Zad. 6: Sterowanie robotami mobilnymi w obecności przeszkód

1 Cel ćwiczenia

Utrwalenie umiejętności modelowania kluczowych dla danego problemu pojęć. Tworzenie diagramu klas oraz czynności. Wykorzystanie dziedziczenia do modelowanie kilku różnych pojęć będących uszczegółowieniem bardziej ogólnego pojęcia. Zastosowanie rzutowania *w górę* do stworzenia listy obiektów różnych typów. Wykorzystanie polimorfizmu i wskaźników dzielonych (tzw. inteligentnych).

2 Program zajęć

- *Ocena realizacji zadania z poprzedniego laboratorium* – ocenie podlega poprawność realizacji zadania, styl pisania programu oraz dokumentacja wygenerowana za pomocą programu *doxygen*.
- *Ocena przygotowania do zajęć* – ocenie podlega diagram klas.
- *Modyfikacja programu wg wskazań osoby prowadzącej* – ocenie będzie podlegała poprawność realizacji modyfikacji. Pracę nad modyfikacją programu (wszystkie operacje należy wykonywać na kopii) należy rozpocząć już w trakcie pierwszej fazy laboratorium, gdyż prowadzący nie będzie w stanie ocenić wcześniejszego programu wszystkim jednocześnie.
- *Realizacja wstępnej fazy prac nad nowym zadaniem* – w ramach wstępnej realizacji zadania należy zdefiniować klasy modelujące podstawowe pojęcia i doprowadzić od poprawnej kompilacji i konsolidacji programu.
- *Ocena realizacji wstępnej fazy zadania*

3 Opis zadania programowego

Niniejsze zadanie jest rozszerzeniem wcześniejszego zadania. Nowymi elementami są przeszkody reprezentowane przez prostokąty. Orientacja wspomnianych prostokątów powinna być taka, aby ich boki były równoległe do osi OX lub OY. Na scenie powinny być co najmniej trzy tego typu przeszkody. Zakłada się, że są one automatycznie tworzone na początku działania programu w taki sposób, że nie będą się nakładać na siebie. Oprócz tego na scenie w wolnej przestrzeni powinny być umieszczone co najmniej trzy roboty. Program powinien umożliwiać sterowanie każdym z robotów w analogiczny sposób jak w przypadku poprzedniego zadania. Identyfikacja robota i jego selekcja będzie realizowana poprzez wyświetlenie numerów kolejnych robotów i ich aktualnych współrzędnych. W trakcie wykonywania ruchu ma być sprawdzane, czy nastąpiła kolizja z przeszkodą lub innym robotem czy też nie, w przypadku kolizji ruch powinien zostać zakończony. Menu programu powinno zostać rozszerzone tylko o opcję umożliwiającą wyselekcjonowanie aktualnego robota, do którego będą odnosiły się polecenia sterowania.

4 Przygotowanie do zajęć

Należy przygotować diagram klas proponowanych struktur danych.

5 Wymagania co do konstrukcji programu

Oprócz wymagań sformułowanych w opisie zadania należy uwzględnić uwarunkowania przedstawione poniżej.

- Ze względu na to, że test kolizyjności ruchu robota dotyczy zarówno kolizji z przeszkodami, jak też z innymi robotami, należy wszystkie wszystkie obiekty tych typów umieścić na jednej liście. Aby to było możliwe, wspomniana lista powinna być listą wskaźników na obiekty klasy bazowej. Dzięki dziedziczeniu klasy `ObiektGraficzny` będzie możliwe wykorzystanie niejawnego rzutowania *w górę*.
- Listę obiektów należy stworzyć w oparciu o szablon `std::list<>`.
- Jako wskaźników, będących elementami listy, należy użyć wskaźników *dzielonych* (`std::shared_ptr<>`).
- W klasie `Scena` powinny być dwie listy. Jedna lista powinna być listą wskaźników do obiektów klasy `Robot` (należy wykorzystać szablon `std::shared_ptr<>`). Druga lista natomiast powinna dawać dostęp zarówno do obiektów klasy `Robot`, jak też obiektów klasy `Przeszkoda`. Przy konstrukcji drugiej listy należy skorzystać z mechanizmu rzutowania *w górę* oraz z szablonu `std::shared_ptr<>`. Lista ta będzie wykorzystana na potrzeby testu kolizyjności. Lista robotów będzie użyteczna na potrzeby procedury selekcji robota, który ma być sterowany.
- W obiekcie klasy `ObiektGraficzny`, podobnie jak w obiekcie klasy `Wektor2D` należy wprowadzić pole statyczne, które pozwoli zliczyć ile zostało utworzonych łącznie tego typu obiektów oraz ile ich pozostało na końcu działania programu. W dobrze skonstruowanym programie, gdy program kończy swoje działanie, nie powinno już być żadnego obiektu tego typu.

