ZESZYTY NAUKOWE POLITECHNIKI ŚLĄSKIEJ 1998
Seria: AUTOMATYKA z. 125 Nr kol. 1391

Aleksander STASZULONEK
Politechnika Śląska

WIZUALIZACJA RUCHU EKSPERYMENTALNEGO ROBOTA PRZEMYSŁOWEGO

Streszczenie. W pracy prezentowany jest system wizualizacji ruchu robota eksperymentalnego. Przedstawiono założenia dotyczące tego systemu, sposób implementacji oraz środowisko, w którym system pracuje. Ponadto przedstawiono podstawowe zagadnienia grafiki komputerowej, takie jak: rzutowania i animacja. Omawiane algorytmy wykorzystano do stworzenia oprogramowania rzeczywistego systemu, a uzyskane wyniki przedstawiono w niniejszej pracy.

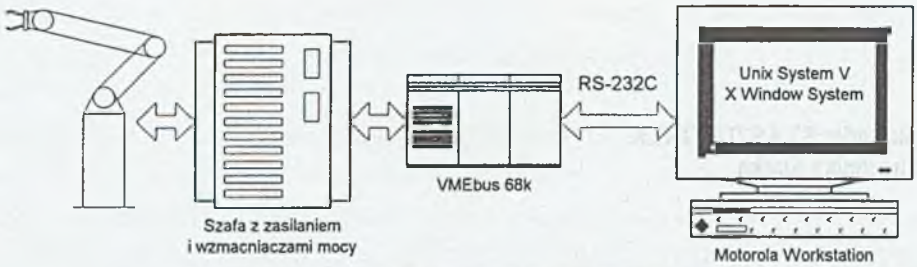
MOTION VISUALISATION OF EXPERIMENTAL INDUSTRIAL ROBOT

Summary. In this work the system of visualisation of industrial robot trajectory is presented. In particular, hardware and software environment, basic concepts of the system, and implementation problems have been discussed. Special protocol for the data transmission between the robot and supervising system has been developed and implemented. Basic ideas of computer graphics and animation have been shortly discussed. The ideas presented in this work have been implemented and tested on the experimental setup.

1. Sformułowanie problemu

Celem pracy jest stworzenie oprogramowania umożliwiającego wizualizację ruchu robota eksperymentalnego na ekranie monitora. Wejściem dla programu wizualizacji są dane pochodzące z łącza szeregowego, z którego pobierane są pozycje zadane i rzeczywiste robota realizującego trajektorię zadaną, generowaną przez programy zadawania położenia (np. program sterowania ręcznego robota, program generowania trajektorii, język programowania robota itp.). Wyjściem programu jest pseudoprzestrzenna animacja ruchu robota, dokonywana w czasie rzeczywistym na podstawie danych pobranych z łącza szeregowego.

Na rys. 1 przedstawiono schematycznie strukturę sprzętową stanowiska, dla którego przeznaczony jest system prezentowany w niniejszej pracy.



Rys. 1. Struktura sprzętowa stanowiska do badań robota eksperymentalnego
Fig. 1. Hardware setup for experimental robot

Jak widać na rysunku 1, system sterowania robota jest zbudowany w oparciu o dwa podstawowe elementy:

- system komputerowy oparty na magistrali VMEbus, realizujący funkcję sterownika napędów (sterownik podrzędny),
- stację roboczą Motorola, pracującą pod kontrolą systemu operacyjnego UNIX System V/68, której zadaniem jest realizacja funkcji sterownika nadrzędnego.

Stacja robocza połączona jest ze sterownikiem podrzędnym za pomocą łącza szeregowego RS232C. Dla komunikacji pomiędzy stacją roboczą a sterownikiem podrzędnym zdefiniowany jest specjalny protokół transmisji. Dzięki temu program pracujący na stacji roboczej może wysyłać do sterownika podrzędnego (sterownika napędów) położenia zadane dla poszczególnych napędów manipulatora.

