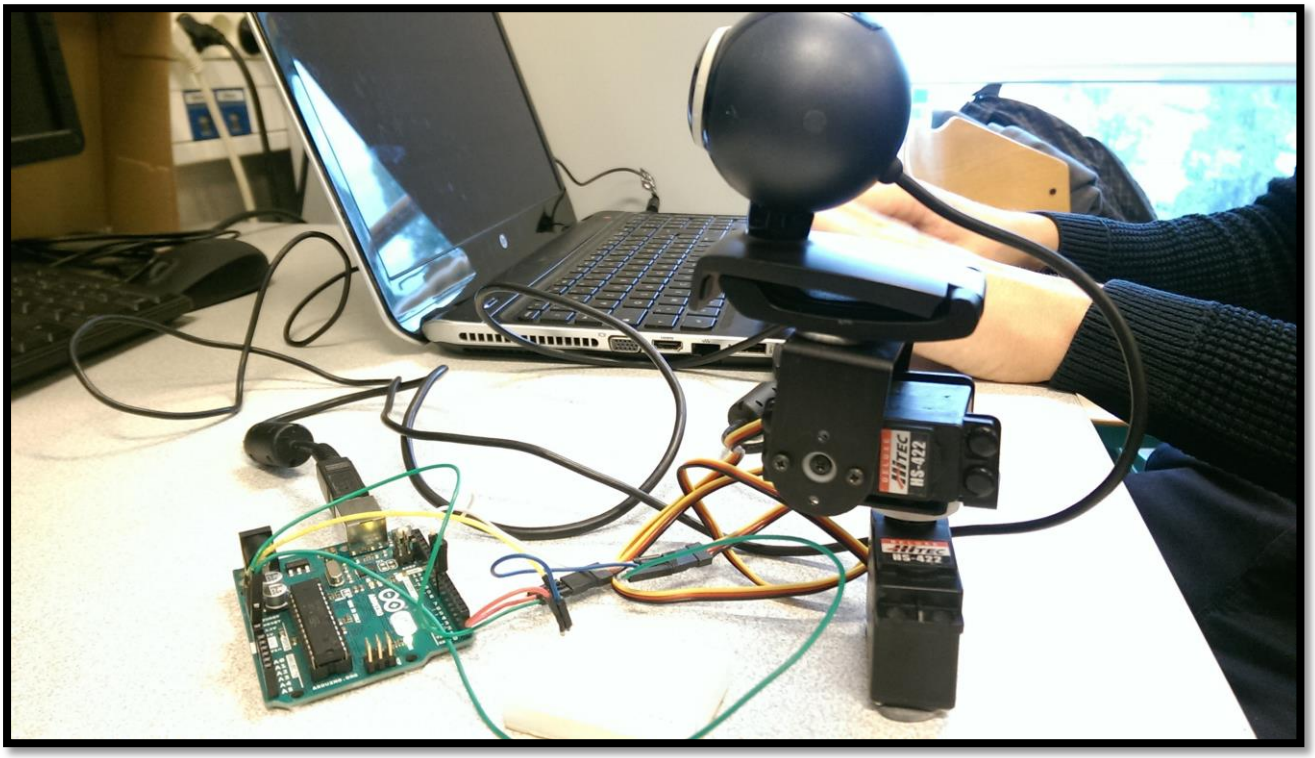
RAPPORT D'ACTIVITÉ

PROJET DE ROBOTIQUE



BACHELOT Loïc et RÉMOND Amélie
CMI SIC L3

Introduction

Nous avons pour projet de réaliser le tracking d'un objet en mouvement. Il s'agit de fixer une caméra sur des servomoteurs et programmer ces derniers pour suivre un objet de couleur spécifiée.

Ce projet est constitué de 4 grandes parties : le montage physique, le traitement d'images, la retranscription des informations aux moteurs et la certification.

