2015

Localisation d'entité par analyse d'image

LUDOVIC BOUAN DU CHEF DU BOS
THOMAS LEFORT

TIPE ECAM LYON PTSI2

RESUME

FRANÇAIS

Nous avons réalisé un support de webcam ainsi qu'un programme en Python permettant à un sujet d'être suivi par la webcam grâce à la reconnaissance de visage. Pour cela, nous avons étudié le système de fonctionnement de la reconnaissance faciale, avons conçu un mécanisme adapté, puis avons, par l'expérience, améliorer l'efficacité de ce mécanisme. Nous sommes finalement arrivés au résultat attendu en début de réflexion.

ENGLISH

We have built a rotating webcam support alongside a Python program that follows the subject thanks to facial recognition. To do this, we have studied the workings of facial recognition algorithms, created a suitable mechanism, and experienced with our setup to optimize its functionality. In the end, we successfully accomplished our original goals.

TABLE DES MATIÈRES

INTRODUCTION

L'analyse informatique de la position d'un objet dans l'espace permet à l'ordinateur d'interagir avec son environnement, de la même manière que le ferait l'Homme. L'individu peut percevoir son espace avec 4 de ses sens (vue, ouïe, odorat et toucher). Pour un ordinateur il existe aussi plusieurs méthodes de perception de l'environnement. Nous avons décidé d'utiliser une de ces méthodes pour permettre à une webcam de suivre la personne qu'elle filme. La première solution envisagée pour localiser un objet par informatique était la triangulation d'un signal radio. Le problème de cette solution était la portée entre l'émetteur et le récepteur, la nécessité de multiplier ces capteurs, ainsi que la précision. Nous avons ensuite envisagé la solution de l'analyse d'image. Pour cela nous utilisons OpenCV, une bibliothèque qui permet de faciliter l'analyse d'image en appliquant la méthode de Viola-Jones.

L'intégralité de notre projet, ainsi que les instructions pour le faire fonctionner, se trouve sur GitHub à l'adresse suivante : http://goo.gl/awhLUy

METHODE DE DETECTION DE VISAGE

INTRODUCTION

La méthode de Viola-Jones a été introduite en 2001 par Paul Viola et Michael Jones. Révolutionnaire en matière de vitesse, efficacité, et précision, cet algorithme a provoqué la banalisation de la détection de visages au début du 21ème siècle. Aujourd'hui on le trouve dans la plupart des appareils photo, téléphones portables, ou logiciels de photographie. Dans l'industrie cette méthode est le plus souvent utilisée par le biais d'OpenCV, une bibliothèque que l'on utilisera dans notre production.

Cette méthode se compose de deux étapes majeures : l'entrainement et la détection. L'entrainement est un apprentissage préalable effectué par l'algorithme Adaboost. Dans le cas de la détection de visages, une base de données d'entrainement massive, composé de photos contenant ou non des visages, permet de créer un fichier de données que l'algorithme de Viola-Jones pourra utiliser par la suite pour la détection. Nous allons comprendre en détail ce fonctionnement.

CARACTERISTIQUES PSEUDO-HAARS ET CLASSIFIEURS FAIBLES

Dans le domaine de la vision par ordinateur, les motifs pseudo-Haars sont des éléments qui permettent de rapidement détecter des attributs dans une image numérique. Voici une visualisation de quelques exemples utilisés lors des premières étapes de l'algorithme de Viola-Jones.

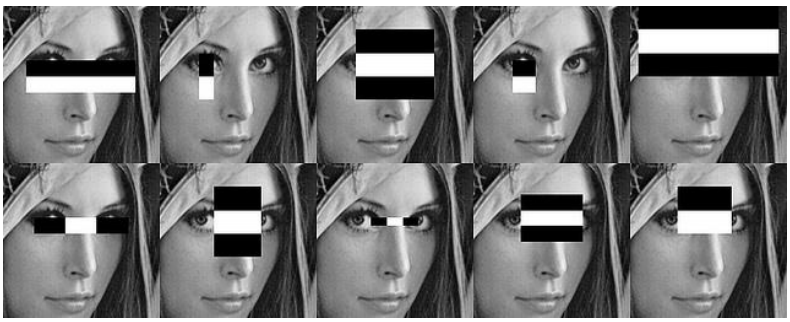
Les caractéristiques pseudo-Haars sont simplement un calcul de différence. L'image est tout d'abord convertie en nuance de gris. Chaque pixel de l'image est donc caractérisé par une seule valeur d'intensité (entre 0 et 255 pour une image en 8 bits.) On évalue la différence entre la somme des pixels dans la zone blanche et la somme des pixels dans la zone noire. Cette différence sera par la suite comparée à un seuil établi préalablement par l'algorithme d'entrainement, Adaboost.

Individuellement, chacune de ces caractéristiques ne permet pas d'affirmer ou non la présence d'un visage dans une image. Chaque caractéristique pseudo-Haars permet de construire une fonction de seuil appelée classifieur faible. Si la différence calculée est supérieure ou inférieure au seuil, cela correspond à la présence ou à l'absence d'un visage (cette polarité est aussi déterminée par Adaboost). Le classifieur faible prend comme valeur ±1 selon les cas :

$$h_j(x) = \begin{cases} -s_j \; si \; f_j < \theta_j \\ s_j \; sinon \end{cases}$$

(h_j est le classifieur faible, f_j est la caractéristique pseudo-Haars, s_j et θ_j correspondant à la polarité et au seuil sont déterminé par Adaboost)

Cependant, il existe plus de 160 000 caractéristiques pseudo-Haars possibles dans une fenêtre d'analyse de 24x24 pixels. Pour chacune de ces caractéristiques, et pour chaque section de 24x24 pixels d'une image, il faudrait faire la somme de chaque pixel concerné pour ensuite faire la différence entre la zone blanche et la zone noire. Autrement dit, c'est une tache calculatoire quasi-impossible à faire et certainement pas adaptée pour une analyse en temps réel. Viola et Jones ont donc apporté une solution à ces problèmes : l'image intégrale qui facilite les calculs de somme, et une adaptation de Adaboost qui réduit le nombre de classifieurs faibles utilisés.



