Artur KOZIEJ, Andrzej KUŁAKOWSKI

DYDAKTYCZNY MODEL POJAZDU AUTONOMICZNEGO STEROWANEGO ZNACZNIKAMI

Streszczenie

W artykule omówiony został problem konstruowania i oprogramowania dydaktycznego modelu pojazdu autonomicznego sterowanego przy pomocy znaczników. Zaprezentowano konstrukcję pojazdu. Opisano architekturę oprogramowania sterującego i rozpoznającego znaczniki, oraz problem planowania ruchu pojazdu. Przeprowadzono testy ruchowe pojazdu oraz szybkości przetwarzania informacji sterujących.

WSTĘP

Rozwój technologii komputerowych doprowadził do powstania pojazdów autonomicznych, które zaczynają się poruszać po naszych drogach. Oczywiście wzbudza to zainteresowanie nie tylko realizujących takie projekty inżynierów, potencjalnych klientów, ale również przyszłych adeptów budowania takich maszyn. Dostępne „proste" technologie komputerowe osiągnęły już poziom, na którym nawet posiadając niewielkie środki finansowe można zbudować mały pojazd o coraz lepszych możliwościach. Niniejszy artykuł przedstawia opis takiego modelu pojazdu autonomicznego, którym można sterować za pomocą dwuwymiarowych znaczników [23].

Wiele obecnie istniejących na rynku pojazdów - robotów wykorzystuje znaczniki, ale często nie pozwala to na swobodne poruszanie się, np. roboty typu line follower [7, 24]. W pracy przyjęto założenie, że znacznik graficzny może dostarczyć informacji o swoim położeniu względem pojazdu - robota i o samym identyfikatorze. Możemy więc dowiedzieć się, czy znacznik został wykryty, jaki ma identyfikator oraz jaka jest jego pozycja i obrót względem osi pojazdu. Dla porównania, stosowana w robotyce metoda kontaktu zbliżeniowego zakłada jedynie odczyt detekcji znacznika (np. kodu kreskowego), co daje nam wartość kodu identyfikatora i informację o jego wykryciu, a więc dużo mniej informacji [18].

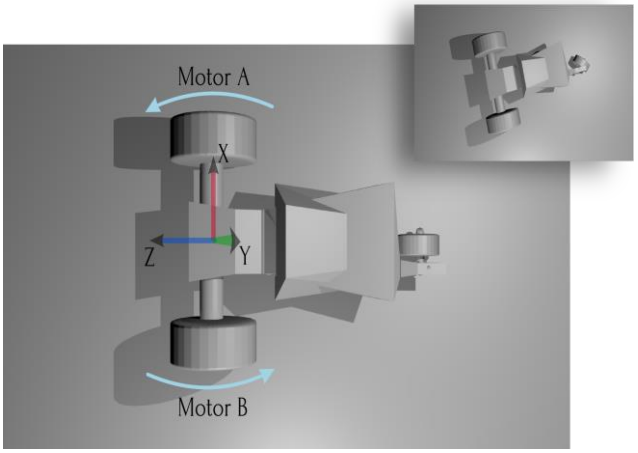
Pomysł realizacji pracy zaczerpnięto z możliwości sterowania wirtualnymi modelami 3D wykorzystywanymi w grach komputerowych do kontrolowania zachowania obiektów, gdy te oddziałują na siebie. Podstawą do swobodnego ruchu tego typu wehikułu jest posiadanie informacji z rozpoznanych znaczników graficznych, podobnie jak w aplikacjach z rozszerzoną rzeczywistością [21].

Podstawą do swobodnego ruchu pojazdu - robota sterowanego znacznikami jest brak ograniczeń motorycznych (zależnych tylko od jego konstrukcji) oraz wykorzystanie informacji z rozpoznanych znaczników graficznych. Informacje te potraktowane jako zdarzenia mają sprawić, że robot będzie mógł reagować na nie podobnie jak w grach komputerowych [10].

1. KONSTRUKCJA MODELU POJAZDU

Jako model pojazdu zaproponowano platformę jeżdżącą, która składa się z trzech kół [17], z czego dwa są połączone z silnikami nazywanymi A i B. Taka konstrukcja pozwala na prosty i swobodny system sterowania jazdą. Pozwala ponadto na obrót wehikułu w miejscu, a zatem na pozbycie się problemów z promieniem skrętu [17]. Widoczny model platformy jeżdżącej na rysunku 1 bardzo dobrze uwidacznia jego możliwości ruchowe. W tym układzie pojazd porusza się względem osi Z (do przodu i do tyłu), a obraca względem osi Y, jak widać na lokalnym układzie współrzędnych. Zatem przestrzeń konfiguracyjna (przestrzeń parametrów) zawiera się w: (x,y,α), gdzie x i y oznaczają współrzędne punktu odniesienia, a α kąt obrotu [8].