Na najwyższym poziomie w strukturze sterowania robotem przemysłowym znajduje się warstwa planowania trajektorii. W przedstawionym rozwiązaniu zadania tej warstwy realizuje komputer MVME167, który jest sterownikiem nadrzędnym dla sterownika napędów i odpowiada za generowanie trajektorii i komunikację z użytkownikiem.

Sterownik podrzędny oparty jest na 32-bitowym komputerze przemysłowym MVME162 o następujących parametrach [5]: mikroprocesor MC68040 25MHz, 4MB dynamicznej pamięci RAM z kontrolą parzystości, 512KB statycznej pamięci RAM z podtrzymaniem bateryjnym, 1MB pamięci Flash, sterownik magistrali VME, interfejs SCSI z DMA, 2 poty szeregowe RS-232C, interfejs LAN Ethernet, 6 timerów, watchdog, gniazdo PROM, 4 gniazda dla interfejsów Industry Pack (IP).

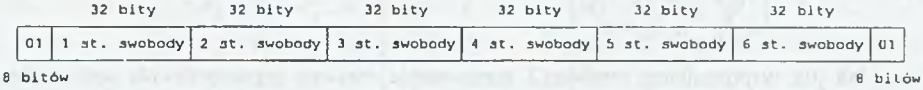
Do sterowania poszczególnymi stopniami swobody zastosowano moduły "industry pack". Głównym elementem każdego z modułów IP jest układ LM628. Jest to specjalizowany sterownik silników prądu stałego [7]. Każdy z układów LM628 posiada dwa niezależne kanały sterowania realizujące cyfrowy, 16-bitowy algorytm PID. Do komunikacji z układem dostępne są 22 komendy zorganizowane w pięć grup realizujących funkcje inicjalizacji, sterowania przerwaniami, generowania komend regulatora, sterowania trajektorią oraz wymiany danych. Rozproszony system sterowników poszczególnych osi odciąża procesor główny, który ma

wystarczająco dużo czasu na inne zadania, w tym na komunikację ze sterownikiem nadrzędnym.

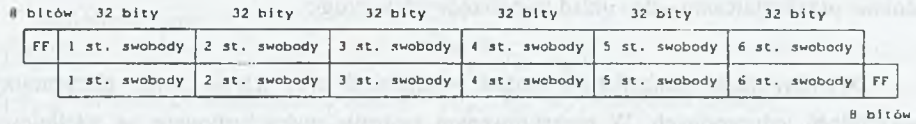
2. Protokół transmisji dla komunikacji ze sterownikiem podrzędnym

Jak już wspomniano, do komunikacji pomiędzy sterownikiem nadrzędnym (stacja robocza) a sterownikiem podrzędnym (sterownik napędów) zdefiniowano specjalny protokół transmisji. W punkcie tym zostanie przedstawiony format trzech rozkazów, będących częścią protokołu transmisji sterownik nadrzędny - sterownik podrzędny. Rozkaz zerowania powoduje wykonanie funkcji synchronizacji wszystkich stopni swobody, zerowania enkoderów i przetworników C/A i w końcu ustawia robota w położenie zerowe. Na rozkaz składają się dwa bajty zerowe, następujące po sobie. Jego format jest następujący:

Rozkaz pozycjonowania zawiera żądaną pozycję robota dla poszczególnych stopni swobody poprzedzoną i zakończoną bajtem o wartości 01. Format pełnego rozkazu jest następujący:



Rozkaz pochodzący ze sterownika napędów, zawierający pozycję zadaną i rzeczywistą, ma następujący format:



W powyższym rozkazie sześć pierwszych długich słów zawiera pozycję zadaną, natomiast kolejne sześć zawiera pozycję rzeczywistą. Rozkaz rozpoczyna się i kończy bajtem o wartości 0xFF (szesnastkowo).

3. Zastosowane elementy grafiki komputerowej

Ponieważ pojęcia i metody wykorzystywane w grafice komputerowej nie różnią się od tych znanych z geometrii dwu- i trójwymiarowej, to przedstawione zostaną tylko niektóre zagadnienia, dotyczące reprezentacji i przekształceń obiektów 3D oraz zagadnienia specyficzne dla przejścia od opisu geometrycznego do wyświetlania grafiki na ekranie (czyli przekształcenie współrzędnych do układu obserwatora związanego z ekranem monitora, rzutowanie).

