S U P É L E C

Rapport de projet de première année

RECONNAISSANCE GESTUELLE

Utilisation de Python pour reconnaître et interpréter des mouvements de la main

CHARLES GABOULEAUD

AYMERIC HOUMEAU

VALÉRIAN JUSTINE

Avril - Juin 2013

Campus de Rennes • charles.gabouleaud@supelec.fr • aymeric.houmeau@supelec.fr • valerian.justine@supelec.fr

Table des matières

Introduction

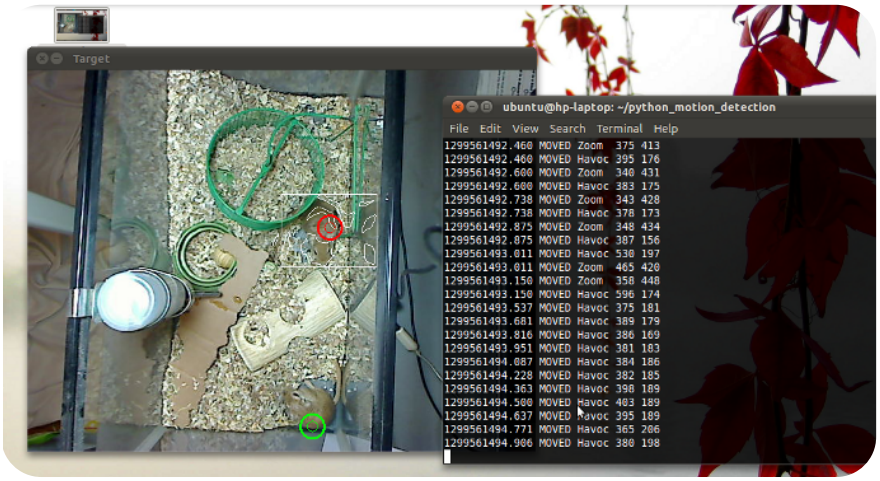
L'essor de la robotique offre de nombreuses possibilités, et en particulier celle de donner des ordres par un simple geste, que ce soit à un robot ou même à un ordinateur, comme de nombreux films de science-fiction tels que Minority Report le laissent imaginer. Nous avons décidé de nous intéresser à la reconnaissance de gestuelle, c'est-à-dire à la conception d'un programme capable d'analyser un geste pour en déduire un ordre ou une instruction. L'objectif de cette étude est de réaliser une application en Python qui pourra être réutilisée ensuite sur le chien Aibo ou le drone Parrot pour leur donner des ordres simples correspondant aux déplacements « élémentaires » : avancer, reculer, tourner, s'arrêter.

Pour ce faire, nous avons décidé d'utiliser un matériel facile à reconnaître pour améliorer les possibilités de détection. La forme retenue est celle d'une forme géométrique colorée sur un gant blanc. Ainsi, il suffira à la caméra de détecter ce « dispositif de commande » pour étudier ses mouvements et en déduire des instructions.

Notre démarche

À l'origine, nous souhaitions concevoir une application pour le chien Sony Aibo uniquement. Cependant, au vu de l'ancienneté du robot, et de l'absence de support concernant la plate-forme de développement, il ne nous a pas été possible de continuer dans cette voie. C'est pourquoi nous avons décidé de nous tourner vers une démarche plus générale et indépendante des machines qui pourraient l'utiliser. Le projet se concentre dès lors sur l'utilisation d'une webcam, et se veut adaptable à tout système qui possède ce type de matériel.

Nous avons effectué des recherches en ce sens, et avons découvert le travail de M. Derek Simkowiak, qui avait déjà réalisé une application de détection de mouvement pour observer l'activité de rongeurs. Nous avons alors repris et remanié le code Python que ce dernier avait laissé en libre accès.



Originellement, le programme servait à suivre les déplacements de gerbilles.

Détection du mouvement

Cette partie a pour but de mettre en avant le fonctionnement du programme ainsi que la manière dont il a été construit. En particulier, on commence par expliquer le fonctionnement du code de D. Simkowiak tel que nous l'avons adapté pour le faire fonctionner sur notre matériel, avant de nous intéresser aux fonctions que nous avons ajoutées afin de remplir notre objectif de transmission d'ordre par la gestuelle.

Repérage des différences

Pour détecter le mouvement, la démarche retenue est d'utiliser une caméra (dans notre cas une webcam), et de comparer deux images consécutives pour en déduire un « mouvement. » Les fonctions utilisées se trouvent dans la librairie OpenCV, qui possède une API complète pour Python.

On commence par créer une moyenne courante des images reçues, ici sur un intervalle de temps de l'ordre de 0,25 secondes. L'objectif est de flouter un peu l'image de manière temporelle pour éliminer le « bruit » de la caméra et ne garder que ce qui est significatif. Pour ce faire, on utilise la fonction RunningAvg de la librairie OpenCV. De plus, afin de limiter plus encore le bruit et les risques de faux positifs, on floue cette fois-ci spatialement à l'aide de la fonction Smooth, de la même librairie, qui effectue un flou gaussien de l'image.

