GESTION DE DÉPLACEMENT ET DE COLLABORATION DE ROBOTS

Florian Boisnon, Blaise Pruvost et Vincent Maurice
Responsable du projet Z. Mammeri

Février 2021

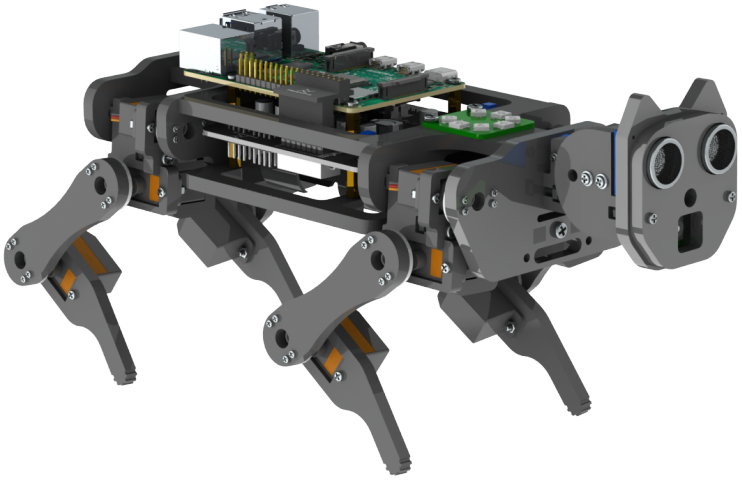

Figure 1: FreeNove RobotDog

Contents

1 Introduction **3**

2 Le Robot **3**
2.1 Présentation 3
2.2 Son Fonctionnement 3
2.3 La calibration 5
2.4 Les chaussettes 7

3 La gestion du projet **8**

4 Suivi du robot **9**

5 Radio-commandé **13**
5.1 ESP32 13
5.2 Joystick 14
5.3 Bluetooth Low Energy 16
5.4 Le fonctionnement 18

6 Scénario **20**
6.1 Introduction 20
6.2 La gestion d'obstacle 20
6.3 Les déplacements 21
6.4 La reconnaissance de l'objectif 21

7 Les Compétences acquises **23**

8 Les problèmes **24**

9 Github **24**

10 Conclusion **24**

1 Introduction

Notre projet de fin d'études consiste à partir de plusieurs robots de pouvoir gérer une collaboration entre eux, de récupérer des informations sur l'état et un bonus de le contrôler depuis un joystick. Pour mener à bien ce projet, nous avons eu 3 robots chiens du fabricant FreeNove.

2 Le Robot

2.1 Présentation

Durant ce projet nous utilisons un seul et unique modèle, un robot chien du fabricant FreeNove. Le robot est composé de :

- Raspberry pi 3.
- 12 moteurs à pas (ES08MAII) , trois par jambe et 1 moteur à pas (TS90A) pour la tête.
- Un capteur à ultrason HC-SR04.
- Un accéléromètre/gyroscope MPU6050.
- Un bandeau led.
- Une Pi caméra.
- 2 batteries 3,7v 18650.

2.2 Son Fonctionnement

Le robot est contrôlé par une carte de contrôle produit par FreeNove. Elle possède un accéléromètre/gyroscope, l'ensemble des branchements moteurs pas à pas, ici des pins. Ces moteurs sont contrôlés par un microcontrôleur PCA9685 I2C avec 16 ports de 12-bit PWM. Un bouton CTRL pour allumer le robot et un bouton LOAD pour appareiller la carte de contrôle et la RPY. 2 batteries 3,7V sont admises sur le dos de la carte de contrôle. Des régulateurs de tension pour garder une tension de 5V.

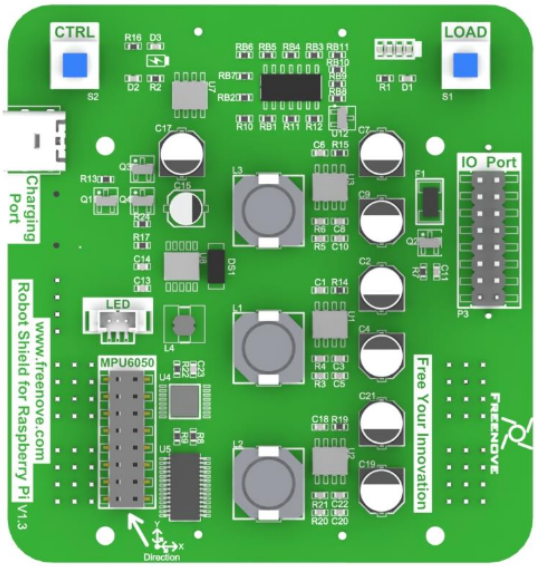

Figure 2: Shield Top

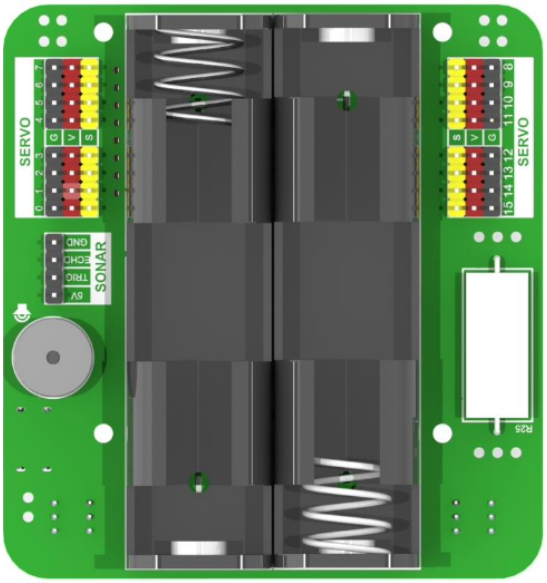

Figure 3: Shield Bottom

2.3 La calibration

La calibration est une étape importante. Une bonne calibration va nous permettre de donner un angle précis au moteur lors de son utilisation. C'est à dire une fluidité dans ses déplacements. Première étape il faut installer les moteurs avec un angle de 90.

