Sprawozdanie z pracy przejściowej inżynierskiej

Mapowanie otoczenia robota mobilnego

Promotor: dr inż. Andrzej Chmielniak, ZTMiR, wydział MEiL

Autor: Wojciech Celej

1. Wstęp

Problem mapowania otoczenia robota jest współcześnie bardzo modnym i szybko rozwijającym się zagadnieniem. Świadczą o tym lata wydań publikacji ukazujących się na ten temat. Zagadnienie to, występuje w literaturze przedmiotu pod nazwą *SLAM (simultaneous localization and mapping)* – czyli jednoczesna lokalizacja robota i budowa mapy. Ponieważ oba te procesy obarczone są błędami, w efektywnym algorytmie zadania te powinny być rozpatrywane łącznie. Zadania te bazują na metodach gaussowskich, metodach optymalizacji bądź metodach gridowych (np. filtr cząsteczkowy). Z racji na złożoność tych problemów, w swojej pracy ograniczyłem się do problemu mapowania otoczenia (bez jednoczesnej lokalizacji robota w utworzonej przez siebie mapie).

Stworzona przez mnie mapa w literaturze nosi nazwę *Occupancy Grid Map*, co można przetłumaczyć jako mapę zajętości. Warto wspomnieć, że dwa inne stosowane modele map, to mapa wektorowa i mapa topologiczna. Ich istotną zaletą w porównaniu do mapy zajętości jest zdecydowanie mniejsze zapotrzebowanie na pamięć.

2. Occupancy Grid Map

Pierwszym krokiem jest dyskretyzacja fizycznego obszaru działania i mapowania na N komórek. Komórki te reprezentowane są przez dwuwymiarową tablicę liczb z zakresu od 0 do 1. Liczba ta dla konkretnej komórki wyraża prawdopodobieństwo zajętości danej komórki. Np. prawdopodobieństwo wynoszące 0,5 zakłada, że nie mamy informacji czy dana komórka jest zajęta, czy też nie. Zakładamy oczywiście, że otoczenie jest statyczne. Kolejnym ważnym założeniem jest niezależność komórek od siebie nawzajem. Jest to ważne przy budowie modelu matematycznego pozwalającego wyznaczyć prawdopodobieństwo zajętości danej komórki.

3. Realizacja naiwnego filtra Bayesa

Filtr jest de facto zastosowaniem prawa Bayesa dla prawdopodobieństwa warunkowego. Przy budowie mapy model ten zakłada, że prawdopodobieństwo zajętości danej komórki mapy nie zależy od otaczających jej komórek, a jedynie od historii pomiarów zebranych z czujnika laserowego. Algorytm ten (po odpowiednich przekształceniach, które znajdują się w pozycji *Learning Occupancy Grid Maps With Forward Sensor Models* str.4-6) możemy zapisać w postaci pseudokodu:

/ Initialization */*
for all grid cells $\langle x, y \rangle$ *do*
$l_{x,y} = \log p(m_{x,y}) - \log[1 - p(m_{x,y})]$
endfor

/ Grid calculation using log-odds */*
for all time steps t *from 1 to* T *do*
for all grid cells $\langle x, y \rangle$ *in the perceptual range of* z_t *do*
$l_{x,y} = l_{x,y} + \log p(m_{x,y} \mid z_t) - \log[1 - p(m_{x,y} \mid z_t)] - \log p(m_{x,y}) + \log[1 - p(m_{x,y})]$
endfor
endfor

/ Recovery of occupancy probabilities */*
for all grid cells $\langle x, y \rangle$ *do*
$p(m_{x,y} \mid z_1, \ldots, z_T) = 1 - \frac{1}{e^{l_{x,y}}}$
endfor

Gdzie:

$p(m_{x,y})$ – prawdopodobieństwo zajętości danej komórki a priori
$p(m_{x,y}|z_t)$ – prawdopodobieństwo zajętości danej komórki dla konkretnego pomiaru
$p(m_{x,y}|z_1, \ldots, z_T)$ – prawdopodobieństwo zajętości danej komórki a posteriori

Algorytm ten został przeze mnie zaimplementowany przeze mnie jako metoda *inversebayes* w klasie *ArMapa*. Istotnym zagadnieniem jest dobór prawdopodobieństwa $p(m_{x,y}|z_t)$, które w swoim programie przyjąłem odpowiednio:

- jeżeli odczyt z czujnika wskazuje, że komórka jest zajęta jako 0,8
- jeżeli odczyt z czujnika wskazuje, że komórka jest wolna jako 0,2

4. Klasa *ArMapa* i jej metody

Jest to główna klasa programu odpowiadającego za mapowanie. Są w niej zawarte metody odpowiedzialne za realizację algorytmu mapowania. Omówię krótko 4 główne metody w niej występujące:

- Metoda *BresenhamLine* – pobiera aktualną pozycję robota i pozycję przeszkody, a następnie wyznacza zdyskretyzowaną linię prostą pomiędzy nimi. Punkty tej linii możemy oznaczyć jako komórki wolne.
- Metoda *myownAlgorithm* – jest efektywniejszą metodą przeszukiwania mapy, po to aby odnaleźć komórki mapy, które na pewno w danym pomiarze zostały uznane przez czujnik laserowy robota jako wolne. Przeszukuje ona obszar będący prostokątem ograniczonym przez komókę mapy w której aktualnie zajmuje robot oraz przez komórkę, w której została wykryta przeszkoda. Jest to metoda alternatywna dla metody poprzedniej.
- Metoda *measurement* – metoda, która dla zadanego w konstruktorze klasy obszaru kątowego, wyznacza odległości do przeszkód w zdyskretyzowanych obszarach (aby zapewnić efektywne działanie obszar musi wynieść co najmniej 10°). Warto wspomnieć, że robot cały czas pamięta swoją pozycję, dzięki enkoderom położenia kół, czyli odometrii.
- Metoda *mapupdate* – metoda publiczna, wywołująca 3 poprzednie, wywoływana podczas pracy programu *main*

