Krzysztof MAŁECKI, Sławomir JASZCZAK, Rafał SOKOŁOWSKI
ZACHODNIOPOMORSKI UNIWERSYTET TECHNOLOGICZNY W SZCZECINIE
ul. Żołnierska 49, 71-210 Szczecin

Synteza sprzętowa i programowa symulatora sterowania ruchem pojazdów na określonym obszarze

Dr inż. Krzysztof MAŁECKI

Od 1997 jako asystent w Instytucie Informatyki Politechniki Szczecińskiej prowadził badania dotyczące projektowania układów. Od 2001 zatrudniony na stanowisku adiunkta w ZUT w Szczecinie. Bieżące zainteresowania naukowe związane są z modelowaniem i symulacją ruchu drogowego. W szczególności modelowanie procesów z zastosowaniem automatów komórkowych.

e-mail: kmalecki@wi.zut.edu.pl

Dr inż. Sławomir JASZCZAK

Od 1994 jako asystent na Wydziale Techniki Morskiej Politechniki Szczecińskiej prowadzi badania w zakresie sterowania pojazdami głębinowymi z wykorzystaniem metod sztucznej inteligencji. Od 2002 zatrudniony na stanowisku adiunkta w Zakładzie Sztucznej Inteligencji. Bieżące zainteresowania wiążą się z implementacją złożonych algorytmów sterowania dyskretnego i cyfrowego na platformie wykonawczej PLC.

e-mail: sjaszczak@wi.zut.edu.pl

Streszczenie

W niniejszym artykule przedstawiono wieloelementowe, sprzętowo-programowe środowisko symulacyjne do sterowania ruchu drogowego. Główną ideą było wytworzenie łatwo rozbudowywalnego, elastycznego symulatora ruchu drogowego umożliwiającego prowadzenie badań z zakresu sterowania ruchem drogowym. Wykorzystano do tego specjalnie opracowane makiety obszarowe (różne rodzaje skrzyżowań dróg), sterowniki PLC oraz opracowano program komputerowy interpretujący dane pochodzące z makiet.

Słowa kluczowe: symulacja ruchu drogowego, sterowniki PLC.

Hardware and software synthesis of a traffic control simulator for a specific area

Abstract

The multi-component, hardware and software simulation environment for traffic control is presented in this paper. The main idea was to create easy extendable, flexible simulation facility for research in the field of traffic control. A specially designed model area (different types of road junctions: a T-type intersection, crossing four roads classical and a typical junction, which is a model of a real system of roads in Szczecin), PLCs and a proprietary computer program that interprets data from the model were used for this purpose. The paper presents the concept of a hardware-software simulator (Fig. 1) together with a description of individual elements (Figs. 2, 3 and 5). It is worth mentioning that the way to develop a program and the layout and type of PLC allows the connection of any mock-ups that will provide the appropriate signals. These signals are interpreted and on the basis of them the so-called scenario will be generated. Finally, the paper presents a real situation that could occur at the T-type intersection (vehicle enters the intersection and cannot leave it). Due to the fact that the models are equipped with the so-called inductive loops, the system offers an appropriate scenario to avoid blocking the intersection. The paper can be concluded with a statement that the developed hardware-software simulator allows conducting research in the field of traffic control in a safe and collision-free way.

Keywords: traffic simulation, Programmable Logic Controller (PLC).

Inż. Rafał SOKOŁOWSKI

Magistrant I roku kierunku Informatyka na Wydziale Informatyki Zachodniopomorskiego Uniwersytetu Technologicznego w Szczecinie. Reaktywator wydziałowego Koła Naukowego "Grupa .NET", jednocześnie aktywny uczestnik spotkań oraz pasjonat innowacyjnych technologii Microsoft. Pośród swoich zainteresowań dzieli także zamiłowanie do mikrokontrolerów oraz ich zastosowaniach w aplikacjach dedykowanych.

e-mail: rasokolowski@wi.zut.edu.pl

1. Wstęp

Skrzyżowania drogowe wybudowane dwadzieścia lat temu nie są przystosowane do tak znacznej ilości użytkowników ruchu drogowego, którzy codziennie przez nie przejeżdzają. Wzrost liczby pojazdów wpływa wprost proporcjonalnie na zmniejszenie przepustowości istniejących skrzyżowań.

