RÉPUBLIQUE ALGÉRIENNE DÉMOCRATIQUE ET POPULAIRE
وزارة التعـــــــــليم العـــــــــالي والبحث العـــــــــلمـــي
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
جامعة محمد البشير الابراهيمي برج بوعريريج
Université de Mohamed El-Bachir El-Ibrahimi - Bordj Bou Arreridj
Faculté des Sciences et de la Technologie
Département Électromécanique

MÉMOIRE

Présenté pour l'obtention du **diplôme** de **MASTER**
En : Electromécanique
Spécialité : Electromécanique

Par : **HAOUAM WASSIM**
RIGHI MOUNDIR

Sujet

Développement d'un système autonome contrôlé par des marqueurs visuels utilisant un RASPBERRY PI

Soutenu publiquement, le 02 / 07 / 2025, devant le jury composé de :

Bekkouche Tewfik	Prof	Univ-BBA	Examinateur
Meradi Dounia	MCB	Univ-BBA	Président
Bengueddoudj Abdallah	MCB	Univ-BBA	Encadrant

Année Universitaire 2024/2025

Dédicaces

Nous offrons cet humble travail avec un cœur plein de joie : A ceux qui nous ont été source d'inspiration et de volonté : nos très chers pères et nos très chères mères pour leurs sacrifices et encouragement durant toute la période de nous études.

A tous nos frères et sœurs.

A nos amis et collègues. A tous ceux qui nous ont aidés de près ou de loin à faire ce travail.

Remerciement

Avant tout, nous remercions le bon dieu tout puissant qui nous donne de la foi, du courage et de patience afin d'accomplir ce modeste travail.

Nous tenons à remercier notre encadreur **M. ABDALLAH BENGUEDDOUDJ.**

D’avoir accepté de nous encadrer et pour les efforts qu’il est déployé, pour nous aider, conseiller, encourager et corriger avec une grande gentillesse durant toute cette période.

Nous remercions tous les enseignants qui ont contribué à notre formation sans exception.

Résumé :

Ce mémoire vise à concevoir et à développer un système autonome utilisant des marqueurs visuels pour le contrôle, avec une architecture basée sur un Raspberry Pi. Le système combine des éléments matériels et logiciels pour permettre une navigation précise et une interaction intelligente avec l'environnement. Les marqueurs visuels, tels que les ArUco, servent de points de repère pour guider les actions du système.

Le Raspberry Pi agit comme le cœur du système, assurant le traitement des données et le pilotage des composants connectés. Ce mémoire explore les étapes de conception, d'intégration et de mise en œuvre, tout en mettant l'accent sur les défis techniques tels que la reconnaissance visuelle, l'optimisation des algorithmes, et l'interfaçage matériel. Les résultats obtenus démontrent l'efficacité de cette approche dans des applications variées comme la navigation autonome ou la reconnaissance d'objets, ouvrant des perspectives prometteuses pour des systèmes embarqués intelligents.

Mots-clés : Système autonome, marqueurs visuels, Raspberry Pi, vision par ordinateur.

Abstract:

This thises aims to design and develop an autonomous system using visual markers for control, with an architecture based on a Raspberry Pi. The system combines hardware and software elements to enable precise navigation and intelligent interaction with the environment. Visual markers, such as ArUco, serve as reference points to guide the system's actions.

The Raspberry Pi functions as the core of the system, handling data processing and controlling connected components. This project explores the stages of design, integration, and implementation, with a focus on technical challenges such as visual recognition, algorithm optimization, and hardware interfacing. The results demonstrate the effectiveness of this approach in various applications, including autonomous navigation and object recognition, paving the way for promising developments in intelligent embedded systems.

Keywords: Autonomous system, visual markers, Raspberry Pi, computer vision.

ملخص :

تهدف هذه المذكرة إلى تصميم وتطوير نظام ذاتي التحكم باستخدام علامات مرئية للتحكم، مع بنية قائمة على Raspberry Pi. يجمع النظام بين العناصر المادية والبرمجية لتمكين التنقل الدقيق والتفاعل الذكي مع البيئة. تعمل العلامات المرئية، مثلArUco ، كمعالم لتوجيه عمليات النظام.

يعمل Raspberry Pi كقلب النظام، حيث يتولى معالجة البيانات والتحكم في المكونات المتصلة. يستعرض هذا المشروع مراحل التصميم والتكامل والتنفيذ، مع التركيز على التحديات التقنية مثل التعرف البصري، تحسين الخوارزميات، وتوصيل الأجهزة. تظهر النتائج المحققة فعالية هذا النهج في تطبيقات متنوعة مثل التنقل الذاتي والتعرف على الكائنات، مما يفتح آفاقًا واعدة لأنظمة مدمجة ذكية.

الكلمات المفتاحية: نظام مستقل، علامات بصرية، Raspberry Pi، رؤية حاسوبية.

Table des matières

Liste des figures :

Liste des tableaux :

Introduction Générale

Introduction générale :

Le développement des systèmes autonomes a connu une évolution rapide ces dernières années, grâce à l'intégration de technologies avancées en vision par ordinateur. Ces systèmes, conçus pour fonctionner sans intervention humaine, trouvent des applications variées dans des domaines tels que la robotique, l'automatisation industrielle, la logistique, ou encore la domotique.

Parmi les solutions émergentes, l'utilisation de marqueurs visuels pour contrôler des systèmes autonomes s'impose comme une approche efficace et innovante. Ces marqueurs visuels, tels que les QR Codes, ArUco ou les AprilTags, permettent d'identifier et de localiser des objets dans l'espace avec une grande précision. Ils fournissent une interface simple mais puissante pour établir une communication entre un système de vision et les mécanismes qu'il contrôle.

Dans ce contexte, le Raspberry Pi, un nano-ordinateur monocarte accessible et polyvalent, joue un rôle clé en tant que plateforme de traitement et de contrôle. Son faible coût, sa facilité d'utilisation, et sa compatibilité avec une large gamme de logiciels et de matériels font du Raspberry Pi un choix privilégié pour le développement de systèmes embarqués.

Le présent travail s'inscrit dans cette perspective en visant à concevoir et à développer un système autonome contrôlé par des marqueurs visuels à l'aide d'un Raspberry Pi. Ce système sera capable de :

1. Détecter et identifier des marqueurs visuels pour obtenir des informations contextuelles.
2. Utiliser ces informations pour prendre des décisions autonomes et interagir avec son environnement.
3. Exécuter des tâches spécifiques en fonction des données interprétées, comme la navigation, la manipulation ou la signalisation.

L'approche proposée combine les capacités de vision par ordinateur et les fonctionnalités de traitement embarqué du Raspberry Pi pour offrir une solution intégrée et performante.

Le problème qui se pose est de savoir comment exploiter efficacement les marqueurs visuels pour contrôler un système autonome de manière précise et fiable, tout en maintenant une simplicité d'implémentation et un coût réduit ?

Ce mémoire repose sur une combinaison de techniques de vision par ordinateur, de programmation embarquée et de conception matérielle. Le Raspberry Pi sera utilisé pour la capture et le traitement des images, ainsi que pour le contrôle des mécanismes associés. Des outils et bibliothèques open source, tels que OpenCV, seront exploités pour la reconnaissance des marqueurs visuels et la prise de décision.

À travers ce travail, nous espérons démontrer l'efficacité et la simplicité de mise en œuvre d'un tel système dans des contextes variés, tout en explorant de nouvelles possibilités offertes par les technologies actuelles.

Ce mémoire est organisé comme suit :

Dans le premier chapitre, nous présentons des généralités sur les robots, leur classification ainsi que leurs principales caractéristiques.

- Le deuxième chapitre est consacré à la conception du système, en abordant l'architecture matérielle, les logiciels utilisés ainsi que les algorithmes développés.
- Dans le troisième chapitre, nous nous concentrons sur l'implémentation et l'évaluation des performances. Cette partie couvre la réalisation du système, les algorithmes appliqués et les résultats obtenus.
- Enfin, une conclusion générale vient clore ce mémoire.

Chapitre 01 :
Les systèmes robotiques autonomes et la vision par marqueurs

1. Chapitre 1 : Les systèmes robotiques autonomes et la vision par marqueurs

1.1 Introduction :

Le mot **"robot"** a été utilisé pour la première fois en 1921 par l'écrivain tchèque **Karel Čapek** dans sa pièce de théâtre **R.U.R. (Rossum's Universal Robots)**. Il dérive du terme tchèque **"robota"**, qui signifie **"travail forcé"** ou **"corvée"**. Quant au terme **"robotique"**, il a été introduit pour la première fois par **Isaac Asimov** en 1941 [1].

Les systèmes robotiques autonomes représentent un domaine interdisciplinaire à la croisée de la robotique, de l'intelligence artificielle, de l'informatique, de la mécanique et de l'électronique. Ces systèmes sont conçus pour fonctionner de manière indépendante, sans intervention humaine directe, en percevant leur environnement, en prenant des décisions et en exécutant des actions pour accomplir des tâches spécifiques. En s'appuyant sur des technologies avancées telles que les capteurs, les algorithmes d'apprentissage, la vision par ordinateur, et les techniques de planification et de prise de décision.

1.2 Historique :

L'histoire des systèmes robotiques autonomes remonte à plusieurs décennies et reflète l'évolution des technologies et des idées. Voici quelques étapes clés :

Préhistorique :

On trouve, dans les récits de la mythologie de l'Antiquité grecque, des références à des humanoïdes artificiels, comme l'assistant mécanique fabriqué par le dieu Héphaïstos.Au Ier siècle, Héron d'Alexandrie est considéré comme l'inventeur des premiers automates, selon les descriptions qu'il donne dans son *Traité des pneumatiques*. Il propose notamment une machine utilisant la contraction ou la raréfaction de l'air pour ouvrir les portes d'un temple ou faire fonctionner une horloge, dans le but de « susciter l'étonnement et l'émerveillement ». Une autre préfiguration du robot apparaît dans la mythologie juive avec le golem : un être artificiel, généralement humanoïde, façonné dans l'argile, incapable de parole et dépourvu de libre arbitre, mais conçu pour défendre son créateur. Bien plus tard, au XVIe siècle, apparaissent les machines de Léonard de Vinci. Au XVIIIe siècle, Jacques de Vaucanson construit ses premiers automates dans les années 1730 à des fins ludiques, puis, vers 1750, il perfectionne les métiers à tisser en les automatisant grâce à l'hydraulique [2].

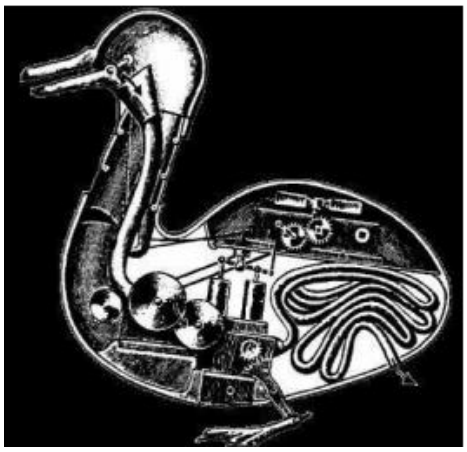


Figure 1.1: Le Canard de Vaucanson, Disparu, a été reconstitue partiellement. Il est exposé actuellement à Grenoble [2].

Années 1960–1980 : Les fondations

- **1961** : Déploiement du robot industriel **Unimate** chez General Motors (premier robot programmable).
- **1966–1972** : **Shakey** (Stanford Research Institute), premier robot mobile autonome utilisant des capteurs et une logique symbolique.
- **Années 1980** : Développement de robots mobiles en laboratoire (ex. : **Genghis** au MIT), équipés de capteurs à ultrasons et de stratégies réactives.

Années 1990–2000 : SLAM et robotique mobile

- **1990s** : Émergence du **SLAM** (Simultaneous Localization and Mapping), permettant aux robots de naviguer dans des environnements inconnus.
- **1997** : Déploiement du rover **Sojourner** sur Mars par la NASA (premier robot extraterrestre semi-autonome).
- **2002** : Lancement du **Roomba** (iRobot), robot domestique autonome utilisant des algorithmes réactifs. [3]

Années 2010–2020 : Révolution de l'IA et des capteurs

- **2010s** : Adoption du **LiDAR** et des caméras 3D pour la perception robotique (ex. : voitures autonomes Tesla, Waymo).
- **2011** : **AprilTag**, système de marqueurs visuels open-source, facilite la localisation précise en robotique.
- **2016** : Démonstrations de robots humanoïdes autonomes (**Atlas** de Boston Dynamics) utilisant le *reinforcement learning*.

Années 2020–Aujourd'hui : Autonomie cognitive et collaboration

- **2020** : Expansion des drones autonomes (ex. : **Amazon Prime Air**) et robots de livraison (Starship Technologies).
- **2021** : Robots chirurgicaux autonomes (**da Vinci 5**) capables d'assister des opérations complexes.

- **2022** : Systèmes **V-SLAM** (ORB-SLAM3) pour la cartographie 3D sans marqueurs.
- **2023** : Intégration de l'**IA générative** (GPT-4) dans la planification de tâches pour robots de service [4].

1.3 Définition et Principes d'un Robot :

La robotique, ou technologie des robots, est un domaine pluridisciplinaire qui couvre la conception, la fabrication, la programmation et l'exploitation des robots [5].

➢ **Définition d'un Robot :**

Un **robot** est une machine programmable, capable d'exécuter des tâches de manière autonome ou semi-autonome, en interagissant avec son environnement physique. Les robots peuvent être conçus pour effectuer des actions variées, allant de manipulations simples à des tâches complexes impliquant de la perception, de la prise de décision et des mouvements précis. Il peut être équipé de capteurs pour percevoir son environnement, de processeurs pour traiter des informations et prendre des décisions, et d'actionneurs pour interagir physiquement avec le monde. Les robots sont utilisés dans divers domaines, allant de l'industrie à la médecine, en passant par les services domestiques et l'exploration spatiale.

Selon l'Organisation Internationale de Normalisation (ISO), un robot est défini comme :

Un manipulateur programmable à plusieurs axes, conçu pour déplacer des matériaux, des pièces, des outils ou des dispositifs spécialisés selon des trajectoires variables programmées pour accomplir des tâches spécifiques [6].

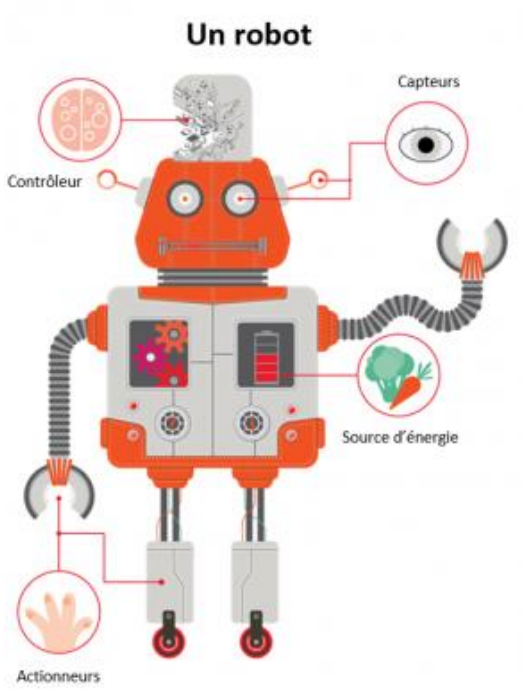


Figure 1.2: Definition d'un robot [6].

➢ **Principes d'un Robot :**

Les principes fondamentaux d'un robot reposent sur sa capacité à interagir avec son environnement, à traiter des informations, et à exécuter des tâches de manière autonome ou semi-autonome.

Dans le domaine de la robotique mobile, la navigation désigne la capacité d'un robot à se déplacer en toute sécurité et de manière efficace d'une position initiale à une position finale. Cela implique de passer d'un point A, défini par les coordonnées (x1, y1), à un point B, défini par les coordonnées (x2, y2). Ce processus repose sur l'intégration de fonctions complexes regroupées en trois grandes étapes, comme illustré à la **Figure** 1.3 :

- **Perception** : Cette étape consiste à détecter les obstacles et, si nécessaire, à modéliser l'environnement. Elle a pour objectif de collecter et structurer les informations indispensables à la prise de décision concernant le déplacement à effectuer.
- **Décision** : Une fois l'environnement perçu et éventuellement modélisé, il s'agit de déterminer le type de mouvement à réaliser en générant des instructions de position ou en choisissant une trajectoire optimale à suivre.
- **Action** : Cette phase consiste à exécuter le mouvement ou la trajectoire définie. Elle implique de respecter les consignes de vitesse reçues en appliquant des commandes adaptées au robot [7].

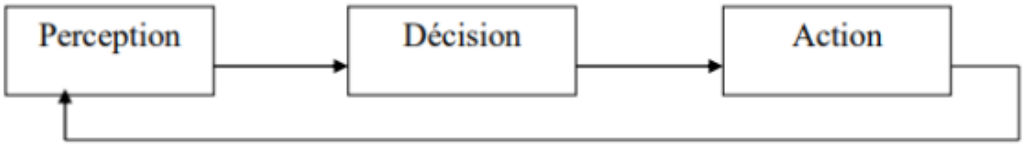


Figure 1.3: Le cycle perception, décision, action que doit réaliser par un robot [7].

1.4 Types des Robots :

Les robots se répartissent en différentes catégories selon leur fonction, leur conception et leur domaine d'application. Voici quelques exemples de types de robots couramment rencontrés :

➢ **Les robots mobiles autonomes (AMR) :**

Les AMR se distinguent par leur capacité à se déplacer de manière autonome tout en prenant des décisions précises en temps réel. De nombreuses entreprises, notamment dans les secteurs de l'agriculture et de la santé, explorent des solutions innovantes pour améliorer l'efficacité opérationnelle, la rapidité, la précision et la sécurité. Les robots mobiles autonomes s'imposent ainsi comme une option privilégiée pour répondre à ces besoins [5].

Figure 1.4 : Robotique Agricole (L'Agrobot Arrosoir) [5].

- **Les humanoïdes :**

Un robot humanoïde désigne un type de robot dont la forme et les caractéristiques rappellent celles de l'être humain, d'où leur appellation de "robots humains". Ces robots sont conçus pour effectuer des fonctions inspirées des capacités humaines et adaptées à la morphologie du corps humain.

Sur le plan technique, de nombreux humanoïdes peuvent accomplir les tâches réalisées par les robots mobiles autonomes (AMR) et vice versa, grâce à la similarité de leurs composants technologiques. Ces robots trouvent leur utilité dans divers domaines, tels que la manipulation d'objets, la soudure ou encore l'assemblage de pièces, notamment dans l'industrie automobile. Ils sont également largement déployés dans des environnements variés, allant des usines aux laboratoires pharmaceutiques et jusqu'aux établissements hospitaliers [5].

Figure 1.5 : Le robot Asimo de Honda [5].

- **Les robots articulés :**

Les robots articulés, également appelés bras robotiques, sont conçus pour reproduire les mouvements d'un bras humain. Dotés de 2 à 10 articulations rotatives, ils offrent une grande

flexibilité et une large gamme de mouvements. Ces robots sont particulièrement adaptés à des tâches telles que la manutention, la soudure à l'arc, l'emballage ou encore l'entretien des machines. Ils sont couramment utilisés dans le secteur industriel pour leur efficacité et leur précision [5].

Figure 1.6 : Cobot UR10e : Robot collaboratif industriel capable d'automatiser des tâches jusqu'à 12,5 kg sans compromettre la précision [5]**.**

- **Les véhicules à guidage automatique (AGV) :**

À la différence des AMR, capables de se déplacer librement dans divers environnements, les AGV (Automated Guided Vehicles) sont confinés à des pistes ou chemins prédéfinis, nécessitant souvent la supervision d'un opérateur. Ces véhicules automatisés sont principalement utilisés pour transporter des matériaux et déplacer des articles au sein d'espaces de travail tels que les usines et les entrepôts [5].

Figure 1.7 : AGR le transport automatisé de marchandises [5]**.**

- **La Téléprésence :**

Les robots de téléprésence offrent la possibilité d'être virtuellement présent dans un lieu sans y être physiquement. En se connectant à un avatar robotique via Internet, il est possible de le piloter à distance, de voir ce qu'il voit et d'interagir avec les personnes sur place [5].

Figure 1.8 : Buddy, le robot qui permet de se téléporter en classe [5]**.**

- **Les robots sous-marins :**

Ces robots trouvent leur terrain de prédilection dans les environnements aquatiques. Ils incluent des submersibles pour les grandes profondeurs, tels qu'Aquanaute, des humanoïdes plongeurs comme Ocean One, ainsi que des systèmes bio-inspirés, à l'image du robot-serpent ACM-R5H [5].

Figure 1.9: Robot-Serpent ACM-R5H [5]**.**

1.5 Critères de choix des robots :

Le choix d'un robot est étroitement lié à son usage prévu. Pour sélectionner le robot le plus adapté, qu'il soit destiné à un usage industriel, domestique, éducatif ou professionnel, il convient de considérer plusieurs critères essentiels : [6].

- La fonction ou l'application visée.
- Les capacités techniques du robot.
- Le degré d'intelligence et d'autonomie.
- La facilité d'intégration et de programmation.
- Le coût en regard du budget disponible.
- L'environnement dans lequel le robot sera utilisé.
- Les aspects liés à la sécurité et à la fiabilité.

1.6 Domaines d'Application des Robots :

- **Domaine industriel :**

Depuis leur introduction sur les chaînes de production dans les années 1970, les robots industriels ont pris une place de plus en plus importante dans de nombreux secteurs où les tâches d'assemblage et de fabrication sont répétitives. Au cours des trente dernières années, les avancées technologiques ont permis de rendre ces robots plus flexibles, rapides et précis. Aujourd'hui, ils occupent une position clé dans plusieurs domaines, notamment :

- La manipulation de produits à haute cadence ;
- Les opérations de conditionnement, incluant la manipulation primaire des produits et la manipulation secondaire liée à l'emballage ;
- L'assemblage de produits dans des secteurs variés comme l'automobile, la pharmacie ou la cosmétique ;
- Les opérations de fin de ligne, telles que la palettisation.

Exemple client – entrepôt :

- Des véhicules automatisés transportent les bacs depuis un système automatisé de stockage et de récupération jusqu'aux postes de tri manuel.
- Ces mêmes véhicules récupèrent les colis terminés pour les acheminer vers le convoyeur de sortie fixe.

Figure 1.10 : Robots industriel [8].

Les domaines d'application présentés montrent que, grâce aux avancées technologiques, les robots peuvent atteindre des performances élevées et accomplir des tâches bénéfiques pour leur environnement. Toutefois, certaines missions restent complexes, voire impossibles à réaliser de manière isolée dans l'espace. Ainsi, en robotique, le recours à un groupe de robots permettant d'intégrer des notions sociales, plutôt qu'à un robot unique, peut considérablement améliorer les capacités globales du système robotique [8].

- **Domaine spatial :**

Les manipulateurs mobiles spatiaux sont conçus pour explorer des environnements inaccessibles, voire dangereux, pour les humains. En d'autres termes, ils interviennent dans des milieux généralement mortels pour l'homme. Aujourd'hui, la conquête spatiale est étroitement liée à la robotique : plusieurs robots sont en activité sur la Station spatiale internationale ainsi que sur Mars. Les futures missions d'exploration continueront d'utiliser des robots (voir Figure 1.12) afin de relever de nouveaux défis et d'approfondir notre compréhension du système solaire. L'étude de robots de tailles variées, capables d'accomplir différentes fonctions et modes de déplacement, vise à développer des machines capables d'exécuter de manière autonome ou complémentaire une large gamme de tâches, rendant leur utilisation toujours plus polyvalente [8].

Figure 1.11: Robot Rover Martien [8]**.**

- **Domaine d'agriculture :**

Après des décennies de recherches et d'expérimentations, le robot a enfin fait son entrée dans les exploitations agricoles. Cette machine entièrement autonome, alimentée par l'énergie solaire, se déplace entre les rangées de cultures pour surveiller et analyser les plantes (voir Figure 1.13). Ayant réussi de nombreux tests sur des parcelles de légumes, le robot est capable de contrôler la santé des cultures. Équipé de nombreux capteurs et caméras, il détecte rapidement toute anomalie comme la présence de mauvaises herbes, de ravageurs ou une croissance insuffisante et informe immédiatement les agriculteurs, leur permettant d'intervenir sans délai [8].

Figure 1.12 : Robot utilisé en agriculture [8]**.**

- **Domaine de service**

Ces dernières années, la révolution robotique a vu de nombreux robots s'installer dans les foyers privés pour accomplir diverses tâches au service de leurs propriétaires [8].

Figure 1.13 : Robots services dans les hôtels et les restaurants [8]**.**

- **Domaine militaire**

L'utilisation des robots dans le domaine militaire est en pleine expansion. Grâce aux avancées en miniaturisation, il est désormais possible de concevoir des robots discrets, dotés de nombreux capteurs, parfaitement adaptés aux missions de reconnaissance ou d'espionnage, comme illustré à la figure I.16 ci-dessous. Par ailleurs, certains de ces robots sont armés et peuvent intervenir dans des environnements hostiles, remplaçant ainsi les soldats afin de réduire les pertes humaines [8].