La détection de mouvement se fait ensuite par différence entre la nouvelle image et la moyenne des images précédentes. Les pixels inchangés seront alors affichés en noir, et les pixels différents de la moyenne (donc caractérisant un mouvement), seront affichés en couleur.

A cette étape, nous avons donc la possibilité d'afficher une image dont les pixels sont de couleur égale à la différence entre la couleur de l'image étudiée et de la moyenne courante, ce qui rend compte des zones statiques et en mouvement dans la vidéo.

Délimitation des objets mouvants

On souhaite détecter le mouvement de façon plus exacte, et limitant les zones d'incertitude. Pour ce faire, on utilise un seuil de détection, qui va permettre d'obtenir une image en noir et blanc, les pixels blancs correspondant aux pixels « qui ont bougé. » On applique ensuite un flou gaussien (fonction Smooth) puis on seuille à nouveau l'image, ce qui diminue à nouveau le risque de faux positifs.

Analyse et repérage du centre de l'objet

C'est ici que notre programme diffère de celui d'origine : là où le programme de D. Simkoviak repère tous les mouvements et doit donc effectuer de nombreuses boucles pour déterminer le nombre de cibles à suivre et leur centre, nous nous limitons à un seul objet. Pour améliorer la précision, on décide donc de repérer un objet de couleur vive, qui se démarquera du fond.

Filtre par couleur

Deux choses étaient importantes pour le filtrage par couleur de notre objet : la *calibration*, et le *filtrage* en lui-même. Le filtre étant une simple itération sur tous les pixels d'une image, accompagné d'un seuil de couleur permettant de savoir si oui ou non on recopie le pixel en question dans notre nouvelle image, on va surtout décrire ici la calibration du système, et ainsi détecter un objet vert fluo.

Calibration

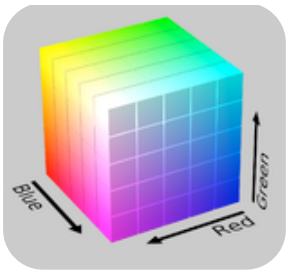
Selon la luminosité de la pièce, l'heure du jour, où même le déplacement des personnes présentes dans la pièce, les variations de couleurs d'un objet sont significative pour une analyse précise, malgré l'adaptation naturelle de l'oeil. On ne peut donc pas se permettre de rentrer directement dans le code la couleur à filtrer : il faut que l'utilisateur puisse la choisir. Pour cela, nous avons opté pour une méthode assez simple pour l'utilisateur et qui, grâce aux fonctions d'OpenCV, est assez simple à coder aussi : le clic. Ainsi, un clic de la souris sur l'image originale permet d'enregistrer la valeur de la couleur du pixel ciblé, et de l'enregistrer dans une variable utilisée plus loin pour le filtrage. Une petite astuce est cependant utilisée : afin que la couleur choisie soit simple à analyser, nous ne travaillons pas dans l'espace des couleurs utilisé par défaut par OpenCV (le RGB, Red-Green-Blue, très largement répandu), mais dans un autre espace : HSV.

HSV

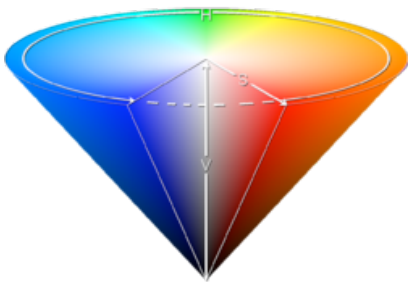
L'espace colorimétrique HSV (en français TSV pour Teinte, Saturation, Valeur) a l'énorme avantage de représenter la teinte d'une couleur en un seul entier. Ceci permet à un utilisateur de choisir en priorité la teinte, et la roue HSV est d'ailleurs souvent utilisé pour choisir une couleur dans les interfaces graphiques : la saturation et la valeur sont alors secondaires.

Dans notre cas, l'avantage vient du fait que le seuillage utilisé par le filtre de notre programme va pouvoir avoir une très faible tolérance pour la teinte (on veut une couleur très proche de celle sélectionnée), mais très grande pour la valeur et la saturation. Ainsi, du vert-presque-blanc et du vert-presque-noir seront toujours acceptés par un filtre acceptant le vert. On élimine ainsi des problèmes venus des ombres et des sur-éclairages sur l'objet !

