TD 4 - Principes des Architectures des Systèmes Autonomes Intelligents

BEZES Bastien, GRAFFAN Jérémy, EL KATEB Sami

Code source et vidéos

Le code complet de réponse aux différentes questions ainsi que les vidéos sont disponibles sur notre Repository GitHub.

Positionnement et nom des capteurs

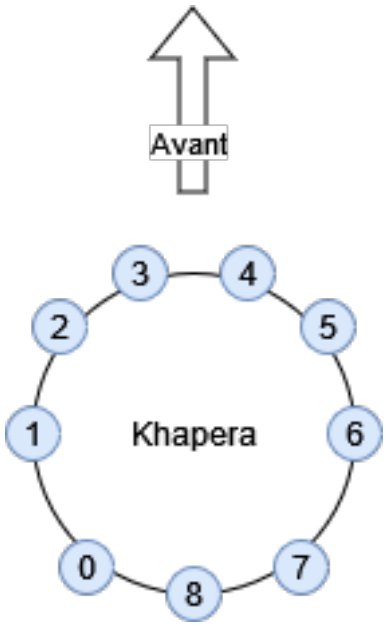


Figure 1: Positionnement des capteurs Khapera III

Question 2 (Simulateur)

Reprenez votre TD précédent et implémentez le déplacement de votre robot en ligne droite et son arrêt devant le premier obstacle rencontré.

Principe de fonctionnement

Pour arrêter le robot devant le premier obstacle rencontré nous utilisons les 4 capteurs situés a l'avant du robot. Le robot avance en ligne droite avec une vitesse constante. Lorsque l'un des capteurs détecte un obstacle (valeur supérieur a une limite définie), le robot s'arrête. Si l'obstacle disparaît, le robot recommence a avancer en ligne droite.

Vidéo

Téléchargement de la vidéo

Code

```
static const int front_sensor_indexes[FRONT_SENSOR_COUNT] = { 2, 3, 4, 5 };
static const double speed = 10;

while (wb_robot_step(time_step) != -1) {
  double sensors_value[FRONT_SENSOR_COUNT];
  double current_speed = speed;

  for (int i = 0; i < FRONT_SENSOR_COUNT; i++) {
    sensors_value[i] = wb_distance_sensor_get_value(sensors[front_sensor_indexes[i]]);
    if (sensors_value[i] > 100.0 )
    {
      current_speed = 0;
      break;
    }
  }

  wb_motor_set_velocity(left_motor, current_speed);
  wb_motor_set_velocity(right_motor, current_speed);
}
```

Code complet

Equation

$$V_r = \begin{cases} 9, & \text{if } \forall x \in X_{front}, x \leq 100 \\ 0, & \text{otherwise} \end{cases}$$

$$V_l = \begin{cases} 9, & \text{if } \forall x \in X_{front}, x \leq 100 \\ 0, & \text{otherwise} \end{cases}$$

Question 3 (Simulateur)

Avec le simulateur Webots, en utilisant le robot khepera III, implémentez cet algorithme en utilisant tous les proximétres du robot. Choisissez les bons poids

pour que le robot est un comportement satisfaisant.

Matrice de poids pour les capteurs 1 à 9

$$W = \begin{pmatrix} -2.67 & -10.86 & -16.03 & -37.4 & 37.4 & 26.71 & 21.37 & -2.67 & -5.34 \\ -2.67 & 21.37 & 26.71 & 37.4 & -32.06 & -21.37 & -10.86 & -2.67 & -5.34 \end{pmatrix}$$

Équation

$$Vr = k * \sum_{x=0}^{8} (W_{ri}.X_i)$$

$$\begin{aligned} Vr = 1 * (&-2.67.X_0 + 21.37.X_1 + 26.71.X_2 \\ &+37.4.X_3 - 32.06.X_4 - 21.37.X_5 \\ &-10.86.X_6 - 2.67.X_7 - 5.34.X_8) \end{aligned}$$

$$Vl = k * \sum_{x=0}^{8} (W_{li}.X_i)$$

$$\begin{aligned} Vl = 1 * (&-2.67.X_0 - 10.86.X_1 - 16.03.X_2 \\ &-37.4.X_3 + 37.4.X_4 + 26.71.X_5 \\ &+21.37.X_6 - 2.67.X_7 - 5.34.X_8) \end{aligned}$$

Code

Code complet

Question 4

Comment implémenteriez-vous un réseau de neurones pour l'évitement d'obstacles selon braintenberg dans le cas de l'alphabot2 ?