Dowolny punkt w przestrzeni określony trójką uporządkowaną współrzędnych (x, y, z) jest reprezentowany przez wektor 1x4, wyrażony w postaci jednorodnej:

$$\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Układ współrzędnych jest jednocześnie macierzą 4x4 przekształcenia jednorodnego, jakie należy wykonać na układzie bazowym (odniesienia), aby przejść do danego układu współrzędnych:

$$T = \begin{bmatrix} a_x & b_x & c_x & d_x \\ a_y & b_y & c_y & d_y \\ a_z & b_z & c_z & d_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Układ współrzędnych kartezjańskich prawoskrętny spełnia następujące warunki:

1. $a_x^2 + a_y^2 + a_z^2 = 1$,
2. $b_x^2 + b_y^2 + b_z^2 = 1$,
3. $a_x b_x + a_y b_y + a_z b_z = 0$,
4. $\overline{a} \times \overline{b} = \overline{c} \Leftrightarrow \begin{cases} a_y b_z - a_z b_y = c_x \\ a_z b_x - a_x b_z = c_y \\ a_x b_y - a_y b_x = c_z \end{cases}$.

Jak już wspomniano, macierz T reprezentuje macierz przekształcenia jednorodnego. Aby przekształcić punkt w przestrzeniu należy wykonać następujące działanie:

$$r' = Tr$$

Podobnie przekształcamy jeden układ współrzędnych w drugi:

$$\varpi' = T\varpi$$

Dowolny ruch manipulatora można przedstawić przy użyciu ciągu elementarnych przekształceń jednorodnych. W prezentowanym systemie wykorzystywane są zdefiniowane poniżej macierze przekształceń jednorodnych przesunięcia, zmiany skali oraz obrotów.

Przesunięcie:

$$Trans(dx, dy, dz) = \begin{bmatrix} 1 & 0 & 0 & dx \\ 0 & 1 & 0 & dy \\ 0 & 0 & 1 & dz \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Skalowanie:

$$Scale(sx, sy, sz) = \begin{bmatrix} sx & 0 & 0 & 0 \\ 0 & sy & 0 & 0 \\ 0 & 0 & sz & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Obrót wokół osi x:

$$Rot(x,\theta)=\begin{bmatrix}1 & 0 & 0 & 0\\ 0 & \cos\theta & -\sin\theta & 0\\ 0 & \sin\theta & \cos\theta & 0\\ 0 & 0 & 0 & 1\end{bmatrix}$$

Obrót wokół osi y:

$$Rot(y,\theta)=\begin{bmatrix}\cos\theta & 0 & \sin\theta & 0\\ 0 & 1 & 0 & 0\\ -\sin\theta & 0 & \cos\theta & 0\\ 0 & 0 & 0 & 1\end{bmatrix}$$

Obrót wokół osi z:

$$Rot(z,\theta)=\begin{bmatrix}\cos\theta & -\sin\theta & 0 & 0\\ \sin\theta & \cos\theta & 0 & 0\\ 0 & 0 & 1 & 0\\ 0 & 0 & 0 & 1\end{bmatrix}$$

Ogólna postać przekształcenia jest złożeniem przesunięcia i obrotu i ma następującą postać:

$$T=\begin{bmatrix}a_x & b_x & c_x & d_x\\ a_y & b_y & c_y & d_y\\ a_z & b_z & c_z & d_z\\ 0 & 0 & 0 & 1\end{bmatrix}=\begin{bmatrix}1 & 0 & 0 & d_x\\ 0 & 1 & 0 & d_y\\ 0 & 0 & 1 & d_z\\ 0 & 0 & 0 & 1\end{bmatrix}\begin{bmatrix}a_x & b_x & c_x & 0\\ a_y & b_y & c_y & 0\\ a_z & b_z & c_z & 0\\ 0 & 0 & 0 & 1\end{bmatrix}$$

$$T=Trans(dx,dy,dz)Rot(n,\theta)$$

Jeśli oznaczymy:

$P=\begin{bmatrix}d_x\\ d_y\\ d_z\end{bmatrix}$ (macierz przemieszczeń) oraz $R=\begin{bmatrix}a_x & b_x & c_x\\ a_y & b_y & c_y\\ a_z & b_z & c_z\end{bmatrix}$ (macierz rotacji)

to przekształcenie możemy zapisać jako:

$$T=\begin{bmatrix}R & P\\ 0\ 0\ 0 & 1\end{bmatrix}$$

Przekształceniem złożonym nazywamy przekształcenie powstałe przez co najmniej dwa przekształcenia podstawowe nie jednostkowe:

$$T=T_1T_2\ldots T_n, n>1$$

$$T=\begin{bmatrix}R_1 & P_1\\ 0\ 0\ 0 & 1\end{bmatrix}\begin{bmatrix}R_2 & P_2\\ 0\ 0\ 0 & 1\end{bmatrix}\cdots\begin{bmatrix}R_n & P_n\\ 0\ 0\ 0 & 1\end{bmatrix}=$$

$$\begin{bmatrix}R=\prod_{k=1}^{n}R_k & P=\sum_{k=1}^{n-1}\prod_{l=1}^{n-k}R_lP_{n+1-k}+P_1\\ 0\ 0\ 0 & 1\end{bmatrix}$$

Rzutowanie:

Rzutowanie jest podstawowym przekształceniem w grafice komputerowej. Wizualizacja jakiegokolwiek obiektu przestrzennego wymaga odwzorowania go na płaszczyznę. Dopiero mając dwie współrzędne punktu (zamiast trzech) możemy narysować odcinek łączący dwa punkty na ekranie komputerowym.

Metody rzutowania dzielimy na dwie klasy:

- rzutowanie równoległe,
- rzutowanie perspektywiczne (środkowe).

Rzutowanie równoległe zachowuje proporcje rzutowanych przedmiotów. Odcinki równoległe przed rzutowaniem są także równoległe po rzutowaniu. Zachowane są stosunki odległości między punktami. Najprostszy sposób rzutowania na płaszczyznę ekranu obiektów trójwymiarowych polega na pominięciu odpowiednich współrzędnych wszystkich punktów przestrzeni. Na przykład, gdy pomijamy współrzędną z, otrzymujemy rzutowanie prostopadłe wzdłuż osi z. Pominięcie współrzędnej y - rzutowanie prostopadłe wzdłuż osi y, natomiast pominięcie składowej x - rzutowanie prostopadłe wzdłuż osi x. Właśnie ten specjalny rodzaj rzutowania (po uprzedniej zmianie skali) zastosowano w programie do wizualizacji robota w trzech rzutach: z boku, z przodu i z góry.

Rzutowanie pionowe ogólne można traktować jako przypadek, w którym obserwator znajduje się w pewnym dowolnym, ale ustalonym punkcie [Ex,Ey,Ez] i patrzy w kierunku początku tego układu. Prosta łącząca te dwa punkty jest normalną płaszczyzny rzutowania. Rzutowanie to uzyskujemy zmieniając osie układu współrzędnych, aby utworzyć układ obserwatora, którego początek jest wspólny z początkiem układu aktualnego (związanego z ekranem) i taki, że obserwator znajduje się na ujemnej części tej osi, wzdłuż której dokonujemy rzutowania. Podobne rozwiązanie zastosowano w programie wizualizacji. Różnice sprowadzają się do tego, że położenie obserwatora definiowane jest za pomocą azymutu i elewacji, a nie przez podanie jego współrzędnych. Azymut i elewacja pozwalają na zdefiniowanie położenia obserwatora na kuli o pewnym ustalonym promieniu, tak że spogląda on na jej środek, w którym znajduje się początek układu współrzędnych wizualizowanego obiektu. Jednocześnie azymut i elewacja są kątami, o jakie należy obrócić osie układu współrzędnych, aby przejść z układu współrzędnych obiektu do układu współrzędnych obserwatora:

$$T = Rot(z,\varphi)Rot(y,\theta)$$

Rzutowanie perspektywiczne:

Rzut perspektywiczny, zwany rzutem środkowym, stwarza u odbiorcy wrażenie przestrzenności oglądanego obrazy dwuwymiarowego. Rzuty leżące dalej od płaszczyzny rzutowania są mniejsze niż rzuty takich samych obiektów leżących bliżej tej płaszczyzny. Jest to symulacja zjawiska perspektywy, występującego w świecie rzeczywistym. Proste równoległe zbiegają się na horyzoncie, a przedmioty maleją w miarę oddalania się od nich.

Najprostszą metodą rzutowania perspektywicznego jest umieszczenie obserwatora np. na osi z, globalnego układu odniesienia, w punkcie E o współrzędnych [0,0,-d]. Płaszczyzną rzutowania jest w tym przypadku płaszczyzna rozpięta na osiach x i y. Rzutowanie dowolnego punktu P polega na znalezieniu współrzędnych punktu P', będącego punktem przecięcia płaszczyzny x-y przez prostą poprowadzoną od położenia obserwatora do rzutowanego punktu. Współrzędne punktu P' wyznaczane są z następujących zależności:

$$x' = x\frac{d}{z+d},\quad y' = y\frac{d}{z+d}$$

Postać jednorodna macierzy przekształceń dla rzutu perspektywicznego jest następująca:

$$T = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{d} & -1 \end{bmatrix}$$

4. Technika animacji komputerowej

Najprostszym sposobem uzyskania wrażenia ruchu obiektu jest skasowanie go ze starej pozycji i narysowanie w nowym miejscu. Jeżeli proces ścierania i odrysowywania obiektu w innym niż poprzednio położeniu będzie powtarzany, uzyska się wrażenie ruchu obiektu. Główną wadą tego rozwiązania jest efekt migotania ekranu podczas przerysowywania ekranu.

Jednym ze sposobów usunięcia efektu migotania jest przełączanie stron pamięci ekranu, jeżeli karta graficzna pozwala na wykorzystanie kilku stron. W takim przypadku obraz jest rysowany na stronie nieaktywnej, po czym strony są zmieniane i na ekranie ukazuje się uaktualniony rysunek. Proces ten jest powtarzany w pętli. Niestety, nie wszystkie karty graficzne czy środowiska okienkowe pozwalają na wykorzystanie techniki przełączania stron. Na przykład ani system Microsoft Windows ani X Window nie wykorzystuje techniki przełączania stron. Tworzenie pozbawionych efektu migotania animacji dla tych systemów wymaga innego podejścia.

Efekt migotania w trakcie ścierania i odrysowywania obiektów pojawia się dlatego, że wszystkie operacje ekranowe są widoczne. Kiedy obraz jest ścierany, widać, jak znika i jak pojawia się w innym miejscu, gdy jest odrysowywany. Kiedy używane są dwie strony pamięci ekranu, efekt migotania znika, bo wszystkie operacje odbywają się na stronie w danym momencie niewidocznej. Uaktualniony, gotowy obraz pojawia się od razu w chwili przełączania stron. Należy więc przypuszczać, że można uniknąć efektu migotania, jeżeli kolejne ekrany będą przygotowywane w pamięci (rysując na mapie bitowej), a następnie szybko wyświetlane na ekranie. Jest to technika ukrytych map bitowych [3]. Właśnie ta technika została wykorzystana w programie wizualizacji ruchu robota.