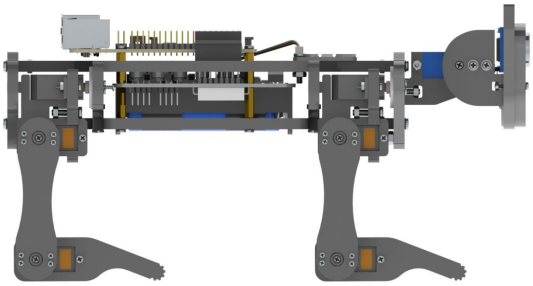

Figure 4: Calibration moteur à 90°

Puis il faut lancer l'application python (figure 6) proposé par FreeNove qui nous permet de faire bouger le robot et le calibrer. Cette application est réliée à une autre application python jouant le rôle de serveur (figure 5) qui contrôle les moteurs, la caméra, le panneau de LED. Ainsi dans cette configuration il est très facile de paramétrer le robot depuis la raspberry ou depuis un autre ordinateur grâce à l'adresse IP.

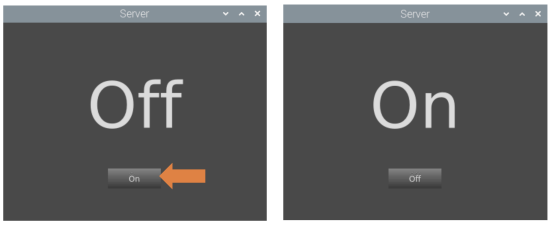

Figure 5: Le serveur

Sur cette application nous avons la possibilité de faire avancer, reculer, tourner à droite et à gauche le robot. D'allumer ou éteindre la caméra, bref l'ensemble des actions de base du robot. Ici ce qui nous intéresse c'est la case calibration (figure 7). Sur cette page nous pouvons créer un décalage sur l'ensemble des moteurs. La configuration sera enregistrée dans un fichier point.txt.

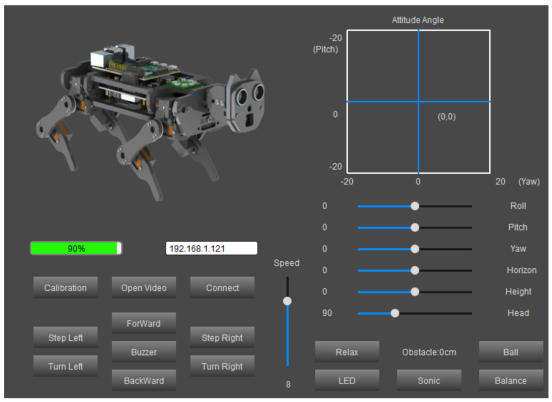

Figure 6: L'application

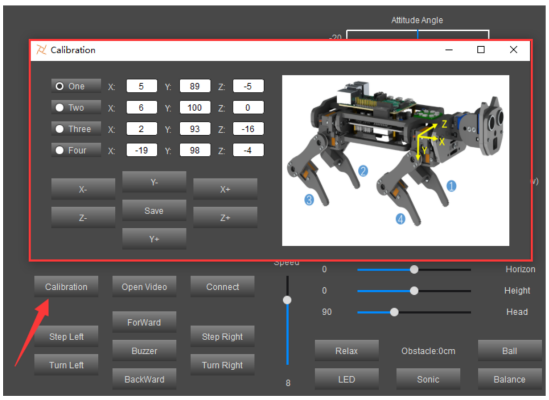

Figure 7: La calibration

Pour une calibration optimale il faut poser le robot sur la feuille de calibration proposer par FreeNove (figure 8). Il faut changer la valeur de chaque moteur afin que chaque pied touche son encoche (figure 9).

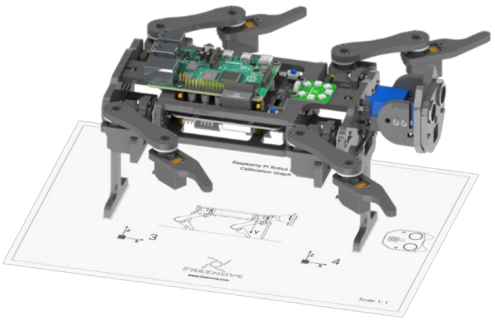

Figure 8: Début de calibration

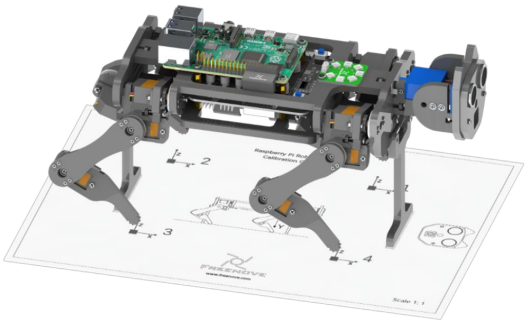

Figure 9: Fin de calibration

2.4 Les chaussettes

Un des soucis avec notre robot est que les pieds ont tendance à glisser sur le sol ce qui pose des problèmes lors des mouvements. Pour ce fait, nous avons mis des gaines rétractables afin d'avoir une meilleure accroche.

