« Simulation et contrôle d'un robot terrestre »
FISE - 2021

ENSTA Bretagne
2 rue F. Verny
29806 Brest Cedex 9, France

LAVAIVRE, Jacques,
jacques.lavaivre@ensta-bretagne.org

Contact :

ZERR, Benoît,
benoit.zerr@ensta-bretagne.org

Remerciements

Je remercie M. Benoît ZERR mon maître de stage pour m'avoir donné la possibilité de réaliser se stage. Je le remercie aussi pour sa confiance, les connaissances qu'il a su partager avec moi et sa disponibilité malgré le contexte particulier dans lequel s'est déroulé ce stage.
Je tiens aussi à remercier l'équipe pédagogique de l'ENSTA Bretagne qui m'a permis d'acquérir les connaissances nécessaires à la réalisation de ce stage.
Un grand merci à mes parents pour leur soutien et leurs conseils.
Enfin, j'adresse mes remerciements à l'ensemble de mes camarades de robotique qui m'ont aidé à rattraper mon retard et m'ont aiguillé vers les bonnes ressources quand j'en avais besoin.

Résumé

On entend de plus en plus parler de robotique autonome avec par exemple l'autopilot de Tesla et il est certain que dans le monde de demain robots et humains seront amener à se côtoyer et à travailler main dans la main. C'est à nous, ingénieurs, de mettre en place ces innovations.

C'est dans ce contexte que j'ai effectué mon stage d'assistant ingénieur au sein du département de robotique de l'ENSTA Bretagne. Durant cette période, j'avais pour objectif d'implémenter un logiciel pour permettre à un robot de réaliser des missions de manière autonome. Le robot devait être capable d'effectuer un parcours donné tout en évitant les obstacles disposés sur son passage via sa caméra embarquée.

A cause du contexte sanitaire particulier, le stage s'est effectué en télétravail et les résultats ont été déposés en ligne.

A l'issue de ce stage, j'ai pu grandement améliorer mes connaissances en robotique et apprendre à travailler en autonomie totale.

Abstract

We hear more and more talk about autonomous robotics with, for example, Tesla's autopilot and it is certain that in tomorrow's world robots and humans will be brought together and work hand in hand. It is up to us, engineers, to implement these innovations.

It is in this context that I did my internship as an assistant engineer in the robotics department of ENSTA Bretagne. During this period, my objective was to implement software to enable a robot to carry out missions autonomously. The robot had to be able to perform a given route while avoiding obstacles in its path via its on-board camera.

Because of the particular health context, the training course was carried out in teleworking and the results were submitted online.

At the end of this course, I was able to greatly improve my knowledge of robotics and learn to work in total autonomy.

Sommaire

Introduction

L'objectif de ce stage était de concevoir un logiciel permettant à un robot mobile d'effectuer des missions en autonomie totale. Le robot devait effectuer un parcours tout en évitant les obstacles sur son chemin grâce à sa caméra. La position du robot était calculée par un GPS intégré. Il fallait donc que le logiciel puisse traiter les images de la caméra pour en détecter les différents obstacles

Ce projet s'inscrit dans la continuité de ma formation en robotique et traite d'une problématique actuelle : les robots autonomes ont pour vocation de remplacer les emplois à faibles valeurs ajoutées et le secteur est en plein expansion.

Au vu de la crise sanitaire, ce stage a été effectué en télétravail et le but de ce rapport est de fournir un retour sur le travail que j'ai effectué. Le logiciel n'ayant pas été entièrement fini, il pourra servir de point de départ ou d'inspiration pour une personne souhaitant réaliser un travail similaire. L'ensemble du travail réalisé lors du stage est notamment disponible en ligne sur le dépôt GitHub.

Dans un premier temps, je présenterais l'organisme au sein duquel j'ai effectué mon stage, ensuite nous étudierons le besoin auquel répondent les robots autonomes, puis j'expliquerais comment j'ai procédé pour concevoir les logiciels et les problèmes que j'ai rencontré lors de sa conception.

1. Présentation de l'ENSTA Bretagne et de son centre de recherche

1.1.Historique

Créée il y a plus de deux siècles en 1819, l'ENSTA Bretagne était à l'origine une école de maistrance chargée de former les futurs officiers mariniers. Elle devient par la suite habilitée à délivrer un diplôme d'ingénieur en 1934. En 1971 elle forme avec les autres écoles d'armements terrestre et d'aéronautique l'Ecole Nationale Supérieure des ingénieurs des études et techniques d'armement (aussi appelé ENSIETA). Ce n'est qu'en 1988 que l'école commence à accueillir des élèves civils. En 2010, l'école change de nom et se renomme ENSTA Bretagne pour former le groupe ENSTA avec l'école ENSTA Paris tech.

Les laboratoires de recherche ont été inaugurés au sein de l'école en 2005 et ensuite étendus en 2010. Aujourd'hui, l'ENSTA Bretagne compte plus d'une centaine de doctorants et 74 enseignants-chercheurs.

1.2.Domaine d'activité du centre de recherche

L'école mène des recherches dans trois domaines différents : les sciences mécaniques, les sciences technologiques de l'information (STIC) et les sciences humaines et sociales (SHS).

Le département dans lequel j'ai effectué mon stage est le département STIC et plus particulièrement au sein du laboratoire LAB-STICC qui est un laboratoire de recherche multidisciplinaire dans lequel les chercheurs travaillent autour d'un même thème : « du capteur à la connaissance ». Ce laboratoire est rattaché au centre national de la recherche scientifique (CNRS) et regroupe plusieurs écoles d'ingénieurs et université (ENSTA Bretagne, IMT Atlantique, ENIB, UBO, UBS).