W większości przypadków zagęszczenie urbanistyczne miast nie pozwala nawet, na dodanie choćby jednego dodatkowego pasa ruchu na dowolnym z wlotów do skrzyżowania. Przyczyny są różne: ogromne koszta, które wiązałyby się z takim przedsięwzięciem czy choćby fizyczny brak takich możliwości.

Dotychczas najbardziej efektywną metodą zwiększania przepustowości skrzyżowań jest udoskonalanie algorytmów sterowania strumieniami sygnałów obsługujących przepływ pojazdów. Aby prowadzić badania w warunkach bezpiecznych przygotowuje się odpowiednie środowiska programowe lub sprzętowo-programowe (tab.1). Stały rozwój metod symulacyjnych, pozwala na osiąganie coraz bardziej złożonych rozwiązań.

Tab. 1. Sposoby symulowania ruchu drogowego
Tab. 1. Methods for traffic simulation

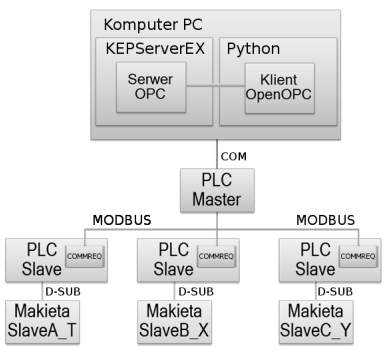
Lp.	Rodzaj symulacji	Zalety	Wady
1	Badania w środ. rzeczywistym	dokładne odzwierciedlenie sytuacji	zagrożenie testujących, koszta, czas
2	Modele matematyczne	tanie, doskonale wspierają pozostałe metody, precyzyjne obliczenia	często założenia są mocno ograniczone w stosunku do rzeczywistych sytuacji
3	Symulacja komputerowa	nie naraża życia ludzkiego, możliwość zamodelowania dowolnej sytuacji	ograniczone wyobraźnią projektanta, często bez wsparcia odpowiednich modeli matematycznych
4	Środowisko symulacyjne wraz z makietami	„złoty środek" pomiędzy symulacją rzeczywistą a komputerową, możliwość testowania dowolnie ustalonej sytuacji	„urojeni" użytkownicy ruchu, kosztowność w przypadku wykorzystania dużej ilości sprzętu (np. sterowników PLC)

Źródło: opracowanie własne na podstawie literatury

2. Realizacja symulatora

Poglądowy schemat symulatora do sterowania ruchem drogowym przedstawiono na rys. 1. Opracowany system składa się z: trzech makiet będących modelami rzeczywistych skrzyżowań różnego typu, czterech sterowników przemysłowych oraz programu napisanego w języku Python. Makiety, których dokładny opis znajduje się w [3] są elementem interakcji użytkownika z systemem i pozwalają wprowadzać dane (sygnały) do systemu oraz obserwować efekty działania programu. Dane wprowadzane są przy wykorzystaniu dwu-stanowych przycisków odpowiadających