Na rysunku 1 pokazano także mniejszy widok z góry, gdzie widoczny jest efekt działania obrotu kół podłączonych do silników A i B zgodnie ze strzałkami.



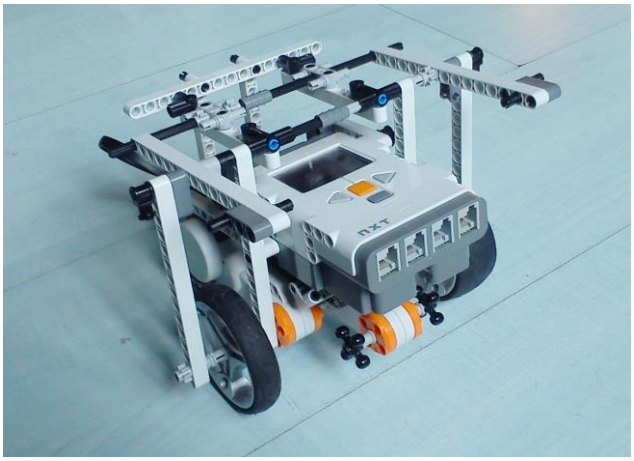
Rys. 1. *Trójkołowy układ jezdny [13]*

Model pojazdu został skonstruowany na bazie zestawu LEGO Mindstorms. Klocki te oferują możliwość budowania różnych modeli wehikułów wykorzystujących dołączone części budownicze, silniki, sensory i sterującą kostkę NXT (lub podobny element sterujący). Istnieje również możliwość zbudowania aplikacji sterującej dla takiego pojazdu przy użyciu oprogramowania dostarczanego przez producenta (LEGO Mindstorms Education NXT Software) [25]. Konstrukcja została zaprezentowana na zdjęciu z rysunku 2.

Model posiada dwa duże koła napędzane silnikami i jedno małe, poruszające się swobodnie. Duże koła są twarde i mają niski bieżnik. W rezultacie daje to mniejsze tarcie i większe przeniesienie siły napędowej. Na górze widać skonstruowaną płaską platformę, mającą na celu umieszczenie komputera, tabletu lub kamery.

2. STEROWANIE MODELEM POJAZDU

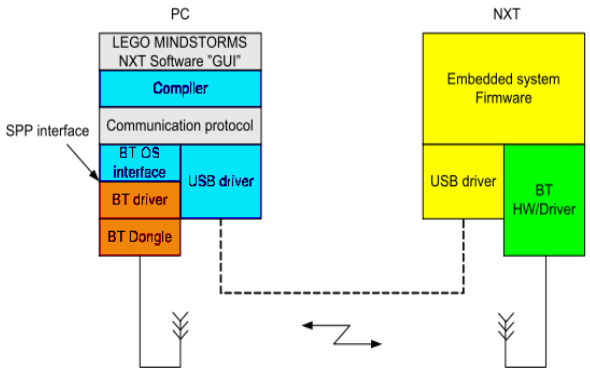
Do sterowania pojazdem niezbędna jest komunikacja z układem sterującym. W tym modelu niskopoziomowym układem sterującym jest kostka NXT. Połączenie między komputerem, a kostką NXT jest możliwe na dwa sposoby: poprzez interfejs USB lub interfejs Bluetooth – rysunek 3. Wykorzystując jedną z tych metod komunikacji można wysłać komendy z komputera do modułu NXT lub wgrać gotowy program, który zostaje zapamiętany i wykonany [1,15].



Rys. 2. *Zdjęcie pokazujące model pojazdu z perspektywy*

Sterowanie zbudowanym pojazdem ma następujący przebieg:

1. Program sterujący na komputerze tłumaczy akcję na komendę.
2. Komenda zostaje wysłana z komputera za pomocą interfejsu USB lub Bluetooth.
3. Następuje odbiór komendy poprzez interfejs kostki NXT.
4. Komenda zostaje przetworzona i zamieniona na sygnały sterujące efektorami przez wbudowany system pojazdu.

