EPFL

Ecole Polytechnique Fédérale de Lausanne

Rapport Miniprojet Robotique MT-BA6

Groupe 55

Naël Dillenbourg
Gaël Sprüngli

Mai 2022

Table des matières

1 Introduction

Notre projet est axé sur la mesure de couleur avec la caméra du e-puck 2 (PO8030D). Sur impulsion du bouton utilisateur, notre robot se déplace alors sur un cercle chromatique pour finir sa course à l'endroit correspondant à la couleur capturée. Les capteurs de proximité permettent au robot de ne pas rentrer dans un obstacle.

2 Implémentation

2.1 Choix d'implémentation

2.1.1 Mesures des couleurs RGB

La mesure d'une couleur se fait à l'aide de moyennes afin d'obtenir une valeur la plus représentative possible de la valeur d'origine. Nous récoltons donc la moyenne de la valeur RGB d'un carré de 20 pixels fois 20 pixels situé au milieu de notre image. Cette technique est répétée sur 10 images pour obtenir une valeur RGB moyenne globale qui est transformée en valeur HSV. Cette technique est décrite en Figure 1.

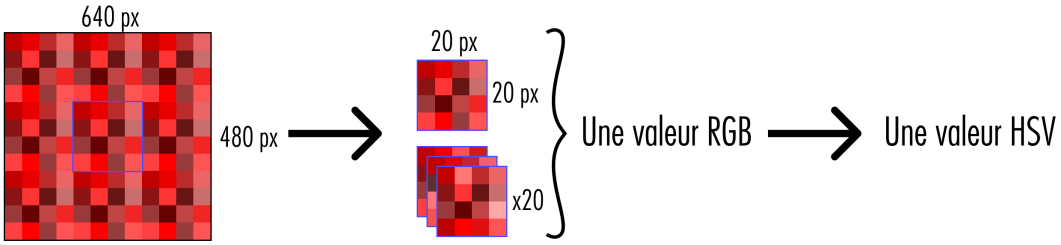

FIGURE 1 – Résumé de la méthode d'évaluation par moyenne de la couleur capturée.

La fonction de transformation des valeurs RGB à HSV est une fonction que nous avons modifié à partir du code de *Sunidhi Bansal*[1]

2.1.2 Représentation 2D d'un système à trois axes

L'information que nous fourni la caméra est deux registres 8 bits qui ensemble fournit le code RGB565 [2] d'un pixel. Le code RGB565 est un système de représentation des couleurs avec 3 valeurs :

- 5 bits pour décrire l'intensité de la couleur rouge, pouvant donc aller de 0 à 31
- 6 bits pour décrire l'intensité de la couleur verte, pouvant donc aller de 0 à 63
- 5 bits pour décrire l'intensité de la couleur bleu, pouvant donc aller de 0 à 31

Afin de représenter sur un plan 2D la couleur, nous avons fait le choix d'un système de représentation de la couleur compatible. Il est évident que tout choix se fera nécessairement avec des compromis car nous perdons de l'information en passant d'une représentation 3-dimensionnelle de la couleur vers une représentation 2-dimensionnelle. Après considération, nous avons donc choisi de faire représenter la couleur par un format HSV avec un paramètre "value" toujours à 1. [3]

2.1.3 Détection d'objet

La détection d'objet se fait par *pooling* sur les valeurs déclarées par le thread du capteur. Le thread qui gère le mouvement des moteurs a la responsabilité de vérifier si les capteurs IR de l'epuck détectent un objet à proximité.

2.2 Mouvement du robot

2.2.1 Passage de couleur HSV en coordonnée

Les deux autres paramètres du format HSV sont toutefois utilisés. Le paramètre "hue" correspond à un angle compris entre 0 et 360° sur le cercle de couleur. Le paramètre "saturation" correspond à la norme du vecteur par rapport au centre du cercle, allant de 0 à 100, correspondant à un pourcentage d'intensité. Afin de simplifier l'opération, un changement de variable passant de coordonnées polaires à cartésiennes est effectué. Nous avons donc :

- $x = saturation * cos(hue), x \in [-saturation, saturation]$
- $y = saturation * sin(hue), y \in [-saturation, saturation]$

Cependant, pour effectuer ce changement de variable, il faut avant tout passer du référentiel du cercle de couleur HSV au cercle unitaire. En effet, sur le cercle chromatique du format HSV, présenté en Figure 2, le 0 est situé au 90° du cercle trigonométrique, et le sens de rotation se fait dans le sens inverse. Il faut donc passer l'angle par un changement :

$$angle_{polaire} = (-angle_{HSV} + 90) mod 360$$

Ces valeurs de x et y sont ensuite converties en centimètre. Pour cela, étant donné que r varie de 0 à 100, on normalise x et y en les divisant par 100, puis ces valeurs sont multipliés par le rayon du cercle de couleur en centimètre.

