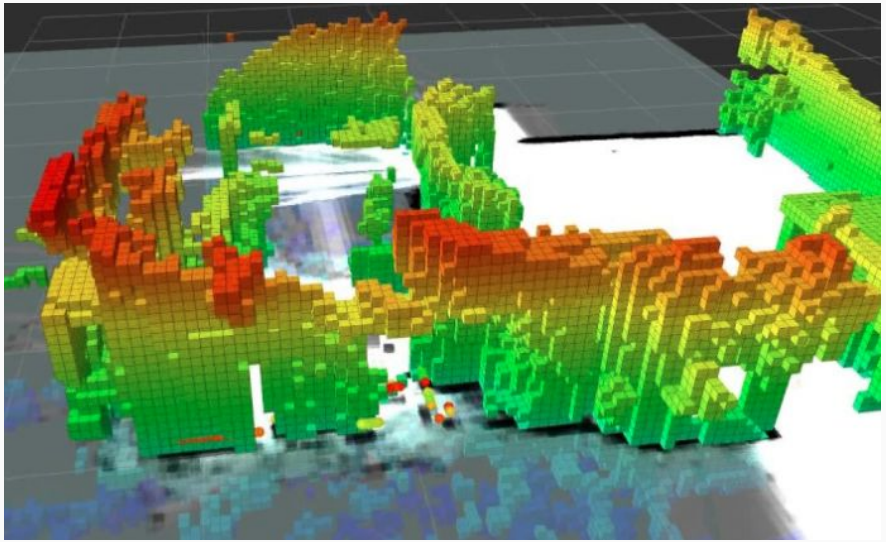
AKAYAD Noa Wissam Omar, Candidat : 39239, Session 2023.

Cartographie d’une pièce de maison par la méthode de localisation simultanée appliquée sur un robot.

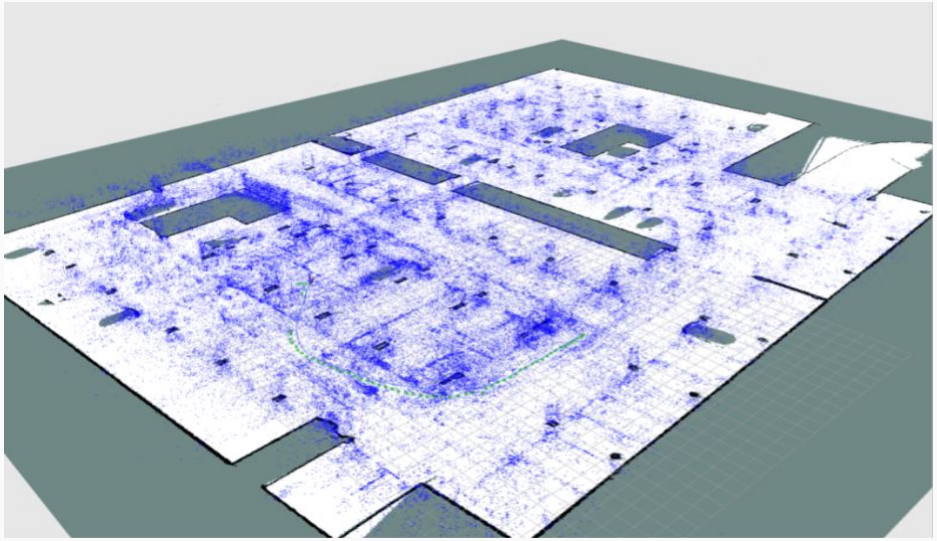
Comment obtenir une cartographie précise d'une pièce de maison en s’appuyant sur la méthode de localisation et de cartographie simultanées ?

Différentes méthodes courantes

SLAM



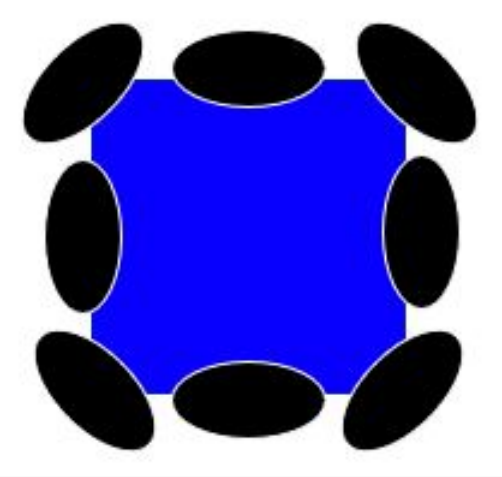
LIDAR

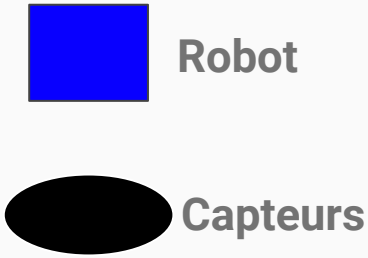


Cartographier une pièce de sa maison :

- **I : Algorithme**
- **II : Application sur un robot et une pièce de maison**
- **III : Études de complexités**
- **IV : Annexe**

I : Robot pour l’algorithme





I : Algorithme

Étape 1

Capter les points Nord Sud Est Ouest et aux diagonales.

Étape 2

Ajouter ces points à la liste des points à visiter

Étape 3

Réactualiser la carte

Étape 4

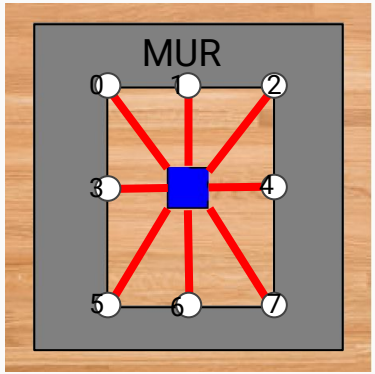
Se diriger vers le point (x,y) le plus proche de la liste.