Figure 1.14 : Robots utilisés dans le domaine militaire.

- **Domaine de transport**

Les véhicules électriques, grâce à leur fonctionnement propre, silencieux et efficace, représentent un pilier clé de la mobilité de demain. Les constructeurs automobiles développent des solutions performantes pour électrifier les moteurs, qui étaient à l'origine entièrement mécaniques [8].

Figure 1.15 : Robot voiture pour le transport [8]. **Figure 1.16 :** Le robot taxi.[8].

1.7 Systèmes de Perception en Robotique :

La notion de perception en robotique mobile fait référence à la capacité du robot à collecter, traiter et interpréter les informations nécessaires pour agir et réagir dans son environnement. Autrement dit, il s'agit de la faculté du robot à détecter et appréhender son environnement, qu'il soit proche ou éloigné. Alors que dans des tâches de manipulation, l'environnement du robot est souvent relativement structuré, la navigation autonome dans des espaces partiellement connus est beaucoup plus complexe. Pour extraire les informations essentielles à la réalisation de ses missions, le robot doit être équipé de nombreux capteurs mesurant à la fois son état interne et celui de son environnement. Le choix des capteurs dépend naturellement de l'application ciblée. La perception est indispensable pour assurer la sécurité du robot, modéliser son environnement et éviter ou contourner les obstacles [9].

1. Capteurs proprioceptifs

Les capteurs proprioceptifs fournissent, par intégration, des informations élémentaires sur les paramètres cinématiques du système mobile. Ils mesurent généralement des grandeurs telles que vitesses, accélérations, angles de giration et d'altitude. Cependant, ils ne fournissent pas de données lorsque le système est à l'arrêt. On distingue deux grandes familles de capteurs proprioceptifs :

- **Capteurs de déplacement :** incluent les odomètres, accéléromètres et radars Doppler. Ils mesurent les déplacements élémentaires, ainsi que les variations de vitesse ou d'accélération sur des trajectoires rectilignes ou courbes.
- **Capteurs d'attitude :** mesurent les angles de cap, de roulis et de tangage. On y retrouve les gyroscopes, gyromètres, gyrocompas, capteurs inertiels composites, inclinomètres et magnétomètres, la plupart étant de type inertiel. [9]

2. Capteurs extéroceptifs

Les capteurs extéroceptifs servent à recueillir des informations sur l'environnement extérieur au système mobile, complétant ainsi les données proprioceptives. Pour exploiter ces informations issues de différentes sources, des méthodes de fusion de données sont utilisées. Ces capteurs sont essentiels pour des applications telles que l'évitement d'obstacles, la localisation, la navigation et la modélisation d'environnements. Parmi les principaux capteurs extéroceptifs utilisés en robotique mobile, on trouve : les télémètres (ultrasons, lasers, infrarouges), le GPS, LiDAR (Light Detection And Ranging) et les caméras [9].

- **Degré d'autonomie :**

Le degré d'autonomie d'un système correspond au niveau d'indépendance avec lequel ce système peut fonctionner sans intervention humaine.

Autrement dit, c'est la capacité du système à prendre des décisions, à percevoir son environnement, et à agir tout seul pour accomplir ses tâches.

Par exemple :

- Un robot avec un faible degré d'autonomie a besoin d'être constamment guidé ou supervisé par un humain.
- Un robot avec un haut degré d'autonomie peut naviguer, détecter des obstacles, et accomplir ses missions sans aide extérieure.

Le degré d'autonomie peut varier selon plusieurs critères:

- La complexité des tâches réalisées automatiquement

- La capacité à s'adapter à des environnements changeants
- Le niveau de prise de décision autonome

Dans mon mémoire avec le Raspberry Pi, le degré d'autonomie, c'est un peu comme ça : le robot peut repérer et reconnaître les marqueurs visuels tout seul pour comprendre où il est. Il peut décider de suivre un chemin ou de s'arrêter devant un obstacle indiqué par un marqueur. Par contre, pour des situations plus compliquées ou inattendues, il faudra quand même que quelqu'un intervienne. Donc, on peut dire que mon système est un semi autonome, mais pas complètement il fait les trucs basiques tout seul, mais a besoin d'aide pour le reste.

1.8 Architecture des Robots Mobiles :

L'architecture des robots mobiles se structure en quatre éléments :

- La structure mécanique et la motricité
- Les organes de sécurité
- Le système de traitement des informations et gestion des tâches.
- Le système de localisation [9].

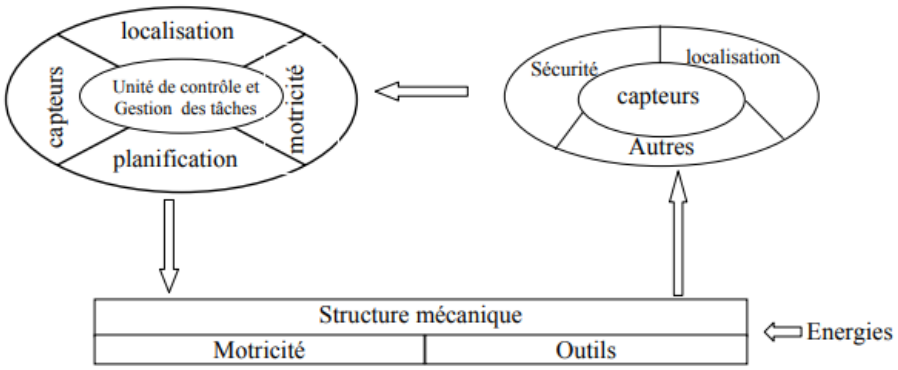


Figure 1.17: Architecture d'un robot mobile [9].

1.9 Systèmes de Localisation :

Pour qu'un robot mobile puisse se déplacer efficacement, il doit connaître sa position dans un espace donné. Les systèmes de localisation peuvent être statiques (lorsque le robot est à l'arrêt) ou dynamiques (en temps réel pendant le mouvement). On distingue principalement deux approches : la localisation à l'estime (relative) et la localisation absolue. La première repose sur des mesures internes, comme les odomètres et les capteurs inertiels, pour estimer la position à partir de déplacements précédents. Elle est indépendante de l'environnement, mais souffre d'une accumulation progressive d'erreurs, nécessitant un recalage régulier. La localisation absolue, en revanche, permet au robot de se repérer directement à l'aide de points de référence fixes dans l'environnement, comme les signaux GPS ou des marqueurs artificiels. Cette méthode repose sur des capteurs extéroceptifs et une carte de l'environnement, indispensable à une navigation précise. En combinant les deux approches (relative et absolue), les robots peuvent corriger leurs erreurs de

positionnement tout en conservant une bonne autonomie, ce qui est essentiel pour leur efficacité dans des environnements variés, qu'ils soient intérieurs ou extérieurs.

1.10 Conclusion :

Ce chapitre propose une introduction globale à la robotique, en abordant ses fondements, son évolution et ses applications. Il débute par une définition du robot. Un aperçu historique. Les différents types de robots sont ensuite explorés. Leurs domaines d'application ..etc. Le chapitre se focalise enfin sur les robots mobiles.

La robotique, à la croisée de la mécanique, de l'électronique et de l'IA, transforme nos sociétés. Les robots mobiles, symboles d'autonomie et d'innovation, illustrent l'importance cruciale de la sécurité dans des environnements dynamiques et interconnectés.

Chapitre 02 :
Conception du Système

2. Chapitre 2 : conception du système

2.1. Introduction :

La vision par ordinateur est un domaine fascinant de l'intelligence artificielle qui permet aux machines d'interagir avec leur environnement en interprétant les images et les vidéos. Grâce à des algorithmes sophistiqués, les robots peuvent détecter, classifier et analyser leur environnement, ce qui leur confère une capacité d'auto-apprentissage et d'adaptation.

2.2. Les systèmes de vision basés sur marqueurs :

2.2.1. Rôle de la vision artificielle dans les robots autonomes :

Les avancées en intelligence artificielle appliquée à la robotique évoluent à un rythme remarquable, permettant la conception de robots capables d'exécuter des tâches de plus en plus complexes avec une intervention humaine réduite. Un exemple notable est **RoboCat** (figure 2.1), développé par DeepMind, qui utilise l'IA pour apprendre de nouvelles tâches après seulement 100 démonstrations. RoboCat exploite ensuite ces apprentissages pour générer des données supplémentaires, augmentant ainsi ses capacités. Cette approche lui permet de faire passer son taux de réussite de 36 % à 74 % après un entraînement complémentaire. Des innovations comme RoboCat marquent une étape significative dans le développement de robots polyvalents capables de s'adapter à une large gamme de missions avec un minimum d'assistance humaine [10].

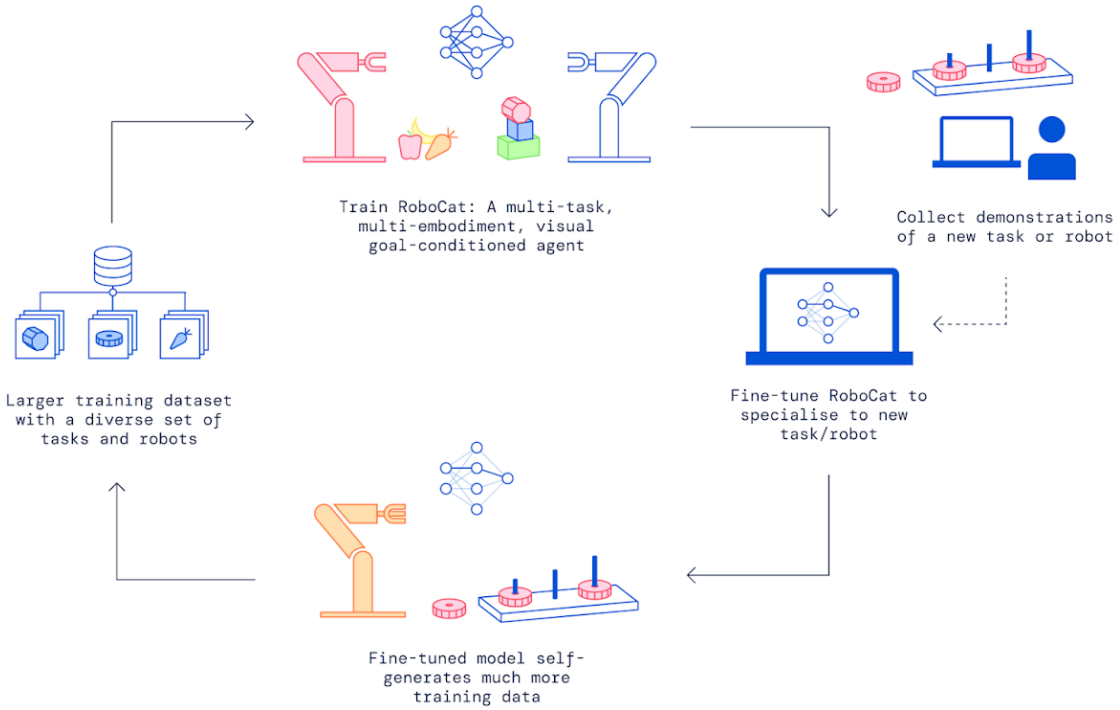


Figure 2.1 : Comment fonctionne le RoboCat de DeepMind [10].

Les systèmes de vision artificielle en robotique, souvent comparés aux yeux d'un robot, permettent à ces machines de reconnaître et de comprendre leur environnement. Ils s'appuient sur des caméras et des capteurs pour capturer des données visuelles, qui sont ensuite traitées par des algorithmes de vision par ordinateur. Grâce à des capacités telles que la détection d'objets, la perception de la profondeur et la reconnaissance des formes, ces systèmes permettent aux robots d'identifier des éléments, d'évaluer leur environnement et de prendre des décisions en temps réel [10].

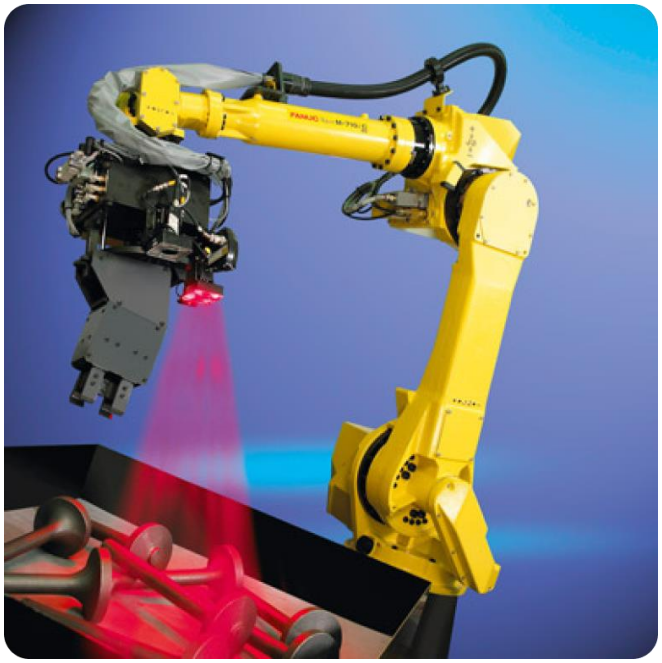


Figure 2.2 : Un robot doté d'un système de vision artificielle [10]**.**

La vision artificielle, joue un rôle crucial dans l'autonomie des robots, en leur permettant d'opérer dans des environnements dynamiques et non structurés. Par exemple, pour qu'un robot puisse saisir un objet, il doit d'abord être capable de le localiser grâce à la vision artificielle. Ce cas simple illustre l'importance de cette technologie, qui constitue également la base pour des applications avancées telles que l'inspection de produits en fabrication ou l'assistance chirurgicale de haute précision.

En fournissant des données sensorielles essentielles à la prise de décisions en temps réel, les systèmes de vision artificielle permettent aux robots d'interagir de manière plus intuitive avec leur environnement. Cela élargit considérablement la gamme de tâches qu'ils peuvent accomplir, rendant leur utilisation pertinente dans un grand nombre de secteurs d'activité.

2.2.2. Avantages de la vision artificielle dans les applications robotiques

La vision artificielle apporte des avantages considérables en robotique, permettant aux machines de réaliser des tâches avec plus d'autonomie, de précision et d'adaptabilité. Voici quelques bénéfices majeurs de son intégration dans les systèmes robotiques :

- **Rentabilité :** En automatisant des tâches exigeant une précision élevée et une cohérence constante, la vision artificielle réduit la dépendance au travail manuel, diminue les erreurs, et accroît la productivité, ce qui génère des économies significatives à long terme.
- **Apprentissage adaptatif :** Grâce à l'analyse continue des données visuelles, les robots peuvent améliorer leurs performances au fil du temps, s'adapter à de nouvelles tâches et répondre aux changements dans leur environnement.
- **Sécurité et conformité :** La vision artificielle renforce la sécurité des robots opérant à proximité des humains en leur permettant de détecter et d'éviter les obstacles, d'identifier les situations dangereuses et de respecter les normes réglementaires.
- **Polyvalence :** Les capacités d'analyse d'images permettent aux robots d'exécuter plusieurs tâches simultanément, comme trier des objets tout en les inspectant, augmentant ainsi leur efficacité globale [10].

2.2.3. Défis de la vision artificielle en robotique

Malgré ses nombreux avantages, l'intégration de la vision artificielle en robotique pose certains défis. Ces obstacles peuvent affecter les performances des robots dans divers environnements et réduire la fiabilité de leurs opérations. Voici quelques défis clés à considérer :

- **Intégration avec d'autres capteurs :** Les systèmes de vision doivent souvent collaborer avec d'autres capteurs, tels que le LiDAR ou les capteurs à ultrasons. Assurer une synergie fluide entre ces capteurs pour fournir une compréhension complète de l'environnement peut être complexe.
- **Coûts élevés de mise en œuvre :** Le développement et le déploiement de systèmes de vision avancés impliquent des coûts substantiels. Trouver un équilibre entre ces investissements et les avantages attendus représente un défi majeur pour de nombreuses organisations.
- **Qualité et accessibilité des données :** Les systèmes de vision artificielle nécessitent de vastes ensembles de données pour leur apprentissage. Cependant, obtenir des données étiquetées de haute qualité, reflétant les diverses situations rencontrées par un robot, peut être difficile. Des données inadéquates peuvent entraîner des modèles moins performants et des résultats peu fiables.
- **Fiabilité dans des environnements variés :** Les systèmes de vision doivent fonctionner de manière cohérente dans des contextes variés, qu'il s'agisse d'environnements intérieurs ou extérieurs. Assurer cette durabilité sans nécessiter d'ajustements fréquents ou d'interventions manuelles reste un défi technique significatif [10].

2.2.4. Présentation des marqueurs visuels (ArUco, QR Code, AprilTag) :

a. QR Codes :

L'histoire des QR Codes, ou « Quick Response Codes », débute au Japon dans les années 1990. Créés par Denso Wave, une filiale de Toyota, ces codes avaient pour objectif initial de suivre

les pièces au sein des usines automobiles. Le besoin était clair : disposer d'un moyen rapide et fiable pour accéder à des informations détaillées sur les pièces tout au long du processus de fabrication. Contrairement aux codes-barres classiques, qui étaient limités tant en capacité de stockage qu'en vitesse de lecture, les QR Codes ont été conçus pour contenir une grande quantité d'informations et être décodés rapidement, d'où leur appellation [12].

- **Fondements Technologiques :**

Les QR Codes sont des codes-barres bidimensionnels capables de stocker des données numériques sous forme de motifs carrés noirs et blancs. Contrairement aux codes-barres traditionnels, qui se limitent à 20 à 25 caractères, un QR Code standard peut contenir plusieurs milliers de caractères, permettant d'y encoder non seulement des chiffres, mais également du texte, des URL, et bien d'autres types d'informations.

La structure d'un QR Code est organisée en plusieurs éléments essentiels :

Marqueurs de position : Placés dans trois des coins du code, ces motifs distinctifs aident le lecteur à identifier l'orientation du QR Code.

Motifs de séparation : Des lignes blanches entourant les marqueurs de position assurent leur distinction par rapport au reste du QR Code, facilitant ainsi le décodage.

Données et clés de correction d'erreur : La partie centrale contient les informations encodées ainsi que des mécanismes de correction d'erreur, garantissant une lecture fiable même si le QR Code est partiellement endommagé ou masqué.

Zone de format : Ce segment fournit des détails sur le niveau de correction d'erreur et le schéma d'encodage utilisé [12].

- **Encodage des Informations**

L'encodage d'un QR Code repose sur un processus élaboré dans lequel les données sont transformées en motifs binaires. Divers modes d'encodage, tels que numérique, alphanumérique, binaire (byte), et kanji, sont utilisés pour maximiser l'efficacité et optimiser la capacité de stockage. Grâce à une technologie avancée de correction d'erreur, un QR Code demeure lisible même si jusqu'à 30 % de sa surface est détériorée [12].

- **Avantages Uniques**

Les QR Codes présentent plusieurs avantages significatifs par rapport aux codes-barres traditionnels :

Grande capacité de stockage : Ils permettent d'intégrer un volume important d'informations dans un espace réduit.

Lecture rapide : Leur conception facilite une lecture instantanée, accélérant ainsi l'accès aux données.

Polyvalence : Ils s'adaptent à de nombreuses applications, du marketing à la gestion logistique.

Résilience : Grâce à leur correction d'erreur, ils restent lisibles même en cas de détérioration partielle.

L'introduction des QR Codes représente une avancée majeure dans la technologie des codes-barres, les positionnant comme un outil essentiel de l'ère numérique [12].

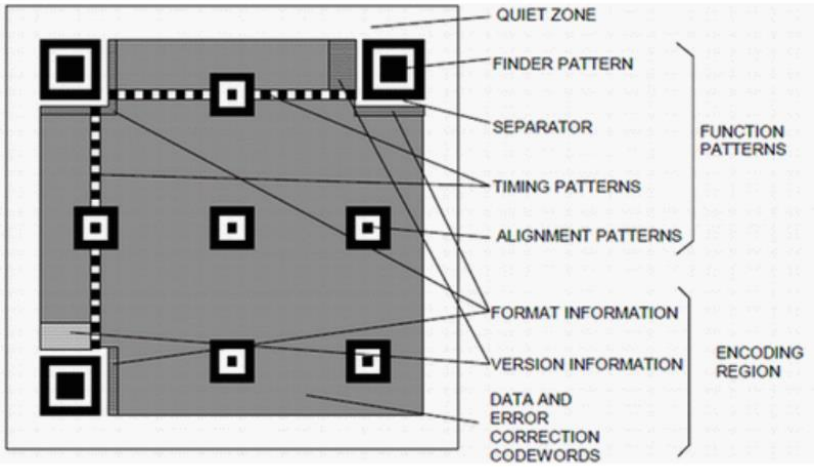


Figure 2.3: Un QR Code [12].

- **Comment fonctionne le QR Code ?**

La particularité des QR Codes réside dans leur capacité à encapsuler et transmettre des informations de manière à la fois compacte et rapide. Mais comment un simple carré peut-il contenir autant de données, et comment ces dernières sont-elles décodées ? Explorons ensemble le mécanisme qui rend cela possible [12].

- **Lecture et Interprétation**

La lecture d'un QR Code commence généralement par un smartphone ou un scanner dédié. L'appareil capture l'image du code via sa caméra, et un logiciel spécialisé interprète les motifs pour en extraire les informations. Voici les étapes principales du processus :

Identification et orientation : Le lecteur détecte les trois marqueurs de position situés dans les coins du QR Code. Ces marqueurs permettent de déterminer l'orientation, la taille et l'échelle du code.

Décodage des données : Le logiciel analyse les motifs noirs et blancs, convertissant les contrastes en une séquence binaire de 0 et 1. Ces motifs binaires représentent les données encodées.

Correction des erreurs : Grâce à la technologie de correction d'erreur intégrée, le logiciel peut reconstituer les informations même si une partie du QR Code est endommagée ou illisible, garantissant ainsi une extraction fiable des données [12].

- **Applications pratiques du QR Code**

Les QR Codes, initialement conçus pour l'industrie manufacturière, sont devenus des outils polyvalents adoptés dans de nombreux secteurs grâce à leur capacité à stocker une grande quantité d'informations dans un format compact, facilement accessible via smartphone. Dans la logistique, ils permettent le suivi en temps réel des colis et la gestion efficace des stocks en entrepôt. En marketing, ils facilitent l'interaction entre marques et consommateurs à travers les publicités imprimées, les emballages de produits ou les événements. Dans le domaine des paiements, ils rendent les transactions mobiles rapides et sécurisées, que ce soit en magasin ou entre particuliers. Enfin, dans l'éducation, les QR Codes enrichissent l'apprentissage en donnant accès à des ressources numériques via des supports pédagogiques ou des visites guidées interactives dans les musées et sites historiques.

b. ArUco :

Les marqueurs ArUco ont été conçus par M. S. Rafael et son équipe à l'Université de Cordoue, en Espagne, dans le cadre de leurs travaux sur la réalité augmentée et la vision par ordinateur. Bien qu'ils aient été initialement développés pour des applications en réalité augmentée, ces marqueurs sont aujourd'hui largement adoptés en robotique pour faciliter une estimation précise de la pose et permettre la navigation autonome dans des environnements confinés.

Les marqueurs ArUco sont des repères fiduciels constitués d'un carré noir contenant un motif blanc unique, associé à un identifiant (ID). Ce design permet une identification rapide et précise. Par exemple, la figure 2.1 illustre un marqueur appartenant à un dictionnaire ArUco de taille 4x4_50, portant l'ID 0. L'encodage sous forme de motifs binaires garantit une détection robuste à l'aide d'algorithmes de vision par ordinateur.

En identifiant ces marqueurs dans les images, il est possible d'utiliser les techniques de localisation basées sur l'image de la caméra, comme évoqué précédemment, pour extraire des informations précises de pose (translation et rotation), assurant ainsi un positionnement fiable.

Figure 2.4. ID de marqueur ArUco 0 dans le dictionnaire ArUco 4*4_50

Les marqueurs ArUco se distinguent par leur simplicité, leur polyvalence et leur facilité de détection, ce qui en fait une solution prisée pour de nombreuses applications en intérieur. Ils améliorent la précision de localisation et les capacités de navigation des robots, leur permettant de déterminer leur position avec exactitude dans divers environnements. Pour une utilisation pratique, il est essentiel d'adapter la taille et la complexité des marqueurs aux dimensions des objets et aux détails de la scène. Un choix judicieux de la taille des marqueurs est donc crucial pour garantir leur détection efficace.

