Dokumentácia – Plánovanie dráhy

Úloha

Úlohou plánovania dráhy je vygenerovať optimálnu trasu pre mobilný robot na základe existujúcej mapy prostredia. Na tento účel sa využije typ mapy z predchádzajúcej úlohy a použitý algoritmus na plánovanie dráhy je záplavový algoritmus.

Algoritmus

Algoritmus plánovania cesty pre robota, ktorý používa záplavový algoritmus a plánovanie cesty na základe 2D mriežky alebo mapy, ktorá predstavuje prostredie okolo robota. Nižšie sú zhrnuté podstatné body kódu:

- Čítanie mapy a predbežné spracovanie: Táto časť načítava mapu z textového súboru ("occGrid_ideal.txt"), kde každý znak predstavuje bunku v mape prostredia. Preskočí všetky medzery v súbore, ktoré možno použiť na oddelenie hodnôt buniek. Vykoná aj určité predbežné spracovanie na mape, rozšíri prekážky o jednu bunku vo všetkých ôsmich smeroch okolo každej prekážky.

```
584     std::ifstream myfile("C:/Users/pao/Desktop/RMR/RMR2023/uloha4/occGrid_ideal.txt");
585
586     while (std::getline(myfile, eachrow))
587     {
588         std::vector<char> row;
589
590         for (char &x : eachrow)
591         {
592             if (x != ' ')row.push_back(x);
593         }
594
595         mapa.push_back(row);
596     }
597
598     for(int k = 0; k < 6; k++){
599         for (int i = 0; i < mapa.size(); i++)
600         {
601             for (int j = 0; j < mapa[i].size(); j++)
602             {
603                 if(mapa[i][j] == '1'){
604
605                     for (int k = 0; k < 8; k++)
606                     {
607                         int indexX = i + offset[k][0];
608                         int indexY = j + offset[k][1];
609                         if(mapa[indexX][indexY] != '1')
610                             mapa[indexX][indexY] = '2';
611                     }
612                 }
613             }
614         }
615
616         for (int i = 0; i < mapa.size(); i++)
617         {
618             for (int j = 0; j < mapa[i].size(); j++)
619             {
620                 if(mapa[i][j] == '2')mapa[i][j] = '1';
621             }
622         }
623     }
```

Obr. 1

- Algoritmus plánovania trasy: Najprv identifikuje pozíciu cieľa označenú na mape „G“. Potom označí všetky bunky okolo cieľa ako „nenavštívené“ (reprezentované „u“) a zaradí ich do frontu (nenavštívené).

```
134     if(test1){
135         printf("IDEM0\n");
136         int algMapa[300][300] = {0};
137
138
139         int idx[2] = {0,0};
140         Index nodeIdx;
141         Alg alg;
142
143
144         for (int i = 0; i < mapa.size(); i++)
145         {
146             for (int j = 0; j < mapa[i].size(); j++)
147             {
148                 if(mapa[i][j] == 'G'){
149                     idx[0] = i;
150                     idx[1] = j;
151                 }
152             }
153         }
154         alg.current.x = idx[0];
155         alg.current.y = idx[1];
156
```

Obr. 2

- Potom spustí slučku, v ktorej vyberie prvý uzol z nenavštíveného uzla, skontroluje všetkých jeho susedov a vykoná niekoľko akcií na základe stavu suseda. Ak je sused prázdna bunka ('0'), označí ju ako 'nenavštívené' a pridá ju do zoznamu nenavštívených. Ak má susedná bunka hodnotu väčšiu ako 1 a nie je cieľom, považuje ju za výpočet minimálnej hodnoty. Ak je cieľom sused („G“), nastaví minimum na 1.

- Po kontrole všetkých susedov označí aktuálnu bunku ako „navštívenú“ (reprezentovanú „v“) a priradí jej hodnotu o jednu väčšiu, než je minimálna nájdená hodnota. Toto pokračuje, kým sa nenavštívený rad vyprázdni, čo znamená, že neboli navštívené všetky dostupné bunky.

```
157         for (int k = 0; k < 8; k++)
158         {
159             int indexX = alg.current.x + offset[k][0];
160             int indexY = alg.current.y + offset[k][1];
161             mapa[indexX][indexY] = 'u';
162             nodeIdx.x = indexX;
163             nodeIdx.y = indexY;
164             alg.unvisited.push(nodeIdx);
165         }
166         int test = 0;
167         while(!alg.unvisited.empty()){
168             nodeIdx = alg.unvisited.front();
169             alg.unvisited.pop();
170             alg.current.x = nodeIdx.x;
171             alg.current.y = nodeIdx.y;
172             int min = 999999;
173 
174             for (int k = 0; k < 8; k++)
175             {
176                 int indexX = alg.current.x + offset[k][0];
177                 int indexY = alg.current.y + offset[k][1];
178                 if(mapa[indexX][indexY] == '0'){
179                     mapa[indexX][indexY] = 'u';
180                     nodeIdx.x = indexX;
181                     nodeIdx.y = indexY;
182                     alg.unvisited.push(nodeIdx);
183                 }else if(algMapa[indexX][indexY] > 1 && mapa[indexX][indexY] != 'G'){
184                     if(algMapa[indexX][indexY] < min){
185                         min = algMapa[indexX][indexY];
186                     }
187                 }else if(mapa[indexX][indexY] == 'G'){
188                     min = 1;
189                 }
190             }
191             mapa[alg.current.x][alg.current.y] = 'v';
192             algMapa[alg.current.x][alg.current.y] = min+1;
193             ///std::cout<<mapa[alg.current.x][alg.current.y]<<std::endl;
194 
195             //std::cout<<test<<std::endl;
196         }
197 
```