Étape 5

Pendant le déplacement, recommencer l'étape 1 et 2.

Étape 6

Longer le mur puis recommencer à l' étape 1.



Onde des capteurs

Robot

Point capté

I : Algorithme

Étape 1

Capter les points Nord Sud Est Ouest et aux diagonales.

Étape 2

Ajouter ces points à la liste des points à visiter.

Étape 3

Réactualiser la carte.

Étape 4

Se diriger vers le point (x,y) le plus proche de la liste.

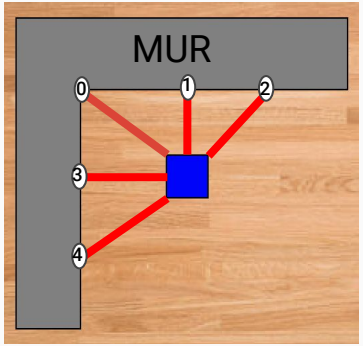
Étape 5

Pendant le déplacement, recommencer l'étape 1 et 2.

Étape 6

Longer le mur puis recommencer à l' étape 1.

Carte à l'instant t = Δt



Liste d'attente des points à t = Δt

[|0; 1; 2;
3;
4|]

I : Algorithme

Étape 1

Capter les points Nord Sud Est Ouest et aux diagonales.

Étape 2

Ajouter ces points à la liste des points à visiter

Étape 3

Réactualiser la carte

Étape 4

Se diriger vers le point (x,y) le plus proche de la liste.

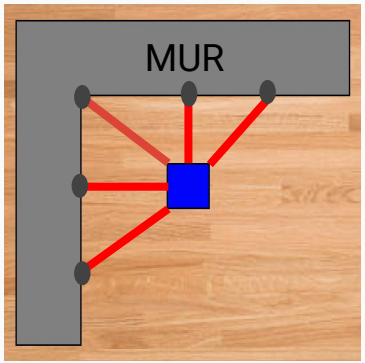
Étape 5

Pendant le déplacement, recommencer l'étape 1 et 2.

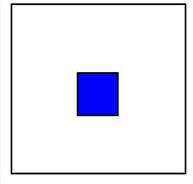
Étape 6

Longer le mur puis recommencer à l' étape 1.

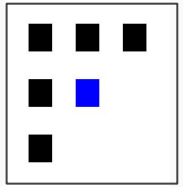
Carte à cartographier



Carte à l'instant t = 0



Carte à l'instant t = Δt



I : Algorithme

Étape 1

Capter les points Nord Sud Est Ouest et aux diagonales.

Étape 2

Ajouter ces points à la liste des points à visiter

Étape 3

Réactualiser la carte

Étape 4

Se diriger vers le point le plus proche de la liste.

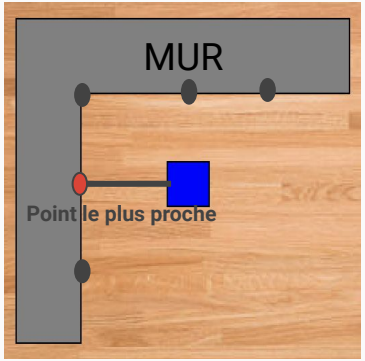
Étape 5

Pendant le déplacement, recommencer l'étape 1 et 2.

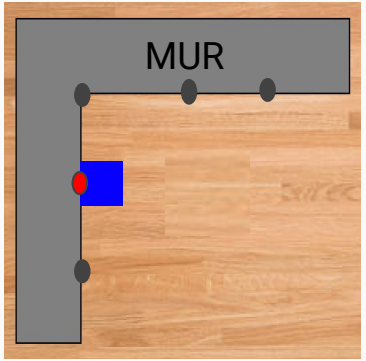
Étape 6

Longer le mur puis recommencer à l' étape 1.

Carte à l'instant t



Carte à l'instant t + Δt



I : Algorithme

Étape 1

Capter les points Nord Sud Est Ouest et aux diagonales.

Étape 2

Ajouter ces points à la liste des points à visiter

Étape 3

Réactualiser la carte

Étape 4

Se diriger vers le point (x,y) le plus proche de la liste.

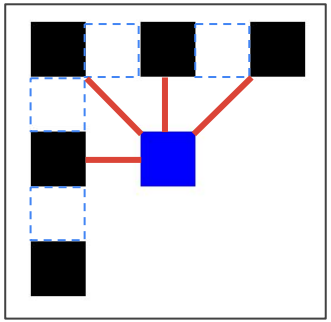
Étape 5

Pendant le déplacement, recommencer l'étape 1 et 2.

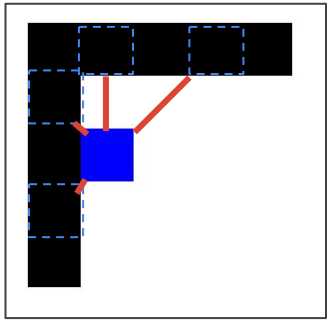
Étape 6

Longer le mur puis recommencer à l' étape 1.

Déplacement 0



Déplacement 1



I : Algorithme

Étape 1

Capter les points Nord Sud Est Ouest et aux diagonales.

Étape 2

Ajouter ces points à la liste des points à visiter

Étape 3

Réactualiser la carte

Étape 4

Se diriger vers le point (x,y) le plus proche de la liste.

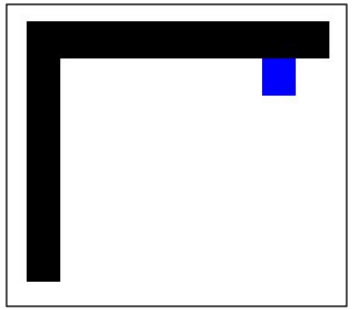
Étape 5

Pendant le déplacement, recommencer l'étape 1 et 2.

