AKADEMIA GÓRNICZO-HUTNICZA
im. St. Staszica w Krakowie
WEAIiE, Katedra Automatyki
Laboratorium Biocybernetyki

Przedmiot: Sztuczna inteligencja i sensoryka. PRSI114_Assistive2

Temat projektu: Wspomaganie lokalizacji osób niepełnosprawnych w pomieszczeniach

Spis treści:

Wykonali: Kamil Neczaj, Ernest Staszuk
studenci Automatyki i Robotyki
konsultant: *dr inż. Jaromir Przybyło*

Wersja 1.0
Kraków, styczeń 2010.

1. Abstrakt

Celem projektu było zbudowanie systemy nawigacji dla osób niepełnosprawnych w budynkach. System na podstawie tzw. markerów pozycjonuje osobę, która po wybraniu celu dostaje wskazówki dotarcia do niego.

Rdzeniem nawigacji jest mapa środowiska w którym ma działać aplikacja (w postaci grafu reprezentującego punkty orientacyjne w przestrzeni oraz trasy pomiędzy nimi). Do lokalizacji w przestrzeni mapy użytkownika naszej aplikacji wykorzystano markery, które zostały rozwieszone na ścianach pomieszczeń. Do ich wykrywania użyto biblioteki ARToolKit. Wykrywa ona markery w polu widzenia kamery i zwraca informacje pozwalające obliczyć pozycję oraz orientację kamery względem markera. Dodając informacje pozwalające pozycjonować markery na mapie otrzymaliśmy możliwość wyliczenia pozycji oraz orientacji kamery w pomieszczeniach. W rezultacie system może zaproponować użytkownikowi aplikacji w którą stronę powinien się kierować, aby dość do wybranego celu.

Aplikacja spełnia swoją funkcję. Działa bez większych zarzutów przy dobrym oraz równomiernym oświetleniu. Niestety pojawiają się problemy z prawidłowym rozpoznawaniem markerów przy zbyt słabym, zbyt mocnym lub nierównomiernym oświetleniu.

Nasza aplikacja (wraz z kodem źródłowym) jest dostępna za pomocą systemu kontroli wersji SVN pod adresem: http://code.google.com/p/assistive2/ oraz na załączonej płycie DVD. Płyta DVD zawiera dodatkowo specjalnie przygotowane filmy pozwalające zademonstrować działanie aplikacji w trybie „off-line” na przykładzie mapy przygotowanej przez nas dla jednego z korytarzy budynku C3 Akademii Górniczo-Hutniczej.

Słowa kluczowe:

Marker, ARToolKit, openCV, nawigacja w pomieszczeniach, wspomaganie niepełnosprawnych, rozszerzona rzeczywistość.

2. Wstęp

a) Cele i założenia projektu.

Celem projektu jest zbudowanie systemu umożliwiającego pozycjonowanie oraz nawigację osób niepełnosprawnych w budynku na podstawie obrazu otrzymywanego z kamery.

b) Zarys ogólny proponowanego rozwiązania.

Do osiągnięcia naszego celu zostały wykorzystane biblioteki programistyczne ARToolKit oraz OpenGL, specjalnie przygotowane markery oraz zapisana w odpowiednim formacie (który zostanie omówiony w dalszej części niniejszej pracy) mapa pomieszczeń w których system ma działać. Wykorzystywana przez program biblioteka ARToolKit odpowiada za interpretację obrazu (wyszukiwanie markerów, klasyfikację ich oraz określanie położenia kamery względem nich). Same markery służą do oznaczania punktów nawigacyjnych w budynku. Mapa pozwala określić pozycję oraz orientację zastosowanych markerów oraz przechowuje informacje o ścieżkach pomiędzy punktami nawigacyjnymi. Sam program na podstawie danych o widzianych markerach oraz położenia kamery względem nich określa pozycję użytkownika systemu w przestrzeni budynku, szuka najkrótszej trasy do celu za pomocą algorytmu Dijkstry oraz proponuje kierunek w którym należy się udać.

c) Dyskusja alternatywnych rozwiązań.

Wartym uwagi rozwiązaniem, jednak niekorzystającym z systemu wizyjnego jest użycie akcelerometru oraz żyroskopu w celu uzyskania informacji o przemieszczeniach użytkownika. Ma ono jednak bardzo poważne niedogodności, a mianowicie konieczności kalibracji położenia początkowego, powtórnej kalibracji co jakiś z powodu zakłóceń pomiaru oraz wymagałoby bardzo dokładnej mapy odwzorowującej pomieszczenia. System wizyjny nie wymaga kalibracji położenia początkowego, ani powtórnej kalibracji podczas działania, gdyż błędy pomiarowe się w tym przypadku nie kumulują. Mankamentem jest zaś fakt, że przez cały czas działania takiej aplikacji markery muszą być widoczne. Ciekawe mogłoby być połączenie obu rozwiązań, nawigacji za pomocą akcelerometru i żyroskopu, z automatyczną kalibracją położenia przy pomocy systemu wizyjnego, gdy markery znajdą się w polu widzenia kamery.

