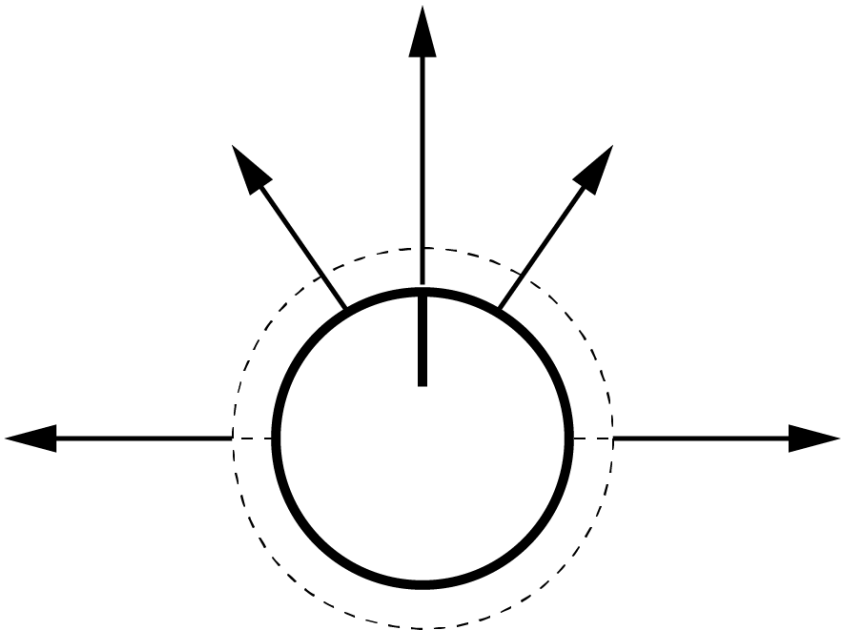
Dokumentácia – Navigácia

Úloha

Navigácia mobilného robota zabezpečuje bezkolízny prechod robota prostredím. Na svoju činnosť využíva aktuálne údaje zo snímačov a nepotrebuje poznať prostredie vopred (mapu). K najjednoduchším algoritmom navigácie patria hmyzie algoritmy (bug algorithm), z toho len niektoré na svoje fungovanie využívajú laserový diaľkomer.