Obr. 3

- Trasa cesty: V tomto bloku kód sleduje najkratšiu cestu zo štartovacej pozície (100, 100) k cieľu tak, že sa vždy presunie do susednej bunky s najnižšou hodnotou v algMapa. Zastaví sa, keď je aktuálna hodnota bunky menšia ako 3, čo môže znamenať, že dosiahla cieľ alebo oblasť blízko cieľa.

```
198         Index smallestIndex;
199         int min = algMapa[100][100];
200         alg.current.x = 100;
201         alg.current.y = 100;
202         bool end = false;
203 
204         while(!end){
205 
206             for (int k = 0; k < 8; k++)
207             {
208                 int indexX = alg.current.x + offset[k][0];
209                 int indexY = alg.current.y + offset[k][1];
210                 if(indexX < 300 && indexX >= 0 && indexY < 300 && indexY >= 0){
211                     if(algMapa[indexX][indexY]<min && algMapa[indexX][indexY] > 0){
212 
213                         min = algMapa[indexX][indexY];
214                         smallestIndex.x = indexX;
215                         smallestIndex.y = indexY;
216 
217                     }
218 
219                 }
220 
221             }
222             alg.current.x = smallestIndex.x;
223             alg.current.y = smallestIndex.y;
224             path.push_back(smallestIndex);
225             ///cout << algMapa[smallestIndex.x][smallestIndex.y] << endl;
226             if(algMapa[smallestIndex.x][smallestIndex.y] < 3) end = true;
227 
228         }
```

Obr. 4

- Vyhladzovanie a konverzia cesty: Tiež kontroluje možné skratky, kde sa cesta môže priamo pohybovať z jedného bodu do druhého bez toho, aby narazila na prekážku.

```
229     Index prevIdx = path.front();
230     Index prevSmer;
231
232     prevSmer.x = 0;
233     prevSmer.y = 0;
234
235     for(int k = 0; k < path.size();k++){
236
237         int dx = path[k].x - prevIdx.x;
238         int dy = path[k].y - prevIdx.y;
239         cout << "dx: " << dx << " x : " << prevSmer.x << " | dy: " << dy << " y: " << prevSmer.y;
240         if(prevSmer.x != dx || prevSmer.y != dy){
241            Index addThis;
242            addThis.x = prevIdx.x;
243            addThis.y = prevIdx.y;
244            if(pathPoints.size() > 1){
245                int pk = pathPoints.size()-1;
246                Index test;
247                test.x = pathPoints[pk].x - addThis.x;
248                test.y = pathPoints[pk].y - addThis.y;
249                if(test.x < 2 && test.x > (-2) && test.y < 2 && test.y > (-2)){
250                    if(mapa[pathPoints[pk].x][(pathPoints[pk].y + 1)] == 'v' && mapa[addThis.x][(addThis.y-1)] == '1'){
251                        addThis.x = pathPoints[pk].x;
252                        addThis.y = (pathPoints[pk].y + 1);
253                    }else if(mapa[pathPoints[pk].x][(pathPoints[pk].y - 1)] == 'v' && mapa[addThis.x][(addThis.y+1)] == '1'){
254                        addThis.x = pathPoints[pk].x;
255                        addThis.y = (pathPoints[pk].y - 1);
256                    }else if(mapa[pathPoints[pk].x+1][(pathPoints[pk].y)] == 'v' && mapa[addThis.x-1][addThis.y] == '1'){
257                        addThis.x = (pathPoints[pk].x+1);
258                        addThis.y = pathPoints[pk].y;
259                    }else if(mapa[(pathPoints[pk].x-1)][(pathPoints[pk].y)] == 'v' && mapa[(addThis.x+1)][addThis.y] == '1'){
260                        addThis.x = (pathPoints[pk].x-1);
261                        addThis.y = pathPoints[pk].y;
262                    }
263                    pathPoints.erase(pathPoints.begin()+pk);
264
265                }
266
267            }
268            pathPoints.push_back(addThis);
269
270            cout << " => TARGET";
271        }
272        cout << endl;
273        prevIdx.x = path[k].x;
274        prevIdx.y = path[k].y;
275        prevSmer.x = dx;
276        prevSmer.y = dy;
277
278    }
```

