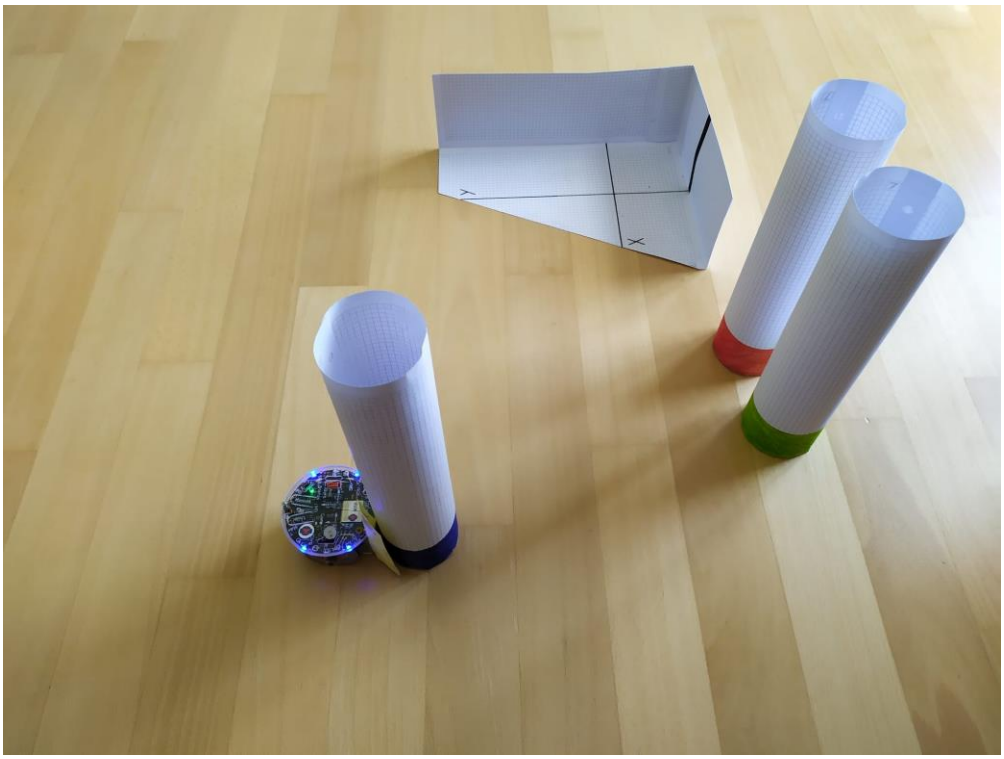
Microinformatique

Micro-315

Professeur:
Francesco Mondada

# Rapport du Projet de Microinformatique



Loïk Vuilleumier
Tim Bürgel

EPFL

# 1 Introduction

Le but du projet est que le robot range des cylindres colorés. Il y a trois cylindres: un rouge, un vert et un bleu. Pour ranger un cylindre, en premier le robot le détecte avec le capteur “Time of flight” (TOF). Ensuite il s’approche pour détecter la couleur du cylindre avec la caméra. Finalement le robot pousse le cylindre jusqu’à l’endroit désigné, dépendant de la couleur du cylindre. Ce processus est répété pour chaque cylindre.
Ce projet est réalisé lors du cours Microinformatique pendant le 6ème semestre de Bachelor en Microtechnique et sert à appliquer les concepts appris pendant les cours d’introduction et de s’habituer travailler avec des microcontrôleurs et des capteurs en pratique.
Tous les mots étants écrit entièrement en majuscule sont des constantes ou des états de la machine d’état provenant du code.

# 2 Principe de fonctionnement

## 2.1 Matériel utilisé

Cylindres colorés:

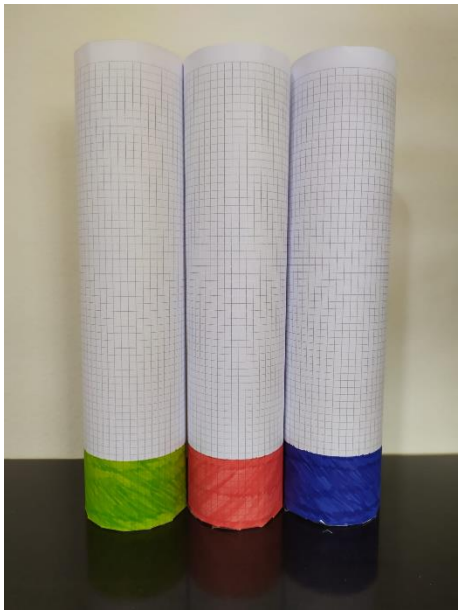


*Figure 1:Cylindres colorés lestés avec de la monnaie*

Base:

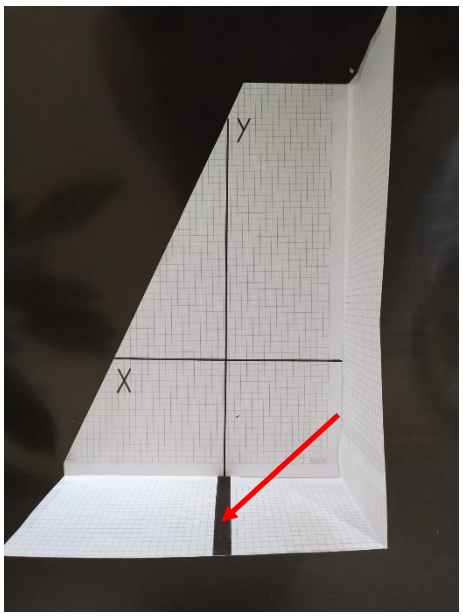



*Figure 2: Base utilisée le homing du robot. La flèche rouge pointe sur la ligne large utilisée pour la réorientation angulaire.*

Robot e-puck2:

- Capteur TOF VL53L0X
- Caméra PO8030D
- 2 Moteurs Pas-à-Pas
- 4 LED RGB et 4 LED rouges
- attachement pour pousser les cylindres