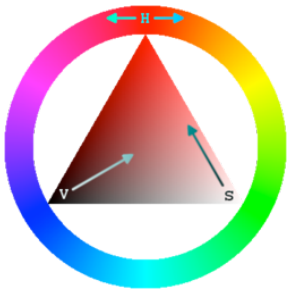
Heureusement pour nous, la conversion des couleurs est gérée automatiquement par OpenCV, car les transformations d'un espace à l'autre ne sont pas toujours simple.



Le cube RGB, pour comparaison



L'espace 3D HSV, qui permet de représenter toutes les couleurs.



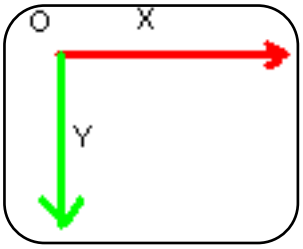
La roue de sélection des couleurs en HSV, souvent utilisée dans les interfaces graphiques.

Algorithmes utilisés par D. Simkowiak

Une fois les objets en mouvements repérés, la méthode générale programmée par Simkowiak consiste à détecter les contours des objets (OpenCV a une fonction intégrée permettant de les calculer directement), suivi de la méthode dite « AABB ». qui est issu de la théorie des collisions, et enfin l'algorithme des k-moyennes (*k-means algorithm*), afin de repérer les centres des objets en mouvements. Ce n'est que ce dernier que nous utilisons effectivement dans notre programme : n'ayant qu'un seul objet à traquer, la méthode AABB nous est inutile.

Méthode AABB et Théorie des collisions

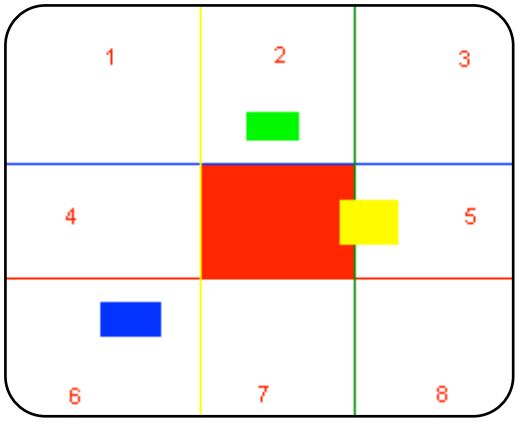
La théorie des collisions est très utilisée, en particulier dans le monde du jeu vidéo, afin de déterminer si un objet ou un personnage en touche un autre. La méthode que nous allons étudier consiste en une découpe de l'espace par des rectangles particuliers, les **AABB** *(Axis Aligned Bounding Box)*. Il s'agit d'un rectangle aligné avec les axes, c'est-à-dire que ses cotés sont parallèles aux axes de l'écran pour les cas standards.



Une AABB est définie par 4 paramètres : la position (x, y) de son coin supérieur gauche (en 2D, l'axe Y va vers le bas). Ainsi que de sa largeur w (comme *width*) et sa hauteur h (comme *height*).

Une fois ces rectangles déterminés, les conditions de collision se déterminent simplement. Soient deux rectangles de type AABB définis par $X=(x_0, y_0, w_0, h_0)$ et $Y=(x_1, y_1, w_1, h_1)$, la condition de collision s'écrit plus simplement sous la forme de la négation de la condition de non collision :

$$\mathbf{C} = \textit{not}\ (\ (x_1 >= x_0 + w_0)\ \textit{or}\ (x_1 + w_1 <= x_0)\ \textit{or}\ (\ y_1 >= y_0 + h_0)\ \textit{or}\ (y_1 + h_1 <= y_0)\ )$$



Dans le cadre du programme tel que nous l'avions récupéré, cette méthode permettait de détecter le mouvement en fusionnant les AABBs adjacents, afin de constituer une unique entité mouvante. Dans le cadre de notre travail, l'utilisation de cet algorithme était inutile, puisqu'il permettait de déterminer le nombre de cibles potentielles à suivre, alors que nous savons que nous ne nous intéressons qu'à une unique cible.

Algorithme des k-moyennes (k-means)

Cet algorithme a été employé dans le programme tel que D. Simkowiak l'avait écrit afin de compléter la méthode AABB, qui traitait assez mal le mouvement des appendices des animaux observés, en particulier les mouvements de leurs queues.

En pratique, l'algorithme calcule la position de barycentres d'ensembles de points. Dans le cadre de notre programme, on ne s'intéresse qu'à un unique centre de commande, on se contente donc d'éliminer les pixels noirs de l'image filtrée et de faire fonctionner l'algorithme en demandant un unique point moyennant tous les autres. Ce point est le barycentre de notre zone de mouvement, et permettra de caractériser le mouvement de la zone de commande.