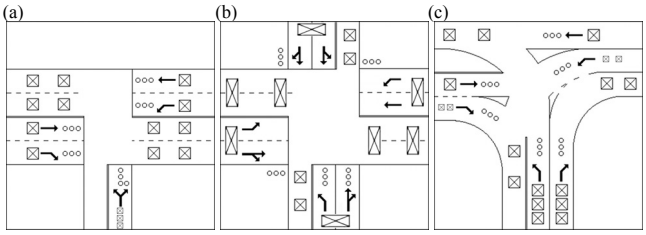
za pojawienie się pojazdu w danej lokalizacji a odczytywane przy pomocy sygnalizatorów świateł umieszczonych na makietach. Na rys. 2 przedstawiono schematy ideowe wszystkich opracowanych makiet: (a) skrzyżowanie typu T – zwane dalej SlaveA_T, (b) skrzyżowanie z czterema wlotami – zwane dalej SlaveB_X, (c) skrzyżowanie typu Y – zwane dalej SlaveC_Y. Dodatkowo zaznaczono możliwe kierunki ruchu, pętle indukcyjne na drogach wlotowych i wylotowych ze skrzyżowań oraz sygnalizatory świetlne. Dla przykładu, sytuacja pokazana na rys. 2a ukazuje pięć możliwych strumieni pojazdów, które należy tak pokierować aby nie doszło do zakleszczenia na skrzyżowaniu, bądź do kolizji a czas przejazdu pojedynczego pojazdu zdążał do minimum. Skrzyżowaniem wykonanym w formie makiety przedstawionej na rys. 2a steruje sterownik realizujący algorytm uruchamiania sygnalizacji świetlnej w taki sposób aby rozkład emitowanych sygnałów zezwalających na przejazd przez skrzyżowanie zapewnił bezkolizyjności kolejnych potoków pojazdów. W tym przypadku wydzielono trzy fazy sygnalizacyjne, gdzie zgodnie z [1] fazą sygnalizacyjną jest czas obejmujący wszystkie sąsiadujące ze sobą przedziały sygnalizacyjne, w których dla określonego zbioru strumieni ruchu nadawany jest sygnał zielony. Zatem pozostałe makiety są także sterowane w taki sposób aby zapewnić bezkolizyjny tor jazdy dla każdego strumienia pojazdów, który otrzymuje sygnał zezwalający na przejazd przez skrzyżowanie.



Rys. 1. Schemat logiczny opracowanego środowiska
Fig. 1. Logic diagram of the prepared environment

W przypadku makiety przedstawionej na rys. 2b zrealizowano skrzyżowanie o czterech wlotach z dwoma pasami ruchu. Aż trzy z nich są zarządzane tylko jednym sterownikiem. W tym przypadku wydzielono cztery różne fazy sygnalizacyjne pozwalające na przepuszczenie przez skrzyżowanie poszczególnych strumieni pojazdów. Makieta SlaveC_Y (rys. 2c) przedstawia układ trójwlotowego skrzyżowania typu Y, na wzór skrzyżowania ulic Derdowskiego oraz Witkiewicza w Szczecinie. Każdy z wlotów składa się z dwóch pasów. Każdy z pasów ruchu posiada indywidualny zestaw sygnalizatorów świetlnych. W tym przypadku sterownik realizuje program trój-fazowy.

Warto jeszcze wytłumaczyć sens implementacji pętli indukcyjnych znajdujących się na drogach wylotowych z poszczególnych skrzyżowań. Umożliwiają one rozpoznanie sytuacji, w której każdy dodatkowy pojazd wjeżdżający na skrzyżowanie nie będzie w stanie go opuścić, co w konsekwencji powoduje blokadę drogi dla innych użytkowników nadjeżdżających z kolejnego wlotu.

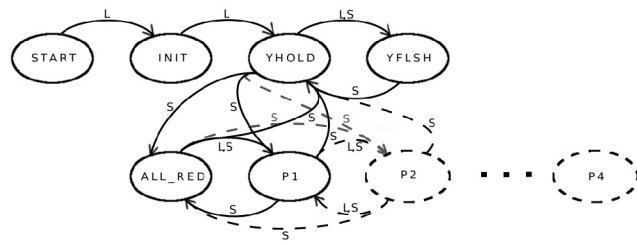


Rys. 2. Schematy makiet wykorzystanych w symulatorze
Fig. 2. Diagrams of mock-ups used in the simulator

Makiety są wyposażone w dwa porty (D-SUB25), za pomocą których są podłączone do sterowników przemysłowych (sterowniki GE VersaMax Micro [4]). Każdy z portów charakteryzuje się dwudziestoma sygnałami dyskretnymi oraz odpowiada kolejno za dane wchodzące i wychodzące do/z makiety. Sterowniki podrzędne, komunikując się w protokole MODBUS RTU [5], łączą się ze sterownikiem nadrzędnym VersaMax. Sterownik nadrzędny jest połączony z komputerem PC, poprzez port RS232c. Aplikacja napisana w języku Python, wykorzystując bibliotekę OpenOPC oraz serwer KepwareOPC, wymienia dane ze sterownikiem nadrzędnym, który z kolei pobiera i rozsyła dane do sterowników podrzędnych z wykorzystaniem sieci szeregowej [6].