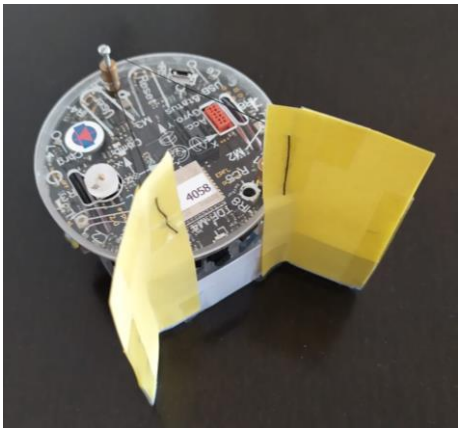


*Figure 3: Robot e-puck2 avec l'attachement pour pousser les cylindres*

## 2.2 Trajectoire

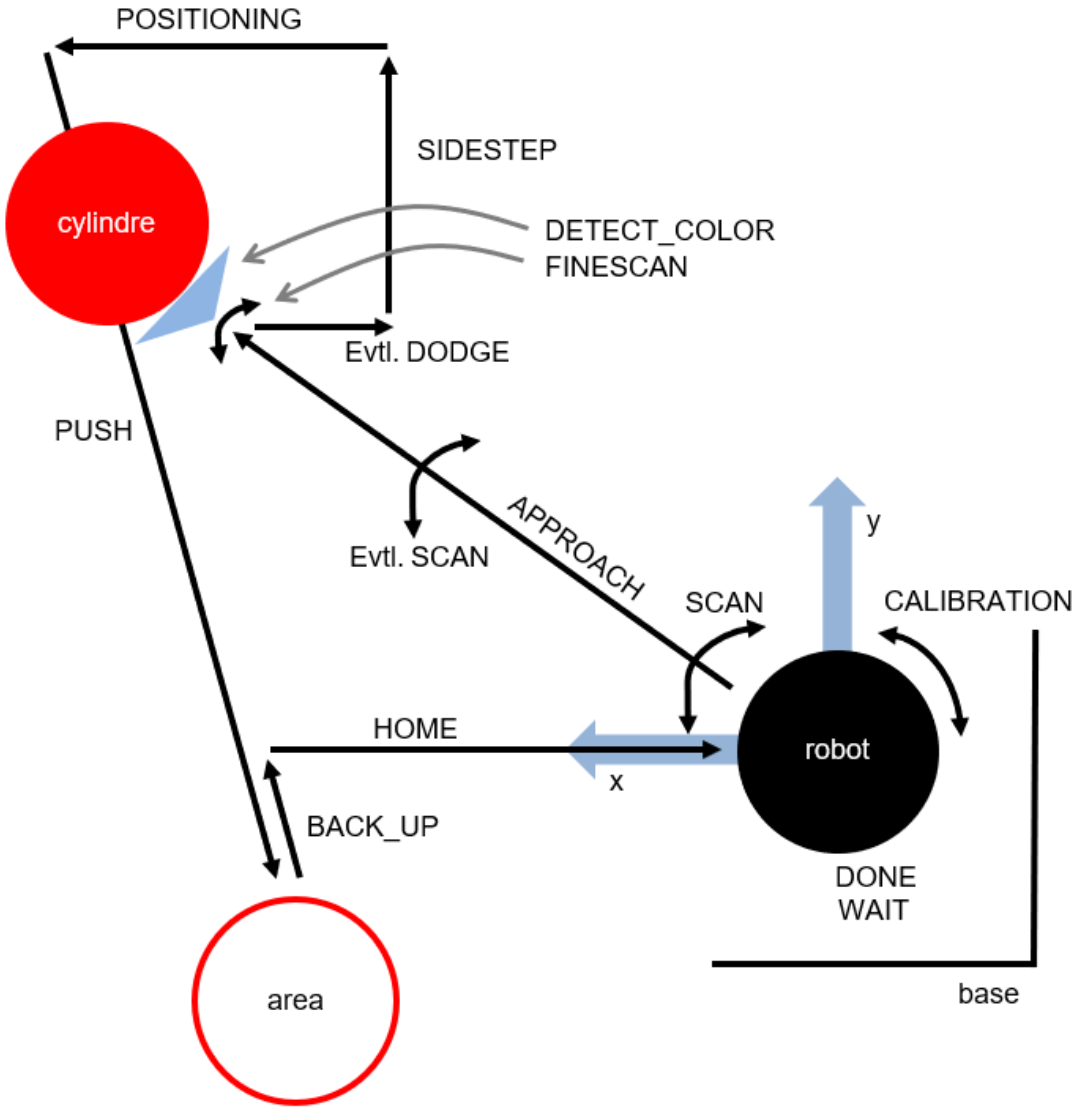



*Figure 4: La trajectoire du robot pour ranger un cylindre est représentée ici. À côté des flèches sont les noms des états de la machine d'état. Les flèches bleues indiquent le système de coordonnées. Dans ce diagramme le robot se trouve à l'origine étant orienté selon l'axe y ce qui est la position de départ .*

Le selector sert comme bouton de démarrage et d'arrêt. Lorsque le bit le plus important est à 1 la machine d'état commence à tourner.
Le robot débute à l'origine et démarre par la calibration du capteur TOF. Ensuite il commence à scanner. Cela veut dire de tourner sur lui-même jusqu'à ce qu'un cylindre est détecté (c.a.d. le capteur TOF mesure une distance faible). Quand il détecte un cylindre il s'y approche. Si le cylindre est perdu de vue, le robot retourne dans l'état SCAN. Une fois arrivé à proximité du cylindre, le robot fait un FINESCAN pour viser le milieu du cylindre en cherchant d'abord les bords du cylindre. Puis la couleur du cylindre peut être détectée par la caméra. Le robot indique la couleur détectée par les LEDs RGB. Ensuite le robot contourne le cylindre et se met en position pour pouvoir pousser le cylindre dans une ligne droite jusqu'à la position finale. Quand le cylindre est à son endroit désigné le robot se remet à l'origine en effectuant le homing et refait le même cycle pour les deux autres cylindres.

## 2.3 Odométrie

Le robot a deux modes de déplacement: se déplacer sur une ligne droite ou tourner sur lui même. Ceci permet de calculer assez facilement la position absolue du robot par rapport à l'origine à partir du nombre de pas qu'effectuent les moteurs. Toutes les 10ms le nombre de pas des moteurs et la position précédente sont convertis en une nouvelle position absolue. Cette position absolue et aussi les mesures du capteur TOF déterminent la trajectoire du robot.

## 2.4 Calibration

Au début du programme, le robot fait un quart de tour à droite et pointe donc directement vers le mur parallèle à l'axe y de la base. Le robot effectue une mesure avec le capteur TOF. Cette mesure est comparée avec la distance réelle qui est de 8cm entre le centre du robot et le mur, pour définir l'offset du capteur TOF.