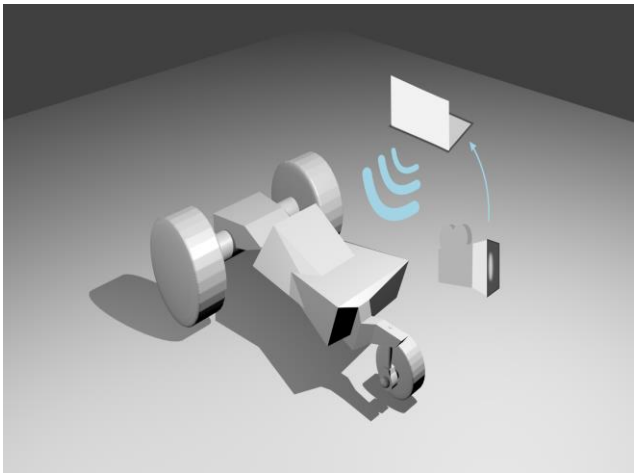


Rys. 3. *Diagram blokowy komunikacji komputera z modułem* NXT [1]

W celu ułatwienia sterowania modelem i uproszczenia konstrukcji jego programu sterującego wykorzystano bibliotekę MonoBrick [4]. Biblioteka ta to zestaw klas służących do komunikacji i sterowania pojazdem. Często jeden element lub podsystem wehikułu ma swój odpowiednik w klasie. Dzięki architekturze zorientowanej obiektowo za większość akcji jest odpowiedzialna jedna metoda lub wywołanie kilku w określonej sekwencji [4, 16]. Metody klas (np. klasy Motor) opakowują niskopoziomowe komendy dostarczane przez producenta układu NXT.

3. ARCHITEKTURA APLIKACJI

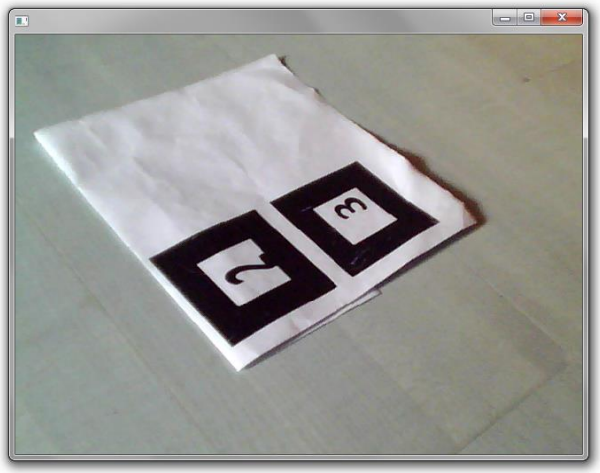
W systemie modelu pojazdu można wyróżnić trzy podsystemy: kamera, komputer i platforma jeżdżąca - rysunek 4. Aby wyjaśnić pracę poszczególnych podsystemów, przedstawiono je na podstawie ogólnego toku przetwarzania informacji. Początkiem jest podsystem kamery dostarczający wizję, następnie komputer przetwarza przechwycone klatki obrazu z kamery i poddaje je analizie w celu wykrycia znaczników. Aplikacja sterująca na podstawie informacji zebranych z analizy, podejmuje decyzję, jakie komendy należy przesłać układowi NXT. Moduł NXT będący kontrolerem, odbiera komendy i wykonuje je sterując końcowymi elementami platformy jeżdżącej (w chwili obecnej są to silniki).



Rys. 4. *Ogólny schemat współdziałania podsystemów modelu pojazdu autonomicznego [13]*

Architektura aplikacji została zaprojektowana tak, aby oddzielić sterowanie robotem od zbierania informacji o znacznikach. Do tego celu zostały zaprojektowane dwa programy spełniające wymagania podziału zadań.

Pierwszy z omawianych to program przechwytujący klatki z kamery i analizujący je w celu dostarczenia informacji o znacznikach graficznych - rysunek 5. Na początku, gdy następuje inicjalizacja, program łączy się z kamerą poprzez interfejs USB i wykorzystuje pamięć współdzieloną oraz wczytuje wzorce (znaczniki) do rozpoznania.



Rys. 5. *Znaczniki w przestrzeni 3D; Zdjęcie znaczników widocznych pod kątem [13]*

Drugi program realizuje zadanie sterowania pojazdem. Z punktu widzenia architektury systemu można go umieścić między pierwszym programem dostarczającym informacje o znacznikach, a kostką NXT. To właśnie w nim podejmowana jest decyzja jakie komendy przesłać do modułu NXT, aby poprawnie wysterować układ jezdny.

Diagram na rysunku 6 prezentuje przepływ danych między kolejnymi podsystemami. Wyjaśnia również, w jaki sposób dane zmieniają swoją postać po przejściu przez kolejne elementy przetwarzające.

Rys. 6. *Diagram przepływu danych*

4. APLIKACJA ROZPOZNAWANIA ZNACZNIKÓW

W podsystemie rozpoznawania znaczników w obrazie został wykorzystany algorytm biblioteki ARToolKit [3, 5, 26]. Podstawy jego działania opierają się na wykryciu czarnych pogrubionych krawędzi kwadratu znacznika w kolejnych klatkach przechwytywanych przez kamerę – rysunek 5.