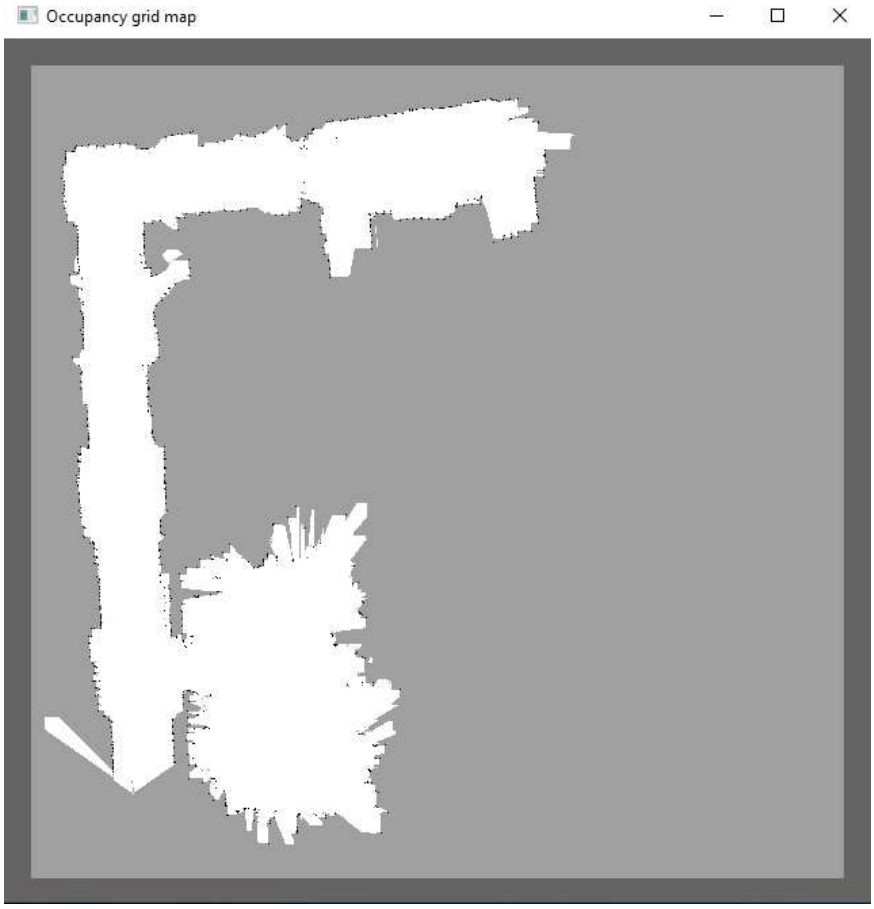
5. Program *main*

Program umożliwia sterowanie robota za pomocą strzałek na klawiaturze. Wciśnięcie klawisza „a” zapisuje mapę do pliku *OGmap.txt*, z którego korzysta aplikacja odpowiadająca za wyświetlanie mapy. W programie wywoływana jest cyklicznie metoda *mapupdate*. Dodałem również tryb bezpiecznej jazdy (robot w pewnej odległości hamuje przed przeszkodą a następnie całkowicie zwalnia).

6. Program *wyświetl*

Program umożliwia wyświetlenie stworzonej przez program *main* mapy w okienku Windows. W programie zastosowano progowanie wyświetlania (kolory czarny, szary i biały). Program odczytuje dane o mapie z pliku *OGmap.txt*

7. Ocena działania programu



Pomimo niedoskonałości, można się zorientować, że mapa przedstawia korytarz ZTMiR i obszar laboratorium. Błędy, które można dostrzec w działaniu programu:

- Oznaczanie jako wolny obszaru który jest za ścianą – spowodowane jest to występującymi błędnymi odczytami z czujnika laserowego (jeśli robot nie znajduje przeszkody, to zwraca wartość 20000, dlatego w swoim programie zastosowałem licznik *errcount*, który dla danego obszaru kątowego zlicza ilość wystąpień 20000 i uznaje ja jako poprawne, jeżeli licznik przekroczy ustaloną wartość)
- Ściany nie są ciągłe – spowodowane jest to działaniem funkcji *checkRangeDevicesCumulativePolar*, która działa poprawnie dla dość szerokiego zakresu 10° oraz zwraca mocno uśrednione wyniki
- Algorytm przeszukujący działa mało efektywnie jeżeli przeszkoda została znaleziona w linii pionowej bądź poziomej względem robota

8. Możliwości rozwoju pracy

- Udoskonalenie algorytmu przeszukującego
- Użycie funkcji *checkRangeDevicesCurrentPolar* i własnoręczna filtracja pomiarów
- Rozwinięcie programu do pełnego *SLAM*
- Budowa mapy wektorowej bądź topologicznej
- Autonomiczna jazda robota w nieznanym terenie
- Użycie sonaru (oprócz czujnika laserowego) i poprawa odczytów

9. Zawartość płyty

- W folderze *Bibliografia* umieściłem wszystkie pozycje w formacie pdf, z których korzystałem podczas zagłębiania się w tematykę pracy
- W folderze *Kod* zawarte są kody programów napisanych w C++ oraz przykładowy plik tekstowy, będący plikiem źródłowym mapy zawartej w tym sprawozdaniu
- W folderze *Testy* umieściłem screeny map z symulatora MobileSim, oraz zdjęcia przykładowych map. Są tam również nagrania testów w laboratorium.