Le laboratoire compte au total, sur l'ensemble de ces sites répartis en Bretagne, plus de 500 personnes. Lors des cinq dernières années, c'est plus de 700 articles et 40 brevets qui ont été rédigés et conçus au sein de ce laboratoire.

2. Cahier des Charges Fonctionnel

2.1.Contexte et présentation générale du problème

Le secteur de la robotique mobile autonome est en plein expansion : bien qu'il n'en soit qu'à ses prémices, les robots mobiles autonomes permettent déjà de répondre à plusieurs problématiques industrielles.

Ainsi, ces robots sont déjà présents au sein de nombres d'entreprises. Ils permettent de réaliser des tâches plus rapidement et plus efficacement que des humains normaux. Ils sont souvent utilisés pour réaliser des tâches à faibles valeurs ajoutée comme par exemple transporter des marchandises.

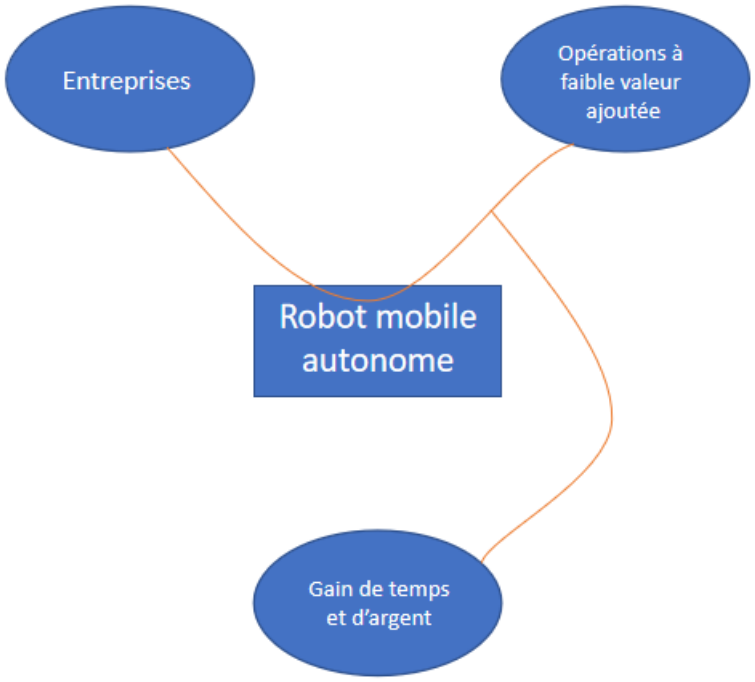


Figure 1 Diagramme "bête à corne" robotique autonome

De plus, la commercialisation de ces robots pour des particuliers représente déjà un marché juteux. D'après l'institut d'études de marché GFK, en 2019 le secteur de la domotique représentait à lui seul 790 millions d'euros en France. Qui n'a jamais rêvé d'avoir un robot qui s'occupe de toutes ses tâches domestiques, ou bien d'avoir une voiture qui se conduit tout seule ?

On en est encore loin, mais on compte nombre d'innovations ont été fait dans ce sens. Une des plus médiatisés est bien évidemment la voiture autonome qui est en train d'être développé par plusieurs sociétés comme Tesla.

Cependant la mort récente d'une personne percutée par une de ces voitures a relancé les interrogations de l'opinion publique vis-à-vis de l'intégration de ce type de robots dans notre quotidien.

Ces robots autonomes représentent sûrement le futur. Ils permettent aux entreprises un gain de temps et d'argent, mais aussi, ils peuvent proposer différents services pour particuliers. Cependant, la crainte d'une IA dangereuse et les questions d'éthiques posée par les décisions que doivent prendre les machines sont des choses sur lesquelles il faut réfléchir. C'est à nous, ingénieurs, de proposer des solutions innovantes permettant de répondre à l'ensemble de ces interrogations.

2.2.Expression Fonctionnelle du Besoin

Comme on l'a vu précédemment, ces robots peuvent intéresser aussi bien les entreprises que les particuliers, et peuvent effectuer plusieurs tâches différentes. On va donc s'intéresser aux besoins fondamentaux auxquels vont répondre ces robots.
Tout d'abord,

Tout d'abord, pour que ces robots puissent intéresser des entreprises, il faut qu'ils soient rentables. Ainsi, il faut qu'ils aient des faibles coûts de développement et qu'ils puissent remplacer véritablement le travail des salariés. Certains robots mobiles autonomes déjà développés et utilisés en entreprises peuvent travailler en continu, en se rechargeant lorsqu'ils n'ont plus de tâches, ils remplacent jusqu'à deux salariés à plein temps et leur coût peut être amorti en une année.

Il faut ensuite que ces robots soient sûrs, qu'ils puissent évoluer au côté des humains. Par exemple qu'il puisse les éviter, ou bien s'arrêter lorsque quelqu'un trébuche devant lui. Ils doivent aussi respecter des normes relatives aux robots qui travaillent avec des travailleurs humains (qui concernent des caractéristiques techniques et les consignes à respecter lorsqu'ils évoluent au côté d'humains).