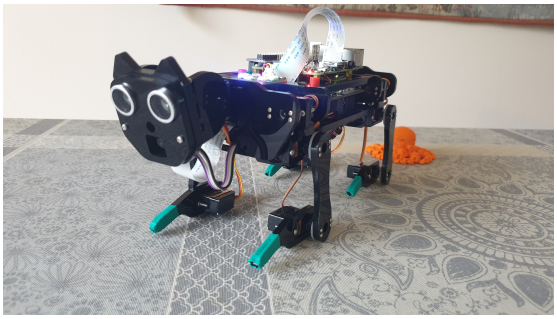

Figure 10: Les chaussettes

3 La gestion du projet

Étant donnée que nous sommes dans une période particulière, la gestion du projet c'est faite de manière différente. En effet M. Mammeri notre responsable du projet nous a prêté à chacun un robot identique afin qu'on puisse travailler chacun de notre côté et d'avoir une même base afin d'échanger entre nous. Nous avons récupéré les robots courant mois de décembre pendant les partiels de fin d'année.

Le premier objectif commun est de monter ou de vérifier l'installation, le fonctionnement et la calibration du robot. Ceci est une tache qui a pris du temps entre des batteries qui ne fonctionnaient pas correctement, des moteurs défaillants sur un robot.

Le deuxième objectif est de trouver des choses apportées à notre robot répondant à notre parcours Master et de répondre aux besoins du client c'est à dire au responsable M. Mammeri. Un ensemble d'idées est venu, avec comme particularité commune une collaboration et un remplissage de tâche automatisé entre les robots. Cependant il fallait faire attention à ne pas trop tomber dans de l'intelligence artificielle étant donnée que la majorité des données reçues étaient celle de la pi caméra et que le temps nécessaire à la réalisation d'un apprentissage optimisé pour un raspberry pi en temps réel risquait d'être trop court. Ainsi il était primordial d'apporter une valeur supplémentaire grâce aux compétences acquises de notre Master.

Nous avons réalisé différents modules, le premier est le suivi en temps réel de l'état du robot, c'est à dire l'autonomie énergétique, l'angle des moteurs, la position dans l'espace avec l'accéléromètre et le gyroscope. Le deuxième est le contrôle du robot à partir d'un joystick branché sur un ESP32. Cet ESP32 est relié avec du Bluetooth Low Energy. Et le dernier est un jeu entre les robots à celui qui trouvera la balle rouge en premier avec une communication afin que le premier robot trouvant la balle l'informe aux autres.

Nous avons divisé le groupe en deux, un groupe composé de Florian et Blaise s'occupe du scénario de recherche avec un objectif représenté par la balle rouge, avec de la coopération entre les robots. Vincent quant à lui gère le suivi en temps réel, le joystick et la détection d'obstacle et de vide.

Les vendredis nous avions un rendez-vous hebdomadaire avec notre responsable pour discuter des avancés et des objectifs du projet, petit à petit. Certaines idées n'ont pas été retenues par rapport à leur manque de pertinence, ou leur faisabilité. On peut citer, entre autres l'idée de réaliser une cartographie de l'espace en 3D, qui s'est avérée trop difficile au vu de la précision, de la faible capacité des calculs des robots et du temps qui restait jusqu'au rendu du projet, c'est à dire environ une semaine.

En termes de méthodologie pure et dure, pour poser une terminologie plus précise, on était donc sur des cycles de développement très courts, pour s'adapter aux besoins exprimés par le MOA (Notre professeur responsable). On est donc proche d'une méthode Agile qui respecterait le temps de développement inhérent à chaque itération, sans pour autant fixer des objectifs précis temporellement.

Une sorte de gros développement au forfait très flexible, en somme.

4 Suivi du robot

Dans notre projet il peut vite être important de suivre en temps réel notre robot ou du moins garder une trace de ses mouvements. En effet sur ce genre de robot nous pouvons avoir des taches à lui faire sans qu'on soit à côté lui. Pour cela on utilise la base de données InfluxDB qui a pour avantage de ranger les données par étiquette (tag). Ainsi un robot peut être représenté par une étiquette et que celle-ci possède l'ensemble des étiquettes moteurs, batterie, etc...