- $x\_cm = R\_cercle\_cm * x/100$
- $y\_cm = R\_cercle\_cm * y/100$

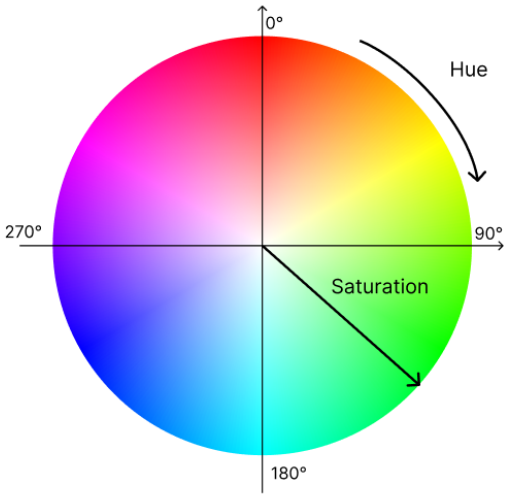

FIGURE 2 – Cercle de couleur du format HSV, avec un angle décrit par le paramètre "Hue" allant de 0 à 360° et une norme décrite par le paramètre "Saturation" allant de 0 à 100%. Source image : [4]

2.2.2 Aller à la position

La méthode pour passer les coordonnées du robot ainsi que les coordonnées cible est de déclarer des variables globales dans le fichier "motor_coordinate.c" puis sont prisent via des setters dans le fichier "process_image.c". L'angle et la norme de la couleur sont donc transférée dans le fichier des moteurs, puis la conversion se fait en x et y, afin de respecter l'encapsulation. La structure "pos_robot" représente la position de l'epuck, tandis que la structure "pos_obj" représente l'objectif à atteindre. Quand il reçoit un set de coordonnée, le robot se retrouve face à quatre situations possible, comme nous pouvons le voir en Figure 3.

Suite à la détermination du cas, le robot se déplace jusqu'en (x,y) à partir de (robot_x, robot_y) en se tournant dans la bonne direction, puis en avançant de $|robot\_x - x|$ pour arriver en (x,robot_y). A cette position, le robot se tourne ensuite dans la bonne direction puis avance de $|robot\_y - y|$. La valeur absolue est appliquée sur la norme du déplacement pour éviter des problèmes de distance négative. Un schéma de déplacement du robot est montré en Figure 4.

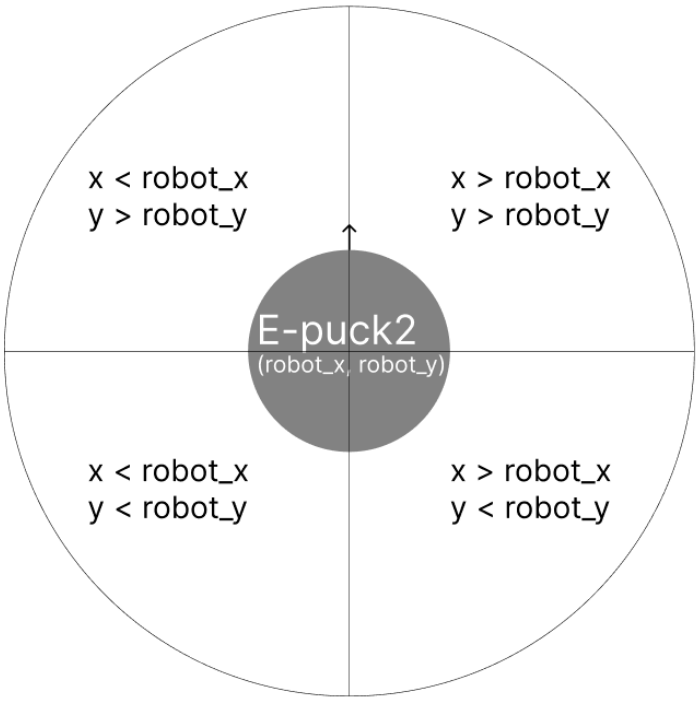

FIGURE 3 – Le robot en gris, orienté dans le sens de la flèche noire, positionné en (robot_x,robot_y), se retrouve face à quatre cas possible au moment où il reçoit une nouvelle position. Chaque quadrant correspond à une zone autour de lui vers laquelle il doit aller en fonction des valeurs de x et y, représentant la position à atteindre