Enfin, il faut que ces robots puissent s'intégrer facilement au sein d'une entreprise, ce qui fait écho au point précédent puisque ceci nécessite qu'ils ne mettent pas en danger la vie des opérateurs. Leur intégration doit se faire via des logiciels assez simplistes pour que les personnes non formées sur ce type de machines puissent les faire fonctionner facilement.

2.3.Contraintes de conception

De manière générale, puisque ces robots doivent évoluer dans un environnement qui leur est hostile et auquel ils doivent s'adapter les contraintes de conception vont principalement se trouver au niveau de la conception du logiciel. Il faut prendre connaissance de l'environnement d'évolution du robot et l'adapter en conséquence.

De plus, dans certains environnements très hostiles qui implique l'application de contraintes mécaniques sur le robot, il faudrait penser à adapter sa robustesse.

3. Recherche d'architectures et choix d'une solution

3.1.Présentation du projet et des logiciels et systèmes d'exploitation utilisés

Mon projet était divisé en deux parties : la partie simulation du robot, c'est-à-dire créer sur un logiciel de simulation un robot et l'environnement dans lequel il va évoluer, puis la partie code pur. Le but était de simuler un robot autonome capable d'effectuer un parcours tout en évitant les obstacles sur son chemin.

Pour la partie simulation j'ai utilisé le logiciel V-REP qui est simulateur robotique open source qui contient un environnement de développement. De plus, ce logiciel dispose d'une grande

communauté, ce qui facilite grandement la mise en place de la simulation puisque nombre de tutoriels sont disponibles sur Internet.

Pour la partie codage, j'ai utilisé le middleware ROS (robot operating system) et j'ai programmé en C++. ROS va permettre la communication entre les différentes tâches. Les deux principaux modes de communications entre tâche sont publisher/subscriber et Client/Server. Dans mon cas, j'ai seulement utilisé une communication de type publisher/subscriber. Le publisher va envoyer des données sur un topic que le subscriber va recevoir (cette opération s'effectue sans garantie de réception des données). Les données récupérées sont ensuite traitées au sein de nœuds (un nœud est une instance d'un exécutable, qui peut correspondre par exemple à un algorithme de traitement). Tout comme V-REP, ROS bénéficie aussi d'un très grand nombre d'utilisateurs et d'un site avec plusieurs tutoriels disponibles.

3.2.Choix d'une solution

Pour simuler le robot et l'environnement dans lequel celui-ci va évoluer, j'ai récupéré un robot déjà créé et ensuite, j'ai ajouté des obstacles de géométrie différentes : 3 obstacles cubiques de couleur orange et 4 obstacles sphériques de couleur verte. Enfin, j'ai ajouté des waypoints sous la forme de cercles bleus au sol. Pour finir sa mission, le robot devra rejoindre tous les waypoints dans un ordre défini tout en évitant les différents obstacles qui sont sur son chemin.

