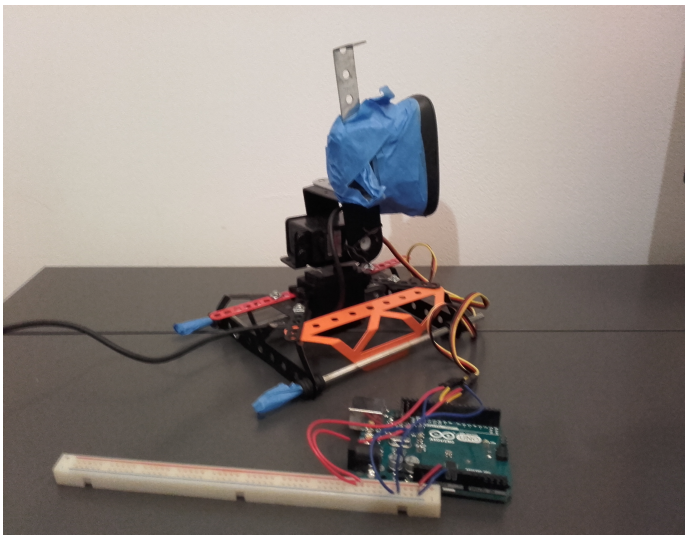
Université de Cergy-Pontoise
Licence d'Informatique troisième année

Dorian CHENET - Aurélien LEROY

Conception d'un montage pan-tilt et suivit de couleur par vidéo

RAPPORT



Janvier 2020

Table des matières

1 Conception physique **2**

2 Interractions au sein du système **2**
2.1 Chaîne d'actions . . . 2
2.2 Protocole de communication ordinateur -arduino . . . 3

3 Traitement de l'image **4**
3.1 Détection des couleurs . . . 4
3.2 Génération des données de traitement . . . 5
3.3 Calcul de la position de l'objet sur l'image et calcul d'erreur . . . 5

4 Tests, Résultats et Performance **6**
4.1 Performances de la détection de couleurs . . . 6
4.2 Performances de tracking . . . 6
4.2.1 Tracking Horizontal . . . 6
4.2.2 Tracking Diagonal . . . 7

Table des figures

1 Schéma cinématique du montage . . . 2
2 Notre robot . . . 2
3 Chaîne d'actions du robot . . . 2
4 Repésentation des zones de vitesse sur une image de la caméra . . . 3
5 Illustration de la détection de couleur par seuils . . . 4
6 Illustration de la construction des données d'analyse à partir d'une image . . . 5
7 Résultats de la détection des couleurs primaires RVB . . . 6
8 Photographie de l'expérience . . . 6
9 Schéma de l'expérience . . . 6
10 Graphiques d'erreur de l'expérience de tracking horizontal . . . 7
11 Photographie de l'expérience . . . 7
12 Schéma de l'expérience . . . 7
13 Erreur de placement de la caméra . . . 8
14 Erreur de placement par axe . . . 8

Nous avons réalisé la conception la fabrication et la programmation d'un montage pan-tilt permettant le suivi caméra d'une couleur donnée par le biais d'un algorithme d'analyse vidéo.

1 Conception physique

Notre robot mouvoit grâce à un montage pan-tilt permettant à la caméra montée par dessus de pouvoir voir à 180°horizontalement et verticalement comme illustré sur la Figure 1. En réalité, nous avons moins de 180°d'angle en vertical car la caméra est trop volumineuse pour cela. Nous avons, monté nos moteurs sur un câdre bas, large et lourd pour éviter au maximum le balancement dûs aux mouvements des moteurs. Cependant une chose que nous n'avions pas prévu était que en montant la caméra verticalement, son poids exerce un bras de levier non-négligeable sur le moteur. Ceci fait que le moteur de tilt a parfois un peu de mal à se stabiliser quand il est dans une position défavorable et que l'angle qu'il doit décrire est petit. Cela reste néanmoins un invénient mineur.