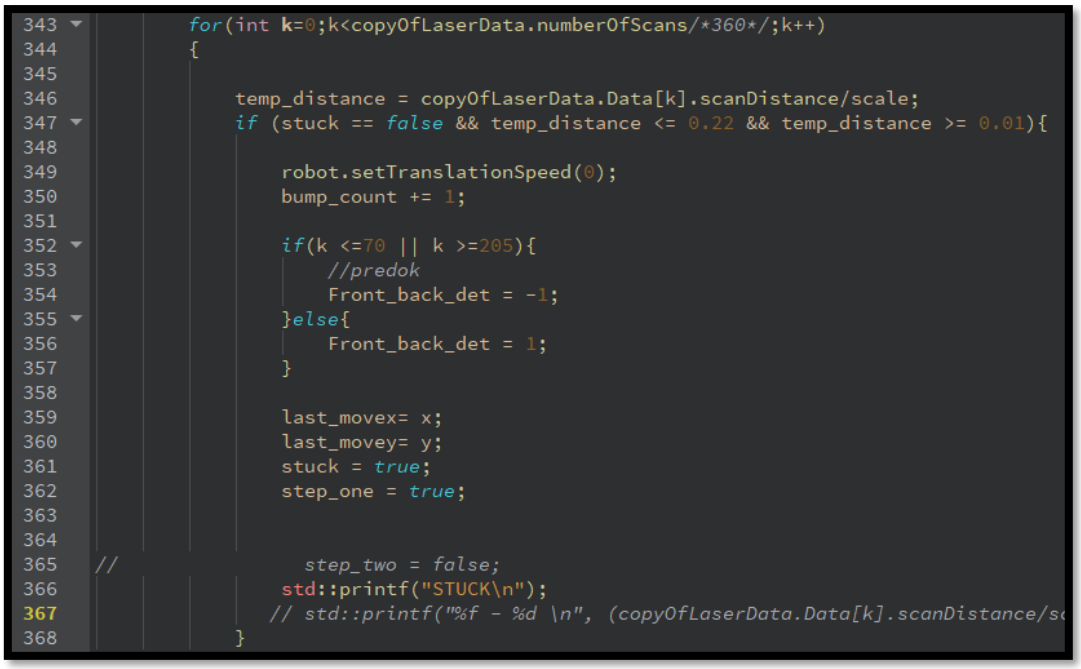


Obr. 1

Algoritmus

Tento algoritmus ovláda robota v neznámom prostredí pomocou LIDARu. Vykonáva sériu kontrol pri rozhodovaní, aby zistil najlepší postup v závislosti od údajov snímača a stavu robota. Pohyb robota je usporiadaný do niekoľkých rôznych sekcií:

- Prvá časť skenuje prostredie a pozerá sa na vzdialenosť k prekážkam vo všetkých smeroch (0 až 360 stupňov). Ak robot narazí na prekážku v určitej vzdialenosti (ako je naznačené na obrázku čiarkovaným kruhom), zastaví sa, zaznamená svoju aktuálnu polohu a zaznamená smer prekážky (vpredu alebo vzadu). Potom nastaví stuck na hodnotu true, čo znamená, že sa zasekol a potrebuje vykonať prvý krok vo svojej stratégii vyhýbania sa prekážkam.



```
343         for(int k=0;k<copyOfLaserData.numberOfScans/*360*/;k++)
344         {
345
346             temp_distance = copyOfLaserData.Data[k].scanDistance/scale;
347             if (stuck == false && temp_distance <= 0.22 && temp_distance >= 0.01){
348
349                 robot.setTranslationSpeed(0);
350                 bump_count += 1;
351
352                 if(k <=70 || k >=205){
353                     //predok
354                     Front_back_det = -1;
355                 }else{
356                     Front_back_det = 1;
357                 }
358
359                 last_movex= x;
360                 last_movey= y;
361                 stuck = true;
362                 step_one = true;
363
364
365 //                  step_two = false;
366                 std::printf("STUCK\n");
367                // std::printf("%f - %d \n", (copyOfLaserData.Data[k].scanDistance/s
368             }
```

Obr. 2

- Po dokončení počiatočného skenovania program skontroluje určité strategické smery (0, 68, 207, 28, 247 indexy v LIDARy ukázané na obrázku šipkami). Zaznamenáva vzdialenosti v týchto smeroch a tiež vypočítava a zaznamenáva premietané polohy X a Y na 2D mape, ak sa robot pohol týmto smerom.

```
374         if(k == 0 || k == 68|| k == 207  || k == 28 || k == 247){
375
376             if( k == 0){
377                 //front 0deg
378                 side_int=0;
379             }else if ( k == 68){
380                 //left 90deg
381                 side_int=1;
382
383             }else if( k == 207){
384                 //right271 deg
385                 side_int=2;
386             }else if(k == 28){
387                 //left 38deg
388                 side_int=3;
389             }else if (k == 247){
390                 //right38 deg
391                 side_int=4;
392             }
393
394             //distance
395             cloud[side_int][0]=copyOfLaserData.Data[k].scanDistance/scale; ///vzdialenost nahodne predelena 20 aby to nejako vyzeralo v okne,. zmen podla uvazenia
396             //x
397             cloud[side_int][1]=(rect.width()-(rect.width()/2+cloud[side_int][0]*2*sin((360.0-copyOfLaserData.Data[k].scanAngle)*3.14159/180.0))+rect.topLeft().x())/scale; //prepocet do obrazovky
398             //y
399             cloud[side_int][2]=(rect.height()-(rect.height()/2+cloud[side_int][0]*2*cos((360.0-copyOfLaserData.Data[k].scanAngle)*3.14159/180.0))+rect.topLeft().y())/scale;//prepocet do obrazovky
400             //std::printf("%f - %d \n", (((360.0-copyOfLaserData.Data[k].scanAngle)*3.14159/180.0) * 180/3.14159), k);
401             //std::printf("%f - %d \n", (copyOfLaserData.Data[k].scanDistance/scale), k);
402         }
403
404     }
```