5. Struktura programowa systemu wizualizacji ruchu robota

System wizualizacji ruchu robota eksperymentalnego działa w czasie rzeczywistym i na bieżąco przetwarza rozkazy pobierane z łącza szeregowego. Wymaganie działania programu w czasie rzeczywistym nakłada na program poważne uwarunkowania czasowe. Przetwarzanie odebranych danych musi się zakończyć zanim pojawią się następne. W związku z powyższym w programie użyto prostych, a przez to efektywnych sposobów wizualizacji. Zdecydowano, że program będzie zrealizowany w postaci dwóch współpracujących ze sobą procesów. Pierwszy proces, będący procesem głównym, zajmuje się wizualizacją robota i dialogiem z użytkownikiem, natomiast proces drugi, będący procesem podrzędnym, odpowiedzialny jest za odczytywanie danych z łącza szeregowego i umieszczanie ich w kolejce. Rozwiązanie takie niesie ze sobą następujące korzyści:

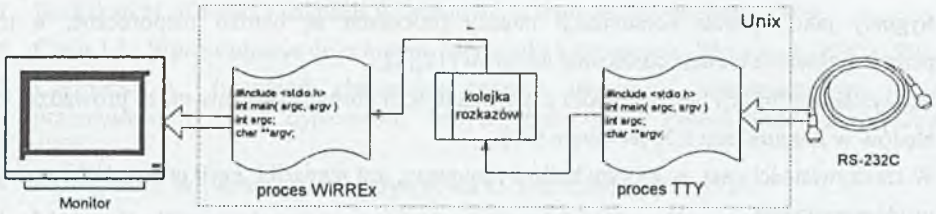
- zarówno operacje wizualizacji i czytania z łącza, jak i przydział procesora do poszczególnych zadań jest niezależny,
- rozdzielenie tych zadań stwarza możliwość przetwarzania danych pochodzących z innego źródła (zachowując jednak protokół transmisji).

W pierwszym przypadku znaczy to, że program czytający dane z terminala będzie je mógł odbierać zawsze wtedy, gdy takie dane się pojawią i nie musi oczekiwać na zakończenie obliczeń związanych z wizualizacją, jak i na zakończenie samej wizualizacji (wyświetlanie grafiki jest operacją bardzo czasochłonną). Ma to znaczenie, gdy dane do łącza są zapisywane z dużą prędkością. Wtedy bowiem opóźnienie w odebraniu danych mogłoby doprowadzić do ich zgubienia. Opóźnienie takie może być związane z obsługą przez X serwer niektórych zdarzeń pochodzących np. od Window Managera. Wtedy właśnie X serwer zawiesza przetwarzanie żądań użytkownika (czyli wstrzymuje funkcje rysujące, które w procesie wizualizacji odgrywają kluczową rolę). Opóźnienia te są krótkotrwałe i mogą być „odrobione" później, pod warunkiem, że dane pochodzące z łącza umieścimy w buforze pośrednim.

Druga korzyść polega na łatwym zastąpieniu procesu czytającego z łącza szeregowego innym procesem. Zadaniem tego procesu jest bowiem tylko oczekiwanie na dane z łącza szeregowego i umieszczanie ich w kolejce dla programu głównego. Można sobie wyobrazić zastąpienie tego procesu innym procesem, czytającym dane pochodzące z innego urządzenia (np. z sieci). Trzeba jednakże zachować dotychczasowy format danych. Gdyby dane miały inną postać, to łatwo umieścić w tym oddzielnym procesie moduł konwersji.

6. Współpraca procesów systemu wizualizacji ruchu robota eksperymentalnego

Schemat współpracy dwóch procesów składających się na prezentowany system przedstawiono na rys. 2.



Rys. 2. Schemat przepływu danych w systemie wizualizacji ruchu robota
Fig. 2. Diagram of the data transmission in the visualization system