Pojazd powinien interpretować otoczenie pod wieloma względami. Odczytywanie znaczników graficznych jest swego rodzaju ułatwieniem zadania interpretacji otoczenia poprzez uproszczone obiekty, jakimi są znaczniki składające się z grubych czarnych krawędzi i identyfikatora graficznego w środku. Jednak ten sposób interpretacji bardzo ogranicza zbiór obiektów podlegających rozpoznaniu.

Potok przetwarzania informacji przez bibliotekę można opisać w następujących krokach [12,20]:

1. Przechwycenie klatki obrazu z kamery.
2. Analiza i przetwarzanie obrazu z klatki w celu znalezienia znacznika i wyznaczenia jego przekształceń.
3. Identyfikacja znacznika na podstawie symbolu wewnątrz kwadratu.
4. Ustawienie obiektu na scenie 3D według przekształceń znacznika.
5. Zmiksowanie obrazu rzeczywistego z wirtualnym obiektem 3D i odrysowanie go w oknie programu.

Do rozpoznawania złożonych obiektów jakie występują w naturalnym świecie można stosować metody oparte o cechy lokalne obrazu. Porównanie więc polega na analizie cech lokalnych obrazów wzorcowych z tymi, które zostały znalezione na obrazie poddawanemu rozpoznaniu. Przykłady metod opartych o cechy lokalne [6] to: Edge Detection (lub Edge Histogram) i SIFT (Scale invariant feature transform) – cechą lokalną jest gradient zmian jasności pikseli wokół punktu charakterystycznego.

Stosując takie metody można nauczyć pojazd - robota dużo szerszej interpretacji jego otocznia. Wehikuł byłby w stanie odczytywać napisy, numery pokoi na drzwiach, rozpoznawać samochody, twarze, itd. [6].

5. PLANOWANIE RUCHU POJAZDU

Planowanie ruchu pojazdu polega na odpowiednim sterowaniu wehikułem tak, aby był zdolny przebyć drogę z jednego miejsca do drugiego lub wykonać ruch swoimi elementami. Wykonanie obydwu tych czynności powinno być bezkolizyjne i bezpieczne [19].

Pojazd taki zawsze porusza się według jakiegoś planu. Nawet jeśli nie został on dostarczony w całości, to plan jest tworzony dynamicznie na podstawie rozpoznanych obiektów, tutaj znaczników [8].

Generalnie istnieją dwie metody planowania ruchu pojazdu-robota różniące się pod względem znajomości przestrzeni, po której wehikuł ma się poruszać. Pierwsza metoda to metoda globalna, w której wszystkie przeszkody są znane przed wykonaniem planu. Drugą metodą jest metoda lokalna, w której pojazd zna tylko lokalne przeszkody, więc musi rozwijać plan dynamicznie dzięki informacjom z sensorów [22, 19]. Różnica między dwoma metodami planowania polega na tym, że znając plan całości otoczenia, jesteśmy w stanie z góry zaplanować całą ścieżkę ruchu między dowolnymi punktami według algorytmu planowania drogi, który utworzy optymalną trasę. W przypadku dynamicznego poznawania otoczenia pojazd musi najpierw „nauczyć się", jak się w nim poruszać. W tym projekcie zastosowano metodę lokalną.

Do obliczenia reprezentacji przestrzeni swobodnej [17, 9], która pozwoli wyznaczyć bezkolizyjną ścieżkę od punktu początkowego do końcowego wykorzystano mapę trapezową [13, 8].

6. ALGORYTM STEROWANIA SILNIKAMI

Program aplikacji sterującej realizuje zadanie poruszania pojazdem. Może to zrobić poprzez odpowiednie sterowanie użytymi silnikami. Algorytm sterowania silnikami składa się z następujących elementów [13]:

Dane wejściowe:

$M_{4,4}$ macierz transformacji znacznika (względem lokalnego układu współrzędnych wehikułu),

zmienna *Visible* określająca stan wykrycia znacznika, przyjmując wartości typu boolean,

A i *B* - oznaczenia silników pojazdu jak na modelu z rysunku 2,

stała *STOP_Z* będąca warunkiem stopu wyrażonym w odległości od celu,

stała maksymalnej prędkości pojazdu *MAX_SPEED.*

Funkcje:

ScaleToCentimeters - skalowanie jednostek translacji macierzy *M* do centymetrów,

SetMotorSpeed - ustawienie prędkości danego silnika,

MotorStop - zatrzymanie danego silnika.

Wynik:

wysterowanie prędkości silników *A* i *B* lub ich całkowite zatrzymanie.

