Guitter Julien

Projet Spider

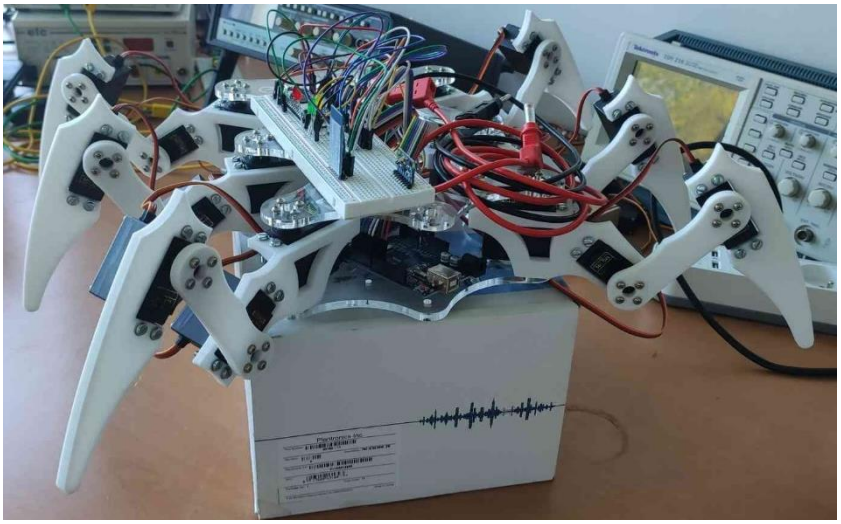


Table des matières

Introduction

Les idées initiales

Le projet initial était de construire un robot capable de se déplacer en autonomie dans l'enceinte de l'ESEO. De plus il doit être doté d'un bras robotique pour pouvoir déplacé différent objets. Pour le déplacement en autonomie, il doit être équiper d'un capteur lidar et de différents algorithmes pour trouver le chemin adéquate et éviter les obstacles.

Ce qui a été réalisé

Pour le projet d'EON, je me suis concentré sur les déplacements du robot, assigné par un humain soit sur une application téléphone ou sur une page web avec un retour vidéo. En laissant de coté toute la partie autonome et bras robotique. Pour un point de vue esthétique, et un défie de programmation, j'ai décider de le faire se déplacer comme une araignée et éviter d'utiliser des roues.

Les futures améliorations

Pour que le déplacement soit plus fluide et plus efficace, générer une simulation sur ordinateur de l'araignée et d'utiliser un algorithme de génération pour trouver les mouvement adéquate dans chaque situation. Comme par exemple la capaciter de monter des escaliers à l'aide des différents capteurs qu'elle détient comment un gyroscope un lidar.

Mais aussi lui ajouter un bras robotique à l'avant pour pouvoir interagir avec l'environnement extérieur comme pousser des objets ou encore récupérer des objets qu'on lui aura demander à l'avance.