Pour simuler l'AlphaBot dans le cadre du logiciel Webot nous avons réutilisé le Khepera III en conservant uniquement 2 capteurs avant et en binarisant leur valeur (0 ou 1 à partir d'un seuil).

```
double binarize_sensor_value(double sensor_value) {
  return sensor_value > 40 ? 1 : 0;
}

void process() {
  const double alphabot_matrix[2][2] = {{-9, 9}, {9, -9}};
  const int alphabot_sensor_indexes[ALPHABOT_SENSOR_COUNT] = {2, 5};
  const double base_speed = 10;
  double speed[2] = {0, 0};
```

```
  while (wb_robot_step(time_step) != -1) {
    double sensors_value[ALPHABOT_SENSOR_COUNT];

    for (int i = 0; i < ALPHABOT_SENSOR_COUNT; i++) {
      double initial_sensor_value =
          wb_distance_sensor_get_value(sensors[alphabot_sensor_indexes[i]]);
      sensors_value[i] = binarize_sensor_value(initial_sensor_value);
    }

    for (int i = 0; i < 2; i++) {
      speed[i] = base_speed;

      for (int j = 0; j < ALPHABOT_SENSOR_COUNT; j++) {
        speed[i] += alphabot_matrix[j][i] * (1.0 - sensors_value[j]);
      }
      speed[i] = BOUND(speed[i], -max_speed, max_speed);
    }

    wb_motor_set_velocity(left_motor, speed[0]);
    wb_motor_set_velocity(right_motor, speed[1]);
  }
}
```

Pour implémenter le réseau de neurone selon Braintenberg, nous utilisons une vitesse constante de 10 et des poids symétriques de -9 et 9.

Code complet

Équation

Nous considérons X_0 le capteur gauche a X_1 le capteur droit.

$$Vr = k * \sum_{x=0}^{2} (W_{ri}.X_i) = 1 * (-9.X_0 + 9.X_1)$$

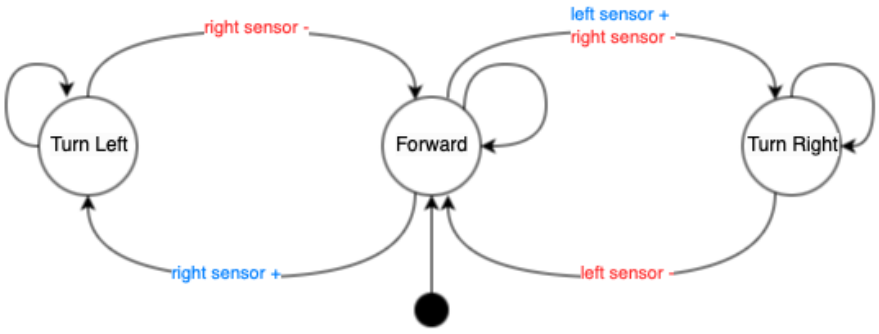
$$Vl = k * \sum_{x=0}^{2} (W_{li}.X_i) = 1 * (9.X_0 + -9.X_1)$$

Question 5

Si vous deviez fournir un algorithme d'évitement d'obstacle pour l'Alphabot2, sous forme d'un automate à état fini, quel serait-il (1) ? Implémentez et testez.

Automate

L'AlphaBot démarre dans l'état FORWARD où il avance tout droit. Si le capteur gauche détecte un objet et non le capteur droit, l'AlphaBot tourne vers la droite. Lorsque le capteur ne détècte plus d'objet, l'AlphaBot recommence à se déplacer en ligne droite. Si le capteur droit détecte un object (seul ou accompagné du capteur gauche), l'Alphabot tourne vers la gauche. Lors des mouvements de rotation, une seule des deux roues est en mouvement.