Programowalne sterowniki PLC zostały użyte w prezentowanym rozwiązaniu, gdyż umożliwiają stosunkowo łatwą implementację algorytmów sterowania logicznego z wykorzystaniem dedykowanych języków np. LD, IL, FBD i innych ujętych w normie IEC 61131-3, bez konieczności prowadzenia syntezy na poziomie sprzętowym. Należy zwrócić uwagę, że dotyczy to głównie funkcji liczników czasu, zdarzeń, funkcji logicznych oraz realizacji operacji sekwencyjnych. Dodatkowo programowalne sterowniki przemysłowe umożliwiają swobodne tworzenie rozległych sieci w oparciu o standard RS485 (2 wire lub 4 wire) w przypadku sterowników klasy mikro, które w praktycznych rozwiązaniach tj. sterowania ruchem drogowym spełniają funkcje pomocnicze jako koncentratory danych i/lub elementy wstępnie przetwarzające dane. Zaproponowana architektura systemu odpowiada w istocie strukturze dwupoziomowej ze sterownikami podrzędnymi w warstwie bezpośredniej DDC (ang. Direct Digital Control) oraz sterownikiem nadrzędnym i komputerem z serwerem OPC i klientem OPC w warstwie nadrzędnej SC (ang. Supervisory Control). Tak zdefiniowany system umożliwia jego swobodną rozbudowę od strony fizycznej oraz programowej przy zachowaniu wydajności obliczeniowej.

Pomimo różnych wzorów makiet algorytm zaimplementowany w urządzeniach PLC jest identyczny. Pomiędzy poszczególnymi sterownikami różni się tylko ilością faz i konfiguracją zapalanych świateł sygnalizacyjnych. Takie podejście pozwala na zachowanie otwartej polityki tworzenia oprogramowania i późniejsze rozszerzanie programu. Program sterownika nadrzędnego został zaprojektowany w taki sposób aby ingerencja ze strony urządzenia nadrzędnego miała miejsce wtedy i tylko wtedy, kiedy jest to wymagane. W pierwszym przypadku, gdy sterownik zostanie uruchomiony, przejdzie on przez 5-cio sekundowy sygnał żółty, ostrzegający o zmianie programu i pozostanie w trybie pulsującego żółtego światła, zgodnie z [1]. W tej sytuacji będzie on oczekiwać na sterownik nadrzędny, który wyzwoli program trójkolorowy. Dopiero po takim wymuszeniu, sterownik podrzędny będzie samoistnie przechodzić pomiędzy następnymi fazami. Sterownik nadrzędny może w każdej chwili narzucić rodzaj kolejnego podprogramu; może także zabronić wjazdu na skrzyżowanie nakazując emisję sygnału czerwonego dla wszystkich torów jazdy, przejść do innej fazy poza tradycyjną kolejnością lub też zawiesić pracę sygnalizacji świetlnej na skrzyżowaniu przechodząc do trybu pulsującego sygnału ostrzegawczego. Oba omówione przykłady zostały zaprezentowane na rysunku 3, w postaci deterministycznego automatu skończonego (DAS).



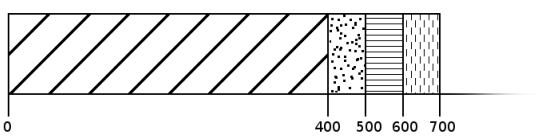
Legenda: S – Master (Serwer), L – Slave (Local), START, INIT, YHOLD, YFLSH – Nazwy programów; ALL_RED, Pn – Nazwy faz sygnalizacyjnych

Rys. 3. Diagram automatu skończonego dla programu pracy sterownika podrzędnego
Fig. 3. Diagram of the finite state machine for the slave PLC