Obr. 3

- Ďalšia časť kódu vypočíta vzdialenosť, ktorú robot prešiel od svojej poslednej zaznamenanej polohy, a skontroluje, či robot dosiahol svoj cieľ. Ak áno, robot sa zastaví. Ak je blízko cieľa, pohybuje sa smerom k nemu. Ak nie, skontroluje, či nie je zaseknutý, a spustí stratégiu vyhýbania sa prekážkam.

```
427 if(abs(calculate_Distance(finish_X,finish_Y,x,y)) <= 0.03){
428     robot.setTranslationSpeed(0);
429     robot.setRotationSpeed(0);
430     std::this_thread::sleep_for(std::chrono::milliseconds(500));
431     distance_to_finish = distance_to_finish*2;
432     std::printf("FINISH stop\n");
433 }
434 else if(abs(calculate_Distance(finish_X,finish_Y,x,y)) <= distance_to_finish){
435     robot.setRotationSpeed(0);
436     robot.setTranslationSpeed(0);
437     wall_follow = false;
438     Left_wall = false;
439     MoveRobot( x,  y, rads);
440
441     std::printf("FINISH move\n");
```

Obr. 4

- Ak sa zasekne, posunie sa o určitú vzdialenosť, otočí sa na mieste alebo sa obráti v závislosti od podmienok, potom zaznamená svoju novú polohu a snaží sa vrátiť k normálnemu chodu ak znovu narazí tak robot zopakuje vyhýbanie.

```
454     if(stuck){
455
456         if(distance_travelled >= 0.1 ){
457     //        std::printf("stuck 3\n");
458             robot.setTranslationSpeed(0);
459             robot.setRotationSpeed(1);
460             std::this_thread::sleep_for(std::chrono::milliseconds(500));
461
462
463             last_movex= x;
464             last_movey= y;
465
466             switch_go = true;
467             step_one = false;
468             stuck = false;
469             //v
470
471         }else if(step_one){
472
473             //robot.setTranslationSpeed(Front_back_det*speed);
474             robot.setTranslationSpeed(Front_back_det*300);
475     //        std::printf("stuck 1\n");
476         }
```

Obr. 5

- Ak nie je zaseknutý robot sa začne pohybovať dopredu.

- Ak pri zaseknutí robot narazí viac krát spustí sa operácia bump_count, robot sa zastaví a bude sa otáčať, kým vzdialenosť prednej časti neprekročí určitú hranicu. Tento program uisťuje aby sa pri otáčaní na rohu robot nezatúlal preč od steny

```
480     }else if(switch_go){
481 
482 //        if(((cloud[0][0] <= 0.10 && cloud[0][0] >= 0.01) || (cloud[1][0] <= 0.10 && cl
483 //            ramp(1, false);
484 //            distance_count = false;
485 //            switch_go = false;
486 //        }
487 //        else
488     if(distance_travelled >= 0.2 && distance_count){
489             robot.setTranslationSpeed(0);
490                 std::printf("stop move\n");
491                 last_movex= x;
492                 last_movey= y;
493             distance_count = false;
494             switch_go = false;
495             bump_count = 0;
496         }else if(distance_count){
497 
498                 robot.setTranslationSpeed(300);
499     }
500     }else if(switch_go == false){
501 
502      if(bump_count >=1){
503 
504          robot.setTranslationSpeed(0);
505          robot.setRotationSpeed(-1);
506 
507          //if(copyOfLaserData.Data[0].scanDistance/scale >= 0.6 && copyOfLaserData.Data[
508          temp_distance = copyOfLaserData.Data[0].scanDistance/scale;
509              while(temp_distance >= 0.4 && temp_distance >= 0.01){
510                  std::this_thread::sleep_for(std::chrono::milliseconds(100));
511                  //std::printf("%f  \n", (copyOfLaserData.Data[1].scanDistance/scale));
512                  robot.setRotationSpeed(-1);
513                  std::printf("loop3 \n");
514                  temp_distance = copyOfLaserData.Data[0].scanDistance/scale;
515              }
516         robot.setRotationSpeed(0);
517         std::printf("bump \n");
518          //}
519         bump_count =0;
```