Obr. 5

- Potom zmení mierku cesty podľa faktora (veľkosť bunky) a uloží upravenú cestu do dvoch frontov (qyr, qxr).

```
279         Index addThis;
280         addThis.x = idx[0];
281         addThis.y = idx[1];
282         pathPoints.push_back(addThis);
283         pathPoints.erase(pathPoints.begin());
284
285         float cellSize = 0.05;
286         Index start;
287         start.x = 102;
288         start.y = 100;
289         for(int i = 0; i < pathPoints.size();i++){
290            double testX = pathPoints[i].x*cellSize-start.x*cellSize;
291            double testY = pathPoints[i].y*cellSize-start.y*cellSize;
292            cout << testX << " | " << testY << endl;
293            qyr.push(testX);
294            qxr.push(testY);
295
296
297         }
```

Obr. 6

- Uloženie výsledkov: Posledná časť zapíše dve verzie spracovanej mapy do dvoch textových súborov. Prvá verzia označuje bunky pozdĺž vyhladenej cesty

písmenom „T“ a ostatné bunky na pôvodnej ceste písmenom „X“. Druhá verzia nahradí všetky bunky s hodnotou väčšou ako 2 v algMapa za „U“.

- Znaky 'T', 'X' a 'U' možno použiť na lepšiu vizualizáciu cesty robota a spracovanej mapy.

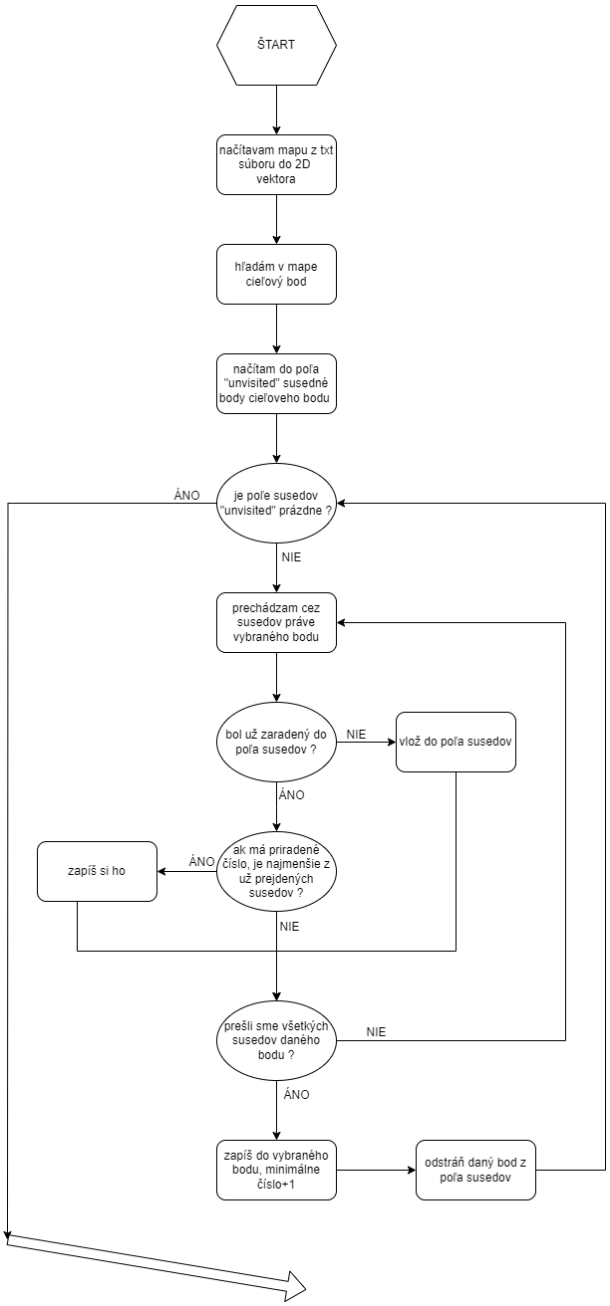
```
303         ofstream occGridALG("C:/Users/pao/Desktop/RMR/RMR2023/uloha4/occGridALG.txt");
304         ofstream occGridALG2("C:/Users/pao/Desktop/RMR/RMR2023/uloha4/occGridALG2.txt");
305         printf("Zapisujem do mapy");
306
307         for (int i = 0; i < mapa.size(); i++) {
308             for (int j = 0; j < mapa[0].size(); j++) {
309
310
311                 bool pathWriten = false;
312
313                 for(int k = 0; k < pathPoints.size();k++){
314                     if(pathPoints[k].x == i && pathPoints[k].y == j){
315                         occGridALG << 'T';
316                         pathWriten = true;
317                         ///path.erase(path.begin() + k);
318                     }
319                     if(pathWriten){
320                         break;
321                     }
322
323                 }
324
325
326                 for(int k = 0; k < path.size();k++){
327                     if(pathWriten){
328                         break;
329                     }
330                     if(path[k].x == i && path[k].y == j){
331                         occGridALG << 'X';
332                         pathWriten = true;
333                         ///path.erase(path.begin() + k);
334                     }
335
336                 }
337
338
339                 if(!pathWriten) occGridALG << mapa[i][j];
340
341                 if(algMapa[i][j] > 2){
342
343                     occGridALG2 << 'U';
344                 }else occGridALG2 << algMapa[i][j];
345
346                 ///std::cout << i << ";" << j << " = " << algMapa[i][j] << std::endl;
347
348
349
350             }
351             occGridALG << endl;
352             occGridALG2 << endl;
353         }
354         occGridALG.close();
355         occGridALG2.close();
356
357         test1=false;
```

