

# SAE 5.01 **RAPPORT**

*Conception d'un robot connecté*

2024-2025

GEii - FA - ESE1

AGA Amel & GADA Ayoub

## Table des matières

1.	Introduction .....	3
1.1.	Sujet .....	3
1.2.	Compétences ciblées.....	4
1.3.	Apprentissages critiques .....	4
2.	Partie conception théorique .....	5
2.1.	Introduction partie conception théorique .....	5
2.2.	Cahier des charges à respecter : .....	5
2.3.	Choix des composants.....	5
2.4.	Fonctionnement global théorique .....	6
2.5.	Vérifications des compatibilités.....	7
3.	Analyse fonctionnelle .....	10
3.1.	Schéma synoptique .....	10
3.2.	Diagramme bête à cornes .....	10
3.3.	Diagramme pieuvre.....	11
4.	Organisation et répartitions des tâches .....	12
5.	Partie Bluetooth .....	13
5.1.	Communication grâce au module et utilisation du potentiomètre .....	13
5.2.	Mise en place du système IMU .....	15
6.	Partie moteur & encodeur .....	18
7.	Partie FreeRTOS .....	19
7.1.	Objet.....	19
7.2.	Logique du système.....	19
7.3.	Ordonnance des tâches.....	20
7.4.	Assemblage des fonctions dans un unique code .....	20
8.	Conclusion.....	23
8.1.	Le travail qui a été fournis et les compétences acquises .....	23
8.2.	Points d'améliorations .....	23

# 1. Introduction

## 1.1. Sujet

Ce rapport présente le travail effectué dans le cadre de la SAE 5.02 « Mettre en œuvre un système électronique et / ou embarqué spécifique du BUT Génie Électrique et Informatique Industrielle, parcours ESE, en 3<sup>e</sup> année, semestre 5. Notre groupe, composé de GADA Ayoub et AGA Amel, a travaillé sur la conception d'un robot autonome répondant à un cahier des charges précis.

L'objectif du projet était de concevoir un robot capable de se diriger vers un faisceau lumineux tout en évitant de manière autonome les obstacles présents sur son chemin. Nous avons donc pris en charge l'ensemble de la phase de conception en veillant à respecter les exigences du cahier des charges.

Ce projet va principalement être séparé en deux parties, une concernant la partie Bluetooth que nous avons mise au point et une autre traitant de la partie moteur / encodeur ainsi que de l'assemblage des fonctions pour un système RTOS (real-time operating system).

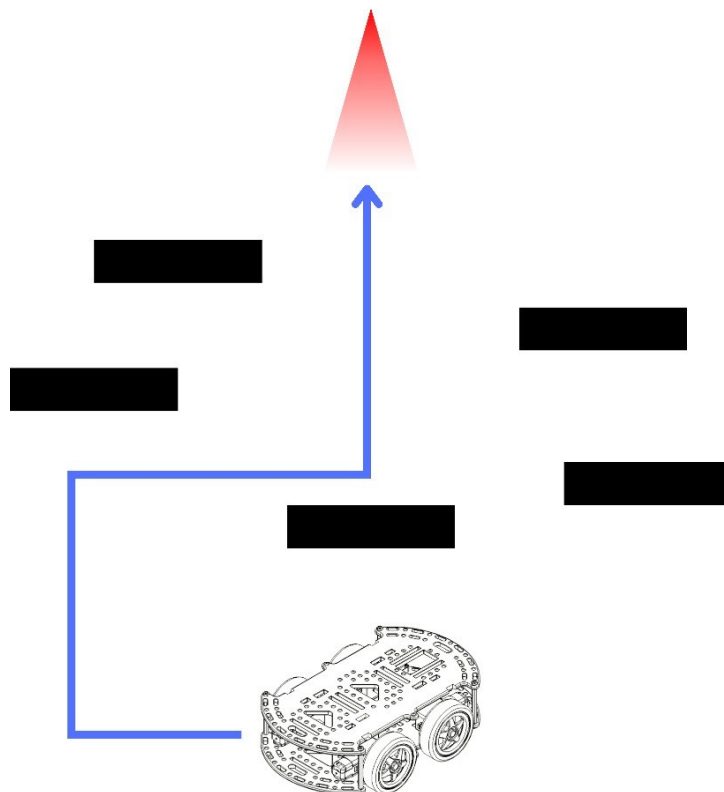


Illustration de l'objectif du projet

## 1.2. Compétences ciblées

Durant ce projet, nous avons développé plusieurs compétences :

- **Assurer** le maintien en condition opérationnelle d'un système.
- **Implanter** un système matériel ou logiciel.
- **Concevoir** la partie GEII d'un système.
- **Vérifier** la partie GEII d'un système.

## 1.3. Apprentissages critiques

Au cours de cette SAE, nous allons mobiliser et approfondir plusieurs apprentissages critiques essentiels à la conception, la mise en service et la maintenance d'un système électronique ou embarqué. Ces compétences nous permettront de répondre efficacement aux exigences du projet et aux défis techniques rencontrés.

### Maintenance et gestion de l'indisponibilité

- AC33.01 : Proposer une solution de maintenance adaptée au système.
- AC33.02 : Évaluer les coûts d'indisponibilité et de maintenance afin d'optimiser la gestion du système.
- AC33.03 : Produire une procédure de maintenance claire et efficace.
- AC33.04 : Fournir un appui technique aux différents acteurs à l'échelle nationale et internationale.

### Mise en service et conformité

- AC34.02ESE : Exécuter la mise en service d'un système en respectant la procédure établie.
- AC34.03ESE : Produire le dossier de conformité du système tout en assurant une gestion rigoureuse du versionnage.