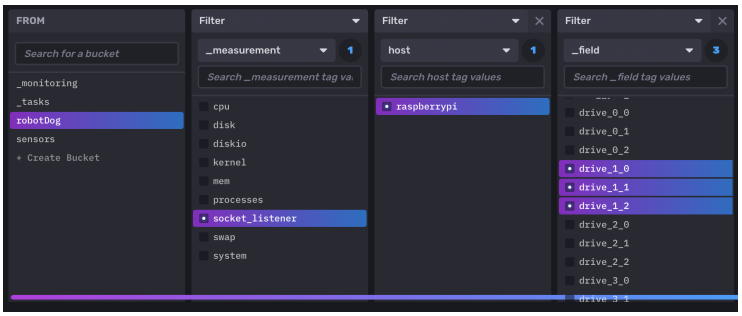

Figure 11: InfluxDB étiquettes

Pour récupérer l'angle des moteurs nous avons eu besoin de faire un code inverse de setServoAngle(), c'est à dire de récupérer la valeur PWM de chaque moteur à partir du microcontrôleur PCA9685 (carte de contrôle) et de faire les calculs inverses afin de retrouver l'angle appliqué aux moteurs. Cette méthode permet de récupérer les valeurs indépendamment au programme qui exécute une action sur le robot. Pour l'accéléromètre/gyroscope, la batterie et le capteur d'ultrason, nous utilisons les fonctions proposées par FreeNove.

```
# get PWM
    def getPWM(self, num):
        return
        ↪  round(float(self.I2Cbus.read_byte_data(self.slaveAddress,
        ↪  LED0_OFF_L + 4 * num)),3)

    # from PWM to angle
    def inverse(self, n):
        n += 256.0
        return round(((n * 20000.0 / 4096.0) - 850.0) / 10.0, 0)

    # map value
    def map(self,value,fromLow,fromHigh,toLow,toHigh):
        return (toHigh-toLow)*(value-fromLow) /
        ↪  (fromHigh-fromLow) + toLow

    # borne the result
    def restriction(self, n, min, max):
        if n < min:
            return min
        elif n > max:
            return max
        else:
            return n

    # For each leg
    for i in range(2):
        self.servo_angle[i][0] =
        ↪  self.restriction(self.inverse(self.getPWM(4+i*3)) -
        ↪  self.calibration_angle[i][0], 0, 180)
        self.servo_angle[i][1] =
        ↪  self.restriction(self.inverse(self.getPWM(3+i*3)) +
        ↪  self.calibration_angle[i][1], 0, 180)
        self.servo_angle[i][2] =
        ↪  self.restriction(self.inverse(self.getPWM(2+i*3)) -
        ↪  self.calibration_angle[i][2], 0, 180)
```

```
        self.servo_angle[i+2][0] =
        ↪  self.restriction(self.inverse(self.getPWM(8+i*3)) -
        ↪  self.calibration_angle[i+2][0], 0, 180)
        self.servo_angle[i+2][1] =
        ↪  self.restriction(self.inverse(self.getPWM(9+i*3)) -
        ↪  self.calibration_angle[i+2][1], 0, 180)
        self.servo_angle[i+2][2] =
        ↪  self.restriction(self.inverse(self.getPWM(10+i*3)) +
        ↪  self.calibration_angle[i+2][2], 0, 180)
```

Pour envoyer les données à la base de données, chaque robot envoie l'ensemble de ses données sous forme JSON à une interface d'influxDB qui s'appelle telegraf. Influxdb ne traite pas les JSON directement. L'interface a pour rôle de récupérer les données de les traités afin de pouvoir les ranger dans la base de données. Nôtre code python envoie une requête UDP à l'interface telegraf. Ce qui nous permet soit d'installer celle-ci sur le raspberry pi du robot ou bien sur un serveur à distance.

```
# envoie le message JSON vers l'interface influxDB
try:
    self.socket.sendto(json_msg.encode('utf8'), (self.host, self.port))
    print ("Send")
except (socket.error, RuntimeError):
    print ("Error send")
```

Un exemple de message JSON envoyé avec les moteurs à 90 degré (mode relaxe).


```
{
"IMU": {
    "accel": {
        "x": 0.27966669592285165,
        "y": 0.33775002819824224,
        "z": -0.2971981326904345
    },
    "gyro": {
        "x": -3.4232824427480906,
        "y": -0.586641221374046,
        "z": 0.7890839694656487
    }
},
"drive": {
    "0": {
        "0": 90.0,
        "1": 90.0,
        "2": 90.0
    },
```

```
        "1": {
            "0": 90.0,
            "1": 90.0,
            "2": 90.0
        },
        "2": {
            "0": 90.0,
            "1": 90.0,
            "2": 90.0
        },
        "3": {
            "0": 90.0,
            "1": 90.0,
            "2": 90.0
        }
    },
    "sonic": {
        "angle": 40.0,
        "distance": 47
    },
    "voltage": 7.71
}
```


Lorsque les données sont enregistrées dans la base, il faut pouvoir les traiter ou de faire un aperçu. InfluxDB dépose directement la possibilité de présenter les données sous la forme d'un panel. C'est à dire de créer différentes cellules pour chaque étiquette.