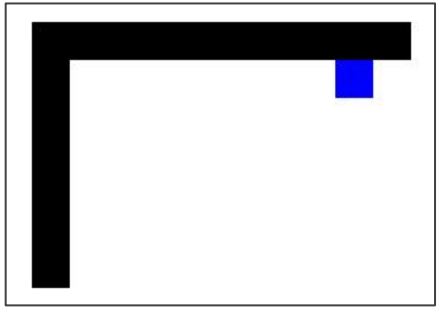
Étape 6

Longer le mur puis recommencer à l' étape 1.

Déplacement 5

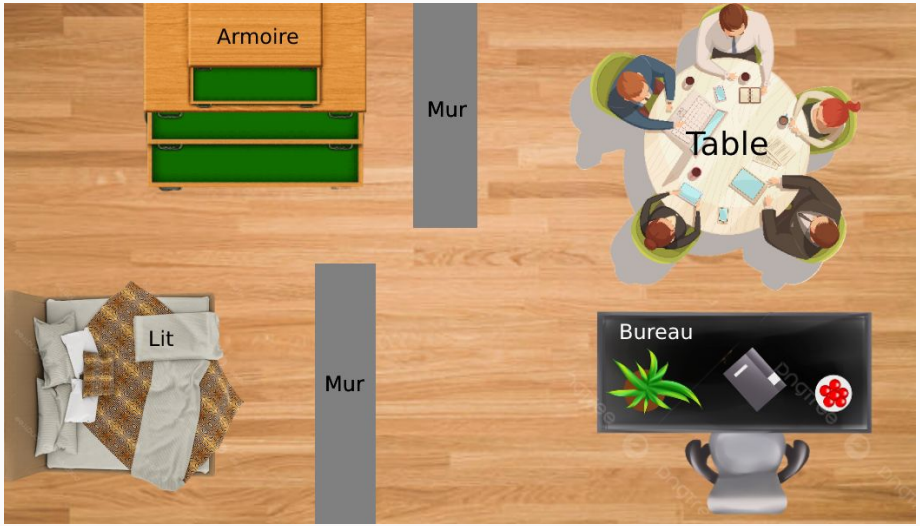


Déplacement 9

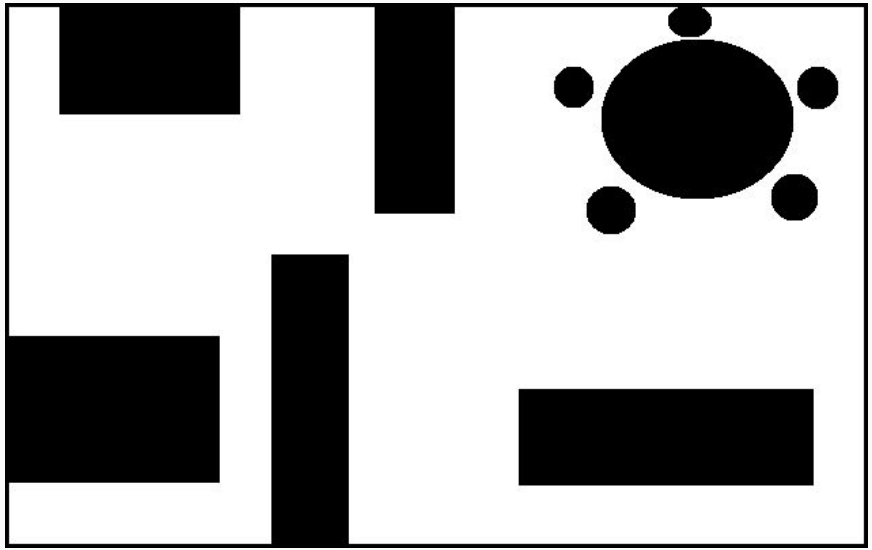


II : Application

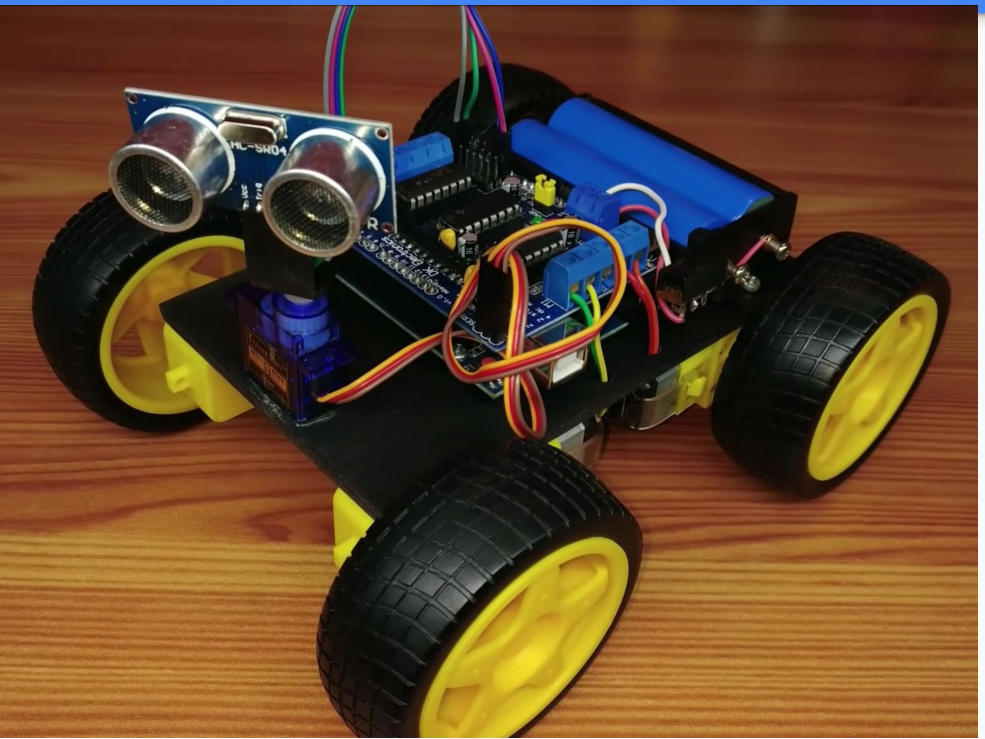