De plus, OpenCV offre une méthode intuitive pour générer ces marqueurs. Le module ArUco d'OpenCV comprend 25 dictionnaires prédéfinis, chacun structuré avec des marqueurs de tailles variées, comme 4×4, 5×5, 6×6 ou 7×7. Chaque dictionnaire contient un nombre fixe de marqueurs, par exemple 50, 100, 250 ou 1000. OpenCV fournit également des fonctions de traitement dédiées aux marqueurs ArUco, ce qui facilite leur intégration dans des environnements à l'aide des bibliothèques système correspondantes.

Un exemple notable est le dictionnaire ArUco original, qui utilise les bits des 2^{e} et 4^{e} colonnes du marqueur pour encoder l'identifiant en binaire naturel, tandis que les autres bits sont réservés à la vérification de parité.

Value	**Data**				
0	1	0	0	0	0
1	1	0	1	1	1
2	0	1	0	0	1
3	0	1	1	1	0

Figure 2.5 : Système binaire utilisé [11]**.**

En examinant la matrice de signature, on constate qu'elle ne permet pas d'assurer l'unicité de la rotation possible pour chaque marqueur. Par exemple, le marqueur 1023 présente une symétrie horizontale [11].

Figure 2.6: Marqueur 1023 [11]**.**

c. AprilTag :

Les AprilTags sont des marqueurs visuels conçus pour être facilement détectés et identifiés par les systèmes de vision par ordinateur. Ces étiquettes carrées affichent un motif noir et blanc unique, permettant aux caméras et aux logiciels de les reconnaître rapidement, ainsi que de déterminer leur position et leur orientation précises dans l'espace 3D. Plusieurs familles d'AprilTags existent, comme la famille Circle 21h7, qui comprend 38 tags numérotés de 0 à 37. Chaque AprilTag possède un identifiant unique, ce qui permet au système de vision IA de les différencier aisément. Cette caractéristique rend les AprilTags particulièrement utiles pour la navigation dans un environnement ou poaur identifier des points d'intérêt spécifiques [13].

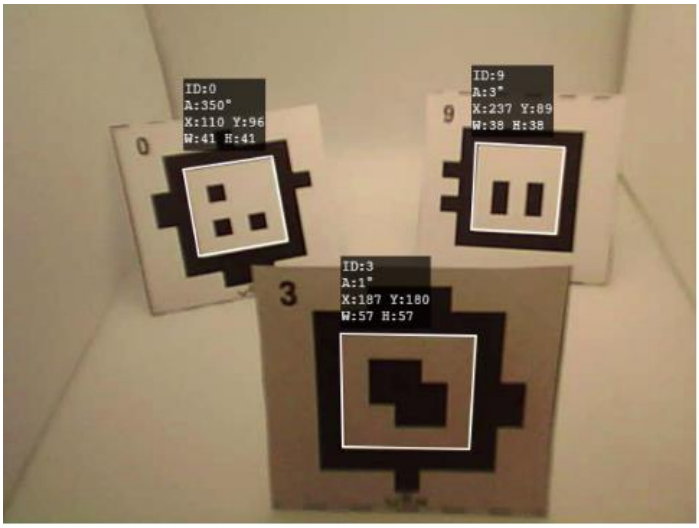


Figure 2.7 : Marqueurs AprilTag [13].

AprilTag est un système de repère visuel, utile pour une grande variété de tâches, notamment la réalité augmentée, la robotique et l'étalonnage des caméras. Les cibles peuvent être créées à partir d'une imprimante ordinaire et le logiciel de détection AprilTag calcule la position 3D précise, l'orientation et l'identité des balises par rapport à la caméra. La bibliothèque AprilTag est implémentée en C sans dépendances externes. Il est conçu pour être facilement inclus dans

d'autres applications, ainsi que pour être portable sur des appareils embarqués. Les performances en temps réel peuvent être obtenues même sur les processeurs de qualité téléphone portable.

La conception du repère et le système de codage sont basés sur un système de codage lexicographique presque optimal, et le logiciel de détection est robuste aux conditions d'éclairage et à l'angle de vue.

Les AprilTags sont conceptuellement similaires aux codes QR, dans le sens où ils constituent un type de code à barres bidimensionnel. Cependant, ils sont conçus pour coder des charges utiles de données beaucoup plus petites (entre 4 et 12 bits), ce qui leur permet d'être détectées de manière plus robuste et à des distances plus longues. De plus, ils sont conçus pour une précision de localisation élevée : vous pouvez calculer la position 3D précise de l'AprilTag par rapport à la caméra [14].

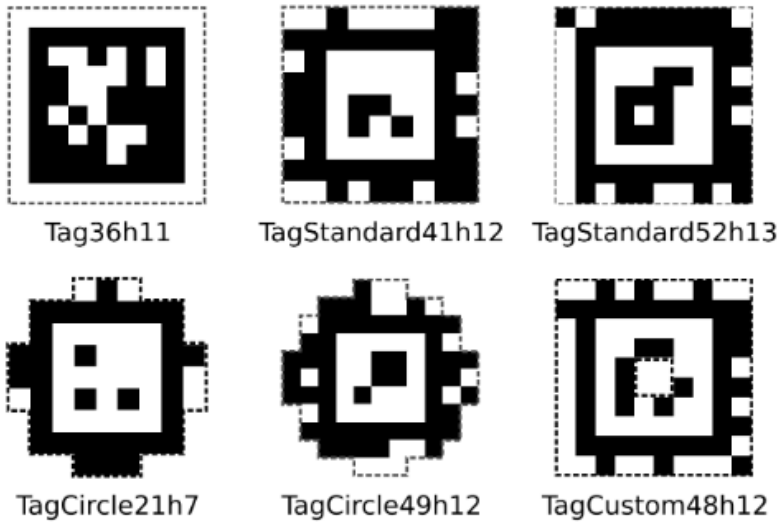


Figure 2.8: Différent Marqueurs AprilTag [14].

2.2.5. Comparaison des différentes technologies de marquage :

En 2005, Mark Fiala a proposé un nouveau système de marqueurs nommé ARTa**g**. Pour démontrer son utilité et ses performances, il a comparé ARTag avec d'autres systèmes existants : Data Matrix, Maxicode, QR Code, ARStudio et ARToolkit. Les critères de comparaison incluaient le taux de faux positifs, le taux de confusion inter-marqueurs, le taux de faux négatifs, la taille minimale des marqueurs et leur résistance aux variations d'éclairage. Les expériences ont conclu que Data Matrix, Maxicode et QR Code ne sont pas adaptés à la réalité augmentée en raison de leurs faibles performances en détection à longue distance, de leur sensibilité aux distorsions et de leur incapacité à permettre une estimation de pose 3D. Ainsi, ARTag n'a été comparé qu'à ARToolkit, un autre système conçu spécifiquement pour la réalité augmentée. Les résultats ont montré qu'ARTag présentait des taux de faux positifs, de faux négatifs et de confusion inter-marqueurs inférieurs à ceux d'ARToolkit. Dans un article ultérieur, Fiala a

confronté ARTag à ARToolkit Plus (une version améliorée d'ARToolkit), confirmant la supériorité des performances d'ARTag [15].

Filippo Bergamasco et al. Ont comparé leur système Rune-Tag avec ARToolkit et ARToolkit Plus. Leur analyse indique que Rune-Tag offre une précision accrue pour l'estimation de position. De plus, la conception circulaire de ses marqueurs confère à Rune-Tag un avantage en résistance à l'occlusion : le système détecte le marqueur même avec deux tiers de l'image occultés.

Un autre système basé sur des marqueurs circulaires résistants à l'occlusion est le CCTag. Comparé à RuneTag et ARToolKit Plus, il affiche le taux de détection le plus élevé en termes de distance, occlusion et flou de mouvement.

ArUco et AprilTag ont été évalués sur des métriques comme les variations lumineuses, les motifs au sol, le flou d'image et la distance de détection. Leurs performances étaient similaires pour la lumière et les motifs, quelle que soit la taille du marqueur. Cependant, AprilTag s'est révélé plus rapide, tandis qu'ArUco était plus flexible pour générer des marqueurs personnalisés.

Le système ChromaTag a été comparé à AprilTag, Rune-Tag et CCTag**.** ChromaTag s'est montré 13 fois plus rapide que le deuxième système, tandis que CCTag était 180 fois plus lent que ChromaTag et 13 fois plus lent qu'AprilTag. L'étude recommande ChromaTag pour une détection à courte distance et perpendiculaire *à* la caméra, et AprilTag pour des angles prononcés ou des distances élevées.

Dans les applications de véhicules aériens sans pilote (UAV), AprilTag, ARTag et ArUco ont aussi été comparés**.** AprilTag a obtenu le meilleur taux de détection en variation de distance, contre le plus faible pour ARTag. En orientation, AprilTag et ArUco ont détecté le marqueur dans 90 % des cas, contre 45 % pour ARTag. En coûts computationnels, AprilTag était le plus élevé, et ARTag le plus bas.

Francisco J. Romero-Ramirez et al. Ont proposé ArUco3, une version optimisée d'ArUco détectant les marqueurs dans une image réduite pour accélérer le traitement. Résultats : ArUco3 est 17 à 40 fois plus rapide qu'ArUco (version OpenCV) et surpasse AprilTag et ARToolkit. Les expériences ont aussi confirmé qu'ArUco était plus rapide qu'AprilTag et ARToolkit [15].

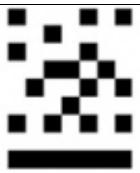
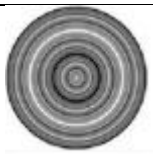
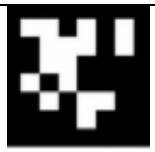
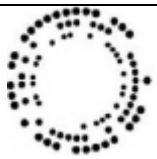
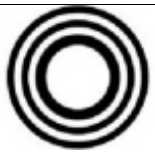
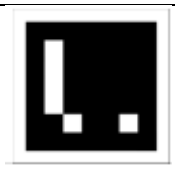
Marker	Year	Marker example	Used in
ARToolkit [29]	1999		Augmented reality
Cybercode [30]	2000		Augmented reality
ARTag [31]	2005		Augmented reality
FourierTag [32]	2007		Robotics, Virtual reality
CALTag [33]	2010		Camera calibration
AprilTag [34]	2011		Augmented reality, Camera calibration, Robotics
Rune-Tag [35]	2011		Localisation, position estimation
CCTag [36]	2012		Camera calibration
ArUco [37]	2014		Augmented reality, Camera calibration, Robotics
ChromaTag [38]	2017		Robotics

Tableau 2.1: Exemple des marqueurs [15].

Voici une comparaison détaillée des technologies de marquage **ARUCO**, **QR Code** et **AprilTag**, utilisées pour la détection visuelle, la réalité augmentée, la robotique ou la traçabilité industrielle :

	Origine	**Structure**	**Capacité**	**Détection**	**Applications**	**Avantages**	**Inconvénients**
ARUCO Markers	Bibliothèque OpenCV	Carré noir avec bordure et grille interne binaire (noir/blanc). Contient un ID unique codé dans la grille.	Nombre limité d'IDs (dépend du dictionnaire choisi, ex : 50, 100, 1000). Pas de stockage de données personnalisées (uniquement un identifiant).	Rapide et efficace, même à faible résolution. Résistance partielle aux occlusions ou distorsions.	Réalité augmentée (estimation de pose caméra). Robotique (navigation, calibration). Suivi d'objets industriels.	Intégration facile avec OpenCV. Léger en calcul.	Capacité de données limitée. Risque de confusion entre IDs similaires dans de mauvaises conditions.
QR Code	Standard ISO (ISO/IEC 18004).	Carré avec trois carrés de repérage aux coins et une grille de modules noirs/blancs. Encodage de données personnalisées (texte, URL, etc.).	Jusqu'à 3 000 caractères alphanumériques (selon la version et le niveau de correction d'erreur). Correction d'erreur intégrée (niveaux L, M, Q, H).	Nécessite une résolution suffisante pour les petits codes. Moins robuste aux angles extrêmes ou aux déformations.	Marketing (liens vers sites web). Traçabilité logistique (stock, emballages). Paiements mobiles (ex : QR Code de paiement).	Standard universel, lisible par la plupart des smartphones. Stockage de données personnalisées.	Taille physique souvent plus grande pour une détection fiable. Sensible aux dommages physiques (ex : pliage, salissure).
AprilTag	Développé par l'Université du Michigan (APRIL Lab).	Carré avec une bordure noire et une grille interne codée (similaire à ARUCO, mais avec une conception optimisée). Codes prédéfinis	Nombre d'IDs limité mais très bien optimisé (ex : 36h11 = 2 048 IDs uniques). Aucun stockage de données personnalisées (ID uniquement).	**Très robuste** aux faibles résolutions, rotations, occlusions et éclairage variable. Algorithmes optimisés pour réduire	Robotique de précision (drones, robots mobiles). Systèmes industriels exigeants (traçabilité haute précision).	Meilleure précision de détection de pose (position et orientation). Résistance supérieure aux conditions difficiles.	Moins répandu que les QR Codes (nécessite des bibliothèques spécifiques). Capacité de données limitée.

		(familles comme 36h11, 16h5).		les faux positifs.	Réalité augmentée professionnelle.		

Tableau 2.2 : Comparaison entre ARUCO Markers, QR Code, AprilTag.

- **QR Code** : Idéal pour le grand public et le stockage d'informations.
- **ARUCO** : Équilibre entre simplicité et performance pour des applications basiques.
- **AprilTag** : Meilleure option pour les systèmes critiques nécessitant robustesse et précision (robotique, industrie 4.0).

2.3. Architecture matérielle :

2.3.1. Choix du Raspberry Pi est ses composants associés :

Le Raspberry Pi est un nano-ordinateur monocarte doté d'un processeur ARM, conçu par David Braben, programmeur britannique de jeux vidéo, dans le cadre de la fondation Raspberry Pi. Ce dispositif de la taille d'une carte de crédit a été créé pour encourager l'apprentissage de la programmation informatique. Il prend en charge plusieurs variantes du système d'exploitation libre GNU/Linux-Debian ainsi que des logiciels compatibles, et peut également fonctionner avec des systèmes comme Windows 10 IoT Core et Android Pi.

Le Raspberry Pi est fourni à l'état brut (carte mère seule, sans boîtier, alimentation, clavier, souris ni écran) afin de réduire les coûts et de favoriser l'utilisation de matériel de récupération. Toutefois, des kits regroupant l'ensemble des accessoires nécessaires sont disponibles [16].

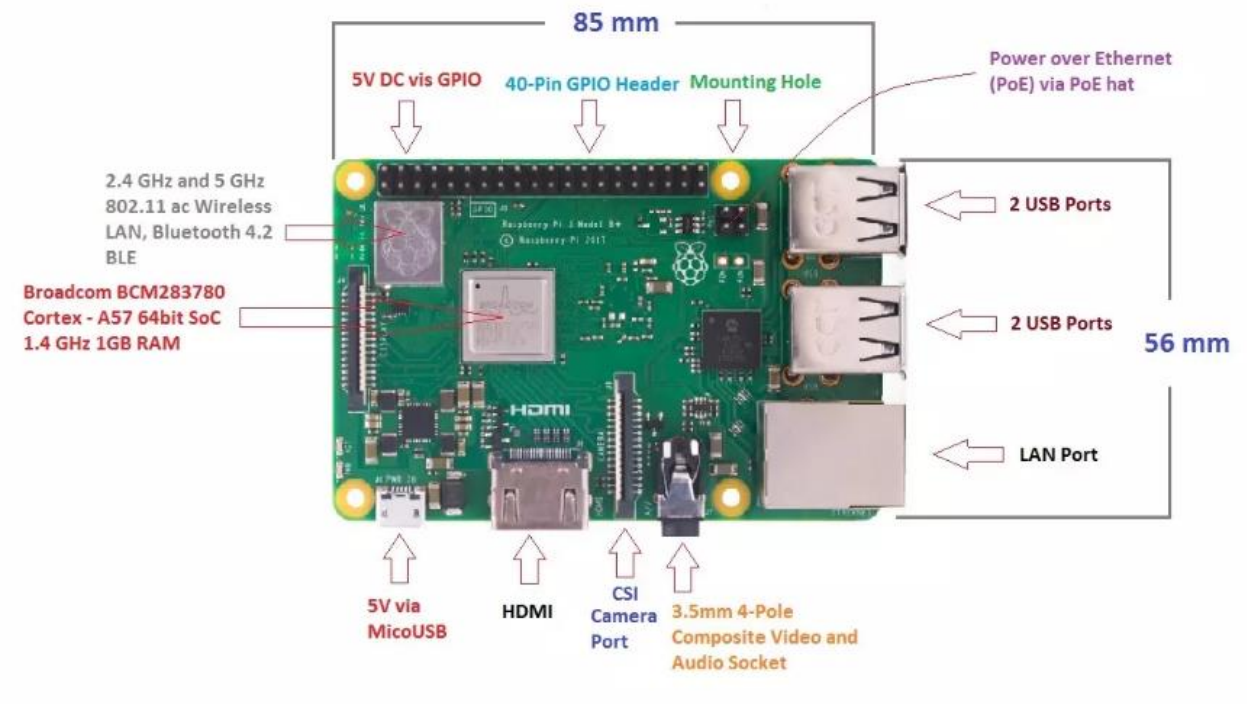


Figure 2.9 : Les E/S du Raspberry pi 3 B+ [16]**.**

Le Raspberry Pi 3 B+ est un ordinateur monocarte compact conçu pour se connecter à un moniteur, ainsi qu'à un clavier et une souris. Il intègre des interfaces Wi-Fi et Bluetooth, offrant une connectivité sans fil polyvalente. Ce modèle démarre à partir d'une carte micro-SD et prend en charge les systèmes d'exploitation Linux et Windows 10 IoT. Afin de réduire les coûts et de promouvoir l'utilisation de matériel de récupération, il est livré sans boîtier, alimentation, clavier, écran ni souris. Le Raspberry Pi 3 B+ repose sur un processeur ARM Cortex-A53 à quatre cœurs, 64 bits, cadencé à 1,4 GHz, accompagné de 1 Go de mémoire RAM. Il dispose d'une interface Wi-Fi, d'une interface Bluetooth, de 4 ports USB, d'un port Ethernet, d'un port HDMI, d'un port micro-SD et d'un connecteur GPIO à 40 broches pour les entrées/sorties. Par rapport à son prédécesseur, le Pi 3, les interfaces Wi-Fi et Bluetooth du Pi 3 B+ ont été améliorées pour prendre en charge le Wi-Fi 2,4 GHz et 5 GHz, ainsi que le Bluetooth 4.2. L'interface Ethernet a également été optimisée, permettant des débits atteignant 300 Mbps, soit deux fois plus rapides que ceux du Pi 3 [21].

Caractéristiques :

- Alimentation à prévoir : 5 Vcc/maxi 2,5 A* via prise micro-USB (* intensité maxi si toutes les fonctions sont utilisées)
- CPU : ARM Cortex-A53 quatre coeurs 1,4 GHz
- Wi-Fi : Dual-band 2,4 et 5 GHz, 802.11b/g/n/ac (Broadcom BCM43438)
- Bluetooth 4.2 (Broadcom BCM43438)
- Mémoire : 1 GB LPDDR2
- Ethernet 10/100/1000 : jusqu'à 300 Mbps
- 4 ports USB 2.0
- Port ethernet 10/100 base T : RJ45
- Bus : SPI, I2C, série
- Support pour cartes micro-SD
- Sorties audio :
  - HDMI avec gestion du 5.1
  - Jack 3,5 mm en stéréo
- Sorties vidéo : HDMI
- Dimensions : 86 x 54 x 17 mm
- Poids : 50 g

Les GPIO

GPIO, acronyme de General Purpose Input/Output, désigne les ports d'entrée/sortie polyvalents présents sur les cartes de développement comme le Raspberry Pi. Ces ports, associés à une adresse spécifique dans la mémoire, permettent à la carte de communiquer et d'interagir avec le monde extérieur.

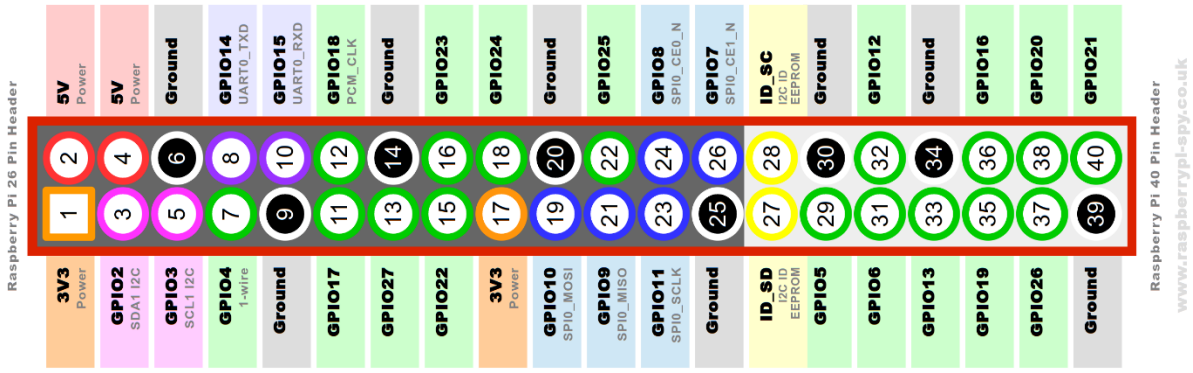


Figure 2.10: GPIO (general purpose for input and output) du Raspberry pi [16]**.**

Comme illustré dans la figure 2.7, il est important de distinguer le nombre de broches physiques (*pins*) du numéro de GPIO attribué à chacune d'elles. Les autres broches peuvent être programmées comme entrées ou sorties numériques, ainsi qu'en sortie analogique. Cependant, elles ne peuvent pas être utilisées comme entrées analogiques. Pour mesurer des valeurs analogiques (comme la température, l'humidité ou la pression), un convertisseur analogique-numérique (CAN) est nécessaire, comme le MCP3208 de Microchip, qui se connecte via une interface SPI [16].

2.3.2. Capteur utilisés (caméra, capteurs de distance, moteurs, etc.) :

Caméra :

La caméra est utilisée pour capturer des images ou des flux vidéo en temps réel afin de détecter et d'interpréter les marqueurs visuels.

Une webcam, c'est une petite caméra connectée à un ordinateur ou une autre plateforme (comme un-Raspberry Pi) qui permet de capturer des images ou des vidéos en temps réel.

- Elle est souvent utilisée pour:
- La visioconférence
- La capture vidéo
- La détection d'objets ou de marqueurs visuels dans les projets de vision par ordinateur

Figure 2.11 : Caméra web 2k [24].

Capteurs de distance :

Les capteurs de distance mesurent la distance entre le système et les obstacles environnants pour éviter les collisions et naviguer en toute sécurité.

- **Types de capteurs couramment utilisés** :
  - **Capteur à ultrasons (par ex. HC-SR04)** :
    - Portée : 2 cm à 4 m.
    - Fonctionnement : Utilise des ondes sonores pour mesurer la distance.
  - **Capteur infrarouge (IR)** :
    - Portée : Quelques centimètres à quelques mètres.
    - Fonctionnement : Détection par réflexion de lumière infrarouge.
  - **LIDAR (Light Detection and Ranging)** :
    - Portée : Plusieurs mètres.
    - Fonctionnement : Utilise un laser pour une mesure de distance précise et cartographie en 2D/3D.

Figure 2.12 : Capteur à ultrasons [25].

Moteurs :

Les moteurs sont utilisés pour permettre le mouvement du système autonome, qu'il s'agisse d'un robot roulant, volant, ou autre.

- **Types de moteurs couramment utilisés** :
  - **Moteurs à courant continu (DC)** :
    - Fonctionnement : Fournissent un mouvement linéaire ou rotatif avec une commande de vitesse variable.
    - Utilisation : Robots roulants.
  - **Servomoteurs** :
    - Fonctionnement : Fournissent des mouvements précis pour contrôler la direction ou les articulations.
    - Utilisation : Bras robotisés ou contrôle de direction.
  - **Moteurs pas à pas** :
    - Fonctionnement : Offrent un contrôle précis de la position en mouvements incrémentaux.
    - Utilisation : Systèmes nécessitant un positionnement précis.
- **Contrôleurs de moteur** :
  - Un contrôleur (ex. L298N, PCA9685) est souvent utilisé pour gérer les moteurs et ajuster la vitesse ou la direction.

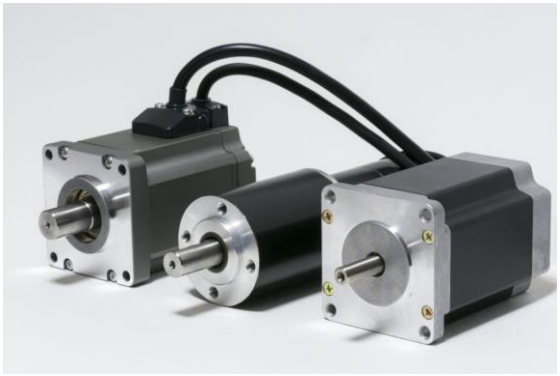


Figure 2.13 : Moteurs pas à pas et Servomoteurs [26]**.**

Interaction avec le Raspberry Pi

- **Caméra** : Connectée via l'interface CSI ou USB. Les images sont traitées avec des bibliothèques telles qu'OpenCV.
- **Capteurs de distance** : Connectés aux broches GPIO pour l'acquisition des données en temps réel.
- **Moteurs** : Commandés via des contrôleurs connectés au Raspberry Pi pour ajuster la vitesse, la direction ou la position.

