

Rapport de plateau projet

Filière

Mécatronique options

macromécatronique et micromécatronique

Sujet

Coopération Homme-Robots:

Implémentation et test de nouveaux algorithmes,
de planification de trajectoire et de localisation

Soutenu par :
FASQUELLE Cédric
GOIDIN Aymeric
ZHAO Zhenjie
FARSI Nassuf
ASRY Karim
XUAN Tianyun

Sous la direction de:
Mme PACAUX-LEMOINE Marie-Pierre
Mme. CHAABANE Sondès

TABLE DES MATIÈRES

0.Remerciement.....	p3
1.Contexte du projet.....	p3
2.Partie video.....	p4
3.Partie slam.....	p14
4.Partie communication.....	p27
5.Problèmes et Pistes.....	p38
6.Conclusion.....	p39
7.Bibliographie.....	p40

1) REMERCIEMENT

Nous voudrions exprimer notre reconnaissance envers M. Quentin BERDAL nous a apporté son soutien moral et intellectuel tout au long du projet.

Enfin, nous tenons à témoigner toute notre gratitude à Mme PACAUX-LEMOINE Marie-Pierre pour sa compréhension, sa confiance et son soutien durant le projet.

2) CONTEXTE

En effet les robots aujourd'hui transmettent des données imprécises au poste de supervision quant à leur position en temps réel. Ce problème est dû au fait que certains facteurs comme les frottements, les glissements, etc ne sont pas pris en compte lors du calcul de la position.

La solution pour ce problème a été d'utiliser des codes SLAM: on utilise les caméras des smartphones afin de visionner l'environnement et de tracer une carte où le robot se déplacerait tout en affichant les caractéristiques de son environnement.

Nous devons améliorer ces programmes afin de permettre un partage de la carte en temps réel, optimiser le rendu graphique de la carte partagée tout en repositionnant les robots s'ils déviaient de leur trajectoire.

Pour se faire nous avons décidé d'utiliser des QR codes comme points remarquables, points que la caméra reconnaîtrait et saurait à quelle distance elle se situe par rapport à eux et à quel angle.

Nous allons donc essayer d'implémenter ce système de QR codes tout en améliorant la communication, afin d'améliorer le système SLAM actuel et permettre le partage d'une carte représentant l'environnement des robots en temps réel

3) PARTIE VIDÉO

Dans cette partie, l'objectif principale était de mettre en place une technologie qui permet au robot l'identification des points spéciaux. Dans notre projet on a opté d'utiliser le QR code (Quick Response), mais plusieurs solutions sont envisageables.

Dans un premier temps, nous allons définir les avantages de la technologie QR code dans notre application.

Ensuite, nous détaillerons les étapes qu'on a fait afin de déterminer la distance entre la caméra du robot et le QR Code.

1. Introduction

Les QR codes sont utilisés partout où les données doivent être lues rapidement. La technologie QR Code présente plusieurs avantages, on cite les suivants:

- Mise en oeuvre rapide et facile
- Enregistrer une information conséquente dans un espace réduit
- Lecture et traitement rapide
- Compact et flexible
- Peu cher

Pour ce projet, on a utilisé un QR Code de dimension 6.3x6.3 cm.

2. Détermination de la distance entre le Robot et le QR code

2.1. Calibration de la caméra

La calibration de la caméra est une étape primordiale afin de déterminer la distance entre le robot et le QR code. En effet, les photos d'une caméra non calibrée présente de distorsion.

L'objectif de cette partie est d'éliminer la distorsion de l'image afin d'obtenir un résultat correct.

Dans cette partie on essayera d'éliminer la distorsion, principalement la distorsion tangentielle et la distorsion radiale. Les lentilles des caméras, par leur symétrie sphérique, provoquent une distorsion, c'est-à-dire une déformation de l'image réelle. On décompose la distorsion en une composante radiale et une composante tangentielle. La première est due à l'asymétrie des lentilles, la deuxième au mauvais alignement des lentilles.

Il est possible de corriger l'image captée si l'on possède les paramètres liés à la caméra (les paramètres intrinsèques), on peut alors reformer l'image par interpolation des pixels ayant subi préalablement un déplacement inverse à celui de la distorsion.

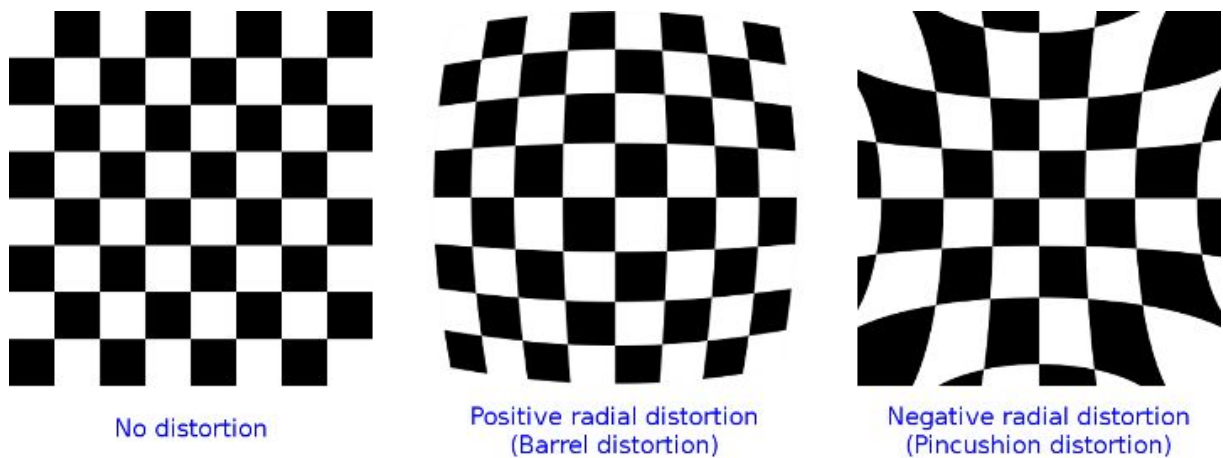


Fig 1: Les différents types de distorsion

Pour chaque point de pixel de coordonnées (x,y) , après distorsion sa position dans l'image est (X_{dis}, Y_{dis}) .

Pour corriger la distorsion radial, on va utiliser les formules suivantes:

$$\begin{aligned}
 x_{distorted} &= x(1 + k_1r^2 + k_2r^4 + k_3r^6) \\
 y_{distorted} &= y(1 + k_1r^2 + k_2r^4 + k_3r^6)
 \end{aligned}$$

De même, pour la correction de la distorsion tangentielle, on utilise les formules suivantes:

$$\begin{aligned}
 x_{distorted} &= x + [2p_1xy + p_2(r^2 + 2x^2)] \\
 y_{distorted} &= y + [p_1(r^2 + 2y^2) + 2p_2xy]
 \end{aligned}$$

On définit la variable *distorsion* qui contient les 5 paramètres inconnus qu'on doit déterminer.

$$\text{distorsion} = (k1 \ k2 \ p1 \ p2 \ k3)$$

Enfin, pour déterminer les coordonnées exacte de chaque pixel on utilise l'équation d'état suivante:

$$\begin{matrix} \begin{bmatrix} x \\ y \\ w \end{bmatrix} & = & \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} & \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \\ A & & B & C \end{matrix}$$

Avec:

- **(fx, fy)** : focale de la caméra
- **(cx,cy)** : centre optique exprimées en pixels
- **B** : matrice de la caméra

2.2 Programmation de la calibration

2.2.1 Introduction

Afin de calibrer la caméra on réalise un programme en langage Python qui permet grâce à l'utilisation de la bibliothèque OpenCV de calibrer notre caméra.

Initialement développé par Intel, OpenCV (Open Computer Vision) est une bibliothèque graphique. Elle est spécialisée dans le traitement d'images, que ce soit pour de la photo ou de la vidéo.

2.2.2 Mise en place du programme de la calibration:

Pour déterminer les paramètres mentionnés ci-dessus on a réalisé un traitement d'image sur un échantillon de 15 images d'un échiquier.

Dans un premier temps, grâce à la fonction **cv2.findChessboardCorners()** on peut déterminer les coins des échiquiers capturés par la caméra.

Une fois que nous avons trouvé les coins, nous pouvons augmenter leur précision en utilisant **cv2.cornerSubPix ()**. Nous pouvons également dessiner le motif en utilisant **cv2.drawChessboardCorners ()**. Toutes ces étapes sont incluses dans le code ci-dessous:

```
ret, corners = cv2.findChessboardCorners(gray, (7,6),None)

# correction de la précision
if ret == True:
    objpoints.append(objp)

    corners2 = cv2.cornerSubPix(gray,corners,(11,11),(-1,-1),criteria)
    imgpoints.append(corners2)
```

Nous avons donc maintenant nos points d'objet et nos points d'image, nous sommes prêts à procéder à la calibration. Pour cela, nous utilisons la fonction **cv2.calibrateCamera ()**. Il renvoie la matrice de la caméra, les coefficients de distorsion, les vecteurs de rotation et de translation.

```
ret, mtx, dist, rvecs, tvecs = cv2.calibrateCamera(objpoints, imgpoints, gray.shape[::-1],None,None)
```

2.2.3 Distorsion

A cette étape, on élimine la distorsion grâce à la fonction **cv2.undistort()**.

```
dst = cv2.undistort(img, mtx, dist, None, newcameramt)
```

2.2.4 Erreur de projection

Afin de trouver une bonne estimation de l'exactitude des paramètres trouvés on a mis en place un code qui permet de déterminer l'erreur de projection.

Nous transformons d'abord l'objet point en point image en utilisant **cv2.projectPoints ()**. Ensuite, nous calculons la norme absolue entre ce que nous avons obtenu avec notre transformation et l'algorithme de recherche de coin. Pour trouver l'erreur moyenne, nous calculons la moyenne arithmétique des erreurs calculées pour toutes les images d'étalonnage.

```

mean_error = 0
for i in range(len(objpoints)):
    imgpoints2, _ = cv2.projectPoints(objpoints[i], rvecs[i], tvecs[i], mtx, dist)
    error = cv2.norm(imgpoints[i],imgpoints2, cv2.NORM_L2)/len(imgpoints2)
    tot_error += error

print ("total error: ", error/len(objpoints))

```

1) Méthode d'analyse vidéo

1. Qu'est-ce qu'OpenCVSharp

Afin de résoudre le problème de l'écriture de programmes Opencv sous C#, nous avons choisi d'utiliser OpenCVSharp. Du point de vue temporel, il a été développé depuis plus longtemps, est plus complet et stable, et peut être utilisé directement comme référence ou directement.