Kroki algorytmu (pseudokod) [13]:

1. Zapisanie translacji macierzy *M* względem osi X i Z do zmiennych *x* i *z*: *x = M[0][3], z = M[2][3]*

2. Przeskalowanie wartości zmiennych *x* i *z* do centymetrów:
x = ScaleToCentimeters(x), z = ScaleToCentimeters(z)

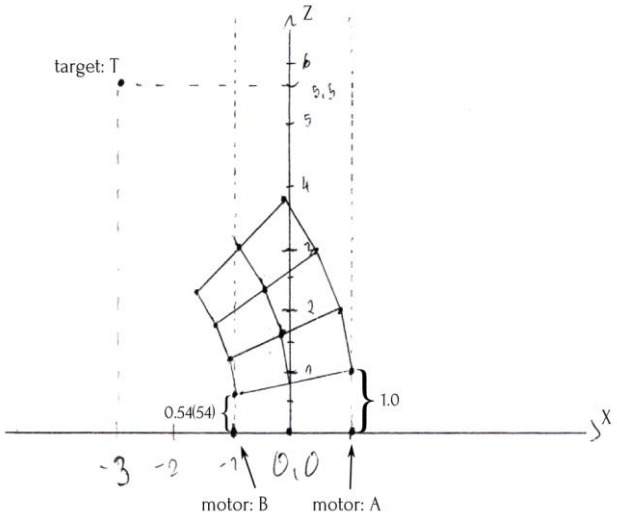
3. **if** *z >= STOP_Z* **then** cel osiągnięty

4. **if** *Visible* **then** wyznaczenie ogólnej prędkości (*speed*) pojazdu na podstawie odległości od celu:
speed = z
if *speed > MAX_SPEED* **then** *speed = MAX_SPEED*
wysterowanie silnikami *A* i *B* w zależności po której stronie osi X znajduje się cel:

```
if x < 0 then
  SetMotorSpeed(A,speed),
  SetMotorSpeed(B,speed + (speed*(x/z)))
else SetMotorSpeed(A,speed - (speed*(x/z))),
  SetMotorSpeed(B,speed)
```

else
w przeciwnym wypadku silniki zostają zatrzymane:
MotorStop(A), MotorStop(B)

Na rysunku 7 przedstawiono wykres obrazujący efekt działania algorytmu, na przykładzie sterowania silnikami *A* i *B*. Cel do którego zmierza pojazd znajduje się w punkcie T o współrzędnych (-3, 5.5). Osie na rysunku są tożsame z osiami X i Z. Rysunek przedstawia widok z góry. Gdy w algorytmie sterowania silnikami podstawimy współrzędne (-3, 5.5) do zmiennych *x* i *z* oraz ustawimy prędkość na 1, to w wyniku działania algorytmu otrzymujemy 1 dla silnika A i 0,54(54) dla silnika B, przy pierwszym przejściu przez algorytm. Wyniki pierwszego i kolejnych trzech przejść są zobrazowane na rysunku, jednak należy pamiętać, że za każdym razem istnieje potrzeba aktualizacji zmiennych *x* i *z*, gdyż przechowują zmieniającą się pozycję celu.



Rys. 7. *Przesunięcia kół pojazdu według algorytmu sterowania silnikami, gdzie: T - cel, A - prawy silnik, B - lewy silnik [13]*

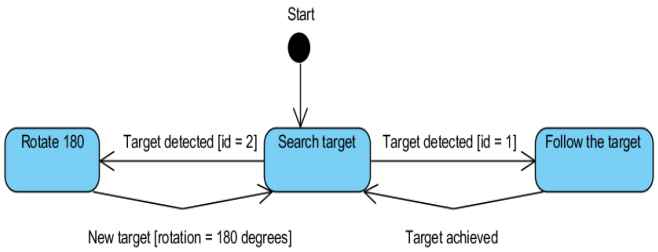
7. APLIKACJA STERUJĄCA

Kontrolowanie pojazdu zmienia się dynamicznie wraz ze zmieniającą się sytuacją. Wehikuł odbiera zdarzenia poprzez kamerę, którą w połączeniu z programem dostarczającym dane o znacznikach można potraktować jak sensor zgłaszający zdarzenia. Tak więc, każde zdarzenie musi zostać obsłużone przez program sterujący, co w wyniku powoduje zmianę jego stanu wewnętrznego. Dobrym przykładem obsługi zdarzenia jest sytuacja, w której został rozpoznany znacznik (pod identyfikatorem numer 2) obrotu o 180 stopni, wtedy pojazd musi podjąć akcję obracania względem osi Y, pozostając w tym stanie do momentu uzyskania łącznego obrotu 180 stopni.