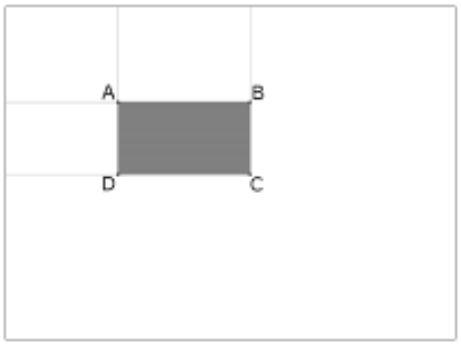
Exemples de bonnes caractéristiques pseudo-Haars

IMAGE INTEGRALE

Pour leur algorithme, Viola et Jones ont aussi introduit le concept d'image intégrale. L'image intégrale est une représentation intermédiaire de l'image qui permet de calculer la somme d'une zone de pixels très rapidement. Dans une image intégrale chaque pixel prend la valeur de la somme de l'ensemble des pixels à la gauche et au-dessus de lui-même, comme l'indique cette formule :

$$ii(x, y) = \sum_{x' \leq x,\, y' \leq y} i(x', y')$$

(ii(x, y) correspond à l'intensité intégrale du pixel (x, y) et i(x, y) correspond à l'intensité du pixel (x, y))



Il suffit donc de construire cette image intégrale au début de l'analyse. Ensuite le calcul de la somme des pixels dans un rectangle donné est simplifié à quatre opérations :

$$\sum_{\substack{x_A < x' \leq x_C \\ y_A < y' \leq y_C}} i(x', y') = ii(A) + ii(C) - ii(B) - ii(D)$$

ADABOOST

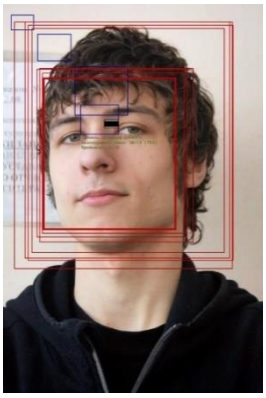
Adaboost est un algorithme d'apprentissage automatique (machine learning), développé par Yoav Freund et Robert Schapire et adapté par Viola et Jones. Il permet de sélectionner les caractéristiques pseudo-Haars importantes et significatives parmi les 162 336 possibles dans une fenêtre d'analyse de 24x24 pixels, établir le seuil minimal optimal pour les classifieurs faibles, et ensuite créer des classifieurs forts.

Adaboost sélectionne les caractéristiques pseudo-Haars qui sont efficaces à la reconnaissance faciale en les appliquant à des images d'une base de données, où l'on a précédemment défini la présence ou non de visage. Il compare le taux d'erreur minimal de chaque classifieur faible pour les classer par ordre de pertinence et leurs attribuent une pondération. Un classifieur fort est simplement une combinaison linéaire de classifieurs faibles avec leurs coefficients correspondant.

C'est un processus qui est fait une fois et n'est pas effectué à nouveau lors de la détection. Son procédé sort du cadre de notre analyse, mais un complément sur le fonctionnement d'Adaboost est donné en annexe.

CASCADES

Le système de cascade est le dernier élément qui assure la rapidité de la méthode de Viola-Jones. Pour analyser une image, l'algorithme parcourt l'intégralité de l'image avec une "fenêtre" d'analyse de 24x24 pixels (par défaut). Pour éviter de faire une analyse complète sur chaque fenêtre, les différentes cascades fortes sont organisées en différentes étapes (une trentaine pour la détection de visages). Les premières étapes sont grossières et demandent relativement peu de calculs. Ils détectent quasiment 100% des visages, mais laissent passer environ 50% de faux positifs. On passe seulement à la prochaine étape si un visage est détecté; chaque étape successive confirme ou non la présence d'un visage avec des critères de plus en plus restrictifs et avec de plus en plus de classifieurs faibles mis en jeu. Cela permet d'éliminer très rapidement et avec peu de calculs les sections d'une image qui ne contiennent pas de visage.



Lorsque la fenêtre passe plusieurs fois sur le même visage il la détecte plusieurs fois. Dans OpenCV, on peut demander alors que l'algorithme considère un visage seulement s'il a au moins un certain nombre de fenêtres à résultat positif voisines ; deux fenêtres étant dites voisines si le centre de l'une est contenu dans l'autre. Cela permet d'augmenter d'autant plus la précision de l'algorithme.

Voici une vidéo permettant de visualiser au ralenti les étapes de l'analyse d'une image. Cette procédure est en réalité effectuée en un instant :

http://goo.gl/wo9kkJ

MISE EN ŒUVRE PRATIQUE

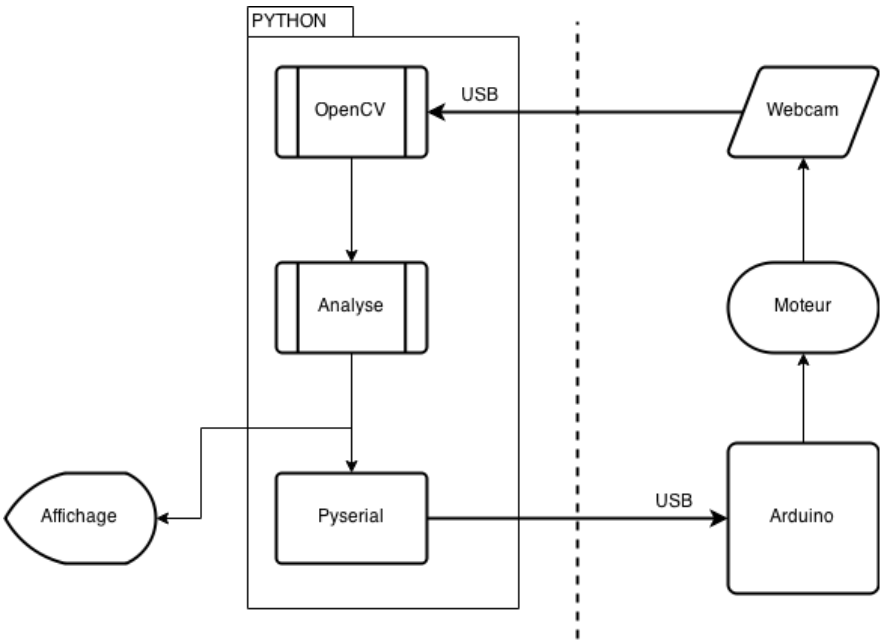
ANALYSE DU PROGRAMME

APPROCHES ADOPTEES

Avant d'entamer la réalisation de ce programme, nous avons établi un cahier des charges, établissant les fonctionnalités principales que nous avons voulu mettre en œuvre.