OpenCVSharp est développé par un ingénieur japonais. L'adresse du projet est: <https://github.com/shimat/opencvsharp>. Il s'agit du wrapper .NET d'Opencv, qui est plus proche de l'Opencv original qu'Emgucv, et possède de nombreuses références d'exemples. Il est publié par LGPL et est convivial pour les applications commerciales.

2. Pourquoi nous choisissons d'utiliser OpenCVSharp

Lors du développement de l'interface utilisateur, nous avons deux façons de l'implémenter. La première consiste à utiliser QT, qui utilise le langage C++. Cependant, pendant le processus de développement, la fonction Bluetooth n'a pas pu être déboguée, nous avons donc choisi de changer la direction et utiliser Visual studio et C # à l'objectif de créer une interface interactive.

Dans ce cas-là, nous devons porter le programme C++ vers l'interface C #, et nous avons envisagé deux méthodes :

- 1) Utilisez Dll (Dynamic Link Library) pour porter les programmes C ++ vers l'interface C #
- 2) Réécrivez le programme via OpenCVSharp dans l'interface C #

3. Quelles sont les caractéristiques d'OpenCVSharp

- Encapsuler directement plus de méthodes Opencv, ce qui facilite son utilisation
- La plupart des fonctions héritent de l'interface IDisposable, ce qui est pratique pour utiliser l'instruction using

- Vous pouvez appeler directement la méthode Opencv de style original
- L'objet image peut être directement converti en Bitmap utilisé par GDI et WriteBitmap de WPF

2) Programme d'analyse vidéo

1. Importer les bibliothèques

Nous ajoutons les trois bibliothèques au tout début de programme comme le figure ci-dessous. Le bibliothèque RegularExpressions nous permet de filtrer les différentes types d'informations. Le Timers nous donne une moyenne de répéter certaines fonctions avec une delay fixée.

```
using System.Text.RegularExpressions; //Regular Expression
using OpenCvSharp;
using System.Timers;
```

2. Déclarer les variables

Nous déclarons les variables nécessaires, ces trois variables et fonctions en couleur cyan sont d'origine de la bibliothèque OpenCVSharp.

VideoCapture est une méthode pour obtenir le flux de vidéo.

Mat est une variable spécifique dans OpenCVSharp, qui fonctionne comme une matrice.

QRCodeDetector s'applique sur une variable de type Mat et nous fournit les coordonnées de quatre sommets d'un QR code si il y en a un dans cette matrice.

La variable focale est obtenue selon la calibration de la caméra. Normalement nous prenons la valeur du fx dans la matrice de calibration. Dans notre cas, nous travaillons avec une résolution de 960*720, et la calibration est fait sous une résolution de 320*260, du coup nous multiplions la valeur fx par 3 et nous obtenons 1050 comme un coefficient constant.

```
//creat webcamera connection
VideoCapture cameraNXT1 = new VideoCapture();
QRCodeDetector qrdecoderNXT1 = new QRCodeDetector();
Mat frameNXT1 = new Mat();
//
const double foucale = 1050;
```

3. Détection du QR code

Nous créons un timer qui rappelle notre programme de détection toutes les 5 secondes.

```
//gestion du OpenCV_Timer
OpenCV_Timer.Interval = (5 * 1000); //5s
OpenCV_Timer.Tick += new System.EventHandler(OpenCv_TimerTick);
OpenCV_Timer.Start();
```

Nous précisons l'adresse IP de la caméra, et stockons les images captées dans Mat frame. Nous appliquons la méthode QRCodeDetector à la variable frame chaque fois qu'il y a une nouvelle image. Normalement, ce flux de vidéo est de 30FPS, c'est-à-dire 30 images par seconde, mais il peut aussi travailler sous 60FPS, ceci dépend de la caméra.

```

cameraNXT1.Open("http://192.168.0.100:8080/video");

if (cameraNXT1.IsOpened())
{
    Console.WriteLine("NXT1 open");
    int essais = 0;
    while (essais!=10)
    {
        cameraNXT1.Read(frameNXT1);
        string decodeDataNXT1 = qrdecoderNXT1.DetectAndDecode(frameNXT1, out Point2f[] bbox);
        if (decodeDataNXT1.Length > 0)

```

Les informations codées dans le QR code vont être stockées dans le string decodeDataNXT1, nous utilisons une Expression régulière au vu de retirer les valeurs souhaitées et les mémoriser dans une array.

```

Regex rg = new Regex(@"\d+");
List<string> arr = new List<string>();
MatchCollection mc = rg.Matches(decodeDataNXT1);
foreach (Match i in mc)
    arr.Add(i.ToString());

```

4. Calculer la distance est l'angle

La fonction DistanceTo est une méthode qui calcule la distance entre deux points. Nous calculons la distance réelle par l'équation suivante:

$$\frac{f}{d} = \frac{p}{x}$$

Le rapport entre la focale et la distance mesurée est équivalent au rapport entre la longue pixelle et la longue réelle d'un certain segment.

```

//Calcule de la distance
double lengthPixl = bbox[1].DistanceTo(bbox[2]);
double lengthReel = Convert.ToInt32(arr[1]);
double distance = (foucale * lengthReel) / lengthPixl;

```

Au vu de calculer l'angle, nous réalisons la projection d'un côté de QR code sur le plan horizontal, et le rapport de la longue sur axe Y et la longue sur axe X nous donne la valeur tangent d'angle rotation. Puis nous appliquons la fonction Atan de la bibliothèque Math pour calculer l'angle final. Nous ajoutons à la fin une boucle if pour identifier le quadrant de notre angle, la sortie finale est dans l'intervalle (-180°,180°);

```

//Calcul de l'angle
float a=(bbox[1].X - bbox[0].X);
float b = (bbox[0].Y - bbox[1].Y);
double angle = Math.Atan(b / a)/Math.PI*180;
if (bbox[1].X<bbox[0].X)
{
    if (bbox[1].Y>bbox[0].Y)
    {
        angle = angle - 180;
    }
    else
    {
        angle = 180 + angle;
    }
}

```

5. Transmission des mesures

Après avoir obtenu les mesures souhaitées, nous mettrons à jour les valeurs dans le tableau d'affichage sur l'interface.

```

//Distance:
txtData.Rows[0].Cells[1].Value = Convert.ToInt32(distance); //NXT1
//Beta
txtData.Rows[1].Cells[1].Value = Convert.ToInt32(angle);
//QrNum
txtData.Rows[2].Cells[1].Value = arr[0];
break;

```

A la fin de tous les processus, il faudra fermer le flux d'image.

```
cameraNXT1.Release();
```

3) Résultats

Nous appliquons la matrice de la calibration au programme à l'objectif de mesurer la distance entre le QR code et la caméra.

Selon notre test, avec un QR code de taille 63mm*63mm, notre programme peut détecter le QR code si la distance entre le code et la caméra est inférieur à 750 mm. Sous cette situation, l'erreur entre la distance mesurée par notre programme et la distance réelle est limitée à ± 40 mm.

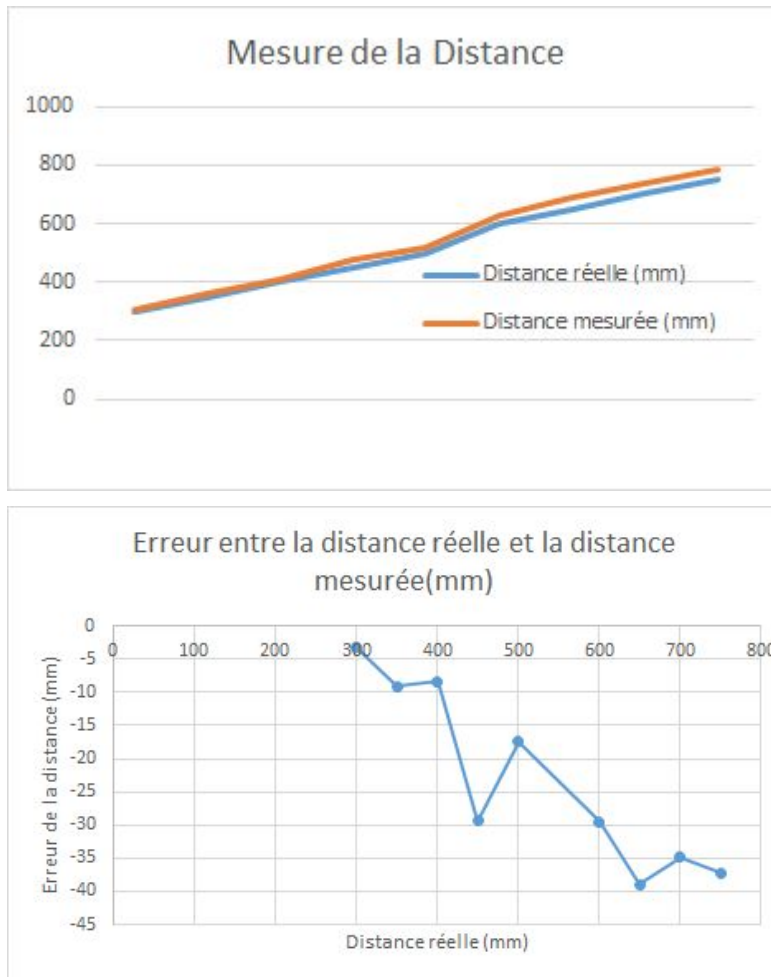


Figure: Résultats sur la mesure de la distance
Ce résultat satisfait bien la précision demandée par notre client ($\pm 50\text{MM}$).

La deuxième mesure est effectuée sur l'angle de rotation du QR code. Cette mesure a une meilleure performance que celle d'avant parce qu'elle n'est pas influencée par la calibration de la caméra. L'erreur entre l'angle mesuré par notre programme est limitée entre ± 4 degré, cette précision contente nôtre l'objectif cad une précision de ± 10 degré.