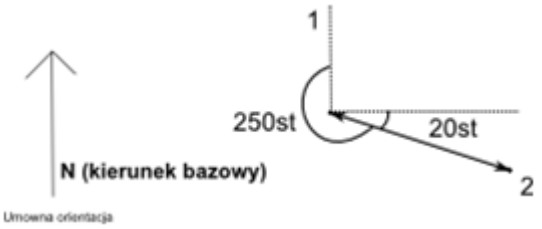
Innym rozwiązaniem mogłoby być zrezygnowanie z użycia markerów na rzecz analizy cech obrazu. W naszej opinii to podejście pozwoliłoby uzyskać interesujące rezultaty i jest warte uwagi. Zrealizowanie i przetestowanie takiego systemu niestety wychodziło poza ramy naszego projektu.

3. Koncepcja proponowanego rozwiązania

Działanie systemu polega na wykonywaniu w pętli następujących operacji: wczytywania i analizy ramki obrazu przez ARToolkit, wyznaczenia pozycji użytkownika i kierunku do celu, a na końcu wizualizacji wskazówek nawigacji na obrazie. Dokładny schemat głównej pętli programu wraz z przydziałem zadań do obiektów zajmujących się nimi przedstawiony jest na schemacie 1 w dalszej części opracowania.

Format grafu

Dane budynku przechowywane są w grafie. Wierzchołek grafu reprezentuje miejsce na mapie, do którego program może nas zaprowadzić. Każdy wierzchołek zawiera listę połączeń i listę markerów przypisanych do wierzchołka (może ich być więcej niż jeden). Sam marker nie jest tożsamy z wierzchołkiem (mogą mieć różne położenia). Do określenia wzajemnego położenia wierzchołków i markerów wprowadzono pojęcie kierunku bazowego (N). Jest to kierunek umowny dla każdego zestawu danych wejściowych. Wszystkie kąty wzajemnego położenia wierzchołków i markerów mierzone są względem kierunku bazowego. Dzięki takiemu rozwiązaniu przekształcenia układów współrzędnych za pomocą macierzy transformacji powstałych na podstawie połączeń grafu i uzyskanych z kamery można ze sobą łączyć. Wynikowe kąty także są określone względem kierunku bazowego.



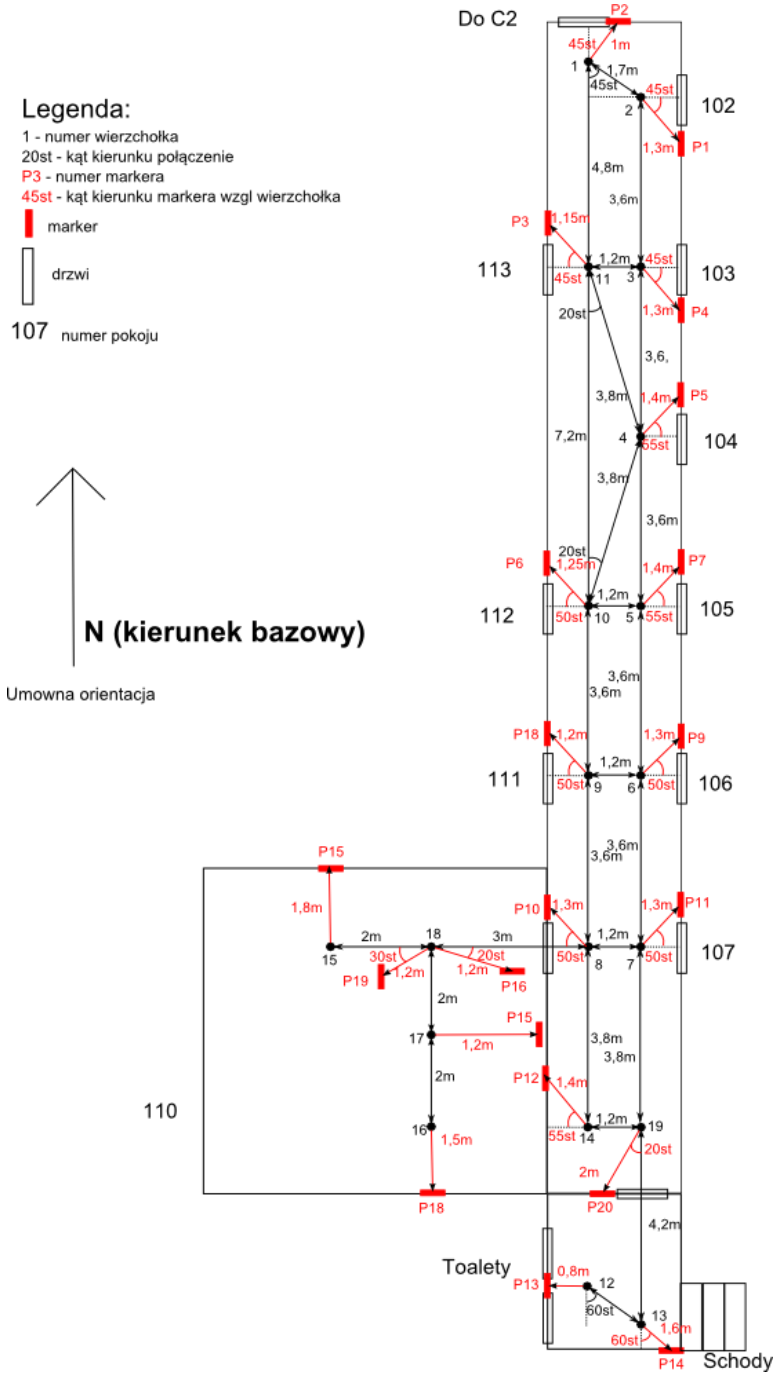
Rysunek 3.1 Przykład kątów połączenia grafu