Aby wehikuł rzeczywiście mógł odpowiednio się zachować w odpowiedzi na zdarzenia, zostały zdefiniowane trzy stany [13]:

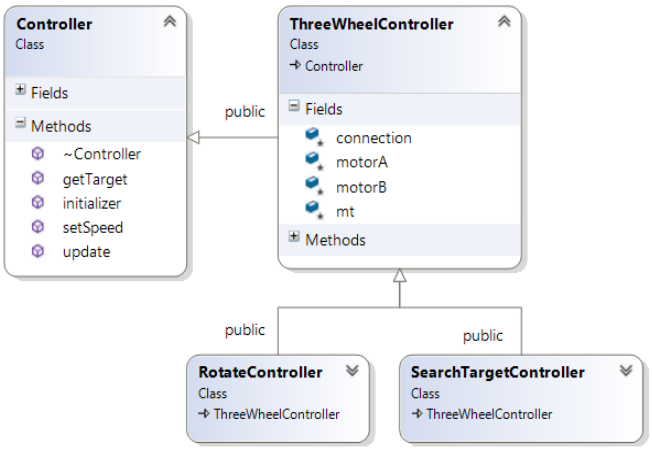
1. Szukanie celu (*Search target*) – stan, który jest podejmowany jako pierwszy, lub gdy poprzednie zadanie zostało wykonane. Szukanie odbywa się poprzez obrót co kilka stopni, tak aby kamera zdążyła za każdym obróceniem przechwycić obraz bez rozmycia.
2. Podążanie za celem (*Follow the target*) – stan jest podejmowany, gdy w stanie *Search target* nastąpi wykrycie znacznika o identyfikatorze numer 1, a jego końcem jest dojechanie do celu, czyli ustwieniem pojazdu w odpowiedniej pozycji względem znacznika [2].
3. Obrót o 180 stopni (*Rotate 180*) – stan jest podejmowany, gdy w stanie *Search target* nastąpi wykrycie znacznika o identyfikatorze numer 2, wtedy koła są obracane, aż do momentu osiągnięcia obrotu robota wynoszącego 180 stopni względem osi Y.

Diagram stanów robota (rysunek 8) prezentuje opisane wcześniej stany i możliwości ich automatycznych zamian. Gdy zaistnieje potrzeba obsłużenia nowych sytuacji, w których może się znaleźć pojazd powyższy diagram będzie należało rozbudować o nowe stany i przejścia.



Rys. 8. *Diagram stanów pojazdu [13]*

Klasy kontrolerów - rysunek 9 - zostały zaprojektowane według wzorca *State* [10], gdyż w ten sposób udało się odzwierciedlić stany pojazdu-robota i przejścia między nimi (rysunek 8).



Rys. 9. *Diagram klas kontrolerów [13]*

Model klasy oparty o wzorzec *State*, składa się z abstrakcyjnej klasy *Controller*, z której wywodzą się trzy klasy konkretne. Najważniejszą metodą interfejsu klasy *Controller* jest metoda *update*, której definicja w klasach pochodnych ma zawierać algorytm sterujący silnikami.

ThreeWheelController jest odpowiednikiem stanu *Follow the target*. Adoptuje trójkołowy układ jezdny. Przeznaczeniem tej klasy jest więc, sterowanie silnikami pojazdu tak aby, dojechać do celu. W skład atrybutów wchodzą dwa pola typu *Motor*. Szybkość zmierzania wehikułu robota do celu jest uzależniona od odległości od celu.

Klasa *RotateController* będąca odpowiednikiem stanu *Rotate 180* oraz klasa *SearchTargetController* będąca odpowiednikiem stanu *Search target* dziedziczą po klasie *ThreeWheelController*. Mają więc dostęp do pól *motorA* i *motorB* typu *Motor* i posiadają własne implementacje metody *update* zmieniając tym samym zachowanie klasy według swojego przeznaczenia.

Tak zaprojektowana hierarchia klas kontrolerów spełnia zasadę open-closed principale, co oznacza otwartość kodu na rozszerzenia [11]. Poprzez oparcie się na abstrakcyjnym interfejsie, wspólnym dla

wszystkich kontrolerów, mamy łatwość tworzenia i podmieniania ich na inne, w celu rozszerzenia funkcjonalności.

Ponadto zbudowana została także aplikacja KeyboardController, której zadaniem jest umożliwienie sterowania modelem pojazdu bez wykorzystywania części systemu odpowiadającej za rozpoznawanie znaczników graficznych [13].

8. TEST PRECYZJI POZYCJONOWANIA

Aby zweryfikować poprawność działania systemu przeprowadzono kilka testów. Poniżej zostanie przedstawiony prosty test pozycjonowania mający na celu sprawdzenie jak precyzyjny jest ruch pojazdu.