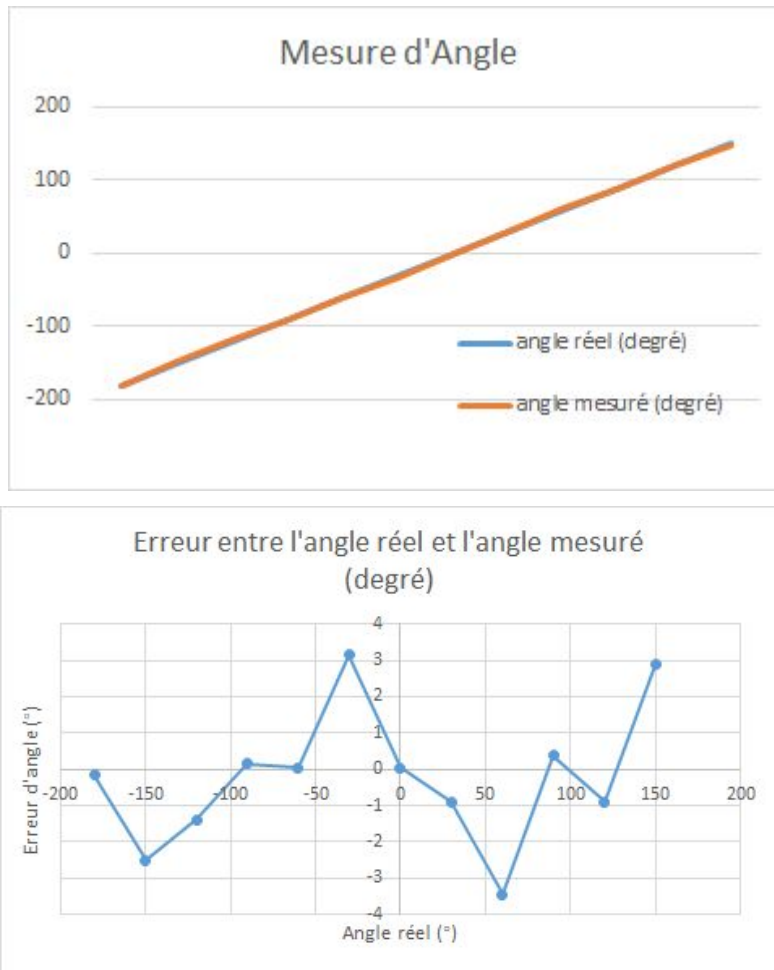


Figure: Résultats sur la mesure d'angle

Ces deux mesures vont être affichées sur l'interface commune.

4) Perspectives

Opencv fournit un traitement d'image multi-niveaux et multidirectionnel. Dans ce projet, nous n'utilisons que le détecteur QR code et l'interface Videocapture, car :

Premièrement, le QR code peut stocker des informations et on peut obtenir des informations codées supplémentaires lors de l'identification, ce qui est pratique pour la télémétrie monoculaire

Deuxièmement, les conditions de reconnaissance de QR code sont plus détendues et les exigences pour la qualité de la caméra sont moins importantes.

Troisièmement, il n'est pas nécessaire de configurer la caméra à plusieurs reprises, ce qui est facile à transplanter et à démonter.

Finalement, la routine du programme est légère et rapide.

La méthode d'utilisation des images et de la reconnaissance des couleurs est également réalisable. C'est aussi une direction qui peut se poursuivre à l'avenir.

Il est nécessaire de suivre une formation standardisée sur l'appareil photo pour éliminer les facteurs d'interférence tels que l'effet fish-eye.

De plus, il peut dériver des technologies liées à Machine-Learning et la caméra peut s'auto-corriger.

En termes de programme, comme c'était la première fois que nous utilisons C#, nous n'avons encapsulé aucune classe et toutes les routines étaient dans un seul arrangement linéaire.

Dans le futur, une partie des fonctions de base peut être encapsulée d'abord pour les rendre portables et individuellement débloquables, et deuxièmement, la programmation multithread peut être introduite pour prioriser les routines. Cette méthode nous permet de réduire le fardeau du programme, d'éviter les blocages et d'optimiser le processus.

4) PARTIE SLAM

Pour cette première partie, l'objectif était de réutiliser les programmes des années précédentes, les programmes SLAM, afin de corriger les trajectoires du robot, et de permettre une création et un partage d'une carte en 2D avec des points et des couleurs représentant les robots, les obstacles, et des points spéciaux, ici, des QR codes.

Nous allons dans un premier temps détailler différentes parties importantes du code SLAM, pour expliquer les modifications que nous devons leur apporter afin de répondre aux problèmes posés.

Ensuite nous montrerons les différents prérequis afin de lancer un code SLAM.

Puis pour finir nous montrerons les différentes modifications à apporter aux codes et où elles doivent s'implémenter.

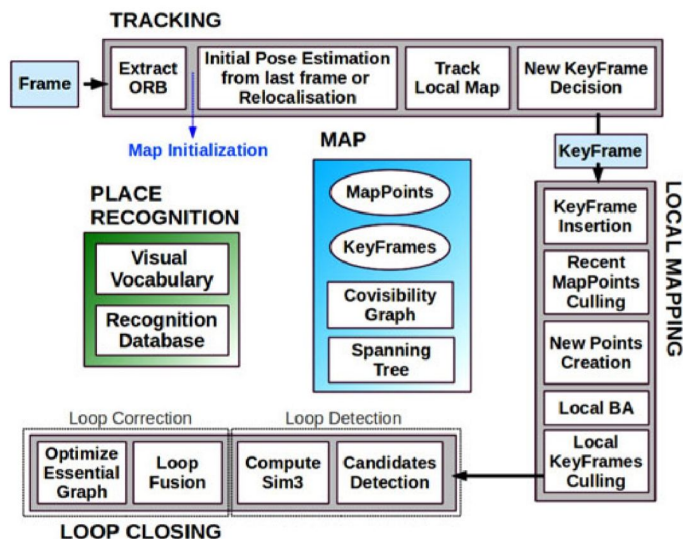
1) Introduction au processus algorithmique

SLAM signifie simultaneous localization and mapping, c'est-à-dire que c'est un système qui permet de localiser et de cartographier un environnement et ses caractéristiques. Le SLAM est principalement utilisé pour résoudre des problèmes de positionnement de robots mobiles dans un environnement inconnu. Ce qui correspond assez bien à notre cas. Les codes SLAM permettent de repérer des points et de les mettre en correspondance s'ils ont des caractéristiques assez similaires.

Nos codes SLAM viennent d'un projet open source : phdsky/ORBSLAM24Windows:ORBSLAM2 Project 4(for) Windows Platform auquel des modifications ont été apportées les années antérieures.

Nous allons détailler grâce à la thèse ci-dessous différentes parties de ces codes SLAM:

[Monocular] Raúl Mur-Artal, J. M. M. Montiel and Juan D. Tardós. ORB-SLAM: A Versatile and Accurate Monocular SLAM System. *IEEE Transactions on Robotics*, vol. 31, no. 5, pp. 1147-1163, 2015. (2015 IEEE Transactions on Robotics Best Paper Award).



L'algorithme contient 3 threads qui marchent parallèlement: Tracking, Local Mapping et Loop Closing. Avant tout, on met des traitements initiaux pour construire les images de cadre (Frame). Dans un cas monoculaire, les images originales récupérées par la caméra sont transmises à images gris. Puis l'algorithme construit les cadres (Frame) parmi les images. L'image de cadre entre la procession de tracking.

Thread 1:Tracking

Le travail principal dans cette partie consiste à extraire les caractéristiques ORB de l'image, à effectuer une estimation de la pose basée sur l'image précédente, ou à effectuer une initialisation des poses par repositionnement global, puis à suivre la carte locale déjà reconstruite, à optimiser les poses, et enfin à déterminer de nouvelles images clés selon certaines règles. 4 étapes sont comportées dans le travail: initialisation, suivi, repositionnement, détermination des images clés.

Si le nombre de points de caractéristiques des deux images est supérieur à 100, elles peuvent être utilisées pour l'initialisation et ensuite mises en correspondance, et si le nombre de points de caractéristiques mis en correspondance des deux

images est inférieur, les deux images sont déterminées en correspondance. L'initialisation monoculaire est effectuée par le modèle H ou le modèle F pour obtenir le mouvement relatif entre deux trames, les points sur map (Map Points) initiaux, et supprimer les points qui ne peuvent pas être triangulés pour obtenir les points sur map.

Dans le cadre du suivi, il existe trois autres types de suivi : le suivi dans vitesse uniforme, le suivi sur des images clés, le repositionnement et le suivi de cartes locales.

Les conditions de sélection des images clés sont plus souples. Par exemple, aucune image clé n'a été insérée depuis longtemps, la carte locale est inactive, le suivi est sur le point de ne plus pouvoir continuer, et la proportion de MapPoints dans la carte de suivi est relativement faible, dans ces cas, l'image actuelle est construite comme une image clé, et l'image clé actuelle est définie comme l'image de référence pour l'image actuelle.

Thread 2: Local Mapping

Cette section se concentre sur l'achèvement de la construction de la carte locale. Si les cartes partielles sont superposées et assemblées sans erreur, on obtient la carte complète.

La construction de carte locale comprend l'insertion d'images clés, la validation des points cartographiques récemment générés et sélectionner les meilleurs image clés, puis la génération de nouveaux points cartographiques en utilisant l'ajustement de l'ensemble des données locales (Local BA), et enfin le filtrage des images clés insérées à nouveau pour supprimer les images clés redondantes.

Correspondant à ce processus, le thread <Local Mapping> se compose de cinq parties principales : traitement de nouvelles images clés, rejet de points de carte, génération de nouveaux points de carte, optimisation locale et rejet d'images clés.

1. traitement de nouvelles images clés: (fonction <ProcessNewKeyFrame()>) lire d'abord une image clé à partir de la mémoire tampon, puis calculer la carte BOW de l'image clé, associer l'image clé actuelle au point de carte local, mettre à jour la relation de connexion de l'image clé, puis insérer l'image actuelle dans la carte.

2. élimination des points de la carte (fonction <MapPointCulling()>) : les points de la carte qui sont déjà de mauvais points sont directement supprimés de la liste de contrôle, les images qui peuvent trouver le point sont inférieures à 1/4 des images qui ont théoriquement observé le point, au moins 2 images clés ont passé depuis la création du point de la carte, mais pas plus de 2 images clés ont observé le point.

3. génère un nouveau point de carte (fonction `<CreateNewMapPoints()>`) : le mouvement de la caméra et les images clés adjacentes sont triangulés pour récupérer certains points de positions sur carte. On trouve d'abord les 20 images adjacentes ayant le plus haut degré de co-visionnement parmi les images clés de co-visionnement de l'image actuelle, itérer à travers ces 20 images, et lorsque la ligne de base est suffisamment longue, calculez la matrice de mouvement en fonction de la pose des deux images. La correspondance est effectuée par des contraintes limites, puis triangulée, et ces points sont ensuite mis en détection. Tous les points cartographiques générés doivent être détectés.