Montage physique

Les composants :

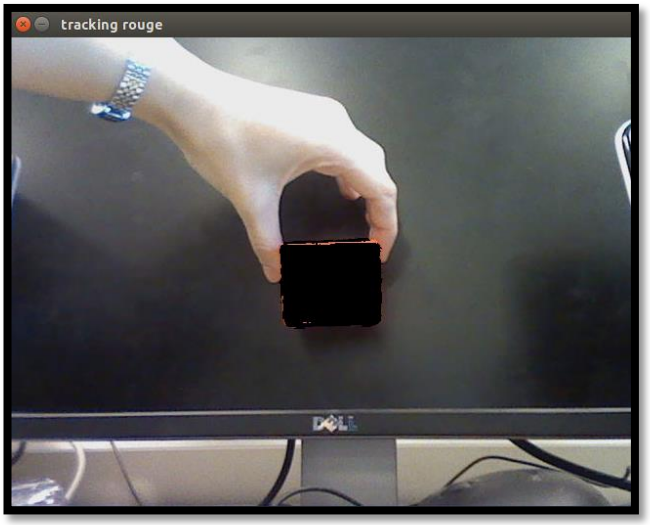
Pour ce projet, nous avons utilisé le matériel suivant :
- Carte arduino
- Breadboard
- Deux servomoteurs
- Une caméra (type webcam)
- Câbles et Fils de liaison
- Un ordinateur portable

Et nous avons formé le montage disponible en annexe (cf. Annexe A. Schéma du montage physique).

Nous avons ensuite effectué toutes les configurations nécessaires (port série, servomoteurs), installé Opencv et utilisé un programme simple d'affichage pour tester si tout fonctionnait.

Traitement d'images

Dès que la configuration a permis l'affichage d'une image claire, nous avons créé une fonction permettant de "scanner" chaque image et de reconnaître une couleur (dans notre cas le rouge). Nous avons recoloré en noir chaque pixel rouge détecté comme on peut le voir ci-dessous afin de vérifier le bon fonctionnement du programme.



Capture d'écran du résultat du traitement d'images

Sur l'image ci-dessus, on peut voir un cube de couleur rouge recoloré en noir par le programme.

La fonction correspondant à ce traitement d'images est disponible en annexe (Cf. 2.Traitement d'images).
L'algorithme:
- Parcours l'image pixels par pixels
- Pour chaque pixel, il regarde si la proportion de rouge est supérieure à un certain seuil.
- Si c'est le cas, on ajoute le pixel au calcul du barycentre.

Nous avons rajouté un calcul de la position moyenne de l'objet.

Retranscription des informations aux moteurs

Après avoir récupéré le barycentre suite au traitement d'images, il faut traiter l'information pour envoyer une valeur d'angles aux servomoteurs.

On sépare les calculs selon deux axes (horizontal, vertical), les calculs sont identiques. Le programme détermine la position de l'objet par rapport au centre de l'image en calculant l'erreur (distance en pixel entre le centre de l'image et le barycentre de l'objet). Puis selon la position par rapport au 0 sur l'axe (en haut, en bas ou à droite, à gauche) avec une marge d'erreur de 10 pixels permettant de stabiliser une position convenable. L'angle est alors calculé de la façon suivante:

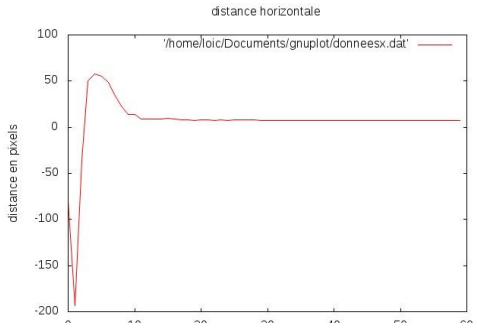
Nouvelle valeur de l'angle = (valeur précédente de l'angle) +/- PID;
D = la distance entre l'objet et le centre de l'image;
PID = (D/constante) + ((valeur précédente de D + nouvelle valeur de D)/constante) + ((nouvelle valeur de D - valeur précédente de D)/constante);

Le programme vérifie que les valeurs limites du servomoteur ne sont pas atteintes et garde ou non la nouvelle valeur de l'angle.

Ainsi le programme arduino (disponible en annexe : Cf.3.1 Retranscription) permet de faire déplacer les moteurs grâce aux valeurs envoyées dans le programme c (disponible en annexe: Cf.3.2 Retranscription).