Pièce à cartographier



Carte que le robot doit renvoyer



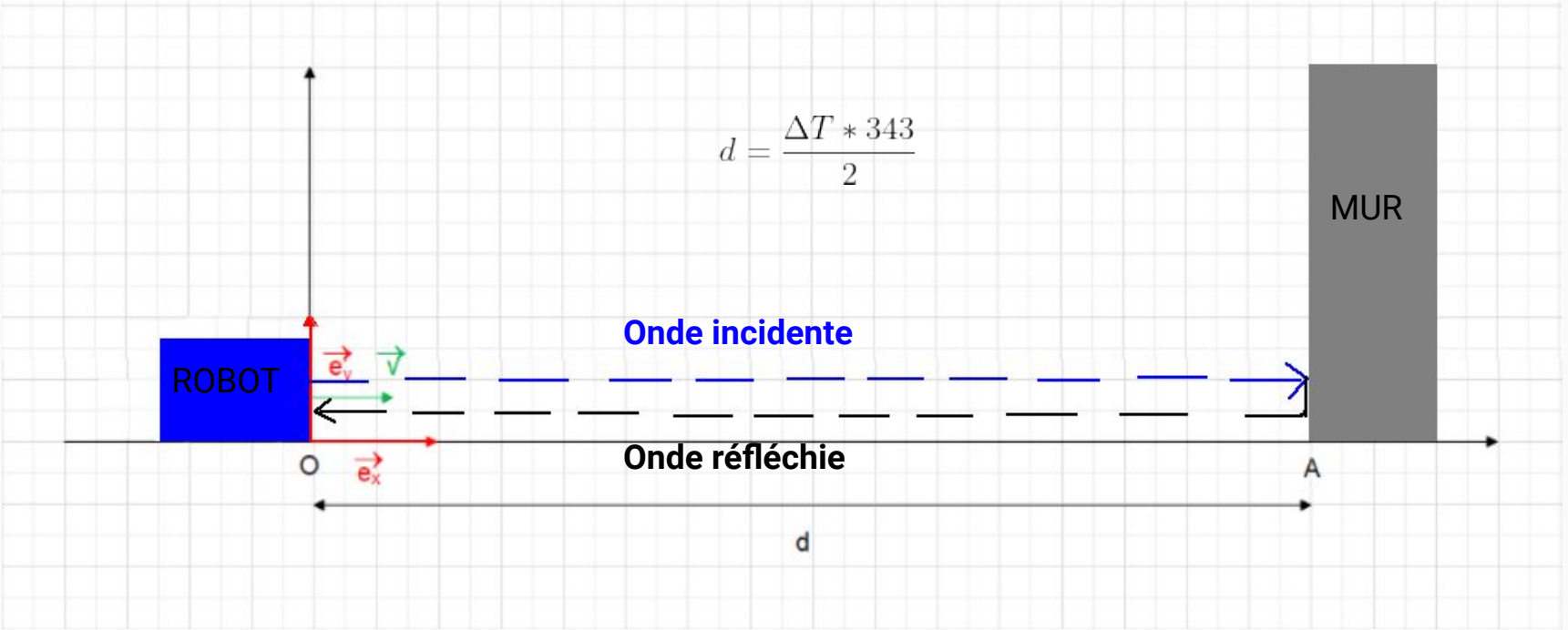
II : Le robot



- Microcontrôleur
- Moteur
- Roues
- Servo-moteur
- Emetteur-récepteur tournant
- Batteries
- Câbles de raccordement
- Plateau en bois