### Conception et justification technique

- AC31.01 : Contribuer à la rédaction d'un cahier des charges détaillé et structuré.
- AC31.02 : Justifier la pertinence des choix technologiques retenus pour le projet.
- AC31.03 : Rédiger un dossier de conception complet et documenté.

### Analyse des dysfonctionnements et validation

- AC32.01 : Identifier la cause racine d'un dysfonctionnement sur le système.
- AC32.02 : Proposer des solutions correctives adaptées pour remédier aux anomalies détectées.
- AC32.03 : Élaborer une procédure d'essais permettant de valider la conformité et le bon fonctionnement du système.
- Grâce à ces apprentissages critiques, nous serons en mesure de mener à bien la conception, la mise en place et la validation de notre projet tout en garantissant sa fiabilité, sa conformité et sa maintenabilité.

## 2. Partie conception théorique

### 2.1. Introduction partie conception théorique

Dans cette phase du projet, nous allons réaliser la conception théorique de notre robot. Cela implique la mise en place d'un système fonctionnel en sélectionnant rigoureusement les composants tout en respectant les exigences du cahier des charges et en optimisant les coûts de production.

Cette étape est essentielle dans le cadre d'un projet professionnel, car elle permet d'anticiper les contraintes techniques, d'évaluer la faisabilité et de garantir l'efficacité du système avant son implémentation pratique.

### 2.2. Cahier des charges à respecter :

Pour assurer le bon fonctionnement du robot, nous devons respecter les spécifications suivantes :

- Alimentation : Le robot doit obligatoirement être alimenté par deux piles de 3,7V en série, fournissant une tension totale de 7,4V.
- Microcontrôleur : Le système doit être contrôlé par un microcontrôleur **NUCLEO-F446RE**, dont la documentation constructeur est disponible [ici](#).
- Communication : Le robot doit utiliser un système de communication sans fil (exemple : Bluetooth, Wi-Fi, etc.).
- Autonomie et mobilité : Il doit être autonome en énergie, capable de se déplacer de manière fluide d'un point A à un point B.
- Coût : Le coût de production doit rester raisonnable afin d'optimiser la viabilité du projet.

Le choix des composants sera réalisé en fonction des critères techniques définis ci-dessus. Nous devons nous assurer que chaque composant est compatible avec le microcontrôleur et répond aux exigences en termes d'alimentation, de communication et de mobilité. Pour cela, nous analyserons attentivement les documentations constructeurs de chaque élément sélectionné.

Cette approche nous permettra d'anticiper d'éventuels problèmes d'intégration et d'optimiser les performances du robot avant la phase de mise en œuvre pratique.

### 2.3. Choix des composants

Nous avons d'abord commencé à déterminer une liste de composants qui nous paraissent adapter à notre projet et à nos critères et que nous allons ensuite étudier en détails pour déterminer si oui ou non ces composants sont adaptés.

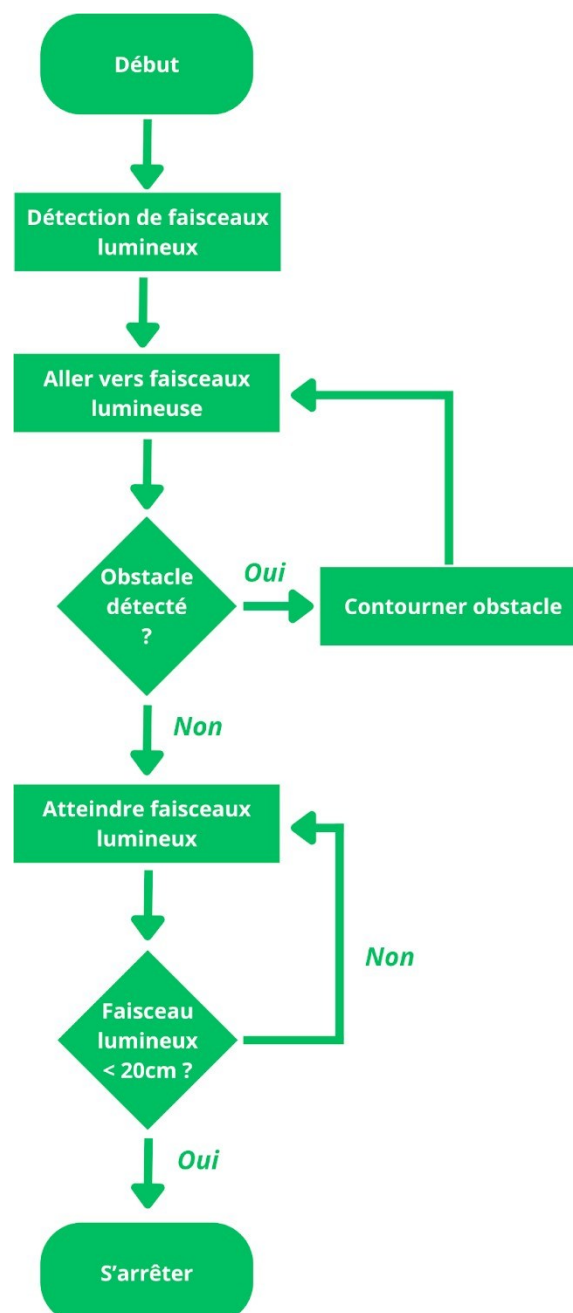
Liste :

- Module moteur DRV8833 : [datasheet](#)
- Capteur photosensible pour détecter le signal lumineux : [achat + datasheet](#)
- Capteur laser pour éviter les obstacles : [achat + datasheet](#)
- Roues : [achat + datasheet](#)
- Roue folle : [achat + datasheet](#)
- Module Bluetooth HC05 : [achat + datasheet](#)

- Régulateur de tension MIC2915X pour que les roues soient continuellement correctement alimentées : [datasheet](#)
- Carton renforcé pour la création du châssis : [achat + datasheet](#)
- Vis et boulons : [achat + datasheet](#)
- Moteur + encodeur : [achat + datasheet](#)

#### 2.4. Fonctionnement global théorique

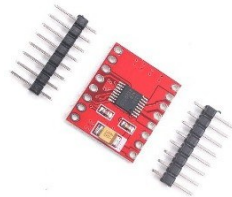



Afin de mieux comprendre le fonctionnement de notre robot, nous avons réalisé un premier schéma théorique global du fonctionnement de celui-ci, cela va nous permettre de mieux anticiper les problématiques pour le reste du projet.