Sposób wyznaczania kąta połączenia będzie przedstawiony na podstawie powyższego przykładowego połączenia. Tutaj $\measuredangle(1; 2)$ wynosi 250°, zaś $\measuredangle(2; 1)$ wynosi 110°.

Każdy marker przechowuje informacje o swoim położeniu względem najbliższego wierzchołka. Są to odległość, kąt – zapisywany w sposób identyczny co kąt połączenia, i orientację. Orientacja jest to kąt pomiędzy kierunkiem bazowym, a wektorem wychodzącym z płaszczyzny markera, prostopadłym do niej, skierowanym w stronę patrzącej na niego osoby.

W programie zastosowany został lewoskrętny układ współrzędnych (ma on oś Y skierowaną ku górze pomieszczenia, dla kąta 0° oś Z pokrywa się z kierunkiem bazowym, zaś oś X skierowana jest w lewo).

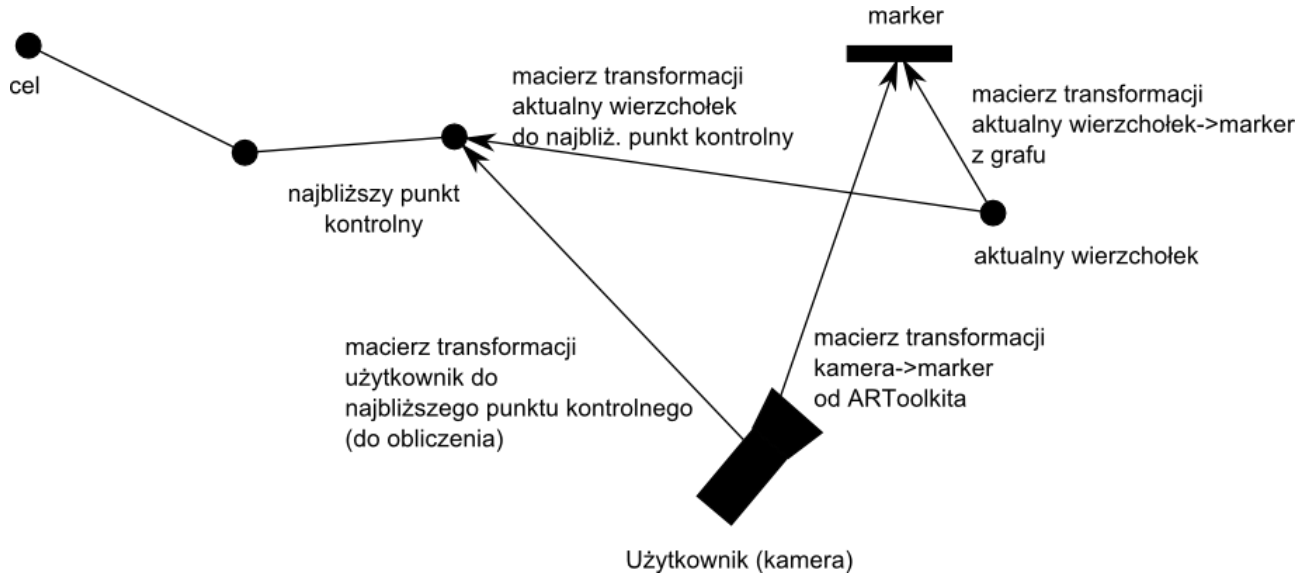
Ostatecznie graf w formie rysunku wygląda tak:



Rysunek 3.2 Mapa-graf pierwszego piętra budynku C3 AGH

Sposób wyznaczania wskazówek nawigacji

Klasa „Guider" odpowiedzialna za wyznaczanie wskazówek nawigacji dostaje od klasy „ARToolkitWrapper" listę wszystkich znalezionych markerów. Następnie spośród nich wybiera najbliższy i za pomocą algorytmu Dijkstry zawartego w metodzie „getPath" obiektu „Graph" wyznacza ścieżkę do celu.



