Rapport de projet temps réel 4AE

Nom du binôme : JUMIN Noël, MARCÉ Clément

Enseignant de TP : Thibaud LASGUIGNES

**Résumé**

Ce rapport a pour but d’expliquer le déroulement du bureau d’étude « Robot Mobile » du binôme Noël JUMIN et Clément MARCÉ. Il s’agit d’un projet s’inscrivant dans le cursus d’ingénieur Automatique Électronique de l’INSA de Toulouse et plus particulièrement de l’enseignement de Temps Réel en 4ème année.

L’objectif consiste à réaliser une entité logicielle de supervision des opérations d’un petit véhicule-robot. Pour cela, nous avons dans un premier dû nous accommoder aux ressources à notre disposition telles que le cahier des charges du client ou encore le système physique dans lequel notre solution doit être implémentée. Nous avons par la suite déroulé une phase de conception qui vise à s’approprier le besoin du client pour proposer une solution théorique réalisable avec nos moyens. Suite à cela, il nous a été possible de réaliser le superviseur grâce à un travail de programmation suivi de tests d’implémentation. Finalement, nous avons pu présenter les résultats de cette étude et conclure sur nos performances vis-à-vis du besoin initial.

**Abstract**

The purpose of this report is to explain the progress of the « Mobile Robot » project carried out by Noël JUMIN and Clément MARCÉ. This project is part of the Automatic Electronics engineering cursus at INSA Toulouse, and more specifically the Real Time course in 4th year.

The aim was to create a software entity to supervise the operations of a small robotic vehicle. To do this, we first had to familiarise ourselves with the available resources, such as the customer's specifications or the physical system in which our solution would be implemented. We then carried out a design phase aimed at appropriating the customer's needs to propose a theoretical solution that we could provide. Following this, we were able to create the supervisor through programming work and implementation tests. Finally, we were able to present the results of this study and to conclude on our performance regarding the initial requirements.

**Sommaire**

[I. Introduction 5](#_Toc165737234)

[II. Architecture fonctionnelle 6](#_Toc165737235)

[III. Architectures physiques 7](#_Toc165737236)

[IV. Codage et livraisons incrémentales 10](#_Toc165737237)

[V. Analyse et validation du logiciel livré par rapport aux exigences 11](#_Toc165737238)

[VI. Commentaires et conclusion 12](#_Toc165737239)

**Définitions**

|  |  |
| --- | --- |
| **Nom** | **Définition** |
| Xenomai | Système d’exploitation basé sur l’utilisation de Linux en tant que tâche pour répondre aux contraintes du temps réel. |
| Xbee | Module de communication sans fil assurant un échange d’informations à faible consommation et débit. |
| Netbeans | Environnement de programmation permettant de compiler du code C et C++ sur carte Raspeberry Pi. |
| Raspberry Pi | Ordinateur miniature permettant la réalisation de tâches simples. |

# Introduction

Dans le cadre de la formation d’ingénieur Automatique Électronique de l’INSA de Toulouse et de son enseignement en systèmes temps réel, nous avons eu l’occasion de prendre part au bureau d’étude « Robot Mobile ».

L’objectif principal de ce projet consiste en la réalisation d’une unité de supervision des opérations d’un robot-véhicule. Cette solution technique doit d’abord être conçue à partir des besoins de supervision établis. Elle devra ensuite être programmée puis implémentée dans le système physique.

Les ressources à disposition sont :

* Un Rasperrby Pi 3 avec l’OS Xenomai et un module de communication Xbee. Il s’agit du composant qui viendra accueillir le code de notre superviseur et assurera sa communication avec le robot. La carte dispose également d’un module caméra.
* Un robot véhicule avec un module de communication Xbee et un microcontrôleur. Les fonctions de déplacement du robot sont préalablement codées.
* Une arène dans laquelle circulera le robot.
* Un PC opérateur équipé d’un moniteur faisant office d’IHM pour l’envois de consignes au robot. C’est également l’ordinateur qui servira au développement du superviseur grâce à l’IDE Netbeans.
* Un cahier des charges des tâches à superviser.

Cette étude est réalisée en 4 séances de TP et couvre un vaste panel d’activités de l’ingénieur. En commençant par la prise en main d’un nouvel environnement de travail et l’établissement de besoins, il sollicite également les phases de conception et de développement d’une solution technique, jusqu’à sa livraison au client.