Sygnały które pozwalają na przejście pomiędzy poszczególnymi stanami programu to LOCAL i SERVER. Flaga LOCAL jest odpowiednikiem zmiennej %T9 w pamięci sterownika; będąca domyślnie w stanie wysokim oznacza, że sterownik podrzędny aktualnie pracuje według wytycznych zapisanych w swojej pamięci. Sterownik nadrzędny, chcąc wymusić niestandardowe działanie sterownika podrzędnego, musi wpisać w to miejsce wartość "0", która samoistnie powróci do stanu wysokiego pod koniec następnego podprogramu sterownika przemysłowego. Natomiast flaga SERVER oznacza zanegowaną flagę LOCAL; wprowadzenie dodatkowej zmiennej do schematu ma na celu zachowanie przejrzystości diagramu. Przejścia pomiędzy programem trójkolorowym a pulsującym zawsze skutkują 5-cio sekundową emisją sygnału ostrzegawczego.

Program sterownika nadrzędnego został tak sformułowany aby, pełnił rolę pośredniczącą pomiędzy sterownikami podrzędnymi a serwerem OPC. Na żądanie serwera wywołuje on odpowiednią funkcję COMM_REQ, która odpytuje wybrany sterownik podrzędny. Wartość, która zostanie odczytana i przesłana przez protokół MODBUS, jest przechowywana w pamięci sterownika nadrzędnego, skąd jest później odczytywana przez serwer OPC.

Sterownik nadrzędny posiada tylko jeden tryb pracy. Zaraz po uruchomieniu, zgodnie z poleceniem programu w języku LD wpisuje do pamięci słowa instrukcji dla poleceń COMM_REQ. Następnie oczekuje na sygnały przychodzące od serwera OPC. Serwer ten wyzwala wykonanie odpowiedniego bloku COMM_REQ poszczególnych funkcji; samoistnie nie odpytuje sterowników podrzędnych. Sterownik wykonuje osiemnaście odpowiednio przygotowanych poleceń do wywołania przy pomocy bloku COMM_REQ; sześć dla każdego ze sterowników podrzędnych. Do każdego sterownika podrzędnego można wywołać trzy zapytania odczytujące wartości rejestrów oraz trzy wywołania zapisujące wartości do jego pamięci.



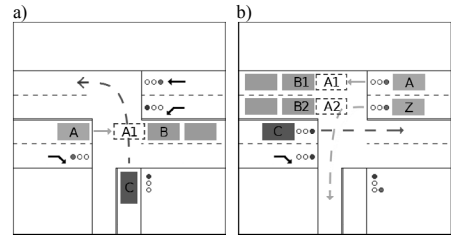
Rys. 4. Mapa pamięci rejestrowej sterownika nadrzędnego
Fig. 4. The register-memory map of the master controller

Na rys. 4 przedstawiono mapę pamięci sterownika nadrzędnego. Dla kodów bloków funkcji COMM_REQ została przeznaczona początkowa część pamięci sterownika. Następnie miejsca dla zmiennych, które mają zostać zapisane do sterowników podrzędnych, mieszczą się w granicach adresów 400-500. Od adresu 500 do 600 jest zarezerwowane miejsce dla wartości odczytywanych ze sterowników podrzędnych. Rejestry 600-700 to statusy wywoływanych funkcji COMM_REQ, w tym miejscu należy szukać informacji o błędach w przypadku występowania problemów przy komunikacji.

Do analizy danych pochodzących z makiet służy program napisany w języku Python. Dane pozwalają na zinterpretowanie zaistniałej sytuacji w modelu. Następnie w celu podjęcia odpowiednich działań, zostanie sformułowany komunikat do sterownika, poprzez serwer OPC. Działania te mogą dotyczyć zmiany wykonywanego programu bądź modyfikacji np. stałych czasowych poszczególnych faz, co w praktyce odpowiada realizacji sygnalizacji zmiennoczasowej w układzie zamkniętym, ze sprzężeniami od sensorów.