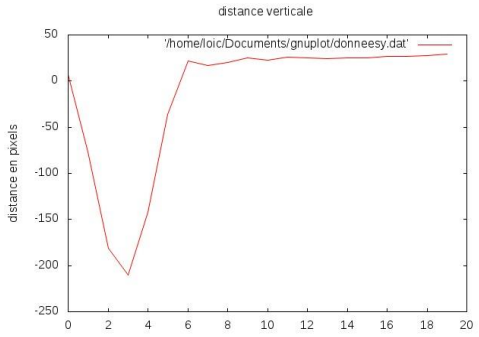
Certification

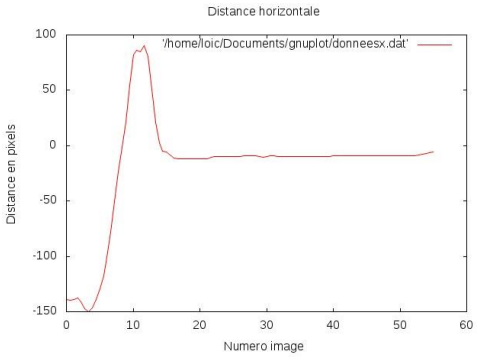
Notre dernière version permet d'obtenir ces 4 graphiques représentant l'évolution de l'erreur au cours du temps selon les axes X, Y.



Courbe de l'évolution de l'erreur en X en fonction du temps

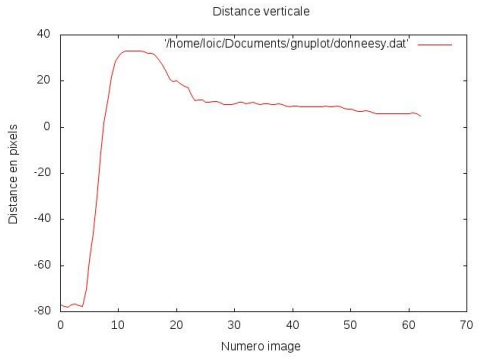
Courbe de l'évolution de l'erreur en Y en fonction du temps





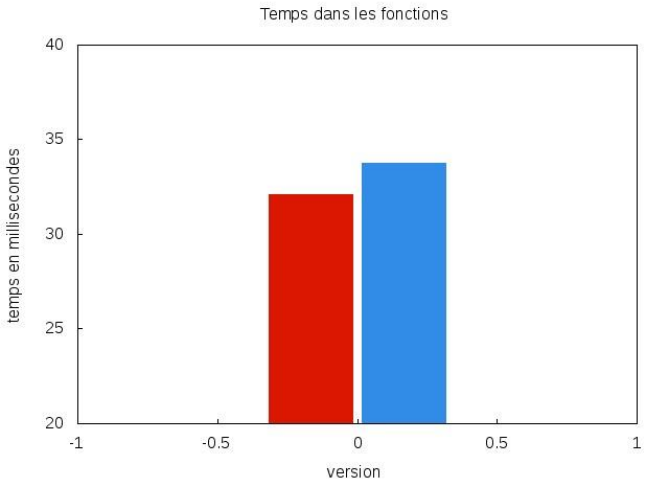
Courbe de l'évolution du PID en X en fonction du temps

Courbe de l'évolution du PID en Y en fonction du temps

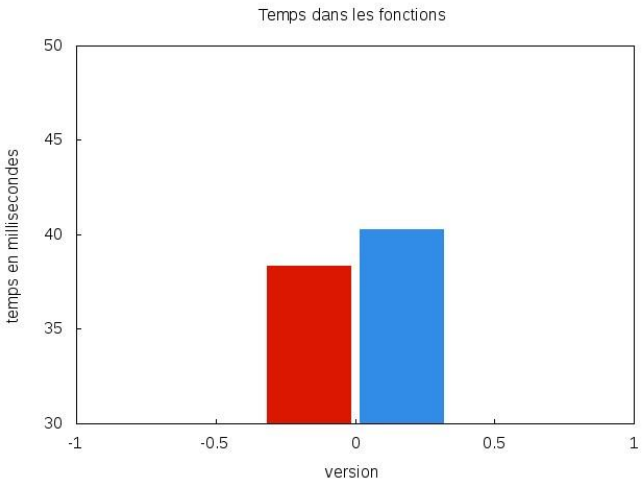


On peut voir que la caméra essaie de se stabiliser et que l'écart se réduit de plus en plus pour finir nul (ou presque nul). En utilisant seulement un coefficient proportionnel à l'erreur, on obtient une courbe plus lisse, cependant en utilisant la formule complète du PID l'exécution est beaucoup plus rapide.

