جامعة محمد الخامس بالرباط
Université Mohammed V de Rabat
المدرسة الوطنية العليا للفنون والمهن بالرباط
ⵜⵉⵎⵙⵙⵓⵜ ⵏ ⵜⵣⵣⵓⵍⵉ ⴰ ⵜⵣⵣⵓⵖⵍⵉ ⵖ ⵜⵎⵣⵡⴰⵔⵓ
Ecole Nationale Supérieure d'Arts et Métiers de Rabat

ENSAM Rabat, FI-EEIN/2

Elément de module : système temps réel

Activité pratique

Conception et Programmation d'un système de Commande d'Essuie-Glace :

Réalisé par :

KADDOURI Nassira

JEBBARI Hamza

HABBAZ Ayman

Enseignant :

Mr. A. JBARI

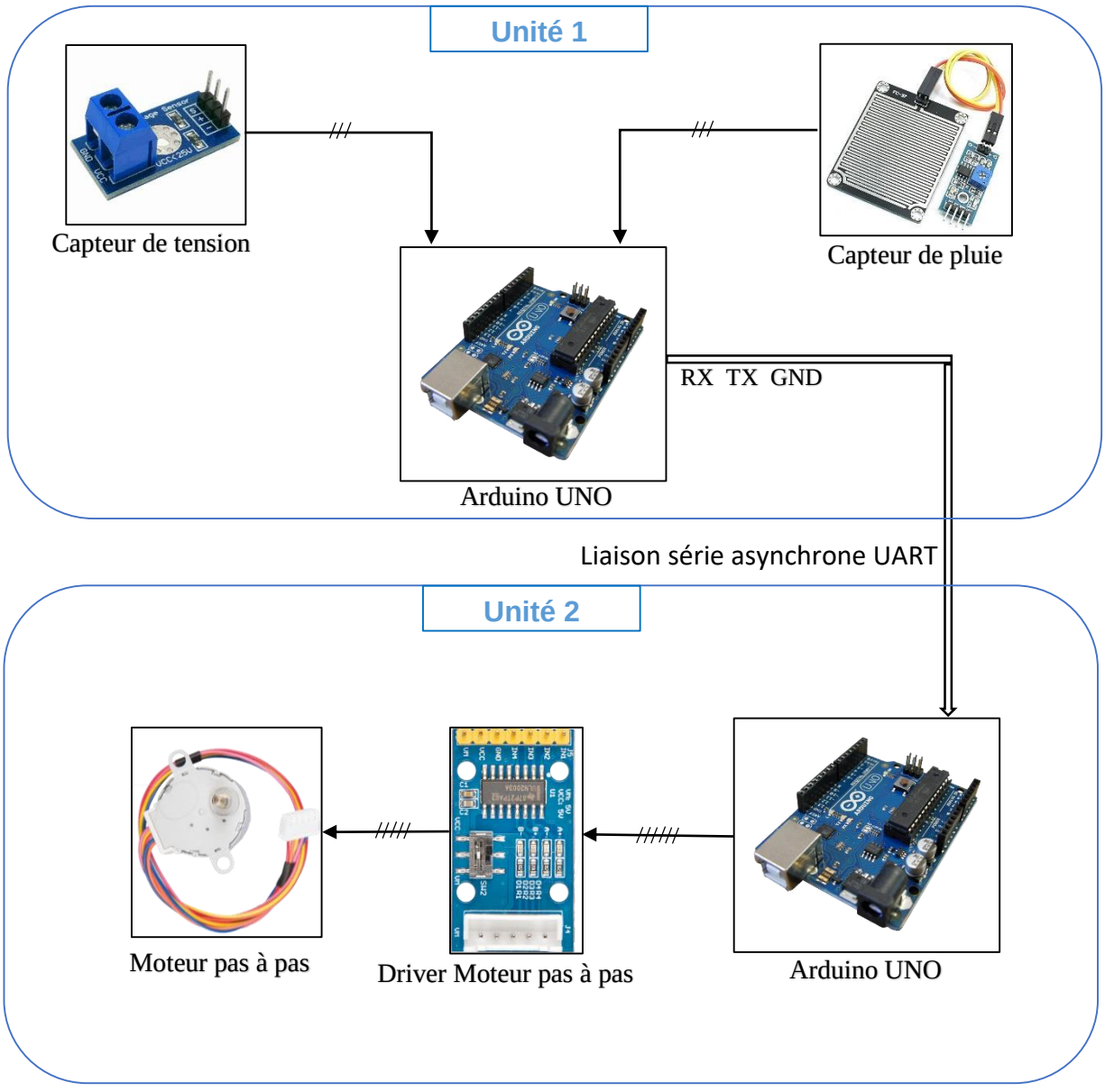
Année universitaire 2023-2024

Introduction :