3. Możliwości wykorzystania symulatora

Opracowane środowisko jest w stanie identyfikować nieefektywne sytuacje, mogące wystąpić w ruchu drogowym oraz podejmuje jednoznacznie efektywne działania, w przypadku kiedy one nastąpią. Sytuacje takie zostały nazwane scenariuszami (rys. 5). Przykładowy scenariusz został zobrazowany na rys. 5a: strumień pojazdów otrzymuje sygnał zezwalający na przejazd przez skrzyżowanie, niestety sytuacja na drodze nie pozwala na kontynuowanie jazdy (pojazd A). W momencie, kiedy na skrzyżowaniu pojawia się nowy aktor (pojazd C), który chce wykonać skręt w lewo, system identyfikuje tę sytuację z zaimplementowanym scenariuszem. Sygnał zielony w danym momencie nie jest efektywnie wykorzystywany, czas sygnalizatora można przekierować na korzyść pojazdu oczekującego i natychmiast zezwolić mu na przejazd. W ten sposób zyskuje się cenny czas poprzez przedwczesne rozładowanie kolejki oczekującej. Jednocześnie zapobiegając powstawaniu zatoru na środku skrzyżowania, w wyniku którego mogłoby dojść do kompletnego zatorowania i uniemożliwienia przejazdu do trzech przecinających się strumieni pojazdów w danym miejscu.



Rys. 5. Przykładowe scenariusze zaimplementowane w opracowanym symulatorze
Fig. 5. Some scenarios implemented in the developed simulator

Tego typu scenariusze mogą być wytworzone dla wszystkich przypadków, dla każdej z makiet. Łatwość rozbudowy o kolejne makiety umożliwia generowanie kolejnych scenariuszy.

4. Wnioski

W artykule przedstawiono reprogramowalny i rekonfigurowalny symulator wykonany na bazie specjalnie opracowanych makiet, symbolizujących układy dróg (skrzyżowania dróg), sieci sterowników PLC i aplikacji w języku Python. Opracowany system pozwala na prostą implementację tzw. scenariuszy służących do testowania nowych rozwiązań w ruchu drogowym, przy jednoczesnym ich weryfikowaniu. Warto nadmienić, że powstałe środowisko symulacyjne umożliwia prowadzenie bezpiecznych badań ruchu drogowego, którego wyniki można przenieść do sytuacji rzeczywistych. Co więcej, od strony sprzętowej (standardy sygnałów, protokołów komunikacji), system umożliwia szybkie podłączenie do rzeczywistego układu sterującego ruchem drogowym lub odbieranie rzeczywistych danych pomiarowych. Podsumowując można wymienić następujące możliwości zaprojektowanego symulatora: (1) Sterowanie makietami skrzyżowań z interakcją od strony człowieka (zmiany stanów sensorów, ustalanie stanów awaryjnych itp.); (2) Pobieranie danych z rzeczywistych skrzyżowań i realizacja algorytmów sterowania z wizualizacją na makietach; (3) Sterowanie rzeczywistymi skrzyżowaniami z wizualizacją po stronie komputera z serwerem OPC (sytuacja hipotetyczna ze względu na konieczność stosowania układów z redundancją).

5. Literatura

[1] Datka S., Suchorzewski W., Tracz M.: Inżynieria ruchu. Wydawnictwo Komunikacji i Łączności, Warszawa, 1999.
[2] Broel-Plater B.: Sterowniki Programowalne Właściwości i zasady stosowania. Wydawnictwo Politechniki Szczecińskiej, 2000.
[3] Sokołowski R. pod opieką Małeckiego K.: Opracowanie programu do sterowania makietami symulującymi ruch pojazdów na określonym obszarze. Praca inż. na Wydziale Informatyki ZUT w Sz-nie, 2010.
[4] Sterowniki programowalne VersaMax Micro i Nano, GFK-1645C - PL – Podręcznik użytkownika, GE Fanuc Automotion, 2002.
[5] Modbus RTU Communications, GFK-2220B. GE Fanuc Automotion, 2003.
[6] Guzdial M.: Introduction to Computing and Programming In Python. Prentice Hall, 2005.

otrzymano / received: 26.04.2012
przyjęto do druku / accepted: 01.06.2012 *artykuł recenzowany / revised paper*