Rysunek 3.3 Schemat użycia macierzy transformacji stosowanych w programie

Przyjmując oznaczenia macierzy transformacji:

`user2marker` – użytkownik -> marker

`node2marker` – aktualny wierzchołek -> marker

`node2aim` –aktualny wierzchołek -> najbliższy punkt kontrolny

`user2aim` – użytkownik -> najbliższy punkt kontrolny

`node2user` - aktualny wierzchołek -> użytkownik

Odległość do następnego punktu kontrolnego obliczana jest według następującego schematu:

```
node2user = node2marker*user2marker.inverse();
user2aim = node2user.inverse()*node2aim;
```

Na podstawie powyższej macierzy wyznaczane są odległość do najbliższego punktu kontrolnego i kąt połączenia względem osi Y skierowanej ku niebu.

Przyjmując, że macierz transformacji to m, tangens kąta obliczono ze wzoru:

$$tg \propto = \frac{m_{1,4}}{m_{3,4}}, \qquad \textbf{(wzór 3.1)}$$

Odległość zaś ze wzoru:

$$d = \sqrt{m_{1,4}^2 + m_{2,4}^2 + m_{3,4}^2}\,. \qquad \textbf{(wzór 3.2)}$$

4. Rezultaty i wnioski

W rezultacie naszej pracy powstał działający system, który wyświetla wskazówki dojścia do wybranego celu. Jego działanie było satysfakcjonujące przy dobrym oświetleniu.

Problemem przy używaniu aplikacji jest jej niezadowalające działanie przy niedostatecznym lub nierównomiernym oświetleniu. Biblioteka wykorzystywana do wykrywania markerów nie radzi sobie dostatecznie dobrze z powierzonym jej zadaniem.

Kolejnym problemem jest również konieczność ręcznego wprowadzania dużych grafów do programu. Istnieje możliwość pomyłki przy wpisywaniu parametrów połączeń do pliku tekstowego. Rozwiązaniem tego problemu byłoby stworzenie dla aplikacji specjalnego edytora mapy.

5. Wyniki działania programu oraz wnioski z analiz

Działanie programu można sprawdzić uruchamiając program używając przygotowane przykładowe filmy nagrane w korytarzu budynku C3 Akademii Górniczo-Hutniczej (są one nagrane na załączonej płycie DVD).

System nawigacji opiera się głównie na pomiarze odległości, dlatego też poniżej załączamy analizę skuteczności tego pomiaru dla używanych markerów o boku 20cm.

Pomiar miarką [mm]	Pomiar wizyjny [mm]	błąd względny
425	450	0,059
535	555	0,037
600	620	0,033
828	830	0,002
990	998	0,008
1265	1280	0,012
1420	1422	0,001
1610	1630	0,012
1770	1800	0,017
2060	2010	0,024

Tabela 5.1 Porównanie odległości mierzonych za pomocą systemu wizyjnego i miarki

Pomiar dokonywany był przy kamerze skierowanej na wprost markera za pomocą przykładowego programu dołączonego do biblioteki ARToolkit „exview”.

Wizyjny pomiar odległości jest wystarczająco dokładny. Błąd względny w niewielu przypadkach przekracza 3%. Przy interpretacji wyników należy wziąć pod uwagę, że sam pomiar za pomocą miarki mógł być nieprecyzyjny. Zatem błąd rzędu kilku mm może być spowodowany

niedokładnościami pomiaru referencyjnego. Otrzymana dokładność mierzenia odległości znacznie przewyższa wymaganą.

Wykres 5.1 Dokładność pomiaru – punkty pomiarowe względem prostej y=x

6. Podsumowanie

Wynikiem prac jest system nawigacji, który po wykryciu markera potrafi bardzo dokładnie wyliczyć pozycję użytkownika i podać mu prawidłowe wskazówki nawigacji. Niestety proces wykrywanie oraz klasyfikacji markerów przeprowadzany przy pomocy biblioteki ARToolkit jest bardzo wrażliwy na warunki oświetleniowe i niektórych warunkach nie wykrywa markerów prawidłowo.