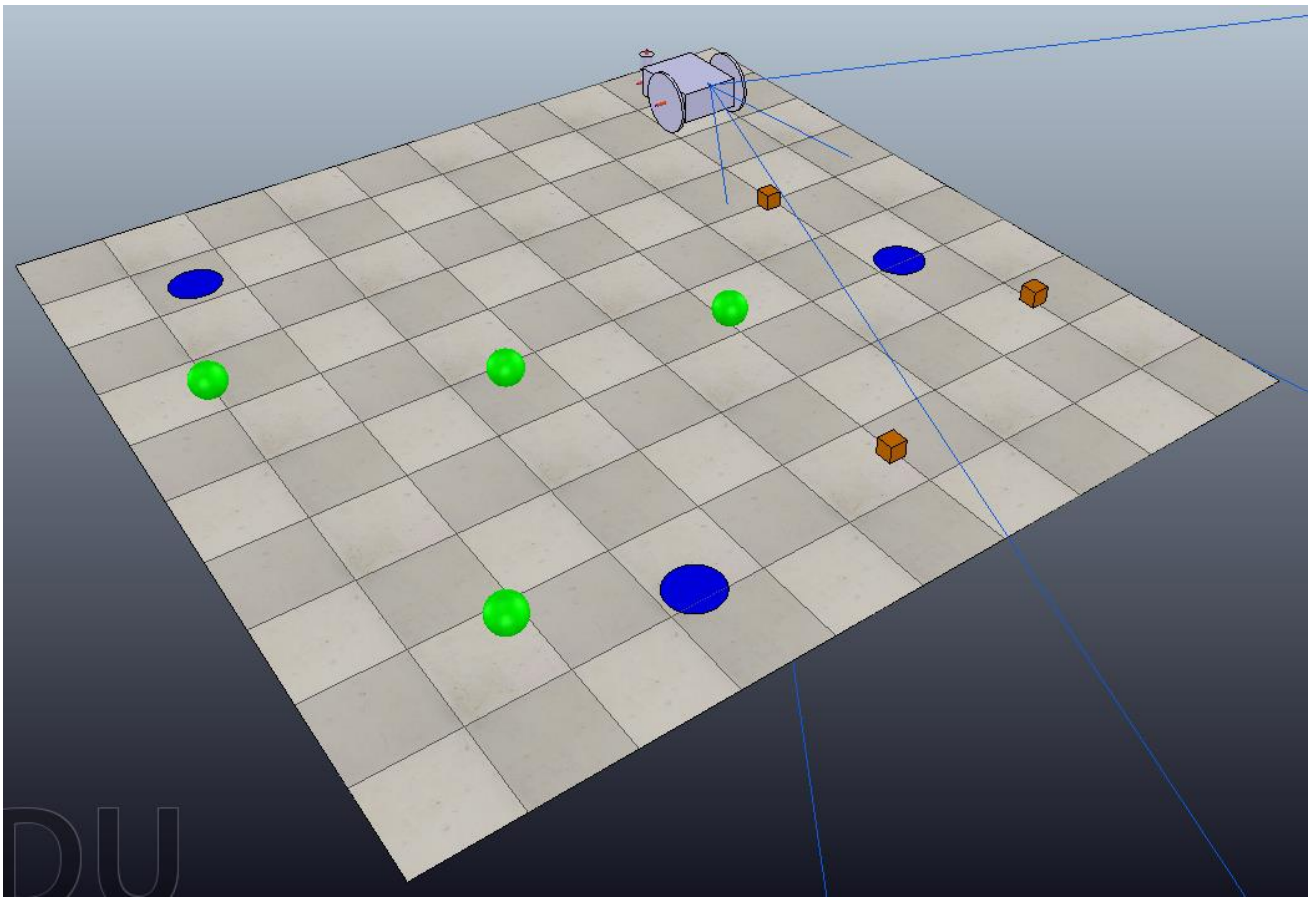


Figure 2 Simulation du robot et de son environnement sur le logiciel V-REP

Enfin, pour pouvoir traiter les données envoyées par le GPS et la caméra, j'ai conçu plusieurs nœuds. Un nœud qui va bruiter les mesures du GPS, un nœud qui va traiter ces mesures bruitées et faire rejoindre le robot en rejoignant les obstacles et un nœud qui va traiter les images de la caméra pour détecter les obstacles et leurs positions

3.3.Déroulé du projet

Le projet a été divisé plusieurs objectifs atteignables :

1. *Mis en place de la communication entre V-REP et ROS*

   Une que la scène V-REP est créée, il faut que le robot puisse envoyer les informations de ses différents capteurs sur ROS pour les traiter et il faut en retour qu'on puisse commander les moteurs avec ROS, il faut donc établir une communication entre V-REP et ROS. Ainsi, on va créer des publishers pour les données GPS et pour l'image et un subscriber pour la commande des moteurs.

2. *Modélisation des capteurs*

   Les données relatives aux différents capteurs du robot envoyées par V-REP sont parfaites. Il faut donc les bruitées pour que la simulation soit réaliste. Pour cela, j'ai créé un nœud qui va récupérer les données envoyées par V-REP et leur ajouter un bruit gaussien.

3. *Faire le parcours sans obstacles*

   Dans un premier temps, il faut que le robot puisse aller à un waypoint sans obstacles.

4. *Faire le parcours avec les obstacles en connaissant leur position*

   Une fois que le robot arrive à faire le parcours sans obstacles, on fait en sorte qu'il arrive à faire le parcours avec les obstacles en connaissant au préalable leur position.

5. *Identifier les obstacles dans les images et leur position*

   Une fois que le robot arrive à faire le parcours avec les obstacles, on va traiter les images reçus pour identifier les obstacles et déterminer leur position.

6. *Réaliser la mission en autonomie totale*

4. Description de la solution retenue

4.1.Filtrage des données

Pour effectuer le filtrage des données, j'ai créé un nœud dans lequel on va se subscribe aux topics sur lesquelles sont publiés les données GPS.

```
ros::Subscriber pose_robot = n.subscribe("pose", 1000, chatterCallback);
```

Figure 3 Codage du subscriber

Cela va permettre de récupérer les données envoyées par le GPS via la fonction chatterCallback().

```
void chatterCallback(const geometry_msgs::Pose::ConstPtr &msg)
{
    poseRob1.position.x = msg->position.x;
    poseRob1.position.y = msg->position.y;
    poseRob1.position.z = msg->position.z;

    poseRob1.orientation.x = msg->orientation.x;
    poseRob1.orientation.y = msg->orientation.y;
    poseRob1.orientation.z = msg->orientation.z;
    poseRob1.orientation.w = msg->orientation.w;
}
```

Figure 4 Codage de la fonction chaterCallback()

Ensuite, comme présenté sur la figure ci-dessous on va bruiter manuellement les mesures avec un bruit gaussien, puis on va publier les données bruitées sur un topic dédié. Ces données bruitées vont permettre de modéliser un capteur réel qui n'est pas parfait et vont être traitées dans un autre nœud.

```
poseRob1.position.x = poseRob1.position.x + ((double)rand() / (double)RAND_MAX)*0.0002-0.0001;
poseRob1.position.y = poseRob1.position.y + ((double)rand() / (double)RAND_MAX)*0.0002-0.0001;
poseRob1.position.z = poseRob1.position.z + ((double)rand() / (double)RAND_MAX)*0.0002-0.0001;

poseRob1.orientation.x = poseRob1.orientation.x +((double)rand() / (double)RAND_MAX)*0.00000000002-0.00000000001;
poseRob1.orientation.y = poseRob1.orientation.y +((double)rand() / (double)RAND_MAX)*0.00000002-0.00000001;
poseRob1.orientation.z = poseRob1.orientation.z +((double)rand() / (double)RAND_MAX)*0.0000002-0.0000001;
poseRob1.orientation.w = poseRob1.orientation.w +((double)rand() / (double)RAND_MAX)*0.002-0.001;
```

Figure 5 Codage du bruit gaussien pour la position et l'orientation du robot

Concernant l'image, le bruit gaussien sera directement appliqué dans le nœud dédié à son traitement via une fonction implémentée dans OpenCv.

```
cv::GaussianBlur(cv_ptr->image, cv_ptr_blurred, cv::Size(5, 5), 0);
```

Figure 6 Codage du bruit Gaussien pour les images de la caméra du robot

4.2.Traitement de l'image

Concernant le traitement de l'image de la caméra, je me suis appuyé sur le tutoriel fourni par ROS. La structure de base du noeud était ainsi déjà codé, il me restait plus qu'à me subscribe au topic sur lequel était publié l'image et à la traiter.