Obr. 6

- Ak je prekážka v určitom rozsahu vzdialenosti vpredu, vľavo alebo vpravo, zastaví sa, otáča sa, kým sa predná časť neuvoľní, potom sa pohne dopredu a zapne režim wall_follow.

```
}else if (((cloud[0][0] <= 0.40 && cloud[0][0] >= 0.01) || (cloud[1][0] <= 0.40 && cloud[1][0] >= 0.01) || (cloud[2][0] <= 0.40 && cloud[2][0] >= 0.01))){

                robot.setTranslationSpeed(0);


                robot.setRotationSpeed(1);


                //std::printf("%f \n", (copyOfLaserData.Data[0].scanDistance/scale));
                    temp_distance = copyOfLaserData.Data[0].scanDistance/scale;
                    if(temp_distance <= 1 && temp_distance >= 0.01){

                        while((temp_distance <= 1 && temp_distance >= 0.01) || (copyOfLaserData.Data[28].scanDistance/scale <= side_distance
                                                                                && copyOfLaserData.Data[28].scanDistance/scale >= 0.01)|| (copyOfLaserData.Data[247].scanDistance/scale
                                                                                <= side_distance && copyOfLaserData.Data[247].scanDistance/scale >= 0.01)){
                            std::this_thread::sleep_for(std::chrono::milliseconds(10));
//                            std::printf("%f  \n", (copyOfLaserData.Data[1].scanDistance/scale));
                            //std::printf("here x1 \n");
                            robot.setRotationSpeed(1);
                            std::printf("loop2 \n");
                            temp_distance = copyOfLaserData.Data[0].scanDistance/scale;
                        }

                    }

                    robot.setRotationSpeed(0);

                std::printf("here1 \n");
                //robot.setTranslationSpeed(speed);
                if(distance_count == false){
                    last_movex= x;
                    last_movey= y;
                    distance_count = true;

                }
                switch_go = true;
                wall_follow = true;
                bump_count =0;
```

Obr. 7

- Ak je aktívny režim wall_follow a v určitej vzdialenosti vľavo alebo vpravo od robota nie sú žiadne prekážky, zastaví sa, otáča sa, kým nie je predná časť voľná na určitú vzdialenosť, a potom sa pohne dopredu. V respektíve sa robot udržiava pri stene na základe dát z LIDARa na obrázku a podľa toho ako ďaleko je od steny sa buď natáča smerom od steny alebo ku stene.

- Ak neplatí žiadna z predchádzajúcich podmienok, presunie sa smerom k cieľovému cieľu. Táto funkcia slúži v prípade ak sa robot stratí alebo s ním niekto pohne

```
560             }else if((wall_follow) && ((cloud[1][0] >= 0.6) || (cloud[2][0] >= 0.6) || (copyOfLaserData.Data[140].scanDistance/scale >= 0.6))
561                 //std::printf("here2 \n");
562                 robot.setTranslationSpeed(0);
563 
564 
565                     robot.setRotationSpeed(-1);
566 
567                     //if(copyOfLaserData.Data[0].scanDistance/scale >= 0.6 && copyOfLaserData.Data[0].scanDistance/scale != 0){
568                     temp_distance = copyOfLaserData.Data[0].scanDistance/scale;
569                         while(temp_distance >= 0.8){
570                             std::this_thread::sleep_for(std::chrono::milliseconds(10));
571                             //std::printf("%f  \n", (copyOfLaserData.Data[1].scanDistance/scale));
572                             std::printf("loop1 \n");
573                             robot.setRotationSpeed(-1);
574                             temp_distance = copyOfLaserData.Data[0].scanDistance/scale;
575                         }
576 
577                     //}
578 
579 
580                 std::printf("here3 \n");
581                 robot.setRotationSpeed(0);
582 
583                 if(distance_count == false){
584                     last_movex= x;
585                     last_movey= y;
586                     distance_count = true;
587                 }
588                 //robot.setTranslationSpeed(speed);
589                 switch_go = true;
590 
591 
592             }else{
593                 std::printf("here4 \n");
594                 wall_follow = false;
595                 Left_wall = false;
596                 MoveRobot( x,  y, rads);
597 
598             }
599 
```

Obr. 8