7. Literatura

Strona domowa projektu ARToolkit - http://www.hitl.washington.edu/artoolkit/

8. DODATEK A: Opis opracowanych narzędzi i metody postępowania

Tworzenie własnego grafu

W celu przystosowania programu do nawigacji w innym otoczeniu należy sporządzić graf połączeń. W tym celu użytkownik powinien wygenerować markery zgodnie z instrukcją na stronie internetowej projektu ARToolKit. Po ich wydrukowaniu kolejnym krokiem jest rozwieszenie ich w budynku i dokonanie pomiarów odległości i kątów w celu przygotowania mapy podobnej do tej

przedstawionej na rysunku 3.2. Użyte markery należy również zapisać w formacie rozpoznawanym przez bibliotekę ARToolKit za pomocą dołączonego do niej programu „mk_patt”.

Format danych wejściowych

Dane wejściowe (graf z mapą połączeń) wczytywany jest z pliku tekstowego mapa.graph w katalogu data programu.

Oto format tego pliku (komentarze to linie zaczynające się od znaku „#”):

```
GR
Ilość_wierzchołków_grafu

nr wierzchołka
nazwa wierzchołka
plik z markerem 1
odległość markera 1 od wierzchołka
kąt wektora wierzchołek->marker 1 względem kierunku bazowego
kąt orientacji markera
# można powtórzyć 4 ostatnie linijki dla kolejnych markerów w danym wierzchołku

nr wierzchołka
nazwa wierzchołka
plik z markerem 1
odległość markera 1 od wierzchołka
kąt wektora wierzchołek->marker 1 względem kierunku bazowego
kąt orientacji wierzchołka
# można powtórzyć 4 ostatnie linijki dla kolejnych markerów w danym wierzchołku
#kolejne wierzchołki rozdziela pusta linia
*
#Polaczenia pomiedzy nodami (należy zwrócić uwagę na gwiazdkę przed tą sekcją)
Wierzchołek_1 Nr_wierzchołek_2 odległość kąt_skierowany_od_1_do_2_wzgl_kier_baz
Wierzchołek_1 Nr_wierzchołek_2 odległość kąt_skierowany_od_1_do_2_wzgl_kier_baz
Wierzchołek_1 Nr_wierzchołek_2 odległość kąt_skierowany_od_1_do_2_wzgl_kier_baz
# itd
*
Po gwiazdce nic już nie zostanie wczytane.
```

Przykładowy plik z danymi wejściowymi zbudowany w oparciu o mapę korytarza w budynku C3 AGH znajduje się w katalogu „data” w folderze programu, może on posłużyć jako dodatkowy wzór.

Instrukcja użytkowania

Po uruchomieniu programu w konsoli zostanie wyświetlone menu z dostępnymi trybami pracy (pierwsza opcja jest trybem interaktywnym przechwytującym obraz z kamery, pozostałe opcje to demonstracje przygotowanych przez nas przykładów). W trybie interaktywnym należy

wybrać miejsce docelowe. Na początku pojawi się spis nazw wierzchołków wczytanego grafu. Użytkownik wpisuje numer interesującego go wierzchołka i naciska klawisz „Enter”. W tej chwili rozpoczyna się właściwa nawigacja. Program wyświetla wskazówki dotarcia do celu, gdy tylko wykryje któryś z markerów zapisanych w pliku wejściowym. Aby zmienić cel należy ponownie nacisnąć klawisz „Enter”. Po wejściu do menu można z niego wyjść klawiszem „Esc”. Z programu wychodzimy naciskając klawisz „Esc” będąc poza menu wyboru celu.

9. DODATEK B: Realizacja proponowanego rozwiązania

Do realizacji programu wykorzystano język programowania C++. Program testowany był na systemie operacyjnym Windows. Pliki projektów zapisane są w formacie Visual Studio 2008, gdyż jest to jedyne środowisko programistyczne, które jest wspierane przez bibliotekę ARToolkit pod systemem Microsoft Windows.

Program testowano z użyciem kamery Logitech EyeToy z wybraną rozdzielczością 640x480 pikseli.