## 2.5. Vérifications des compatibilités

Afin de nous assurer du bon fonctionnement de notre robot, nous devons vérifier si les caractéristiques techniques des composants que nous avons sélectionnés sont compatibles avec les caractéristiques techniques qui nous sont imposées.

Pour cela nous allons visualiser cela sous forme de tableau (à partir de page suivante) :

Nom	Caractéristiques	Compatibilités	Avantages	Défauts	Image
DRV8833 Dual H-Bridge Motor Driver	<ul style="list-style-type: none"> <li>• Transistor Mosfet</li> <li>• Pont en H</li> </ul>	Oui	<ul style="list-style-type: none"> <li>• Faible résistance</li> </ul>		
Capteur photosensible	<ul style="list-style-type: none"> <li>• 3 Pin Photosensitive Sensor Module</li> </ul>	Oui	<ul style="list-style-type: none"> <li>• Tension de fonctionnement : 3,3V-5V</li> </ul>		
Régulateur MIC2915X		Oui	<ul style="list-style-type: none"> <li>• <math>V_{IH} &lt; 1V</math></li> <li>• <math>R_{DC} = 4\Omega</math></li> <li>• LDO</li> </ul>		
Capteur laser		Oui	<ul style="list-style-type: none"> <li>• Range de 150cm</li> <li>• Tension de fonctionnement : 5V</li> <li>• Consommation de 30mA</li> </ul>		



			<ul style="list-style-type: none"> <li>Compatible STM32</li> </ul>		
Module bluetooth HC-005	<ul style="list-style-type: none"> <li>2.0 + EDR (Enhanced Data Rate)</li> <li>Vitesse de transmission : 9600 bps par défaut (modifiable)</li> </ul>	Oui	<ul style="list-style-type: none"> <li>Communication UART</li> <li>Maître et esclave</li> <li>10m de portée</li> </ul>		
Roue folle		Oui	<ul style="list-style-type: none"> <li>Meilleure gestion des directions</li> </ul>	<ul style="list-style-type: none"> <li>Moins bonne répartition du poids</li> </ul>	
Roue		Oui	<ul style="list-style-type: none"> <li>Compatible entre 3V-12VDC</li> </ul>		
Kit moteur + encodeur FIT0450	6 Vcc - 160 t/min - 0,8 kg.cm	Oui	<ul style="list-style-type: none"> <li>Permet de faire un système PPL</li> </ul>		

### 3. Analyse fonctionnelle

Dans cette partie nous listons les analyses fonctionnelles nécessaires à notre projet.