Les activités pratiques représentent une occasion pour les étudiants de se familiariser avec les différents aspects et mécanismes de l'analyse et la réalisation technologique. Elles ont également pour intérêt, bien évidemment, de faire la liaison entre le théorique académique et la pratique.

Alors ce travail a été fait afin de réaliser une application de conception et programmation d'un système de commande d'Essuie-Glace sur la base de la mesure de niveau de pluie ainsi que le niveau de la batterie, ce travail nous a permis de se familiariser avec le package freeRTOS aussi que les algorithmes sémaphore et mutex et l'objet queue. Cela a été un pas d'avance dans notre apprentissage des systèmes embarqués et systèmes d'exploitation temps réel.

1- Schéma synoptique du système :



2- Définition des Tâches et des Objets Temps Réel

Identifiez les différentes tâches nécessaires pour votre système :

- Acquisition des niveaux de pluie et de batterie
- Calcul du mode en fonction des niveaux
- Transmission du mode
- Contrôle du moteur d'essuie-glace via signal PWM

Pour gérer ces tâches, en utilisant les objets temps réel suivants :

- Software Timers pour la temporisation Tcyc=2s.
- Queues pour la communication entre les tâches.
- Sémaphores pour assurer l'exclusion mutuelle et protéger les ressources partagées.

3- Modèle de Tâches du Système :

On définit un modèle de tâches qui spécifie comment les différentes activités seront gérées par Notre système.

- tache_Acquerir_niveau_pluie : pour mesurer le niveau de pluie.

- tache_Acquerir_niveau_Battery : pour mesurer la tension de la batterie.

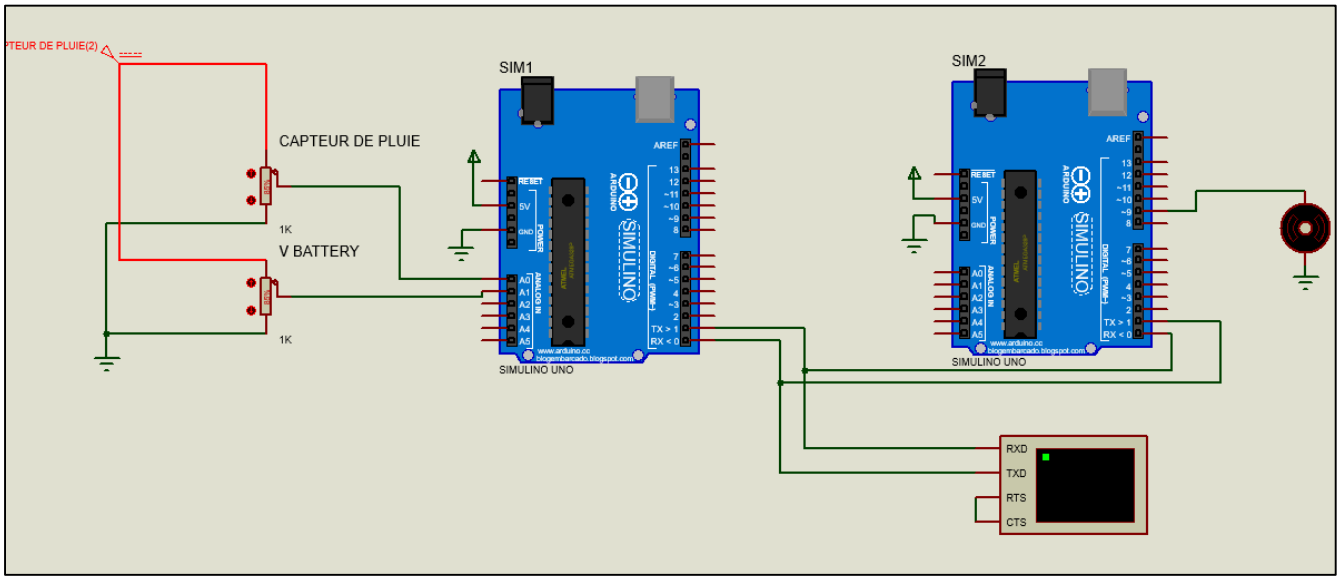
- tache_Calcule_Mode: pour déterminer le mode en fonction des niveaux mesurés.