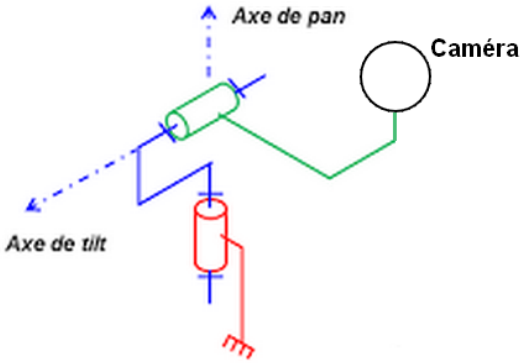


FIGURE 1 – Schéma cinématique du montage

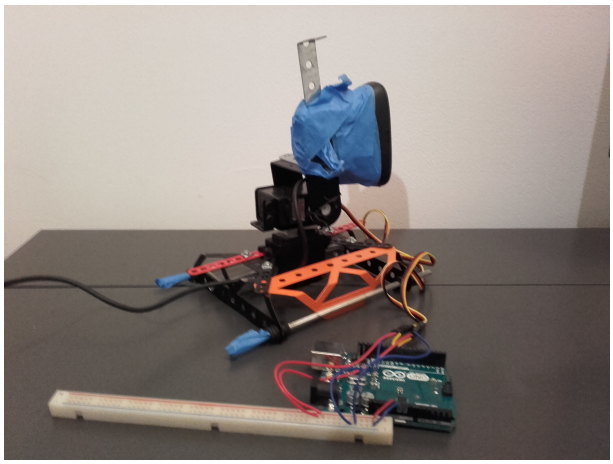


FIGURE 2 – Notre robot

2 Interractions au sein du système

2.1 Chaîne d'actions

Le montage du robot comporte 4 systèmes principaux : l'arduino, le montage pan-tilt (servo-moteurs), la caméra et un ordinateur. Les interractions entre ces derniers sont les suivantes :

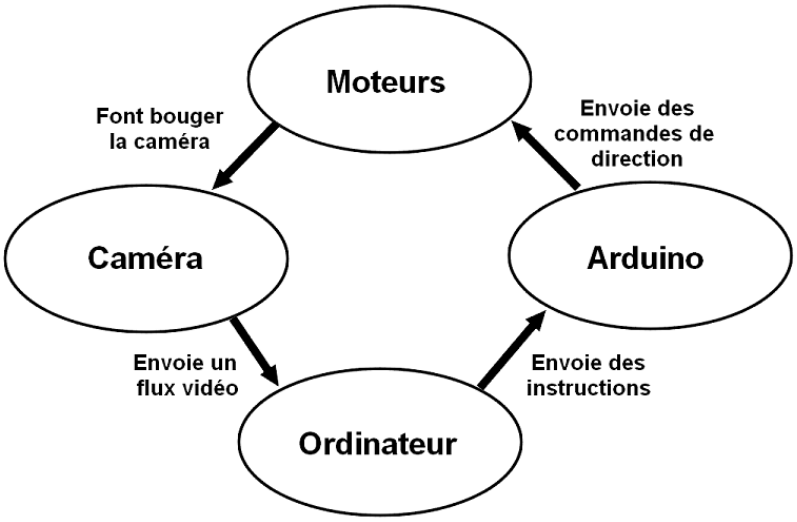


FIGURE 3 – Chaîne d'actions du robot

— Caméra : Capte une image qu'elle envoie à l'ordinateur
— Ordinateur : Traite l'image et envoie des instructions à l'arduino
— Arduino : Traite les instuctions et commande les moteurs
— Moteurs : Font bouger (ou non) la caméra

C'est grâce à ce cycle d'actions que nous pouvons réaliser un suivi caméra d'un objet se déplaçant dans l'espace ; il ne s'arrête que sur commande de l'utilisateur. Si jamais l'objet sort du champ de vision de la caméra, les mouvements s'arrêtent jusqu'à ce que l'objet revienne (à moins que la caméra capte du broit auquel cas elle continuera à bouger).

2.2 Protocole de communication ordinateur -arduino

Pour transmettre les commandes (résultantes du traîtement des images de la caméra) jusqu'aux moteurs, l'ordinateur communique avec une carte arduino comme intermédiare. Le protocole de connection utilise une liaison série paramétrée en 9600 8N1. Les commandes envoyées par l'ordinateur à la carte arduino sont simplement des caractères :
— 'z' pour faire monter la caméra
— 's' la faire descendre
— 'q' pour la tourner vers la gauche
— 'd' pour la tourner vers la droite

La valeur de déplacement de base d'un moteur est 1°pour chaque caractère reçu. Cela nous permet de nous stabiliser façilement sur la cible et c'est assez pour pouvoir réaliser un suivi fluide car un caractère ASCII vaux 1 octet et nous utilisons une liaison 9600 bauds donc nous pouvons transmettre (en théorie) 1200 commandes par seconde. Par conséquent on peut aussi dire que nous pouvons transmettre 600 commandes par secondes pour la direction verticale et 600 commandes par secondes pour la commande horizontale. Chaque image caméra analysée par l'ordinateur résulte en une instruction de mouvement vertical et une instruction de mouvement horizontal (soit 2 caractères en tout).