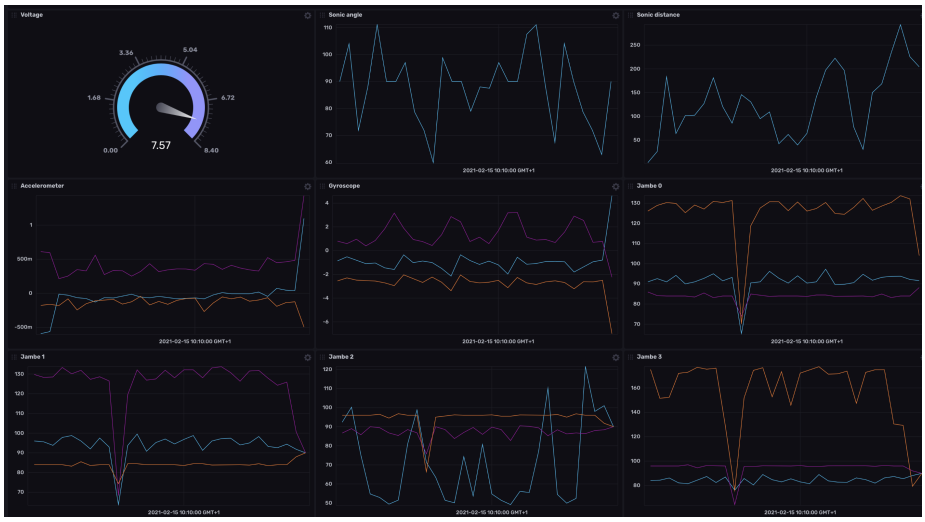

Figure 12: InfluxDB board

5 Radio-commandé

L'objectif ici est de commander le robot depuis un joystick ce qui peut être pratique surtout quand on veut reprendre la main sans interrompre le code. Pour réaliser cela, nous avons utilisé un joystick arduino qui sera branché sur un ESP32. Pour ce projet, nous avons utilisé le protocole Bluetooth Low Energy qui a beaucoup d'avantage dans ce type de mise en oeuvre.

5.1 ESP32

ESP32 est une petite carte électronique produit par Espressif. Elle compte de nombreux avantages pour de l'IoT comme l'intégration du WiFi et du Bluetooth sur une fréquence de 2,4 GHz. Un nombre de GPIO intéressant, des BUS standardisés, un convertisseur ADC de 12 bits, un mode " sleep ". Ainsi l'ESP32 est idéal pour la manette de plus elle est facile d'utilisation et possède de nombreuses librairies dont une pour le Bluetooth Low Energy.

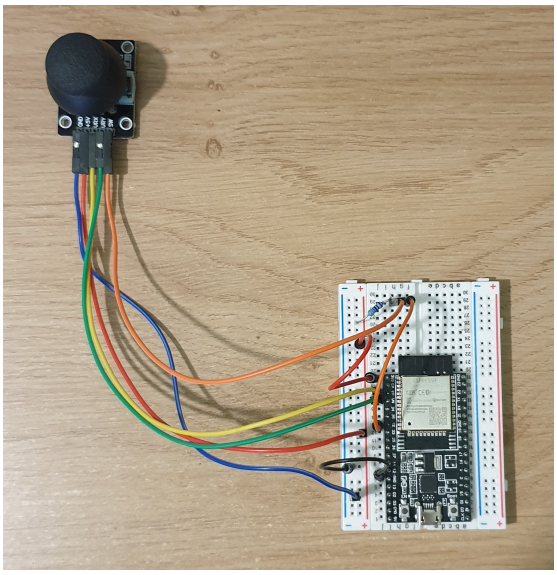

Figure 13: ESP32 et Joystick

5.2 Joystick

Un joystick est branché sur l'ESP32 en 3V3. C'est un capteur de positionnement, fonctionnant sur le principe du pont diviseur, c'est à dire que selon le positionnement du joystick on a une tension qui diffère. C'est tout simplement des potentiomètres. Plus un bouton pour jouer le rôle de Z quand on pousse le joystick. Si dessous (figure 14), le fonctionnement du joystick est en lien avec les mouvements de base du robot proposé par FreeNove.

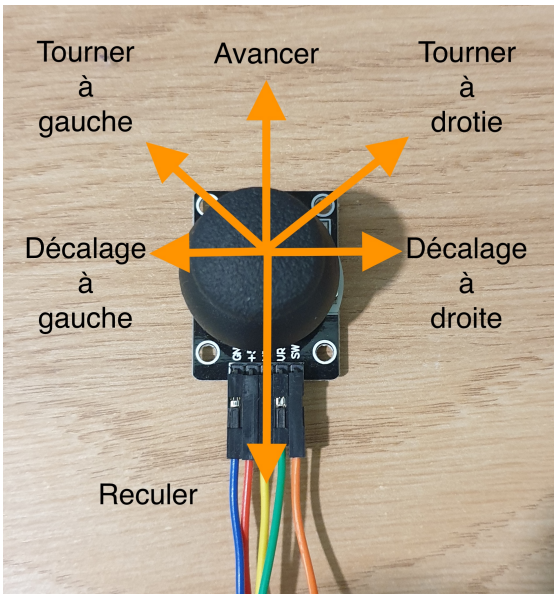

Figure 14: Contrôle

De plus la manette propose un autre jeu de sélection. Lorsqu'on clique sur le joystick on a la possibilité de faire animer le robot.

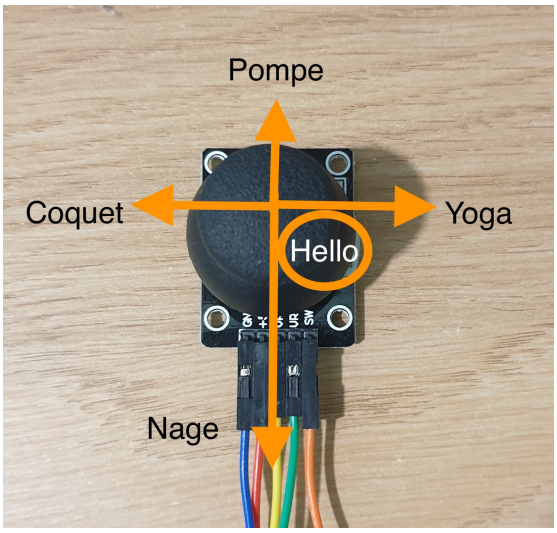

Figure 15: Animation

Ci-dessous une capture d'écran de l'animation " hello ", c'est animation

fonctionne lorsqu'on laisse de joystick au centre et on clique dessus.