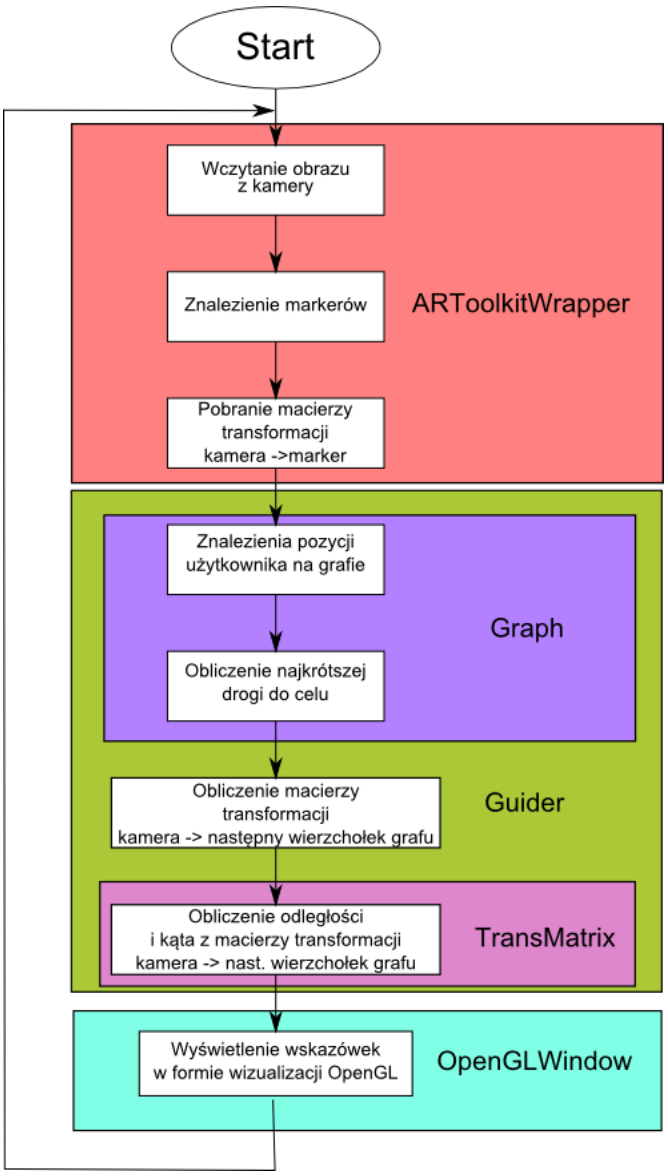
Organizacja programu

Podstawowe typy obiektów (danych) w programie:

- Pattern – klasa opisująca marker, jego położenie względem kamery, położenie w grafie (wskaźnik na najbliższy wierzchołek), informacje o dopasowaniu do przetwarzanej klatki.
- TransMatrix – klasa macierzy transformacji. Oprócz macierzy zawiera metody pozwalające na operacje na macierzach transformacji: mnożenie, odwracanie, a także umożliwiające wyciągnięcie z niej informacji o przemieszczeniu i kącie przemieszczenia.
- Node – klasa wierzchołka grafu. Zawiera listę połączeń i listę markerów przyporządkowanych do danego wierzchołka.

Poza tym program dzieli się na kilka głównych komponentów zajmujących się przetwarzaniem danych, także zorganizowanych w klasy:

- ARToolkitWrapper – obsługa backendu ARToolkit, jego zadaniem jest przechwytywanie obrazu z kamery i jego analiza – znajdowanie markerów na rysunku.
- Graph – wczytywanie grafu z pliku i szukanie najkrótszej ścieżki
- Guider – zajmuje się ogólnie pojętą nawigacją. Metoda update dostarcza aktualną listę markerów, na podstawie której Guider generuje wskazówki nawigacji
- OpenGLWindow – zawiera metody służące do wizualizacji nawigacji.