Il met finalement en place différents aspects théoriques et pratiques, tant sur des bases acquises comme la programmation C que sur de nouveaux concepts fondamentaux des systèmes temps réel.

Nous allons par la suite explorer ces axes en entrant plus en détails dans le cœur du projet.

# Architecture fonctionnelle

Dans un premier temps, il nous a fallu nous intéresser au cahier des charges, c’est-à-dire la description des besoins du client. Il s’agit concrètement de la définition des tâches que devra réaliser notre superviseur.

Afin de prendre en main ce cahier des charges et débuter une démarche d’organisation de notre travail, nous avons regroupé les besoins du projet en fonctions.

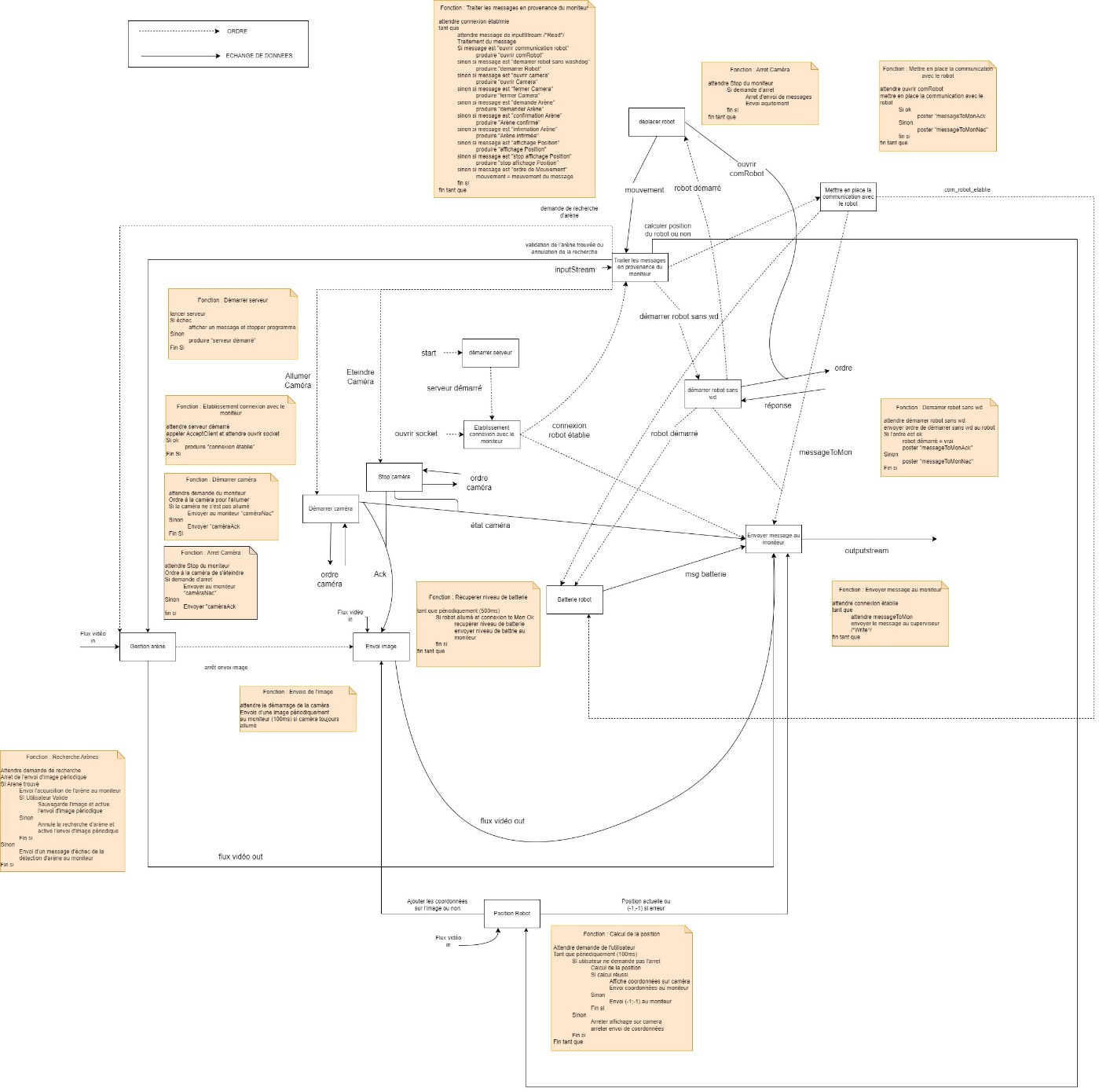
Recensement des fonctions :