Calibration de la caméra :

En raison de la courbure irrégulière de la lumière sur les bords de la lentille réelle ou de défauts de fabrication qui rendent la lentille non parfaitement alignée avec le plan de l'image, des distorsions radiales et tangentielles peuvent apparaître. Ces distorsions introduisent des biais dans l'acquisition des images et nuisent à la précision des processus de localisation. Ainsi, une calibration de la caméra s'avère indispensable. Dans ce mémoire, cette étape est réalisée à l'aide de Python, en s'appuyant sur les fonctions de calibration fournies par OpenCV.

Les étapes de calibration sont les suivantes :
a) Imprimer un motif de damier en noir et blanc, le fixer sur une surface plane rigide, et mesurer précisément les dimensions des cases du damier.
b) Configurer la résolution de la caméra et capturer en temps réel les images du damier à l'aide d'un script Python utilisant cv2.VideoCapture().
c) Définir le nombre de coins internes (intersections horizontales et verticales) du damier et leur taille réelle en unités physiques (par exemple en millimètres).
d) Acquérir plusieurs vues du damier sous différents angles, distances et inclinaisons, afin de garantir une couverture spatiale suffisante pour la calibration.
e) Utiliser les fonctions cv2.findChessboardCorners() pour détecter les coins, et cv2.calibrateCamera() pour estimer les paramètres intrinsèques (matrice de calibration) et les coefficients de distorsion.
f) Une fois la calibration terminée, les résultats (matrice de la caméra, vecteurs de distorsion, rotation et translation) sont sauvegardés dans un fichier YAML (par exemple calibration.yaml).
g) Ces paramètres sont ensuite réutilisés pour corriger les images via cv2.undistort() ou pour des applications de vision par ordinateur nécessitant une géométrie d'image précise.

Ce processus permet d'améliorer la qualité des données visuelles et d'accroître la fiabilité des algorithmes de localisation et de détection basés sur la vision artificielle.

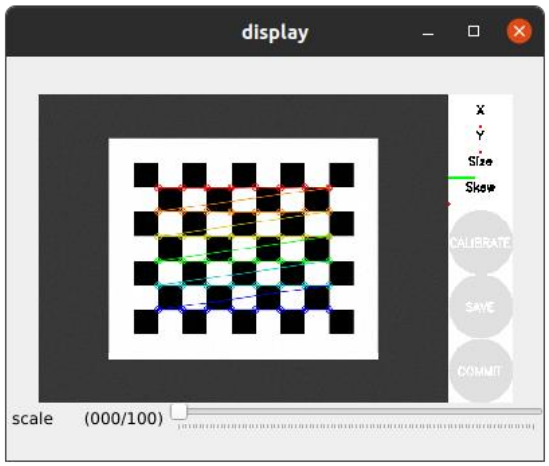


Figure 2.14: Photo de la fenêtre de calibration de la caméra [27]**.**

Lors du processus expérimental, environ 60 images de calibration sont nécessaires pour obtenir les paramètres de calibration. Il est important de noter que, plus le nombre d'images de calibration prises sous différents angles augmente, plus les paramètres de calibration deviennent précis. Cependant, enregistrer un trop grand nombre d'images peut entraîner une charge de calcul excessive, ce qui risque de prolonger le temps de calcul de la calibration et de retarder l'obtention des résultats.

2.3.3. Schéma général du système :

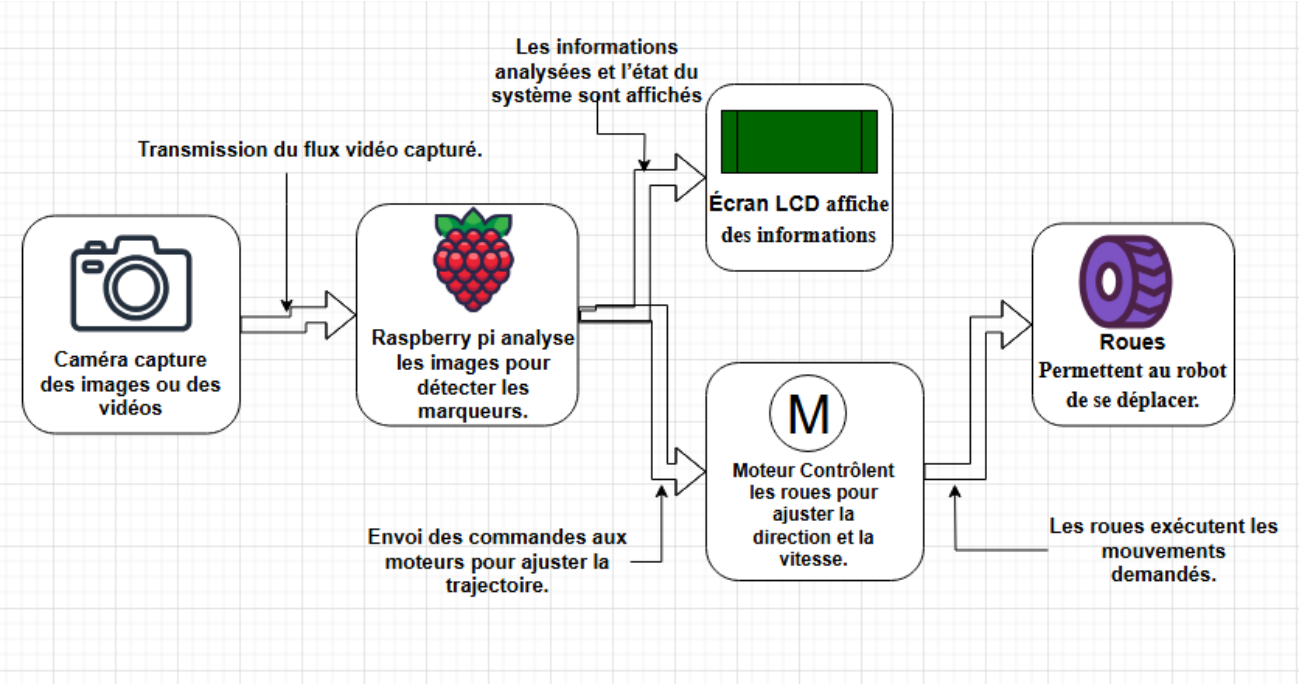


Figure 2.15 : Schéma général du système.

Explication :

La caméra, positionnée en hauteur, capture des images ou des vidéos qui sont transmises au Raspberry Pi via une interface CSI ou USB. Le Raspberry Pi analyse ces données visuelles à l'aide de bibliothèques comme OpenCV pour détecter les marqueurs visuels. En parallèle, des capteurs infrarouges (IR) et à ultrasons (HC-SR04), connectés aux broches GPIO du Raspberry Pi, détectent les obstacles et mesurent la distance par rapport aux objets environnants. Les moteurs, qu'ils soient à courant continu (DC) ou servomoteurs, assurent la mobilité du système. Leur contrôle est assuré par le Raspberry Pi via un contrôleur de moteur, tel que le L298N, permettant de gérer à la fois la vitesse et la direction. En combinant les informations provenant des capteurs de distance et des marqueurs visuels, le Raspberry Pi ajuste les mouvements du système pour atteindre ses objectifs tout en évitant les obstacles.

2.4. Logiciels et bibliothèques

2.4.1. Présentation d'OpenCV et ArUco

- **OpenCV**

(Open Source Computer Vision Library) est une bibliothèque de logiciels de vision artificielle et d'apprentissage automatique. La bibliothèque dispose de plus de 2500 algorithmes optimisés qui comprennent un ensemble complet d'algorithmes de vision artificielle et d'apprentissage automatique classiques et à la pointe de la technologie. Ces algorithmes peuvent être utilisés pour détecter et reconnaitre (des visages, identifier des objets, classer des actions humaines …etc) Opencv est une librairie open source développée en C/C++, elle a été conçu pour l'efficacité de calcul et avec un fort accent sur les applications en temps réel, elle dispose d'interfaces C ++, Python et Java et prend en charge Windows, Linux, Mac OS, iOS et Android.

- **ArUco :**

L'algorithme de détection a été implémenté en utilisant la bibliothèque OpenCV, car elle offre un large ensemble d'algorithmes de traitement d'images. La figure 2.16 montre les étapes appliquées pour détecter et identifier les marqueurs.

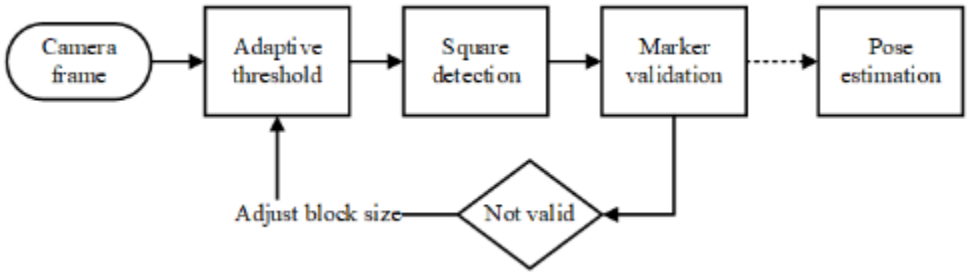


Figure 2.16 : Les étapes appliquées pour détecter et identifier les marqueurs [17].

L'algorithme commence par appliquer un seuillage adaptatif à l'image 2.9. Cet algorithme consiste à calculer, pour chaque pixel, une valeur de seuil en utilisant l'histogramme de son voisinage. Il est particulièrement utile dans des situations avec des conditions d'éclairage variées.

Figure 2.17: Un seuillage adaptatif [17]**.**

Pour déterminer la taille du bloc de seuil (taille du voisinage), une taille de bloc est testée sur chaque image. La taille de bloc choisie est la moyenne des tailles de bloc pour lesquelles le nombre maximum de marqueurs a été détecté. La taille du bloc est retestée lorsque aucun marqueur n'est visible.

Après avoir appliqué le seuillage à l'image, la détection de carrés est effectuée : les contours sont détectés à l'aide d'un algorithme de suivi de bordures, suivi de l'algorithme de simplification de contours de Douglas-Peucker. À partir des contours détectés, la Conjecture de la Somme des Angles d'un Quadrilatère est utilisée comme critère pour détecter les carrés. Même sous une distorsion perspective significative, un carré reste toujours un quadrilatère convexe.

Notre second critère consiste à vérifier que la somme des cosinus de tous les angles internes est inférieure à un seuil défini. Pour filtrer le bruit, un troisième critère a été ajouté : tous les contours formant une géométrie avec une aire inférieure à un seuil défini seront écartés.

Ces trois critères permettent de filtrer correctement les carrés, même en cas de forte distorsion, à partir de la liste des contours. La figure 2.17 représente le résultat obtenu pour une somme maximale des cosinus de 0,25 et une aire minimale de 100 pixels [17].

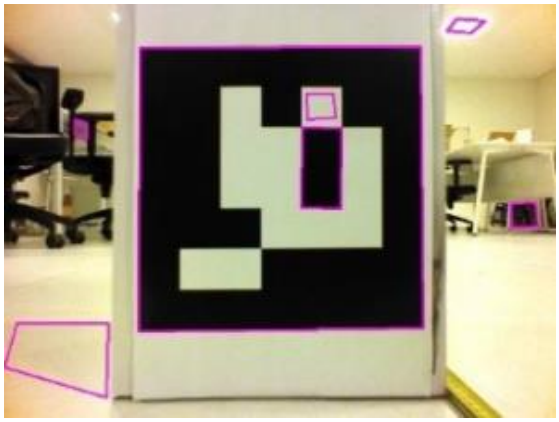


Figure 2.18 : La matrice obtenue après le processus de binarisation [17]**.**

La distorsion de perspective est corrigée dans les carrés détectés, puis ils sont rééchantillonnés en une matrice 7x7 à l'aide d'une interpolation linéaire. Un seuillage est appliqué en utilisant l'algorithme de binarisation d'Otsu. À ce stade, nous obtenons une matrice contenant les données du marqueur. La figure 2.18 représente la matrice obtenue après le processus de binarisation.

Figure 2.19 : La distorsion de perspective [17]**.**

À ce stade, les données du marqueur sont validées en tant que marqueur ArUco à l'aide de la matrice de signature. Les marqueurs peuvent être détectés dans n'importe quelle orientation. L'algorithme teste les données avec différentes rotations (90º, 180º, 270º). Si le marqueur n'est reconnu dans aucune rotation, il est alors écarté.

Pour l'estimation de pose, la méthode **solvePnp** d'OpenCV a été utilisée, en mode itératif avec l'optimisation de Levenberg-Marquardt. Pour obtenir la position de la caméra, les marqueurs doivent être enregistrés dans le programme. Un marqueur est représenté par son identifiant et une pose réelle (position et rotation). Les coins obtenus à partir de tous les marqueurs visibles connus sont utilisés pour estimer la pose de la caméra [17].

- **Types de dictionnaires ArUco dans OpenCV :**

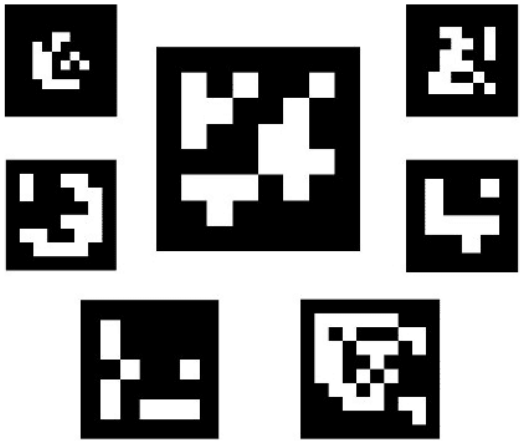


Figure 2.20 : Exemples de balises ArUco générées par OpenCV [17].

Il y a 21 dictionnaires ArUco différents intégrés dans la bibliothèque OpenCV. Je les ai listés ici dans le dictionnaire Python suivant :

```
ARUCO_DICT = {
"DICT_4X4_50": cv2.aruco.DICT_4X4_50,
"DICT_4X4_100": cv2.aruco.DICT_4X4_100,
"DICT_4X4_250": cv2.aruco.DICT_4X4_250,
"DICT_4X4_1000": cv2.aruco.DICT_4X4_1000,
"DICT_5X5_50": cv2.aruco.DICT_5X5_50,
"DICT_5X5_100": cv2.aruco.DICT_5X5_100,
"DICT_5X5_250": cv2.aruco.DICT_5X5_250,
"DICT_5X5_1000": cv2.aruco.DICT_5X5_1000,
"DICT_6X6_50": cv2.aruco.DICT_6X6_50,
"DICT_6X6_100": cv2.aruco.DICT_6X6_100,
"DICT_6X6_250": cv2.aruco.DICT_6X6_250,
"DICT_6X6_1000": cv2.aruco.DICT_6X6_1000,
"DICT_7X7_50": cv2.aruco.DICT_7X7_50,
"DICT_7X7_100": cv2.aruco.DICT_7X7_100,
"DICT_7X7_250": cv2.aruco.DICT_7X7_250,
"DICT_7X7_1000": cv2.aruco.DICT_7X7_1000,
"DICT_ARUCO_ORIGINAL": cv2.aruco.DICT_ARUCO_ORIGINAL,
"DICT_APRILTAG_16h5": cv2.aruco.DICT_APRILTAG_16h5,
"DICT_APRILTAG_25h9": cv2.aruco.DICT_APRILTAG_25h9,
"DICT_APRILTAG_36h10": cv2.aruco.DICT_APRILTAG_36h10,
"DICT_APRILTAG_36h11": cv2.aruco.DICT_APRILTAG_36h11
}
```

- **Seuil adaptatif (adaptive threshold) :**

Le seuillage :

Le **seuillage** est une technique en traitement d'images qui permet de distinguer différentes régions en convertissant des images en niveaux de gris en images binaires.

Essentiellement, il s'agit de définir un point de coupure — appelé le seuil — et tout ce qui est au-dessus de ce seuil devient blanc, tandis que tout ce qui est en dessous devient noir.

Par exemple, si vous travaillez sur la détection de texte dans un document numérisé, le seuillage peut vous aider à isoler le texte du fond en rendant le texte noir et le fond blanc.

C'est un outil simple mais puissant, particulièrement utile lorsque vous devez simplifier une image pour une analyse ultérieure [28].

Le seuil adaptatif :

Vous avez probablement rencontré des situations où le seuil global ne suffit pas, surtout lorsque vos images présentent des conditions d'éclairage variables ou des contrastes inégaux. C'est ici que le seuil adaptatif devient votre meilleur allié.

Le seuil adaptatif, c'est un peu comme offrir à votre image un costume sur mesure plutôt qu'un vêtement standard qui s'adapte à tous. Au lieu d'appliquer un seuil unique sur toute l'image,

le seuil adaptatif divise l'image en petites régions et calcule le seuil optimal pour chacune en fonction des valeurs locales des pixels.

Cette approche localisée garantit que vous capturez les détails à travers toute l'image, même lorsque l'éclairage est incohérent.

Par exemple, imaginez que vous travaillez avec une photographie ancienne où la moitié supérieure est bien éclairée, mais la moitié inférieure est ombragée. Un seuil global pourrait laisser une moitié supérieure délavée et une moitié inférieure trop sombre.

Mais avec le seuil adaptatif, la méthode s'ajuste pour chaque région en fonction de son éclairage, vous offrant une image binaire équilibrée et détaillée [28].

Si vous avez du mal avec le seuillage global et le trouvez insuffisant pour des images complexes, le seuillage adaptatif est votre solution.

Il est particulièrement utile dans les situations où l'éclairage ou le contraste varie considérablement à travers l'image, garantissant que vous ne manquez aucun détail important.

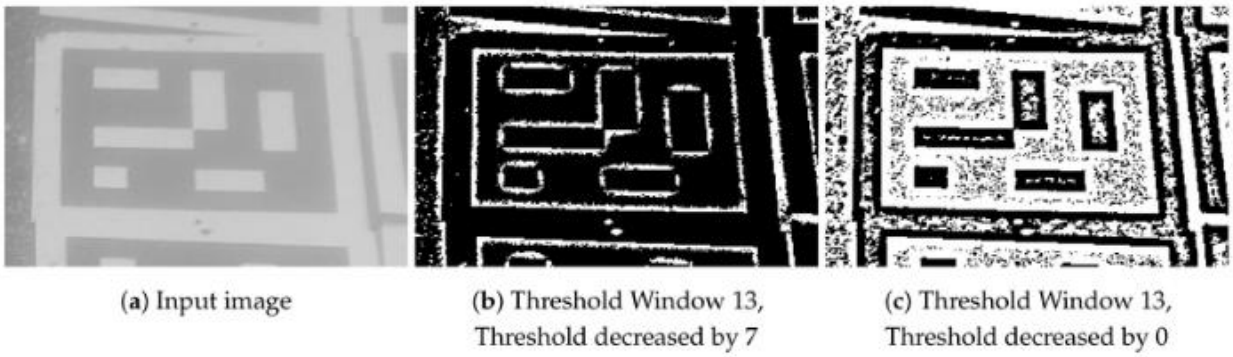


Figure 2.21 : Le seuil adaptatif [28]**.**

(a) montre un marqueur enregistré dans des conditions de faible visibilité. ARUco utilise un seuillage adaptatif et compare les pixels à un seuil diminué de sept (voir (b)) ; remarquez une bordure discontinue du marqueur. La version de base de UWARUco ne diminue pas le seuil (voir (c)), ce qui permet de préserver la bordure, mais introduit de nombreux petits objets.

- **Détection de carrés (square detection) :**

La détection de carrés est le processus, en traitement d'images, qui consiste à identifier des objets ou des régions de forme carrée dans une image. Cela implique d'analyser l'image pour trouver des zones ayant quatre côtés de longueurs à peu près égales et quatre angles droits, indiquant une forme carrée ou rectangulaire.
La détection des carrés est importante :

- **Numérisation de documents :** Pour trouver les bords des pages afin de les recadrer ou de corriger la perspective.
- **Réalité augmentée (RA) :** Pour détecter des marqueurs (comme les marqueurs ARUco) souvent de forme carrée.
- **Reconnaissance d'objets :** Pour identifier des objets carrés dans des images, utile en robotique ou dans l'industrie.

La détection de carrés fonctionne de la manière suivante :

- **Convertir l'image en niveaux de gris et réduire le bruit** pour simplifier l'analyse.
- **Détecter les contours** à l'aide de méthodes comme l'algorithme de Canny.
- **Trouver les contours** dans l'image obtenue.
- **Approximer chaque contour par un polygone** et vérifier s'il a quatre côtés.
- **Valider la forme** en contrôlant les longueurs des côtés et les angles pour confirmer qu'il s'agit bien d'un carré ou d'un rectangle.

La détection de carrés permet d'isoler et d'identifier des formes carrées dans une image, ce qui est essentiel pour de nombreuses tâches de vision par ordinateur impliquant la reconnaissance ou la localisation de formes.

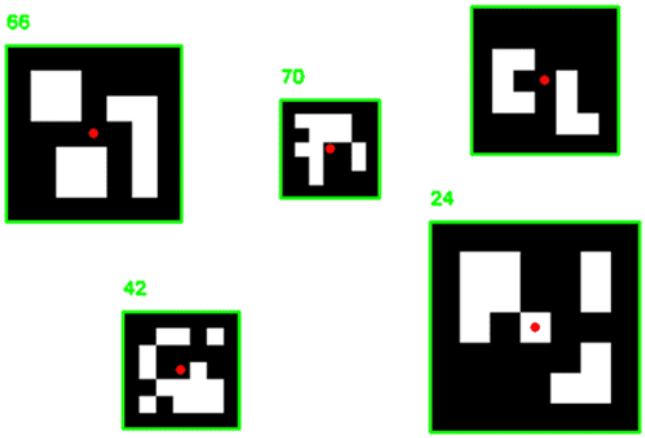


Figure 2.22 : La détection de carrés.

➢ **Validation de marqueur (marker validation) :**

La validation de marqueurs consiste à vérifier si un marqueur détecté (par exemple, un marqueur ARUco, un code QR ou tout autre marqueur fiduciaire) répond à certains critères pour être considéré comme valide et utilisable dans une application spécifique. Cela est particulièrement important dans les tâches de vision par ordinateur et de réalité augmentée, où les marqueurs servent de points de référence pour l'alignement, l'orientation ou l'identification.

Étapes de la validation de marqueurs

1. **Détection** :
    - Détecter les marqueurs potentiels dans l'image à l'aide d'un algorithme spécifique (par exemple, la détection de marqueurs ARUco).
    - Récupérer des informations comme la position, la taille et l'identifiant unique du marqueur.
2. **Critères de validation** :
    - **Taille** : Vérifier si le marqueur détecté est d'une taille acceptable (ni trop petit ni trop grand par rapport à l'image).
    - **Forme** : S'assurer que la forme du marqueur correspond aux attentes (par exemple, carré ou rectangulaire avec des bords droits).
    - **Qualité des contours** : Vérifier que les bords du marqueur sont continus et bien définis.
    - **Encodage** : Confirmer que le marqueur contient des informations encodées valides (par exemple, un identifiant ou des données lisibles).
    - **Orientation** : Valider que le marqueur n'est pas déformé ou trop incliné.
3. **Vérifications liées au bruit et à l'environnement** :
    - Vérifier la présence de bruit ou d'obstructions autour du marqueur pouvant affecter sa lisibilité.
    - S'assurer que le marqueur n'est pas partiellement masqué.
4. **Rejet des marqueurs invalides** :
    - Si un marqueur ne répond pas aux critères, il est rejeté ou marqué comme invalide.

- **Perspective-n-point (pnp) (pose estimation) :**

Le problème "Perspective-n-Point" (PnP) est une problématique classique en vision par ordinateur et en géométrie 3D. Il consiste à déterminer la position et l'orientation (pose) d'une caméra par rapport à un objet ou une scène en 3D, à partir de plusieurs correspondances entre des points 3D connus (dans un référentiel fixe) et leurs projections 2D sur l'image capturée par la caméra.

Le problème de Perspective-n-Point (PnP) consiste à estimer la pose d'une caméra calibrée en utilisant un ensemble de n points 3D dans le monde réel et leurs projections 2D correspondantes dans l'image. La pose de la caméra comprend 6 degrés de liberté (DOF), constitués de la rotation (roulis, tangage et lacet) et de la translation 3D de la caméra par rapport au monde. Ce problème trouve son origine dans la calibration de caméra et a de nombreuses applications en vision par ordinateur et dans d'autres domaines, notamment l'estimation de pose 3D, la robotique et la réalité augmentée. Une solution fréquemment utilisée pour ce problème existe pour $n = 3$, appelée P3P, et de nombreuses solutions sont disponibles pour le cas général où $n \geq 3$. Une solution pour $n = 2$ est également possible si les orientations des caractéristiques sont disponibles aux deux points.