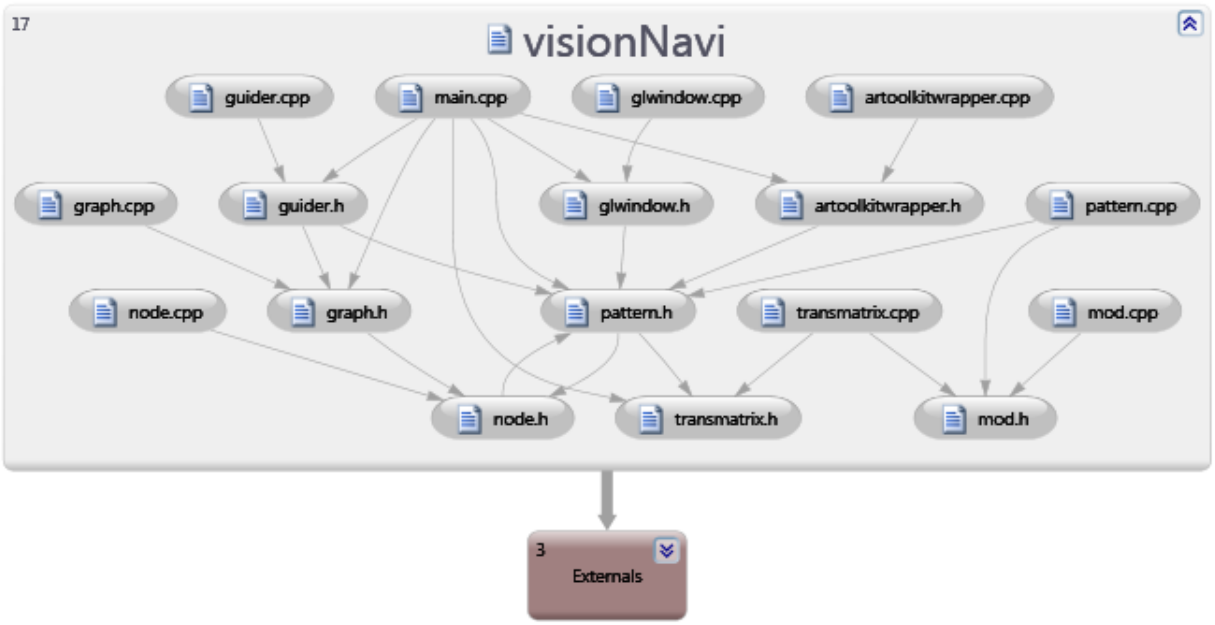


Rysunek 9.1 Pętla główna programu

10. DODATEK C. Opis informatyczny procedur

Projekt został przygotowany w środowisku programistycznym Visual Studio 2008. Wybór taki podyktowany został wsparciem biblioteki ARToolKit. Użyto języka programowania C++. Aplikacja korzysta z bibliotek ARToolKit oraz GLut (ich licencje pozwalają na niekomercyjny użytek), więc odpowiednie pliki nagłówkowe powinny znaleźć się w katalogu „include" projektu, aby proces kompilacji się powiódł. W czasie procesu linkowania w katalogu „lib" projektu muszą znaleźć się następujące biblioteki dołączane statycznie: libAR.lib, libARgsub.lib, libARvideo.lib (z ARToolKit'a) oraz glut32.lib (z GLut). Podczas uruchamiania aplikacji niezbędne są następujące biblioteki dołączane dynamicznie: DSVL.dll, libARvideo.dll oraz glut32.dll. Dodatkowo w systemie powinny znajdować się standardowe biblioteki, a w szczególności: msvcp71.dll, msvcr71.dll oraz OpenGL.

Utworzone pliki należące do projektu oraz zależności pomiędzy nimi:



Opis najistotniejszych funkcji zawartych w programie:

- `list<Pattern*> ARToolKitWrapper::getScene()`

  Przeznaczenie: Dopasowuje elementy tablicy markerInfo (składowa klasy), zawierającej informacje o wykrytych obiektach do dostępnych markerów.

  Argumenty funkcji: brak

  Zwracana wartość: Kontener (listę) zawierający znalezione markery (w postaci wskaźników na typ „Pattern").

  Używane funkcje:

```
double arGetTransMat(ARMarkerInfo *marker_info, double center[2],
      double width, double conv[3][4] ); (z AR\ar.h)
std::list<Pattern*>::list();
list<Pattern*>::push_back(Pattern* const &);
list<Pattern*>::iterator::iterator();
list<Pattern*>::iterator->begin();
list<Pattern*>::iterator->end();
list<Pattern*>::iterator->operator++(int);
```

  Używane zmienne:

```
std::list<Pattern*> scene; // Przechowuje wszystkie rozpoznane markery
list<Pattern*>::iterator pat; // Iterator
this->marker_num; // Liczba wykrytych markerów
ARMarkerInfo*   markerInfo;
```

  Uwagi:

```
Funkcja korzysta z biblioteki ARToolKit.
```

  Autor:

```
Kamil Neczaj (2011).
```

  Ostatnia modyfikacja:

```
19 grudnia 2011, dopasowanie progowania prawdopodobieństwa.
```

- `bool Guider::update(const list<Pattern*> scene)`

  Przeznaczenie: Aktualizuje system nawigacji na podstawie widocznych markerów.

  Argumenty funkcji: `const list<Pattern*> scene – lista wykrytych markerów`

  Zwracana wartość: nic

  Używane funkcje:

```
double arGetTransMat(ARMarkerInfo *marker_info, double center[2], double width,
     double conv[3][4] ); (z AR\ar.h)
Konsruktor list<Pattern*>::iterator;
list<Pattern*>::iterator->begin();
list<Pattern*>::iterator->end();
list<Pattern*>::iterator->operator++(int);
list<Pattern*>::empty();
list<Pattern*>::getYAngle();
Konstruktor TransMatrix;
TransMatrix::getDistance();
TransMatrix::inverse();
TransMatrix operator*(TransMatrix const &matrix) const;
Graph::getPath(nearestNode, aim, path);
Guider::getDirection();
Guider::makeHint();
```

Używane zmienne:

```
list<Pattern*>::const_iterator i; // Iterator
Node* Guider::nearestNode; //Najbliższy znaleziony punkt nawigacyjny
Pattern* Guider::nearestPattern;  // Najbliższy znaleziony marker
Node* Guider::aim;   // Cel
list<Node*> Guider::path; // Znaleziona sciezka do celu
TransMatrix node2marker; //Macierz transformacji układu wsp.
TransMatrix user2marker; //Macierz transformacji układu wsp.
TransMatrix node2user; //Macierz transformacji układu wsp.
TransMatrix node2aim; //Macierz transformacji układu wsp.
TransMatrix Guider::user2aim; //Macierz transformacji układu wsp.
Double  Guider::angle; // Kat
Double Guider::nextNodeDistance; // Dystans do najblizszego punktu kontrolnego
Double Guider::aimDistance; // Dystans do celu
list<gConn*>::iterator i; //Iterator
bool Guider::_atAim; // Czy jest u celu ?
```

Autorzy:

Ernest Staszuk, Kamil Neczaj (2011).

