Univerzita obrany
Fakulta vojenského leadershipu
Katedra taktiky

Softwarová dokumentácia

Softwarová knižnica pre optimalizáciu pohybu prostriedkov pri vyhľadávaní objektov

Software library for optimization of vehicle movement when searching for objects

Autori: RNDr. Miroslav KULICH, Ph.D.

pplk. doc. Ing. Petr STODOLA, Ph.D.

Brno 2018

Obsah

1. Popis knižnice

Predkladaná softwarová knižnica je výsledkom riešenia dlhodobého zámeru rozvoja organizácie PASVŘ II (Pokročilý automatizovaný systém velení a řízení II). Cieľom tohoto riešenia bolo vyvinúť algoritmus pre optimalizáciu pohybu robota pri vyhľadávaní stacionárneho objektu v známom prostredí, v minimálnom čase, pri čom poloha objektu nie je vopred známa.

Medzi základné funkcie, ktoré knižnica ponúka, patria:

- Správa polohy uzlov v mape.
- Správa polohy a kinematických veličín robotov.
- Výpočet optimálnej trasy pohybu robotov pri vyhľadávaní objektov.
- Správa trás robotov.

Riešenie je formulované ako problém cestujúceho dodávateľa TDP (Traveling Delivery Man), za predpokladu, že graf reprezentuje prostredie a pravdepodobnosť výskytu objektu je pre všetky vrcholy grafu rovnaká.

Súčasťou tejto dokumentácie je predovšetkým popis jednotlivých služieb a funkcií, ktoré knižnica ponúka.

2. Technické požiadavky

Softwarovú knižnicu je možné využívať na počítačoch vybavených operačným systémom Windows XP, alebo vyššími verziami. Knižnica je vytvorená v programovacom jazyku C++ s využitím princípov objektovo orientovaného programovania pod prekladačom Microsoft Visual Studio.

Knižnica sa nazýva GPSSolver a je distribuovaná v podobe dynamickej knižnice DLL vo verziách x86 a x64. Knižnica pozostáva z nasledujúcich súborov:

- **gps_solver.h**: deklarácia dátových štruktúr a triedy GPSSolver.
- **mgps.dll**: programová realizácia knižnice GPSSolver.
- **mgps.lib**: pomocný súbor knižnice GPSSolver pre linkovanie funkcií knižnice.

3. Popis funkcií a služieb knižnice

V tejto kapitole nasleduje prehľad jednotlivých funkcií knižnice *GPSSolver* vrátane stručného popisu.

Kapitola je rozdelená do tematických celkov:

- Správa polohy uzlov.
- Správa robotov.
- Správa trasy.
- Trieda GPSSolver.
- Výpočet optimálnej trasy.

3.1 Správa polohy uzlov

Správa polohy slúži k výmene informácií o polohe uzlov.

Vstupnými parametrami sú zemepisné súradnice uzlov x a y.

Pre správu polohy uzlu a pravdepodobnosti výskytu hľadaného objektu v uzle je nutné zaviezť dátovú štruktúru definície uzlov:

```
// Uzel
struct SNode {
    int id;
    double x,y;       // Poloha uzlu
    double prob;      // Pravdepodobnost vyskytu hledaneho objektu v uzlu
    double dist(SNode &n);
};
```

Pre vyvolanie zoznamu uzlov je možné použiť nasledujúcu dátovú štruktúru:

```
typedef std::vector<SNode> NodeVector;
```

3.2 Správa robotov

Správa robotov slúži pre získanie informácií o počte robotov a ich kinematických veličín, predovšetkým o priemernej rýchlosti robota a uhlovej rýchlosti pri zmene smeru.