Vidéo

Téléchargement de la vidéo

Code

```
enum State {
  FORWARD,
  ROTATE_RIGHT,
  ROTATE_LEFT
};

void process() {
  const int alphabot_sensor_indexes[ALPHABOT_SENSOR_COUNT] = {3, 4};
  double speed[2] = {0, 0};
  enum State state = FORWARD;

  while (wb_robot_step(time_step) != -1) {
    double sensors_value[ALPHABOT_SENSOR_COUNT];

    for (int i = 0; i < ALPHABOT_SENSOR_COUNT; i++) {
      double initial_sensor_value =
          wb_distance_sensor_get_value(sensors[alphabot_sensor_indexes[i]]);
      sensors_value[i] = binarize_sensor_value(initial_sensor_value);
    }
```

```
    if(sensors_value[0] == 0 && sensors_value[1] == 0) {
      state = FORWARD;
    } else if (sensors_value[0] == 0 && sensors_value[1] == 1) {
      state = ROTATE_LEFT;
    } else if (sensors_value[0] == 1) {
      state = ROTATE_RIGHT;
    }

    switch (state) {
      case FORWARD:
        speed[0] = 19;
        speed[1] = 19;
        break;
      case ROTATE_LEFT:
        speed[0] = 0;
        speed[1] = 19;
        break;
      case ROTATE_RIGHT:
        speed[0] = 19;
        speed[1] = 0;
        break;
    }

    wb_motor_set_velocity(left_motor, speed[0]);
    wb_motor_set_velocity(right_motor, speed[1]);
  }
}
```

Code complet

Question 6 (Simulateur)

Quel algorithme mettriez-vous en place pour un suivi de contours d'obstacles par la droite sur le Khepera III ? Implémentez et testez.

Pour éviter un obstacle par la droite, il est dans un premier temps nécessaire de tourner vers la gauche lorsque nous rencontrons un obstacle pour que celui-ci se trouve à droite du robot. Pour effectuer le suivit de l'obstacle nous pouvons ensuite accélérer la roue droite lorsque nous nous rapprochons trop de l'obstacle et la ralentir lorsque nous nous en éloignons pour conserver la distance désirée. Les capteurs du Khepera sont sensibles il est possible de perdre de vue un obstacle que nous sommes entrain de suivre. Pour palier ce problème nous pouvons utiliser un compteur et une machine à état pour conserver le suivit de l'obstacle sur une durée convenable.

Vidéo

Téléchargement de la vidéo

Code

```
enum State {
  AVOIDING,
  FORWARD
};

void process() {
  enum State state = FORWARD;

  double CRITICAL_DISTANCE = 100;
  int obstacle_avoidance_counter = 0;

  while (wb_robot_step(time_step) != -1) {
    double speed[2] = {0, 0};
    double sensors_value[SENSOR_NUMBER];
    for (int i = 0; i < SENSOR_NUMBER; i++) {
      sensors_value[i] = wb_distance_sensor_get_value(sensors[i]);
    }

    double max_right = fmax(sensors_value[4], fmax(sensors_value[5], sensors_value[6]));
    double max_left = fmax(sensors_value[1], fmax(sensors_value[2], sensors_value[3]));

    if (max_right > CRITICAL_DISTANCE || max_left > CRITICAL_DISTANCE) {
      state = AVOIDING;
      obstacle_avoidance_counter = STICKYNESS;
    } else if(obstacle_avoidance_counter <= 0){
        state = FORWARD;
    }

    if (state == AVOIDING) {
      if (max_left > CRITICAL_DISTANCE) {
          speed[1] = 10.0;
          speed[0] = -10.0;
      } else if (max_right > CRITICAL_DISTANCE) {
          speed[1] = 10.0;
          speed[0] = -10.0;
      } else {
        speed[0] = 12.0;
        speed[1] = 3.0;
      }
      obstacle_avoidance_counter--;
```

```
    }

    if(state == FORWARD) {
        speed[0] = 10.0;
        speed[1] = 10.0;
    }

    speed[0] = BOUND(speed[0], -max_speed, max_speed);
    speed[1] = BOUND(speed[1], -max_speed, max_speed);

    wb_motor_set_velocity(left_motor, speed[0]);
    wb_motor_set_velocity(right_motor, speed[1]);
  }
}
```

Code complet

Question 7

Si vous voulez implémenter un suivi de ligne grâce aux capteurs du robot Alphabot2, que proposeriez-vous comme algorithme ? Implémentez et testez.