F1	Cadrer le(s) sujet(s) au centre de l'image	
	F1.1	Envoyer aux moteurs les angles de manière à ce que le sujet soit au centre de l'image horizontalement et verticalement
	F1.2	Définir une zone autour du center pour laquelle on considérera le sujet centré
	F1.3	Etre capable de centrer plusieurs sujets
F2	S'adapter à différentes performances d'ordinateurs	
	F2.1	Permettre un choix de la résolution de la vidéo analysée
	F2.2	Permettre un choix du ratio de nombre d'images analysées
F3	Affichage de la vidéo analyse	
	F3.1	Permettre un choix d'afficher ou non la vidéo
	F3.2	Mettre en évidence les faces détectées par le programme

La partie informatique de ce projet repose sur 3 éléments : le langage de programmation Python, la bibliothèque OpenCV, et le microcontrôleur programmable Arduino. Le principe de fonctionnement peut se décomposer en trois étapes :



- La webcam montée sur les deux moteurs envoie à l'ordinateur un flux images qui sont analysées avec Python et OpenCV.
- Selon les résultats de cette analyse, le programme Python envoi, ou pas, des ordres à la carte Arduino grâce ou module PySerial.
- Finalement, la carte Arduino fait pivoter les moteurs.

Nous avons choisi Python comme langage de programmation car c'est un des quatre langages pour laquelle OpenCV possède une interface, il est gratuit et libre de droits, et c'est un langage dont nous avons tous les deux une certaine maitrise. De plus, son utilisation rentre dans la continuation des cours d'initialisation suivis en début d'année.

OpenCV est une bibliothèque open source d'algorithmes, essentiellement orientée vers la vision artificielle. Elle nous permet, par exemple, d'associer les images de la webcam à des variables Python, détecter des entités et leurs positions dans ces images, ainsi que les afficher sur l'écran.

PySerial est une bibliothèque qui permet d'accéder au port serial de l'ordinateur, assurant ainsi la communication entre notre programme et l'Arduino.

Le programme Arduino est codé en C. La différence de langage n'est pas problématique car la communication par sérial est standardisée.

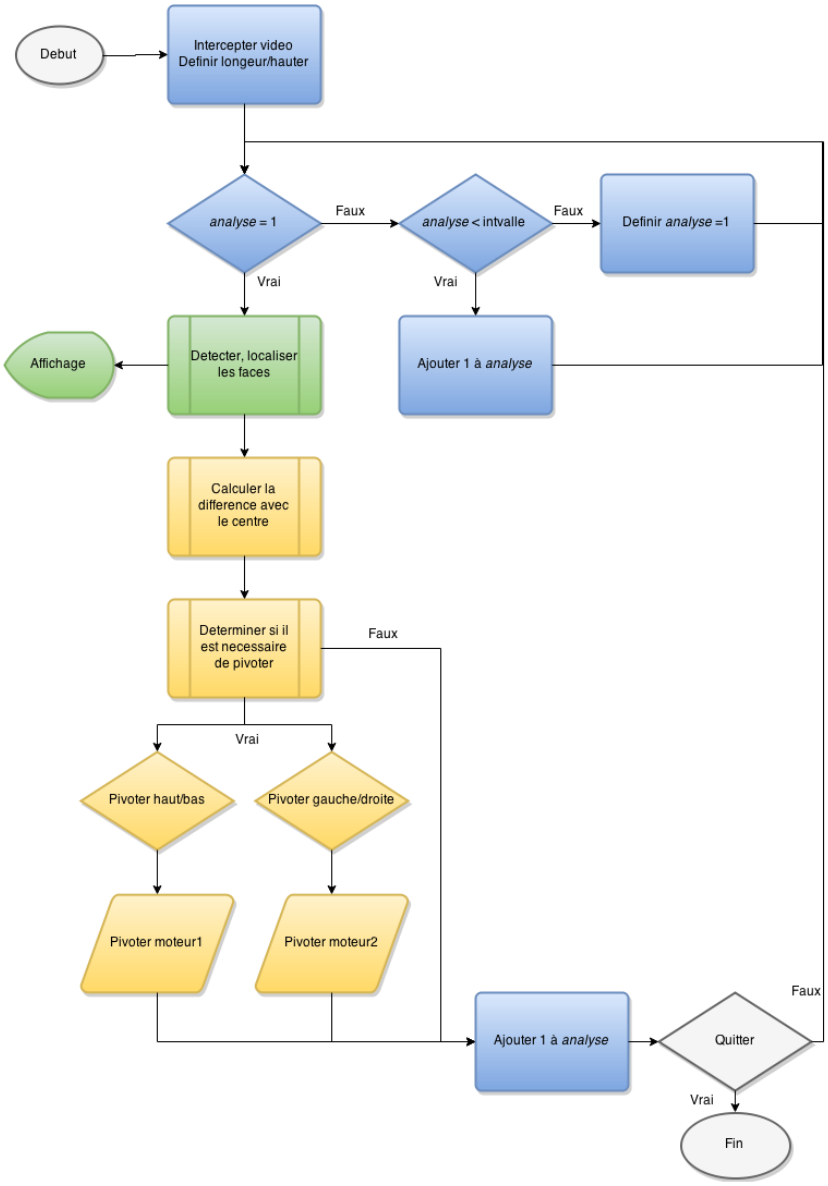
Voici des précisions sur la réalisation des différents points du cahier des charges :

F1 : La vidéo est reçue image par image. Tant que le programme n'est pas arrêté par l'utilisateur, on utilise les algorithmes d'OpenCV pour détecter et localiser les visages présents dans l'image. S'il n'y a qu'un sujet, il suffit de calculer la différence entre celui-ci et le centre de l'image (horizontalement et verticalement), sinon il faut calculer la différence entre le centre et le point médian de toutes les faces détectées (F1.3). Si cette différence est supérieure à un nombre de pixels prédéfini (F1.2), on envoie des nouveaux angles aux moteurs (F1.1). La façon dont on détermine l'angle à envoyer aux moteurs peut se faire de deux manières. On

peut augmenter ou diminuer l'angle du moteur proportionnellement à l'écart du sujet du centre. L'autre solution consiste à augmenter ou diminuer l'angle d'un certain nombre de degrés fixe, on se base ainsi sur le grand nombre d'images analysés par unité de temps.