Zdjęcia z rysunku 10 pokazują test, w którym pojazd miał do przejechania pewien dystans, skręcając po drodze. Dzięki temu widoczne jest jak bardzo rzeczywisty obrót wehikułu będzie odbiegał od oczekiwanego. Pierwsze zdjęcie (A) pokazuje pozycję startową, z której pojazd rusza i zatrzymuje się dopiero w pozycji pokazanej na zdjęciu środkowym (B). Na tym zdjęciu widać pojazd znajdujący się w połowie drogi. Dalej wehikuł rusza i dojeżdża do pozycji końcowej pokazanej na zdjęciu trzecim (C). Pozycję startowa jest zaznaczona żółtą kartką (K) na zdjęciach (B i C), aby można było prześledzić przebytą drogę. Można zauważyć, że pojazd skręcił, aby dojechać i zatrzymać się w połowie drogi, następnie jadąc do końca skręcił, wracając do swojego początkowego ułożenia, co z kolei widać na pozycji końcowej (zdjęcie C). Do sterowania wehikułem w teście został wykorzystany program KeyboardController.

Wynik testu został wyrażony jako rzeczywisty kąt obrotu pojazdu względem obrotu oczekiwanego. Wartość niedokładności wynosi ok. 3 stopnie. Niedokładność jest niewielka co oznacza, że tak sterowany pojazd będzie mógł wykonywać zadania wymagające pewnej precyzji.

9. TEST DZIAŁANIA APLIKACJI ROZPOZNAJĄCEJ

Drugi z eksperymentów sprawdził wydajność detekcji i przetwarzania informacji ze znaczników. Test ten w kontekście funkcjonowania pojazdu - robota pokazał, po jakim czasie nastąpi aktualizacja danych, a tym samym stanu wehikułu. Wynikowy czas jest wystarczająco krótki, aby pojazd mógł reagować w czasie rzeczywistym na zmiany obrazu z kamery. W najgorszym przypadku opóźnienie wynosiło 0,09 sekundy.

Dane zebrane podczas testów działania pojazdu i jego oprogramowania wykorzystano dodatkowo do przeprowadzenia wstępnej wersji symulacji działania wielu robotów. Tę część działań opisano w [13].

PODSUMOWANIE

Konstrukcja modelu pojazdu autonomicznego i implementacja jego oprogramowania pozwala na podążanie za celem oraz wykonywanie poleceń zależnych od zdarzeń generowanych poprzez znaczniki graficzne. Dodatkowo można zrealizować inne zadania, wykraczające poza podstawowe założenia projektu.

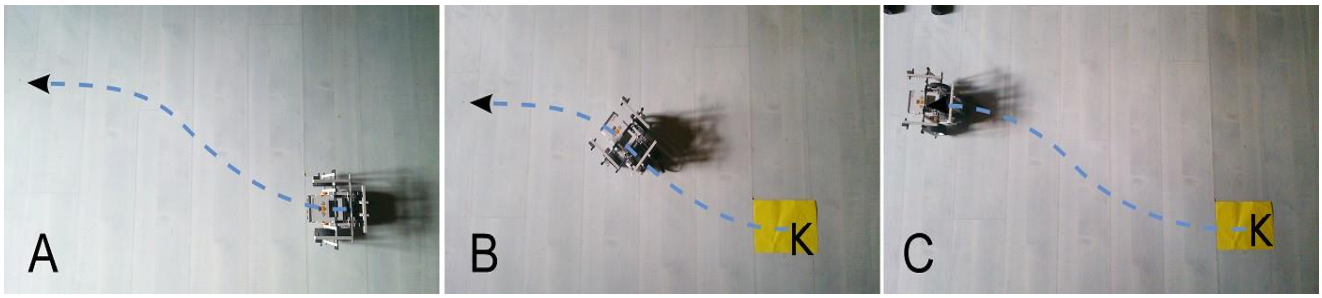
Obecna konstrukcja i sterowanie modelem z powodzeniem pozwala na swobodny ruch, dlatego w tym miejscu warto zastanowić się nad rozszerzeniem jego interakcji z otoczeniem. Dobrym przykładem są chwytaki [14], dzięki którym pojazd-robot byłby w stanie przenosić przedmioty. Wyspecjalizowanie interakcji wehikułu, jak w przykładzie z chwytakami, pozostaje więc kwestią uzależnioną od rodzaju wykonywanych zadań.

Funkcjonalność robota można dynamicznie rozwijać, m.in. dzięki elastycznemu modelowi klas kontrolerów, które kontrolują zachowanie pojazdu w zależności od jego stanu. Aby wehikułowi dodać nową reakcję na zaistniałe zdarzenie, wystarczy jedynie dodać nowy stan, poprzez implementację metody update klasy dziedziczącej po ThreeWheelController. Architektura wieloprocesowa zaś, pozwala łatwo podmieniać części systemu nawet podczas jego pracy, jak to ma miejsce z programem dostarczającym dane o znacznikach i aplikacją KeyboardController do szybkiego testowania (dostarczającym dane poprzez wprowadzanie pozycji celu z klawiatury).