- tache_Transmission_Mode : pour transmettre le mode calculé à l'Unité 2.

- tache_réception_Mode : pour lire le mode calculé dans le port série de la part de l'unité 2.

- Task_setMotorSpeed : pour contrôler le moteur d'essuie-glace en fonction du mode.

4- Schéma électronique :



5- Les programmes des deux unités puis tester le système :

5.1. Programme de l'unité 1

Pour mettre en avant les lignes cruciales du code C qui gère la première unité , nous pouvons nous concentrer sur les aspects clés liés à la synchronisation des tâches, à l'accès à l'ADC, à la gestion des files d'attente et à la configuration de FreeRTOS. Le code entier en annexe.

Utilisation des Sémaphores pour l'Accès aux Ressources

Lignes en Évidence :

```
SemaphoreHandle_t mutexADC;
```

L'implémentation utilise des sémaphores FreeRTOS pour gérer l'accès aux ressources critiques au sein du système. La sémaphore `mutexADC` a été déclaré pour contrôler l'accès au convertisseur analogique-numérique (ADC). Cette sémaphore est essentielle pour assurer l'exclusion mutuelle et éviter la corruption des données ou les conditions de concurrence lorsque plusieurs tâches tentent d'accéder simultanément à des ressources partagées.

Implémentation des Tâches pour l'Acquisition des Données des Capteurs

Lignes en Évidence :

```
void tache_Acquerir_niveau_pluie(void *pvParameters);

void tache_Acquerir_niveau_Battery(void *pvParameters);
```

L'application utilise plusieurs tâches chargées d'acquérir les données des capteurs à des intervalles prédéfinis. Deux tâches distinctes, `tache_Acquerir_niveau_pluie` et `tache_Acquerir_niveau_Battery`, sont mises en œuvre pour lire les valeurs analogiques du capteur de pluie et du capteur de batterie, respectivement. Chaque tâche utilise un mutex (`mutexADC`) pour garantir un accès exclusif à l'ADC lors des lectures de capteur, évitant ainsi les conflits potentiels lors de l'accès à des ressources matérielles partagées.

Gestion des Files d'Attente pour la Communication entre Tâches

Lignes en Évidence :

```
QueueHandle_t queueRainTransmission;

QueueHandle_t queueBatteryTransmission;

QueueHandle_t queueModeTransmission;
```

La communication entre les tâches est facilitée par les files d'attente FreeRTOS (`queueRainTransmission`,`queueBatteryTransmission`,`queueModeTransmission`). Ces files d'attente permettent aux tâches d'échanger des données de manière efficace et de synchroniser leurs opérations.

Coordination des Tâches et Calcul du Mode de Contrôle

Lignes en Évidence :

```
void tache_Calcule_Mode(void *pvParameters);

void tache_Transmission_Mode(void *pvParameters);
```

La fonctionnalité principale du système est orchestrée par des tâches chargées du calcul et de la transmission du mode de contrôle. La tâche `tache_Calcule_Mode` calcule le mode de contrôle en fonction des données de capteur reçues (niveau de pluie, niveau de batterie) et des valeurs seuils prédéfinies (`Rain_low`, `Rain_medium`, `Rain_high`, `Battery_low`, `Battery_high`). Cette tâche utilise des files d'attente et des mutex pour synchroniser l'accès aux données et transmettre le mode calculé à la tâche `tache_Transmission_Mode`. La tâche `tache_Transmission_Mode` gère la transmission périodique du mode de contrôle calculé via le port série (`Serial.println(mode)`).

5.2. Programme de l'unité 2

Pour mettre en évidence les lignes cruciales du deuxième code et fournir des paragraphes explicatifs, nous allons nous concentrer sur l'initialisation du moteur d'essuie-glace, le calcul du rapport cyclique PWM en fonction du mode reçu via UART, et l'application du rapport cyclique au moteur.