F2 : Pour pouvoir s'adapter aux différentes performances d'ordinateurs, nous avons décidé d'incorporer deux paramètres qui jouent sur la performance du programme : le ratio d'images analysées et la résolution de la vidéo. Avec OpenCV on peut définir la taille de la vidéo que l'on souhaite capturer. Il suffira alors de demander à l'utilisateur les dimensions qu'il souhaite utiliser au démarrage du programme (F2.1). On utilisera par défaut 640x480. Pour définir un ratio d'images analyse, il faudra utiliser une variable compteur dans le programme. L'utilisateur pourra choisir un intervalle qui correspond au nombre d'images non-analysé entre deux images analysé (F2.2).

F3 : On utilisera des fonctions d'OpenCV pour afficher la vidéo et encadrer les visages détectés.



REALISATION

L'intégralité du code se trouve en annexe (n°1), commenté et expliqué ligne par ligne. Les nombres mis en exposant dans l'explication qui suit correspondent aux lignes du programme.

INITIALISATION

Pour lancer notre programme, il faut saisir les valeurs des quatre variables d'initialisation (annexe n°2). Ces valeurs peuvent être saisies directement dans le cmd/terminal au lancement:

```
python detect.py [a_intvl] [display] [width] [height]
```

Pour faciliter l'utilisation, ces arguments sont optionnels. Dans le cas de leur absence, les valeurs sont demandées une fois la commande exécutée [207]

On initialise ensuite les angles de départ des deux moteurs : 90 pour le moteur d'axe de rotation vertical, et 115 pour le moteur d'axe horizontal (ce qui correspond environ au plat) [265].

LA BOUCLE PRINCIPALE

Le programme fonctionnera jusqu'à ce que l'utilisateur décide de quitter manuellement. Pour cette raison, une fois l'initialisation effectuée, le programme capture la vidéo et rentre dans une boucle [175], arrêtée seulement par la touche "Q" [242]. La boucle est parcourue pour chaque image qui compose la vidéo [178].

Etant donné que l'on veut pouvoir choisir le nombre d'image analysé, à l'initialisation on choisit un nombre supérieur ou égal à 1, qui correspond au nombre d'image non analysées entre 2 images analysées [256].

Si l'image à l'entrée de la boucle est à analyser [181]:

1. Convertir l'image en nuance de gris [184]
2. Utiliser la fonction `findfaces` pour trouver les visages [187]
3. Déterminer les décalages avec le centre grâce à la fonction `offset` [200]
4. Déterminer si les décalages sont en dehors de la zone centrale grâce à la fonction `mvt_filter` [200]
5. Si le décalage est suffisant : [203]
    a. Méthode (1)
        i. Déterminer la moyenne des décalages horizontaux et verticaux [214]
        ii. Multiplier ces moyennes par k (la provenance cette de constante est expliquée par la suite) [214]
        iii. Convertir ce résultat en string en concaténant les résultats obtenus [223]
        iv. Envoyer à l'Arduino [225]
    b. Méthode (2)

   i. Selon le signe du décalage du visage par rapport au centre, ajouter ou enlever un nombre fixe de degré aux angles actuels des moteurs (ici 4) [207]
   ii. Convertir ce résultat en string en concaténant les résultats obtenus [223]
   iii. Envoyer à l'Arduino [225]
6. Ajouter 1 à analyse [228]

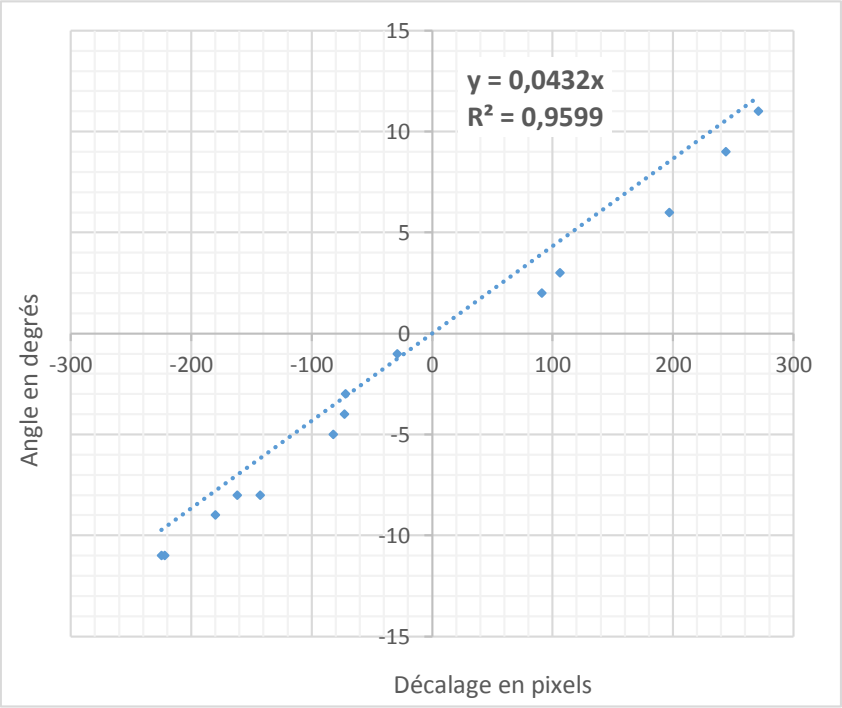
DETERMINATION DE K

Pour améliorer la rapidité et la réactivité de la caméra, nous avons fait évoluer la méthode (2) vers la méthode (1). Au lieu de décaler le moteur progressivement jusqu'à ce qu'il soit bien placé, nous avons essayé de placer le moteur directement à l'angle nécessaire.

k est un coefficient de proportionnalité entre le décalage du visage détecté par rapport au centre de l'image (en pixel) et le nombre de degré de différence qu'il faut au moteur pour centrer le sujet. Ce coefficient dépend de la résolution de la vidéo, et de l'optique de la webcam. On a donc déterminé ce coefficient expérimentalement pour la webcam que l'on utilise.

Pour cela, en utilisant la méthode (2), nous avons fait une dizaine de mesures qui compare le décalage avec le centre et l'angle de rotation nécessaire pour ramener le sujet fixe au centre de la vidéo.