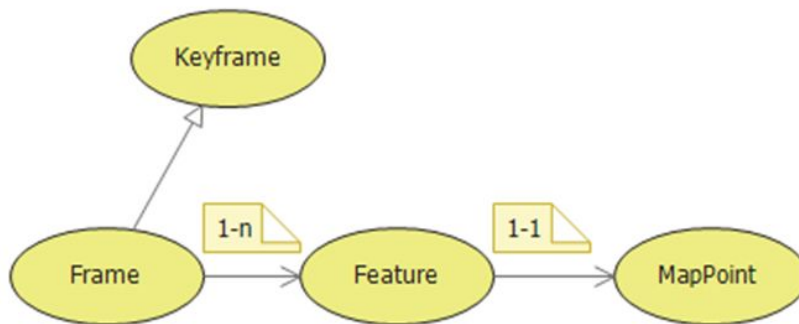


Figure .Relation entre image(Frame), image clé(Keyframe), caractéristique (Feature) et point de carte (MapPoint)

4. fusion des points de la carte (fonction de `<SearchInNeighbors()>`) : il trouve d'abord les 20 premières images avec le plus de points de consensus avec l'image actuelle, puis trouve les 5 images avec le plus de points publics dans ces 20 images, fusionne les points de la carte de ces images clés avec l'image actuelle, et met à jour le descripteur du point de la carte de l'image actuelle. Le point de la carte contient les informations suivantes : sa position dans le système de coordonnées du monde en 3D ; la direction de vue N_i , qui est le vecteur unitaire moyen de toutes ses directions de vue ; un descripteur représentant l'ORB D_i , qui est la moyenne pondérée de tous les descripteurs associés dans les images clés où le point de la carte est observé ; la distance minimale du descripteur associé ; la distance maximale d_{max} et la distance minimale d_{min} à laquelle le point peut être observé selon les contraintes invariantes de l'échelle de la caractéristique ORB.

5. optimisation locale (fonction `<LocalBundleAdjustment()>`) : ajouter d'abord l'image courante aux images clés, trouver ses images associées de premier niveau (point de carte commun supérieur à 10) dans la fonction, itérer à travers toutes les images clés adjacentes. Ici on ajoute tous leurs points de carte observés, trouve les images clés qui peuvent être observées par la fonction de `<LocalMapPoints()>` mais qui n'appartiennent pas à `<LocalKeyFrame()>`, et puis les corriger. Les images clés et les points de la carte sont utilisés comme sommets pour l'optimisation des

graphiques, et les valeurs observées des bords sont les informations de coordonnées des éléments correspondants.

5. Culling d' image clé (fonction `<KeyFrameCulling()>`) : cette étape contient l'extraction des images clés de la vue en cours selon le graphique de visibilité, l'extraction de chaque image clé de la vue en cours, l'extraction du point de carte de chaque image clé de consensus, et l'itération du point de carte de l'image clé locale pour calculer si 90% d'entre elles peuvent être observées par d'autres images, si oui, puis éliminer l'image cadre.

Thread 3: Loop Closures

Le Loop Closures est une étape d'étalonnage dans la construction de carte globale du SLAM. Pendant le fonctionnement de l'algorithme, si deux points de la carte sont déterminés comme étant au même endroit, l'algorithme effectuera un chevauchement de ces deux points dans la carte globale, corrigeant ainsi la carte. Dans l'ORB SLAM, l'algorithme ferme la boucle en déterminant la similarité des images clés pour correspondre aux points de localisation.

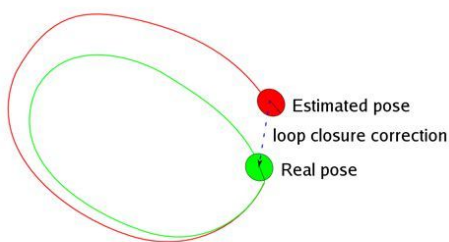


Figure .Fermeture de boucle

Cette section est divisée en deux processus principaux, la détection en boucle fermée et la correction en boucle fermée. La détection en boucle fermée est d'abord détectée, puis la transformation similaire est calculée par l'algorithme Sim3. La correction en boucle fermée, principalement la fusion en boucle fermée et l'optimisation des graphiques de l'Essentiel Graph.

Si le flux d'images est supérieur à 10 images depuis la dernière boucle fermée, une détection en boucle fermée est effectuée. Les images clés locales connectées à l'image actuelle sont éliminées, puis toutes les images clés sont parcourues pour trouver lesquelles ayant les mêmes caractéristiques que l'image clé actuelle.

Selon la méthode BOW, à chaque élément correspond un mot, et tous les éléments d'image identifiables forment un livre de mots. Dans l'algorithme, nous comptons le nombre de mots ayant le plus de mots en commun avec l'image clé actuelle parmi

toutes les images candidates, nous fixons 80 % du plus grand nombre de mots comme seuil, puis nous trouvons toutes les images clés qui dépassent le seuil seul et qui ont une détection de similarité supérieure au score le plus bas des images clés adjacentes. Dix sont un groupe, et celui qui obtient le score le plus élevé dans chaque groupe est l'image du candidat. Ces candidats doivent être résolus par Sim3 répétitivement. Après une optimisation itérative constante, l'algorithme peut traquer une gamme à forte similarité. Ensuite, pour chaque point de la carte dans cette plage, il détermine s'il dépasse 40 points de correspondance en fonction de sa correspondance avec l'image clé. S'il dépasse ce nombre, la correspondance est considérée comme réussie, sinon la file d'attente avec la cartographie locale envoyée est vidée et attend la prochaine fois.

2) Système SLAM

Nous avons utilisé les codes SLAM sous Windows, mais cela peut se faire aussi via Linux (voir d'autres OS). Cependant la procédure que nous avons réalisée ne sera valable que sous Windows.

Afin de lancer les codes SLAM il nous faut tout d'abord certains logiciels et certaines bibliothèques spéciales:

Logiciels:

- Cmake
- Visual Studio

Cmake est un outil qui permet de compiler des bibliothèques et de créer des fichiers projets. C'est un outil de construction de projets.

Visual Studio est un logiciel qui permet d'écrire des codes sous différents langages, de les déboguer, de créer des applications sous différentes plateformes, etc.

Bibliothèques:

- OpenCV
- g2o
- Eigen
- Pangolin
- DBoW2

OpenCV est une bibliothèque open source qui permet de faire du traitement et de l'analyse d'images.

g2o est une bibliothèque d'optimisation d'images.

Eigen permet de réaliser des calculs vectoriels qui sont nécessaires vu que les points sont représentés sous forme vectorielle.

Pangolin est une bibliothèque qui permet d'afficher la carte ainsi que les points que les codes SLAM auront créés.

DBoW2 permet de retranscrire les points sous forme de vecteurs et de les comparer entre eux afin de remarquer une potentielle similitude.

Nous pouvons ensuite suivre les 13 étapes de la procédure SLAM (en pièce jointe) afin d'installer les différentes bibliothèques, logiciels, et de régler les erreurs qui vont s'afficher afin de pouvoir lancer un test SLAM.

Malheureusement faute de temps, et à cause des conditions sanitaires nous n'avons pas pu résoudre tous les problèmes entre les bibliothèques et logiciels afin de lancer un test SLAM. Nous restons avec 3 erreurs qui peuvent se résumer à une seule: une mauvaise version de CMake qui appelle un programme par un chemin qui n'existe pas (les détails sont dans le fichier Procédure SLAM).

3) Modifications SLAM

Cependant après avoir étudié les différentes parties des codes SLAM nous savons quelles modifications nous pouvons apporter et où.

A) Premièrement, pour le partage en temps réel de la map:

Les modifications auront lieu dans le code myslam.cpp:

```
const int loadmap = 0; //0: enregistrer la carte; 1: charger la carte
const int tps = 1000 * 60 * 1; //entrez le temps d'exécution souhaité
```

Nous avons ici la variable de temps que nous pouvons réduire à 3 secondes par exemple, tout en créant une boucle. Le programme se lancera donc toutes les 3 secondes et partagera les informations récupérées pendant ce temps. 3 secondes est une valeur arbitraire, car le temps que met le robot à récupérer des points et à échanger les informations avec le poste de supervisions pour que celui-ci puisse lancer les calculs est assez long, donc un calcul en temps réel pur semble peu faisable, mais on peut très bien imaginer un partage plus rapide tant que cela n'affecte pas la qualité des points capturés.

Par ailleurs il faudra faire attention à ne pas écraser les informations partagées ultérieurement, il faut donc créer un programme qui stocke les données envoyées toutes les x secondes, et qui ajoute les données collectées en temps réel, afin de construire non pas une carte complète toutes les x secondes, mais une carte qui se développe toutes les x secondes, et qui ajoute tous les éléments nouveaux perçus par les robots.

Ce programme devra être un tableau, car les données renvoyées par le SLAM sont rangées dans un fichier: KeyFrameTrajectory.txt. Le programme ajoutera donc les données du nouveau tableau (envoyé toutes les x secondes) à un plus gros tableau qui lui sera envoyé à Map.bin, ce fichier est celui qui permet d'afficher la carte.

En réalisant une superposition de ces éléments toutes les x secondes, nous pourrons même ajouter les éléments capturés par tous les robots en même temps, il faudra juste créer dans ce programme qui collecte et ajoute les nouveaux éléments un ordre de priorité, par exemple le robot maître partage ses informations avant le robot esclave. Ou bien lancer le programme d'un robot 1 seconde après celui du premier robot pour que la carte se crée toutes les 2 secondes, mais ajoute des nouveaux éléments à chaque seconde!

Attention: actuellement le partage ne se fait pas en temps réel pour une bonne raison: si le robot détecte qu'il est dans une boucle, il fait évoluer les caractéristiques de ses KeyFrames (voir point D page X), et l'affichage évolue. Il faut donc que les robots gardent en mémoire les informations collectées pour que, s'il y a détection de boucle, le programme fasse évoluer la carte. Mais dans ce cas, la carte peut évoluer pour un robot mais pas pour l'autre par exemple: le premier a fait une boucle en passant devant le QR code 1, 2, 3, 4 et 5 et de nouveau devant le 1 et le 2. Le second est passé devant le QR code 1, 2, 4, 5, 6 et 7 et de nouveau devant le 4 et 5. Les informations pour les 2 robots seront donc différentes pour la zone où le QR code 1 et 2 et la zone du QR code 4 et 5.

Il faudrait donc soit qu'il y ait un robot maître qui partage sa carte en priorité, soit faire évoluer la carte partagée zone par zone, en regardant où se trouve chaque changement, mais cette solution semble assez complexe. Une dernière solution est possible, c'est de forcer un robot maître à effectuer toutes les boucles de ses esclaves. Ici il n'en aurait qu'un, le second robot, donc le robot maître devra faire son chemin, puis ensuite emprunter le même chemin que son esclave, afin de détecter les mêmes boucles. Ainsi les informations pourront se superposer plus facilement, car les informations avant une boucle et après en avoir fait au moins une (on peut faire 3-4 boucles afin d'avoir une précision extrême!) sont très différentes, mais si les 2 robots effectuent la même boucle, leurs points seront très proches et la carte sera donc lisible.