#### 3.1. Schéma synoptique

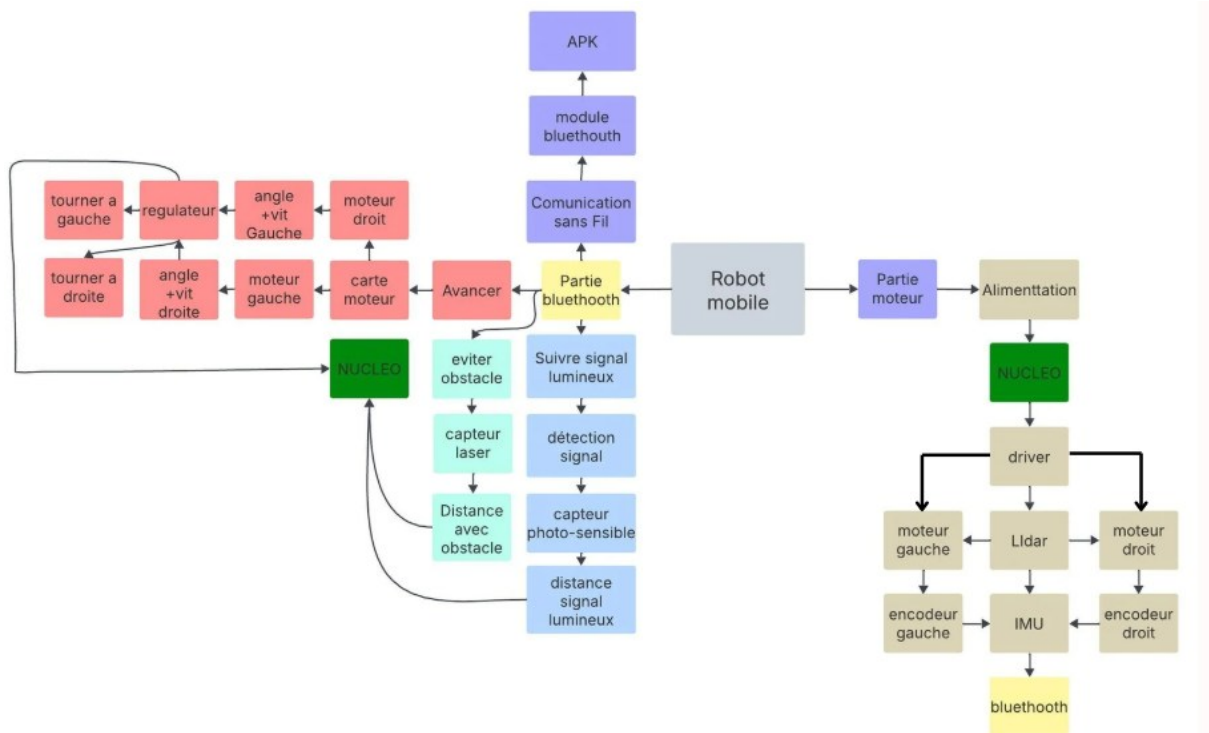


Schéma synoptique

#### 3.2. Diagramme bête à cornes

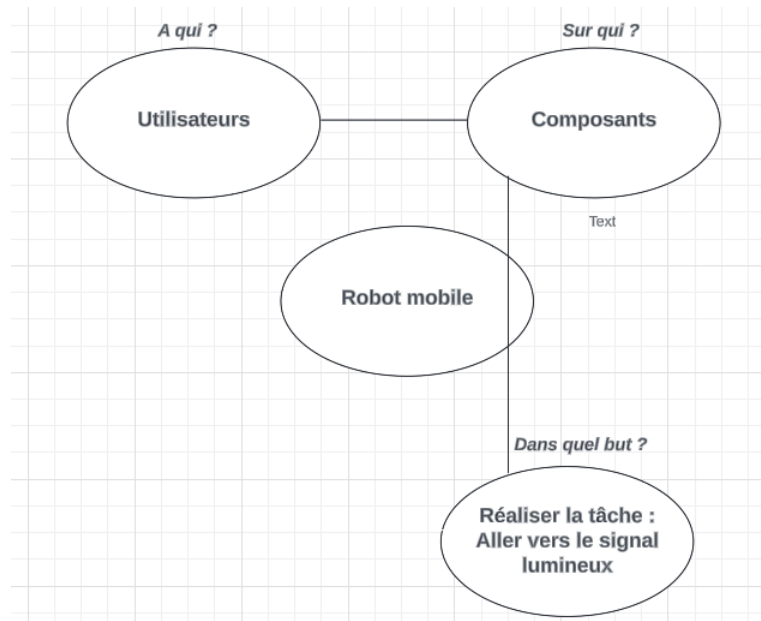


Diagramme bête à cornes

### 3.3. Diagramme pieuvre

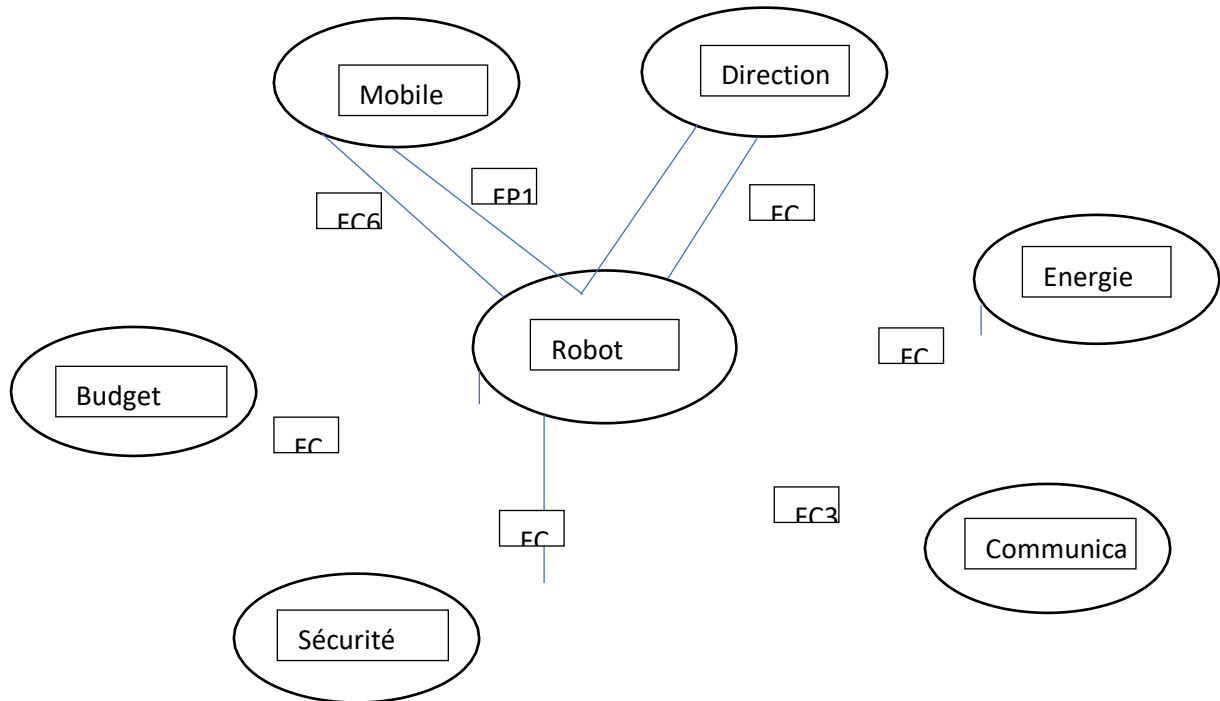


Diagramme pieuvre

- FP1 : Le robot doit être mobile et pouvoir se diriger vers le signal lumineux.
- FC1 : Le robot doit pouvoir se diriger vers le signal lumineux.
- FC2 : Le robot doit être énergétiquement autonome.
- FC3 : Il doit pouvoir communiquer sans fils.
- FC4 : Il doit pouvoir contourner des obstacles.
- FC5 : Il doit avoir un prix raisonnable.
- FC6 : Il doit pouvoir se déplacer.