Dla przypomnienia:

- Program musi zachować strukturę modułową i odpowiednią strukturę kartotek. O ile będzie to konieczne, należy zmodyfikować plik `Makefile` (np. gdy dodany zostanie nowy moduł).
- Każda z klas powinna zostać zdefiniowana w oddzielnym pliku nagłówkowym. Metody tej klasy powinny być natomiast definiowane w osobnym module związanym z daną klasą, np. definicja klasy `Przeszkoda` powinna znaleźć się w pliku nagłówkowym `Przeszkoda.hh`, zaś metody w pliku `Przeszkoda.cpp`. Proste metody można definiować bezpośrednio w ciele klasy.
- Wszystkie metody, które nie zmieniają stanu obiektu, na którym działają, powinny być metodami typu **const**.
- Program powinien umożliwiać graficzną wizualizację wyników działania. Pozwala na to dołączony w załążku moduł łącza do programu `gnuplot`.

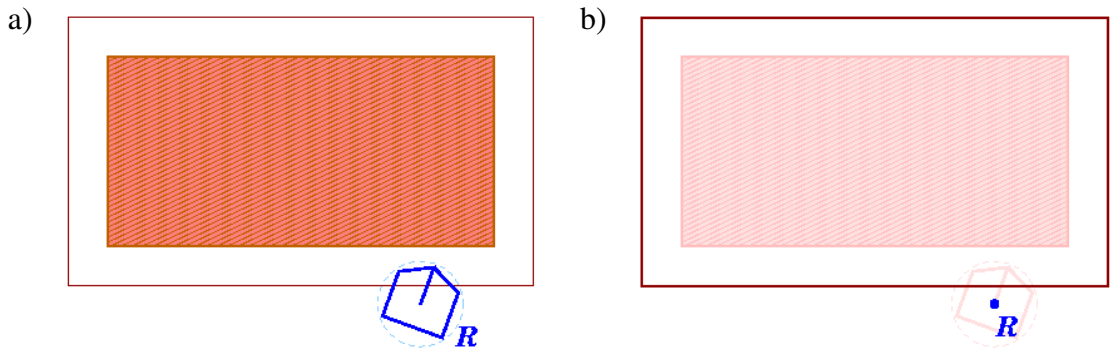
- Wszystkie klasy i metody oraz funkcje powinny zostać opisane. Opis ten powinien być zgodnie z wymogami systemu `doxygen`. Ponadto należy wygenerować dokumentację w formacie HTML za pomocą programu `doxygen`.

Oprócz tego pozostają w mocy wszystkie wcześniejsze wymagania dotyczące struktury katalogów, pliku Makefile, modułowej struktury programu, jak też opisów.

6 Problem detekcji kolizji

6.1 Kolizja z przeszkodą

Ze względu na przyjęte założenia co do kształtu przeszkód i sposobu ich umiejscowienia na scenie, rozwiązanie problemu detekcji kolizji można znacząco uprościć. Zakładając dodatkowo, że korpus robota mobilnego ma zwarty kształt, tak jak to jest przedstawione na rys. 2a, można aproksymować go okręgiem. To z kolei pozwala zastosować podejście opracowane przez Lozano-Pereza znane pod pojęciem *przestrzeni konfiguracyjnej*. Wykorzystywane jest ono do planowania ścieżek bezkolizyjnych dla konstrukcji, których kształt można aproksymować wielobokami lub okręgami. W przypadku, gdy do aproksymacji zastosujemy okrąg, staje się ono jeszcze prostsze. Sprowadza się wówczas do *powiększenia* przeszkody o promień okręgu, zaś