Des implémentations de ces solutions sont également accessibles dans des logiciels open source [23].

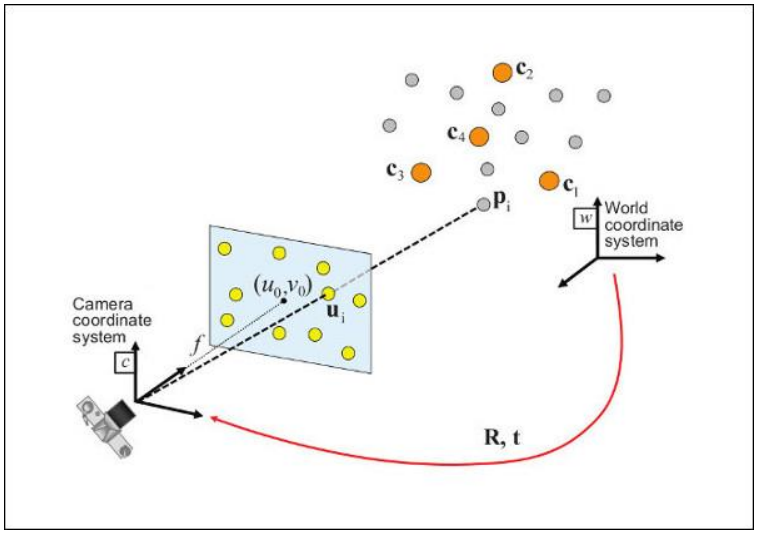


Figure 2.23 : méthode de Perspective-n-point [23]**.**

Détails du problème :

- **Entrées :**
  1. Un ensemble de points 3D dans le monde réel (souvent appelé référentiel monde).
  2. Les coordonnées 2D correspondantes de ces points dans l'image.
  3. Les paramètres intrinsèques de la caméra (matrice intrinsèque décrivant des propriétés comme la focale et le centre optique).
- **Sorties :**
  1. La matrice de rotation (R).
  2. Le vecteur de translation (T).

  Ces deux éléments combinés (R, T) forment la pose de la caméra, permettant de convertir des points 3D du monde en points dans le système de la caméra.

Méthodes de résolution :

- **Méthodes analytiques :** Comme le SolvePnP dans OpenCV, utilisant des algorithmes comme le P3P (Perspective-3-Point) pour le cas de 3 points, ou des algorithmes itératifs pour un plus grand nombre de points.
- **Méthodes robustes :** Pour minimiser l'impact des erreurs et des correspondances bruitées, on peut utiliser RANSAC en combinaison avec PnP.

Le PnP est un élément fondamental pour de nombreux systèmes intégrant de la vision par ordinateur et est largement utilisé dans l'industrie et la recherche.

2.4.2. Environnements et outils utilisés :

- **Matlab 2024a :**

Nous avons utilisé la version matlab 2024a car elle introduit la fonction "readArucoMarker" qui permet de détecter et lire les marqueurs aruco.

- **Visual Studio Code** :

Est un éditeur de code extensible développé par Microsoft pour Windows, Linux et macOS.

Les fonctionnalités incluent la prise en charge du débogage, la mise en évidence de la syntaxe, la complétion intelligente du code (IntelliSense.), les snippets, la refactorisation du code et Git intégré. Les utilisateurs peuvent modifier le thème, les raccourcis clavier, les préférences et installer des extensions qui ajoutent des fonctionnalités supplémentaires.

Le code source de Visual Studio Code provient du projet logiciel libre et open source VS Code de Microsoft publié sous la licence MIT permissive, mais les binaires compilés constituent un freeware, c'est-à-dire un logiciel gratuit pour toute utilisation mais propriétaire.

- **Python :**

Python est un langage de programmation interprété, multiparadigme et multiplateformes. Il favorise la programmation impérative structurée, fonctionnelle et orientée objet. Il est doté d'un typage dynamique fort, d'une gestion automatique de la mémoire par ramasse-miettes et d'un système de gestion d'exceptions ; il est ainsi similair à Perl, Ruby, Scheme, Smalltalk et Tcl. Nous avons utilisé Python version 3.13 dans le Visual Studio code.

2.5. Algorithmes et Modèles

2.5.1. Schémas et branchement globale réalisée :

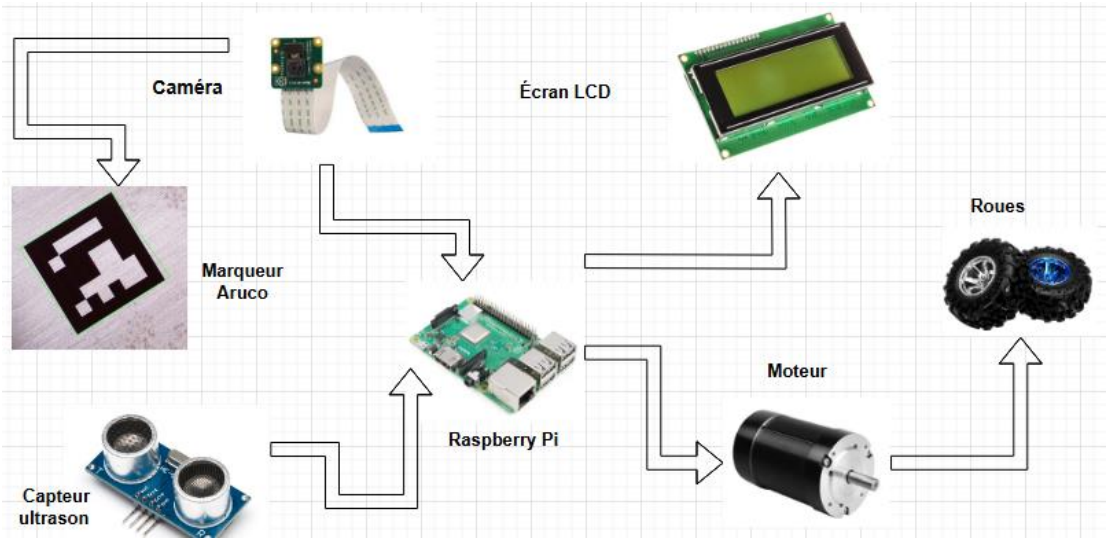


Figure 2.24 : Schéma et branchement globale réalisée.

La figure 2.24 : présente les schémas et le branchement global réalisés pour le système. Elle illustre l'ensemble des connexions entre les différents composants matériels, notamment les

capteurs, actionneurs, modules de communication, et l'unité de commande centrale (comme un microcontrôleur ou un micro-ordinateur). Chaque élément est relié de manière à assurer une transmission correcte des données et de l'alimentation électrique. Le schéma global permet d'avoir une vue d'ensemble du fonctionnement du système, en mettant en évidence l'organisation logique des flux d'informations et d'énergie nécessaires à l'exécution des tâches. Cette configuration a été soigneusement conçue pour garantir la compatibilité entre les composants, optimiser l'efficacité du système et faciliter la maintenance ou les ajustements futurs.

2.5.2. Installation des différents environnements de développements

Pour développer un système autonome contrôlé par des marqueurs visuels utilisant un Raspberry Pi, plusieurs outils et environnements doivent être configurés. Voici les étapes nécessaires pour l'installation des différents environnements de développement :

- **Configuration du Raspberry Pi :**

L'installation du système d'exploitation sur le Raspberry Pi a débuté par le téléchargement et l'installation de Raspberry Pi OS à l'aide de l'outil Raspberry Pi Imager. Une fois l'installation terminée, les paramètres initiaux ont été configurés pour permettre une utilisation optimale. Cela inclut la configuration du réseau pour une connexion stable, l'activation de SSH pour un accès à distance sécurisé et la mise en place de VNC pour une interface graphique accessible à distance. Enfin, les bibliothèques essentielles nécessaires à la gestion des capteurs et des caméras ont été installées, préparant ainsi le Raspberry Pi pour les tâches de traitement et de contrôle liées au système autonome.

- **Configuration de l'environnement Python :**

L'installation de Python sur le Raspberry Pi a commencé par une vérification que Python 3, généralement préinstallé sur la plupart des appareils Raspberry Pi, était disponible et fonctionnel. Ensuite, un environnement virtuel isolé a été créé spécifiquement pour le projet, afin d'éviter tout conflit entre les différentes dépendances nécessaires. Une fois cet environnement configuré, les bibliothèques essentielles ont été installées, notamment OpenCV pour les opérations de vision par ordinateur et NumPy pour les calculs numériques. Ces outils permettent une gestion efficace des tâches de traitement d'image et des calculs nécessaires au fonctionnement du système autonome.

- **Configuration de l'environnement de développement intégré (IDE) :**

Un environnement de développement intégré (IDE) tel que VS Code ou Thonny a été installé pour faciliter l'écriture, le test et le débogage du code Python. Afin d'optimiser l'utilisation de ces outils, des extensions utiles ont été ajoutées, notamment l'extension Python, qui permet une gestion efficace des environnements virtuels ainsi que l'exécution simplifiée des

scripts. Cette configuration offre un cadre de travail performant pour le développement du système autonome.

- **Calibration du Caméra :**

La caméra a été connectée au Raspberry Pi en utilisant l'interface CSI ou USB, selon le type de matériel disponible. Pour s'assurer de son bon fonctionnement, des tests ont été réalisés à l'aide de commandes spécifiques telles que raspistill, qui permet de capturer des images, ou via des scripts utilisant la bibliothèque OpenCV pour vérifier la réception et le traitement des flux vidéo en temps réel. Ces étapes garantissent que la caméra est correctement configurée et opérationnelle pour les applications de vision.

2.5.3. Détection et identification des marqueurs

La détection et l'identification des marqueurs visuels représentent une étape fondamentale pour le bon fonctionnement du système autonome. Elles permettent au Raspberry Pi d'interagir avec son environnement en localisant des repères visuels en temps réel. Pour cela, le système utilise un dictionnaire ARUco prédéfini, tel que *DICT_4x4_50*, qui contient 50 motifs uniques organisés en une grille de 4x4. Ce dictionnaire est chargé à l'aide de la bibliothèque OpenCV en Python. La caméra, connectée au Raspberry Pi, capture en continu des images qui sont converties en niveaux de gris afin de faciliter le traitement. La détection des marqueurs est réalisée à l'aide de la fonction aruco.detectMarkers(), qui identifie les coins des marqueurs, leur identifiant, et rejette les éléments non reconnus. Une fois détectés, les marqueurs sont encadrés graphiquement et leurs identifiants sont affichés dans la console. Grâce aux coordonnées des coins des marqueurs et aux paramètres de calibration de la caméra, il est possible de calculer leur position et leur distance. L'ensemble du processus est exécuté en temps réel, assurant une détection continue et fiable pendant le déplacement du système.

2.5.4. Suivi et localisation des marqueurs

Le suivi et la localisation des marqueurs permettent de déterminer leur position dans l'espace 3D et de suivre leur évolution au fil du temps. Une fois les marqueurs détectés dans l'image, leurs coordonnées sont extraites à partir des coins, puis le centre de chaque marqueur est calculé par la moyenne de ces points. Cette information sert de base pour estimer leur position dans l'environnement. Pour obtenir une localisation en trois dimensions, la fonction aruco.estimatePoseSingleMarkers() convertit les coordonnées 2D en coordonnées 3D à l'aide des paramètres intrinsèques de la caméra et des coefficients de distorsion, préalablement obtenus par calibration. Cette étape permet de corriger les déformations optiques et d'obtenir une estimation fiable de la position et de l'orientation des marqueurs. La pose des marqueurs est ensuite visualisée en superposant des axes sur l'image via la fonction aruco.drawAxis(). Le système assure un suivi en temps réel en enregistrant les positions successives des marqueurs, ce qui permet d'analyser leurs mouvements, de calculer leur trajectoire, leur vitesse ou leur direction. Ce suivi joue un rôle essentiel dans la navigation autonome, en permettant au robot

d'ajuster ses déplacements en fonction des informations visuelles, d'éviter les obstacles ou d'interagir avec des éléments spécifiques. Pour garantir la précision du suivi, il est essentiel d'utiliser des marqueurs imprimés avec soin, de maintenir une bonne qualité d'image grâce à un éclairage adapté, et d'effectuer une calibration rigoureuse de la caméra.

2.5.5. Stratégies de navigation en fonction des marqueurs détectés

Les stratégies de navigation reposant sur les marqueurs visuels permettent au système autonome de se déplacer de manière intelligente et réactive. Lorsqu'un marqueur est détecté, ses coordonnées et son orientation sont exploitées pour planifier et ajuster la trajectoire du robot. Ces marqueurs peuvent jouer plusieurs rôles, notamment celui de balises directionnelles indiquant les chemins à suivre, de repères pour atteindre des cibles précises, ou encore de signaux avertissant la présence d'obstacles ou de zones interdites. Ainsi, le système est capable de modifier dynamiquement son itinéraire pour contourner un obstacle, ralentir en fonction de la distance mesurée ou réorienter son déplacement vers une nouvelle cible. Ces mécanismes permettent une prise de décision en temps réel, garantissant une navigation fluide, précise et adaptée aux changements de l'environnement. Les marqueurs ne sont donc pas de simples repères statiques, mais des éléments interactifs qui jouent un rôle central dans l'intelligence et l'autonomie du système.

2.6. Conclusion :

Dans ce chapitre, nous avons détaillé les différentes étapes et approches adoptées pour concevoir un système autonome contrôlé par des marqueurs visuels en utilisant un Raspberry Pi. Cette conception repose sur une intégration cohérente entre le matériel et le logiciel, permettant de répondre aux exigences fonctionnelles et techniques du projet.

Nous avons d'abord présenté les composants matériels essentiels, tels que la caméra, les capteurs de distance, et les moteurs, en soulignant leur rôle spécifique dans la réalisation des tâches. Ensuite, nous avons abordé les choix des outils logiciels, tels que Python, OpenCV, qui offrent des fonctionnalités robustes pour la vision par ordinateur, la détection des marqueurs, et la gestion des trajectoires.

La conception du système a également mis l'accent sur l'interconnexion entre les différents modules pour garantir une communication fluide et une synchronisation optimale entre les capteurs et les actionneurs. Des stratégies de navigation et d'évitement d'obstacles basées sur les marqueurs ont été développées pour assurer un fonctionnement autonome et fiable.

En conclusion, la conception du système constitue une base solide pour la phase d'implémentation et de tests. Elle offre une architecture claire et modulaire, facilitant l'intégration des différentes composantes et la réalisation des objectifs du projet. Dans les chapitres suivants, nous évaluerons les performances du système en conditions réelles et discuterons des résultats obtenus.

Chapitre 3 : Implémentations et performances

3. Chapitre 3 : Implémentations et performances

3.1. Introduction :

Ce chapitre traite de l'implémentation pratique et de l'évaluation des performances du système autonome contrôlé par des marqueurs visuels, développé à l'aide d'un Raspberry Pi. Après avoir défini les fondements théoriques et conceptuels dans les chapitres précédents, nous abordons ici les détails techniques de la mise en œuvre, les défis rencontrés et les solutions adoptées.

3.2. L'Objectif visé :

L'objectif principal de ce travail est la conception et le développement d'un système autonome de navigation, représenté dans la Figure 3.1. Ce système repose sur l'utilisation de marqueurs visuels (de type ArUco ou similaires) pour guider un robot mobile. Le cœur du système est un micro-ordinateur Raspberry Pi, choisi pour son faible coût, sa compacité et sa capacité à gérer des traitements en temps réel, en particulier dans les domaines de la vision par ordinateur et de la robotique. L'ambition de ce projet est de démontrer qu'il est possible de réaliser une solution intelligente, performante et accessible en combinant ces technologies.

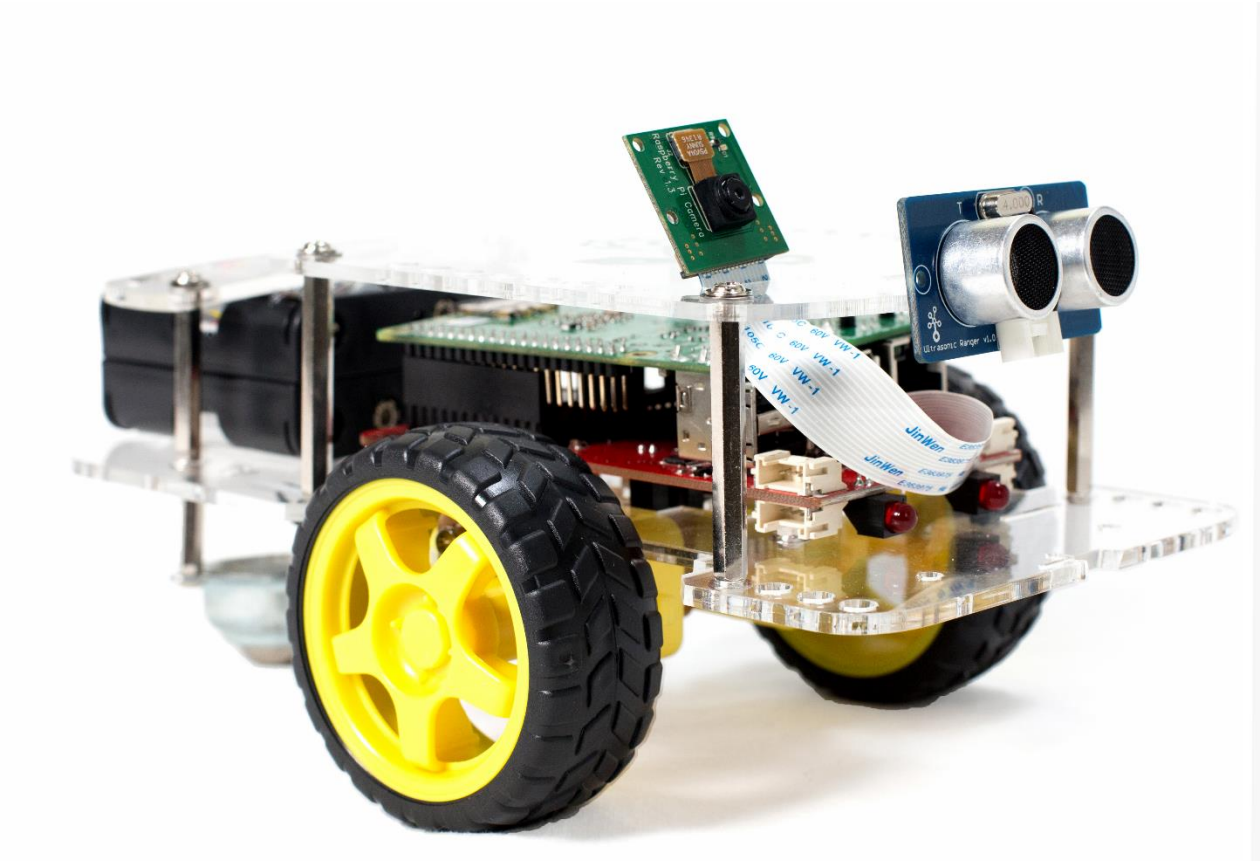


Figure 3.1 : Model de robot équipé d'une caméra et raspberry pi.

3.3. Programme de détection des marqueurs ARUco :

Nous avons utilisé Python et la bibliothèque OpenCV pour détecter des marqueurs ARUco, ainsi que pour déterminer leur distance et leur position.

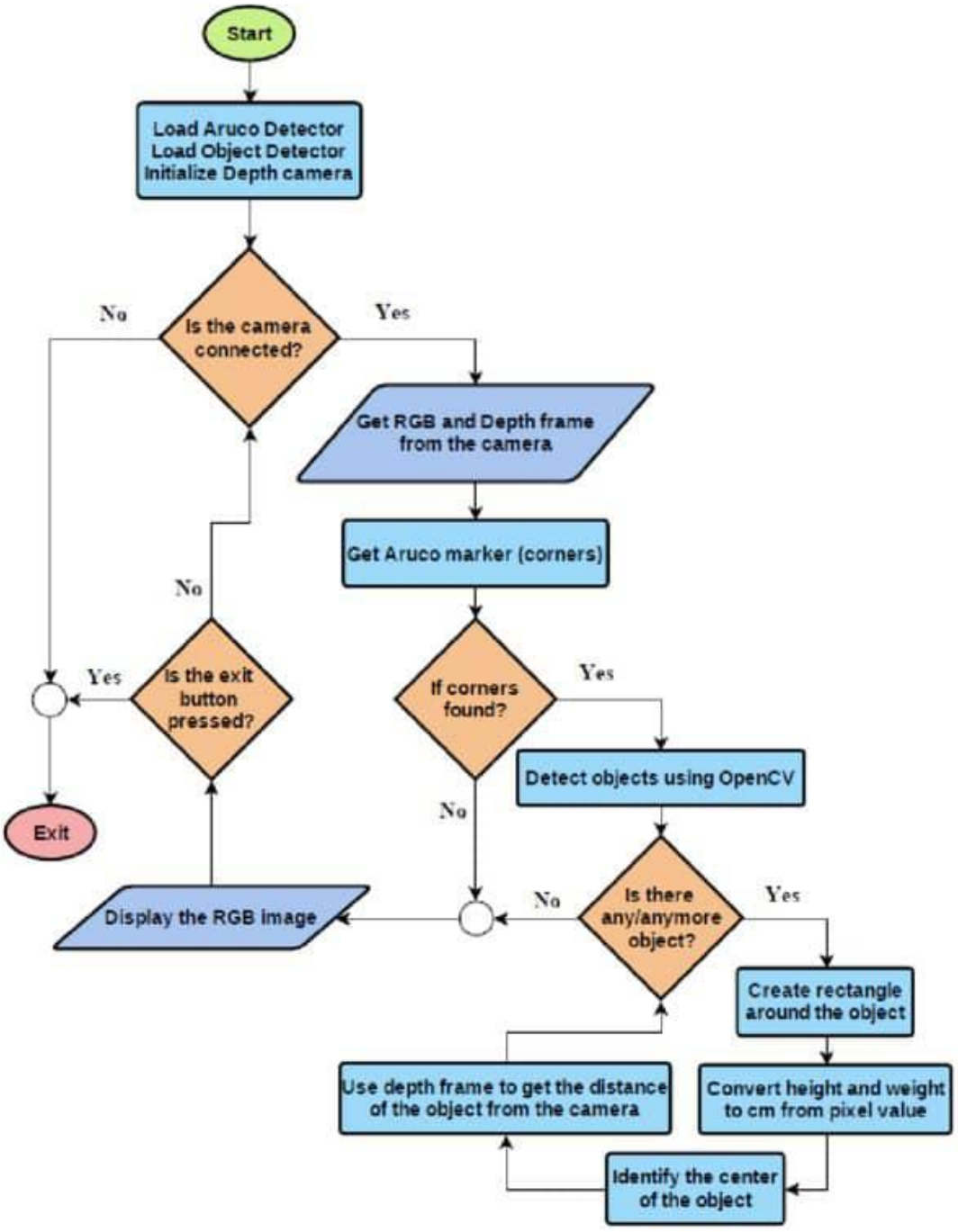


Figure 3.2 : organigramme des processus.