Question 8 (Simulateur)

Mettez en place deux stratégies de coordination différentes et testez les différences

Stratégie 1: Subsomption

Notre première approche à consisté en une architecture par subsomption. Dans notre cas nous avons 2 couches:

- une couche d'évitement (`obstacle_back` & `avoid_right`)
- une couche d'exploration (`wander` & `go_forward`)

Seul le comportement `obstacle_back` peut accéder aux moteurs.

Code:

```
void subsumption_architecture() {
  double speed[2] = {0, 0};
  double proposed_speeds[2] = {0, 0};
  int wander_cooldown = WANDER_COOLDOWN;
  int wander_remaining = 0;
  int wander_direction = 0;

  while (wb_robot_step(time_step) != -1) {
    double sensors_value[SENSOR_NUMBER];
    for (int i = 0; i < SENSOR_NUMBER; i++) {
      sensors_value[i] = wb_distance_sensor_get_value(sensors[i]);
```

```
    }

    go_forward(proposed_speeds);
    obstacle_back(sensors_value, speed, proposed_speeds);

    wander(proposed_speeds, &wander_remaining, &wander_direction, &wander_cooldown);
    obstacle_back(sensors_value, speed, proposed_speeds);

    avoid_right(sensors_value, proposed_speeds);
    obstacle_back(sensors_value, speed, proposed_speeds);

    speed[LEFT] = BOUND(speed[LEFT], -max_speed, max_speed);
    speed[RIGHT] = BOUND(speed[RIGHT], -max_speed, max_speed);

    wb_motor_set_velocity(left_motor, speed[LEFT]);
    wb_motor_set_velocity(right_motor, speed[RIGHT]);
  }
}
```

Code complet Architecture Subsomption

Vidéo:

Téléchargement de la vidéo

Stratégie 2: Vote

Notre première approche à consisté en une architecture par vote. Celle-ci possède 3 comportement: - avancer (`go_foward`) - éviter par la droite (`avoid_right`) - s'arrêter aux obstacles (`obstacle_stop`) Le comportement obstacle stop peut appliquer des vétos sur certains votes (par exemple les vitesses positives sur le moteur droit) pour éviter les collisions avec les obstacles.

Code:

```
void vote_architecture() {
  double speed[2];
  double proposed_speeds[2];

  while (wb_robot_step(time_step) != -1) {
    double sensors_value[SENSOR_NUMBER];
    double veto_above[2] = {INFINITY, INFINITY};
    double veto_below[2] = {-INFINITY, -INFINITY};
    for (int i = 0; i < SENSOR_NUMBER; i++) {
      sensors_value[i] = wb_distance_sensor_get_value(sensors[i]);
    }

    go_forward(proposed_speeds);
```

```
    speed[LEFT] = proposed_speeds[LEFT];
    speed[RIGHT] = proposed_speeds[RIGHT];

    avoid_right(sensors_value, proposed_speeds);
    speed[LEFT] += proposed_speeds[LEFT];
    speed[RIGHT] += proposed_speeds[RIGHT];

    obstacle_stop(sensors_value, veto_below, veto_above);
    apply_vetos(speed, veto_below, veto_above);

    speed[LEFT] = BOUND(speed[LEFT], -max_speed, max_speed);
    speed[RIGHT] = BOUND(speed[RIGHT], -max_speed, max_speed);

    wb_motor_set_velocity(left_motor, speed[LEFT]);
    wb_motor_set_velocity(right_motor, speed[RIGHT]);
  }
}
```

Code complet Architecture Vote

Vidéo:

Téléchargement de la vidéo