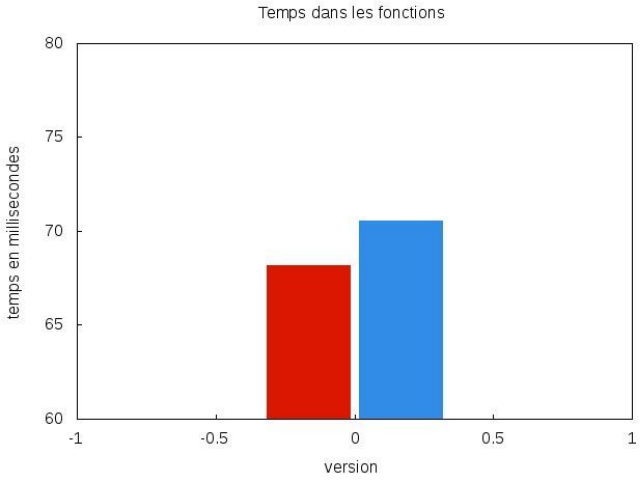
Nous avons isolé le temps du traitement d'images par rapport au reste de l'exécution et nous avons obtenu les graphiques suivants :



Graphique du temps utilisé pour le traitement d'une image de résolution 160 px



Graphique du temps utilisé pour le traitement d'une image de résolution 320 px



Graphique du temps utilisé pour le traitement d'une image de résolution 640 px

Sur le graphique de gauche (pour une résolution de **160px**) le temps en millisecondes du traitement d'image est de **32.1ms** et le temps global est de **33.77ms.**
Sur le graphique de droite (pour une résolution de **320px**) le temps en millisecondes du traitement d'image est de **38.3ms** et le temps global est de **40.2ms**.
Sur le graphique du bas (pour une résolution de **640px**) le temps en millisecondes du traitement d'image est de **68.1ms** et le temps global est de **70.5ms.**

On observe que le traitement d'images prend plus de 90% du temps d'exécution. On remarque que : plus la résolution est petite, plus le traitement est rapide. Ainsi, plus la résolution est petite, et plus l'exécution est rapide.

L'objectif étant d'avoir une exécution rapide, il faut réduire au maximum cette partie.