Le programme aura donc cette structure:

Le robot:

- lance le programme et le timer
- collecte les informations
- détecte une boucle (ou non)
- si oui: fait évoluer son tableau de KeyFrames
- envoie les fichiers Keyframe.txt et Map.bin
- à la fin du timer, se relance tant qu'on n'arrête pas le SLAM

Le poste de supervision:

- créer un tableau
- le remplit en ajoutant les fichiers reçus
- envoie le nouveau tableau à Map.bin
- envoie le nouveau tableau aux robots
- recommence tant que l'on n'arrête pas le SLAM

B) Pour afficher toutes les caractéristiques de l'environnement, y compris les nouveaux points spéciaux, les QR codes:

Les obstacles seront représentés en rouge et les différents déplacements du robot en bleu.

Nous pouvons alors implémenter le programme de la partie Analyse Vidéo, qui reconnaît les différents QR codes, et retourne la distance et l'angle du robot par rapport au QR code.

Les calculs des points caractéristiques qui s'affiche avec le SLAM est assez complexe et peu manipulable donc mieux vaut ne pas trop y toucher. En revanche nous pouvons écrire une condition spéciale, qui prend effet si le robot détecte un QR code (donc ce programme de détection devra tourner en continue et renvoyer 0 s'il n'y a aucun QR code), et qui force le programme à ajouter un point caractéristique.

Nous pouvons alors changer l'affichage de ces points en question, car ils auront été créés d'une manière différente des points habituels, et par exemple afficher un point plus gros, et dans une autre couleur.

Ces modifications de couleurs se feront dans le code colour.h:

```
struct Colour
{
    inline static Colour White() {
        return Colour(1.0f,1.0f,1.0f,1.0f);
    }
    inline static Colour Black() {
        return Colour(0.0f,0.0f,0.0f,1.0f);
    }
    inline static Colour Red() {
        return Colour(1.0f,0.0f,0.0f,1.0f);
    }
    inline static Colour Green() {
        return Colour(0.0f,1.0f,0.0f,1.0f);
    }
    inline static Colour Blue() {
        return Colour(0.0f,0.0f,1.0f,1.0f);
    }
    inline static Colour Unspecified() {
        return Colour(
            std::numeric_limits<float>::quiet_NaN(), std::numeric_limits<float>::quiet_NaN(),
            std::numeric_limits<float>::quiet_NaN(), std::numeric_limits<float>::quiet_NaN()
        );
    }
};
```

Nous pouvons créer des nouvelles couleurs en changeant les 4 paramètres de Colour

Et les modifications de tailles dans le code MapDrawer.h:

```
private:
    float mKeyFrameSize;
    float mKeyFrameLineWidth;
    float mGraphLineWidth;
    float mPointSize;
    float mCameraSize;
    float mCameraLineWidth;
```

Nous pouvons créer une nouvelle taille de points qui différenciera encore plus les points des QR codes des points “normaux” relevés par les robots.

Et pour forcer le programme à rajouter les QR codes comme points caractéristiques ce sera dans le code Map.cc :

```

void Map::AddKeyFrame(KeyFrame *pKF)
{
    unique_lock<mutex> lock(mMutexMap);
    mspKeyFrames.insert(pKF);
    if(pKF->mnId>mnMaxKFid)
        mnMaxKFid=pKF->mnId;
}

void Map::AddMapPoint(MapPoint *pMP)
{
    unique_lock<mutex> lock(mMutexMap);
    mspMapPoints.insert(pMP);
    ///
    //if (pMP->mnId > mnMaxMPid)
    //    mnMaxMPid = pMP->mnId;
    ///
}

```

et KeyFrame.cc:

```

void KeyFrame::AddMapPoint(MapPoint *pMP, const size_t &idx)
{
    unique_lock<mutex> lock(mMutexFeatures);
   .mvpMapPoints[idx]=pMP;
}

```

Il faudra définir une KeyFrame spéciale dans KeyFrame.cc qui sera ajouté dans Map.cc

Dans Map.cc: Ici nous avons déjà une condition if, nous n'avons plus qu'à la transformer en elseif et écrire sur la ligne précédente un if Programme_d'Analyse_Vidéo ne retourne pas 0, alors ajouter NewKeyFrame.

Il faudra cependant faire attention qu'un seul point spécial soit créé par QR code lu, donc une autre condition doit être écrite, un if QR_Code n'est pas présent dans "Bibliothèque QR code" qui est une liste qui se remplit au fur et à mesure que de nouveaux QR codes sont lus.

La condition finale sera donc if "Programme_d'Analyse_Vidéo ne retourne pas 0" and "if Bibliothèque QR code ne retourne pas 0" alors ajouter NewKeyFrame.

Dans KeyFrame.cc: Il faut simplement définir cette NewKeyFrame en lui donnant les mêmes caractéristiques que les KeyFrames "normales", le but étant de forcer la création d'une KeyFrame spéciale, nous n'allons pas demander au code de calculer toutes les ressemblances avec les autres points aux alentours etc, nous la créons simplement.

C) Pour afficher des points de manière plus claire et les relier afin de former des lignes

Nous avons étudié comment cette création de lignes pourrait se faire, nous pensions relier des points qui seraient assez proches entre eux, mais des problèmes surviennent lorsqu'il y a un grand amas de points très proches, des lignes se formeraient alors dans tous les sens.

C'est aussi ce qu'a révélé la thèse utilisée précédemment: [Monocular] Raúl Mur-Artal, J. M. M. Montiel and Juan D. Tardós. ORB-SLAM: A Versatile and Accurate Monocular SLAM System. *IEEE Transactions on Robotics*, vol. 31, no. 5, pp. 1147-1163, 2015. (2015 IEEE Transactions on Robotics Best Paper Award).

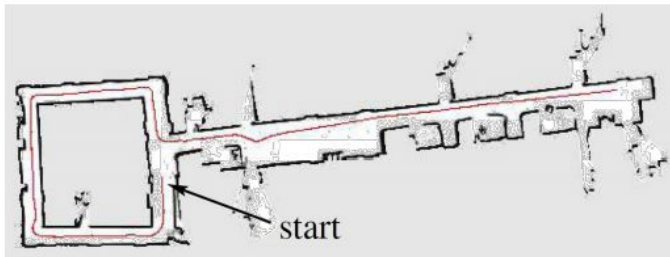
Nous pouvons voir que relier des points proches afin de former des lignes est un exercice plus que ambitieux, et qui aujourd'hui n'a toujours pas été réalisé car il nécessite énormément de calculs et requiert une précision de caméra très élevée que nous n'avons pas en utilisant les caméras monoculaires des smartphones.

Mais nous pouvons alors essayer une autre technique: combiner les informations des 2 robots, les caméras étant différentes et leur calibrage aussi, les points relevés par les 2 robots seront assez proche pour rentrer dans la tolérance des 5 cm, mais assez éloignés pour que pris dans leurs ensemble ils permettent une création de carte bien plus détaillée. Nous n'aurons donc pas des lignes à proprement parler, mais 2 amas de points assez proches qui se superposeraient.

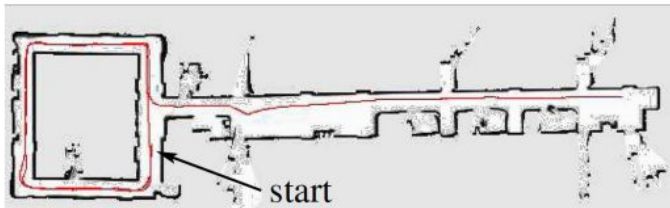
D) Pour permettre un repositionnement du robot:

Dans le code LoopClosing.h nous pouvons constater que le robot peut percevoir une boucle.

C'est-à-dire que s'il repasse devant un ensemble de points caractéristiques il va pouvoir s'en souvenir et donc améliorer la précisions de la carte, comme dans l'exemple ci-dessous:



(a)



Lorsqu'il ne fait pas de boucle son chemin est bien différent! D'où le problème du point A: essayez de superposer ces 2 cartes, cela n'aurait aucun sens!

Donc c'est dans ce code que nous devrions appeler les fonctions qui permettront de repositionner le robot. En effet si le robot détecte une boucle en repassant devant un QR code qu'il a déjà vu, il pourra alors comparer sa position qu'il affiche à sa position réelle : vu qu'il passe devant un QR code, le programme du QR code se lance et retourne sa distance et l'angle. Il compare alors cette valeur à sa position actuelle qu'il retourne à Map.bin, si les valeurs sont différentes il se replace jusqu'à que les valeurs soient à moins de 5 cm d'écart et moins de 30° .

Le programme de repositionnement en lui-même n'est pas long à écrire, c'est une boucle while qui ne s'arrête que si la précision des 5cm et des 30° est atteinte.

Dans cette boucle on peut écrire la fonction de transfert qui aura comme consigne une distance de 3 cm et un angle de 0° par exemple (nous prenons des chiffres inférieurs à la précision demandée comme cela, s'il y a une imprécision autour des 3 cm (par exemple 4) nous resterons toujours dans la limite des 5 cm à ne pas dépasser).

Cette fonction commandera alors les moteurs des roues. Cette fonction pourra être une simple fonction de premier ordre (avec un second ordre on pourrait avoir des dépassements ce qui n'est pas souhaitable), avec comme gain K, la différence entre la position actuelle et la position où le robot serait à 3 cm du QR code. La constante de temps tau détermine le temps que va mettre le repositionnement, ce temps n'étant pas une contrainte nous pouvons le fixer arbitrairement à 5 secondes (et le diminuer si besoin tout en faisant des test pour ne pas que cela influe sur sa précision).

Attention: il faudra faire attention que le robot détecte en permanence le QR code, s'il n'y a plus de détection le robot continuera sa route comme s'il n'avait rien vu (dû au retour de Programme_d'Analyse_Vidéo qui vaudrait 0). Donc dans un premier temps dans le repositionnement du robot il faut le faire avancer proche du QR code, et face à lui, pour l'avoir toujours face à lui. Ensuite on peut lancer la seconde partie du programme (la fonction de transfert) qui compare les distances et angle et ordonne aux roues d'avancer pour permettre un repositionnement.

Donc le programme aura cette structure:

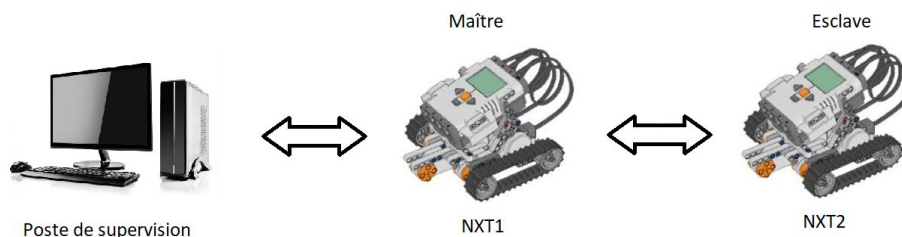
- se lance s'il y a détection d'un QR code déjà enregistré
- s'approche du QR code se tourne afin d'être à peu près face à lui
- calcule les variables qui permettront de créer une fonction de transfert
- lance la boucle "fonction de transfert"
- sort de la boucle lorsque la consigne est atteinte à 95% (la consigne étant plus précise que la précisions demandée, si nous sommes à 95% de la consigne nous sommes dans les intervalles de précision demandés).