Communiquer des vitesses Afin de suivre efficacement un objet rapide ainsi que d'accélérer le positionnement initial de la caméra quand l'objet entre dans le champ de vision de la caméra, nous utilisons le principe illustré en Figure 4.

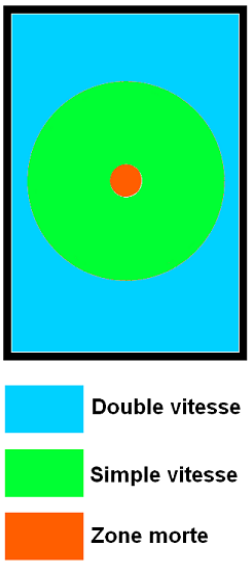


FIGURE 4 – Repésentation des zones de vitesse sur une image de la caméra

Zone Morte Au centre de l'image caméra, on trouve la zone morte permettant la stabilisation de la caméra sur un point fixe.

Simple Vitesse Quand la cible se trouve dans la zone de simple vitesse, on utilise le protocole de communication décrit précédemment. Chaque image caméra analysée par l'ordinateur résulte en une instruction de mouvement vertical et une instruction de mouvement horizontal.

Double Vitesse Si la cible se trouve à une positions éloignée du centre de l'image, on utilise la double vitesse, c'est à dire qu'à chaque image analysée nous envoyons deux fois la commande de mouvement. C'est à dire que plutôt que d'envoyer seulement un caractère pour la verticale puis l'horizontal, on envoie deux fois le caractère (soit 4 caractères en tout, 2x vertical, 2x horizontal). De ce fait, l'arduino va pouvoir se centrer plus vite de lui-même sur la cible éloignée sans attendre la suite des instructions de l'ordinateur. Comme la cible est éloignée du centre de l'image, il est fort probable que l'ordinateur arrive plusieurs fois au même résultat en traïtant le flux caméra. En envoyant deux fois l'instruction nous anticipons donc la prochaine instruction et nous ciblons plus vite l'objet. Nous gagnons ainsi le temps de traitement d'une image ce qui a un impact sur la performance du suivi comme nous le verrons par la suite.

3 Traitement de l'image

Le traitement du flux vidéo est la partie du système qui nous a prit le plus de temps à développer. Comme nous l'avons vu avec la chaîne d'actions, c'est grâce à ce processus que nous déterminons les mouvements que doivent décrire les moteurs.

3.1 Détection des couleurs

Notre programme de traîtement est capable de reconnaître 3 couleurs : rouge, vert et bleu. Pour se faire, nous utilisons un système de seuils comme illustré sur la Figure 5 :

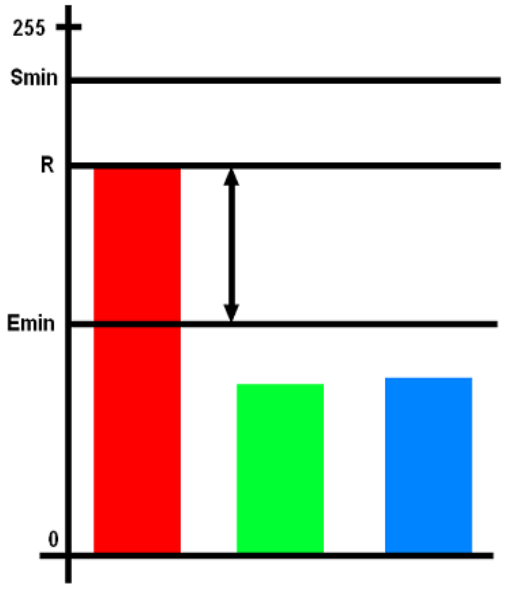


FIGURE 5 – Illustration de la détection de couleur par seuils

Pour qu'une couleur soit reconnue comme rouge, vert ou bleu il faut que les valeurs rgb respectent les conditions suivantes (ici pour détecter du rouge) :

- Le rouge doit être supérieur au seuil Smin pour éviter de capter des couleurs trop sombres et faire des erreurs
- Les autre couleurs doivent être à un écart Emin en dessous de la valeur R du rouge pour être sûr de ne pas avoir un mélange de couleurs difficilement reconnaissable et limiter le bruit.