## 4. Organisation et répartitions des tâches

À l'issue de la phase de conception théorique, nous avons acquis une compréhension approfondie des spécificités techniques du projet ainsi que des différentes tâches à réaliser pour mener à bien sa mise en œuvre.

Afin d'optimiser notre efficacité et de garantir une gestion rigoureuse du projet, nous avons défini une organisation claire et structuré notre travail à l'aide d'un diagramme de Gantt, présenté ci-dessous :

	Semaine 1	Semaine 2	Semaine 3	Semaine 4	Semaine 5	Semaine 6	Semaine 7	Semaine 8	Semaine 9	Semaine 10	Semaine 11	Semaine 12
	Conception théorique		Partie bluetooth			Partie moteur					rendu SAE	
Sélection des différents composants du robot	Ayoub	Amel										
Conception électronique et branchement		Ayoub	Amel									
Étude du microcontrôleur et développement du code de base		Ayoub	Amel									
Programmation du capteur photosensible			Amel									
Intégration Bluetooth			Ayoub	Amel								
Tests complets			Ayoub	Amel								
Calcul des Paramètres du Moteur						Ayoub	Amel					
Programmation et test des encodeurs							Ayoub	Amel				
Programmation et test capteur								Ayoub	Amel			
Programation final avec RTOS									Amel			
Rapport			Rapport avancement projet					Rapport avancement projet		oral SAE	Rapport projet	

Ce planning nous servira de référence tout au long du projet. En respectant cette organisation, nous visons à améliorer notre productivité, à assurer une bonne répartition des tâches et à anticiper d'éventuels défis techniques.

## 5. Partie Bluetooth

### 5.1. Communication grâce au module et utilisation du potentiomètre

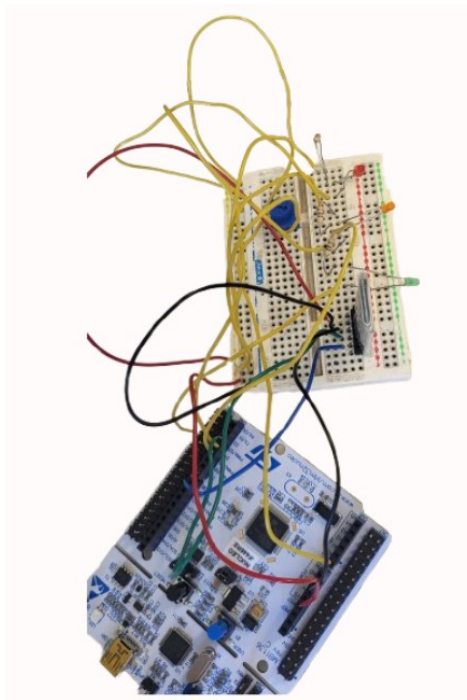
L'objectif de cette partie Bluetooth est d'établir une connexion entre notre objet (robot mobile) et un APK sur téléphone.

Cette application devra servir à l'utilisateur pour visualiser diverses données du robot en direct.

Pour cela on va utiliser une carte de développement STM32 – NUCLEO – F446RE qui va nous servir de cerveau ainsi qu'un module Bluetooth HM-10.

Pour simuler une communication entre notre module, notre carte et un APK, on va envoyer des commandes AT qui doivent être reçus sur un terminal Bluetooth sur notre téléphone.

Pour envoyer les commandes AT, nous réalisons d'abord le montage électrique suivant.



*Montage réalisé pour la communication Bluetooth*

Dans montage on constate également un potentiomètre relié à des LEDs en plus du montage permettant de faire fonctionner le module Bluetooth.

En effet, ce potentiomètre va nous permettre de simuler l'affichage de l'état de la batterie du robot, les LEDs nous servent uniquement pour vérifier que notre système fonctionne et est cohérent. Chaque couleur de LED délimite une plage de valeur.

Pour envoyer nos commandes AT nous utilisons le code suivant.

```
char *message = "Ayoub & Amel SAE5.01 GEii\r\n"; // Message à envoyer
char *renameCommand = "AT+NAMEpostel\r\n"; // Commande pour renommer le module Bluetooth
uint16 t readValue;
```

### *Déclaration commandes AT et de la variable pour lire les messages*

```
void Send_AT_Command(char* command);
```


### *Envoi d'une commande AT au module Bluetooth*

```
void Send_AT_Command(char* command) {  
    HAL_UART_Transmit(&huart1, (uint8_t*)command, strlen(command), HAL_MAX_DELAY);  
}  
uint16_t readValue;
```

### *Fonction d'envoi de commande AT via UART et déclaration d'une variable pour la lecture de données*

```
if (readValue <= 1000) {  
    // Allumer la LED 1 (PB3), éteindre les autres  
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_3, GPIO_PIN_SET); // LED1 ON  
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_5, GPIO_PIN_RESET); // LED2 OFF  
    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_8, GPIO_PIN_RESET); // LED3 OFF  
} else if (readValue > 1000 && readValue <= 2000) {  
    // Allumer la LED 2 (PB5), éteindre les autres  
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_3, GPIO_PIN_RESET); // LED1 OFF  
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_5, GPIO_PIN_SET); // LED2 ON  
    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_8, GPIO_PIN_RESET); // LED3 OFF  
} else if (readValue > 2000) {  
    // Allumer la LED 3 (PA8), éteindre les autres  
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_3, GPIO_PIN_RESET); // LED1 OFF  
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_5, GPIO_PIN_RESET); // LED2 OFF  
    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_8, GPIO_PIN_SET); // LED3 ON  
}
```

### *Contrôle des LEDs en fonction des valeurs du potentiomètre*

Expression	Type	Value
(x)= readValue	uint16_t	166
 Add new expression		

### *Affichage des valeurs du potentiomètre*

Comme on peut le constater dans cette partie, nous n'avons pas affiché les valeurs du potentiomètre sur un terminal Bluetooth, nous les avons uniquement affiché sur le console de lecture d'expressions sur STM32CubeIDE, c'est un manquement de notre part.