W związku z możliwością zaistnienia sytuacji, w której dane są zapisywane i odczytywane z bufora z różnymi szybkościami, kolejkę rozkazów stanowi bufor cykliczny. Bufor cykliczny jest to ciągły obszar pamięci, z którym komunikacja odbywa się za pomocą dwóch wskaźników: główki (ang. head) i stopki (ang. tail). Stopka stanowi wskaźnik danych zapisywanych, natomiast główka - danych odczytywanych. Gdy któryś ze wskaźników osiągnie koniec bufora, to jest ustawiany na jego początku. Stąd w nazwie tej struktury danych słowo cykliczny. Jeśli wskaźnik zapisu jest o jeden mniejszy od wskaźnika odczytu, to bufor jest pełny. Natomiast gdy wskaźniki są sobie równe - bufor jest pusty. W celu umożliwienia procesom jednoczesnego i możliwie najszybszego dostępu do bufora zrealizowano go jako segment współdzielony pamięci. Dostęp do bufora synchronizowany jest poprzez semafor, aby zapewnić niepodzielność operacji zapisu (odczytu) danych i aktualizacji wskaźników. Proces WIZ sprawdza na bieżąco, czy w buforze są dane do przetworzenia. Jeśli takie dane są, to je przetwarza. Proces TTY (nazwany tak aby zaznaczyć komunikację poprzez linię terminalową) czeka na pojawienie się danych w łączu szeregowym, a gdy one się pojawią, czyta je z łącza i zapisuje do bufora. Nie jest to jedyny możliwy schemat współpracy pomiędzy procesami systemu WIZ. Istnieją rozwiązania alternatywne. W przedstawionym schemacie komunikacji proces WIZ aktywnie oczekuje na dane, być może trwoniąc zasoby procesora, natomiast proces TTY zasypia do czasu pojawienia się danych, zwalniając procesor dla innych zadań. Idealnym rozwiązaniem byłoby, gdyby zamiast ciągłego sprawdzania, czy dane znajdują się w buforze, proces WIZ także zasypiał. Jednak aby umożliwić takie działanie należałoby skorzystać z sygnałów (proces TTY po zapisaniu do bufora danych wysyłałby sygnał do procesu WIZ). Zrezygnowano z tej możliwości z następujących powodów:

- Traciłoby sens wykorzystanie bufora pośredniego, bowiem po otrzymaniu każdego sygnału proces musiałby odebrać dane i je przetworzyć, co mogłoby być kłopotliwe ze względu na wspomniane wyżej opóźnienia. Proces WIZ mógłby wprawdzie utrzymywać swoją prywatną kolejkę rozkazów (do której dane byłyby zapisywane w funkcji obsługi sygnału) i korzystać z niej, „nadrabiając" opóźnienia, jednak w przypadku, gdyby sygnały wysyłane były bardzo często, mogłoby się zdarzyć, że proces nie zdążyłby ponownie zainstalować funkcji obsługi sygnału, co doprowadziłoby do zakończenia się programu.

- Sygnały jako sposób komunikacji między procesami są bardzo nieporęczne, a ich przechwytywanie bardzo kosztowne czasowo [12][2].
- Korzystanie z funkcji biblioteki Xlib z wnętrza funkcji obsługi sygnałów może prowadzić do błędów w komunikacji z X serwerem [10].
- W rzeczywistości czas, w którym bufor byłby pusty, jest niewielki, czyli proces WIZ prawie wcale nie zasypia.

7. Wnioski

W pracy został stworzony systemu wizualizacji ruchu robota eksperymentalnego. Oprogramowanie wchodzące w skład tego systemu zostało uruchomione na stanowisku laboratoryjnym oraz przetestowane. Aby to zrealizować rozszerzono możliwości sterownika podrzędnego o funkcję zwracania pozycji zadanej i rzeczywistej robota, realizowaną jako przerwania zegarowe, o których częstości może zadecydować użytkownik.

Opracowany system umożliwia prezentację aktualnego położenia robota na ekranie terminala graficznego, tworząc atrakcyjne stanowisko do analizy pracy systemu sterowania robota, w tym np. do badania przebiegów przejściowych.

Zastosowanie w programie prostych środków wizualizacji było koniecznością związaną z wydajnością stacji roboczej. Z jednej strony mamy bowiem dosyć słabą stację roboczą, z drugiej strony realizacje systemów X Window nie są dobrze oceniane pod względem szybkości działania [7], szczególnie w zastosowaniach wizualizacji procesów przemysłowych. Rozszerzenie sytemu WIZ o możliwość bardziej realistycznego przedstawienia animowanego robota, jak również zwiększenie częstości wysyłania z robota danych jest możliwe, lecz pociąga za sobą konieczność pracy w bardziej wydajnych systemach.