Décalage en pixel	Décalage en angle
-82	-5
244	9
-222	-11
-143	-8
-225	-11
-73	-4
-180	-9
197	6
91	2
271	11
106	3
-72	-3
-162	-8
-29	-1



REALISATION DU SUPPORT

APPROCHES ADOPTEES

Nous avons réalisé, avant d'imaginer le support, un cahier des charges :

F1	Entrainer la webcam en rotation	
	F1.1	Assurer le maintien de la webcam sur le support
	F1.2	Entrainer sur 2 axes
	F1.3	Normaliser les éléments qui constituent les liaisons pivots
F2	Intégrer la carte Arduino au support	
	F2.1	Assurer la protection de la carte Arduino
	F2.2	Permettre la connexion des servomoteurs à la carte
F3	Doit être imprimé grâce à une imprimante 3D	
	F3.1	Limiter la quantité de matière utilisée
	F3.2	Respecter les dimensions de l'imprimante
	F3.3	Limiter le poids à supporter

Le support est adapté à la webcam utilisée. Nous avons reproduit la fixation du support de la webcam pour que celle-ci soit facile à monter. Nous avions d'abord pensé mettre une fixation pour appareil photo mais il y avait 2 points difficiles à prendre en compte : les pas de vis Kodak, qui permettent de fixer des appareils photo n'ont pas un pas métrique, ni américain en pouces mais un pas anglais qui a un angle de filetage différent. Le deuxième difficulté était de concevoir un support qui ne casse pas sous le poids d'un appareil photo.

Ensuite, pour que la caméra puisse filmer l'ensemble de son environnement, nous avons créé un système qui assure une rotation autour de l'axe vertical et d'un des axes horizontaux.

REALISATION

L'imprimante 3D n'ayant pas une très bonne précision d'impression il a fallu usiner certaines pièces.

Tous les trous avaient été définis avec un diamètre nominal de 3mm. Après impression certains trous mesurait moins de 2 mm. Ils ont donc été tous percé à nouveau avec un foret qui a une meilleure précision.

Ensuite, à la conception, nous n'avions pas prévu l'épaisseur des têtes de vis. Nous avons donc fraisé tous les trous pour pouvoir mettre des têtes fraisées. Cela permet aussi d'améliorer le côté esthétique.

Toutes les pièces sont vissées entres elles à l'exception de la liaison entre le palonnier et l'arbre. Cette liaison doit être la plus rigide possible car elle entraine les axes. La réaliser avec une vis n'aurait pas assuré une résistance suffisante. Ce montage a donc été réalisé avec une colle sous forme de gel, ce qui permet une certaine élasticité au montage.

Les fils de connexion entre le moteur supérieur et la carte étant trop court, nous les avons rallongés. Pour assurer la pérennité des connexions, nous avons ajoutés une gaine autour des fils qui ajoute un côté esthétique au montage.

CONCLUSION

Nous sommes heureux d'avoir pu réaliser l'ensemble de nos objectifs pour ce projet. Nos premiers essais avec la méthode 2 étant fonctionnelles mais peu satisfaisant, nous avons donc conçu et mis en œuvre la méthode 1 avec succès. Notre production répond à toutes les contraintes du cahier des charges. En plus de cela nous avons décidé de détecter et encadrer les yeux dans l'image pour avoir un affichage plus complet.

Cependant, nous aurions voulu implémenter une fonction supplémentaire pour améliorer le projet. Il arrive que lorsque il y a un sujet devant la webcam, un faux-positif soit détecté en arrière-plan dans seulement une ou deux images successives; la webcam fait donc un brusque aller-retour. Si nous avions eu le temps, nous aurions pu chercher une manière de filtrer les visages cohérents pour éviter ces mouvements brusques.

Si notre production n'a pas d'applications réelles en elle-même, des technologies similaires sont employé à des fins militaires et commerciales. Avec les récentes avances en robotique, la vision par ordinateur possède un intérêt exponentiel dans de multiples domaines. Son avancement permet d'extraire une grande quantité d'informations dans les images et les vidéos, ce qui est essentiel à la fois pour l'amélioration des systèmes automatisés et pour l'optimisation des processus et des organisations.

Skybox Imaging est une compagnie américaine dont l'analyse d'image et la localisation d'objet sont au cœur de leur modèle économique. Ils envoient des petits satellites peu couteux en basse altitude avec des appareils photo de haute définition. L'imagerie détaillée de leurs satellites permet de suivre de près l'évolution des ressources à petite et grande échelle. Par exemple, ils peuvent analyser l'évolution de la déforestation, suivre les mouvements de refugiés dans une zone de conflit, ou même veiller sur la santé d'une surface agricole. Contrairement aux autres entreprises dans ce domaine, Skybox ne vend pas seulement les images capturées, mais l'analyse de ces derniers. Sans aucune intervention de l'homme, Skybox peut, par exemple, avertir son client s'il y a une fuite dans son pipeline dans une zone difficile d'accès, ou compter le nombre de conteneurs qui entre et qui sort d'un port n'importe où dans le monde.

Dans les décennies qui viennent, l'analyse d'image et la reconnaissance d'objet auront un impact important sur l'économie mondiale, notamment au niveau des flux de ressources comme nous l'avons vu précédemment.

BIBLIOGRAPHIE

CODE

http://docs.opencv.org
https://opencv-python-tutroals.readthedocs.org/en/latest/py_tutorials/py_tutorials.html
http://docs.opencv.org/doc/tutorials/tutorials.html

https://docs.python.org/2/

http://pyserial.sourceforge.net
http://pyserial.sourceforge.net/shortintro.html

THEORIE

http://makematics.com/research/viola-jones/
http://docs.opencv.org/modules/objdetect/doc/cascade_classification.html

http://fr.wikipedia.org/wiki/Méthode_de_Viola_et_Jones
http://en.wikipedia.org/wiki/Viola–Jones_object_detection_framework
http://www.wikiwand.com/fr/Caract%C3%A9ristiques_pseudo-Haar
http://en.wikipedia.org/wiki/AdaBoost?oldformat=true

https://www.cs.cmu.edu/~efros/courses/LBMV07/Papers/viola-cvpr-01.pdf
Viola, Paul, et Michael Jones. "Rapid object detection using a boosted cascade of simple features." Computer Vision and Pattern Recognition, 2001. CVPR 2001.