Par contre nous avons bien affiché les commandes AT sur notre terminal Bluetooth et nous avons également renommé notre module « poste1 » (représentant notre numéro de poste).

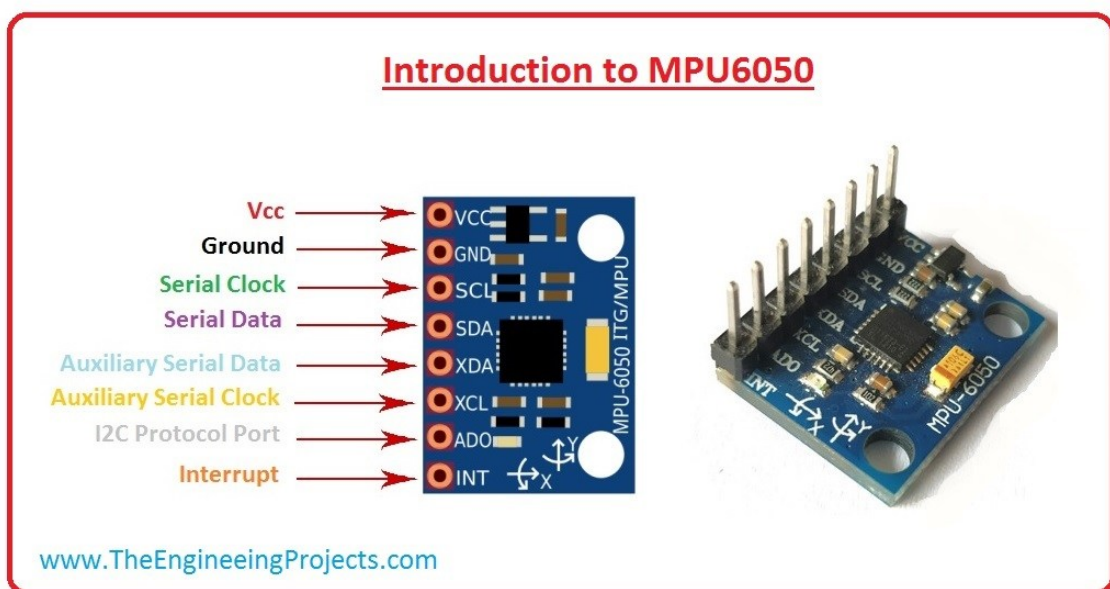


*Affichage sur notre console Bluetooth sur téléphone*

## 5.2. Mise en place du système IMU

Dans cette partie, nous devons mettre en œuvre un système IMU (unité de mesure inertielle) qui va nous permettre d'avoir différentes mesures de notre objet (comme l'angle  $\omega_G$  et  $\omega_D$ ) pour pouvoir repérer notre objet dans l'espace et d'ajuster nos mouvements.

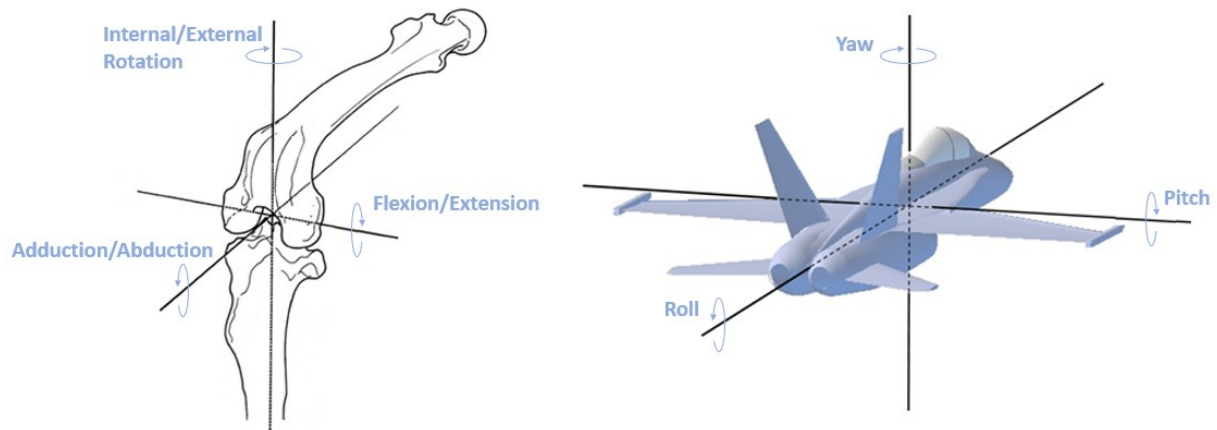
Pour cela, on va utiliser le module MPU6050.