Avant de chercher à identifier les obstacles au sein des images de la caméra, il a fallu flouter l'image de la caméra pour correspondre à la réalité.

```
cv::GaussianBlur(cv_ptr->image, cv_ptr_blurred, cv::Size(5, 5), 0);
```

Figure 7 Code du flou de la caméra

1. *Traitement colorimétrique de l'image*

Pour traiter l'image dans le but d'identifier les obstacles, j'ai utilisé deux approches. Une approche colorimétrique : les obstacles étant de couleur définie, il suffisait d'identifier ces couleurs dans l'image fournie par la caméra.

L'identification des couleurs a été réalisée par une fonction de OpenCv. Comme montré sur l'image ci-dessous, il a d'abord fallu convertir l'image dans le bon format via la fonction cvtColor() pour avoir le résultat escompté.

```
cvtColor(cv_ptr_blurred, hsv, cv::COLOR_BGR2HSV);

inRange(hsv, cv::Scalar(36, 0, 0), cv::Scalar(86, 255, 255), cv_ptr_output1); // détection de la couleur verte

inRange(hsv, cv::Scalar(10, 100, 20), cv::Scalar(25, 255, 255), cv_ptr_output2); // détection de la couleur orange
```

Figure 8 Code permettant d'identifier les différentes couleurs de l'image

Cette solution a été plutôt fructueuse, puisque les obstacles étaient clairement identifiés comme le montre les screens ci-dessous.

2. *Traitement de l'image via ses contours*

Une deuxième solution pour trouver les obstacles consiste à passer pour les contours de l'image. Un code open-source permet de réaliser cette opération.

```
//détection de contours

cvtColor(cv_ptr_blurred, gray1, cv::COLOR_BGR2HSV);
Canny(gray1, canny_output1, thresh, thresh*2);
vector<vector<cv::Point> > contours;
vector<cv::Vec4i> hierarchy;

findContours( canny_output1, contours, hierarchy, cv::RETR_TREE, cv::CHAIN_APPROX_SIMPLE );
cv::Mat drawing = cv::Mat::zeros( canny_output1.size(), CV_8UC3 );

for( size_t i = 0; i< contours.size(); i++ )
{
    cv::Scalar color = cv::Scalar( rng.uniform(0, 256), rng.uniform(0,256), rng.uniform(0,256) );
    drawContours( drawing, contours, (int)i, color, 2,cv::LINE_8, hierarchy, 0 );
}
```

Figure 9 Code de détection des contours

3. *Résultats*

Finalement, avec ces deux méthodes différentes, on n'obtient pas tout à fait le même résultat. Avec une vision du robot comme celle-ci-dessous :

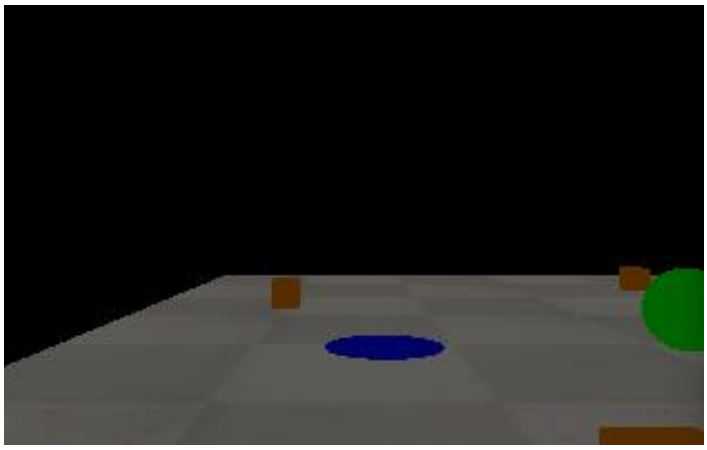


Figure 10 Vision du robot

On obtient ces résultats-ci avec l'approche colorimétrique :

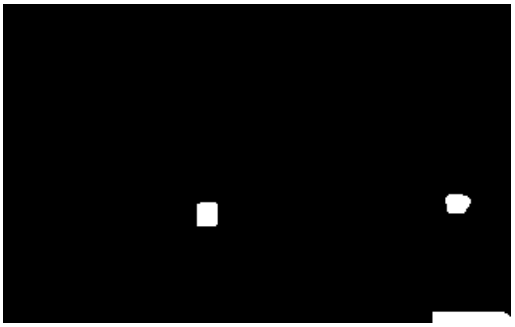


Figure 11 Identification de l'orange

Figure 12 Identification du vert

Tandis qu'avec l'approche de la détection seule des contours via les différentiels de couleurs on obtient :

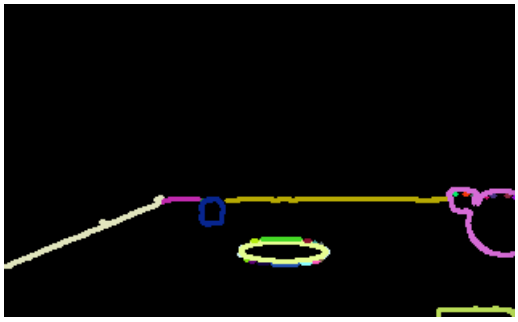


Figure 13 Détection des contours de l'image

On voit bien que la détection des contours est moins efficace puisqu'elle détecte tous les contours de l'image et donc ceux non désirés.