http://www.face-rec.org/algorithms/Boosting-Ensemble/decision-theoretic_generalization.pdf
Freund, Yoav, and Robert E Schapire. "A decision-theoretic generalization of on-line learning and an application to boosting." Journal of computer and system sciences 55.1 (1997): 119-139.

https://www.youtube.com/watch?v=WfdYYNamHZ8

CONCEPTION

http://torqpro.com/?product=mg995

AUTRES

http://www.skyboximaging.com/

ANNEXE

1 - CODE PYTHON

```
1   ####################
2   # Localisation d'objet par analyse d'image
3   # Ludovic Bouan et Thomas Lefort
4   # TIPE 2015 - ECAM Lyon
5   ####################
6
7
8   # Importer les modules et bibliothèques nécessaires
9   import sys
10  import serial
11  import cv2
12
13  # paramètres de connexion avec le port serial
14  ser = serial.Serial('COM3', 115200)
15
16
17
18  ########## FONCTIONS #########
19
20  def findfaces(grayframe):
21      '''
22      Description: Trouve les visages dans l'image
23      ----------------------------------------
24      Entrée: Image en nuances de gris
25      Sortie: Liste des visages (liste de tuples: (x, y, largeur, hauteur))
26      ----------------------------------------
27      '''
28      return faceCascade.detectMultiScale(
29                  grayframe,                      # Image à analyser
30                  scaleFactor=1.2,                # Facteur d'agrandissement
31                  minNeighbors=5,                 # Nombre minimale de voisins
32                  minSize=(60, 60),               # Taille minimum d'un visage
33                  flags=cv2.cv.CV_HAAR_SCALE_IMAGE # Fichier cascade
34              )
35
36  def findeyes (grayframe):
37      '''
38      Description: Trouve les yeux dans l'image
39      ----------------------------------------
40      Entrée: Image en nuance de gris
41      Sortie: Liste des yeux (liste de tuples: (x, y, largeur, hauteur))
42      ----------------------------------------
43      '''
44      return eyesCascade.detectMultiScale(
45                  grayframe,                      # Image à analyser
46                  scaleFactor=1.3,                # Facteur d'agrandissement
47                  minNeighbors=5,                 # Nombre minimale de voisins
48                  minSize=(20, 20),               # Taille minimum d'un visage
49                  flags=cv2.cv.CV_HAAR_SCALE_IMAGE # Fichier cascade
50              )
```

```
51
52  def disp(frame, faces, eyes):
53      '''
54      Description: Encadre les Eléments détectés
55      ----------------------------------------
56      Entrée: Image
57              Liste des visages (liste de tuples: (x, y, largeur, hauteur))
58              Liste des yeux (liste de tuples: (x, y, l, h))
59      Sortie: Image avec visages encadrés en vert
60              et les yeux encadrés en jaune
61      ----------------------------------------
62      '''
63      # Si il n'y a pas de visages
64      if len(faces) == 0:
65          # Encadrer en jaune tous les yeux détectés (le cadre fait 2 pixels
    d'Epaisseur)
66          for(ex,ey,ew,eh) in eyes:
67              cv2.rectangle(frame,(ex,ey),(ex+ew,ey+eh),(255,255,0),2)
68
69      #Si il y a des visages
70      else :
71          # Encadrer en vert tous les yeux détectés
72          for (x, y, w, h) in faces:
73              cv2.rectangle(frame, (x, y), (x+w, y+h), (0, 255, 0), 2)
74              roi_color = frame[y:y+h, x:x+w]
75
76          # Encadrer en jaune tous les yeux détectés
77          for(ex,ey,ew,eh) in eyes:
78              cv2.rectangle(roi_color,(ex,ey),(ex+ew,ey+eh),(255,255,0),2)
79
80  def center(width, height):
81      '''
82      Description: Trouve le centre de la vidéo
83      ----------------------------------------
84      Entrée: largeur (int); hauteur (int)
85      Sortie: tuple du center (x,y)
86      ----------------------------------------
87      '''
88      return (width/2, height/2)
89
90  def offset(centers, elements):
91      '''
92      Description: Trouve le décalage entre le centre et les éléments
93      ----------------------------------------
94      Entrée: Centres (tuple); Eléments (liste de tulpes)
95      Sortie: Liste de décalages (liste de tuples)
96      ----------------------------------------
97      '''
98      # Creer des listes vides
99      xoff = list()
100     yoff = list()
101
102     # Pour chaque élément, calculer la différence entre son centre et le
    centre de la vidéo en x puis en y
103     for (x, y, w, h) in elements:
104         xoff = xoff + [x+(w/2)-centers[0]]
```

```
105             yoff = yoff + [y+(h/2)-centers[1]]
106
107         # Renvoyer une liste de tuple dans lequel le ieme tuple contient le ieme
    élément de xoff et le ieme élément de yoff
108         return zip(xoff,yoff)
109
110     def mvt_filter(offset):
111         '''
112         Description: Renvoi le décalage seulement s'il est assez important
113         ----------------------------------------
114         Entrée: Décalage (liste de tuples)
115         Sortie: Décalage (liste de tuples) ou rien
116         ----------------------------------------
117         '''
118         # Créer une liste vide
119         L = list()
120
121         # Pour chaque Elément dans la liste offset
122         for i in range(len(offset)):
123             if len(offset[i]) > 0:
124                 # Si le décalage en x de cet élément est supérieur ŧ 10, rt1
    prend la valeur de ce décalage, sinon 0
125                 if abs(offset[i][0]) > 10: rt1 = offset[i][0]
126                 else: rt1 = 0
127                 # Si le décalage en y de cet élément est supérieur ŧ 10, rt2
    prend la valeur de ce décalage, sinon 0
128                 if abs(offset[i][1]) > 10: rt2 = offset[i][1]
129                 else: rt2 = 0
130
131                 # Si rt1 ou rt2 diffèrent de 0, ajouter ce couple ŧ la liste L
132                 if rt1 != 0 or rt2 != 0: L = L + [(rt1,rt2)]
133
134         if L != list(): return L
135
136     def send_arduino(instructions):
137         '''
138         Description: Etablie la connexion avec le port serial et envoi le string
    fourni
139         ----------------------------------------
140         Input: Instructions (string)
141         Output: Serial
142         ----------------------------------------
143         '''
144         ser.write(instructions+"\n")
145
146
147
148     ########## BOUCLE PRINCIPALE ##########
149
150     def main(a_intvl,width, height, display, angle1, angle2):
151
152         # Faire des cascades des variables globales
153         global faceCascade
154         global eyesCascade
155
156         # Importer les haarcascades
```

```
157     faceCascade=cv2.CascadeClassifier('haarcascade_frontalface_default.xml')
158     eyesCascade = cv2.CascadeClassifier('haarcascade_eye.xml')
159 
160     # Définir la constante de proportionnalité k
161     # Cette valeur est obtenue expérimentalement
162     k = 0.0432
163 
164     # Capturer la vidéo
165     vid = cv2.VideoCapture(0)
166 
167     # Définir les dimensions de la vidéo
168     vid.set(3,width)
169     vid.set(4,height)
170 
171     # Défini le compteur
172     analyse = 1
173 
174     # Boucle infini (interrompu par l'utilisateur par la touche 'Q')
175     while True:
176 
177         # Lire la vidéo image par image
178         ret, frame = vid.read()
179 
180         # Si l'image est ŧ analyser
181         if analyse == 1:
182 
183             # Faire une copie de l'image en nuances de gris
184             grayframe = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
185 
186             # Trouver les visages
187             faces = findfaces(grayframe)
188 
189             # Si il n'y a pas de visages, chercher les yeux dans toute
    l'image
190             if len(faces) == 0:
191                 eyes = findeyes(grayframe)
192 
193             # Si il y a des visages, chercher les yeux dans le cadre des
    visages
194             else:
195                 for (xf, yf, wf, hf) in faces:
196                     fsection_grayframe = grayframe[yf:yf+hf, xf:xf+wf]
197                     eyes = findeyes(fsection_grayframe)
198 
199             # Définir f comme la liste des décalages seulement si le
    décalage est assez important
200             f = mvt_filter(offset(center(width, height), faces))
201 
202             # Si f existe
203             if f:
204 
205                 ##### Méthode 2 #####
206                 #Selon le signe du décalage, diminuer ou augmenter l'angle
207 #                 if f[0][0] > 0: angle1 -= 10
208 #                 elif f[0][0] < 0: angle1 += 10
209 
```

```
210                 ##### Méthode 1 #####
211                 # Calculer de la moyenne des décalages horizontaux
212                 # Multiplier cette moyenne par k
213                 # Arrondir ce résultat et la soustraire à l'angle du moteur
214                 angle1 -= int(k*(sum([f[n][0] for n in
    range(len(f))])/len(f)))
215                 instructions = str(angle1)
216
217                 ##### Méthode 2 #####
218 #                if f[0][1] > 0: angle2 += 4
219 #                elif f[0][1] < 0: angle2 -= 4
220
221                 ##### Méthode 1 #####
222                 angle2+= int(k*(sum([f[n][1] for n in
    range(len(f))])/len(f)))
223                 instructions = instructions + ',' + str(angle2)
224
225                 send_arduino(instructions)
226                 print(instructions)
227
228             analyse += 1
229
230         # Augmenter / Remettre ŧ 0 la variable compteur
231         elif analyse <= a_intvl: analyse += 1;
232         else: analyse -= a_intvl;
233
234         # Encadrer les Eléments détectés
235         disp(frame, faces, eyes)
236
237         # Afficher l'image analysée
238         if display:
239             cv2.imshow('Video', frame)
240
241         # Quitter la boucle quand la touche 'Q' est enfoncée
242         if cv2.waitKey(1) & 0xFF == ord('q'):
243             break
244
245     # Quand tout est fini, fermer l'affichage et libérer la mémoire
246     ser.close()
247     video_capture.release()
248     cv2.destroyAllWindows()
249
250
251 ########## INITIALISATION ##########
252
253 if __name__ == "__main__":
254
255     # Prendre ou demander les variables d'initialisation
256     try: a_intvl = (int(sys.argv[1]))
257     except: a_intvl = int(raw_input("Analyse interval ? "))
258     try: display = bool(int(sys.argv[2]))
259     except: display = bool(int(raw_input("Display ? (Y=1 / N=0) ")))
260     # Prendre par défaut 480x640 comme dimension de la vidéo
261     try: width = (int(sys.argv[3]))
262     except: width = 640
263     try: height = (int(sys.argv[4]))
```

```
264         except: height = 480
265         angle1 = 90
266         angle2 = 115
267         ser.write(str(angle1)+','+str(angle2)+"\n")
268 
269 
270         main(a_intvl, width, height, display, angle1, angle2)
```