Figure 16: Animation hello

5.3 Bluetooth Low Energy

Le Bluetooth Low Energy (BLE) comme le nom l'iindique, le BLE est un protocole bluetooth consommant peut de ressource ce qui le rend intéressant dans le monde de l'IoT. En effet BLE consomme moitié moins que le bluetooth ce qui intéressant quand l'appareil est sur batterie.

Le BLE possède plusieurs modes de connections.

- Le mode " Diffuseur ", l'appareil est en mode serveur diffusant des données.
- Le mode " Observeur ", l'appareil est un client.
- Le mode " Central ", l'appareil en mode avertissement, soit en mode connecté, en règle général les smartphones et tablettes sont dans ce mode.
- Le mode " Périphérique ", envoie des messages périodiques à l'appareil central.

Ici la raspberry pi du robot chien joue le mode " Observeur " et la manette (ESP32) quand à lui le mode " Diffuseur ".

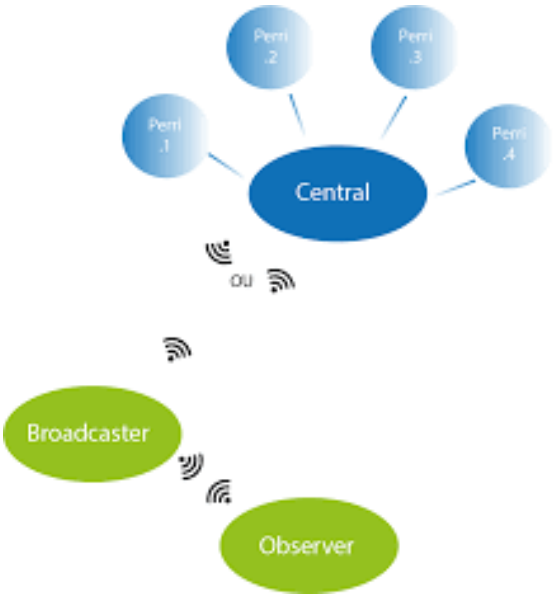

Figure 17: BLE mode

Un autre avantage du Bluetooth Low Energy est la structure des données qui est rangée par service possédant des caractéristiques qui lui même possèdent des descriptions.

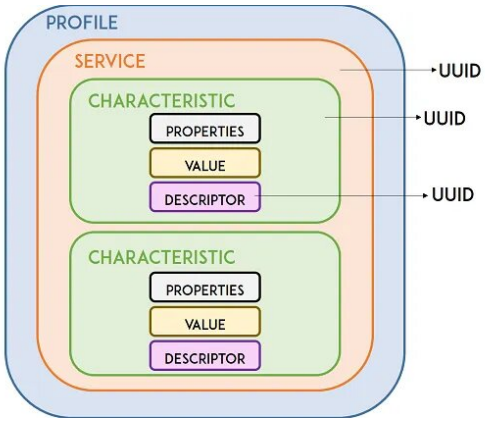

Figure 18: BLE profile

Chaque partie possède un identifiant qui s'appelle UUID. Il en existe des génériques (GATT) ce qui permet à une application de reconnaître un service ou de signer des données. On peut retrouver les différents identifiant sur le site officiel Bluetooth ou directement sur ce pdf.

Pour ce projet nous avons mis des identifiants arbitraires. Pour X l'UUID 0x2AAE qui correspond à la latitude. Pour Y, l'UUID 0x2AAF qui correspond à la longitude et pour Z l'UUID 0x2AB3 qui correspond à l'altitude. Nous n'avons pas trouvé de UUID plus précis correspondant à une manette. Nous pouvons créer notre propre protocole UUID mais c'était pour que le protocole GATT reconnaisse et que l'application mobile Nordic affiche quelque chose autre que " Unknown Service " pour la présentation.

5.4 Le fonctionnement

Le robot se connecte à l'EPS32 grâce à l'adresse mac BLE. Le robot récupère la valeur de x (entre 0 et 255), y (entre 0 et 255) et z (0 ou 1) puis effectue la tache. Une fois la tache effectuée le robot récupère de nouveau les valeurs x, y et z. Ainsi de suite. Ci-dessous le code python du robot chien pour récupérer les valeurs.

```
# UUID
self.service_uuid = btle.UUID("2A9F")
self.char_uuid_x = btle.UUID("2AAE")
self.char_uuid_y = btle.UUID("2AAF")
self.char_uuid_z = btle.UUID("2AB3")

# Pour se connecter à la manette
def connect(self):
    print ("Connecting...")
    self.dev.connect("B8:F0:09:CC:88:92")
    print ("Done")

#  Récupère les valeurs puis exécute
def run(self):
    for i in self.dev.getServices():
        if i.uuid == self.service_uuid:
            result = i.getCharacteristics()
            break

    for i in result:
        if i.uuid == self.char_uuid_x:
            x = int(i.read().hex(), 16) # convert hex to int
        elif i.uuid == self.char_uuid_y:
            y = int(i.read().hex(), 16)
        elif i.uuid == self.char_uuid_z:
            z = int(i.read().hex(), 16)
```