5) PARTIE COMMUNICATION

Nous disposons d'un robot maître qui va établir les connexions avec ces esclaves. Comme esclaves, nous aurons un 2ème robot ainsi que le poste de supervision.

Objectif: Etablir la connexion Bluetooth entre:

- le maître et le poste de Supervision
- le maître et le 2ème robot



Au vu de cette architecture pour que le poste de supervision obtienne les informations de NXT2, celles-ci doivent forcément passer par NXT1 avant d'être récupérées par le poste de supervision.

1) Protocole Bluetooth "Radio Frequency Communication" (RFCOMM)

Les robots NXT possèdent une carte bluetooth CSR Bluecore 4 v.2. Le protocole de communication utilisé est le protocole RFCOMM. La fréquence porteuse est de 2.4 GHz avec une portée de 10 mètres.

Le Bluetooth RFCOMM, aussi connu sous le nom de “Bluetooth SPP” (Serial Port Profile), est dit être excellent pour envoyer et recevoir des informations entre deux appareils. On distingue le SPP-A du SPP-B par le fait qu’avec le SPP-A la connexion est sortante, c’est-à-dire que le périphérique lance la connexion à un autre périphérique.

2) Ports, Mailboxes et messages

La communication Bluetooth des NXTs est configurée comme une communication maître-esclave, le maître étant celui qui lance la connexion. Il faut éviter d’avoir plus d’un maître car c’est **le maître qui impose l’ «horloge»**. Le robot maître peut lancer la connexion par Bluetooth à 3 périphériques en même temps mais il ne peut communiquer avec un seul esclave à la fois. Mais **cette configuration ralentie les communications** et il faut savoir qu’un robot ne peut pas être à la fois esclave et maître.

Chaque NXT possède 4 canaux/ports de connexion pour la communication Bluetooth numérotés de 0 à 3. Les robots esclaves n’utilisent que le port 0 pour communiquer avec le maître tandis que le maître utilise les 3 autres ports pour envoyer ces messages aux esclaves.

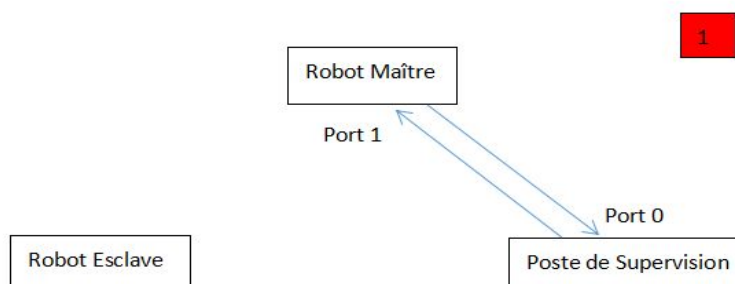
Chaque robot possède **20 mailboxes/queues** pour la réception des données, qui elles-mêmes contiennent **10 messages** maximum. Les messages transmis sont de **58 octets maximum** et sont lus en **FIRST IN-FIRST OUT (FIFO)**. Une fois le message lu, il est supprimé de la mailbox.

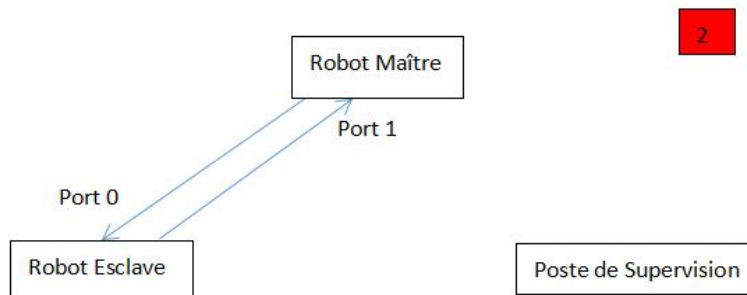
3) Schémas du réseau

Il est recommandé de:

- n’utiliser qu’un robot maître afin de n’avoir qu’une horloge.
- se connecter qu’à un robot esclave à la fois pour accélérer les communications.

En accord avec les travaux des années précédents, les messages feront 16 octets et le maître enverra ces messages aux esclaves que via le port 1.





4) Types de messages transmis

Lors de l'écriture d'un message on renseigne le type du message, il est défini par des lettres correspondant à des codes ASCII bien précis.

Il existe plusieurs type de messages que nous avons listés ci dessous:

Type de Message	ID ASCII
Requête de l'État de charge d'une batterie	'Q'=81
Niveau de batterie	'W'=87
Acknowledge batterie	'I'=73
Requête de données	'R'=82
Données du Robot (Distance et orientation par rapport à un QR code, ID du QR code)	'P'=80
Acknowledge data	'G'=71

5) Destinataire / Émetteur

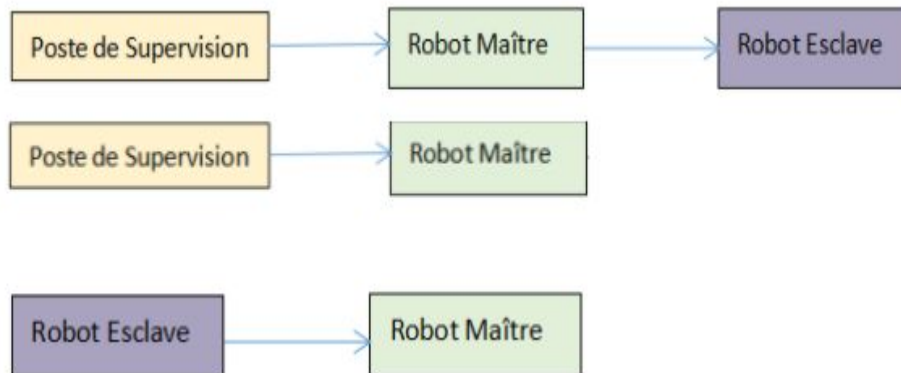
Lors de l'écriture d'un message on renseigne l'émetteur et le récepteur, ils sont définis par des lettres correspondant à des codes ASCII bien précis.

Il existe un identifiant par périphérique, ils sont trouvables ci dessous:

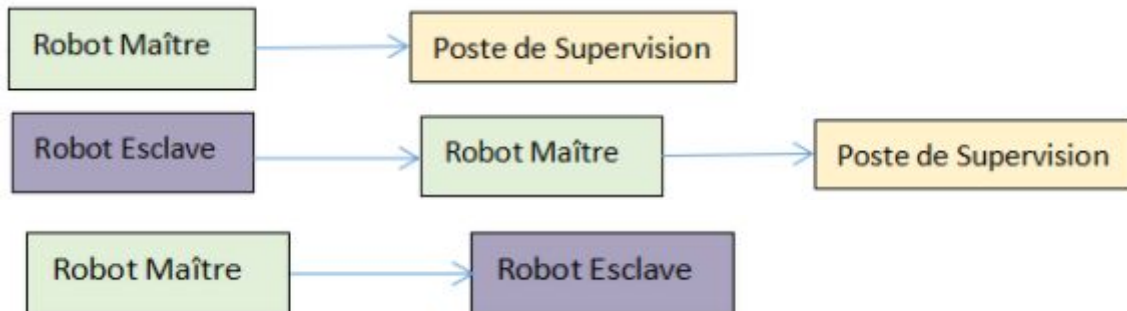
Périphérique	ID ASCII
Robot NXT 1	'A'=65
Robot NXT 2	'B'=66
poste de Supervision	'C'=67

6) Circulation des messages

Requêtes de l'État de charge d'une batterie 'Q'=81:



Niveau de batterie 'W'=87:



```
int nAvgBatteryLevel
```

(int) The average battery level in millivolts. (A value of 9458 represent 9.458 volts.) It's the average of 20 recent samples and smoothes out sudden voltage transients.

```
int nBatteryAverage = nAvgBatteryLevel; // create and set integer variable 'nBatteryAverage'
// to the average batter level of the device
```

Acknowledge Batterie 'I'=73

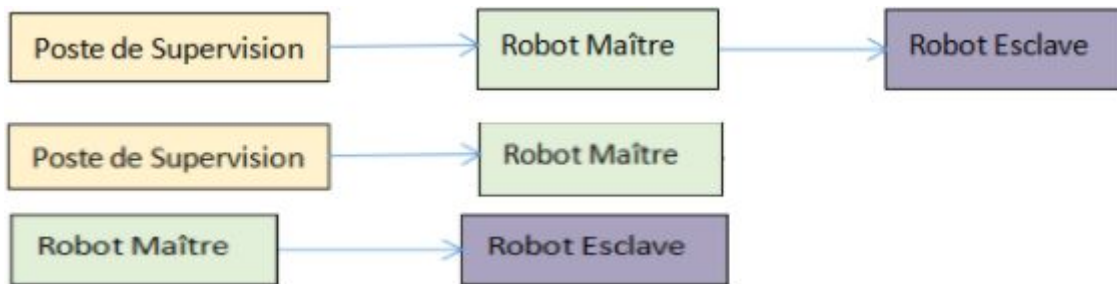


Requête de données 'R'=82



Le robot maître sauvegardera les données du NXT2 avant de les lui transmettre. Il n'a pas à les lui demander.

Données du Robot 'P'=80



Acknowledge Data 'G'=71



7) Mise en forme des buffers

La taille des buffers étant limitée, il faudra rajouter un octet pour différencier les buffers envoyés/reçus OU créer d'autres types de Message.

Les ID sont des codes ASCII.