Voici le programme utilisé :

```python
import cv2
import numpy as np
import datetime

# Camera calibration data (in pixels)
camera_matrix = np.array([[927.84196598, 0, 366.26935095],
                          [0, 968.79451461, 642.64699993],
                          [0, 0, 1]])
dist_coeffs = np.array([0.20854384, -0.43923496, 0.01763289, 0.00301464, 0.35367725])

# ArUco setup
aruco_dict = cv2.aruco.getPredefinedDictionary(cv2.aruco.DICT_4X4_50)
aruco_params = cv2.aruco.DetectorParameters()
detector = cv2.aruco.ArucoDetector(aruco_dict, aruco_params)
marker_length_cm = 4.8

# 3D points of the marker corners in marker coordinate system (cm)
marker_corners_3d = np.array([
    [-marker_length_cm/2, marker_length_cm/2, 0],
    [marker_length_cm/2, marker_length_cm/2, 0],
    [marker_length_cm/2, -marker_length_cm/2, 0],
    [-marker_length_cm/2, -marker_length_cm/2, 0]
])

cap = cv2.VideoCapture(0)
if not cap.isOpened():
    print("Camera failed to open.")
    exit()
```

Figure 3.3 : Programme de détection des marqueurs ARUco partie 1

```
# video properties
frame_width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
frame_height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
fps = cap.get(cv2.CAP_PROP_FPS)
if fps == 0: fps = 30

# Define video codec and create VideoWriter object
filename = f"aruco_output_{datetime.datetime.now().strftime('%Y%m%d_%H%M%S')}.avi"
fourcc = cv2.VideoWriter_fourcc(*'XVID')
out = cv2.VideoWriter(filename, fourcc, fps, (frame_width, frame_height))

while True:
    ret, frame = cap.read()
    if not ret:
        break

    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    corners, ids, _ = detector.detectMarkers(gray)

    if ids is not None:
        for i in range(len(ids)):
            image_points = corners[i].reshape(4, 2)
            success, rvec, tvec = cv2.solvePnP(
                marker_corners_3d,
                image_points,
                camera_matrix,
                dist_coeffs,
                flags=cv2.SOLVEPNP_ITERATIVE
            )
```

Figure 3.4: Programme de détection des marqueurs ARUco partie 2.

```
            if success:
                cv2.aruco.drawDetectedMarkers(frame, [corners[i]], ids[i])
                cv2.drawFrameAxes(frame, camera_matrix, dist_coeffs, rvec, tvec, 3)

                distance_cm = np.linalg.norm(tvec)
                top_left = tuple(image_points[0].astype(int))
                cv2.putText(frame, f"ID: {ids[i][0]} Dist: {distance_cm:.1f} cm",
                            top_left, cv2.FONT_HERSHEY_SIMPLEX, 0.6, (0, 255, 0), 2)

    #Write frame to output video
    out.write(frame)

    cv2.imshow("ArUco Marker Detection", frame)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

cap.release()
out.release()
cv2.destroyAllWindows()
```

Figure 3.5: Programme de détection des marqueurs ARUco partie 3

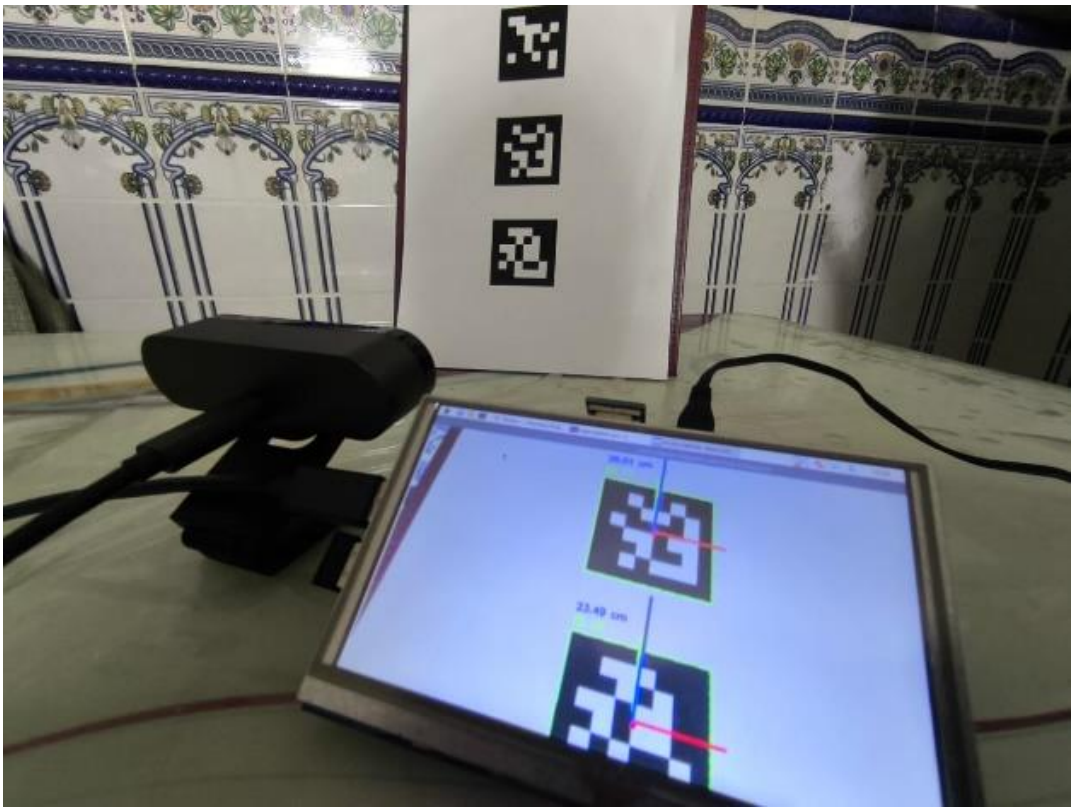
- **Explication :**

Nous avons utilisé un dictionnaire ARUco prédéfini, précisément le **DICT_4x4_50**, qui contient des motifs uniques de marqueurs organisés en une grille de 4x4 cellules. Les marqueurs sont détectés dans une image en niveaux de gris afin de simplifier le traitement. La fonction **aruco.detectMarkers()** permet d'identifier les coins des marqueurs détectés, leurs identifiants (IDs), ainsi qu'une liste des candidats rejetés. Les marqueurs identifiés sont encadrés par des boîtes, et leurs IDs sont affichés dans la console. Le programme capture des images en temps réel à partir de la caméra, permettant ainsi une détection continue des marqueurs.

3.4. Scénario simulé et logique de fonctionnement

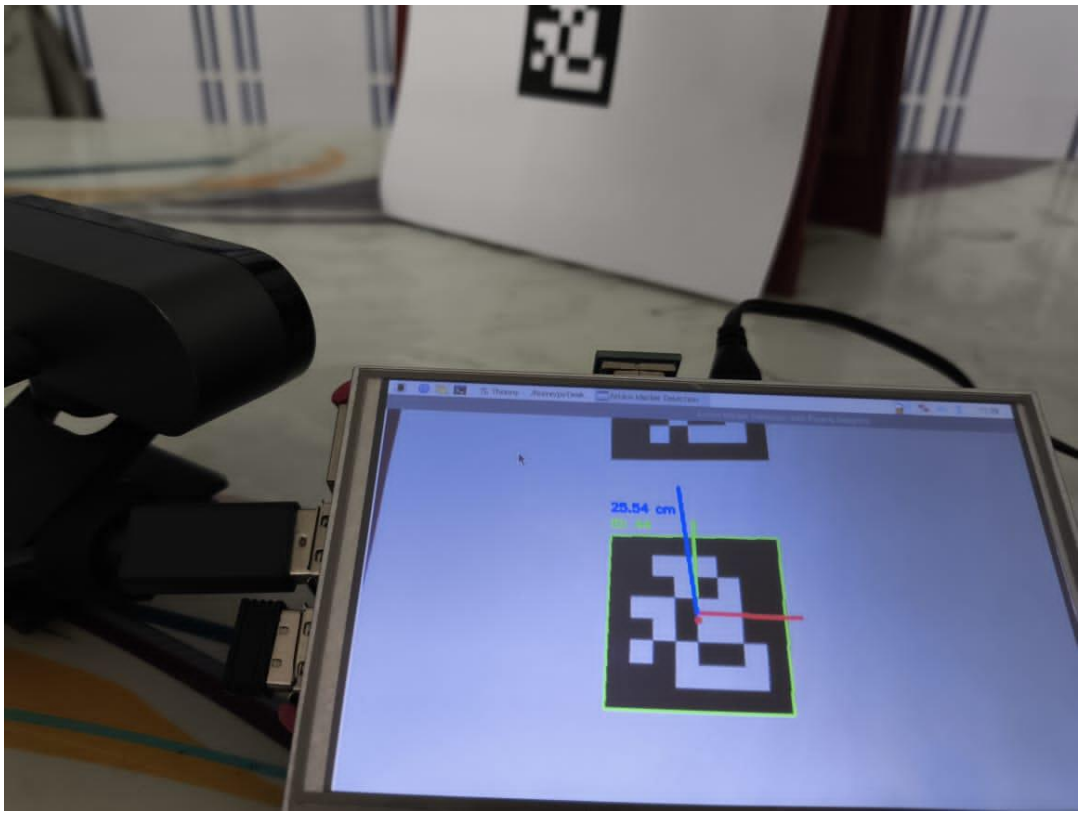
Afin de tester le système directement sur le robot, nous avons opté pour une approche alternative. Ainsi, les algorithmes de détection, de prise de décision et de navigation ont été testés à l'aide de vidéos capturées dans un environnement simulé. Ces vidéos reproduisent fidèlement les conditions réelles de fonctionnement du robot, en particulier la détection des marqueurs et l'évaluation des distances.

La figure 3.6 montre une démonstration de calcul de la distance entre la caméra et le marqueur ainsi que l'estimation de la pose illustrée en 3 axes sur le marquer.



(a)

(b)



(c)

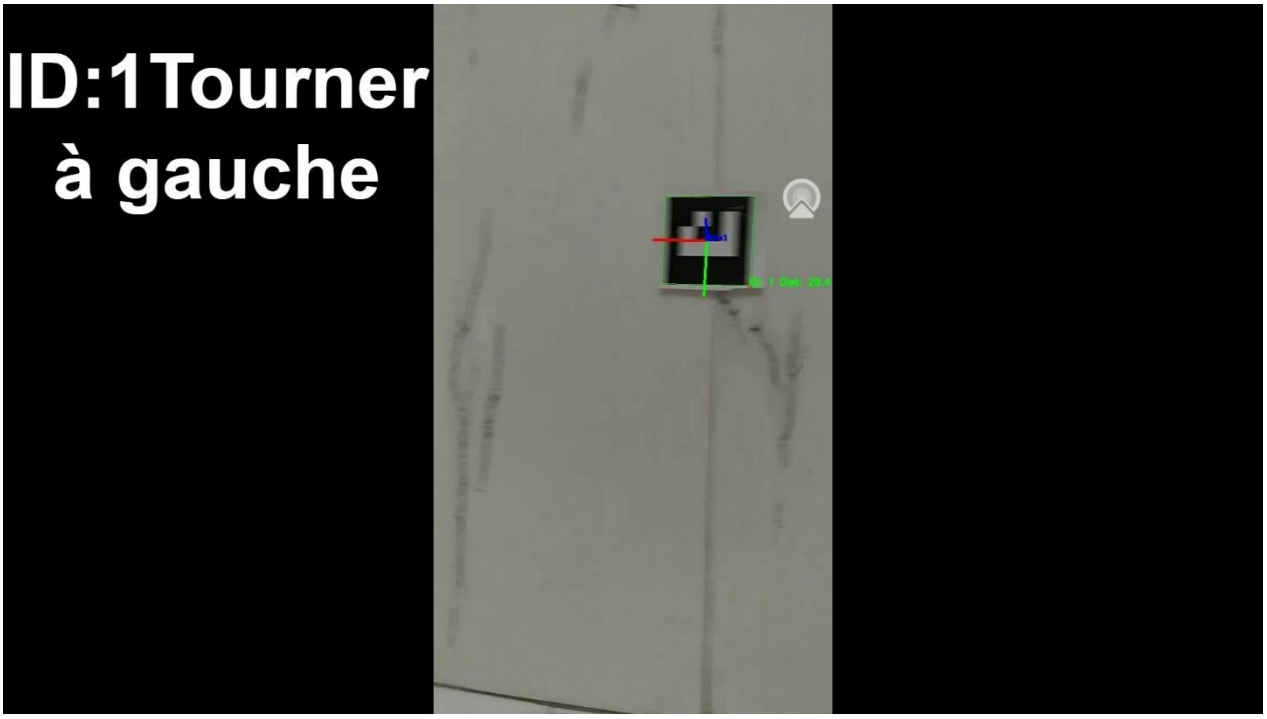
Figure 3.6 : (a) (b) (c) Estimation de pose et distance Raspberry Pi.

Le scénario mis en œuvre dans les vidéos suit une logique bien définie, structurée autour de trois marqueurs visuels (ID1, ID2 et ID3), chacun déclenchant une action spécifique de la part du système :

1. **Détection du premier marqueur (ID1) :**
   Lorsqu'un marqueur d'identifiant ID1 est détecté dans l'image capturée, le système interprète cela comme un signal de démarrage. Le robot (virtuel) commence alors à avancer.
   Pendant cette avancée, la distance entre le robot et l'obstacle situé en face est continuellement mesurée à l'aide de données visuelles simulées.
2. **Première action : tourner à gauche**
   Dès que la distance mesurée devient inférieure ou égale à 30 cm, une action de rotation vers la gauche est déclenchée. Cela simule une manœuvre d'évitement ou de changement de direction.
3. **Détection du second marqueur (ID2) :**
   Après le virage, le robot poursuit sa progression jusqu'à la détection du second marqueur ID2. Le comportement est similaire au précédent : le système continue d'avancer tout en surveillant la distance devant lui.
4. **Deuxième action : tourner à droite**
   Une fois la distance de 30 cm atteinte ou franchie, le robot simule cette fois un virage à droite, marquant un autre changement de trajectoire.
5. **Détection du troisième marqueur (ID3)**
   Le robot continue d'avancer après le virage, et lorsqu'il détecte le troisième marqueur ID3, la même logique de distance est appliquée.
6. **Arrêt final**
   Quand la distance devient à nouveau inférieure ou égale à 30 cm, le système interprète cela comme la fin du parcours : le robot s'arrête définitivement**.**

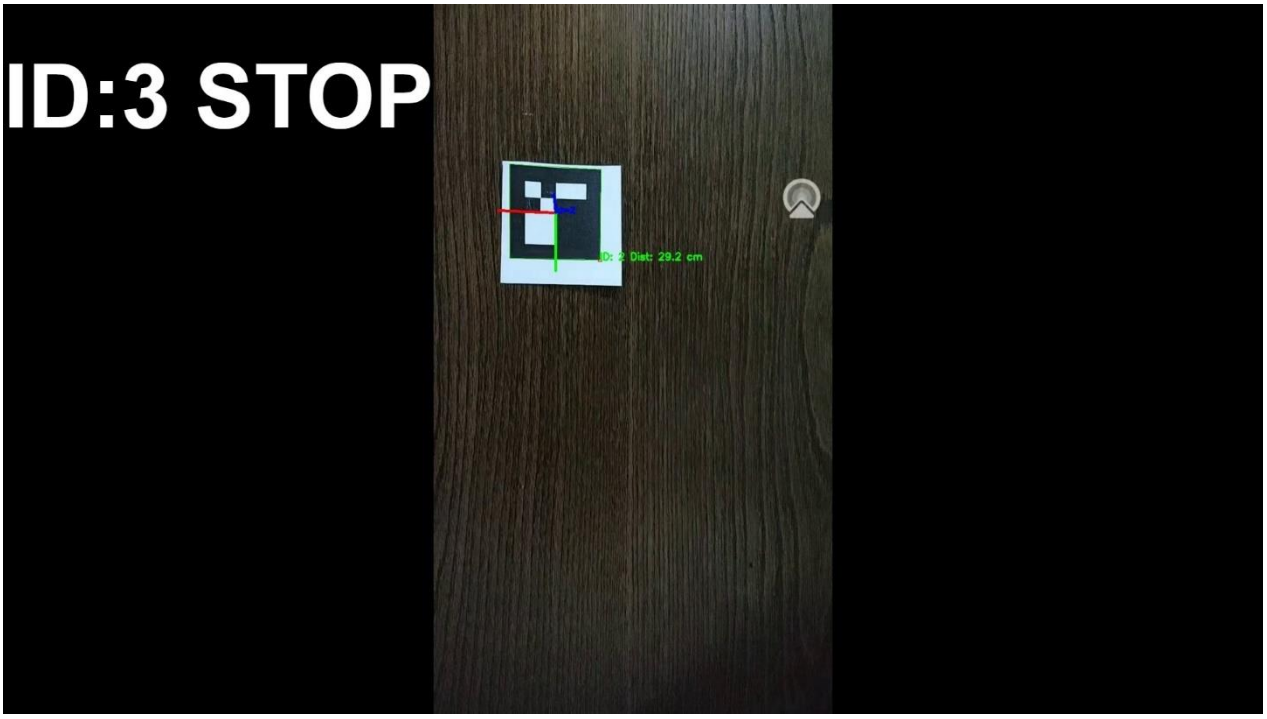
Les étapes précédentes sont illutstées dans la figure 3.7

(a)



(b)

(c)



(d)

Figure 3.7 : Le trajectoire d'ArUco suivi. (a) : avance jusqu'à dist <30. (b) Tourner à gauche. (c) Tourner à droite. (d) Stop.

3.5. Test de précision avec ArUco et AprilTag :

Dans le cadre de notre étude, en complément des simulations précédentes, nous avons réalisé une comparaison entre deux bibliothèques de détection de marqueurs visuels : ArUco et AprilTag. Cette comparaison vise à évaluer leur précision et leur robustesse dans le cadre d'applications robotiques, notamment pour le pilotage autonome de robots mobiles.

Pour ce faire, une vidéo de référence a été capturée dans des conditions contrôlées, puis volontairement dégradée selon différents niveaux de qualité (compression, bruit, réduction de résolution, etc.) afin de simuler des environnements réels et variés. Cette vidéo a été intégrée dans un premier programme chargé d'extraire automatiquement les images (frames) de la séquence. Les images ne contenant aucun marqueur visuel ont été filtrées et éliminées automatiquement.

Les images restantes, c'est-à-dire celles contenant effectivement un marqueur, ont ensuite été traitées par un second programme dédié à la détection, utilisant successivement les bibliothèques ArUco et AprilTag. Cela a permis de mesurer la précision de détection dans chaque cas, en tenant compte de plusieurs critères : taux de détection correct, sensibilité à la qualité de l'image, temps de traitement, et robustesse face aux perturbations visuelles.

Cette comparaison nous a permis de mieux comprendre les performances relatives de ces deux solutions dans un contexte pratique. Elle fournit des éléments clés pour guider le choix de la technologie la plus adaptée à des applications robotiques embarquées, où la fiabilité de la détection est cruciale.

3.5.1 Évaluation de la précision de détection selon la résolution vidéo (480p, 720p, 1080p) :

L'évaluation de la précision de détection des marqueurs visuels a été réalisée en traitant des vidéos de différentes résolutions : 480p, 720p et 1080p, à l'aide de scripts Python. Cette analyse permet de mesurer l'impact de la qualité d'image sur la fiabilité de la détection dans des conditions réalistes.

Le principe de la méthode repose sur l'extraction des images (frames) à partir des vidéos, puis sur la détection automatique des marqueurs présents. Pour chaque résolution, les étapes suivantes ont été appliquées :

- Extraction des images de la vidéo,
- Suppression automatique des images ne contenant aucun marqueur,
- Détection des marqueurs présents dans les images restantes,
- Calcul du taux de détection et du taux d'erreur.

Prenons l'exemple de la vidéo en résolution 480p :

- Nombre total d'images extraites : 992
- Images sans marqueur : 45
- Images avec marqueurs : 947
- Marqueurs non détectés : 6

Le taux d'erreur est alors calculé selon la formule suivante :

$$\boldsymbol{Erreur}\ (\%) = \frac{6 * 100}{959} = \mathbf{0,6336}\%$$

Le taux de précision est donc de :

$$100 - 0.63357 = 99.3664\%$$

La même méthode de calcul de précision a été appliquée aux autres cas de test, en variant la résolution vidéo et le logiciel utilisé (Python ou MATLAB), ainsi que la bibliothèque de détection (ArUco ou AprilTag). Les résultats obtenus sont regroupés dans les tableaux suivants.

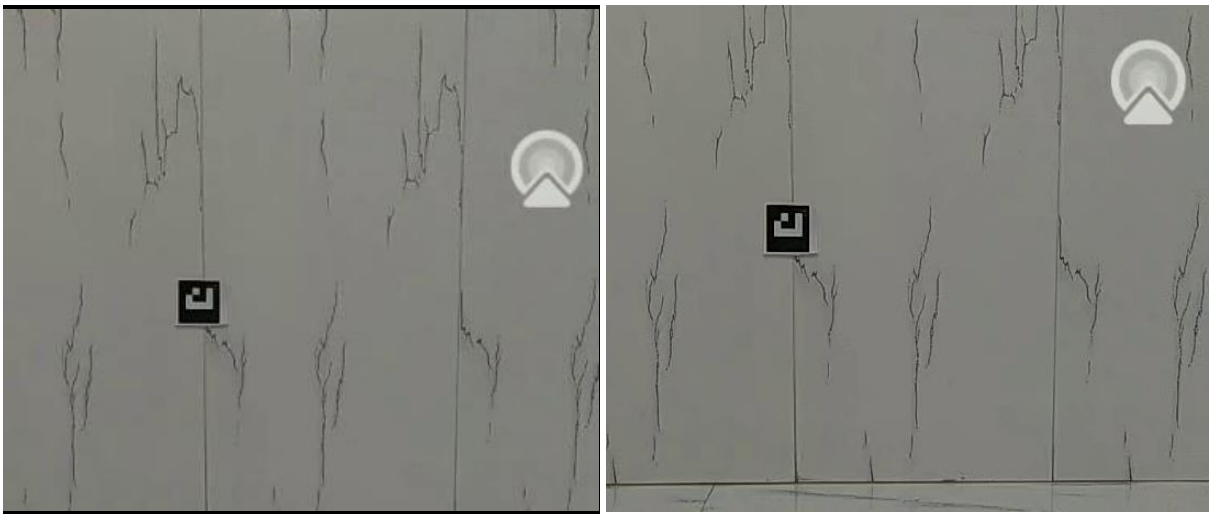
Software / Résolution	Python ArUco %	Python AprilTag %
480p	**99.36**	**99.31**
720p	**99.15**	**98.72**
1080p	**98.94**	**97.34**

Tableau 3.1 : Teste de précision dans le cas normal avec Python.

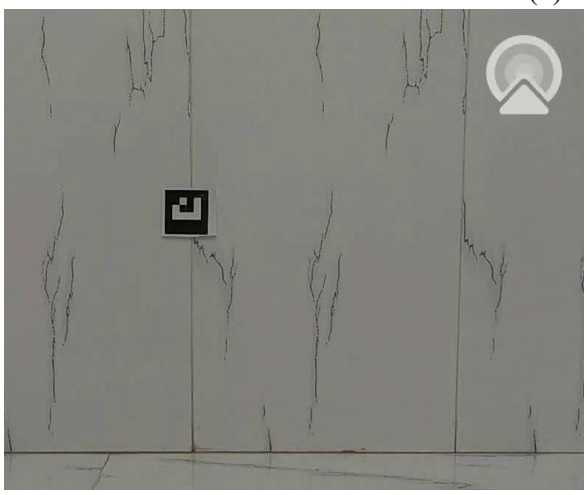
Software / Résolution	Matlab ArUco %	Matlab AprilTag %
480p	**99.68**	**99.21**
720p	**98.94**	**99.01**
1080p	**98.83**	**98.52**

Tableau 3.2 : Teste de précision dans le cas normal avec Matlab.

Ces résultats montrent que, de manière générale, la précision de détection reste très élevée, quelle que soit la résolution ou la bibliothèque utilisée. On note cependant une légère baisse de précision à mesure que la résolution augmente, en particulier pour AprilTag. MATLAB offre des performances comparables, voire légèrement supérieures à Python dans certains cas, notamment avec ArUco en basse résolution.

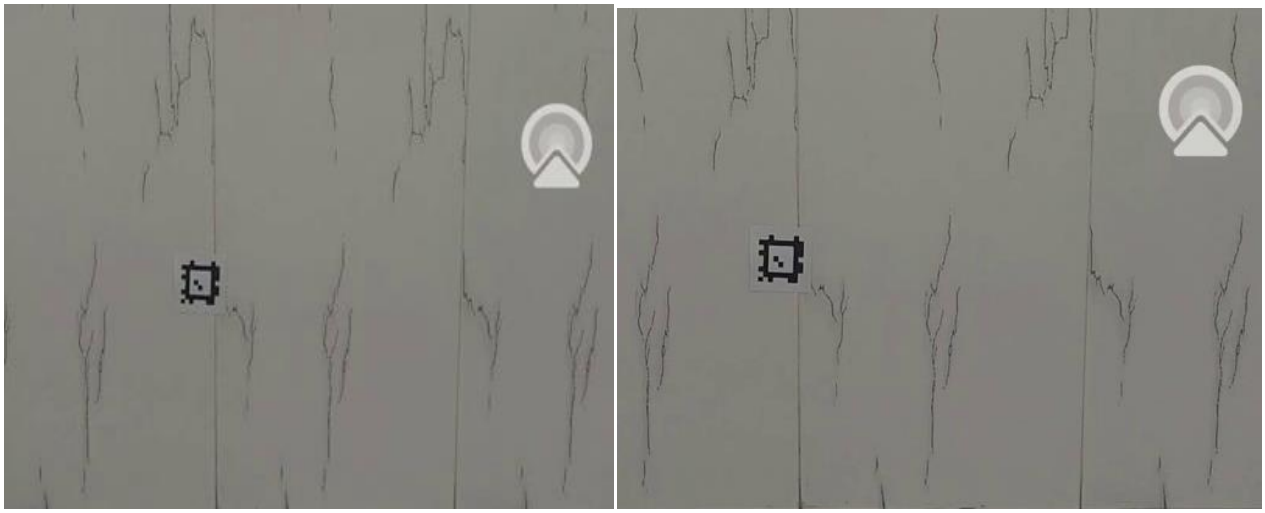


(a) (b)

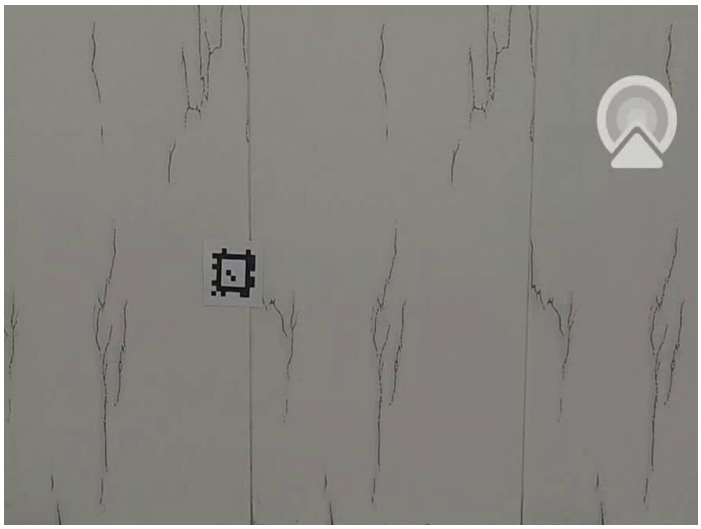


(c)

Figure 3.8: a) Frame 0 qualité 480p ArUco **b)** Frame 0 qualité 720p ArUco **c)** Frame 0 qualité 1080p ArUco



(a) (b)



(c)

Figure 3.9: a) Frame 0 qualité 480p AprilTag **b)** Frame 0 qualité 720p AprilTag **c)** Frame 0 qualité 1080p AprilTag

3.5.2 Comparaison En présence du bruit :

Dans des conditions normales, les bibliothèques **ArUco** et **AprilTag** offrent des performances de détection très similaires, que ce soit dans un environnement de développement Python ou MATLAB. Les différences de précision restent minimes et peu significatives dans un contexte sans perturbations.