Initialisation du Contrôle du Moteur d'Essuie-glace

Lignes en Évidence :

```
void setupMotorControl();

void setup() {
  setupMotorControl(); // Initialise le contrôle du moteur d'essuie-glace
}

void setupMotorControl() {
  pinMode(pinMotorControlCW, OUTPUT);
  analogWrite(pinMotorControlCW, FPWM);
}
```

La fonction `setupMotorControl()` est chargée d'initialiser le contrôle du moteur d'essuie-glace. Elle configure la broche de contrôle du moteur (`pinMotorControlCW`) en tant que sortie et initialise la fréquence PWM du moteur en utilisant `analogWrite()`. La valeur de la fréquence PWM est définie par `FPWM` (1000 Hz dans ce cas). Cette initialisation garantit que le moteur est prêt à recevoir des commandes de vitesse sous forme de rapport cyclique PWM.

Calcul du Rapport Cyclique PWM en Fonction du Mode Reçu

Lignes en Évidence :

```
float dutyCycle = 0.0;
switch (mode) {
  case 0:
    dutyCycle = DUTY_CYCLE_MODE_0;
    break;
  case 1:
    dutyCycle = DUTY_CYCLE_MODE_1;
    break;
  case 2:
    dutyCycle = DUTY_CYCLE_MODE_2;
    break;
  case 3:
    dutyCycle = DUTY_CYCLE_MODE_3;
    break;
  default:
    // Mode par défaut si une valeur non valide est reçue
    dutyCycle = DUTY_CYCLE_MODE_0;
    break;
}
```

Lors de la réception d'un mode via UART, le système calcule le rapport cyclique PWM correspondant en fonction de la valeur du mode reçu. Cette logique est implémentée à l'aide d'une instruction `switch` qui associe chaque mode (0, 1, 2, 3) à un rapport cyclique spécifique (`DUTY_CYCLE_MODE_X`). Si une valeur de mode inattendue est reçue, le système définit un rapport cyclique par défaut (`DUTY_CYCLE_MODE_0`). Le rapport cyclique résultant est stocké dans la variable `dutyCycle` pour être appliqué au moteur.

Application du Rapport Cyclique au Moteur d'Essuie-glace

Lignes en Évidence :

setMotorSpeed(dutyCycle, pinMotorControlCW);

Une fois le rapport cyclique PWM calculé, la fonction `setMotorSpeed()` est appelée pour régler la vitesse du moteur d'essuie-glace en fonction du rapport cyclique calculé (`dutyCycle`). Cette fonction convertit le rapport cyclique en une valeur PWM (de 0 à 255) compatible avec la fonction `analogWrite()`, qui ajuste ensuite la vitesse du moteur en modulant la largeur des impulsions PWM envoyées à la broche de contrôle (`pinMotorControlCW`). Ainsi, le moteur d'essuie-glace est actionné à une vitesse correspondant au mode spécifié via UART.

Conclusion

Ce rapport décrit un système de commande d'essuie-glace automatique pour voiture. Le système ajuste la vitesse du moteur en fonction du niveau de pluie détecté par un capteur et de la tension de la batterie.

Fonctionnement

- Unité d'acquisition et de contrôle (Arduino) :
  - Mesure le niveau de pluie et la tension de la batterie (via CAN).
  - Calcule le mode de fonctionnement du moteur en fonction des mesures et d'un tableau de décision.
  - Transmet le mode calculé à l'unité de commande du moteur (via UART).
- Unité de commande du moteur d'essuie-glace (Arduino) :
  - Reçoit le mode de fonctionnement.
  - Génère un signal PWM pour contrôler la vitesse du moteur en fonction du mode reçu (duty cycle variant de 0 à 1).

Programmation

Le système est programmé avec le noyau temps réel Arduino FreeRTOS. Des tâches distinctes gèrent l'acquisition des mesures, le calcul du mode, la transmission et la commande du moteur via PWM. Des objets temps réel (timers, files d'attente et sémaphores) assurent la synchronisation et la protection des ressources partagées entre les tâches.