La détection des couleurs est quant à elle plus efficace ici puisqu'on connaît directement la couleur des obstacles. Cependant, dans la vie réelle, il faudra sûrement privilégier le deuxième choix puis traiter les contours obtenus pour analyser l'image.

4.3.Traitement des données

Dans cette partie, je vais expliquer comment j'ai traité les données GPS bruités de manière que le robot puisse effectuer l'ensemble du parcours avec les obstacles tout en connaissant leur position. Ce traitement des donnés GPS est totalement indépendant du traitement d'image et est fait un seul nœud.

A. Rejoindre les waypoints

1. Fonctions permettant au robot de se déplacer

Avant de créer les fonctions qui vont permettre de rejoindre les waypoints, il faut coder les fonctions qui vont permettre au robot d'avancer de tourner et de s'arrêter.

Pour cela, on créer un publisher qui va publier sur le topic relatif aux commandes moteurs les différentes instructions.

Une fois ce publisher créé, on va initialiser un message contenant la commande à envoyer ce moteur. Ce message va être modifié par différentes fonctions : une pour avancer tout droit, une pour tourner à gauche et une autre pour tourner à droite.

```
//Fonction permettant au robot d'avancer
void avancer(geometry_msgs::Twist &msg)
{
    msg.angular.z = 0;
    msg.linear.x = 0.5;
}

//Fonction permettant au robot de s'arrêter
void arreter(geometry_msgs::Twist &msg)
{
    msg.linear.x = 0;
    msg.linear.y = 0;
    msg.linear.z = 0;
    msg.angular.x = 0;
    msg.angular.y = 0;
    msg.angular.z = 0;
}

//Fonction permettant de tourner à droite
void tournerdroite(geometry_msgs::Twist &msg)
{
    msg.linear.x = 0;
    msg.angular.z = 0.5;
}

//Fonction permettant de tourner à gauche
void tournergauche(geometry_msgs::Twist &msg)
{
    msg.linear.x = 0;
    msg.angular.z = -1;
}
```

Figure 14 Codage des fonctions permettant de s'arrêter, avancer et tourner

2. *Fonctions permettant au robot de rejoindre les waypoints*

Maintenant que l'on peut faire avancer et tourner le robot, il faut faire en sorte qu'il rejoigne les différents waypoints. Le principe de base est simple : puisqu'on connaît la position du robot et celle des différents waypoints, il suffit de calculer la distance au waypoint et l'orientation du robot par rapport au waypoint: on oriente alors le robot pour qu'il se déplace vers le waypoint, une fois qu'il est «assez près» de celui-ci, on considère qu'on l'a atteint et on change de waypoint.

Pour faire cela, j'ai créé une liste avec l'ensemble des waypoints à rejoindre rangés dans l'ordre souhaité. Une fois cette liste créée, j'ai subscribe au topic sur lequel était publié la position et l'orientation du robot. Ensuite, j'ai créé une variable i qui détermine sur quel waypoints aller. Par exemple, si i = 1, je dois aller sur le waypoint numéro 2 de ma liste (puisque le premier correspond à 0). On regarde alors que vaut cette variable pour savoir si l'on a rejoint tous les waypoints ou bien s'il en reste. Si on a rejoint tous les waypoints, le robot s'arrête. Sinon, on regarde quel waypoint on doit rejoindre : on calcule la distance au waypoint numéro i, si cette distance est faible (inférieure à une constante déterminée expérimentalement) on passe au waypoint i+1, sinon on reste sur le waypoint i.

On regarde ensuite le cap que le robot doit atteindre. Le cap est déterminé d'après un calcul géométrique simple. Ensuite, on calcule le cap du robot : en effet, le cap fourni par V-REP est exprimé en quaternion, il faut donc le convertir en RPY pour l'exploiter plus facilement. Une fois

que l'on a le cap du robot, il suffit de calculer l'écart au cap voulu pour savoir s'il faut que le robot tourne à gauche ou à droite. On considérera que si l'écart entre les deux caps est faible (c'est-à-dire inférieur à une certaine constante une fois de plus déterminée expérimentalement) le robot doit avancer en ligne droite.

```
if(i <=3) // Si jamais on a pas rejoint tous les waypoints
{
    if(sqrt(pow(poseBruit.position.x-waypoint[0],2)+pow(poseBruit.position.y-waypoint[1],2)) < 0.2)// Si on est sur un waypoint
    {
        i +=1;
        waypoint = route[i]; // On passe à l'autre waypoint
    }


    else // Si on est entrain de rejoindre un waypoint
    {

        double headOk = atan((poseBruit.position.y-waypoint[1])/(poseBruit.position.x-waypoint[0]))*180/PI; //On calcule le cap à avoir pour rejoindre le waypoint

        // à améliorer, problèmes car l'image de atan est entre -pi/2 et pi/2
        if( waypoint[0] < 0 )
        {
            if(headOk > 0)
            {
                headOk = headOk -180;
            }

            else
            {
                headOk = headOk + 180;
            }

        }

        //On calcul le cap du robot
        double roll, pitch, yaw;
        tf::Quaternion q(poseBruit.orientation.x, poseBruit.orientation.y, poseBruit.orientation.z, poseBruit.orientation.w);
        tf::Matrix3x3 m(q);
        m.getRotation(q);
        m.getRPY(roll, pitch, yaw);


        // On régule le robot en fonction de l'écart au cap voulu
        if(fabs(headOk-yaw*180/PI)>5)
        {
            if(headOk-yaw*180/PI < 0)
            {
                tournergauche(msg);

                //cout << "Je tourne à gauche pour me diriger vers le waypoint"<< endl;
            }

            else
            {
                tournerdroite(msg);

                //cout << "Je tourne à droite pour me diriger vers le waypoint" << endl;
            }
        }

        if(fabs(headOk-yaw*180/PI)<=5)
        {
            avancer(msg);

            //cout << "J'avance vers le waypoint"<< endl;
        }
    }
}
```