Schéma synaptique global

Schéma initial

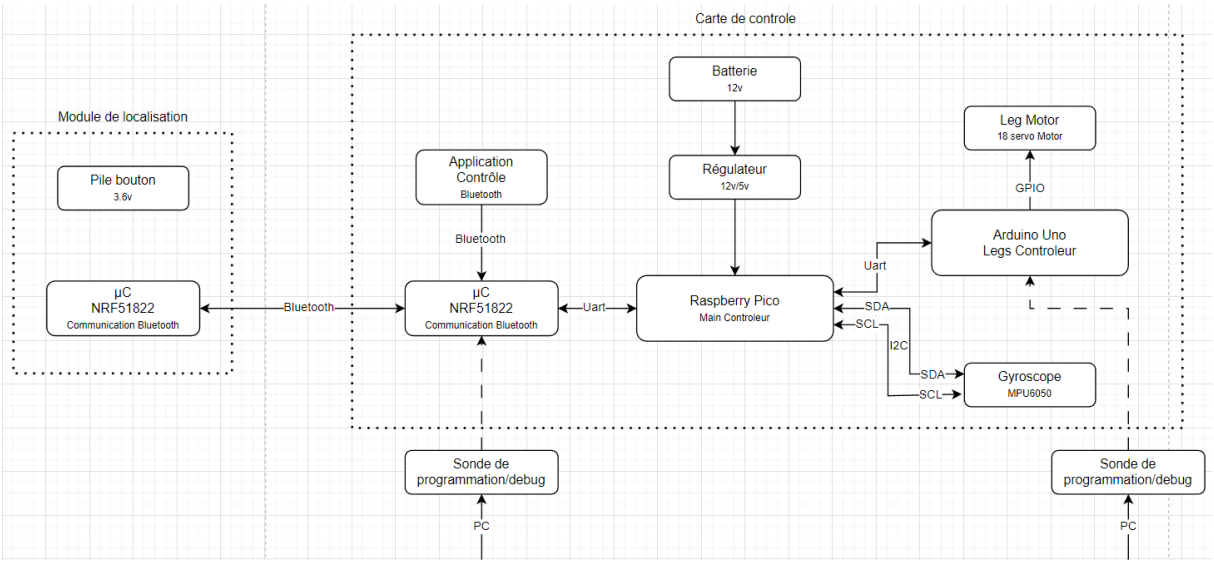
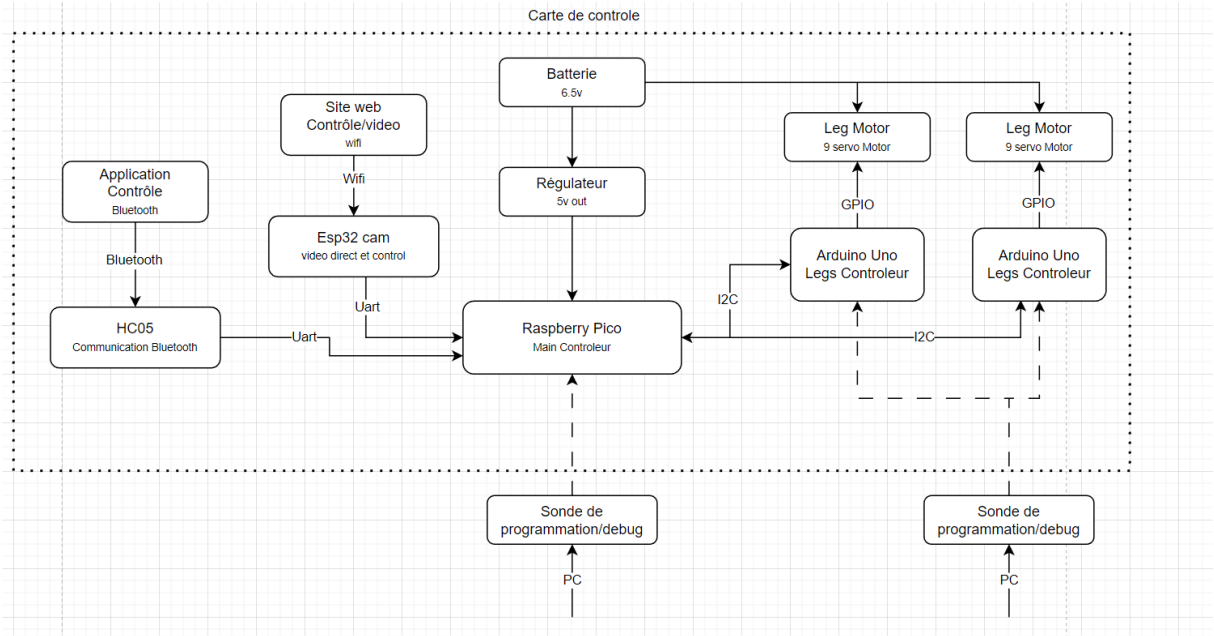