|  |  |  |  |
| --- | --- | --- | --- |
| Nom de la fonction | Description du comportement | Entrées | Sorties |
|  |  |  |  |
| Démarrer serveur | Démarre le serveur | Démarrage du système | État de démarrage du serveur |
| Établissement connexion avec moniteur | Ouvre la communication du moniteur avec le robot | État de démarrage du serveur | État connexion du moniteur |
| Traiter les messages en provenance du moniteur | Gère les messages du moniteur | État connexion du moniteur  Messages du moniteur | Réponses aux messages du moniteur |
| Envoyer messages au moniteur | Transmet les informations vers le moniteur | État connexion du moniteur  Informations pour le moniteur | Informations pour le moniteur |
| Démarrer robot sans watchdog | Démarre le robot sans watchdog | Ordre de démarrage robot sans wd | État démarrage robot |
| Mettre en place la communication avec le robot | Ouvre la communication avec le robot | Ordre d’ouverture de la communication robot | État communication robot |
| Déplacer robot | Déplace le robot | État démarrage robot  Ordre de déplacement | Mouvement du robot |
| Batterie robot | Récupère le niveau de batterie du robot et le transmet au moniteur. | État démarrage robot  État communication avec robot | Niveau de batterie |
| Position robot | Calcule la position du robot et les transmet vers l’image de la caméra et le moniteur périodiquement (100 ms) | Flux vidéo entrant  Ordre de calcul de la position | Coordonnées calculées  Message d’erreur |
| Démarrer caméra | Allume le module caméra | Ordre de démarrage caméra | État caméra |
| Arrêter caméra | Ferme le module caméra | Ordre d’arrêt caméra | État caméra |
| Envoi image | Transmet un flux d’images périodiquement (100 ms) | Flux vidéo entrant  État caméra  Demande d’arène  Demande de position | Flux vidéo sortant |
| Gestion arène | Recherche et transmission d’une image de l’arène. Sauvegarde l’image ou non selon le retour utilisateur.  Modifie la fonction Envoi image pour centrer sur l’arène. | Ordre de recherche d’arène  Validation de l’arène par l’utilisateur  Flux vidéo entrant | Flux vidéo sortant  Ordre d’arrêt de la fonction Envoi image |

Ce travail préliminaire nous permet de mettre en lumière des blocs fonctionnels que nous devrons réaliser pour remplir notre mission.

C’est un bon moyen de segmenter les tâches, ce qui permettra de coder et valider leur fonctionnement séparément, étape par étape (branches git différentes, répartition des tâches dans l’équipe).

Il s’agit également d’une base pour visualiser les différents liens entre ces tâches et les éléments du système qui entrent en jeux dans chacune de leur réalisation. C’est ce que nous avons fais par la suite avec le diagramme d’architecture fonctionnelle.

Architecture fonctionnelle statique :

Une fois ce travail réalisé, nous avons à présent une excellente vision des tâches de notre futur superviseur et de la façon dont elles s’articuleront entre les différents éléments du système.

Ce premier concept reste néanmoins théorique et ne considère que les besoins du client à proprement parler. Pour s’approcher d’une solution faisable, il est nécessaire d’intégrer cette première étude dans le cadre de nos outils et moyens de réalisation de la solution.

# Architecture physique

À présent, nous allons faire le lien entre les fonctions théoriques évoquées précédemment et nos outils de travail.

Comme évoqué en introduction, nous avons à disposition une Raspberry Pi dans laquelle nous allons implémenter notre programme superviseur. Le code en question est développé en C, tout en ayant recours à l’API Xenomai qui permet d’utiliser des fonctions typiquement orientées temps réel.

Entre autres, ces fonctions vont nous permettre de mettre en place la base de l’ordonnancement des tâches en temps réel ; les sémaphores et les mutex. Les sémaphores nous serviront à synchroniser l’accès à une tâches par les différents éléments du système, tandis que les mutex serviront à bloquer l’accès à une variable par plusieurs threads simultanément.