Rysunek 1: Przykład zastosowania sposobu redukcji korpusu robota do punktu: a) faktyczne kształty przeszkody i robota, b) efekt rozszerzenia przeszkody i redukcji robota do punktu materialnego

korpus robota redukujemy do punkty (patrz rys. 2b). Problem detekcji kolizji redukuje się wówczas do sprawdzenia, czy punkt reprezentujący robota znajduje się w prostokącie modelującym przeszkodę, czy też nie.

6.2 Kolizja z innym robotem

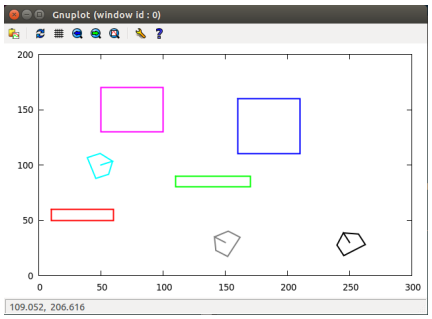
Roboty wzajemnie dla siebie też są przeszkodami. W tym przypadku wystarczy skorzystać z tego, że korpus robota aproksymujemy okręgiem. Dzięki temu problem sprawdzenia, czy zachodzi kolizja, sprowadza się do problemu sprawdzenia, czy odległość między dwoma punktami jest mniejsza niż suma promieni aproksymujących korpusy każdego z robotów. Należy przy tym zwrócić uwagę, że wspomniana odległość jest długością wektora poprowadzonego między punktami reprezentującymi położenie robotów. Z kolei wspomniany wektor jest różnicą wektorów, których współrzędne odpowiadają położeniom robotów.

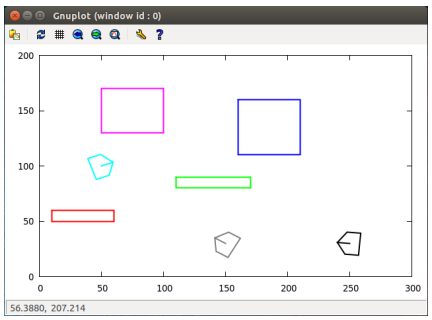
a) b)

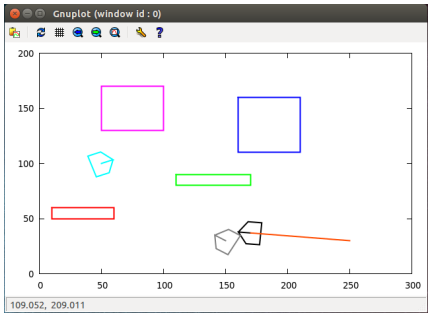
Rysunek 2: Przykład rozwiązania problemu sprawdzenia czy zachodzi kolizja między robotami poprzez aproksymację ich korpusów okręgami, dzięki temu test kolizyjności sprowadza się do problemu sprawdzenia odległości między dwoma punktami a) faktyczne kształty robotów, b) efekt sprowadzenia robotów do okręgów