Figure 15 Code permettant de rejoindre les différents waypoints

B. Eviter les obstacles

Pour éviter les obstacles, on va partir sur le même principe que pour rejoindre les waypoints, mais au lieu de faire en sorte que le robot rejoigne les obstacles, il va devoir les esquiver.
Tout d'abord, j'ai initialisé une pile qui va contenir la position de l'obstacle à éviter. Une fois l'obstacle évité, on videra la pile. Pour déterminer si l'on doit éviter ou non un obstacle, on calculera la distance entre l'obstacle et le robot, si celle-ci est inférieure à une constante déterminée empiriquement, on commandera le robot pour qu'il ne rentre pas dans l'obstacle. Dans le cas contraire, on lui fera rejoindre un waypoint via l'algorithme décrit précédemment.

Dans le cas où l'on se trouve près d'un obstacle, on va calculer la différence entre le cap que le robot devrait avoir pour rentrer dans l'obstacle et le cap du robot. Si le cap est suffisamment

grand, on considérera que le robot ne rentrera pas dans l'obstacle et il ira en ligne droite. Sinon, on fera tourner le robot pour qu'il évite l'obstacle.

```
if(!pile.empty()) //Si on est près d'un obstacle
{

    //cout << "Je dois éviter l'obstacle "<< c << " " << endl;

    // On récupère le cap actuel
    double roll, pitch, yaw;
    tf::Quaternion q(poseBruit.orientation.x, poseBruit.orientation.y, poseBruit.orientation.z, poseBruit.orientation.w);
    tf::Matrix3x3 m(q);
    m.getRotation(q);
    m.getRPY(roll, pitch, yaw);

    double headOk = atan((poseBruit.position.y-obstacle[c][1])/(poseBruit.position.x-obstacle[c][0]))*180/PI;

    //cout << "yawb : " << yawb << " yaw :" << yaw*180/PI << " dif :" << yawb-yaw*180/PI<< endl;
    //cout << headOk-yaw*180/PI << endl;

    // L'ensemble des boucles if permet au robot de tourner à 60 degrés dès qu'il détecte l'obstacle et ensuite d'avancer tou

    if(fabs(headOk-yaw*180/PI)<30)
    {
        if(headOk-yaw*180/PI<0)
        {
            tournerdroite(msg);
        }
        else
        {
            tournergauche(msg);
        }

        //cout << "Je tourne pour éviter l'obstacle" << endl;
    }

    else
    {
        avancer(msg);
        //cout << " J'avance pour éviter l'obstacle" << endl;
    }

    if( sqrt(pow(poseBruit.position.x-obstacle[i][0],2)+pow(poseBruit.position.y-obstacle[i][1],2)) > 0.5)
    {
        pile.pop();
    }
```

Figure 16 Code permettant au robot d'éviter les obstacles

5. Analyse critique de la solution proposée

5.1. Analyse de l'écart entre les prestations attendues et réalisées

L'objectif principal du projet était d'avoir un robot autonome capable d'effectuer un parcours donné tout en détectant et évitant les obstacles disposés sur son chemin.
Le programme proposé doit connaître la position des obstacles pour réussir à effectuer l'ensemble du parcours, bien qu'il arrive à « reconnaître » les obstacles via sa caméra. Je vais donc donner une piste permettant d'arriver à l'objectif final.

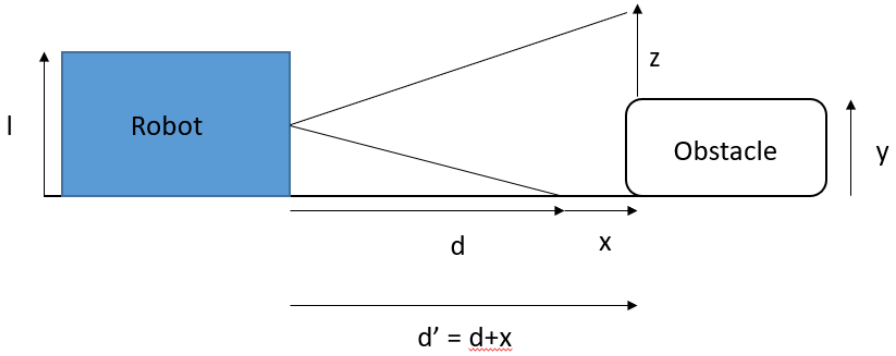
5.2.Défauts de la prestation

Tout d'abord, le fait que là l'image de la fonction arctangente ne soit pas définie sur un intervalle de longueur deux pi, mais sur un intervalle de longueur pi pose des problèmes lorsque le robot doit faire des virages de plus de pi. Ainsi, sur certaines dispositions des waypoints (lorsque la longueur du trajet n'est pas optimisée et qu'il doit faire des zigzags au lieu d'un tour), le robot s'oriente dans le mauvais sens et n'arrive pas à faire l'ensemble du trajet.