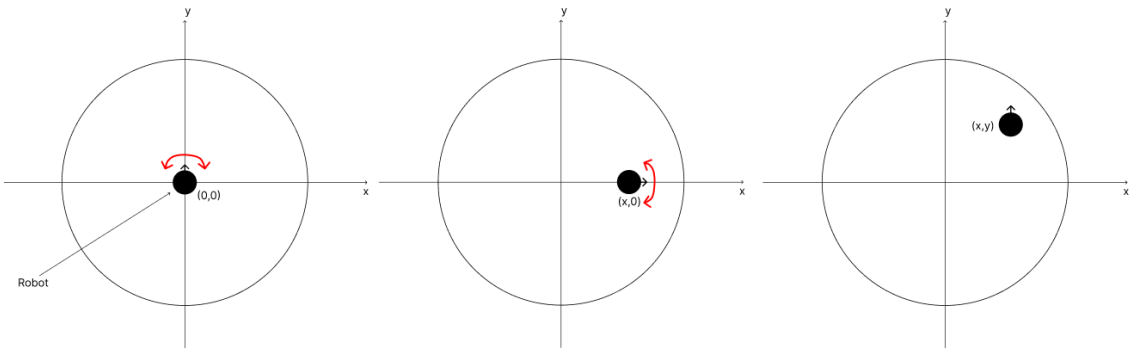

FIGURE 4 – Cette figure représente la séquence de déplacement du robot sur le cercle chromatique. La flèche noire sur le robot représente son orientation et les flèches rouges représentent les choix possibles qu'il a pour s'orienter.

3 Organisation du code

3.1 FSM

Afin de communiquer entre chaque thread, notre code se repose sur une FSM [5] (Finite-State Machine) ou machine d'état fini. Cette machine d'état va permettre, grâce à une condition, de passer d'un état à un autre dans notre code. Grâce au schéma explicatif en Figure 5, nous pouvons voir les transitions entre les différents états.

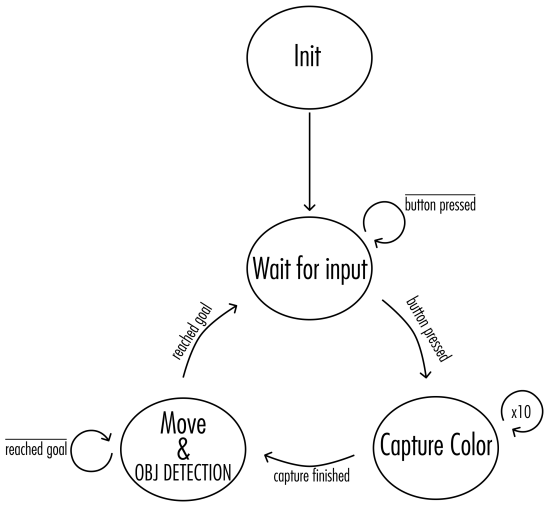

FIGURE 5 – Schéma explicatif de notre FSM. Après l'initialisation, on arrive dans l'état "Wait for input" jusqu'à un appuie sur le bouton "User" du robot. Suite à ça, la FSM passe à l'état "Capture Color", qui, dès qu'une couleur est enregistrée, va passer à l'état "Move & Object detection". Le robot va ainsi se déplacer jusqu'à l'endroit indiqué, jusqu'à ce qu'il y arrive ou qu'il détecte un objet. En cas de détection d'un obstacle, la machine met en pause l'epuck. Si l'objet n'est plus détecté, le robot reprend sa course jusqu'à arriver où il est remis dans l'état "Wait for input.

3.2 Architecture

Nous utilisions une architecture dite "bottom-up". Des sous modules implémentent différentes fonctionnalités de du projet.