7 Przykład działania programu

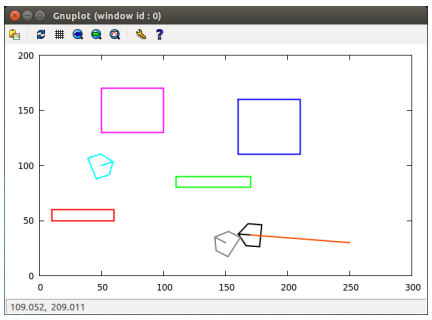
```
jkowalsk@panamint> ./roboty_przeszkody
 Laczna ilosc stworzonych obiektow klasy Wektor2D: 1192
       Ilosc istniejacych obiektow klasy Wektor2D: 124

Aktualnie wyselekcjonowanym robotem jest:
  Robot 2.   Wspolrzedne: (250, 30)

  o - obrot robota
  j - jazda na wprost
  s - selekcja robota

  w - wyswietl ponownie menu

  k - zakoncz dzialanie programu

 Twoj wybor (w - wyswietl menu)> o

Aktualnie wyselekcjonowanym robotem jest:
  Robot 2.   Wspolrzedne: (250, 30)

 Podaj wartosc kata obrotu robota w stopniach.
 Kat obrotu: 55

 Twoj wybor (w - wyswietl menu)> j


Aktualnie wyselekcjonowanym robotem jest:
  Robot 2.   Wspolrzedne: (250, 30)

 Podaj dlugosc drogi ruchu robota na wprost.
 Dlugosc drogi: 150

 !!! Ruch nie moze być kontynuowany ze wzgledu
 !!! na wystapienie kolizji.

 Twoj wybor (w - wyswietl menu)> s
```



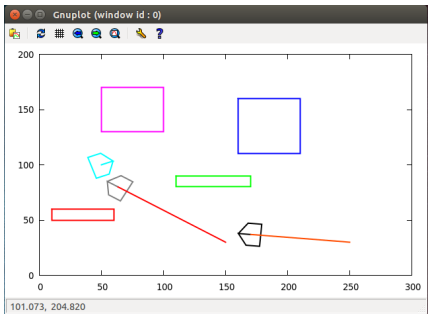




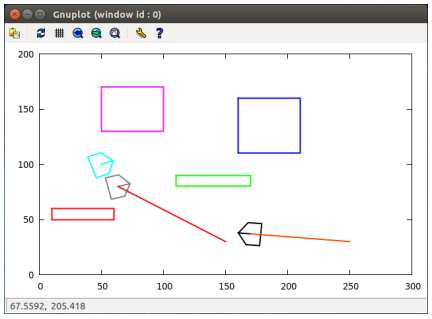
```
Aktualnie wyselekcjonowanym robotem jest:
  Robot 2.    Wspolrzedne: (170.304, 36.9725)


        0 - zaniechaj zmiany selekcji

  Robot 1.    Wspolrzedne: (50, 100)
  Robot 2.    Wspolrzedne: (170.304, 36.9725)
  Robot 3.    Wspolrzedne: (150, 30)

 Podaj numer robota, dla ktorego maja być wykonane operacje sterowania

 Wprowadz numer robota lub 0 > 3

  Robot 3.    Wspolrzedne: (150, 30)

 Twoj wybor (w - wyswietl menu)> j
```

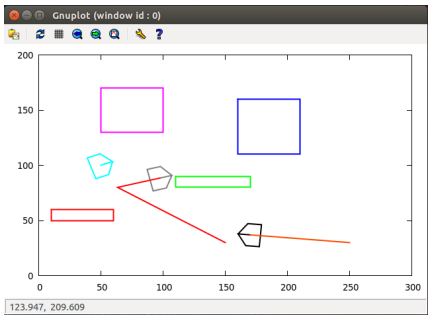



```
Aktualnie wyselekcjonowanym robotem jest:
  Robot 3.    Wspolrzedne: (150, 30)

 Podaj dlugosc drogi ruchu robota na wprost.
 Dlugosc drogi: 150

 !!! Ruch nie moze być kontynuowany ze wzgledu
 !!! na wystapienie kolizji.

 Twoj wybor (w - wyswietl menu)> o
```




```
Aktualnie wyselekcjonowanym robotem jest:
  Robot 3.    Wspolrzedne: (63.3975, 80)

 Podaj wartosc kata obrotu robota w stopniach.
 Kat obrotu: -136

 Twoj wybor (w - wyswietl menu)> j


Aktualnie wyselekcjonowanym robotem jest:
  Robot 3.    Wspolrzedne: (63.3975, 80)

 Podaj dlugosc drogi ruchu robota na wprost.
 Dlugosc drogi: 100

 !!! Ruch nie moze być kontynuowany ze wzgledu
 !!! na wystapienie kolizji.

 Twoj wybor (w - wyswietl menu)> k
```






```
  Laczna ilosc stworzonych obiektow klasy ObiektGraficzny: 10
       Ilosc nieusunietych obiektow klasy ObiektGraficzny: 0

jkowalsk@panamint> _
```

8 Materiały pomocnicze

Do tego zadania nie jest dostarczany żaden zalążek. Pracę należy rozpocząć wykorzystując moduły programu z zadania nr 4 lub 5 (dla wersji rozszerzonej).