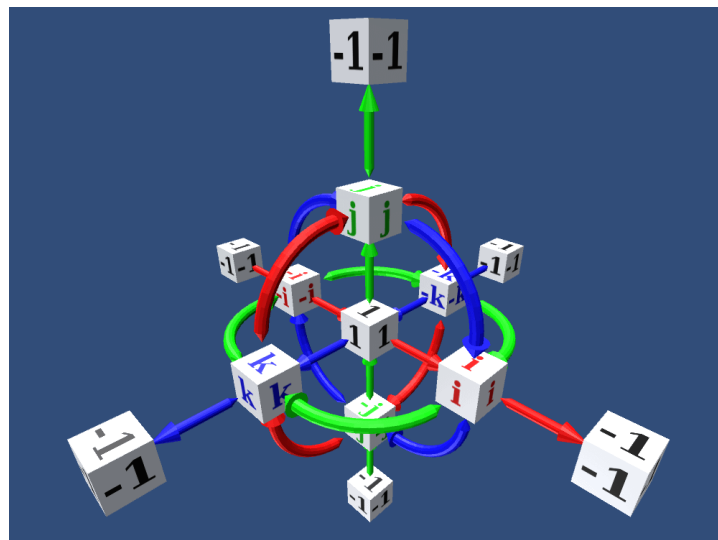
*Visuelle du MPU6050*

Le MPU6050 est un capteur inertielle combinant un accéléromètre et un gyroscope sur un même circuit. Il permet de mesurer l'accélération linéaire et la vitesse angulaire sur trois axes, ce qui le rend idéal pour les applications de stabilisation et de navigation des systèmes embarqués. Il communique via le protocole I2C et nécessite un traitement des données pour extraire les informations utiles.

Dans notre projet on va travailler avec des coordonnées orthogonales (x, y et z) mais ça aurait été préférable de travailler avec des angles d'Euler (Pitch, Roll et Yaw) notamment utilisé en aviation permettant de connaître la position de notre objet de manière plus précise ou même d'utiliser les quaternions.



*Visuelle des angles d'Euler*



*Visuelle des quaternions*

Pour réaliser notre système IMU, nous avons réalisé le montage que vous pouvez observer dans la vidéo suivante, retrouver dans cette vidéo le résultat de notre système : [Liens](#)

Voici le code que nous avons mis en place pour ce système :



```

void MPU6050_Read_Accel_Gyro(int16_t *AccelGyroData) {
    uint8_t Rec_Data[14]; // Buffer pour les données d'accélération et de gyroscope

    // Lire les 14 octets de données commençant à ACCEL_XOUT_H
    HAL_I2C_Mem_Read(&hi2c1, MPU6050_ADDR, MPU6050_REG_ACCEL_XOUT_H, 1, Rec_Data, 14, 1000);

    // Convertir les données en 16 bits
    AccelGyroData[0] = (int16_t)(Rec_Data[0] << 8 | Rec_Data[1]); // Accélération X
    AccelGyroData[1] = (int16_t)(Rec_Data[2] << 8 | Rec_Data[3]); // Accélération Y
    AccelGyroData[2] = (int16_t)(Rec_Data[4] << 8 | Rec_Data[5]); // Accélération Z
    AccelGyroData[3] = (int16_t)(Rec_Data[8] << 8 | Rec_Data[9]); // Gyroscope X
    AccelGyroData[4] = (int16_t)(Rec_Data[10] << 8 | Rec_Data[11]); // Gyroscope Y
    AccelGyroData[5] = (int16_t)(Rec_Data[12] << 8 | Rec_Data[13]); // Gyroscope Z
}

void Transmit_MPU6050_Data(int16_t *AccelGyroData) {
    char buffer[100];
    sprintf(buffer, "Accel X: %d, Y: %d, Z: %d | Gyro X: %d, Y: %d, Z: %d\r\n",
        AccelGyroData[0], AccelGyroData[1], AccelGyroData[2],
        AccelGyroData[3], AccelGyroData[4], AccelGyroData[5]);
    HAL_UART_Transmit(&huart2, (uint8_t*)buffer, strlen(buffer), HAL_MAX_DELAY);
}

```

*Lecture et transmission des données d'accélération et de gyroscope du MPU6050*

## 6. Partie moteur & encodeur



## 7. Partie FreeRTOS

### 7.1. Objet

Dans cette partie notre objectif va être de rassembler les différentes fonctions que l'on a créées en un système RTOS.

Un RTOS (Real-Time Operating System) est un système d'exploitation conçu pour exécuter des tâches dans un temps déterminé et prévisible. Contrairement aux systèmes classiques, il garantit une gestion optimisée des priorités des tâches, de la mémoire et des ressources du processeur, ce qui le rend idéal pour les applications embarquées nécessitant une forte réactivité.

Dans notre projet, l'utilisation d'un RTOS permet d'assurer une exécution efficace des différentes tâches du robot, telles que la lecture des capteurs, la communication et le contrôle des moteurs, tout en respectant les contraintes de temps réel.