ARDUINO™

II : Application du code dans le robot

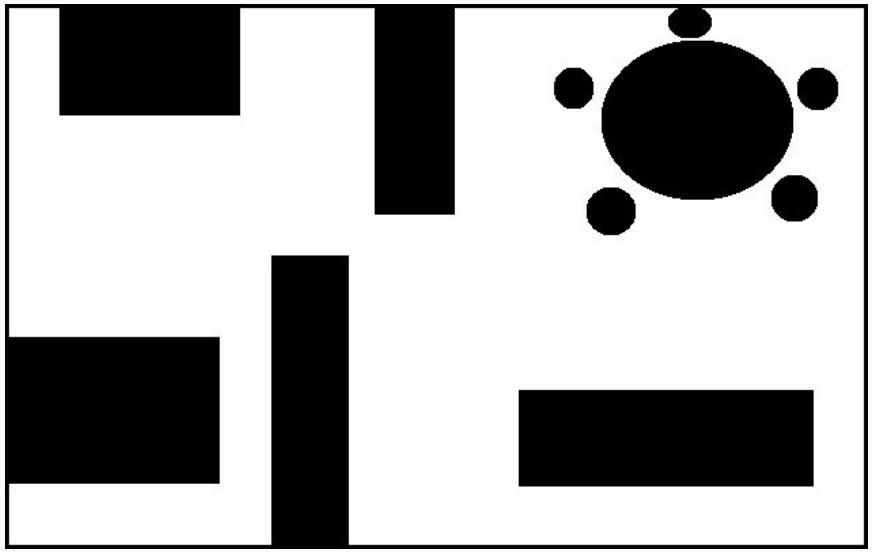


II : Construction du robot

photo de la construction

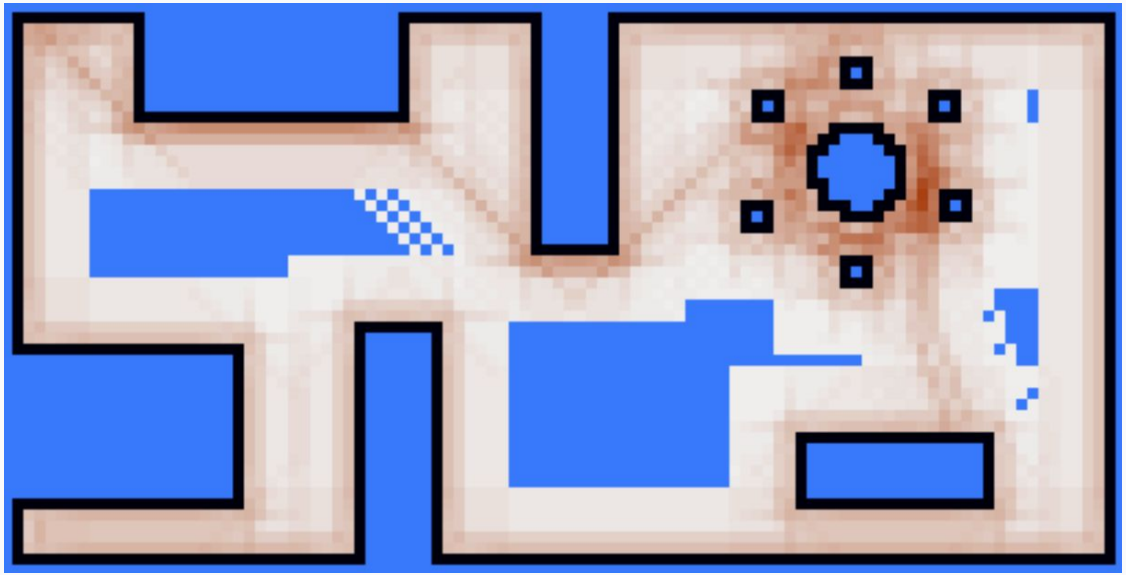
II : Application dans la pièce

Résultat attendu



Résultat obtenu

III : Efficacité de l’algorithme

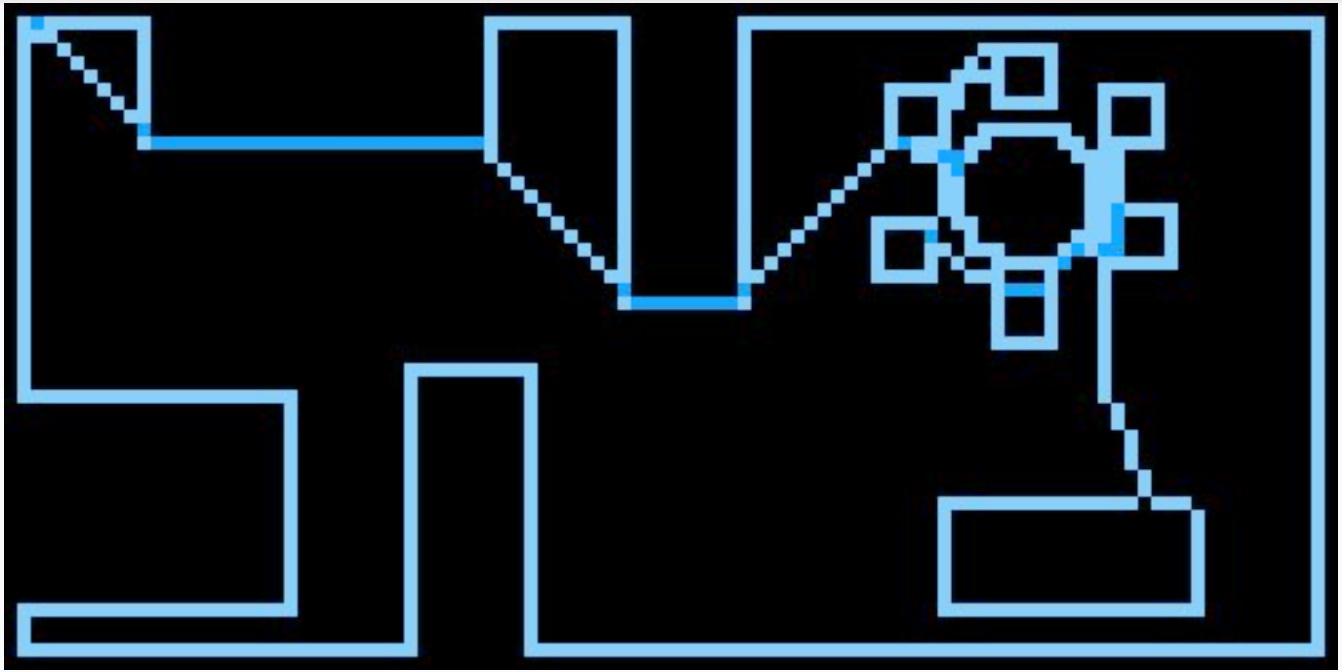


Mur découvert

Zone non découverte

Passage du robot

III : Efficacité de l’algorithme



Déplacement du robot

IV : Annexe

```
// Position du robot
float positionX = 0.0;
float positionY = 0.0;

// Types implémentés
struct Point {
  float x;
  float y;
  bool dec;
};

struct Liste {
  Point* point;
  Liste* suivant;
};

Liste* listeDePointsCarte = NULL;
Liste* listeDePointsVisite = NULL;
```