Pomocou nasledujúcej dátovej štruktúry je možné získať index východzieho uzlu robota, priemernú rýchlosť robota v m/s, uhlovú rýchlosť otáčania robota a tým pádom aj čas potrebný na zmenu smeru v uzle, ktoré sú zásadné pre výpočet optimálnej trasy:

```
struct DllExport SRobot {
    int startNode;         // Index vychoziho uzlu robota
    double velocity;       // Prumerna rychlost robota v m/s
    double angularVel;     // Cas na zmenu smeru v uzlu
};
```

Následne je možné stanoviť vektor robotov pomocou dátovej štruktúry:

```
typedef std::vector<SRobot> RobotVector;
```

3.3 **Trieda GPSSolver**

Instancia triedy predstavuje základný stavebný prvok objektového programovania. Každý taký dátový objekt má svoje vlastné atribúty na základe vzoru triedy. Instancia je obvykle tvorená pomocou tzv. konstruktoru, čo je špeciálna metóda triedy, ktorá sa volá vo chvíli vytvárania instancie tejto triedy. Konstruktory inicializujú dátové členy instancie.

Keďže môže dochádzať k preťaženiu konstruktorov, je možné, aby jedna trieda obsahovala niekoľko konstruktorov s odlišnými parametrami a odlišnou funkcionalitou.

Instancia triedy *CGSPSolver* pozostáva z nasledujúcich dvoch konstruktorov:

```
CGSPSolver(NodeVector &nodes, RobotVector &robots);

CGSPSolver(NodeVector &nodes, Distances &distances, RobotVector &robots);
```

Prvý konstruktor umožňuje inicializáciu instancie triedy prostredníctvom vektoru uzlov a vektoru robotov. Vzdialenosti medzi uzlami sú potom definované Euklidovskou vzdialenosťou medzi všetkými dvojicami uzlov.
Druhý konstruktor umožňuje manuálne definovať vzdialenosti medzi uzlami. Takto je možné vytvoriť nemetrický, poprípade asymetrický graf. Vzdialenosti medzi uzlami sa definujú pomocou štruktúry *Distances*, čo je dvojrozmerný vektor vzdialenosti medzi všetkými dvojicami uzlov:

```
typedef  std::vector<std::vector<double>> Distances;
```

3.4 **Výpočet optimálnej trasy**

Výpočet trasy pozostáva z nájdenia riešenia blízkemu optimálnemu riešeniu.

Pre výpočet je použitá nasledujúca funkcia triedy *GPSSolver*, ktorá vychádza z vektoru uzlu a trasy robota a nájde riešenie postupovania robota medzi jednotlivými uzlami:

```
bool computeRoutes(AngleFunction angleFunc);
```

Funkcia vracia hodnoty typu BOOL, pokiaľ vráti hodnotu TRUE, výpočet prebehol v poriadku a bola nájdená optimálna trasa (trasa blízka optimálnej). Vstupom funkcie je odkaz na funkciu *AngleFunction*, ktorá slúži pre výpočet uhlu medzi trojicou uzlov. Prostredníctvom tejto funkcie je možné realizovať časové oneskorenie robota v závislosti na jeho uhlovej rýchlosti.
Časové oneskorenie je závislé na trojici uzlov: robot nachádzajúci sa v uzle sem dorazil zo smeru z predchádzajúceho uzlu na ceste a pokračuje do nasledujúceho uzlu na ceste. Potrebná zmena uhlu orientácie robota predstavuje oneskorenie. Vstupom funkcie AngleFunction je trojica uzlov, výstupom je uhol, o ktorý musí robot zmeniť smer. Definícia funkcie je nasledujúca:

```
typedef std::function<double(SNode &, SNode &, SNode &)> AngleFunction;
```

3.5 **Správa trasy**

Správa trasy slúži pre získanie trasy robota nájdenej funkcie *computeRoutes*, ktorá bola predstavená v kapitole 3.4. Trasa je vrátená funkciou:

```
bool getRoute(int robotId, Route &route);
```

Vstupný parameter *robotID* označuje index robota číslovaný od 0. Výstupný parameter *route* označuje trasu vybraného robota. Trasa jedného robota sa ukladá do dátovej štruktúry *Route*, čo je vektor uzlov, ktoré musí robot navštíviť v stanovenom poradí.

```
typedef std::vector<int> Route;
```

Trasy všetkých robotov je možné potom definovať na základe dátovej štruktúry:

```
typedef std::vector<Route> Routes;
```