Information	ID ASCII
niveaux de batterie	'L'=76 >7000 'M'=77 Low 'N'=78 very Low <=6300
distance au Qr code	'd'=100
Angle par rapport au Qr code	'b'=98
ID du QR code	'q'=113

Pour notre projet nous avons utilisés quatre types de buffers pour quatre messages différents, on retrouve ci dessous les différents buffers et leur composition:

buffer	de	Byte 0	Byte 1	Byte 2	Byte 3	...	Byte 15
requête		ID type de message	ID émetteur	ID Destinataire	0	0	0x00

buffer	de	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 15
Données		ID type de message	ID émetteur	ID Destinataire	ID de la donnée 1	MSB donnée 1	LSB donnée 1	0	0x00

buffer Niveau de batterie	Byte 0	Byte 1	Byte 2	Byte 3	Byte 15
	ID type de message	ID émetteur	ID Destinataire	Niveau de batterie	0	0x00

buffer Acknowledge	Byte 0	Byte 1	Byte 2	Byte 3	Byte 15
	ID type de message	ID émetteur	ID Destinataire	0	0	0x00

ATTENTION:

Nous avons remarqué que les messages reçus par le poste de supervision comportent au début 6 octets tout le temps identiques: 20 0 128 9 0 16. Les messages envoyés par le poste seront d'une longueur de 22 octets dont les 6 premiers seront ceux ci-dessus. Le robot maître ne lira que 16 octets en commençant par l'indice numéro 6 initial sachant que les indices débutent à 0.

Le transfert de données se fait dans le mode «data mode» des robots qui ne peut être activé en même temps que le «command mode». La valeur 0 est spécialement utilisée pour indiquer qu'un message n'a pas été reçu.

8) Programmes

Chaque périphérique aura une "mailbox principale" parmi les 20 possibles, dans laquelle il va recevoir les messages et les traiter en FIFO.

a) Robot maître

Étape 1: Il faut savoir que notre interface ne peut lire ou écrire que s'il est connecté au robot maître sinon elle se bloque. Ainsi le robot maître se connecte à un esclave et laisse à ce dernier le temps de traiter au moins le message en tête de fil de sa propre mailbox principale.

Étape 2: Le maître ne se déconnecte pas encore. Il lit le message en tête de fil de sa mailbox principale (mailbox1):

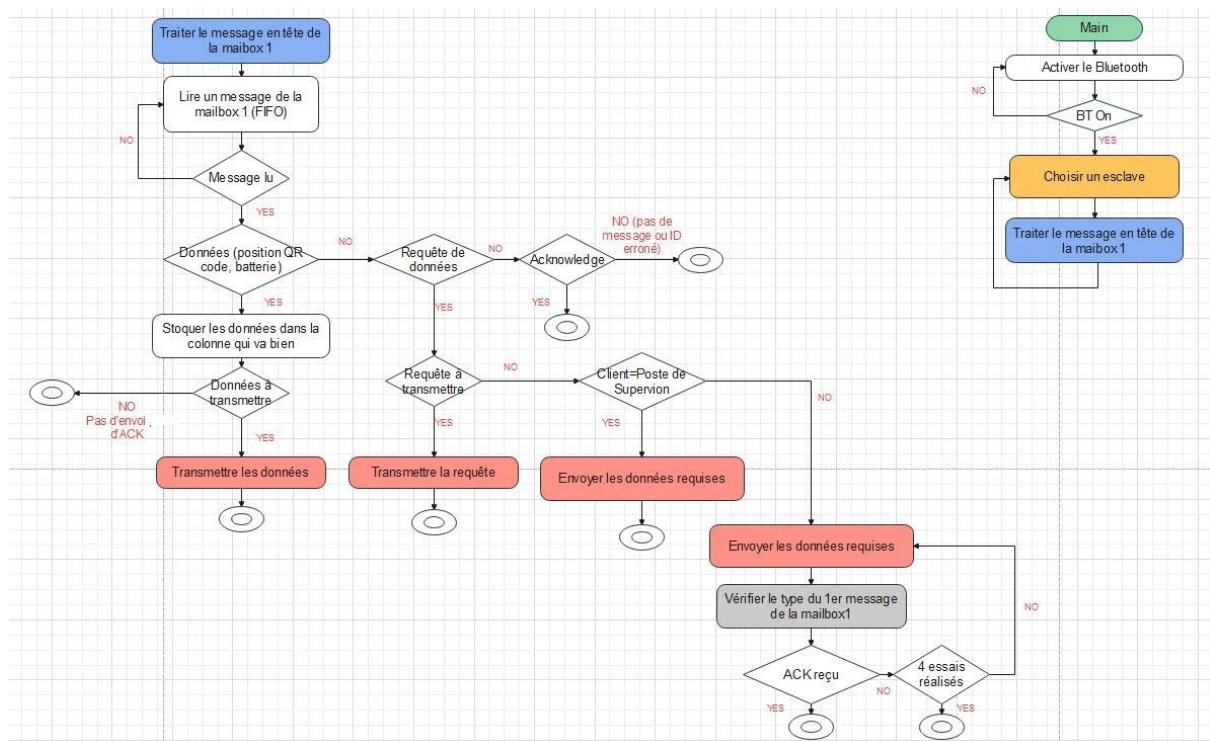
A) Il va se déconnecter et transférer le message s'il ne lui est pas destiné. Il sera donc connecté à un autre appareil à la fin du processus. **S'il s'agit des datas concernant**

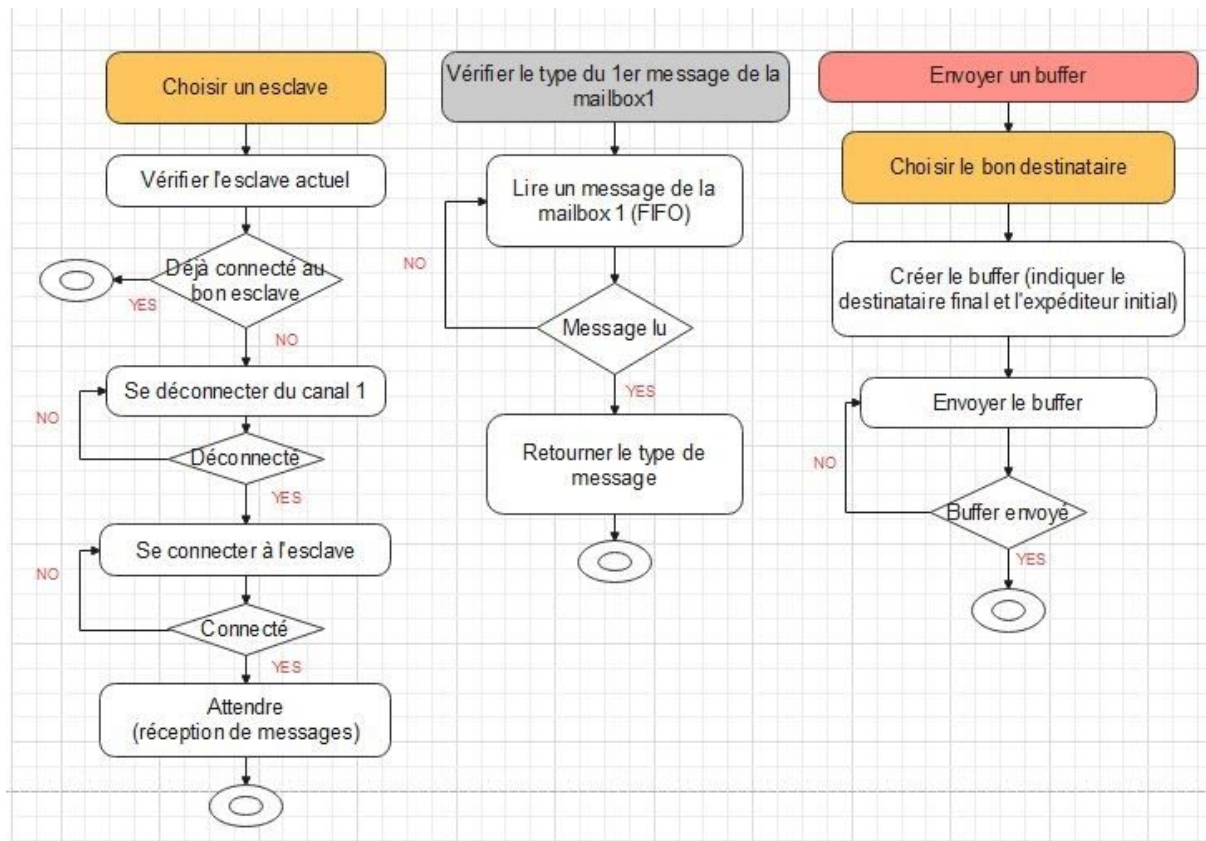
la position du robot esclave par rapport aux QR codes, il va les stocker dans la colonne qui va bien avant de les transmettre. Il n'a donc pas à demander ces informations au robot esclave.

B) S'il s'agit d'une requête de ses informations, il répond. Ainsi s'il était déjà connecté au demandeur, il n'a pas à se déconnecter. Il attendra un ACK de la part du robot esclave en vérifiant le type de message des 4 premiers messages reçus. **Ces messages seront donc perdus.**

C) S'il s'agit d'une information pour lui, il va juste la stocker.

Il se connecte au périphérique suivant si ce n'est déjà fait (en se déconnectant au préalable) pour refaire la même chose.





b) Robot esclave

Une fois connecté au maître, il va traiter au moins le message en tête de sa propre mailbox principale (mailbox1).

A) S'il s'agit d'une information pour lui, il va la stocker. Si l'information provient du robot maître, il doit retourner un ACK dans la mailbox principale du robot maître.

B) S'il s'agit d'une requête, il va envoyer la réponse dans la mailbox principale du robot maître.

Il peut envoyer une requête s'il en a mais il n'attendra pas d'ACK.

Tant que le robot maître ne reprend pas le contrôle sur la ligne, il refait ces étapes en boucle. **Il serait bon d'intégrer une variable pour indiquer à quel moment il devrait demander le niveau de batterie du robot maître.**

c) Interface

Pour pouvoir valider les programmes nous avons mis en place une interface permettant d'envoyer ou recevoir des informations de manière à reproduire tous les cas de communications. le langage utilisé est le **C# (Csharp)**.

Afin de permettre au robot maître de se connecter au Poste de Supervision, il faut:

Étape 1: Lancer l'appairage entre les 2 périphériques soit:

- depuis les paramètres Bluetooth du robot maître: Bluetooth > Search > Choisir le périphérique > Connect > Choix du port 1.
- depuis l'ordinateur.

Le code sera ' 1234 ' (valide en appuyant sur le bouton orange une fois le code apparu sur le robot).