Limité à la détection d'une seule cible, l'utilisation de cet « algorithme » pour déterminer le barycentre noua a paru quelque peu abusive : en effet, il est très puissant et demande beaucoup de ressource. Nous avons donc tenté de la remplacer par une fonction de détection du barycentre des points non noirs plus simple afin d'augmenter le nombre d'images traitées et de diminuer la charge de travail pour le processeur. L'expérience nous a montré que la fonction de barycentre était trop sensible aux artefacts lorsque l'on travaillait en lumière naturelle, et donnait des faux positifs, ce qui rend pratiquement impossible l'analyse du mouvement telle que le faisons. Nous avons donc remis en place la détection du barycentre par la méthode k-means, qui nous a paru être un algorithme plus solide et moins sensible aux erreurs de détection.

Analyse du mouvement

Par manque de temps, nous avons proposé une définition très simplifiée de l'ordre comme étant le vecteur défini par un point de départ (la zone de commande est à peu près fixe) et un point d'arrivé (la zone de commande est en déplacement depuis plusieurs images).

On observe l'image et on interprète un ordre si et seulement si la vitesse est supérieure à 15 pixels par image pendant au moins 4 images. On fait la différence entre les positions du point de départ et du point courant pour déterminer l'ordre actuel. Cet ordre est obtenu en fonction de la composante principale du mouvement, déterminée par différence entre les composantes selon x et y du vecteur mouvement, avec un seuil qui permet d'éviter les confusions lors d'un ordre en diagonale par exemple : il est simplement ignoré. Le programme tel qu'il est conçu actuellement comprendrait donc un long mouvement vers la droite comme une suite de plusieurs ordres « aller à droite », et un mouvement circulaire dans le sens horaire comme une suite d'ordres « aller à droite » puis « aller en bas », puis « aller à gauche ».

Synthèse

Notre projet permet donc de détecter un mouvement simple et d'en déduire un ordre. Nous avons testé et remis en question le code de D. Simkowiak afin de l'adapter à nos besoins, en le simplifiant grandement et en apportant de nouvelles fonctions.

De nombreuses limites apparaissent évidemment dans ce projet. D'abord, l'analyse du mouvement est très approximative : il s'agit là d'une méthode mise en place rapidement, pour voir si notre détection des mouvements était fonctionnelle. En pratique, le programme fait de nombreuses erreurs, en particulier lors des ordres verticaux, probablement à cause de l'amplitude de mouvement inférieure imposée par le format de

capture des vidéos. C'est là que les améliorations pourront être les plus nombreuses : on pourrait imaginer par exemple un algorithme dessinant la trajectoire de la cible et qui essayerait d'y reconnaître des formes. On pourrait alors diversifier la nature de ceux-ci : droites, cercles, angles... Tout est imaginable. De plus, une méthode de détermination du barycentre moins exigeante en temps de calcul pourrait améliorer le nombre d'images par seconde analysées et probablement améliorer l'efficacité du programme : plus les points seront nombreux, plus le détail du trajet pourra être fin.

D'autre part, des limites inhérentes au système sont difficiles à réduire : nous utilisons aujourd'hui une webcam bas de gamme, avec une résolution faible, qui a une balance automatique des blancs qui cause certains problèmes au niveau du filtrage des couleurs. Ceci se rapproche quelque part de la caméra implantée sur l'Aibo, notre système de départ, qui ne pourra de toute façon pas gérer une résolution supérieure : la puissance de calcule nécessaire actuellement est trop grande dès qu'on prend des images en 640x480.

En pratique, notre projet pourrait éventuellement permettre de donner des ordres à n'importe quel système ayant une caméra intégrée. En effet, il suffirait de remplacer les lignes de codes déterminant le texte à afficher à l'écran (gauche, bas, droite, haut...) par des ordres en utilisant l'API fourni par ce système ! Bien sûr, de nombreuses optimisations et améliorations restent à faire, mais d'un point de vue ludique, l'utilisation d'OpenCV et de Python rend le traitement d'image assez agréable, d'autant que la première apparition d'une cible qui suit un objet, ou même du filtrage de couleur fait énormément plaisir et motive à aller plus loin. Nous avons même eu l'occasion de « convertir » certaines personnes, qui disent vouloir s'amuser avec ces technologies pendant les vacances !

Bibliographie

Derek Simkowiak, pour son code source et des explications sur le motion tracking :

- http://derek.simkowiak.net/motion-tracking-with-python/

Le Site du Zero, pour des explications sur les algorithmes utilisés :

- http://www.siteduzero.com/informatique/tutoriels/theorie-des-collisions

Wikipédia, pour de nombreux articles, pour les algorithmes et autre :

- https://en.wikipedia.org/wiki/K-means_clustering
- http://fr.wikipedia.org/wiki/Teinte_Saturation_Valeur

L'API python d'OpenCV :

- http://opencv.willowgarage.com/documentation/python/

Geckogeek, pour le filtrage des couleurs et surtout la sélection de couleur en cliquant :

- http://www.geckogeek.fr/tutorial-opencv-isoler-et-traquer-une-couleur.html