motion と OpenCV を組み合わせる

それでは最後に motion と OpenCV を組み合わせてみよう。 motion に OpenCV の顔検出をあわせると Web ブラウザ上に (つまりネットワーク越しに) カメラ映像を表示させながら、顔を検出すると LED を点灯させることができる。

簡単な例として LED を用いているが、顔を検出したら

- メールを送る
- LINE に送る
- 音声で話しかける

などなにをするか、応用はアイデア次第である。

motion はネットワーク上にライブ映像を流している (ライブストリーミング) ので、 その出力を顔検出プログラムに入力すれば良い。

以下のように motion から枝分かれするので、motion の映像に枠を描画することはできない。 注意しよう。

```
[USBカメラ]
     ↓
[motion: MJPEGストリーム配信]
     ↓                    ↓
[OpenCV: 顔検出 + 枠描画] [ブラウザで表示]
```

プログラムの変更

プログラムの変更は実は簡単である。

motion の起動

motion を起動する。

```
$ sudo systemctl restart motion
```

顔検出プログラムの「カメラモジュール」だったところを「motion のライブストリーミング」に切り替える。

```
# 変更前
cap = cv2.VideoCapture(0) # 0: USB カメラ
# 変更後
stream_url = "http://kubota-pi:8081"
cap = cv2.VideoCapture(stream_url)
```

Python を実行し動作確認してみよう。同時に以下のアドレスにも Web ブラウザでアクセスしてみよう。

```
http://kubota-pi:8081/0/stream
```

■ ここに注目！

USBカメラの映像と、motion のネットワークストリーミングの映像は、まったくその形態が違うにも関わらず、我々のプログラムでは cv2.VideoCapture() に与える引数を変えるだけですんでいる。

むずかしいことは OpenCV が良きにはからってくれている。

これを **ソフトウェアの抽象化 (Abstraction)** という。抽象化とは **複雑な内部の仕組みを隠し、シンプルで使いやすい機能だけを利用者に見せること**を指す。今回の例では、OpenCVがUSBカメラやストリーミングの詳細なデータ取得方法を隠蔽し、read() という統一されたシンプルな操作を提供していることが「抽象化」にあたる。

- **インターフェース (Interface)**
  抽象化された機能を利用するための「窓口」や「接点」のことをインターフェースという。
  cv2.VideoCapture() で作られたオブジェクトが持つ read() や release() といったメソッド (関数のことだ) 群が **インターフェース** である。利用者はこの決められた作法（インターフェース）に従うだけで、内部の実装を気にせず機能を使えるようになる。

われわれもプログラムを書くときは、このように (中をうまく隠すように) 書くように心がけよう。