3.2 Génération des données de traitement

La détection des couleurs est seulement un élément de la chaîne de traitement du flux vidéo. En effet les fonctions de détection ne peuvent nous donner qu'une information sur un seul pixel à la fois. Nous devons donc traiter chaque pixel de chaque image et ensuite déterminer la position de l'objet sur l'image. Pour se faire, nous utilisons l'algorithme illustré en Figure 7 :

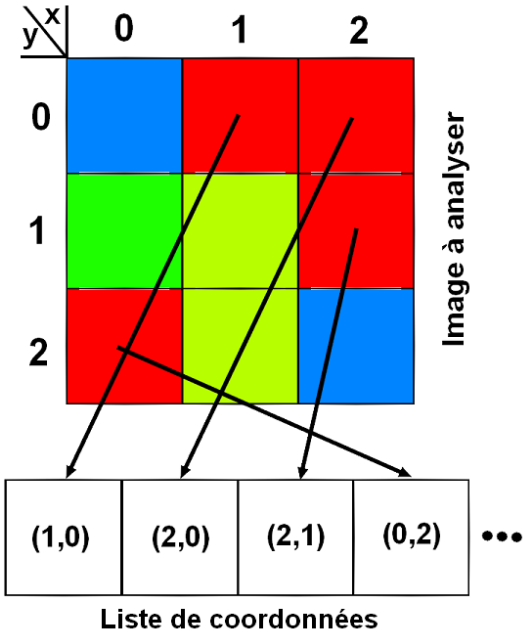


FIGURE 6 – Illustration de la construction des données d'analyse à partir d'une image

Nous parcourons l'image sur les x puis les y et testons si chaque pixel est de la couleur désirée (ici la couleur rouge), si la fonction de détection répond positivement pour un pixel, nous sauvegardons ses coordonnées dans une liste.

3.3 Calcul de la position de l'objet sur l'image et calcul d'erreur

A partir de la liste générée précédemment, nous pouvons calculer le barycentre de tous les points d'intérêt de l'image. Nous aurions pu le calculer directement sans avoir recours à la création d'une liste mais nous avons choisi cette solution car c'est celle qui ouvrait le plus de possibilités si nous avions voulu améliorer le processus de traitement (diminution du bruit ou détection de formes par exemple). La fabrication de données est moins performante en terme de temps qu'un calcul du barycentre tout au long du parcours de l'image, cependant cette technique donne de bons résultats comme nous le verrons par la suite.

Le calcul de barycentre est un moyennage de la position (des vecteurs à l'origine) de chaque points d'intérêt a contenu dans la liste. Il en résulte un b représentant la position du barycentre de l'objet soit l'erreur de position de la caméra en pixels.

$$\vec{b} = \frac{\sum_{i=0}^{n} \vec{a_i}}{n}$$

Il enfin adapter ce vecteur d'erreur pour que son origine soit le centre de l'image et non le coint haut, gauche de l'image.

$$\vec{b_x} = (W/2) - b_x$$
$$\vec{b_y} = (H/2) - b_y$$

Avec W la largeur de l'image et H la hauteur de l'image (en pixels). A la suite de quoi, nous avons enfin notre vecteur d'erreur à partir duquel nous pouvons générer des instructions à transmettre à l'arduino.

4 Tests, Résultats et Performance

4.1 Performances de la détection de couleurs

Les performances de notre méthode d'analyse sont assez concluantes, en utilisant les seuils valeurs de seuil et d'écart que nous avons déterminées manuellement, nous arrivons aux résultats suivants :

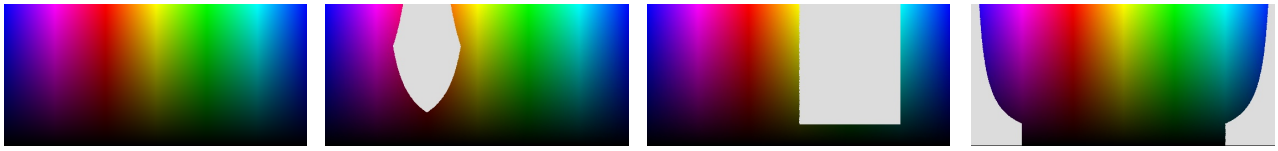


Image vierge Rouge Vert Bleu

FIGURE 7 – Résultats de la détection des couleurs primaires RVB

Nous avons utilisé principalement le rouge pour faire fonctionner le robot c'est pourquoi c'est la couleur avec la détection la plus restreinte car nous voulions éviter au maximum d'avoir du bruit. Les autres détections de couleur ne sont pas si restreint mais fonctionnent quand même dans une certaine mesure. La détection du vert fonctionne aussi plutôt bien même si elle a tendance à capter un peu trop dans le bleu et le jaune. Enfin la détection du bleu fonctionne pour le bleu foncé mais elle a tendance à capter trop de couleurs sombres et donne beaucoup de bruit.