Pour mettre en place ce réseau de gestion des tâches et de communication entre elles, il est important dans un premier temps de toutes les définir précisément

Choix et justification d’une organisation en constituants (découpage/regroupement des fonctions en tâches), caractérisation des tâches :

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Nom de la tâche | Rôle | Entrées | Sorties | Activation / Période | Priorité |
|  |  |  |  |  |  |
| T\_SEND\_TO\_MON | Envoyer des informations au moniteur. | Information pour moniteur | Information transmise au moniteur | Unique, automatique après démarrage serveur | 22 |
| T\_RECEIVE\_FROM\_MON | Traiter les informations du moniteur. | Information envoyée par moniteur | Traitement de l’information | Unique, automatique après démarrage serveur | 25 |
| T\_SERVER | Démarrer le serveur de communication moniteur / superviseur | Démarrage système | Serveur démarré | Unique, automatique au démarrage système | 30 |
| T\_OPEN\_COM\_ROBOT | Démarrer la communication entre le superviseur et le robot | Ordre d’ouverture de la com robot | État com robot | Unique, au démarrage système | 20 |
| T\_START\_ROBOT | Démarrer le robot. | Ordre démarrage robot | Robot démarré | Unique, demande utilisateur | 20 |
| T\_MOVE | Déplacer le robot sur réception d’une consigne. | Ordre de déplacement du robot | Mouvement du robot | Périodique : 100 ms | 20 |
| T\_BATTERY\_STATE | Acquérir et transmettre le niveau de batterie. | État démarrage robot | Niveau de batterie | Périodique : 500 ms | 19 |
| T\_START\_CAMERA | Démarrer la caméra. | Ordre démarrage caméra | Caméra démarrée | Unique, demande utilisateur | 21 |
| T\_STOP\_CAMERA | Arrêter la caméra. | Ordre de stop caméra | Caméra arrêtée | Unique, demande utilisateur | 22 |
| T\_SEND\_IMG | Envoyer des images. | État caméra démarrée, État recherche d’arène, État recherche position | Flux vidéo | Périodique : 100 ms | 19 |
| T\_SEARCH\_MY\_ARENA | Recherche une image de l’arène. | Ordre de recherche d’arène | Image de l’arène et demande de validation | Unique, demande utilisateur | 20 |
| T\_FIND\_POSITION | Calcule les coordonnées du robot. | Ordre de recherche de position | Coordonnées ou message d’erreur | Périodique : 100 ms | 19 |

Une fois ce travail effectué, nous avons pu déterminer pour chaque tâche la nécessité ou non de protéger leur accès par un mutex ou de synchroniser leur accès par un sémaphore. Ces choix ont été basés sur les données communiquées entre les différentes tâches.

Choix et justification des moyens de communication et de synchronisation

|  |  |  |  |
| --- | --- | --- | --- |
| Donnée échangée entre des tâches  (voir diagramme d’architecture logique statique) | Mode de communication (variable globale, messageQueue, …) et caractérisation  (Id, nom, taille, timeout, ..) | Protection (ou pas) par Mutex et Id du Mutex | Justification |
|  |  |  |  |
| Accès fonctions moniteur | Variable globale | Mutex\_monitor | Donnée accédée par plusieurs tâches |
| Accès fonctions robots | Variable globale | Mutex\_robot | Donnée accédée par plusieurs tâches |
| État démarrage robot | Variable globale | Mutex\_robotStarted | Donnée accédée par plusieurs tâches |
| Ordre déplacement robot | Variable globale | Mutex\_move | Donnée accédée plusieurs fois par la même tâche |
| État démarrage caméra | Variable globale | Mutex\_Cam | Donnée accédée par plusieurs tâches |
| Communication caméra | Variable gloable | Mutex\_Cam | Donnée accédée par plusieurs tâches |
| Activation recherche d’arène | Variable globale | Mutex\_SearchingArena | Donnée accédée par plusieurs tâches |
| Validation d’arène par utilisateur | Variable globale | Mutex\_UseArena | Donnée accédée par plusieurs tâches |
| Image de l’arène | Variable globale | Mutex\_ArenaResult | Donnée accédée par plusieurs tâches |
| Activation du calcul de position | Variable globale | Mutex\_CalculPosition | Donnée accédée par plusieurs tâches |
| Coordonnées robot | Variable globale | Mutex\_robotPosition | Donnée accédée par plusieurs tâches |
| État com robot | messageQueue | - | Donnée non utilisée par d’autres tâches |
| Flux vidéo | messageQueue | - | Donnée non utilisée par d’autres tâches |
| Niveau de batterie | messageQueue | - | Donnée non utilisée par d’autres tâches |
| État caméra | messageQueue | - | Donnée non utilisée par d’autres tâches |
| État démarrage robot | messageQueue | - | Donnée non utilisée par d’autres tâches |
| Message moniteur | messageQueue | - | Donnée non utilisée par d’autres tâches |