Z kolei mała częstość wysyłanych z robota pozycji zadanych i rzeczywistych związana jest ze zbyt małą szybkością transmisji pomiędzy sterownikiem nadrzędnym a sterownikiem podrzędnym. Aby zdecydowanie polepszyć pracę systemu należałoby zastosować szybszą stację roboczą oraz zmienić sposób transmisji, korzystając z Ethernetu lub któregoś z szybkich szeregowych protokołów przemysłowych. Jednocześnie, przy zastosowaniu bardziej wydajnego sprzętu, mankamenty związane z realizacją systemów X Window nie będą przeszkodą, a w pełni ujawnią się zalety środowiska systemu UNIX. Wieloprogramowość, wydajne sposoby komunikacji między procesami oraz stabilność systemu znacznie wpłynęły na implementację i pracę programu, ujawniając elastyczność i przydatność tego systemu w podobnych zastosowaniach.

LITERATURA

1. Angell I.O.: Wprowadzenie do grafiki komputerowej. Warszawa, WNT 1988.
2. Bach M.J.: Budowa systemu operacyjnego Unix. Warszawa, WNT 1995.

3. Barkakati N.: Grafika i animacja w Windows. Warszawa, Intersoftland 1994.
4. Craig J.J.: Wprowadzenie do robotyki. Mechanika i sterowanie. Warszawa, WNT 1995.
5. Czarnecki A.: Integracja elementów systemu sterowania doświadczalnym robotem przemysłowym. Praca dyplomowa. Instytut Automatyki, Politechnika Śląska, Gliwice 1996.
6. Kernighan B.W., Ritchie D.M: Język ANSI C. Warszawa, WNT 1994.
7. Marzec B.: Wprowadzenie do standardu magistrali VMEbus. Warszawa, WNT 1994.
8. Mielczarek W.: Szeregowe interfejsy cyfrowe. Gliwice, Helion 1993.
9. Morecki A., Knapczyk J.: Podstawy robotyki. Teoria i elementy manipulatorów i robotów. Warszawa, WNT 1994.
10. Nye A. i in.: The Definitive Guides to the X Window System. Vol. 1-6. O'Reilly & Associates, Inc. 1992.
11. Piotrowski A.J.: Standard interfejsu szeregowego RS-232C. Komputer, nr 6/1986.
12. Rochkind M.J.: Programowanie w systemie Unix dla zaawansowanych. Warszawa, WNT 1997.
13. Stevens W.R.: Programowanie zastosowań sieciowych w systemie Unix. Warszawa, WNT 1996.

Recenzent: Prof.dr hab.inż. Jerzy Klamka

Abstract

In this work the system of visualisation of industrial robot trajectory is presented. In particular, hardware and software environment, basic concepts of the system, and implementation problems have been discussed. As the experimental setup an existing robot control system entirely developed at the Institute of Automation has been used. Developed control system has bilevel structure. At the servocontroller level the VME-bus based system has been developed. Each servomechanism has dedicated axis processor executing the positioning commands. At the higher level Unix based workstation with X-Window system as the graphic environment has been applied. This workstation generates the desired robot trajectory expressed in general coordinates, reads the actual axes position and displays the results on the X-terminal screen. Special protocol for the data transmission between the robot and supervising system has been developed and implemented. The desired trajectory is initially expressed in the Cartesian coordinates. The positioning commands are sent to the servosystem in natural coordinates. The actual positions are transmitted to the workstation also in the natural coordinates but for the graphical presentation they should be expressed in Cartesian coordinates. For this reason the workstation has to perform both stright and inverse kinematic problem solution. All this poses significant computational load. Basic ideas of computer

graphics and animation have been shortly discussed. The ideas used to implement the computer graphics are similar to those commonly used in 2D 3D geometry. The coordinate systems are described by the Hartenberg-Denavit matrices and robot positions are calculated using union of basic coordinates transformations like translation, rotation, scaling. For the graphic presentation different projections have to be used. The ideas presented in this work have been implemented and tested on the experimental setup.