4.2 Performances de tracking

Pour mesurer les performances de tracking de notre système, nous avons réalisé plusieurs expériences. A l'aide d'un rover parallax pour avoir une répétabilité de l'expérience optimale.

4.2.1 Tracking Horizontal

Pour tester le tracking horizontal, nous avons mit en place l'expérience suivante :

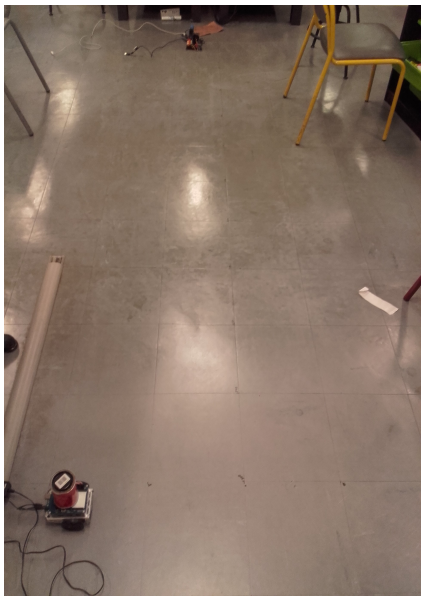


FIGURE 8 – Photographie de l'expérience

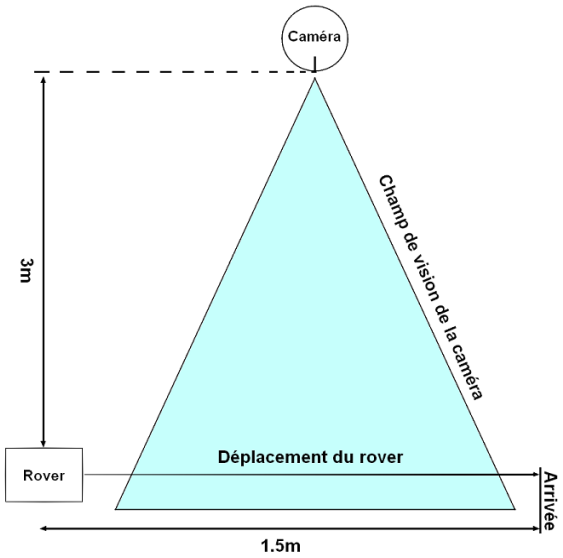
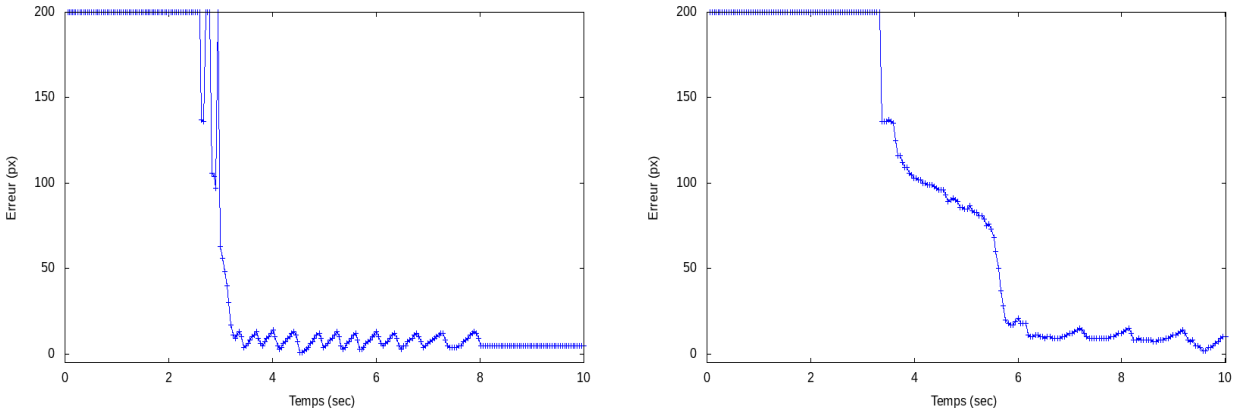