Schéma final

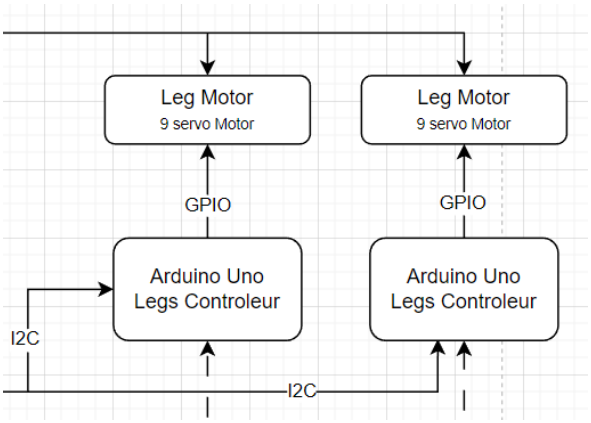


J'ai dû faire quelque changement sur le projet initial par manque de temps et des découvertes au fil du temps. Par exemple un Arduino ne peut contrôler plus de 12 servo-moteur. Il y avait 2 choix possible, changer de carte ou en mettre 2. Ou encore le Module de localisation a été remplacer par le control et la vision par wifi.

Les différentes parties

Les moteurs

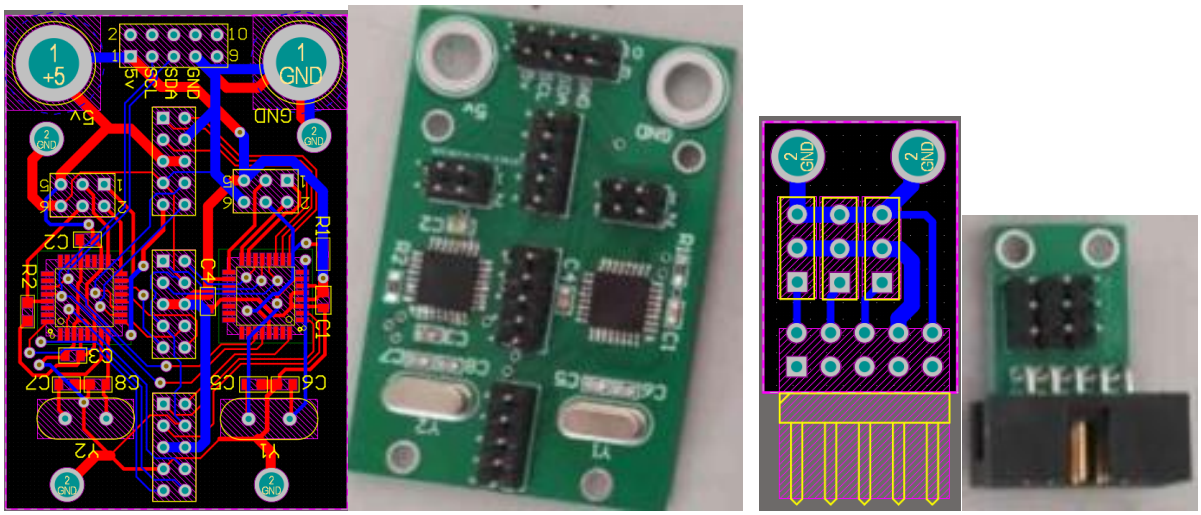
Schéma synaptique



Pour le contrôle des servo-moteur, j'ai choisi de prendre 2 Arduino car, une seule carte ne peut contrôler plus de 12 servos et que le robot en contient 18.

Pour envoyer les commandes au 2 cartes, j'utilise un bus i2c ce qui permet de dialoguer au 2 sur la même file. Il suffit de choisir l'identifiant renseigner dans le code.

Circuit électronique



J'ai choisi de faire plusieurs PCB pour que le jour ou quelqu'un souhaite changer une partie, il n'est qu'à reprendre celle voulut et pas tout.

Le PCB de gauche est la carte de control des moteurs contenant les 2 ATMega328p (Microcontrôleur Arduino) et à droite des connecteurs pour chaque pate, pour éviter d'avoir des câbles dans tous les sens.