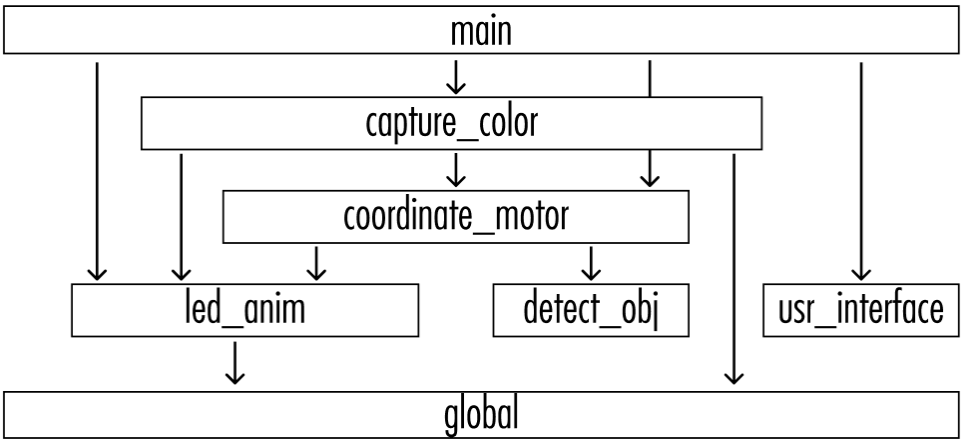

FIGURE 6 – Schéma de l'architecture du projet

4 Fonctionnement

4.1 Manuel

A l'allumage, notre robot se met en mode "Wait for input". Cela se traduit par les leds qui tournent dans le sens des aiguilles d'une montre. Pour qu'il commence la capture d'image et la détection de couleur, il faut appuyer sur le bouton "User". Il se met ainsi en mode "Capture Color". Le temps qu'il capture une couleur, le robot va allumer les leds pour les faire tourner dans le sens inverse des aiguilles d'une montre. Quand il a enregistré une couleur, il allume les quatre leds RGB puis va bouger en direction de la couleur. A l'arrivée, il repasse en mode "Wait for input" en faisant tourner les leds rouges dans le sens des aiguilles d'une montre. Si pendant sa course le robot détecte un obstacle, il s'arrête et allume toutes les leds en rouge.

4.2 Essais

Nous avons réalisé les essais de notre projet à la lumière du jour, en journée. Dans ces conditions, le robot arrive à la bonne position. Le soir, dans une lumière artificielle, il arrive que la lumière perçue soit significativement modifiée. Nous pouvons donc voir l'impact de la lumière ambiante sur la caméra. Pour ce qui est du déplacement du robot, toute erreur lors du placement du robot ou de glissement est multiplié par le nombre de différentes couleurs que nous lui montrons. Il faut donc faire attention à être précis.

5 Annexes

5.1 Code source

Notre code source est disponible sur la page github de notre projet :
https ://github.com/naelopode/Robotics-HSV-Color-Detection

6 Conclusion

En conclusion, ce projet a permis d'apporter une approche pratique à la théorie apprise pendant le cours. Nous avons pu réunir les éléments du cours et des travaux pratiques afin d'arriver à un résultat satisfaisant. Nous avons aussi pu expérimenter une différente manière de coder, au moyen d'un système embarqué. En effet dans ce genre de cas la gestion de la mémoire et certains paramètres "hardware" rentrent en comptent et il ne faut pas les oublier (limite mémoire liée à la caméra, précision de l'image, limite de vitesse des moteurs, etc.). Cela nous a permis d'expérimenter avec les threads, sémaphores et autres fonctionnalités applicables lors du traitement d'un système d'exploitation à temps réel.

Références

[1] S. Bansal, “C program to change rgb color model to hsv color model,” www.tutorialspoint.com. [Online]. Available : https://www.tutorialspoint.com/c-program-to-change-rgb-color-model-to-hsv-color-model

[2] T. Barth, “Rgb565 color picker,” Barth Development. [Online]. Available : http://www.barth-dev.de/online/rgb565-color-picker/

[3] J. Howard Bear, “The hsv color model in graphic design,” https://www.lifewire.com/what-is-hsv-in-design-1078068, Lifewire. [Online]. Available : https://www.lifewire.com/what-is-hsv-in-design-1078068

[4] “Hsl,” Wikipedia. [Online]. Available : https://en.wikipedia.org/wiki/HSL

[5] “Fsm,” Wikipedia. [Online]. Available : https://fr.wikipedia.org/wiki/Automate_fini