## 2.5 Positioning

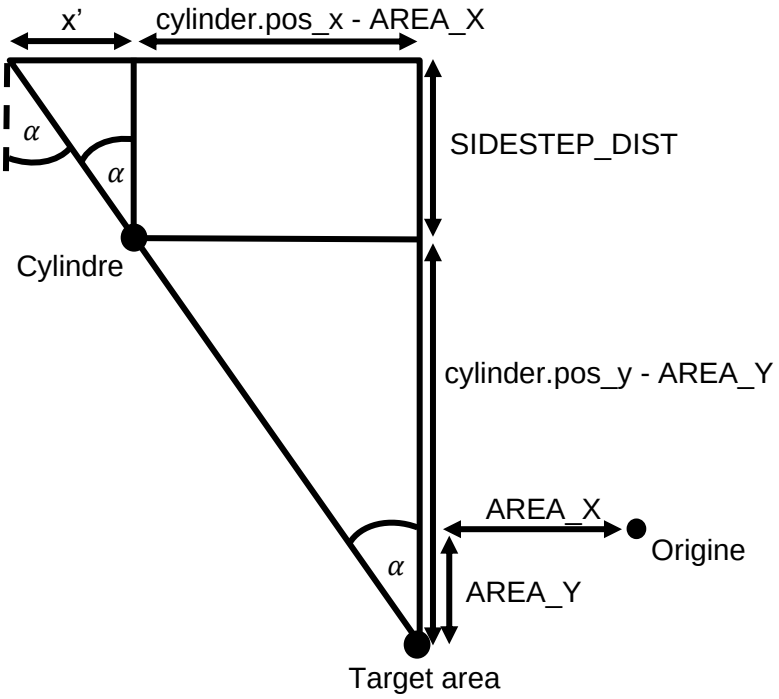



*Figure 5: Calcul de la position finale de l'étape POSITIONING, AREA_Y est négatif*

Les formules (1) à (3) et la figure 5 explicitent les calculs trigonométriques pour obtenir la position que le robot doit avoir après l'état POSITIONING et l'angle pour l'état PUSH.

$$(1)\ \tan(\alpha) = \frac{\text{cylinder.pos\_x} - \text{AREA\_X}}{\text{cylinder.pos\_y} - \text{AREA\_Y}} = \frac{x'}{\text{SIDESTEP\_DIST}}$$

$$(2)\ \text{x\_target} = \text{cylinder.pos\_x} + x' = \text{cylinder.pos\_x} + \text{SIDESTEP\_DIST} \cdot \frac{\text{cylinder.pos\_x} - \text{AREA\_X}}{\text{cylinder.pos\_y} - \text{AREA\_Y}}$$

$$(3)\ \text{target\_angle} = \pi + \text{atan}\left(\frac{cylinder.pos\_x - AREA\_X}{cylinder.pos\_y - AREA\_Y}\right)$$

## 2.6 Homing

Le calcul de la position absolue à partir de l'odométrie dépend de plusieurs paramètres intrinsèques au robot, p. ex. la distance entre les roues et le périmètre des roues. La précision avec laquelle on peut déterminer ces paramètres est limitée. Cela conduit à des erreurs sur la position du robot. Pour limiter l'effet de ces erreurs nous avons mis en place un système de "homing". Après le rangement d'un cylindre, le robot retourne à l'origine pour s'occuper du prochain cylindre. Lors de ce retour à l'origine le robot n'utilise pas la position calculée grâce à l'odométrie, mais le capteur TOF qui détecte les murs de la base. La caméra est utilisée pour détecter la ligne large sur le mur arrière de la base. Ceci permet de corriger l'angle du robot en supposant que le centre du robot se trouve sur l'axe y. Tout ce système permet donc d'éviter l'accumulation d'erreurs pendant les trois cycles. La détection de la ligne noire par la caméra est faite de la même façon que pendant le TP4 en cherchant la région de minimum d'intensité.