Afin de pousser l'analyse plus loin et d'évaluer la robustesse des algorithmes face à des dégradations visuelles, nous avons appliqué un bruit gaussien artificiel aux images utilisées dans les tests précédents (pour les résolutions 480p, 720p et 1080p). Ce type de bruit simule des perturbations fréquentes dans des environnements réels, telles que de faibles conditions d'éclairage, des capteurs de qualité limitée ou des interférences optiques.

a. Applique un bruit d'image (Gaussian) :

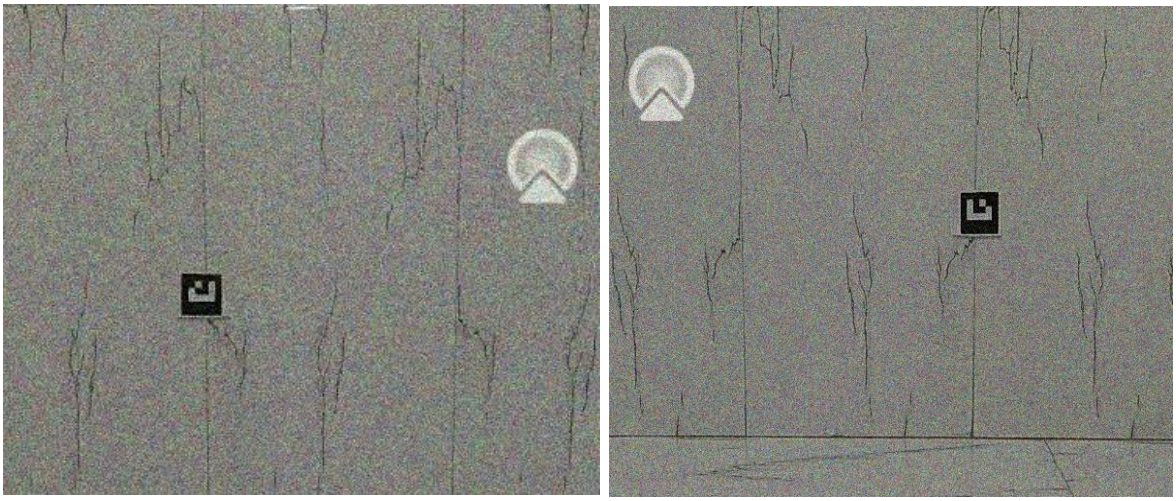
Nous avons ajouté un bruit d'image gaussien (variance =25) aux images de l'expérience précédente, puis répété les mêmes étapes.

Software / Résolution	Python ArUco %	Python AprilTag %
480p	**96.51**	**98.72**
720p	**98.94**	**98.23**
1080p	**98.73**	**96.46**

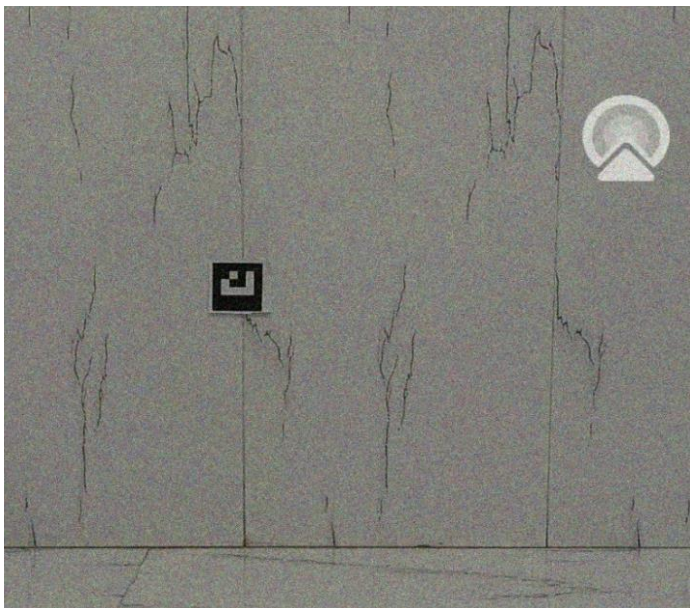
Tableau 3.3 : Teste de précision dans le cas un bruit d'image Gaussian avec Python.

Software / Résolution	Matlab ArUco %	Matlab AprilTag %
480p	96.83	97.93
720p	98.94	98.72
1080p	98.73	98.52

Tableau 3.4 : Teste de précision dans le cas un bruit d'image Gaussian avec Matlab.

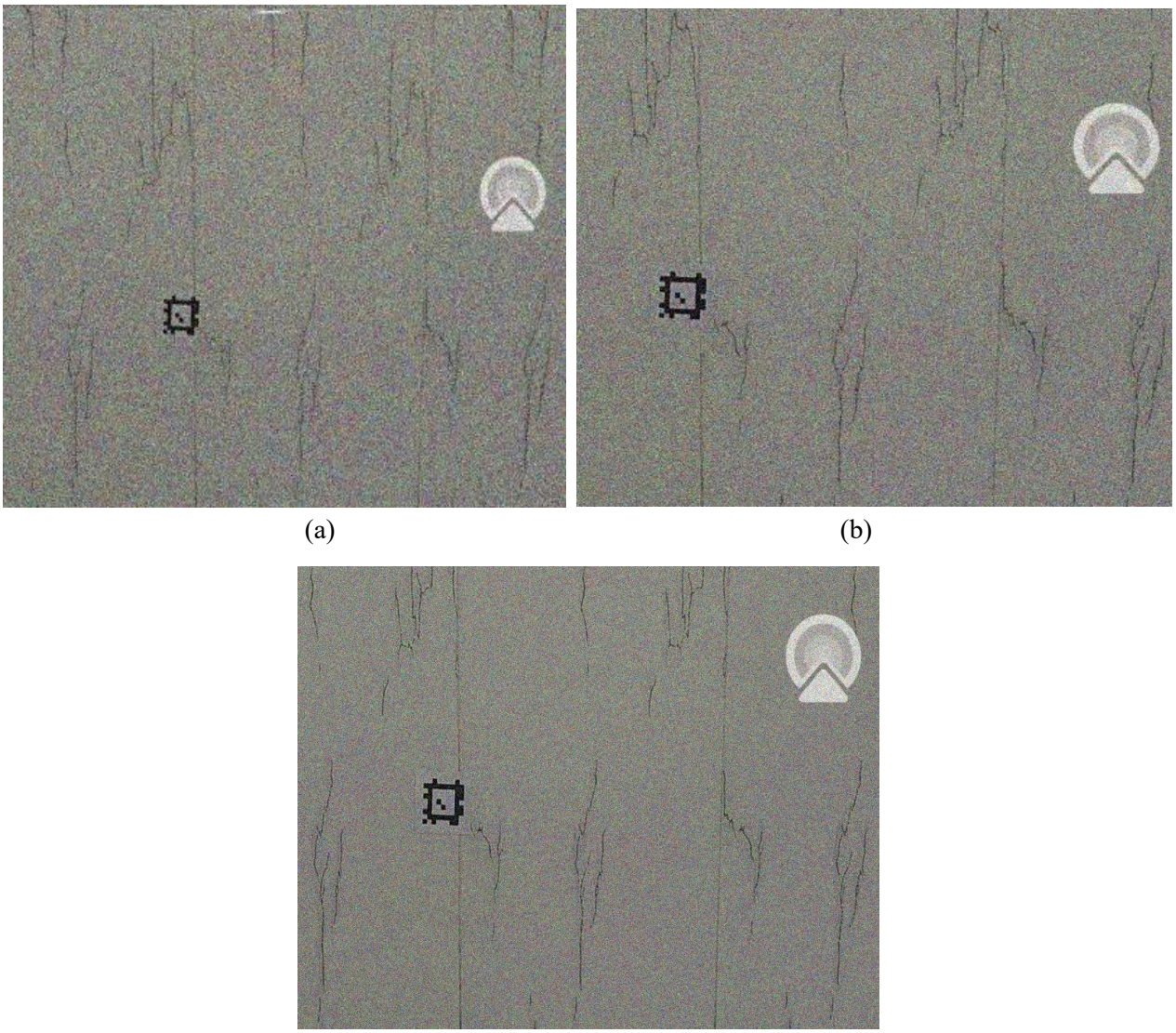


(a) (b)



(c)

Figure 3.10: (a) Frame 0 qualité 480p (Gaussian) ArUco (b) Frame 0 qualité 720p (Gaussian) ArUco (c) Frame 0 qualité 1080p (Gaussian) ArUco



(a) (b)

(c)

Figure 3.11: (a) Frame 0 qualité 480p (Gaussian) AprilTag (b) Frame 0 qualité 720p (Gaussian) AprilTag (c) Frame 0 qualité 1080p (Gaussian) AprilTag

Ces résultats montrent que l'ajout de bruit gaussien affecte légèrement la précision globale de détection, en particulier en résolution 480p. Toutefois, les deux bibliothèques conservent des taux de détection élevés, témoignant d'une bonne robustesse face au bruit. AprilTag semble légèrement plus sensible au bruit en résolution 1080p sous Python, tandis que MATLAB présente des performances plus stables dans l'ensemble.

b. Applique un bruit d'image (salt and pepper) :

Dans cette étape, nous avons appliqué un bruit d'image de type salt and pepper aux images issues des expériences précédentes. Ce type de bruit, caractérisé par l'introduction aléatoire de pixels noirs et blancs dans l'image, est couramment utilisé pour simuler des perturbations liées à des défauts de transmission ou de capteurs. La variance du bruit appliqué est restée constante tout au long de

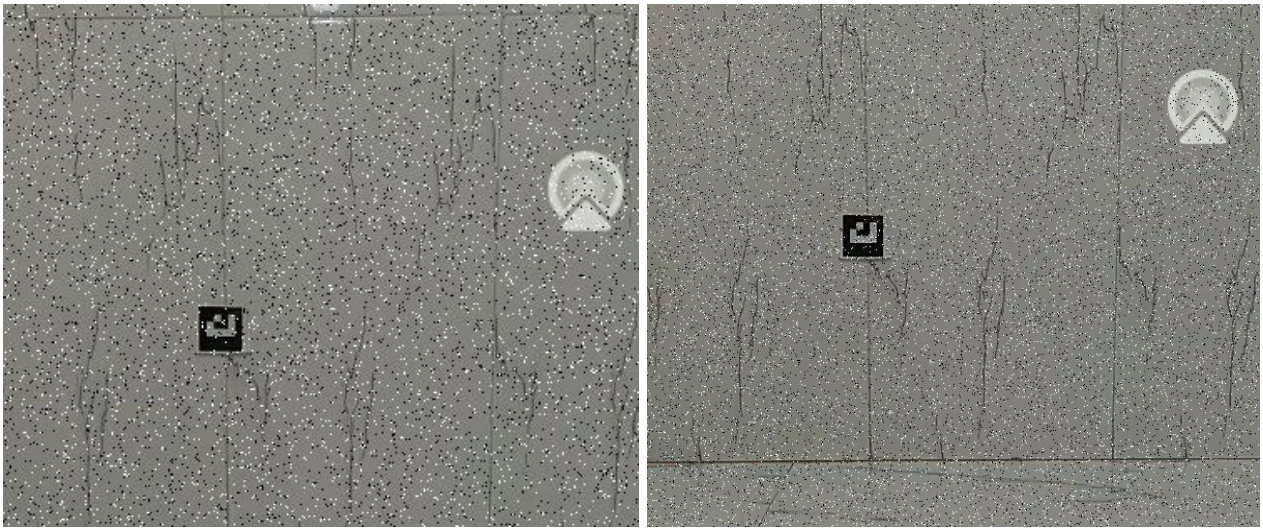
l'expérience (salt and pepper 0.1), et les étapes de détection ont été répétées pour évaluer l'impact sur la précision.

Software / Résolution	Python ArUco (%)	Python AprilTag (%)
480p	**91.97**	**37.72**
720p	**96.72**	**40.07**
1080p	**98.83**	**46.66**

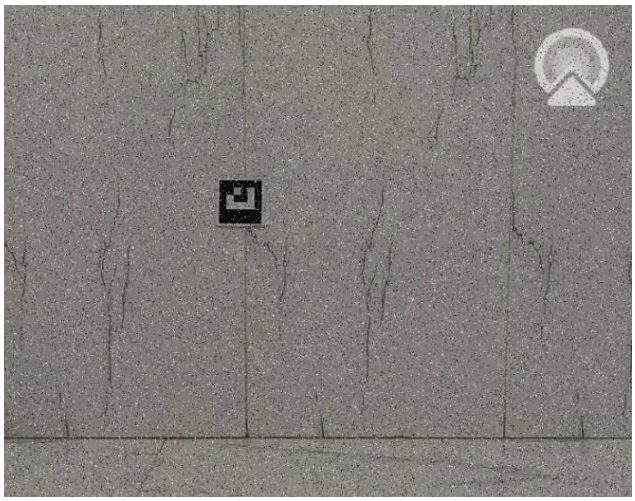
Tableau 3.5 : Teste de précision dans le cas salt and pepper avec Python.

Software / Résolution	Matlab ArUco %	Matlab AprilTag. %
480p	**91.97**	**17.28**
720p	**96.72**	**21.51**
1080p	**98.83**	**24.26**

Tableau 3.6 : Teste de précision dans le cas salt and pepper avec Matlab.

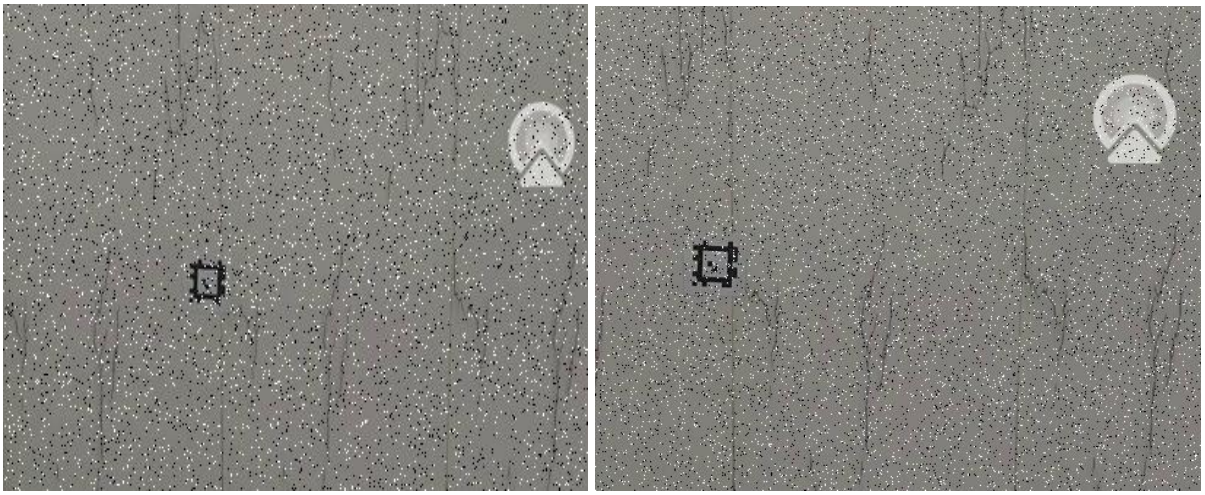


(a) (b)



(c)

Figure 3.12: (a) Frame 0 qualité 480p (salt and pepper) ArUco (b) Frame 0 qualité 720p (salt and pepper) ArUco (c) Frame 0 qualité 1080p (salt and pepper) ArUco



(a) (b)

(c)

Figure 3.13: (a) Frame 0 qualité 480p (salt and pepper) AprilTag (b) Frame 0 qualité 720p (salt and pepper) AprilTag (c) Frame 0 qualité 1080p (salt and pepper) AprilTag

Les résultats révèlent une nette différence de robustesse entre les deux bibliothèques face au bruit salt and pepper. ArUco maintient une bonne précision, même à basse résolution, tandis que AprilTag est fortement impactée, en particulier sous MATLAB où les taux chutent à moins de 25 % en résolution 1080p. Ce type de bruit semble perturber significativement l'identification des motifs pour AprilTag, probablement en raison de sa dépendance à des contrastes nets et à des contours réguliers. ArUco, quant à lui, démontre une plus grande tolérance aux perturbations impulsives, confirmant sa fiabilité dans des environnements bruyants ou dégradés.

3.6 Conclusion :

Dans un premier temps, nous avons présenté l'environnement matériel et logiciel mis en place, en détaillant l'installation des dépendances, la configuration des capteurs (caméra, capteur de distance) et des actionneurs (moteurs), ainsi que l'intégration des bibliothèques indispensables telles qu'OpenCV pour la vision par ordinateur.

- Par la suite, nous avons développé les algorithmes principaux du système, incluant :
- la détection et l'identification des marqueurs visuels,
- le traitement en temps réel des données issues des capteurs,
- la prise de décision et le contrôle des mouvements du robot (avancer, tourner, s'arrêter).

Ces algorithmes ont été testés, ajustés et optimisés afin de garantir la réactivité et la fiabilité du système dans un environnement dynamique.

Enfin, une phase d'évaluation a été menée à travers des simulations vidéo, permettant de valider les comportements attendus du robot. Une comparaison des performances entre les bibliothèques ArUco et AprilTag a également été intégrée, dans le but d'orienter les choix technologiques pour des applications robotiques plus exigeantes.

Ainsi, ce chapitre constitue une étape clé dans la mise en œuvre du projet, en consolidant les fondations techniques nécessaires pour envisager une implémentation réelle et plus avancée du système autonome.

Conclusion générale

Conclusion générale :

Ce mémoire a porté sur la conception, le développement et l'évaluation d'un système autonome basé sur la vision par marqueurs, intégrant des technologies de robotique mobile, de vision par ordinateur et d'électronique embarquée. En s'appuyant sur un Raspberry Pi comme unité centrale, le système a démontré la possibilité de créer une plateforme compacte, flexible et peu coûteuse, capable de percevoir son environnement, de prendre des décisions et d'agir de manière autonome.

L'objectif principal de ce travail était de concevoir un robot capable de naviguer intelligemment à l'aide de marqueurs visuels, tout en exploitant les bibliothèques telles qu'OpenCV, ArUco et AprilTag. À travers une démarche méthodique, nous avons d'abord étudié les fondements des systèmes robotiques autonomes, les principes de la vision artificielle, ainsi que les différentes technologies de marquage. Ces éléments ont servi de socle théorique solide à la conception du système.

La seconde phase s'est articulée autour du développement matériel et logiciel, comprenant le choix et l'intégration des composants (caméra, capteurs de distance, moteurs), la configuration des environnements de développement, et la mise en œuvre des algorithmes de détection, de suivi, et de navigation. Des scénarios de simulation ont été conçus pour pallier l'absence de tests sur le robot physique, permettant néanmoins de valider les performances fonctionnelles du système.

Enfin, une évaluation comparative entre ArUco et AprilTag a été menée selon plusieurs critères (résolution vidéo, présence de bruit, environnement d'exécution), permettant de mieux cerner leurs avantages respectifs dans des contextes robotiques variés. Cette étude expérimentale a confirmé la robustesse du système, tout en soulignant les limites à prendre en compte pour une mise en œuvre en conditions réelles.

En conclusion, ce travail illustre comment des technologies accessibles et open source peuvent être combinées pour concevoir des solutions autonomes, intelligentes et économiques. Il ouvre des perspectives concrètes dans des domaines tels que :

- La logistique automatisée,
- La navigation autonome en milieux structurés,
- Ou encore l'enseignement de la robotique et des systèmes embarqués.

Comme perspective d'amélioration, je pourrais intégrer ROS (Robot Operating System) dans ce projet, ce qui me permettrait de profiter de son écosystème modulaire et de ses outils puissants pour gérer les capteurs, faciliter la communication entre les différents modules, et implémenter des algorithmes robotiques plus avancés.

On peut remplacer le Raspberry Pi par quelque chose de plus puissant, comme NVIDIA Jetson, Google Coral, Odroid ou Intel NUC.

Les pistes futures incluraient notamment l'intégration du système sur un robot physique réel, l'optimisation des performances en temps réel, et l'élargissement à des scénarios multi-robots ou à des environnements non balisés. Ce projet constitue ainsi une base solide pour de futurs travaux de recherche ou de développement industriel.