|  |  |  |  |
| --- | --- | --- | --- |
| Tâches à synchroniser | Événement à signaler | Id du sémaphore binaire qui représente l’événement | Justification |
|  |  |  |  |
| Toutes les tâches | Démarrage système | Sem\_Barrier | Permet de lancer toutes les tâches simultanément au démarrage. |
| T\_OPEN\_COM\_ROBOT | La communication avec le robot est demandée | Sem\_OpenComRobot | La communication avec le robot doit être initiée. |
| T\_SEND\_TO\_MON,  T\_RECEIVE\_FROM\_MON | La communication entre le superviseur et le moniteur est établie | Sem\_ServerOK | Le serveur doit être correctement démarré avant de tenter de communiquer avec le moniteur. |
| T\_START\_ROBOT | Le démarrage du robot est demandé | Sem\_StartRobot | Le démarrage du robot doit être demandé par l’utilisateur. |
| T\_START\_CAMERA | L’ouverture de la caméra est demandée | Sem\_StartCam | L’ouverture de la caméra doit être demandée par l’utilisateur. |
| T\_STOP\_CAMERA | La fermeture de la caméra est demandée | Sem\_StopCam | La fermeture de la caméra doit être demandée par l’utilisateur. |
| T\_SEARCH\_MY\_ARENA | L’arène trouvée est validée | Sem\_ValidArena | L’arène trouvée doit être validée par l’utilisateur. |
| T\_SEARCH\_MY\_ARENA | La recherche d’arène est demandée | Sem\_SearchingArena | La recherche d’arène doit être demandée par l’utilisateur. |

À présent, il nous est possible de construire un diagramme pour visualiser l’ensemble de notre réseau de tâches.

Architecture physique statique :

Une image contenant texte, diagramme, capture d’écran, Police

Description générée automatiquement

Une image contenant texte, diagramme, capture d’écran, cercle

Description générée automatiquement

Une image contenant texte, capture d’écran, diagramme, cercle

Description générée automatiquementUne image contenant texte, capture d’écran, diagramme, Police

Description générée automatiquementUne image contenant texte, diagramme, cercle, capture d’écran

Description générée automatiquement

Une image contenant texte, diagramme, cercle, capture d’écran

Description générée automatiquement

Une image contenant texte, diagramme, capture d’écran, Police

Description générée automatiquement

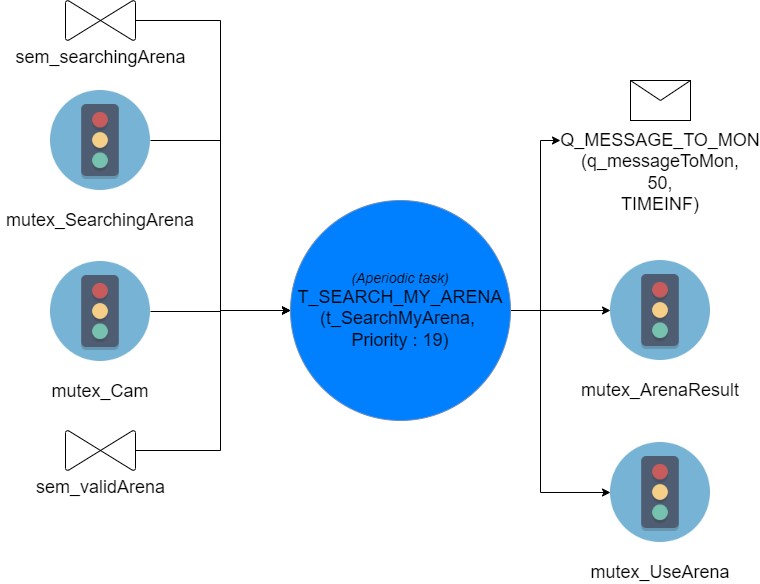
Une image contenant texte, capture d’écran, diagramme, Police

Description générée automatiquementUne image contenant texte, diagramme, capture d’écran, Police

Description générée automatiquement

Une image contenant texte, capture d’écran, diagramme, cercle

Description générée automatiquement



Une image contenant texte, capture d’écran, diagramme, cercle

Description générée automatiquement

# Codage et livraisons incrémentales

Après avoir terminé la phase de conception, nous avons pu entamer la phase de développement du logiciel de supervision.