FIGURE 9 – Schéma de l'expérience

Le rover porte un object que nous désirons suivre avec la caméra et il est placé en dehors du champ de vision de la caméra à 3m de distance. Ensuite, nous lançons la caméra puis nous mettons le rover en route et il se déplace de 1.5m en ligne droite. De ce fait, il entre dans le champ de vision de la caméra qui va rapidement commencer à suivre l'objet. Ensuite, la caméra suit l'objet du le reste de la distance jusqu'à ce que le rover atteigne l'arrivée.

Mesures Les mesures réalisées avec et sans double vitesse sont visibles sur la Figure 10. On peut voir que pour la double vitesse, le mouvement initial de la caméra quand l'objet entre dans le champ de vision est plutôt brutal mais rapide. Au contraire, on peut voir que le mouvement initial de la caméra sans la double vitesse est lent et progressif. Cela démontre bien que notre système de double vitesse est utile et efficace. Ensuite, on peut remarquer qu'après le mouvement initial de la caméra, le tracking est régulier et l'erreur très faible tout au long du mouvement du rover. Il y a cependant une petite différence entre le tracking avec et sans double vitesse, le tracking sans double vitesse semble donner des mises à jour plus lentement que le tracking avec double vitesse. Nous ne savons pas vraiment pourquoi, peut-être qu'il y avait plus de bruit sur l'image quand nous avons testé sans double vitesse ce que a ralentis le calcul de l'erreur.



Avec double vitesse Sans double vitesse

FIGURE 10 – Graphiques d'erreur de l'expérience de tracking horizontal

4.2.2 Tracking Diagonal

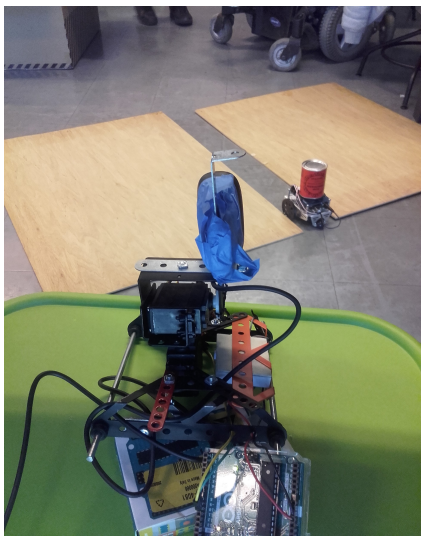


FIGURE 11 – Photographie de l'expérience

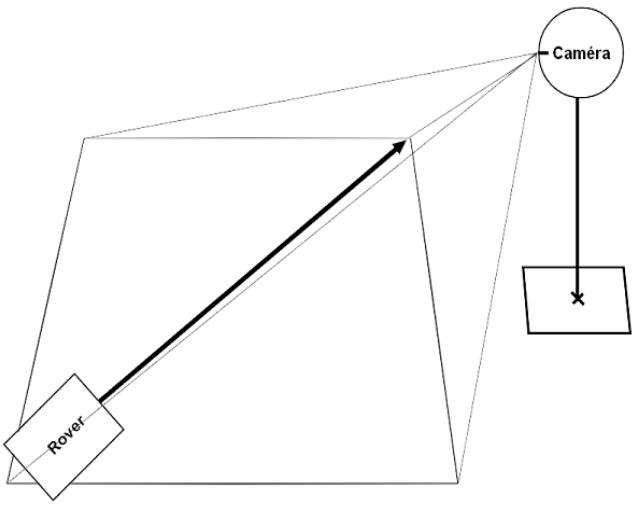


FIGURE 12 – Schéma de l'expérience

Comme le tracking horizontal n'utilisait qu'un axe de déplacement, nous avons voulu mettre à l'épreuve un tracking utilisant aussi la hauteur. Nous utilisons encore un rover parallax sur lequel on place l'objet à tracker pour permettre une répétabilité de l'expérience. Le rover est placé entre deux plaques de bois pour être sûr qu'il ne sort pas de son chemin car il avait tendance à dévier à gauche. Dans cette expérience, le rover commence sa course dans le champ de vision de la caméra puis décrit une ligne droite. La caméra est placée en hauteur pour pouvoir suivre l'objet (qui se déplace en diagonale depuis sa perspective).