Références :

[1] : Laetitia Matignon Introduction à la robotique Licence 1ere année - 2011/2012 https://perso.liris.cnrs.fr/laetitia.matignon/index/coursL1robotique.pdf

[2] : « Histoire de la robotique : des automates aux premiers robots | Dossier ». Disponible sur: https://www.futurasciences.com/tech/dossiers/robotique-robotique-a-z-178/page/2/

[3] : https://ieeexplore.ieee.org/document/1638022 visite ce site 15/03/2025

[4] : Nils J. Nilsson Shakey the robot Technical note 323 April 1984 https://www.sri.com/wp-content/uploads/2021/12/629.pdf

[5] : Belkhatir Wahida et Kassab Feriel réalisation d'un robot nettoyeur Mémoire de fin d'étude Pour l'obtention de diplôme Master 2024/2025 https://ds.univ-oran2.dz:8443/bitstream/123456789/8642/1/projet%20fin%20etude%202024.pdf

[6] : **ISO 8373:2021** (Norme internationale) :
« Machine programmable, autonome ou semi-autonome, capable de déplacement ou d'interaction avec un environnement pour exécuter des tâches spécifiques. »

[7] : Snani Haifaa navigation d'un robot mobile dans un environnement dynamique Mémoire Présenté en vue de l'obtention du Diplôme de Master 2020/2021 https://biblio.univ-annaba.dz/ingeniorat/wp-content/uploads/2022/02/Snani-Haifaa.pdf

[8] : Mahdi Bouloussekh et Bouaza Gharboudje Etude et conception d'une plat forme mobile Projet de Fin d'Etude préparé En vue de l'obtention du diplôme de MASTER 2019/2020 https://opac.centre-univ-mila.dz/z//6210111.pdf

[9] : BOUSIF Fares Conception, réalisation et implémentation d'un robot mobile dans un environnement de travail Projet Fin D'étude En vue d'obtention du Diplôme de Master 2019/2020 http://dspace.univ-tlemcen.dz/bitstream/112/16008/1/Ms.Gee.Bousif.pdf

[10] : https://www.ultralytics.com/fr/blog/understanding-the-integration-of-computer-vision-in-robotics#:~:text=Vision%20L'IA%20ou%20la,aide%20de%20la%20vision%20artificielle. Visite ce site 25/04/2025

[11]: Chun-ting Lin Implementation and Evaluation of Marker-based Visual Localization for Mobile Robots Bachelor Thesis 26.11.2023 https://opus4.kobv.de/opus4-haw/files/4282/I001744202Thesis.pdf

[12] : Charlie Solutions. (2023, janvier). QR Code : Tout ce que vous devez savoir. Consulté Juin 01, 2025, à l'adresse https://www.charlie-solutions.com/blog-qr-code-ce-que-vous-devez-savoir/

[13] : VEX Robotics. (2025, April). Utilisation d'AprilTags avec le capteur de vision AI. VEX Robotics Knowledge Base. https://kb.vex.com/hc/fr/articles/30488063603988-Utilisation-d-AprilTags-avec-le-capteur-de-vision-AI

[14]: APRIL Robotics Lab. (n.d.). AprilTag: A visual fiducial system. University of Michigan. https://april.eecs.umich.edu/software/apriltag

[15]: Dupont, A., & Ionescu, R. (2023, octobre). Fiducial Marker Systems : aperçu et évaluation empirique. Scientific Bulletin – Série *D, 85*(2). Repéré le 14 Mai 2025, à l'adresse https://www.scientificbulletin.upb.ro/rev_docs_arhiva/rez4e8_956916.pdf

[16] : Chaouch Ibrahim & Semmar Hocine Commande d'un bras de robot par vision artificielle Mémoire de Master 2017-2018 https://di.univ-blida.dz/jspui/bitstream/123456789/2302/1/memoire-memoire.pdf

[17] : tentone. (s.d.). **aruco : Aruco marker detector and pose estimation for AR and Robotics with ROS support** [Dépôt GitHub]. GitHub. Repéré le 07 juin 2025, à l'adresse https://github.com/tentone/aruco

[18]: GeeksforGeeks. (s.d.). I2C Communication Protocol. Repéré le 05 juin 2025, à l'adresse https://www.geeksforgeeks.org/i2c-communication-protocol/
[19] : Serial Peripheral Interface. (2025, mars). Wikipédia. Repéré le 14 juin 2025, à l'adresse https://fr.wikipedia.org/wiki/Serial_Peripheral_Interface
[20] : Wi-Fi. (2025, 13 juin). Wikipédia. Repéré le 14 juin 2025, à l'adresse https://fr.wikipedia.org/wiki/Wi-Fi
[21] : Gotronic. (s.d.). Carte Raspberry Pi 3 B+ – processeur Broadcom et 40 broches d'E/S. Repéré le 14 juin 2025, à l'adresse https://www.gotronic.fr/art-carte-raspberry-pi-3-b-27826.htm#:~:text=Le%20modèle%20Raspberry%20Pi3%20B%2B%20est%20basée%20sur%20un%20processeur,broches%20d'E%2FS.
[22] : Silverline Electronics. (s.d.). Raspberry Pi 3 Model B : ordinateur monocarte avec processeur quad-core 1,2 GHz, 1 Go RAM, Wi-Fi et Bluetooth [page produit]. Repéré le 14 juin 2025, à l'adresse https://www.silverlineelectronics.in/products/raspberry-pi-3-model-b-1#:~:text=The%20Raspberry%20Pi%203%20Model…

[23]: Fabbri, Ricardo; Giblin, Peter; Kimia, Benjamin (2012). "Camera Pose Estimation Using First-Order Curve Differential Geometry". Computer Vision – ECCV 2012 *(PDF)*. Lecture Notes in Computer Science. Vol. 7575. pp. *231*–244.
[24] : Voltaat. (s.d.). Raspberry Pi Camera V2 : module caméra 8 MP pour Raspberry Pi [page produit]. Repéré le 14 juin 2025, à l'adresse https://www.voltaat.com/products/raspberry-pi-camera-v2
[25] : Otronic. (s.d.). Capteur de distance à ultrasons HC-SR04 [page produit]. Repéré le 14 juin 2025, à l'adresse https://www.otronic.nl/fr/capteur-de-distance-a-ultrasons-hc-sr04.html
[26]: GreenSky Power Technology Co., Ltd. (2022). Servo motor vs stepper motor : quel est le meilleur ? [Article]. Repéré le 14 juin 2025, à l'adresse https://greensky-power.com/servo-motor-vs-stepper-motor/
[27]: Mehmood, A. (2022, 30 septembre). Camera calibration in Gazebo ROS2 [Article Medium]. Medium. Repéré le 14 juin 2025, à l'adresse https://medium.com/@arshad.mehmood/camera-calibration-in-gazebo-ros2-6bed2620a652
[28] : Čejka, J., Bruno, F., Skarlatos, D., & Liarokapis, F. (2019). Detecting square markers in underwater environments. *Remote Sensing, 11*(4), 459. https://doi.org/10.3390/rs11040459
[29]: H. Kato and M. Billinghurst, "Marker tracking and hmd calibration for a video-based augmented reality conferencing system," in Proceedings 2nd IEEE and ACM International Workshop on Augmented Reality (IWAR'99), pp. 85–94, 1999.
[30]: J. Rekimoto and Y. Ayatsuka, "Cybercode: designing augmented reality environments with visual tags," in Proceedings of DARE 2000 on Designing augmented reality environments, pp. 1–10, 2000.
[31]: M. Fiala, "Artag, a fiducial marker system using digital techniques," in 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05), vol. 2, pp. 590–596 vol. 2, 2005.
[32]: J. Sattar, E. Bourque, P. Giguere, and G. Dudek, "Fourier tags: Smoothly degradable fiducial markers for use in human-robot interaction," in Fourth Canadian Conference on Computer and Robot Vision (CRV'07), pp. 165–174, IEEE, 2007.
[33]: B. Atcheson, F. Heide, and W. Heidrich, "Caltag: High precision fiducial markers for camera calibration.," in VMV, vol. 10, pp. 41–48, 2010. [11] E. Olson, "Apriltag: A robust and flexible visual fiducial system," in 2011 IEEE International Conference on Robotics and Automation, pp. 3400–3407,

2011. [12] F. Bergamasco, A. Albarelli, E. Rodol`a, and A. Torsello, “Rune-tag: A high accuracy fiducial marker with strong occlusion resilience,” in CVPR 2011, pp. 113–120, 2011.
[34]: L. Calvet, P. Gurdjos, and V. Charvillat, “Camera tracking using concentric circle markers: Paradigms and algorithms,” in 2012 19th IEEE International Conference on Image Processing, pp. 1361–1364, 2012.
[35]: S. Garrido-Jurado, R. Mu˜noz-Salinas, F. Madrid-Cuevas, and M. Mar´ın-Jim´enez, “Automatic generation and detection of highly reliable fiducial markers under occlusion,” Pattern Recognition, vol. 47, no. 6, pp. 2280–2292, 2014.
[36]: J. DeGol, T. Bretl, and D. Hoiem, “Chromatag: A colored marker and fast detection algorithm,” in Proceedings of the IEEE International Conference on Computer Vision, pp. 1472–1481, 2017.

Annexe

- **Programme calibration du camera :**

La calibration d'une caméra est un processus essentiel en vision par ordinateur. Elle permet de déterminer les paramètres intrinsèques et extrinsèques d'une caméra, ainsi que les coefficients de distorsion optique. Ces paramètres permettent de corriger les déformations dans les images capturées et de comprendre la relation entre les points 3D dans le monde réel et leurs projections 2D dans une image.

On a effectué la calibration d'une caméra en Python à l'aide de la bibliothèque OpenCV. Voici le programme utilisé :

```python
import cv2
import numpy as np

# Calibration settings
chessboard_size = (8, 6)
square_size = 2.5
num_samples = 20

# Prepare object points
objp = np.zeros((chessboard_size[0]*chessboard_size[1], 3), np.float32)
objp[:, :2] = np.mgrid[0:chessboard_size[0], 0:chessboard_size[1]].T.reshape(-1, 2)
objp *= square_size

# Arrays to store object points and image points
objpoints = []
imgpoints = []

# Start video capture
cap = cv2.VideoCapture(0)

if not cap.isOpened():
    print("Failed to open camera.")
    exit()

print(f"Collecting {num_samples} samples. Press 's' to save a frame with detected corners.")

```

Figure A.1 : Programme calibration du camera partie 1

```python
while len(objpoints) < num_samples:
    ret, frame = cap.read()
    if not ret:
        break

    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    ret_corners, corners = cv2.findChessboardCorners(gray, chessboard_size, None)

    display_frame = frame.copy()

    if ret_corners:
        cv2.drawChessboardCorners(display_frame, chessboard_size, corners, ret_corners)
        cv2.putText(display_frame, f"Detected. Press 's' to save ({len(objpoints)}/{num_samples})",
                    (10, 30), cv2.FONT_HERSHEY_SIMPLEX, 0.6, (0, 255, 0), 2)

    cv2.imshow("Camera Calibration", display_frame)
    key = cv2.waitKey(1)

    if key == ord('s') and ret_corners:
        objpoints.append(objp.copy())
        imgpoints.append(corners)
        print(f"Saved frame {len(objpoints)}")
    elif key == ord('q'):
        print("Quitting early.")
        break

cap.release()
cv2.destroyAllWindows()
```

Figure A.2: Programme calibration du camera partie 2.

```python

# Perform calibration
if len(objpoints) >= 5:
    ret, camera_matrix, dist_coeffs, rvecs, tvecs = cv2.calibrateCamera(
        objpoints, imgpoints, gray.shape[::-1], None, None
    )

    # Save results
    np.savez('live_camera_calibration.npz',
             camera_matrix=camera_matrix,
             dist_coeffs=dist_coeffs,
             rvecs=rvecs,
             tvecs=tvecs)

    print("Calibration complete.")
    print("Camera matrix:\n", camera_matrix)
    print("Distortion coefficients:\n", dist_coeffs)
else:
    print("Not enough valid samples for calibration.")
```

Figure A.3 : Programme calibration du camera partie 3.

Explication :

Nous avons utilisé un échiquier concentrique imprimé sur une feuille comme cible de calibration, en raison de la précision connue de ses coordonnées et de la facilité de détection de ses coins. Pour garantir une meilleure précision des calculs, nous avons capturé 20 images de la cible sous différents angles et positions, couvrant ainsi une large gamme de perspectives. Les coins de l'échiquier ont ensuite été identifiés sur chaque image. Après cette étape, les paramètres de calibration ont été calculés. Enfin, les résultats ont été validés en projetant des points 3D sur l'image et en les comparant aux positions réelles, avant que les paramètres ne soient enregistrés pour une utilisation ultérieure.

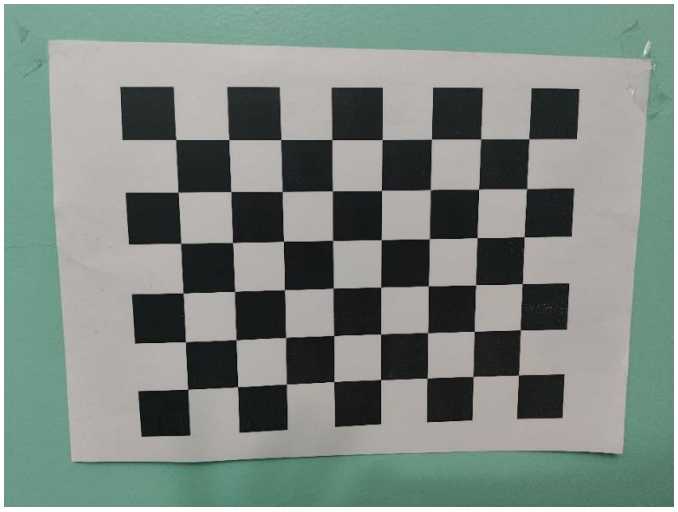


Figure A.4: un échiquier.

- **Programme de transfert vidéo vers images :**

```python
import cv2
video = cv2.VideoCapture("french.mp4")
count = 0

while True:
    success, frame = video.read()
    if not success:
        break
    cv2.imwrite(f"frame_{count:04d}.jpg", frame)
    count += 1

video.release()
```

Figure A.5 : Programme de transfert vidéo vers images.

Explication :

Pour étudier le teste de précision, nous avons utilisé ce programme pour convertir les vidéos capturées en images. Chaque image (frame) a ensuite été analysée pour identifier les cas où les marqueurs étaient visibles mais n'étaient pas détectés.

- **Programme pour détecter les ArUcos dans les photos :**

- **Cas Python :**

```python
import cv2
import numpy as np
import os
import pandas as pd


camera_matrix = np.array([
    [1.45968928e+03, 0.0, 5.82489209e+02],
    [0.0, 1.48375268e+03, 9.65179246e+02],
    [0.0, 0.0, 1.0]
])

dist_coeffs = np.array([[ 0.22991493, -0.42394174,  0.02578426,
                          0.0061953,  0.32459605]])

marker_length = 0.048  # In meters

# --- Path
input_folder = r'C:\Users\Administrator\Desktop\pyhon\cew3 with gussain'
output_csv = 'detection_results.csv'
```

Figure A.6 : Programme pour détecter les ArUcos dans les photos Pour Python partie 1

```python
#aruco detector
aruco_dict = cv2.aruco.getPredefinedDictionary(cv2.aruco.DICT_4X4_50)
aruco_params = cv2.aruco.DetectorParameters()
aruco_detector = cv2.aruco.ArucoDetector(aruco_dict, aruco_params)


obj_points = np.array([
    [-marker_length / 2,  marker_length / 2, 0],
    [ marker_length / 2,  marker_length / 2, 0],
    [ marker_length / 2, -marker_length / 2, 0],
    [-marker_length / 2, -marker_length / 2, 0]
], dtype=np.float32)


results = []

image_files = [f for f in os.listdir(input_folder) if f.lower().endswith(('.jpg', '.jpeg', '.png'))]

for filename in image_files:
    img_path = os.path.join(input_folder, filename)
```

Figure A.7: Programme pour détecter les ArUcos dans les photos Pour Python partie 2

```
    image = cv2.imread(img_path)

    if image is None:
        print(f"Could not read {filename}")
        continue

    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    corners, ids, _ = aruco_detector.detectMarkers(gray)

    if ids is not None:
        for i in range(len(corners)):
            marker_corners = corners[i].reshape(-1, 2)

            success, rvec, tvec = cv2.solvePnP(obj_points, marker_corners, camera_matrix, dist_coeffs)

            if success:
                distance = np.linalg.norm(tvec)
                results.append({
                    'image': filename,
                    'marker_id': int(ids[i][0]),
```

Figure A.8 : Programme pour détecter les ArUcos dans les photos Pour Python partie 3.

```
                    'distance_m': round(distance, 3)
                })


                cv2.aruco.drawDetectedMarkers(image, corners, ids)
                cv2.drawFrameAxes(image, camera_matrix, dist_coeffs, rvec, tvec, 0.03)
            else:
                print(f"Pose estimation failed for marker in {filename}")
    else:
        print(f"No marker found in {filename}")
        results.append({'image': filename, 'marker_id': None, 'distance_m': None})


print(f"\n✅ Detection completed.)
```

Figure A.9 : Programme pour détecter les ArUcos dans les photos Pour Python partie 4.

- **Cas Matlab :**

```
folderPath = 'C:\Users\Administrator\Desktop\pyhon\cew3 with gussain';

imageFiles = dir(fullfile(folderPath, '*.jpg'));

% Camera parameters
cameraMatrix = [957.84196598, 0, 374.26935095;
                0, 968.79451461, 642.64699993;
                0, 0, 1];
radialDistortion = [0.20854384, -0.43923496, 0.35367725];
tangentialDistortion = [0.01763289, 0.00301464];
imageSize = [720, 1280];

cameraParams = cameraParameters( ...
    'IntrinsicMatrix', cameraMatrix', ...
    'RadialDistortion', radialDistortion(1:2), ...
    'TangentialDistortion', tangentialDistortion, ...
    'ImageSize', imageSize);

markerLength = 0.048;

for k = 1:length(imageFiles)
    % Read image
    I = imread(fullfile(imageFiles(k).folder, imageFiles(k).name));

    % Detect markers
    [ids, locs, detectedFamily] = readArucoMarker(I);
    numMarkers = length(ids);

    for i = 1:numMarkers
        loc = locs(:,:,i);
```

Figure A.10 : Programme pour détecter les ArUcos dans les photos Pour Matlab partie 1.

```
        worldPoints = [0, 0, 0;
                       markerLength, 0, 0;
                       markerLength, markerLength, 0;
                       0, markerLength, 0];


        [rotationMatrix, translationVector] = extrinsics(loc, worldPoints(:,1:2), cameraParams);


        distance = norm(translationVector) * 100;


        I = insertShape(I, 'polygon', {loc}, 'Opacity', 1, 'ShapeColor', 'green', 'LineWidth', 4);
        markerRadius = 6;
        markerPosition = [loc, repmat(markerRadius, size(loc,1), 1)];
        I = insertShape(I, 'FilledCircle', markerPosition, 'ShapeColor', 'red', 'Opacity', 1);


        center = mean(loc);
        label = sprintf("ID: %d, Dist: %.1f cm", ids(i), distance);
        I = insertText(I, center, label, 'FontSize', 30, 'BoxOpacity', 1);
    end


    imshow(I);
    title(['Processing image ', num2str(k), ' / ', num2str(length(imageFiles))]);
    pause(0.1);
end
```

Figure A.11 : Programme pour détecter les ArUcos dans les photos Pour Matlab partie 2.

Explication :

Dans ce programme, nous avons fourni des images (frames) contenant des marqueurs visibles. Le programme analyse ces images pour identifier celles où les marqueurs sont détectés et met en évidence celles où ils ne le sont pas.

- **Programme applique du bruit à l'image (Gaussian) :**

```python
import os
import cv2
import numpy as np

input_folder = r'C:\Users\Administrator\Desktop\pyhon\cew3'
output_folder = r'C:\Users\Administrator\Desktop\pyhon\cew3 with gussain'
mean = 0
std_dev = 25


for filename in os.listdir(input_folder):
    if filename.lower().endswith(('.png', '.jpg', '.jpeg', '.bmp', '.tiff')):
        # Read image
        img_path = os.path.join(input_folder, filename)
        image = cv2.imread(img_path)

        # Generate Gaussian noise
        noise = np.random.normal(mean, std_dev, image.shape).astype(np.float32)
        noisy_image = image.astype(np.float32) + noise
```

Figure A.12 : Programme applique du bruit à l'image (Gaussian) partie 1.

```python
        # Clip values to valid range and convert back to uint8
        noisy_image = np.clip(noisy_image, 0, 255).astype(np.uint8)

        output_path = os.path.join(output_folder, filename)
        cv2.imwrite(output_path, noisy_image)

print("Gaussian noise applied to all images.")
```

Figure A.13 : Programme applique du bruit à l'image (Gaussian) partie 2.

Explication :

Dans ce programme, nous avons fourni des images (frames) contenant des marqueurs visibles. Le programme ajoute un bruit d'image (gaussian) a ces images.

➢ **Programme appliquer du bruit à l'image (salt and pepper) :**

```python
import os
import cv2
import numpy as np
from pathlib import Path

def add_salt_and_pepper_noise(image, salt_prob=0.01, pepper_prob=0.01):
    noisy = np.copy(image)
    total_pixels = image.size

    # Add Salt noise (white pixels)
    num_salt = np.ceil(salt_prob * total_pixels).astype(int)
    coords = [np.random.randint(0, i - 1, num_salt) for i in image.shape[:2]]
    noisy[coords[0], coords[1]] = 255

    # Add Pepper noise (black pixels)
    num_pepper = np.ceil(pepper_prob * total_pixels).astype(int)
    coords = [np.random.randint(0, i - 1, num_pepper) for i in image.shape[:2]]
    noisy[coords[0], coords[1]] = 0

    return noisy
```

Figure A.14 : Programme applique sel et poivre partie 1.

```python

def process_folder(input_folder, output_folder, salt_prob=0.01, pepper_prob=0.01):
    input_path = Path(input_folder)
    output_path = Path(output_folder)
    output_path.mkdir(exist_ok=True)

    for image_file in input_path.glob("*.*"):
        img = cv2.imread(str(image_file))
        if img is None:
            continue
        noisy_img = add_salt_and_pepper_noise(img, salt_prob, pepper_prob)
        output_file = output_path / image_file.name
        cv2.imwrite(str(output_file), noisy_img)

# Example usage:
process_folder(r"C:\Users\Administrator\Desktop\pyhon\cew3", r"C:\Users\Administrator\Desktop\pyhon\cew3 with s
```

Figure A.15 : Programme applique sel et poivre partie 2.

Explication :

Dans ce programme, nous avons fourni des images (frames) contenant des marqueurs visibles. Le programme ajoute un bruit d'image (salt and pepper) a ces images.

➢ **Programme pour détecter les AprilTag dans les photos :**

- **Pour Python :**

```python
import cv2
import numpy as np
import os
from pupil_apriltags import Detector

camera_matrix = np.array([
    [1.45968928e+03, 0.0, 5.82489209e+02],
    [0.0, 1.48375268e+03, 9.65179246e+02],
    [0.0, 0.0, 1.0]
])
dist_coeffs = np.array([[0.22991493, -0.42394174, 0.02578426, 0.0061953, 0.32459605]])

marker_length = 0.048

input_folder = r'C:\Users\Administrator\Desktop\pyhon\ap1080salt'

at_detector = Detector(
    families='tagStandard41h12',
    nthreads=1,
    quad_decimate=1.0,
```

Figure A.16 : Programme pour détecter les AprilTags dans les photos Pour Python partie 1

```python
    quad_sigma=0.0,
    refine_edges=1,
    decode_sharpening=0.25,
    debug=0
)
# 3D object points of tag corners
obj_points = np.array([
    [-marker_length / 2,  marker_length / 2, 0],
    [ marker_length / 2,  marker_length / 2, 0],
    [ marker_length / 2, -marker_length / 2, 0],
    [-marker_length / 2, -marker_length / 2, 0]
], dtype=np.float32)

results = []

image_files = [f for f in os.listdir(input_folder) if f.lower().endswith(('.jpg', '.jpeg

for filename in image_files:
    img_path = os.path.join(input_folder, filename)
    image = cv2.imread(img_path)
```

Figure A.17 : Programme pour détecter les AprilTags dans les photos Pour Python partie 2

```
    if image is None:
        print(f"❌ Could not read {filename}")
        continue

    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    detections = at_detector.detect(gray, estimate_tag_pose=False)

    if detections:
        for det in detections:
            corners = np.array(det.corners, dtype=np.float32)
            success, rvec, tvec = cv2.solvePnP(obj_points, corners, camera_matrix, dist_(

            if success:
                distance = np.linalg.norm(tvec)
                results.append({
                    'image': filename,
                    'tag_id': det.tag_id,
                    'distance_m': round(distance, 3)
                })

                for i in range(4):
```

Figure A.18 : Programme pour détecter les AprilTags dans les photos Pour Python partie 3

```
                for i in range(4):
                    pt1 = tuple(corners[i].astype(int))
                    pt2 = tuple(corners[(i + 1) % 4].astype(int))
                    cv2.line(image, pt1, pt2, (0, 255, 0), 2)

                cv2.drawFrameAxes(image, camera_matrix, dist_coeffs, rvec, tvec, 0.03)
            else:
                print(f"❌ Pose estimation failed for tag {det.tag_id} in {filename}")
    else:
        print(f"⚠️ No AprilTag found in {filename}")
        results.append({'image': filename, 'tag_id': None, 'distance_m': None})

print("\n✅ AprilTag detection completed.")
```

Figure A.19 : Programme pour détecter les AprilTags dans les photos Pour Python partie 4

- **Cas Matlab :**

```
inputFolder = 'C:\Users\Administrator\Desktop\pyhon\ap480g';

focalLength = [1459.68928, 1483.75268];
principalPoint = [582.489209, 965.179246];
tagSize = 0.048;

sampleImage = imread(fullfile(inputFolder, 'frame_0000.jpg'));
actualImageSize = size(sampleImage);
actualImageSize = actualImageSize(1:2);  % [height, width]
intrinsics = cameraIntrinsics(focalLength, principalPoint, actualImageSize);

imageFiles = dir(fullfile(inputFolder, '*.jpg'));
for k = 1:length(imageFiles)

    imgPath = fullfile(inputFolder, imageFiles(k).name);
    I = imread(imgPath);
    gray = rgb2gray(I);
    imageSize = size(gray);

    I = undistortImage(I, intrinsics, OutputView='same');

    [id, loc, pose] = readAprilTag(I, "tagStandard41h12", intrinsics, tagSize);

    if ~isempty(id)
        worldPoints = [0 0 0; tagSize/2 0 0; 0 tagSize/2 0; 0 0 tagSize/2];

        for i = 1:length(pose)

            imagePoints = worldToImage(intrinsics, pose(i).Rotation, pose(i).Translation, worldPoints);

            I = insertShape(I, "Line", [
                imagePoints(1,:) imagePoints(2,:);
                imagePoints(1,:) imagePoints(3,:);
```

Figure A.20 : Programme pour détecter les AprilTags dans les photos Pour Matlab partie 1.

```
                imagePoints(1,:) imagePoints(4,:)
            ], ShapeColor=["red","green","blue"], LineWidth=5);

            I = insertText(I, loc(1,:,i), id(i), BoxOpacity=1, FontSize=24);
        end
    else
        disp("⚠ No AprilTag detected in: " + imageFiles(k).name);
    end

    imshow(I);
    drawnow;
end

disp("✅ Finished processing all images.");
```

Figure A.21 : Programme pour détecter les AprilTags dans les photos Pour Matlab partie 2.

Explication :

Dans ce programme, nous avons fourni des images (frames) contenant des marqueurs visibles. Le programme analyse ces images pour identifier celles où les marqueurs sont détectés et met en évidence celles où ils ne le sont pas.