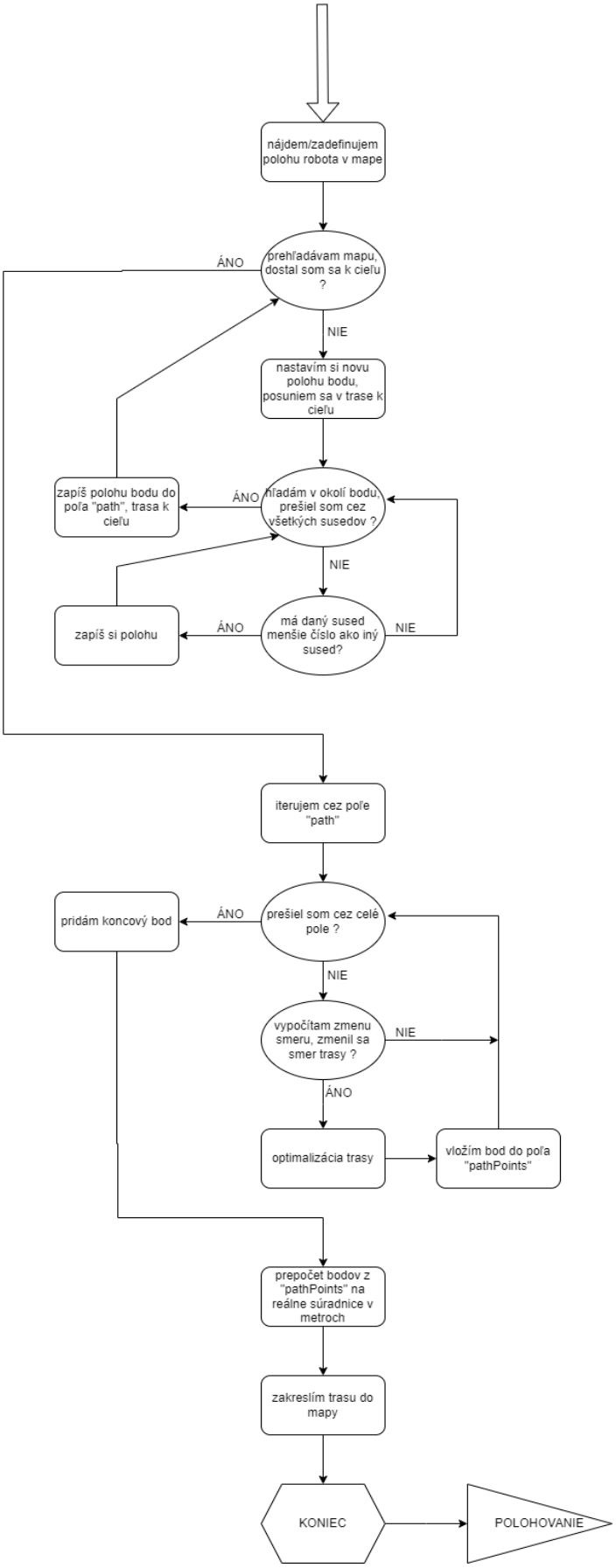
Obr. 7

- Stručne povedané, tento kód načítava mapu, predspracuje ju rozšírením prekážok, vypočíta pole vzdialenosti od cieľa pomocou algoritmu zaplavenia,

sleduje najkratšiu cestu z počiatočnej pozície k cieľu, vyhladzuje cestu, mení mierku cesty, a výsledky zapíše do súborov.

Diagram Algoritmu





Obr. 8 Diagram algoritmu