Mesures Nous avons choisi de ne pas montrer les tests réalisés sans double vitesse parce que l'objet commence sa course dans le champ de vision de la caméra, il n'y a donc pas beaucoup de différence. Sur la Figure 13 nous pouvons voir le graphique de l'erreur de positionnement de la caméra au cours du temps. Nous observons toujours le premier mouvement très brusque qui initialise la caméra sur l'objet. Ensuite, nous pouvons remarquer que le tracking est plus chaotique que pour le tracking horizontal, c'est encore dû aux conditions de l'expérience. En effet de la perspective de la caméra le rover ne décrit pas une diagonale parfaite ce qui a pour effet qu'il n'y a parfois besoin de déplacer la caméra que dans une seule direction au lieu de deux. Du coup, l'arduino ne reçoit pas toujours les mêmes instruction donc l'erreur est plus chaotique. Ce phénomène est bien visible sur la Figure 14 qui représente l'erreur horizontale (rouge) et l'erreur verticale (vert) au cours du temps. On voit bien que l'erreur dans les deux sens est déphasée, l'erreur horizontale se corrige plus vite que l'erreur verticale car nous envoyons d'abord la commande horizontale. Enfin, vers 8s on peut voir que seulement l'erreur verticale est corrigée ce qui rejouint nôtre explication précédente où nous disions qu'à certains moments nous ne corrigeons l'erreur que sur un seul axe. Nous pouvons aussi noter que les périodes que décrit l'erreur sont de plus en plus longues. C'est parce qu'au fur et à mesure que le rover s'éloigne, le champ de vision de la caméra est plus grand et donc l'objet met plus de temps à quitter la zone morte.

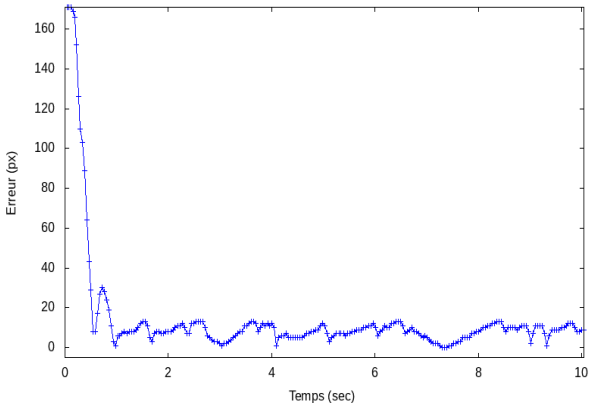


FIGURE 13 – Erreur de placement de la caméra

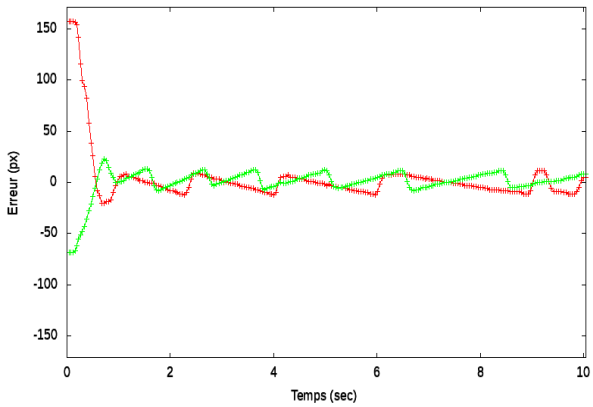


FIGURE 14 – Erreur de placement par axe

Remerciements Merci à Mr Pierre Andry de proposer cette option, nous avons apprécié travailler en autonomie sur cette problématique simple mais faisant appel à beaucoup de connaissances diverses. Merci aussi à mon coéquipier Aurelien Leroy qui a beaucoup apporté à la réflexion et à la conception de ce projet notemment sur la communication avec l'arduino.

Ressources Quelques vidéos parlant de la transmission d'informations par liaison série que vous pouvez recommander et qui m'ont aidé sur le sujet (en anglais) : Ben Eater (youtube).

- Reliable data transmission - https ://www.youtube.com/watch ?v=eq5YpKHXJDM
- Error detection : Parity checking - https ://www.youtube.com/watch ?v=MgkhrBSjhag
- Checksums and Hamming distance - https ://www.youtube.com/watch ?v=ppU41c15Xho