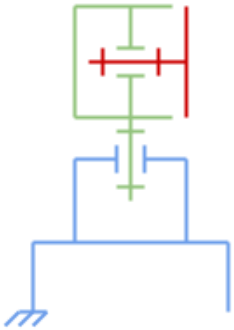
2 – VARIABLES

ID	*Nom*	*Type*	*Description*
1	a_intvl	int	Variable d'initialisation, défini le nombre d'images non analysé entre chaque image analysé
2	display	bool	Variable d'initialisation, Vrai ou Faux selon si l'utilisateur veut afficher la vidéo
3	width	int	Variable d'initialisation, défini la largeur de la vidéo
4	height	int	Variable d'initialisation, défini la hauteur de la vidéo
5	angle1	int	Représente l'angle du moteur d'axe de rotation vertical
6	angle2	int	Représente l'angle du moteur d'axe de rotation horizontale
7	vid	-	Représente la vidéo de la webcam
8	analyse	int	Compteur des images reçu
9	frame	-	Représente l'image interceptée
10	grayframe	-	Représente la version en nuance de gris de l'image interceptée
11	faces	list	Liste de tuples (x, y, largeur, hauteur) qui représente les visages
12	f	list	Identique à faces, mais n'existe que si le décalage est supérieur à la tolérance
13	instructions	str	Chaine de caractère contenant les angles angle1 et angle2, formaté pour être reçu par l'Arduino
14	ser	-	Contient les informations permettant la connexion au port Serial