W na `panamincie` w podkartotece `~bk/edu/kpo/zad/z7` można znaleźć skrypt, który uruchamia program demonstrujący przykładową realizację zadania. Skrypt ten można uruchomić poleceniem:

```
~bk/edu/kpo/zad/z6/przyklad_rozwiazania.sh
```

Ze względu na to, że program tworzy okienko graficzne, w przypadku połączenia z panamintem za pomocą programu `ssh` należy pamiętać, aby użyć opcji `-X`.

9 Rozszerzenia

Możliwych jest kilka wariantów rozszerzenia tego zadania.

9.1 Rozszerzenie grafiki

Zamiast rysowania 2D należy rysować przeszkody i robota oraz jego ścieżkę ruchu wykorzystując grafikę 3D.

9.2 Możliwość dodawania i usuwania przeszkód w trakcie działania programu

Program powinien umożliwiać dodawanie i usuwanie wskazanej przeszkody. W przypadku usuwania przeszkoda powinna być odpowiednio selekcjonowana, co powinno być widoczne na wyświetlanym obrazie sceny roboczej.

9.3 Modyfikacja położenia i rozmiarów przeszkód

Program powinien umożliwić wyselekcjonowanie odpowiedniej przeszkody i zmianę jej rozmiarów, jak też położenia i orientacji.

9.4 Detekcja kolizji dla przypadku dowolnej orientacji przeszkody

Program powinien umożliwiać detekcję kolizji dla dowolnej orientacji przeszkody modelowanej za pomocą prostokątów.

10 Wymagania i zarys programu zajęć w okresie realizacji zadania

Przystępując do pracy nad programem zaleca się, aby rozszerzenie menu programu zrealizować na samym końcu, gdy stworzymy już i przetestujemy wszystkie niezbędne funkcjonalności.

10.1 Tydzień 0

Należy przygotować diagram klas modelujący pojęcia sceny, robota, przeszkody oraz ścieżki ruchu robota. Należy zwrócić uwagę na konieczność transformacji współrzędnych. W tym celu konieczne jest wykorzystanie pojęć wektora i macierzy.

10.2 Tydzień 1

Przed zajęciami muszą zostać przygotowane następujące elementy zadania. Wszystko co będzie ponad to będzie oceniane *in plus* (oprócz menu programu).

- Zdefiniowane powinna być klasa `Przeszkoda` z najistotniejszymi metodami. Należy zwrócić uwagę, że pojęcia `Robot`, `Przeszkoda` oraz `Sciezka` w tym zadaniu wiążą się z odpowiednią reprezentacją graficzną. To jest element wspólny tych pojęć. Tak więc klasa `Przeszkoda`, podobnie jak klasy `Robot` i `Sciezka` powinna dziedziczyć klasę `ObiektGraficzny`.

  W klasie `Scena` powinna być zdefiniowana lista, które zawierać będzie wszystkie obiekty graficzne (przeszkody, roboty i ich ścieżki). Zalecane jest, aby była też druga lista, która będzie zawierała same roboty. Do przechowywania wskaźników należy wykorzystać szablon std::shared_ptr<>.
- Warunkiem koniecznym pozytywnej oceny jest poprawna kompilacja. W trakcie kompilacji nie powinny być generowane żadne ostrzeżenia.
- W funkcji `main` powinien być kod, który demonstruje rysowanie sceny zawierającej co najmniej dwie przeszkody i robota.
- Wszystkie klasy i metody muszą być opisane oraz powinna być generowana dokumentacja za pomocą programu `doxygen`.

10.3 Tydzień 2

Rozliczenie się z gotowego programu i rozpoczęcie następnego zadania.