Code de test

Arduino i2c réception

```cpp
// C++
#include <Wire.h>

#define A_ID          1

void setup() {
  Wire.begin(UNO_ADDR);
  Wire.onReceive(DataReceive);
}

void DataReceive(int numBytes){
  int i=0;
  char data[RESP_SIZE];
  char character;
  memset(data, 0, RESP_SIZE);
  while(Wire.available()){
    character = Wire.read();
    if(character == 'm'){
      newLegPos = true;
    }else if(character == 'e'){
      break;
    }else{
      data[i++]=character;
      delay(10);
    }
  }
  splitString(String(data), ':');
}
```

Raspberry i2c émission

```cpp
// C++
from machine import I2C

#init I2C
i2c = I2C(0, freq=100000)
addr = i2c.scan()

id = 1
command = "m3:90:50:90e";

i2c.writeto(addr[0 if id<=2 else 1], command)
```

Arduino rotation 1 servo moteur

```cpp
// C++
#include <Servo.h>

Servo leg;

Int legPin = 13;

Int angle = 90;


void setup() {
    leg.attach(legPin);
}


void loop() {
    leg.write(angle);
}
```

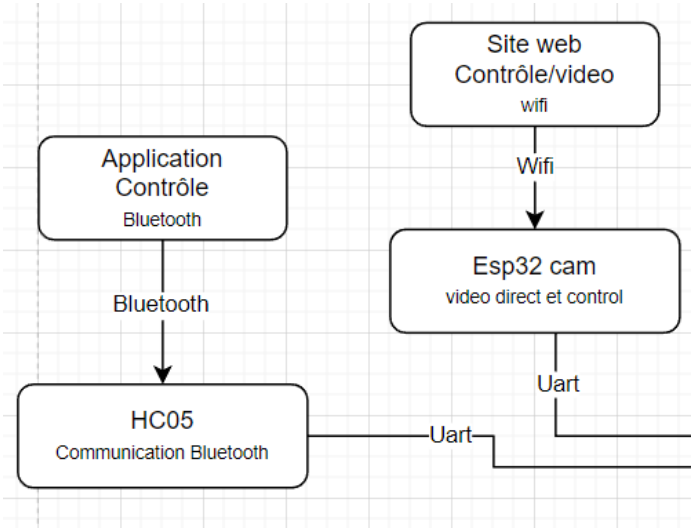
List position moteur pour les animations

```
#micro-pyhton
AnimList = {

    "stay":[
        [   [10,70,-60],        [10,70,-60]     ]
    ],
    "forward":[
        [   [-20, 90, -20],     None            ],
        [   [20, 90, -20],      [-20, 50, -30]  ],
        [   [20, 50, -30],      None            ],
        [   None,               [-20, 90, -20]  ],
        [   [-20, 50, -30],     [20, 90, -20]   ],
        [   None,               [20, 50, -30]   ]
    ],
    "backward":[
        [   [20, 90, -20],       None           ],
        [   [-20, 90, -20],      [20, 50, -30]  ],
        [   [-20, 50, -30],      None           ],
        [   None,                [20, 90, -20]  ],
        [   [20, 50, -30],       [-20, 90, -20] ],
        [   None,                [-20, 50, -30] ]
    ],
    "right":[
        [   [-20, 90, -20],     None            ],
        [   [20, 90, -20],      [20, 50, -30]   ],
        [   [20, 50, -30],      None            ],
        [   None,               [20, 90, -20]   ],
        [   [-20, 50, -30],     [-20, 90, -20]  ],
        [   None,               [-20, 50, -30]  ]
    ],
    "left":[
        [   [20, 90, -20],    None              ],
        [   [-20, 90, -20],     [-20, 50, -30]  ],
        [   [-20, 50, -30],     None            ],
        [   None,               [-20, 90, -20]  ],
        [   [20, 50, -30],    [20, 90, -20]     ],
        [   None,               [20, 50, -30]   ]
    ]
}
```

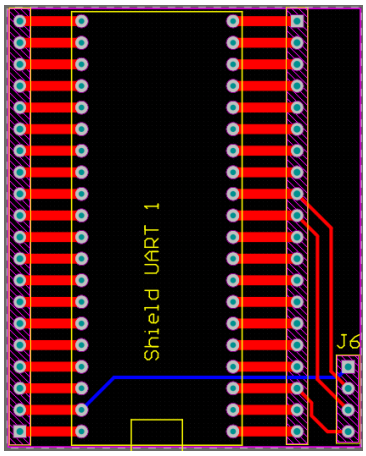
La connectivité

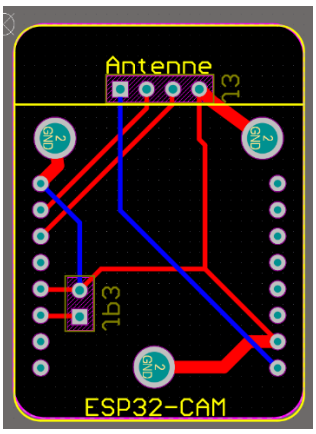
Schéma synaptique



Pour la partie communication, je suis parti sur un HC05 pour le Bluetooth et une Esp32 cam pour le wifi. Les 2 communique en UART avec la carte mère. Le HC05 est utile quand le robot n'a pas accès au wifi et que l'utilisateur l'a à l'œil.