Étape 2: Ajouter le port entrant sur l'ordinateur permettant au robot maître de lancer la connexion (peut-être qu'il sera déjà ajouté)[1]. **Ce port doit être ouvert depuis le poste de supervision pour que la connexion soit autorisée. Il sera fermé automatiquement à la fermeture de l'interface. Il faut s'assurer qu'il ne soit pas contrôlé par une autre application (RobotC...). Voir références [5] et [8] pour configurer le port.**

Paramètres > Périphériques > Paramètres Bluetooth Avancés > Ports COM

Appareils Bluetooth et autres

+ Ajouter un appareil Bluetooth ou un autre appareil

Bluetooth
Activé

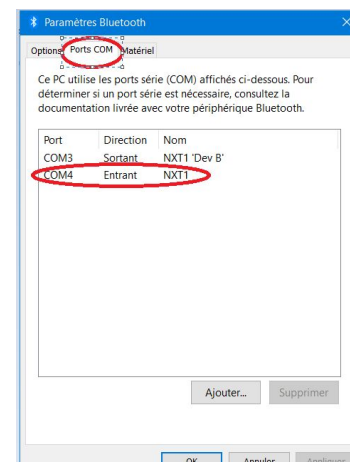
Maintenant détectable en tant que « ASPIRE »

Souris, clavier et stylet

USB Receiver

Activer le Bluetooth encore plus rapidement
Pour activer ou désactiver le Bluetooth sans ouvrir les Paramètres, ouvrez le centre de notifications, puis sélectionnez l'icône Bluetooth.

Paramètres associés
Périphériques et imprimantes
Paramètres audio
Paramètres d'affichage
Paramètres Bluetooth avancés



Il est aussi possible d'utiliser le logiciel "**Hercules setup**" qui est gratuit pour visualiser les informations provenant du maître.

Réception:

Comme indiqué dans la partie 4.6 , les messages reçus par le poste de supervision commencent toujours par les 6 octets **20 0 128 9 0 16**. La lecture se fait octet par octet.

Pour éviter les décalages dans la lecture des buffers et qu'on ne lise pas de buffers vides, il faut vérifier qu'on a bien la suite ci-dessus et que le 7ème octet lu ne soit pas nul.

Les informations reçues (niveau de batterie des robots et mesures du code OpenCV) seront utilisées pour mettre à jour le tableau "txtData"). Il faut identifier le robot auquel appartiennent ces informations.

Envoi:

Il sera possible d'envoyer des requêtes et des données. Il y a un programme pour mettre en forme les buffers. Les informations à envoyer seront prises directement sur le tableau "txtData".

d) Fonctions utiles sur RobotC

- **setBluetoothOn()** : activer le bluetooth
- **setBluetoothOff()** : désactiver le bluetooth
- **btConnect (int nPort, string sFriendlyName)** : se connecter à un esclave sur les ports 1,2 ou 3
- **btDisconnect(int nPort)** : se déconnecter du port 0 si c'est un esclave ou d'un des ports 1 à 3 si c'est un maître.
- **wait10Msec(int k)**: attendre k*10 Msec
- **cCmdMessageWriteToBluetooth (nQueueID,nXmitBuffer,nSizeOfMessage)**: Pour envoyer le message de taille nSizeOfMessage (58 octets max) contenu dans nXmitbuffer dans la mailbox nqueueID du peripherique cible,
- **cCmdMessageRead(nQueueID,nRcvBuffer,nSizeOfMessage)** : pour extraire le premier message de la mailbox nQueueID et le copier dans nRcvBuffer,
- **cCmdMessageGetSize(nQueueID)**: pour obtenir la taille du premier message de la mailbox nQueueID.
- **while (bBTBusy)** : tant qu'il y a une opération déjà en cours sur le Bluetooth
- **if (nBTCurrentStreamIndex>=0)** : si une communication Bluetooth est ouverte
- **while (nBluetoothCmdStatus!=ioRsltSuccess)** : tant que la dernière opération Bluetooth n'a pas été un succès. (attention l'opération peut avoir échoué ou être en progression)

+ voir Help > Recherche > Bluetooth)

9) Expérience

Pour prouver le bon fonctionnement de la communication, nous avons effectué une expérience que l'on a filmé. Nous allons expliquer l'expérience en détails avec des captures d'écran à l'appui. Le nom ASPIRE correspond simplement au nom de l'ordinateur avec lequel nous avons effectué les tests.

La première étape est d'établir la connexion :

- On sélectionne le port via la boîte 
- et on l'ouvre pour permettre au robot maître d'établir la connexion grâce au bouton 

La deuxième étape est de sélectionner le type de message voulu avec le destinataire voulu et d'envoyer le message une fois connecté au robot maître:

- On sélectionne le type via la boîte 

- puis le destinataire avec la boîte
- et enfin envoyer le message avec

Receiver

Send

La troisième étape concerne la lecture du message envoyer par le poste de supervision, la requête ne vise pas NXT1 donc il sert simplement de messager et renvoi le message pour NXT2:



ASPIRE Requests NXT2 Batt

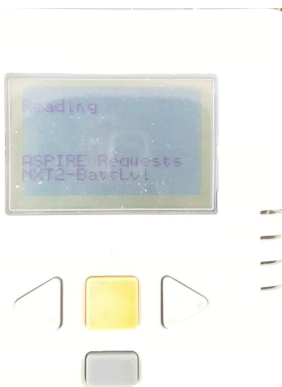


Connected to NXT2

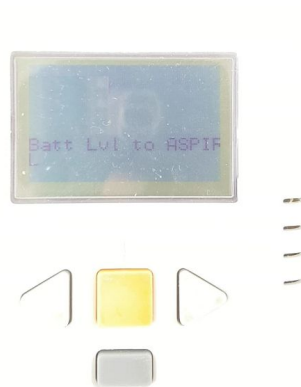


W - BT Busy

La quatrième étape traite de la lecture du message de NXT1 par NXT2 et de la réponse de NXT2 par NXT1, comme dit précédemment pour envoyer ses données au poste, les informations passe d'abord par NXT1:



ASPIRE Request NXT2 Battlvl



Batt Lvl to ASPIRE

La cinquième étape est la transmission du message de NXT2 vers le poste. NXT1 lit le message et comprend qu'il n'est pas le destinataire alors il renvoie le message au poste de supervision:



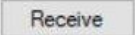
Reading Batt Lvl from NXT2

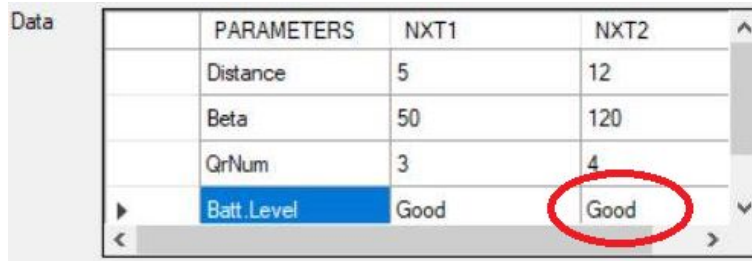


NXT2 Batt to ASPIRE

to ASPIRE

La dernière étape c'est la lecture des informations sur le poste pour cela:

- on appuie sur le bouton 
- et on lit les informations dans le tableau



	PARAMETERS	NXT1	NXT2
	Distance	5	12
	Beta	50	120
	QrNum	3	4
	Batt.Level	Good	Good

on dit que le niveau est "good" car il dépasse un certain seuil

6)PROBLÈMES ET PISTES:

1) Analyse Vidéo

2) SLAM

ORB-SLAM2 est un ensemble de codes qui permettent de lancer un SLAM, mais ces codes sont très gourmands en énergie. Les calculs doivent donc se faire via le poste de supervisions avec qui il faut donc communiquer. Le calcul en temps réel pur semble donc très peu possible, en revanche l'affichage toutes les 2-3 secondes semble déjà bien plus faisable.

La carte créée sera donc remplie de points, voire d'amas de points si la fusion des informations collectées par les 2 robots se fait correctement. Mais ces points ne serviront qu'en grande partie à l'opérateur afin de lui permettre de voir en partie ce que voit le robot. En effet, les seuls points utiles pour les robots seront les points qui leur permettent de se repositionner. Les autres points ne pourront pas être utilisés comme variables dans des fonctions d'évitement d'obstacles par exemple. Il faudrait créer encore une nouvelle classe de points spéciaux qui seront catégorisés et affichés comme obstacles, là les robots pourront s'en servir.

Les caméras utilisées sont des caméras monoculaires, c'est-à-dire qu'à chaque nouveau lancement de programme leur initialisation varie, ainsi que leur calibration. L'échelle de la carte commune créée varie donc à chaque lancement! D'autre part, plus les robots explorent leur environnement, plus la carte partagée se rétrécit, l'échelle évolue donc constamment. Pour éviter des accumulations d'erreurs sur la carte, il faut donc impérativement effectuer des boucles aux robots, ils doivent passer plusieurs fois devant chaque QR code, afin de retravailler leur carte

De plus, lorsque la caméra n'est pas en mouvement, elle ne peut pas calculer la position des Map points. Ce problème est mineur car la caméra sera souvent en mouvement, mais cela empêche le robot de trouver des nouveaux points lorsque celui-ci est à l'arrêt, c'est un "petit gaspillage d'énergie".

3) Communication

Précision:

Les valeurs de position sont des entiers non signées codés sur 2 octets. On peut donc mesurer des distances de 0 à 65 536 mm. Mais on aurait pu se limiter à 11 bits (0 à 2048 mm) car nos caméras ne détectent pas les QR code d'aussi loin et cela n'a aucun intérêt.

Temps de Propagation:

Pour l'instant nous ne sommes qu'en phase de développement. Nous avons rajouté des affichages sur les robots pour suivre leur état d'avancement ainsi que des temps d'attente pour lire ce qui est affiché. Ainsi nous sommes loin du temps réel.

Disponibilité:

- problème de répétabilité (robot maître n'arrive pas à se connecter à un esclave ou les robots ne reçoivent pas les messages)

7) CONCLUSION

- Pas de lancement et d'amélioration du SLAM dû à des erreurs -> pistes étudiées: on sait où coder et ce qu'il faut coder
- Procédure lancement SLAM créée
- Communication robot-robot et robot-poste de supervision OK
- Reconnaissance des QR codes et analyse de leurs informations OK

8) BIBLIOGRAPHIE

- [1] https://www.youtube.com/watch?v=Fg1dDEjk9UA&ab_channel=AyushJain
- [2] https://github.com/introverti/OpenCV_PLP
- [3] [Install and configure OpenCV-4.2.0 in Windows 10 — VC++](#).Aymane Hachcham
- [4] <https://github.com/shimat/opencvsharp>
- [5] https://www.youtube.com/watch?v=Fer_q9LXDnQ&ab_channel=FoxLearn
- [6] https://www.youtube.com/watch?v=Fg1dDEjk9UA&ab_channel=AyushJain
- [8] <https://docs.microsoft.com/fr-fr/dotnet/api/system.io.ports.serialport.stopbits?view=dotnet-plat-ext-5.0>
- [9] *BLUETOOTH BI-DIRECTIONAL COMMUNICATION BETWEEN AN NXT ROBOT AND A PC*, Sanda Paturca, Mariana Ilas, Adrian Greu, Constantin Ilas, Décembre 2009
- [10] Rapports et programmes des années précédentes.