Ostatnia modyfikacja:

17 listopd 2011, usuwanie błędów.

- `static void mainLoop(void)`

Przeznaczenie: główna pętla programu
Argumenty funkcji: brak
Zwracana wartość: brak
Używane funkcje:

```
bool ARToolKitWrapper::grabFrame();
void ARToolKitWrapper::findMarkers();
list<Pattern*> ARToolkitWrapper::getScene();
static GLWindow* GLWindow::instance();
void GLWindow::drawBackground(float startX, float startY,
        float sizeX, float sizeY);
void GLWindow::printString(string str, float x, float y);
void GLWindow::drawArrow(double yAngle);
void GLWindow::drawVideo();
string Guider::aimName();
void Guider::update(list<Pattern*> scene);
string Guider::getHint();
bool string::empty();
void argSwapBuffers();
void menuGL();
```

Używane zmienne:

```
list<Pattern*> scene; // Używane markery
ARToolKit ar; // Obiekt klasy ARToolKit
GLWindow gl; // Obiekt GLWindow
Guider guider; // Obiekt klasy guider
bool enterToMenu; // Flaga sterujaca
string aim; // Nazwa celu
ostringstream os; // Strumien
string hint; // Wskazówka nawigacyjna
```

Uwagi:

Funkcja zawiera pętle główną programu.

Autor:

Ernest Staszuk, Kamil Neczaj (2011).

Ostatnia modyfikacja:

17 grudnia 2011, modyfikacje związane z obsługą filmów demonstracyjnych.

- `void Graph::getPath(Node* start, Node* stop, list<gConn*>& path)`

  Przeznaczenie: znajdowanie najkrótszej ścieżki do celu.

  Argumenty funkcji:

  ```
  Node* start – punkt początkowy ścieżki
  Node* stop – punkt końcowy ścieżki
  list<gConn*>& path – zmienna przechowująca ścieżkę – tutaj zapisywany jest wynik
  ```

  Zwracana wartość: brak, wynik zwracany przez referencję (`list<gConn*>& path`).

  Użyte funkcje:

  ```
  list<Node*>::push_front(Node*);
  list<Node*>::pop_front(Node*);
  list<Node*>::operator++(int);
  list<gConn*>::clear();
  ```

  Użyte zmienne:

  ```
  list<Node*>::iterator it; // Iterator
  Node* act; // Aktualnie rozwijany wierzcholek grafu
  Node* next; // Pomocniczy wskaznik
  list<Node*> toExplore; // Frontier – kolejka priorytetowa
  ```

  Uwagi:

  Funkcja jest implementacją algorytmu Dijkstry służącego do szukania najkrótszej ścieżki w grafie. Algorytm ten został opracowany w 1956 roku przez Edsgera Dijkstrę.

  Autor:

  Ernest Staszuk (2011).

  Ostatnia modyfikacja:

  18 grudnia 2011, usuwanie błędów.

11. DODATEK D. Spis zawartości dołączonych nośników (dyskietek, CD ROMu)

- Umieszczony w katalogu „SRC” gotowy do zbudowania w środowisku programistycznym Microsoft Visual Studio 2008 projekt. Zawiera on kody źródłowe oraz pliki niezbędne do uruchomienia programu (przykładowe konfiguracje oraz biblioteki dołączane dynamicznie) w katalogu „RunEnv”.
- W katalogu „EXE” skompilowana aplikacja wraz z nagranymi wcześniej przykładowymi filmami wejściowymi (w folderze „Data”), oraz plikami mapa.graph dla I piętra budynku C3 Akademii Górniczo-Hutniczej w Krakowie, plikami z markerami w formacie biblioteki ARToolKit, przykładowym plikiem konfiguracyjnym kamery (WDM_camera.xml) oraz plikiem „markery.pdf” zawierającym gotowe do wydrukowania markery, których używaliśmy przy testowaniu aplikacji.
- W katalogu „UTLS” program „mk_patt” z biblioteki ARToolKit służący do zapisywania markerów w formacie umożliwiającym ich wykorzystanie w programie.
- W katalogu „DOC” niniejszy raport w formatach PDF oraz DOCX.