Circuit électronique

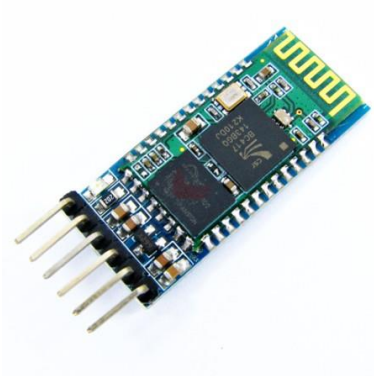


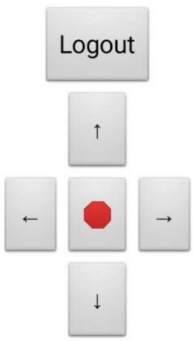


Pour cela j'ai réalisé un Shield qui vient ce clipser sur la carte mère, toujours dans l'intention d'avoir un circuit modulaire, ainsi qu'un PCB pour brancher et fixer l'esp32

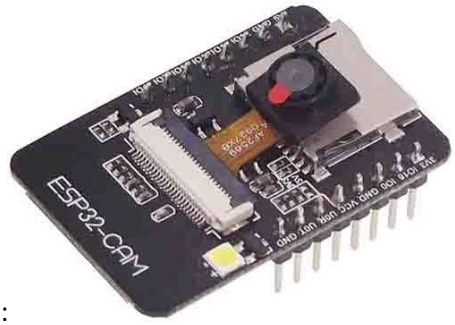
Composant et UI associer

HC05 :





Esp32 cam :



Code de test

Lecture message wifi et Bluetooth dans la carte mère

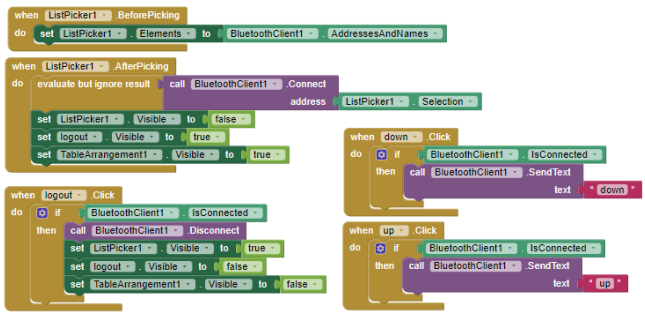
```micro-pyhton
#micro-pyhton
from machine import UART, Pin

#init uart
camUart = UART(1, baudrate=115200, tx=Pin(8), rx=Pin(9))
bluUart = UART(0, baudrate=9600, tx=Pin(16), rx=Pin(17))

while True:
    msg = camUart.read()
    if not msg:
        msg = bluUart.read()

    #execute action from msg
```

AppInventor blocks pour le Bluetooth



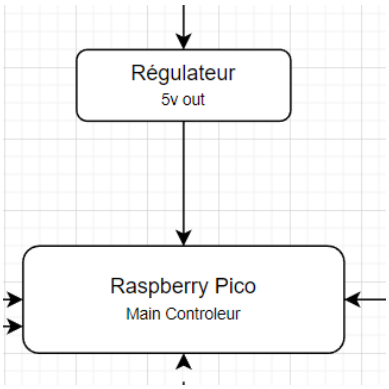
Requête js pour envoyer la commande

```
//js

var baseHost = "http://172.14.3.11"
function btnPressed(btn){
    request = `${baseHost}/move/${btn}`;
    console.log(request);
    fetch(request)
        .then(response => {
            console.log(`${response.status}`)
        })
}
```