# 3 Organisation du code

## 3.1 Threads

| Nom | Description | Priorité |
|---|---|---|
| TOF | Lit les valeurs du capteur TOF et les stocke dans la structure correspondante. Ces valeurs seront appelées par PosControl pour calculer la trajectoire. Le thread est copié collé depuis VL53L0X.c avec une période d'appel modifiée pour obtenir une valeur de distance toutes les 50ms. Comme il s'agit d'un thread d'acquisition de valeurs mesurées il a la plus haute priorité. | NP + 10 |
| PosControl | Contient la machine d'état qui s'occupe de la trajectoire du robot. Ce thread a une priorité élevée pour garantir une fréquence d'appel de 100Hz constante afin de régulièrement calculer la nouvelle position et d'obtenir une bonne précision des mouvements. | NP + 1 |
| CaptureImage | Lit les valeurs envoyées par la caméra, les stocke dans un tableau et envoie des sémaphores à la bonne fonction quand il a terminé. Les sémaphores sont en particulier "image_ready_sem_color" pour déclencher la détection de couleur et "image_ready_sem_center" pour déclencher la détection de la ligne noire large pour faire la réorientation angulaire lors de l'étape homing.<br>Ce thread reste inactif la plupart du temps et n'est exécuté que lors d'un appel take_image() depuis PosControl qui active le sémaphore "take_img_sem". On a quand même décidé de créer un thread pour que la prise d'image puisse être réalisée en arrière-plan lorsque PosControl n'est pas actif. | NP |
| ProcessImage | Traite les informations fournies par CaptureImage et détecte la couleur au centre de l'image en faisant la moyenne pour chaque couleur sur 10 pixels et en trouvant la moyenne la plus grande. Cette fonction nécessite une illumination assez bonne pour bien fonctionner.<br>Ce thread reste inactif le plupart du temps et n'est exécuté que lors de l'activation du sémaphore "image_ready_sem_color" depuis CaptureImage. On a quand même décidé de créer un thread pour que le traitement d'image puisse être réalisé en arrière-plan lorsque PosControl n'est pas actif. | NP - 1 |

| | | |
|---|---|---|
| LEDControl | Lit la couleur détectée par ProcessImage et allume les LEDs correspondantes. Ce thread s'occupe aussi de faire clignoter les 4 LEDs rouges si la couleur n'est pas encore détectée. Il possède une priorité basse car la fréquence d'appel de 2Hz n'est pas essentiel pour le fonctionnement du programme. | NP - 5 |

*Table 1: Aperçu des threads du programme*

## 3.2 Fonctions sinf() et cosf() de math.h

Dans la fonction new_coord_and_angle on utilise les fonctions trigonométriques sinus et cosinus fournies par la librairie math.h. Comme ces fonctions nécessitent beaucoup de calcul nous avons fait des mesures sur la durée de la fonction new_coord_and_angle.
Les mesures nous ont donnés des temps d'exécution pour toute la fonction autour de 30µs. Comme cette fonction n'est appelée qu'une fois par période de 10ms nous avons décidé que cette perte de temps est négligeable et de ne pas créer une lookup table.

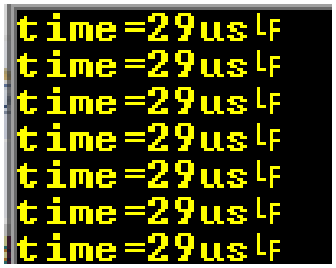



*Figure 6: Exemples des mesures sur la durée de la fonction new_coord_and_angle*

## 3.3 Types des variables

Toutes les variables concernant des angles sont du type float car on a besoin d'une très bonne précision. On calcule tous les angles en radians. De même les variables concernant la position du robot sont du type float. Comme on calcule la nouvelle position très souvent des erreurs d'arrondissement s'accumuleraient et la précision sur la position serait insuffisante.
Toutes les variables concernant le positionnement du cylindre par contre sont du type uint16_t car cette position n'est calculée qu'une seule fois. Avec la position en mm une précision suffisante peut être garantie.
Pour stocker les variables principales du robot et du cylindre actuel nous avons décidé d'utiliser 2 structures statiques globales. Vu que ces variables sont utilisées dans la plupart des fonctions nous avons décidé de les définir hors du thread pour ne pas devoir les envoyer vers chaque fonction séparément.

# 4 Défis rencontrés

## 4.1 Précision

Au début nous avions un problème de précision. Après un cycle (le rangement d'un cylindre) le robot se trouvait à environ 5cm de l'origine. Pour régler ce problème nous avons d'abord ajusté les paramètres comme la distance entre les roues et le diamètres des roues. Ceci a amélioré la précision mais ce n'était pas encore suffisant. Nous avons donc créé la base pour pouvoir introduire le homing. Par contre il n'avait pas que la position qui était imprécise mais aussi l'angle du robot. Pour corriger l'angle nous avons utilisé la caméra qui détecte la ligne large sur le mur de la base. En utilisant la position de la ligne dans l'image une correction de l'angle est faite indépendamment de la distance entre le robot et la ligne.

