Esercitazione 4 - ROBOTICA

Esercizio 1 - Pratica

Implementare un pianificatore globale utilizzando l'algoritmo A*.
Il robot dovrebbe costruire la mappa in maniera "randomica" esplorando l'ambiente.
Usare i grafi di visibilità (o deterministici) per calcolare il percorso migliore.
Miscelare IA e robotica per evitare di rimanere intrappolati in minimi locali.

Ragionamento:

Per lo svolgimento dell'esercizio, io e la mia collega abbiamo pensato che anziché impostare un movimento casuale per il rilevamento della mappa, potevamo controllarlo noi tramite l'utilizzo della seconda esercitazione (teleop_stage) in maniera tale da avere una pianificazione migliore.
La mappa sarà campionata a 3 (ovvero dato che le dimensioni dello stage sono 59x54, un campionamento a 3 significa avere una matrice di dimensione 177x162).
Per la generazione dei grafi di visibilità, noi ci calcoliamo dalla matrice mappa una nuova matrice, con le stesse dimensioni, ma che sarà dei vertici generati dalla mappa stessa.
Verranno generati 2 file .pgm per avere una migliore chiarezza su quello che saranno le 2 matrici (mappa e mappa_vertici).
Il movimento del robot sarà controllato tramite potenziali.

Andremo ora ad analizzare il codice più in dettaglio.

Analisi del Codice:

Abbiamo creato una struttura nodo per tenere traccia dei vertici sui quali passerà il robot. Tali vertici non saranno altro che i "fake_goal" intermedi utili al fine di tracciare il percorso minimo. Le varie variabili dichiarate dentro la struttura tengono traccia delle coordinate x e y, della distanza da un nodo al successivo, della distanza euristica da un nodo al goal, e avremo dei riferimenti al successivo nodo e al genitore al fine di ricostruire il percorso. La struttura nodo sarà la seguente:

```
22 struct node {
23    double x;
24    double y;
25    double g;
26    double h;
27    struct node* parent;
28    struct node* next;
29 };
30
31 typedef struct node* p_node;
```

Le funzioni fondamentali al fine del funzionamento dell'algoritmo sono la funzione raggiungibile, la quale si occupa di verificare se dati due vertici, esiste un cammino libero tra di essi, e la funzione AStar la quale si occupa di sviluppare l'algoritmo del cammino minimo.
Il codice si avvale anche di funzioni ausiliarie svolgenti operazioni su liste.
La funzione raggiungibile sarà la seguente:

```
342 bool raggiungibile(int x1, int y1, int x2, int y2) {
343    int temp_x1;
344    int temp_x2;
345    int temp_y1;
346    int temp_y2;
```

```
347
348
349     // controllo che serve per far si che la variabile temp_x1
350     // contenga il valore di x maggiore.
351     if (x1 > x2) {
352        temp_x1 = x1;
353        temp_x2 = x2;
354        temp_y1 = y1;
355        temp_y2 = y2;
356     } else {
357        temp_x1 = x2;
358        temp_x2 = x1;
359        temp_y1 = y2;
360        temp_y2 = y1;
361     }
362
363     while(1) {
364        if (temp_x1 == temp_x2) {
365           if(temp_y1 > temp_y2) {
366              if (mappa[temp_x1][temp_y1-1] || mappa[temp_x2][temp_y2+1]) {
367                 return false;
368              } else if((temp_y1-temp_y2) <= 2) {
369                 return true;
370              } else {
371                 temp_y1--;
372                 temp_y2++;
373              }
374           } else {
375              if (mappa[temp_x1][temp_y1+1] || mappa[temp_x2][temp_y2-1]){
376                 return false;
377              } else if((temp_y2-temp_y1) <= 2) {
378                 return true;
379              } else {
380                 temp_y1++;
381                 temp_y2--;
382              }
383           }
384        } else if (temp_y1 == temp_y2) {
385           if (mappa[temp_x1-1][temp_y1] || mappa[temp_x2+1][temp_y2]) {
386              return false;
387           } else if ((temp_x1-temp_x2) <= 2) {
388              return true;
389           } else {
390              temp_x1--;
391              temp_x2++;
392           }
393        } else {
394           if(temp_y1 > temp_y2 ){
395              if (mappa[temp_x1][temp_y1-1] || mappa[temp_x1-1][temp_y1]
396                          || mappa[temp_x1-1][temp_y1-1]) {
397                 return false;
398               }
399              if (mappa[temp_x2][temp_y2+1] || mappa[temp_x2+1][temp_y2]
400                          || mappa[temp_x2+1][temp_y2+1]) {
401                 return false;
402               }
403              if ((temp_x1-temp_x2  <= 2) && (temp_y1-temp_y2 <= 2)) {
404                 return true;
405              } else if (temp_x1-temp_x2<= 2) {
406                 temp_y1--;
407                 temp_y2++;
408              } else if (temp_y1-temp_y2 <= 2) {
409                 temp_x1--;
410                 temp_x2++;
411              } else {
412                 temp_x1--;
413                 temp_y1--;
414                 temp_x2++;
415                 temp_y2++;
```

```
            }
         } else {
            if (mappa[temp_x1][temp_y1+1] || mappa[temp_x1-1][temp_y1]
                     || mappa[temp_x1-1][temp_y1+1]) {
               return false;
            }
            if (mappa[temp_x2][temp_y2-1] || mappa[temp_x2+1][temp_y2]
                     || mappa[temp_x2+1][temp_y2-1]) {
               return false;
            }
            if ((temp_x1-temp_x2  <= 2) && (temp_y2-temp_y1 <= 2)) {
               return true;
            } else if (temp_x1-temp_x2<= 2) {
               temp_y1++;
               temp_y2--;
            } else if (temp_y2-temp_y1 <= 2) {
               temp_x1--;
               temp_x2++;
            } else {
               temp_x1--;
               temp_y1++;
               temp_x2++;
               temp_y2--;
             }
         }
      }
   }
   return false;
}
```