IV : Annexe

```
void ajouterPoint(float x, float y, Liste* liste) {
  if (liste->suivant == NULL){
    Point newpts = new Point;
    newpts.x = x;
    newpts.y = y;
    newpts.dec = false;
    Liste newlist = new Liste;
    newlist.point = newpts;
    newlist.suivant = NULL;

    liste->suivant = &newlist;
  }
  else{
    ajouterPoint(x, y, liste->suivant);
  }
}
```

IV : Annexe

```
// Fonction pour capturer un point et l'ajouter à la file d'attente
void capturerPoint1() {
  // Envoyer une onde sonore et recevoir la réponse
  unsigned int deltaT;

  // Envoi de l'onde
  sonar.ping_timer();  // Déclenche l'émission de l'onde

  // Attente de la réception de l'onde
  while (!sonar.check_timer()) {
    // Attendre jusqu'à ce que l'onde soit captée
  }

  // Calcul du temps écoulé (deltaT)
  deltaT = sonar.timer_result();

  // Calculer la distance en utilisant le temps de vol et la vitesse du son
  int distance = deltaT * 340 / 2;

  // Ajout du point dans la liste d'attente
  ajouterPoint(positionX, (positionY + distance), &listeDePointsVisite);

  ajouterPoint(positionX, (positionY + distance), &listeDePointsCarte);

}
```

IV : Annexe

```
void capturerPoint2() {

  rotation_capteur.attach(brocheRotation);  // Attacher le servo à la broche appropriée
  rotation_capteur.write(45); // tourner le capteur de 45 degrés vers la droite afin de recommencer avec l'autre point

  // Envoyer une onde sonore et recevoir la réponse
  unsigned int deltaT;

  // Envoi de l'onde
  sonar.ping_timer();  // Déclenche l'émission de l'onde

  // Attente de la réception de l'onde
  while (!sonar.check_timer()) {
    // Attendre jusqu'à ce que l'onde soit captée
  }

  // Calcul du temps écoulé (deltaT)
  deltaT = sonar.timer_result();

  // Calculer la distance en utilisant le temps de vol et la vitesse du son
  int distance = deltaT * 340 / 2;

  // Ajout du point dans la file d'attente
  ajouterPoint(positionX + sqrt(distance*distance(1 - sin(45)*sin(45))), (positionY + distance*sin(45)), &listeDePointsVisite);

  ajouterPoint(positionX + sqrt(distance*distance(1 - sin(45)*sin(45))), (positionY + distance*sin(45)), &listeDePointsCarte);

}
```

IV : Annexe

```
void capturerPoint3() {

  rotation_capteur.attach(brocheRotation);  // Attacher le servo à la broche appropriée
  rotation_capteur.write(45); // tourner le capteur de 45 degrés vers la droite afin de recommencer avec l'autre point

  // Envoyer une onde sonore et recevoir la réponse
  unsigned int deltaT;

  // Envoi de l'onde
  sonar.ping_timer();  // Déclenche l'émission de l'onde

  // Attente de la réception de l'onde
  while (!sonar.check_timer()) {
    // Attendre jusqu'à ce que l'onde soit captée
  }

  // Calcul du temps écoulé (deltaT)
  deltaT = sonar.timer_result();

  // Calculer la distance en utilisant le temps de vol et la vitesse du son
  int distance = deltaT * 340 / 2;

  // Ajout du point dans la file d'attente
  ajouterPoint(positionX + distance, positionY, &listeDePointsVisite);

  ajouterPoint(positionX + distance, positionY, &listeDePointsCarte);

}
```

IV : Annexe

```
void capturerPoint4() {

  rotation_capteur.attach(brocheRotation);  // Attacher le servo à la broche appropriée
  rotation_capteur.write(45); // tourner le capteur de 45 degrés vers la droite afin de recommencer avec l'autre point

  // Envoyer une onde sonore et recevoir la réponse
  unsigned int deltaT;

  // Envoi de l'onde
  sonar.ping_timer();  // Déclenche l'émission de l'onde

  // Attente de la réception de l'onde
  while (!sonar.check_timer()) {
    // Attendre jusqu'à ce que l'onde soit captée
  }

  // Calcul du temps écoulé (deltaT)
  deltaT = sonar.timer_result();

  // Calculer la distance en utilisant le temps de vol et la vitesse du son
  int distance = deltaT * 340 / 2;

  // Ajout du point dans la file d'attente
  ajouterPoint(positionX + sqrt(distance*distance(1 - sin(45)*sin(45))), (positionY - distance*sin(45)), &listeDePointsVisite);

  ajouterPoint(positionX + sqrt(distance*distance(1 - sin(45)*sin(45))), (positionY - distance*sin(45)), &listeDePointsCarte);

}
```

IV : Annexe

```
void capturerPoint5() {

  rotation_capteur.attach(brocheRotation);  // Attacher le servo à la broche appropriée
  rotation_capteur.write(45); // tourner le capteur de 45 degrés vers la droite afin de recommencer avec l'autre point

  // Envoyer une onde sonore et recevoir la réponse
  unsigned int deltaT;

  // Envoi de l'onde
  sonar.ping_timer();  // Déclenche l'émission de l'onde

  // Attente de la réception de l'onde
  while (!sonar.check_timer()) {
    // Attendre jusqu'à ce que l'onde soit captée
  }

  // Calcul du temps écoulé (deltaT)
  deltaT = sonar.timer_result();

  // Calculer la distance en utilisant le temps de vol et la vitesse du son
  int distance = deltaT * 340 / 2;

  // Ajout du point dans la file d'attente
  ajouterPoint(positionX, (positionY - distance), &listeDePointsVisite);

  ajouterPoint(positionX, (positionY - distance), &listeDePointsCarte);

}
```

