json()

  return req['token']


def get(token, metrics, from_date, to_date, size):
  entities_url = 'https://data.angus.ai/api/1/entities'
  params = {
    "metrics": ",".join(metrics),
    "from_date": from_date.isoformat(),
    "to_date": to_date.isoformat(),
    "time": size,
  }

  headers = {
    "Authorization": "Bearer {}".format(token)
  }

  req = requests.get(entities_url, params=params, headers=headers)
  req.raise_for_status()
  req = req.json()

  return req


def get_overall(token):
  to_date = datetime.datetime.now(pytz.UTC)
  from_date = to_date - datetime.timedelta(hours=24)

  metrics = [
    "passing_by",
    "interested",
    "stop_time",
    "attention_time",
  ]

  return get(token, metrics, from_date, to_date, "global")


def main():
  token = get_token()
  overall = get_overall(token)
  print(json.dumps(overall, indent=2))


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

The metrics

1. Passing By

- **Keyword**: `passing_by`
- **Schema**:

```
{ "value": 0 }
```

- **Description**: Count of people who passed (not necessarily stopping or looking) in front of the camera during at least 1 second.

2. Interested

- **Keyword**: `interested`
- **Schema**:

```
{ "value": 0 }
```

- **Description**: Count of people who stopped for at least 3 seconds and looked in the direction of the camera more than 1 second.

3. Average stopping time

- **Keyword**: `stop_time`
- **Schema**:

```
{ "value": null }
```

- **Description**: Average time a person, among the "interested" people (see *above*), stay still in front of the camera. (in second)

4. Average attention time

- **Keyword**: `attention_time`
- **Schema**:

```
{ "value": null }
```

- **Description**: Average time a person, among the "interested" people (see *above*), spend looking at the camera. (in second)

5. Category

- **Keyword**: `category`
- **Schema**:

```
{
  "senior_female": 0,
  "senior_male": 0,
  "young_female": 0,
  "young_male": 0
}
```

- **Description**: Population segmentation counts of all the "interested" people (see *above*) for each category.

Note: When no age or gender has been found about an interested person, it will not be included in any of these category.

6. Gender

- **Keyword**: `gender`
- **Schema**:

```
{
  "?": 0,
  "female": 0,
  "male": 0
}
```

- **Description**: The gender repartition of all the "interested" people (see *above*).

Data API Reference

Get entities

Api endpoint for fetching filtered aggregated data (called `entities`)

- **URL**

  /api/1/entities

- **Method:**

  GET

- **URL Params**

  Required:

  - `metrics=[string]`: a list of desired information from the db (*comma separated without whitespaces*)
    - Possible values: `interested`, `passing_by`, `stop_time`, `attention_time`, `category`, `gender`, `satisfaction`
    - Default value: none
  - `from_date=[iso-date]`: the date to start the search from in *iso format urlencoded* (ex: 2017-09-03T05:45:00+0200 becomes 2017-09-03T05%3A45%3A00%2B0200)
    - Default value: none

  Optional:

  - `to_date=[iso-date]`: the date to end the search to in *iso format urlencoded*
    - Default value: the current date
  - `time=[string]`: a time bucket to aggregate data into
    - Possible values: `by_hour`, `by_day`, `global`
    - Default value: `global`
  - `page=[integer]`: a page if enough results for pagination.
    - Default value: 1

- **Success Response:**
  - **Code:** 200
  - **Json content:**
    - `entities=[json]`: the actual data returned by the api
    - `time=[str]`: the actual time bucket used to return data
    - `from_date=[iso-date]`: the date from which the search has been made

  * `to_date=[iso-date]`: the date to which the search has been made
  * `total_results=[integer]`: the total number of results for this search
  * `nb_of_pages=[integer]`: the total number of pages for paginated results
  * `page=[integer]`: the current retrieved page
  * `next_page=[str]`: the complete URL to call to get the results for the next page

```
{
  "entities": {
    "date1": {
      "metric1": {
        "value": 3
      }
    },
    "date2": {
      "metric1": {
        "value": 5
      }
    },
    ...
    ...
    ...
  },
  "from_date": "2017-09-03T05:45:00+02:00",
  "to_date": "2017-09-14T16:53:11+02:00",
  "time": "by_day",
  "total_results": 63,
  "nb_of_pages": 2,
  "next_page": "https://data.angus.ai/api/1/........&page=2",
  "page": 1,

}
```