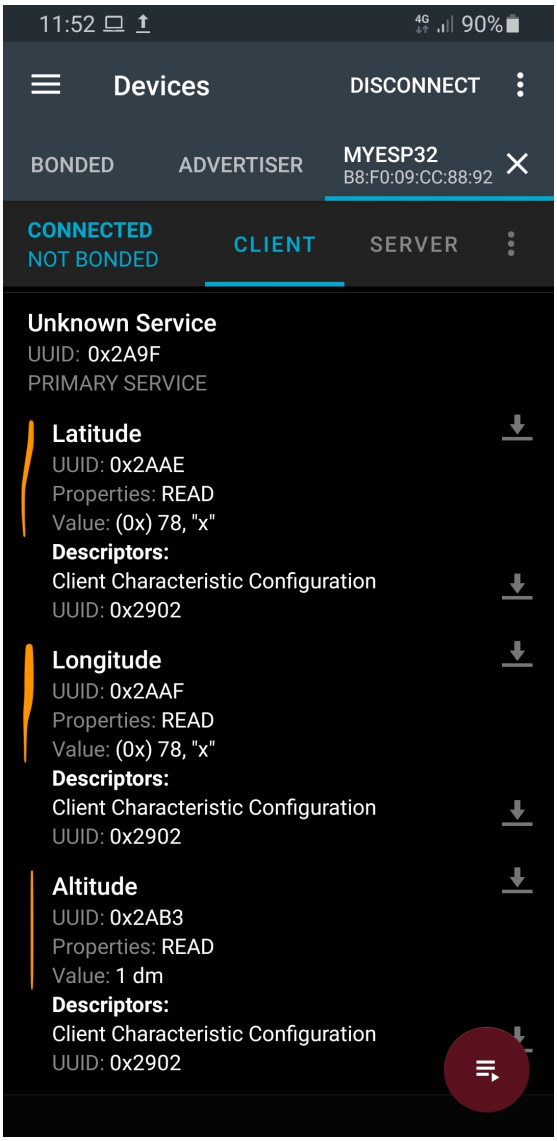

Figure 19: BLE application Nordic

```
        else:
            print ("error uuid")

    # lance les taches
    ...
```

Nous ne pouvons pas faire de temps réel avec le joystick pour deux raisons

si on coupe dans un mouvement le robot s'écraserait sur le sol et les batteries n'ont pas une capacité importante pour se permettre des changements aussi fréquents.

6 Scénario

6.1 Introduction

Le scénario objectif est le suivant : Deux robots sont présents sur un terrain pourvu d'obstacles et doivent trouver un objet en coopération (Ici nous avons choisis une balle rouge), le premier ayant trouvé , avertit l'autre et les deux se rendent à leur position d'origine.

On peut découper ce scénario en plusieurs sous scénarios incrémentaux :

- (i) Un robot se déplace aléatoirement sur le terrain en contournant les divers obstacles sur son parcours.
- (ii) On ajoute au scénario précèdent la reconnaissance de l'objectif (Ici , une balle rouge)
- (iii) On ajoute au scénario précèdent le deuxième robot sur le terrain et l'intercommunication entre lui et le premier , le premier trouvant l'objectif arrêtent le second par l'envoi de messages par wifi.
- (iv) On ajoute pour chacun des robots , le retour à la position de départ une fois l'objectif trouvé

6.2 La gestion d'obstacle

La gestion d'obstacle est primordiale car il permet au robot de se coincer quelque part, de s'abîmer ou bien aussi de tomber. Ainsi une bonne gestion des obstacles permet de garder nos robots dans de bonnes conditions.

Ci-dessous le code python pour déterminer s'il y a un obstacle :

```
# retourne True quand il y a un obstacle sinon faux
def is_in_front_of_obstacle(self):
    self.servo.setServoAngle(15,90) # mur
    time.sleep(0.5)
    if self.getDistance() < 10:
        return True
    else:
        self.servo.setServoAngle(15,45) # livre, objet plat ou vide à venir.
        time.sleep(0.5)
        if self.getDistance() < 150:
            self.servo.setServoAngle(15,90) # remet la tête droite
            return True
```

```
        else:
            self.servo.setServoAngle(15,0) # vide (ex : table)
            time.sleep(0.5)
            if self.getDistance() > 15:
                self.servo.setServoAngle(15,90)
                return True
            else:
                self.servo.setServoAngle(15,90)
                return False
```

6.3 Les déplacements

L'algorithme de base est extrêmement simple, cela est constatable avec le code python donné ci-dessous. Il est évident que le détail des fonctions appellées est disponible sur le github . La fonction doInst() prend ici l'argument "F" pour foward. La fonction findball() est détaillée dans le paragraphe suivant. La fin de l'algorithme concerne la partie de demi tour, et la lecture par symétrie des instructions reçues par le robot afin qu'il retrouve sa position d'origine.

```
while not findBall() :
    if detection.is_in_front_of_obstacle():
        #Fonction de franchissement d'obstacle
        passObstacle()
    else :
        #On va tout droit et on l'écrit dans le log
        doInstr("F")
        log.write("F")
#On fait demi tour
turnBack()
#On inverse le journal de déplacements
log.reverse()
#On execute tous les ordres du journal
while not log.isEmpty:
    doInstr(log.getLastOrder)
```