Zbudowany model pojazdu można wykorzystać w dydaktyce w celu realizacji projektów mających na celu rozbudowę platformy jeżdżącej lub badań własności algorytmów sterowania i rozpoznawania. Może on też stanowić podstawę do fizycznych symulacji działań robotów przemysłowych oraz pojazdów autonomicznych.

BIBLIOGRAFIA

1. Appendix 1 - LEGO MINDSTORMS NXT Communication protocol, 2006. The LEGO Group. (dokument techniczny)
2. Application of Fuzzy-Linguistic Geometry on Robot 3D Path Planning Autorzy: S. Khanmohammadi, A. Aghagolzadeh, M. A. Badamchi Zadeh, S. Ghaemi, 2002.
3. ARToolKit Documentation Copyright © 2004-2006. HIT Lab NZ.
4. Biblioteka MONOBRICK http://www.monobrick.dk/doc/cplusplus/projects/NXT%20C++%20Bluetooth%20Library/project-summary.html, 07.09.2014
5. Computer Vision Algorithm http://www.hitl.washington.edu/artoolkit/documentation/vision.htm, 10.07.2014
6. Computer Vision: Algorithms and Applications, Richard Szeliski, September 3, 2010
7. Cook D., *Robot Building for Beginners*, 2nd Edition (Technology in Action). Wydawnictwo, Apress, 2010
8. de Berg M., van Kreveld M., Overmars M., Schwarzkopf O., Geometria obliczeniowa Algorytmy i zastosowania, Wydawnictwo Nukowo-Techniczne 2007
9. Development of a Robot Localization and Environment Mapping System, Author: Panas, Cynthia Dawn Walker, Publisher: Massachusetts Institute of Technology Date Issued: 2012 http://dspace.mit.edu/handle/1721.1/73805#files-area, 27.08.2014
10. Game Engine Design 3: States, State Diagrams & the State Pattern. Autor: Daniel Livingstone, 2012.



Rys. 10. *Zdjęcia przebytej drogi przez pojazd [13]*

11. Game Engine Design 6: Strategies & Factories. Autor: Daniel Livingstone, 2012.
12. How does ARToolKit work? http://www.hitl.washington.edu/artoolkit/documentation/userarwork.htm, 16.06.2014
13. Koziej A., Opracowanie aplikacji sterującej ruchem robota mobilnego kierowanego znacznikami, praca magisterska, Kielce, 2014
14. LEGO MINDSTORMS User Guide, 2006. The LEGO Group. (instrukcja użytkownika)
15. LEGO MINDSTORMS NXT Bluetooth Developer Kit, 2006. The LEGO Group. (dokument techniczny)
16. LEGO MINDSTORMS NXT Direct commands, 2006. The LEGO Group. (dokument techniczny)
17. Planning Algorithms, Steven M. LaValle, University of Illinois 2006
18. QR code http://en.wikipedia.org/w/index.php?title=QR_code&oldid=489124271. 2012-04-25.
19. Roboty Mobilne – artykuł. Aktualizacja: 20.07.2007, http://artelis.pl/artykuly/3738/Roboty-mobilne, 30.05.2014
20. Strona główna projektu ARToolKit http://www.hitl.washington.edu/artoolkit/, 25.07.2014
21. Supan P., Stuppacher I., Image Based Lighting in Augmented Reality, CESCG, Wiedeń, 2006
22. Systemy robotów autonomicznych, mgr inż. Andrzej Opaliński, Akademia Górniczo-Hutnicza, 12.06.2013, Kraków
23. Unmanned ground vehicle; https://en.wikipedia.org/wiki/Unmanned_ground_vehicle, 21.10.2015
24. Vartakavi A., Building Autonomous Line Followers with Arduino! za:http://www.mbrobotgames.ca/sites/default/files/GuideToBuildAutoRobots.pdf (listopad 2014)
25. Zestaw LEGO MINDSTORMS, http://www.educatec.ch/about/presse/LEGO_MINDSTORMS_NXT/LEGO_MINDSTORMS_NXT_Sys/contentimage/, 26.07.2014
26. Źródło biblioteki ARToolKit http://sourceforge.net/projects/artoolkit/files/artoolkit/2.72.1/, 20.09.2014

DIDACTIC MODEL OF AUTONOMOUS VEHICLE CONTROLLED WITH MARKERS

Abstract

The paper presents the problem of constructing and programming an autonomous vehicle model controlled with markers. The construction of the vehicle has been presented. There has been described architecture of control and recognition marker software as well as the problem of planning the vehicle motion. There have been carried out tests of vehicle mobility and speed of control information processing.

Autorzy:
mgr inż. **Artur Koziej** – Intel, Gdańsk
dr inż. **Andrzej Kułakowski** – Politechnika Świętokrzyska