Annexe

Programme de l'unité 1

```
/*
 *
 * Auteur : Hamza Jebbari & Nassira Kaddouri & Ayman Habbaz
 * Date de création : 11/05/2024
 * Description : Ce programme utilise FreeRTOS sur une carte Arduino pour
surveiller les niveaux de pluie
 *               et de batterie, calculer le mode de contrôle en fonction
des seuils définis, et transmettre
 *               le mode de contrôle via le port série.
 */

#include <Arduino_FreeRTOS.h>
#include <semphr.h>
#include <queue.h>


/********************************************************** Definition
des broches***************************************************/
const int pin_Rain = A0;
const int pin_Battery = A1;


/*********************************************Definition  des  seuils
les niveaux de pluie/batterie*********************************/
const int Rain_low = 50;
const int Rain_medium = 200;
const int Rain_high = 600;
const float Battery_low = 11.6;
const float Battery_high = 12.0;


/*********************************************************Déclaratio
n des tâches*********************************************/
void tache_Acquerir_niveau_pluie(void *pvParameters);
void tache_Acquerir_niveau_Battery(void *pvParameters);
void tache_Calcule_Mode(void *pvParameters);
void tache_Transmission_Mode(void *pvParameters);



/*****************************************************  Déclaration
de sémaphore*****************************************************/

SemaphoreHandle_t mutexADC; // Mutex pour l'accès à l'ADC

/*****************************************Déclaration  des  queues
pour la transmission des données*********************************/
QueueHandle_t queueModeTransmission;
QueueHandle_t queueRainTransmission;
QueueHandle_t queueBatteryTransmission;
```

```
/***********************************************************************
*****************************************************************/
void setup() {

  /**************************************************Initialise le port
série à 9600 bauds********************************************/
  Serial.begin(9600);


  /*******************************************************Initialisati
on des tâches****************************************************/
  xTaskCreate(tache_Acquerir_niveau_pluie, "AcquireRain", 100, NULL, 1,
NULL);
  xTaskCreate(tache_Acquerir_niveau_Battery,   "AcquireBattery",   100,
NULL, 1, NULL);
  xTaskCreate(tache_Calcule_Mode, "CalculateMode", 100, NULL, 2, NULL);
  xTaskCreate(tache_Transmission_Mode,  "TransmitMode",  100,  NULL,  3,
NULL);



  /*****************************************Initialisation  des  queues
pour la transmission des données***********************************/
  queueRainTransmission = xQueueCreate(1, sizeof(float));
  queueBatteryTransmission = xQueueCreate(1, sizeof(float));


  /*******************************************Initialisation de la queue
pour la transmission du mode************************************/
  queueModeTransmission = xQueueCreate(1, sizeof(int));



  /********************************************Initialisation        du
sémaphore                        pour                       l'exclusion
mutuelle************************************/


  mutexADC  =  xSemaphoreCreateMutex();//  Initialisation  du  mutex  pour
l'ADC
}

void loop() {}

/*************************************************      Tâche      pour
l'acquisition               du               niveau              de
pluie*********************************************/
void tache_Acquerir_niveau_pluie(void *pvParameters) {
    (void) pvParameters;

    while (1) {

        // Acquérir le mutex pour l'ADC
        if (xSemaphoreTake(mutexADC, portMAX_DELAY) == pdTRUE) {
         int rainLevel = analogRead(pin_Rain);
```

```
          xSemaphoreGive(mutexADC); // Libérer le mutex pour l'ADC

          float rainVoltage = rainLevel * (5.0 / 1023.0);
          int rainValue = (rainVoltage / 5) * 1000;
 ////////////////////////////////////////////////////////////////////////
////////////////// pour tester les valeurs
   Serial.print("Vpluie : ");
   Serial.println( rainValue);
   ////////////////////////////////////////////////////////////////////////
////////////////////



        xQueueSend(queueRainTransmission, &rainValue, portMAX_DELAY);


        }
        vTaskDelay(pdMS_TO_TICKS(2000));
    }
}



/*****************************************Tâche pour l'acquisition du
niveau de batterie*****************************************/
void tache_Acquerir_niveau_Battery(void *pvParameters) {
    (void) pvParameters;

    while (1) {

      // Acquérir le mutex pour l'ADC
       if (xSemaphoreTake(mutexADC, portMAX_DELAY) == pdTRUE) {

         float batteryLevel = analogRead(pin_Battery);
         xSemaphoreGive(mutexADC); // Libérer le mutex pour l'ADC

         float batteryVoltage = batteryLevel * (5.0 / 1023.0);
         batteryLevel = (batteryVoltage / 5) * 12.8;

////////////////////////////////////////////////////////////////////////
/////////////// just pour tester
   Serial.print("Vbattery: ");
   Serial.println( batteryLevel);
  ////////////////////////////////////////////////////////////////////////
///////////



       xQueueSend(queueBatteryTransmission,&batteryLevel,portMAX_DELA);


       }
        vTaskDelay(pdMS_TO_TICKS(2000));
    }
}
```

```
/**************************************************Tâche pour le calcul
du mode de contrôle*************************************************/
void tache_Calcule_Mode(void *pvParameters) {
    (void) pvParameters;

    int rainValue;
    float batteryLevel;

    while (1) {


        xQueueReceive(queueRainTransmission, &rainValue, portMAX_DELAY);
        xQueueReceive(queueBatteryTransmission,           &batteryLevel,
portMAX_DELAY);


        int mode = 0; // Initialiser mode à 0

        // Calculer le mode de contrôle en fonction des seuils
        if (rainValue < Rain_low) {
            mode = 0;
        } else if (rainValue >= Rain_low && rainValue < Rain_medium &&
batteryLevel >= Battery_low) {
            mode = 1;
        } else if (rainValue >= Rain_medium && rainValue < Rain_high &&
batteryLevel >= Battery_low) {
            mode = 2;
        } else if (rainValue >= Rain_high && batteryLevel >= Battery_high)
{
            mode = 3;
        }

        // Envoyer le mode calculé à la file d'attente de transmission du
mode
        xQueueSend(queueModeTransmission, &mode, portMAX_DELAY);

    }
}

/**************************************************Tâche      pour      la
transmission          périodique          du          mode          de
contrôle***********************************/
void tache_Transmission_Mode(void *pvParameters) {
    (void) pvParameters;

    while (1) {
        // Attendre le mode de contrôle calculé
        int mode;


        xQueueReceive(queueModeTransmission, &mode, portMAX_DELAY);


        // Transmettre le mode de contrôle
    Serial.print("Mode de controle : ");
```

```
    Serial.println(mode);

        vTaskDelay(pdMS_TO_TICKS(2000));
    }
}
```