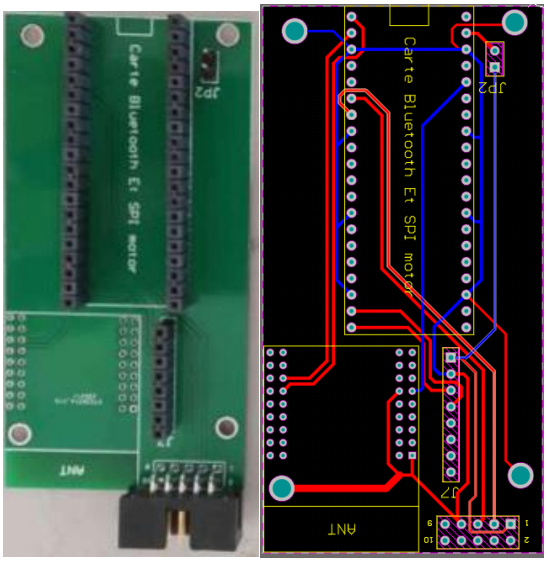
La carte mère

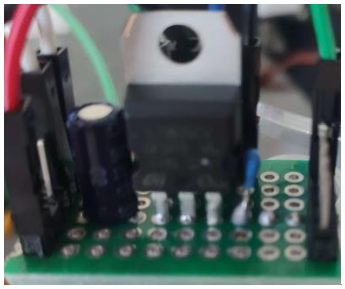
Schéma synaptique

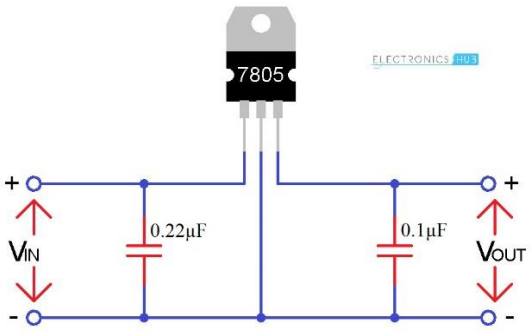


La carte mère est un circuit qui contient seulement la Raspberry pico qui contrôle tous les partis, et sont régulateur de tension car le robot est alimenté en 6.5v mais la Raspberry et le HC05 fonctionne avec une tension maximum de 5.5v.

Circuit électronique







Code de test

Les différents codes dans la Raspberry sont les codes de test montré dans les différentes partie (les codes en micro-python). Pour optimiser le code et la simplicité de programmation, la gestion des pates ce fait en orienté objet. Le code d'exemple suivant est la fonction principale qui appelle les fonctions de chaque pate pour définir leur nouvelle position et les envoyer dans le bus i2c.

```
playedAnim = "stay"
AnimList = {
      …
}
legPacks = [LegPack(0, i2c, addr, [0,0,0], [0,0,0], [0,0,0]),
            LegPack(1, i2c, addr, [0,0,0], [0,0,0], [0,0,0])]

def playStepAnim():
    global playedAnim, iAnim
    print(playedAnim, iAnim)
    newPos = AnimList[playedAnim][iAnim]

    if newPos[0]:
        if playedAnim == "left" or playedAnim == "right":
            legPacks[0].setLegArtsAngles(0, newPos[0])
            legPacks[0].setLegArtsAngles(1, newPos[0])
            legPacks[0].setLegArtsAngles(2, [-(newPos[0][0]), newPos[0][1], newPos[0][2]])
        else:
            legPacks[0].setSameLegsArtsAngles(newPos[0])
        legPacks[0].writeLegsArtsAngles()

    if newPos[1]:
        if playedAnim == "left" or playedAnim == "right":
            legPacks[1].setLegArtsAngles(0, [-(newPos[1][0]), newPos[1][1], newPos[1][2]])
            legPacks[1].setLegArtsAngles(1, newPos[1])
            legPacks[1].setLegArtsAngles(2, newPos[1])
        else:
            legPacks[1].setSameLegsArtsAngles(newPos[1])
        legPacks[1].writeLegsArtsAngles()

    iAnim = 0 if iAnim == len(AnimList[playedAnim])-1 else iAnim+1
```