Les éléments théoriques vus précédemment nous ont permis de structurer cette partie du travail pour gagner en efficacité et optimiser les chances d’atteindre nos objectifs dans le temps imparti.

Stratégie de codage et d’intégration

L’intérêt d’avoir regroupé nos fonctions logicielles en tâches est que nous avons pu segmenter le codage et l’intégration de celles-ci.

En effet, notre stratégie de développement s’est basé sur le traitement complet d’une tâche avant de s’occuper de la suivante.

L’intérêt est ici de pouvoir compléter la seconde partie du cycle en V (production et validation) pour chaque tâche.

Une image contenant Bleu électrique, capture d’écran, ligne, bleu

Description générée automatiquement

Ceci permet d’implémenter une tâche sur le système et de ne plus avoir à se préoccuper de celle-ci. Puisque nous allons jusqu’à la phase de test et validation de la tâche, nous éliminons les risques d’erreurs liés à celle-ci lorsque nous poursuivons le codage de la tâche suivante. Il s’agit donc d’une sécurité précieuse pour notre développement, d’autant plus que certaines tâches sont étroitement liées au fonctionnement des autres (ex. Envoi image doit être fonctionnelle pour implémenter Calcul position et Recherche arène).

Cette stratégie est également pertinente lorsqu’elle est couplée avec l’utilisation de git. Il est ainsi possible de sauvegarder une version du code pour chaque tâche complètement finalisée.

Au niveau de notre gestion personnelle, nous avons trouvé judicieux d’avancer le travail de programmation sur notre temps libre pour consacrer la majorité de nos séances avec matériel aux tests et validation.

Nous avons principalement avancé de pairs pour coller à notre stratégie de codage.

Nous avons cependant pu nous distinguer lorsque le codage d’une tâche fut terminé. À ce moment, l’un d’entre nous a pu s’occuper de la validation de la tâche achevée, permettant au second de commencer la programmation de la tâche suivante. Ce petit gain de temps fut nécessaire compte tenu des délais que nous avions.

Il nous est tout de même arrivés de devoir nous re synchroniser sur la même tâche lorsque les tests de celles-ci n’étaient pas concluants.

L’avance de phase gagnée en divisant notre travail ne devait pas aller à l’encontre du principe de validation des tâches les unes à la suite des autres que nous avions fixé.

# Analyse et validation du logiciel livré par rapport aux exigences

Suite au développement du superviseur, nous avons pu répondre aux contraintes initiales.

En voici un bilan détaillé :

|  |  |  |
| --- | --- | --- |
| Numéro exigence | Description de l’exigence | État |
|  |  |  |
| 1 | **Récupérer le niveau de batterie du robot et l’afficher sur le moniteur.** | **Fonctionnel** |
| 2 | **Ouvrir la caméra.** | **Fonctionnel** |
| 3 | **Fermer la caméra.** | **Fonctionnel** |
| 4 | **Transmettre le flux d’images de la caméra au moniteur après ouverture de la caméra.** | **Fonctionnel** |
| 5 | **Détecter l’’arène et la faire valider par l’utilisateur.** | **Fonctionnel** |
| 6 | **Calculer la position du robot et la transmettre au moniteur.** | **Fonctionnel** |

# Commentaires et conclusion

À l’issue de cette étude, nous sommes parvenus à répondre à la totalité des besoins du client.

En effet, les 6 exigences sont parfaitement prises en charge par la solution que nous avons développée. Nous sommes donc fiers de pouvoir présenter un outil fonctionnel et finalisé.

Une piste d’amélioration pour la suite de ce projet serait de détecter l’ID du cryptogramme de notre robot pour pouvoir différencier sa position de celle d’éventuels autres robots sur l’arène.