Programme de l'unité 2

```
#include <Arduino_FreeRTOS.h>
#include <task.h>

/*********************************************************Fréquence
PWM*************************************************************/
const int FPWM = 1000; // Hz



/*********************************Broches de sortie PWM pour le moteur
d'essuie-glace*************************************************/
const int pinMotorControlCW = 9;  // Sens horaire




/*********************************Définition  des  seuils  pour  les
différents modes************************************************/
const float DUTY_CYCLE_MODE_0 = 0.0;
const float DUTY_CYCLE_MODE_1 = 0.25;
const float DUTY_CYCLE_MODE_2 = 0.75;
const float DUTY_CYCLE_MODE_3 = 1.0;



/*****************************************Déclaration              des
fonctions*******************************************************/
void setupMotorControl();
void setMotorSpeed(float dutyCycle, int pin);

void setup() {
  Serial.begin(9600); // Initialise la communication série à 9600 bauds
  setupMotorControl(); // Initialise le contrôle du moteur d'essuie-glace
}

void loop() {
  if (Serial.available() > 0) {



/*************************************************Lire   le   mode
reçu via UART***************************************/
```

```
    int mode = Serial.parseInt();


/*************************************Calculer le rapport cyclique
PWM en fonction du mode*********************************/
    float dutyCycle = 0.0;
    switch (mode) {
      case 0:
        dutyCycle = DUTY_CYCLE_MODE_0;
        break;
      case 1:
        dutyCycle = DUTY_CYCLE_MODE_1;
        break;
      case 2:
        dutyCycle = DUTY_CYCLE_MODE_2;
        break;
      case 3:
        dutyCycle = DUTY_CYCLE_MODE_3;
        break;
      default:
        // Mode par défaut si une valeur non valide est reçue
        dutyCycle = DUTY_CYCLE_MODE_0;
        break;
    }


/************************************************Appliquer le rapport
cyclique PWM au moteur**********************************/

    setMotorSpeed(dutyCycle, pinMotorControlCW);

  }
}


/*****************************************Initialise le contrôle du
moteur d'essuie-glace*********************************/
void setupMotorControl() {
  pinMode(pinMotorControlCW, OUTPUT);
  analogWrite(pinMotorControlCW, FPWM);

}



/****************************** Régle la vitesse du moteur d'essuie-glace
en fonction du rapport cyclique ********************/
void setMotorSpeed(float dutyCycle, int pin) {
  int pwmValue = dutyCycle * 255; // Conversion du rapport cyclique en
valeur PWM (0-255)
  analogWrite(pin, pwmValue);
}
```