La funzione AStar sarà:

```
p_node AStar() {
   ready_astar = false;
   // creazione delle liste CLOSE e OPEN
   p_node CLOSE = NULL;
   p_node OPEN = NULL;
   // Inseriamo nella lista OPEN per prima la posizione del robot attraverso
   // la funzione ausiliaria insertNode, successivi inserimenti saranno
   // ordinati secondo valori di f minori.
   OPEN = insertNode(OPEN, (int)round(fabs(3*robot_pose_x)),
        (int)round(fabs(3*robot_pose_y)), 0, NULL);

   while (OPEN != NULL) {
   // deleteHeadVertex e' una funzione ausiliaria la quale si occupa di estrarre
   // valori dalla testa di una lista.
      p_node tmp = deleteHeadVertex(&OPEN);
      // discretize e' una funzione ausiliaria che si occupa di discretizzare
      // i valori.
      if(raggiungibile(discretize(goal_x), discretize(goal_y), tmp->x, tmp->y)) {
         return tmp;
      }
      CLOSE = insertNode(CLOSE, tmp->x, tmp->y, tmp->g, tmp->parent);
      for (int i = 0; i < ROW - 1; i++) {
         for (int j = 0; j < COLUMN - 1; j++) {
            // ricordiamo che mappa_vertici sara' una matrice di booleani.
            if (mappa_vertici[i][j]){
               if (raggiungibile(i, j, tmp->x, tmp->y)){
                  // la findNode e' una funzione ausiliaria la quale si occupa di
                  // ricercare un nodo dentro una lista
                  if (findNode(CLOSE, i, j)){
                  } else if (findNode(OPEN, i, j)){
                  } else {
                     if (tmp->parent) {
                        // il calcolo in A* della distanza sara'  f=g+h
                        // dove g e' la distanza effettiva percorsa per
```

```
479                         // arrivare in un dato vertice, h e' invece l'euristica
480                         // per arrivare al goal.
481
482                         double temp_distance = tmp->parent->g +
483                                 distance(tmp->x, tmp->y, i, j);
484                         OPEN = insertNode(OPEN, i, j, temp_distance, tmp);
485                       } else {
486                         double temp_distance = distance((int)round(fabs(
487                          3*robot_pose_x)),(int)round(fabs(3*robot_pose_y)),i,j);
488                         OPEN = insertNode(OPEN, i, j, temp_distance, tmp);
489                       }
490                     }
491                   }
492                 }
493               }
494             }
495           }
496     // ritorna fallimento
497     return NULL;
498 }
```

GRUPPO BACK:
Antonino Buscetta (0610591)
Chiara Capobianco (0609919)