IV : Annexe

```
void capturerPoint6() {

  rotation_capteur.attach(brocheRotation);  // Attacher le servo à la broche appropriée
  rotation_capteur.write(45); // tourner le capteur de 45 degrés vers la droite afin de recommencer avec l'autre point

  // Envoyer une onde sonore et recevoir la réponse
  ned int deltaT;

  // Envoi de l'onde
  sonar.ping_timer();  // Déclenche l'émission de l'onde

  // Attente de la réception de l'onde
  while (!sonar.check_timer()) {
    // Attendre jusqu'à ce que l'onde soit captée
  }

  // Calcul du temps écoulé (deltaT)
  deltaT = sonar.timer_result();

  // Calculer la distance en utilisant le temps de vol et la vitesse du son
  int distance = deltaT * 340 / 2;

  // Ajout du point dans la file d'attente
  ajouterPoint(positionX - distance*sin(45), positionY - sqrt(distance*distance(1 - sin(45)*sin(45))), &listeDePointsVisite);

  ajouterPoint(positionX - distance*sin(45), positionY - sqrt(distance*distance(1 - sin(45)*sin(45))), &listeDePointsCarte);

}
```

Search

IV : Annexe

```
void capturerPoint7() {

  Debug
  rotation_capteur.attach(brocheRotation);  // Attacher le servo à la broche appropriée
  rotation_capteur.write(45); // tourner le capteur de 45 degrés vers la droite afin de recommencer avec l'autre point

  // Envoyer une onde sonore et recevoir la réponse
  unsigned int deltaT;

  // Envoi de l'onde
  sonar.ping_timer();  // Déclenche l'émission de l'onde

  // Attente de la réception de l'onde
  while (!sonar.check_timer()) {
    // Attendre jusqu'à ce que l'onde soit captée
  }

  // Calcul du temps écoulé (deltaT)
  deltaT = sonar.timer_result();

  // Calculer la distance en utilisant le temps de vol et la vitesse du son
  int distance = deltaT * 340 / 2;

  // Ajout du point dans la file d'attente
  ajouterPoint(positionX - distance, positionY, &listeDePointsVisite);

  ajouterPoint(positionX - distance, positionY, &listeDePointsCarte);

}
```

IV : Annexe

```
void capturerPoint8() {

  rotation_capteur.attach(brocheRotation);  // Attacher le servo à la broche appropriée
  rotation_capteur.write(45); // tourner le capteur de 45 degrés vers la droite afin de recommencer avec l'autre point

  // Envoyer une onde sonore et recevoir la réponse
  unsigned int deltaT;

  // Envoi de l'onde
  sonar.ping_timer();  // Déclenche l'émission de l'onde

  // Attente de la réception de l'onde
  while (!sonar.check_timer()) {
    // Attendre jusqu'à ce que l'onde soit captée
  }

  // Calcul du temps écoulé (deltaT)
  deltaT = sonar.timer_result();

  // Calculer la distance en utilisant le temps de vol et la vitesse du son
  int distance = deltaT * 340 / 2;

  // Ajout du point dans la file d'attente
  ajouterPoint(positionX - sqrt(distance*distance(1 - sin(45)*sin(45))), positionY + distance*sin(45), &listeDePointsVisite);

  ajouterPoint(positionX - sqrt(distance*distance(1 - sin(45)*sin(45))), positionY + distance*sin(45), &listeDePointsCarte);

  rotation_capteur.attach(brocheRotation);  // Attacher le servo à la broche appropriée
  rotation_capteur.write(45); // remettre le capteur à 0 degré
}
```

IV : Annexe

```
// Fonction pour déterminer le point le plus proche à visiter
Point determinerPointPlusProche() {
  if (listeDePointsVisite == NULL) {
  return {0.0, 0.0, false}; // Retourner un point par défaut si la liste est vide
  }

  // Initialiser les variables pour le point le plus proche
  float distanceMin = INFINITY;
  Point pointLePlusProche = {0.0, 0.0, false};

  // Parcourir tous les points de la liste et trouver le point le plus proche non découvert
  Liste* listeCourante = listeDePointsVisite;
  while (listeCourante != NULL) {
    // Vérifier si le point est découvert et calculer la distance s'il est non découvert
    if (!listeCourante->point->dec) {
      float distance = sqrt(pow(positionX - listeCourante->point->x, 2) + pow(positionY - listeCourante->point->y, 2));
      if (distance < distanceMin) {
        distanceMin = distance;
        pointLePlusProche = *(listeCourante->point);
      }
    }
    listeCourante = listeCourante->suivant;
  }

  return pointLePlusProche;
}
```