Annexe

1. Schema du montage physique

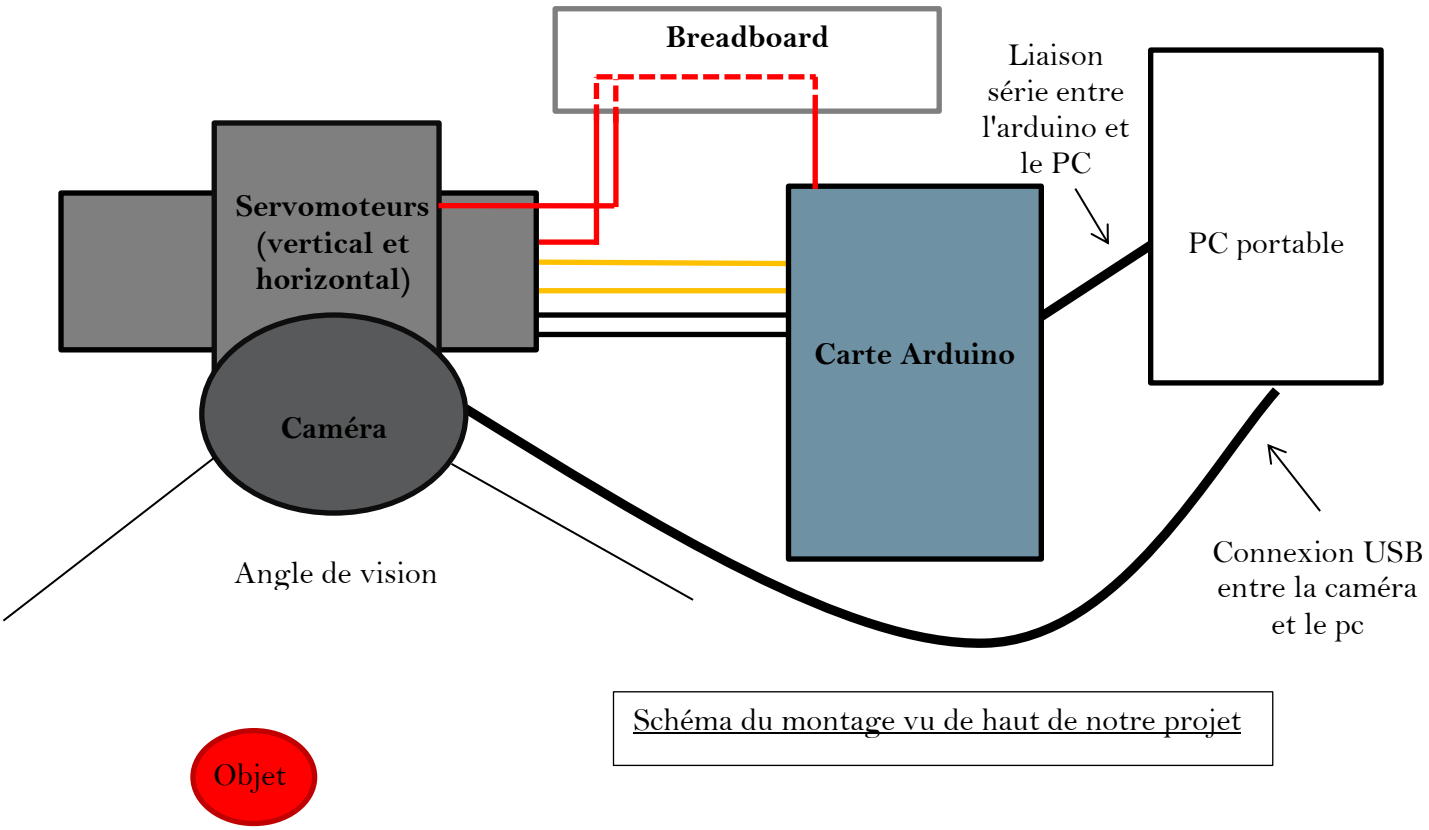


Schéma du montage vu de haut de notre projet

2. Traitements d'images

```
void detectionRouge(IplImage *img, int *avx, int *avy) {
    CvScalar p;
    int x, y, k, compteur = 1;
    for (y = 0; y < img->height; y++) {
        for (x = 0; x < img->width; x++) {
            // récupération d'un pointeur sur le pixel de coordonnées (x,y)
            p = cvGet2D(img, y, x);
            if ((2 * p.val[0]) < (p.val[2]) && (2 * p.val[1]) < (p.val[2])) {
                for (k = 0; k < img->nChannels; ++k) {
                    p.val[k] = 0;
                }
                *avx = *avx + x;
                *avy = *avy + y;
                compteur++;
            }
            cvSet2D(img, y, x, p);

        }
    }
    //recupération de la position moyenne de rouge
    *avx = *avx / compteur;
    *avy = *avy / compteur;
}
```

3.1 Retranscription

```
#include<Servo.h>

int monservox = 7;
int monservoy = 8;
Servo servox;
Servo servoy;
int angle;

void setup() {
  Serial.begin(9600);
  servox.attach(monservox);
  servoy.attach(monservoy);
}

void loop() {
     if(Serial.available()>0){
       angle = Serial.read();
       if(angle <= 127){
         servox.write(angle);
       }else{
         angle = angle - 127;
         servoy.write(angle);
       }
     }
}
```

3.2 Retranscription

```
distancex = avx - (image->width / 2);
distancey = avy - (image->height / 2);

pidx = (distancex / 30) + ((distanceprecx + distancex) / (2 * 40)) + ((distancex - distanceprecx) / 30);
pidy = (distancey / 30) + ((distanceprecy + distancey) / (2 * 40)) + ((distancey - distanceprecy) / 30);

//en x
if (distancex < -10) { //à changer pour qu'il s'arrete
    if (anglex + pidx > 0) {
        anglex = anglex + pidx;
    }
} else if (distancex > 10) {
    if (anglex + pidx < 127) {
        anglex = anglex + pidx;
    }
}
send = (unsigned char) anglex;
//printf("send %d\n", send);
serial_write(&sp, &send);
fprintf(filex, "%d\n", distancex);


// en y
if (distancey < -10) { //à changer pour qu'il s'arrete
    if ((angley - pidy) < 254) {//formule pid
        angley = angley - pidy;
    }
} else if (distancey > 10) {
    if ((angley - pidy) > 128) {
        angley = angley - pidy;
    }
}
send = (unsigned char) angley;
serial_write(&sp, &send);
//printf("valeur envoyée %d\n", send);
fprintf(filey, "%d\n", distancey);

distanceprecx = distancex;
distanceprecy = distancey;
```