## 4.2 TOF

Le capteur TOF ne rend pas toujours des valeurs très correctes. Parfois, mais pas toujours, une valeur correcte est cruciale pour le bon fonctionnement du programme. Si c'est le cas, nous prenons une mesure supplémentaire, jusqu'à avoir 2 mesures consécutives suffisamment proches pour obtenir une meilleure précision et une élimination des fautes de mesures.
Le capteur TOF a aussi un offset qui n'est pas toujours le même. Pour régler ce problème nous avons introduit une étape de calibration au début du programme. L'offset est déterminé de telle manière pour que la distance corrigée soit celle entre le centre du robot et l'objet.

## 4.3 Détection des couleurs

D'abord une illumination suffisamment bonne est nécessaire afin de garantir que la détection de couleur fonctionne proprement. Ensuite la caméra n'a apparemment pas tout à fait la même sensibilité pour les trois couleurs de base rouge, vert et bleu. Pour compenser cela les valeurs moyennes détectées sont corrigées en les multipliant avec des valeurs de correction. Ces valeurs sont:

| | |
|---|---|
| RED_CORRECTION | = 1.2 |
| GREEN_CORRECTION | = 0.8 |
| BLUE_CORRECTION | = 1.2 |

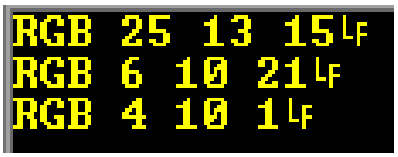



*Figure 7: Exemples des mesures de la détection des trois couleurs des cylindres. Du haut vers le bas: rouge, bleu, vert.*
*L'illumination du cylindre vert était moins forte.*

Comme la caméra possède certaines fonctions automatiques, par exemple la luminosité et une correction automatique des blancs, il est possible que ces paramètres de correction doivent être adaptée pour un environnement différent. Nous avons quand même décidé de ne pas modifier ces fonctions automatiques ou de les fixer sur des valeurs spécifiques car le code fonctionne très bien tout en restant le plus simple possible. Finalement on a déterminé à travers plusieurs essais que la qualité de l'image dépend de la distance à l'objet. La distance optimale est autour de 200mm. Ceci est probablement dû à la correction automatique des blancs qui distord l'image si on est trop proche du cylindre. Si on est trop proche tout l'image aurait la même couleur ce qui provoque des corrections indésirables.

# 5 Conclusion

Pendant les dernières semaines nous avons réussi de réaliser notre projet prévu au début du semestre. Ayant résolu les défis comme discuté dans le chapitre 4 le programme fonctionne à notre satisfaction. Les plus grands soucis que nous avons rencontrés étaient les imprécisions et les limites physiques des capteurs, par exemple la portée limitée et parfois des fautes de mesures du capteur TOF. A travers beaucoup de tests pratiques et en introduisant quelques simples filtres nous avons à la fin réussit de vaincre ces soucis. Cela nous a quand même montrés les difficultés de l'interface entre le monde analogique avec les mesures réelles et le monde digital avec une précision idéale.
A cause du Covid-19 nous avons dû travailler chacun depuis chez nous et qu'une personne (Tim Bürgel) avait le robot. L'utilisation de git et github nous a permis de quand même travailler en équipe. Par contre la distribution des tâches a dû être adaptée. Vu que seulement Tim avait le robot, il s'est occupé de la partie interaction avec le monde physique pendant que Loïk s'est plutôt occupé de la partie du contrôle de position.

# 6 Sources

Cours Microinformatique Micro-315 printemps 2020, donné par Prof. F. Mondada aux étudiants de la troisième année bachelor, EPFL

- librairie e-puck2_main-processor
- TP4_CamReg

Datasheet du capteur TOF VL53L0X https://www.st.com/resource/en/datasheet/vl53l0x.pdf consulté en avril 2020

Datasheet camera PO8030D projects.gctronic.com/E-Puck/docs/Camera/PO8030D.pdf consulté en avril 2020