IV : Annexe

```
// Fonction pour déplacer le robot vers le point cible
void deplacerRobotVersPoint(Point* pointPlusProche) {

  pointPlusProche->dec = true;

  // Calculer la distance et l'angle entre la position actuelle et la destination
  float distance = sqrt(pow(pointPlusProche->x - positionX, 2) + pow(pointPlusProche->y - positionY, 2));
  float angle = atan2(pointPlusProche->y - positionY, pointPlusProche->x - positionX);

  // Calculer la vitesse des roues gauche et droite en fonction de l'angle
  float vitesseGauche = VITESSE_MOTEUR * cos(angle);
  float vitesseDroite = VITESSE_MOTEUR * sin(angle);

  // Variables pour la gestion du temps
  unsigned long tempsDebut = millis();
  unsigned long intervalleCapturePoints = 1000;  // Intervalle de 1 seconde pour capturer les points

  // Avancer vers la destination
  digitalWrite(PIN_MOTEUR_GAUCHE_A, HIGH);
  digitalWrite(PIN_MOTEUR_GAUCHE_B, LOW);
  analogWrite(PIN_MOTEUR_GAUCHE_PWM, vitesseGauche);
  digitalWrite(PIN_MOTEUR_DROIT_A, HIGH);
  digitalWrite(PIN_MOTEUR_DROIT_B, LOW);
  analogWrite(PIN_MOTEUR_DROIT_PWM, vitesseDroite);

  while (distance > 0) {
  // Vérifier si le temps écoulé dépasse l'intervalle de capture des points
  if (millis() - tempsDebut >= intervalleCapturePoints) {
    // Capturer les points
    CapturerPoints();

    // Réinitialiser le temps de début
    tempsDebut = millis();
  }
  // Calculer la distance parcourue pendant l'intervalle de capture des points
  float distanceParcourue = VITESSE_MOTEUR * (intervalleCapturePoints / 1000.0);

  // Mettre à jour la position du robot en fonction de l'angle et de la distance parcourue
  positionX += cos(angle) * distanceParcourue;
  positionY += sin(angle) * distanceParcourue;

  // Mettre à jour la distance restante
  distance -= distanceParcourue;
  }

  // Arrêter les moteurs
  digitalWrite(PIN_MOTEUR_GAUCHE_A, LOW);
  digitalWrite(PIN_MOTEUR_GAUCHE_B, LOW);
  digitalWrite(PIN_MOTEUR_DROIT_A, LOW);
  digitalWrite(PIN_MOTEUR_DROIT_B, LOW);

  // Capturer les points une dernière fois
  CapturerPoints();
}
```

IV : Annexe

```
struct Point_int {
  int x;
  int y;
};

struct Liste_int {
  Point_int* point;
  Liste_int* suivant;
};
```

```
Liste_int* floatListe_to_intListe(Liste* liste) {
  Liste_int* newliste = new Liste_int;
  if (liste != NULL) {
    Point_int* newpoint = new Point_int;
    newpoint->x = int(liste->point->x);
    newpoint->y = int(liste->point->y);
    newliste->point = newpoint;
    newliste->suivant = floatListe_to_intListe(liste->suivant);
  } else {
    newliste->suivant = NULL;
  }
  return newliste;
}
```

IV : Annexe

```
int** tab = new int*[l];
  for (int i = 0; i < l; i++){
    int* tabc = new int[c];
    tab[i] = tabc;
    for (int j = 0; j < c; j++){
      tab[i][j] = 0;
    }
  }
}

void tab2dim (Liste_int* liste){
  if (liste != NULL){
    tab[liste->point->x][liste->point->y] = 1;
    tab2dim(liste->suivant);
  }
}
```

IV : Annexe

```
void genererImageTableau(int** tableau, int lignes, int colonnes) {
  // Créer une image en mémoire
  Adafruit_Image image = Adafruit_Image(lignes, colonnes);

  // Parcourir le tableau et définir les pixels de l'image
  for (int i = 0; i < lignes; i++) {
    for (int j = 0; j < colonnes; j++) {
      int caseValue = tableau[i][j];
      if (caseValue == 1) {
        image.setPixel(i, j, 0); // Noir
      } else {
        image.setPixel(i, j, 255); // Blanc
      }
    }
  }

  // Créer un nom de fichier unique basé sur l'horodatage
  char filename[13];
  sprintf(filename, "%lu.png", millis());
```

```
  // Créer un nom de fichier unique basé sur l'horodatage
  char filename[13];
  sprintf(filename, "%lu.png", millis());

  // Sauvegarder l'image dans la carte SD
  if (SD.begin()) {
    File file = SD.open(filename, FILE_WRITE);
    if (file) {
      Adafruit_ImageReader::drawBMP(file, image);
      file.close();
      Serial.println("Image enregistrée avec succès.");
    } else {
      Serial.println("Erreur lors de l'ouverture du fichier.");
    }
  } else {
    Serial.println("Erreur lors de l'initialisation de la carte SD.");
  }
}
```

IV : Annexe

```
uint8_t* genererPNG(int** tableau, int lignes, int colonnes, size_t* taille) {
  // Créer une image en mémoire
  Adafruit_Image image = Adafruit_Image(lignes, colonnes);

  // Parcourir le tableau et définir les pixels de l'image
  for (int i = 0; i < lignes; i++) {
    for (int j = 0; j < colonnes; j++) {
      int caseValue = tableau[i][j];
      if (caseValue == 1) {
        image.setPixel(i, j, 0x0000); // Couleur noire (RGB565)
      } else {
        image.setPixel(i, j, 0xFFFF); // Couleur blanche (RGB565)
      }
    }
  }

  // Créer un buffer pour les données du PNG
  size_t bufferTaille = image.width() * image.height() * 2; // 2 octets par pixel (RGB565)
  uint8_t* buffer = (uint8_t*)malloc(bufferTaille);
  if (!buffer) {
    Serial.println("Erreur lors de l'allocation du tampon de données.");
    return NULL;
  }

  // Encoder l'image en format PNG dans le buffer
  if (!image.encode(buffer, bufferTaille, ImageFormat_PNG)) {
    Serial.println("Erreur lors de l'encodage de l'image en PNG.");
    free(buffer);
    return NULL;
  }

  // Mettre à jour la taille du PNG
  *taille = image.encodedSize();

  return buffer;
}
```

IV : Annexe

```
void loop() {
  // Capturer les points et les ajouter aux listes
  capturerPoints();

  // Déterminer le point à visiter le plus proche
  Point plusProche = determinerPointPlusProche();

  // Y aller et recapturer les points pendant le déplacement
  deplacerRobotVersPoint(&plusProche);

  // Longer le mur auquel appartient ce point et recapturer les points pendant le longement
  longerMur();

}
```