Pour régler ce problème, j'ai pensé à calculer la position à l'itération suivante pour voir si l'on se rapproche ou pas du waypoint et ensuite à faire tourner le robot de pi si l'on s'en éloignait. Malheureusement, je n'ai pas réussi à mettre cette solution en place.

5.3.Pistes

Pour arriver à la fin du projet, il faudrait trouver un moyen de relier l'identification des obstacles à la distance au robot. Pour cela, j'avais trouvé qu'on pouvait calculer la distance du robot à l'obstacle en connaissant seulement le rapport de la taille de l'objet en pixel sur la taille de l'image comme le montre le schéma ci-dessous.



D'après le théorème de Thalès:
d' = z*d/y
Aussi on a:
d = l/tan(30)

Figure 17 Figure montrant le calcul théorique de la distance du robot à l'obstacle

Puisque l est une donnée constructeur, on voit bien qu'il suffit de trouver le rapport z/y pour trouver la distance du robot à l'objet projetée dans un plan de coupe perpendiculaire au sol. On applique le même raisonnement au plan de coupe transverse et on voit qu'on peut positionner l'obstacle par rapport au robot si l'on arrive à déterminer ces rapports.

Il reste donc à trouver une méthode pour calculer ce rapport. Une fois ce rapport trouvé, on calcule dans le nœud du traitement de l'image la distance du robot à l'objet que l'on publie sur

un topic. On récupère ensuite cette distance sur le nœud gérant le parcours du robot et on aura un robot totalement autonome.

6. Conduite du projet

6.1.Travail en autonomie

Du fait du contexte sanitaire particulier, ce stage a été réalisé en télétravail et j'étais en totale autonomie.
Ainsi, c'était à moi d'avoir une réflexion sur l'organisation de mon travail et les deadlines que je devais m'imposer.

J'ai choisi de travailler tous les jours, sauf le week-end et de garder un rythme « scolaire », c'est-à-dire éviter de se lever trop tard et de finir de travailler trop tard, afin d'être toujours en forme. Cela m'a permis de fournir un travail régulier et de ne pas être dans le rush sur les dernières semaines.

Le plus dur pour moi cependant a été de m'imposer des bons deadlines. En effet, si jamais on n'est pas assez ambitieux, on risque de ne pas travailler et d'être improductif, tandis que si l'on est trop ambitieux, on risque d'être déçu de ne pas arriver à ses objectifs.

Dans mon cas, aucune des deadlines que je m'étais fixé ont était respectés, j'ai été beaucoup trop ambitieux et je me suis surévalué. Cela a causé des grandes baisses de motivations et des moments de remise en question.

Finalement, j'ai beaucoup appris de ce travail en autonomie et je saurais la prochaine mieux gérer les objectifs que je me donne.

6.2.Analyse critique de l'organisation de l'équipe

Au cours de ce projet, j'ai rencontré plusieurs problèmes. Le plus important a été dès le début du stage. Lorsque j'ai voulu lancer V-REP, celui-ci ne fonctionnait pas. S'en ai alors suivi une semaine où j'ai formaté à plusieurs reprises mon ordinateur pour tester plusieurs versions d'ubuntu. Finalement, j'ai dû formater aussi la partie de mon disque contenant Windows pour que ça marche. J'ai bien perdu une semaine de projet, et c'était assez frustrant. Ensuite, j'ai souvent perdu des journées de travail à cause d'erreurs d'étourderies que je n'arrivais pas à trouver.

Tous ces problèmes auraient pu être résolus plus rapidement si j'avais pu avoir un retour instantané de mon tuteur de stage, malheureusement le télé travail nous forçait à communiquer par mail et donc souvent je devais attendre sa réponse avant d'avancer. Cependant, on peut nuancer ces propos car cela m'a appris à me débrouiller vraiment tout seul et à ne pas lâcher le morceau lorsque j'avais une erreur que je n'arrivais pas à trouver. Cela m'a aussi appris à me remettre en question et à bien relire mon code, j'ai aussi fait beaucoup d'erreurs d'étourderies que je ne referais plus.

Conclusion

Pour conclure, j'ai répondu en parti au cahier des charges qui m'était imposé : j'ai réussi à effectuer une simulation du robot et de son environnement sur le logiciel V-REP. J'ai ensuite utilisé cette simulation pour développer un logiciel permettant au robot de se déplacer de waypoint en waypoint tout en évitant les obstacles dont il connaît la position et qui va détecter les différents obstacles présents sur la route via une analyse des images de la caméra.

Pour répondre à la totalité des contraintes du cahier des charges, il manque principalement à relier l'identification des obstacles à la distance aux obstacles du robot. On peut aussi améliorer le programme en corrigeant les défauts précédemment cités.

Finalement, ce stage m'a permis de mobiliser mes connaissances apprises au sein de l'ENSTA Bretagne pour réaliser un projet concret qui se rapproche de ce qu'on pourrait me demander de faire en entreprise. Il m'a aussi permis d'apprendre à travailler en autonomie sur des gros projets qui demandent organisation et rigueur.

Table des figures