- **Error Response:**
  - **Code:** 401 UNAUTHORIZED
  - **Explanation:** If no "Authorization" header is provided or if there is a problem with the JWT token, the error message will explain the problem

  OR

  - **Code:** 400 BAD REQUEST
  - **Explanation:** If the request is not well formatted (for instance, a required param is not provided, etc...) or any other kind of problem with the request, the error message should be self explicit

- **Sample Call:**

  Here is an example of a request for all the metrics between September 3rd 2017, 5:45 GMT+2 until now, using a time bucket of "one day".

```
$ curl -X GET -H 'Authorization: Bearer  eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
↪eyJ1c2VybmFtZSI6ImF1cmVsaWVuLm1vcmVhdUBhbmd1cy5haSIsIm9yaWdfaWFfta0IjoxNTA1Mzk4MDM4LCJleHAiOjE
↪K70YXQYMAcdeW7dfscFGxUhenoXXGBAQTiWhNv-9cVc' 'https://data.angus.ai/api/1/
↪entities?metrics=satisfaction,gender,category,passing_by,interested&from_
↪date=2017-09-03T05%3A45%3A00%2B0200&time=by_day
```

CHAPTER 3

FAQ

Cameras / Images Requirements

Do I need a specific camera?

No, our solutions are made to work with any cameras (IP cam or USB webcam). At Angus.ai, we use 50$ Logitech USB webcam on a daily basis, with no problem at all.

What are the supported image formats?

The supported formats are: JPEG and PNG.

What image resolution should I use?

640x480 (aka VGA) images are a good start. Using bigger images will increase the ability of the system to detect faces that are further away from the camera, but will also lead to bigger latencies.

What frame rate should I use?

To ensure proper analysis from our services, make sure to provide about 10 frames per second.

Angus SDK, Python, OpenCV

What are the requirements to run Angus SDKs?

Nothing, the SDKs come with their dependencies (managed by pip). But, in order to access your webcam stream, you will need a dependency that is not packaged into our SDK. We tend to use OpenCV a lot to do this (see other questions below).

Is Python SDK Python 3 compatible?

Yes, it is. But the documentation code snippets and OpenCV2 are only Python 2. Sorry for the inconvenience, the Python 3 documentation is in progress.

How to install OpenCV2 and its Python bindings on debian-like systems?

Please, use:

```
$ apt-get install python-opencv
```

How to install OpenCV2 on other systems?

Please follow official documentation here. For windows, check the complete guide on this FAQ.

Windows related questions

How can I install Pip in Windows?

Pip is installed by default when you install Python `2.7.x`, please use the latest Python 2.x version available.

How can I run all python code snippets on Windows?

Please use the latest Python 2.x version (with pip) `2.7.12`. Windows installer puts Python in `C:\Python27` by default, if you choose an other directory, please replace "c:Python27" by your chosen directory in the following instructions:

In a Command Prompt go to python `\Scripts` directory:

```
$ cd C:\Python27\Scripts
```

Install numpy and Angus Python SDK:

```
$ pip install numpy angus-sdk-python
```

Configure Angus SDK:

```
$ cd C:\Python27
$ python Scripts\angusme
```

To install OpenCV, download OpenCV for Windows from http://opencv.org/, execute (or unzip) it. Copy `<opencv_directory>\buid\python\2.7\[x86|x64]\cv2.pyd` in `C:\Python27\Lib`.

Now you can run all Python snippets of the documentation.

Message "Input does not appear to be valid...." on Windows?

Make sure you use the binary file mode when opening images:

```
open("/path/to/your/image.png", "rb")
```