6.4 La reconnaissance de l'objectif

Ici, l'objectif à reconnaître est une balle de couleur rouge fournie avec le Kit du robot. Théoriquement le robot était livré avec un logiciel plug-and-play mais qui dans les faits s'est avéré non fonctionnel. Celui réécrit marche grâce aux primitives de la bibliothèque python opencv.

```
class CircleDetection:

    def __init__(self):
        self.camera = PiCamera()
```

```
        self.camera.start_preview()
        time.sleep(2)

    def Looking_for_the_ball(self,image):
        THRESHOLD_LOW = (0, 70, 50)
        THRESHOLD_HIGH = (5,255,255)

        output=image.copy()
        img_filter = cv2.GaussianBlur(image.copy(), (3, 3), 0)
        img_filter = cv2.cvtColor(img_filter, cv2.COLOR_BGR2HSV)
        img_binary = cv2.inRange(img_filter.copy(),
        ↪  THRESHOLD_LOW, THRESHOLD_HIGH)

        img_binary = cv2.bilateralFilter(img_binary, 11, 17, 17)
        edged = cv2.Canny(img_binary, 30, 200)
        edged = cv2.dilate(edged, None, iterations = 1)
        edged = cv2.erode(edged, None, iterations = 1)

        _,contours, hierarchy = cv2.findContours(edged,
        ↪  cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)

        contour_list = []
        for contour in contours:
            approx =
            ↪  cv2.approxPolyDP(contour,0.01*cv2.arcLength(contour,True),True)
            area = cv2.contourArea(contour)
            #k=cv2.isContourConvex(approx)
            #print(k)
            if ((len(approx) >= 8) & (len(approx) < 23) & (area >
            ↪  30)):
                contour_list.append(contour)
        blank_image = np.zeros(edged.shape, np.uint8)

        cv2.drawContours(blank_image, contour_list,  -1,
        ↪  (255,0,0), 2)
        #cv2.imshow('Objects Detected',blank_image)
        #cv2.waitKey(0)

        circles = cv2.HoughCircles(blank_image,
        ↪  cv2.HOUGH_GRADIENT, 1.5,
        ↪  2000,param1=1,param2=50,minRadius=0,maxRadius=0)
        # ensure at least some circles were found
        print(circles)
        if circles is not None:
            # convert the (x, y) coordinates and radius of the
            ↪  circles to integers
```

```
            circles = np.round(circles[0, :]).astype("int")
            # loop over the (x, y) coordinates and radius of the
            ↪  circles
            for (x, y, r) in circles:
                # draw the circle in the output image, then draw
                ↪  a rectangle
                # corresponding to the center of the circle
                cv2.circle(output, (x, y), r, (0, 255, 0), 4)
                cv2.rectangle(output, (x - 5, y - 5), (x + 5, y +
                ↪  5), (0, 128, 255), -1)
            # show the output image
            #cv2.imshow("output", output)
            #cv2.waitKey(0)
            return True
        return False

    def takePict(self):
        my_file = open(lienImage, 'wb')
        self.camera.capture(my_file)
        my_file.close()
        #print("image")

    def findBall(self):
        self.takePict()
        img = cv2.imread(lienImage)
        return self.Looking_for_the_ball(img)
```

7 Les Compétences acquises

Les compétences acquises sont nombreuses :

- Récupérer un robot déjà monté, c'est à dire comprendre ce qui a été fait puis vérifier la fonctionnalité de l'ensemble des pièces. Calibrer le robot si besoin.
- Identifier les problèmes de matériel quand ils adviennent
- Collaborer à distance.
- Développer un logiciel en petites itérations incrémentales
- Gestion de l'autonomie énergétique du robot.
- Bluetooth Low Energy.
- Reconnaître une balle rouge.

- Gestion d'obstacles.
- ESP32 et Joystick.
- InfluxDB depuis un cloud et Telegraf en local ou en cloud.

8 Les problèmes

Nous avons eu différents problèmes lors de ce projet. En premier lieu des moteurs défaillants sur un robot, des batteries fatiguées ce qui diminue le temps d'utilisation. Une raspberry pi qui a grillé.
Une consommation importante du robot ce qui nous oblige a le recharger régulièrement.
Une utilisation de 15 à 20 minutes en continue pour une recharge de 3 à 4 heures.
Le robot n'avance pas de la même manière selon les sols.
Les robots n'ont pas la même vitesse pour la même tache, ce qui rend compliqué lorsqu'on demande un chemin particulier surtout quand il doit tourner lorsqu'un fait un demi-tour alors qu'un autre ne fait à peine un quart de tour.
La capteur d'ultrason HC-SR04 n'est pas assez précis pour déterminer des obstacles.

9 Github

L'ensemble des programmes est retrouvable sur le GitHub du projet.

10 Conclusion

Ainsi grâce à ce projet, nous avons pu voir différents aspects d'un petit kit de robotique et faire le lien avec quelques unes des nombreuses parties abordées lors de cursus en master SIAME. Entre autres, on peut citer un suivi en temps réel de données du robot, un contrôle à distance et une collaboration entre plusieurs robots. Ce projet nous a permis d'approfondir et de mettre en pratique des compétences déjà acquises lors de notre formation, ainsi que d'en développer de nouvelles en ce qui concerne l'aspect télétravail qui à été nécessaire à la réalisation de ce projet à visée pédagogique.