### 7.2. Logique du système

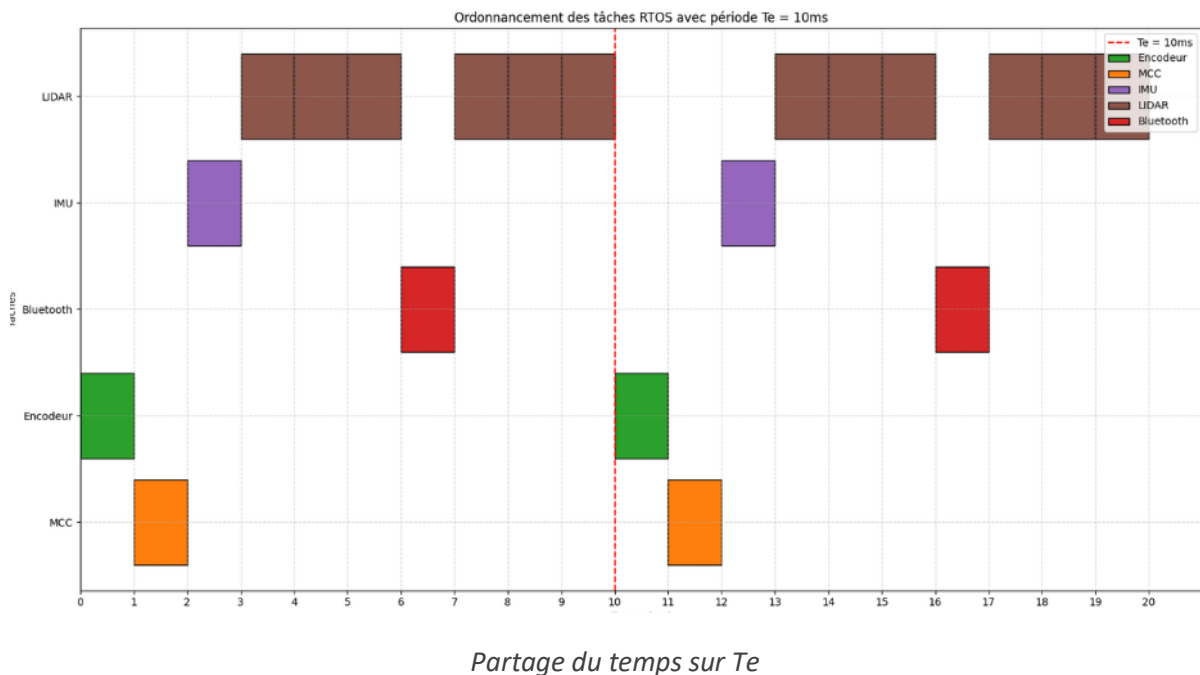
Pour mettre en place notre système RTOS, nous suivrons la logique suivante.

Tâche	Designation	Data in	Data out	Priority	Dépendances
Task_MCC	Control MCC	{ $\omega_d G$ , $\omega_d D$ }	PWM g/d	Haute priorité	Encodeur
Task_enc	Calcul $\omega_G$ , $\omega_D$	{A, B}	{ $\omega_{Gm}$ , $\omega_{Dm}$ }	Très Haute priorité	Aucune
Task_BT	Com_Robot Data	Mesure de la batterie		Basse priorité	Aucune
Task_IMU	Gyro/Acc	Coordonnée polaire ou angle d'Euler ou quaternion	Gyro/Acc	Haute priorité	Moteur
Task_Lidar	Abs	Signaux de retour	Orient	REAL TIME	Aucune

*Caractéristiques des tâches dans notre système*

### 7.3. Ordonnance des tâches

Et pour l'ordonnance du temps on va suivre le schéma suivant.



On privilégie les tâches les plus importantes comme LIDAR, Encodeur ou moteur aux tâches moins importantes comme la tâche Bluetooth.

### 7.4. Assemblage des fonctions dans un unique code

On configure donc notre projet en activant le FreeRTOS et on rentre nos différentes tâches avec leur priorité attribuée. Pour cela, nous avons utilisés le cours suivi dans la matière systèmes embarqués.

Ensuite, on intègre le code nécessaire :

```
/* Create the tasks */
xTaskCreate(MotorTask, "MotorTask", 128, NULL, 1, NULL);
xTaskCreate(EncoderTask, "EncoderTask", 128, NULL, 1, NULL);
xTaskCreate(BluetoothTask, "BluetoothTask", 128, NULL, 1, NULL);
xTaskCreate(EMUTask, "EMUTask", 128, NULL, 1, NULL);

/* Start the scheduler */
vTaskStartScheduler();
```

Création des tâches

```

void MotorTask(void *pvParameters)
{
    while (1)
    {
        TIM3->CCR1 = 30;
        TIM3->CCR2 = 60;
        HAL_TIM_PWM_Start(&htim3, TIM_CHANNEL_1);
        HAL_TIM_PWM_Start(&htim3, TIM_CHANNEL_2);
        vTaskDelay(pdMS_TO_TICKS(1000)); // Delay for 1 second
    }
}

```

*Tâche du moteur*

```

void EncoderTask(void *pvParameters)
{
    while (1)
    {
        // Implémenter la lecture de l'encodeur et le calcul du PID ici
        vTaskDelay(pdMS_TO_TICKS(100)); // Delay for 100 ms
    }
}

```

*Tâche de l'encodeur*

```

void BluetoothTask(void *pvParameters)
{
    char message[] = "Ayoub & Amel SAE5.01 GEii\r\n";
    while (1)
    {
        HAL_UART_Transmit(&huart2, (uint8_t*)message, strlen(message), HAL_MAX_DELAY);
        vTaskDelay(pdMS_TO_TICKS(1000)); // Delay for 1 second
    }
}

```

*Tâche du Bluetooth*

```

void IMUTask(void *pvParameters)
{
    int16_t AccelGyroData[6]; // Tableau pour stocker les données du MPU6050
    while (1)
    {
        MPU6050_Read_Accel_Gyro(AccelGyroData);
        Transmit_MPU6050_Data(AccelGyroData);
        vTaskDelay(pdMS_TO_TICKS(500)); // Delay for 500 ms
    }
}

```

*Tâche de l'IMU*

Nous n'avons pas intégré de tâche pour le LIDAR car nous ne sommes pas parvenus à faire fonctionner le système LIDAR, mais si nous l'avions intégré il aurait la priorité la plus haute.

Nous avons ensuite pu lancer le programme et il a fonctionné comme notre programme qui n'est pas dirigé RTOS, par contre il aurait aussi fallu que l'on gère la gestion du temps et l'intégration de la tâche la plus importante, LIDAR.

## 8. Conclusion

- 8.1. Le travail qui a été fourni et les compétences acquises
- 8.2. Points d'améliorations