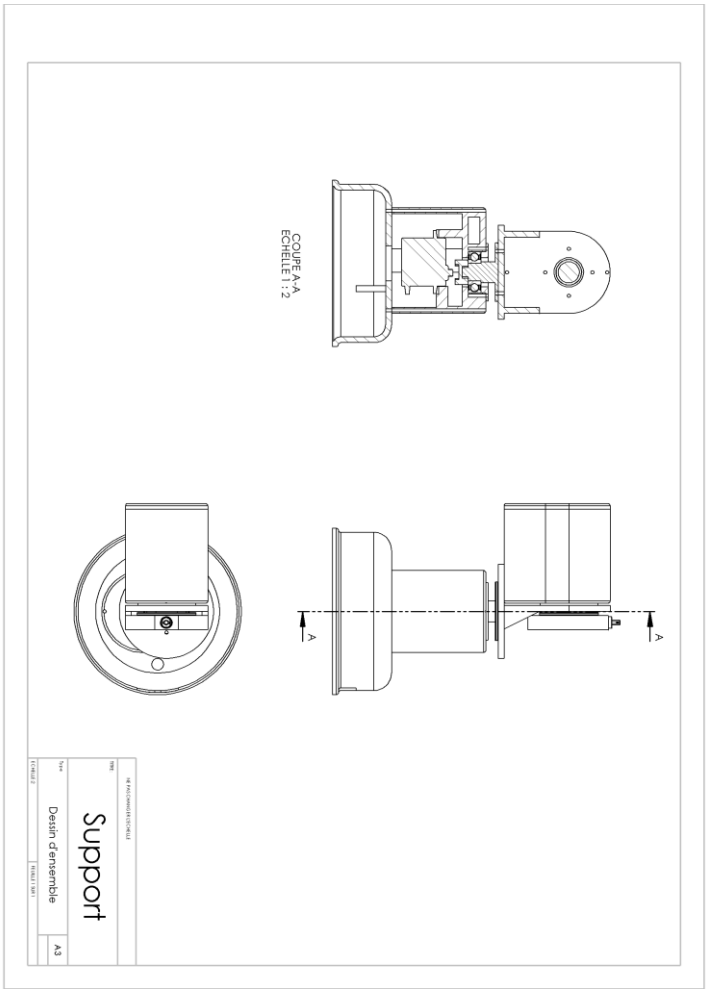
3 - CODE ARDUINO

```
#include <Servo.h>

int octetReception=0; // Variable qui enregistre les octets entrants
long nombreReception=0; // Chaine de caractère reçu

const int posMin=500; // Position minimal des servomoteurs
const int posMax=2300; // Position maximal des servomoteurs

const int brocheServo1=10; // Servomoteur 1 connecté sur la broche 10
const int brocheServo2=9; // Servomoteur 2 connecté sur la broche 9

Servo servo1; // Déclaration du servomoteur 1
Servo servo2; // Déclaration du servomoteur 2

int angleServo1=0; // Déclaration de l'angle du moteur 1
int angleServo2=0; // Déclaration de l'angle du moteur 2

void setup() { // Initialisation du programme
  Serial.begin(115200); // Initialisation de la connexion par Serial
  servo1.attach(brocheServo1, posMin, posMax); // Initialisation du servomoteur 1
  servo2.attach(brocheServo2, posMin, posMax); // Initialisation du servomoteur 2
}

void loop() { // Boucle principale
  while (Serial.available()>0) { // Boucle permettant de recevoir les données du Serial octet par octet
    octetReception=Serial.read(); // Lecture de l'octet suivant
    if (octetReception==10 || octetReception==44) { // Si l'octet suivant est un retour ≠ la ligne ou une virgule
      Serial.print ("Nombre recu = "); // Affichage du nombre reçu
      Serial.println (nombreReception);
      if (octetReception==44){ // Si une virgule est reçu
        angleServo1=nombreReception; // Le nombre reçu correspond au servomoteur 1
      }
      else{ // Sinon correspond au servomoteur 2
        angleServo2=nombreReception;
      }
      servo2.write(angleServo2); // Envoi des angles aux moteurs
      servo1.write(angleServo1);
      nombreReception=0; // Réinitialisation de la variable d'enregistrement
      delay(10);
      break;
    }
    else { // Sinon l'octet correspond ≠ un chiffre
      octetReception=octetReception-48; // Transformer pour obtenir le chiffre de 0 ≠ 9
      if ((octetReception>=0)&&(octetReception<=9)) // Vérification pour Éviter les erreurs
      nombreReception = (nombreReception*10)+octetReception; // Enregistrement de l'octet dans la variable nombreReception
      else Serial.println("La chaine n'est pas un nombre valide !"); // Afficher en cas d'erreur.
      delay(1);
    }
  }
}
```

4 - SCHEMA CINEMATIQUE



5 - DESSIN D'ENSEMBLE



Le moteur est représenté de façon schématique et n'est pas de notre conception.

6 – ADABOOST

1. On se donne des images exemple $(x_1, y_1), \dots, (x_n, y_n)$ où $y_i = 1\ ou\ 0$ pour des images positives et négatives respectivement.
2. On initialise les poids des images avec des coefficients $w_{1,i} = \frac{1}{2m}, \frac{1}{2l}$ pour $y_i = 0\ ou\ 1$ respectivement où m et l sont les nombres de négatives et positives respectivement.
3. Pour $t = 1, \dots, T$:
   a. Normaliser les poids $w_{t,i} \leftarrow \frac{w_{t,i}}{\sum_{j=1}^{n} w_{t,j}}$ pour que w_t soit une distribution probabiliste
   b. Pour chaque caractéristique pseudo-Haars, j, entrainer un classifieur faible h_j, composé d'une seule caractéristique pseudo-Haars. L'erreur est évaluée en prenant en compte $w_t, \epsilon_j = \sum_i w_i \left| h_j(x_i) - y_i \right|$.
   c. Choisir le classifieur, h_t, avec la plus faible erreur ϵ_t.
   d. Recalculer les poids des images :

$$w_{t+1,i} = w_{t,i} \beta_t^{1-e_i}$$

   Où $e_i = 0$ si exemple x_i est classifié correctement, $e_i = 1$sinon, et $\beta_t = \frac{\epsilon_t}{1-\epsilon_t}$
4. Le classifieur fort retenu est : $h(x) = \begin{cases} 1\ si\ \sum_{t=1}^{T} \alpha_t h_t(x) \geq \frac{1}{2} \sum_{t=1}^{T} \alpha_t \\ 0\ sinon \end{cases}$ où $\alpha_t = \log \frac{1}{\beta_t}$