POZNAN UNIVERSITY OF TECHNOLOGY ACADEMIC JOURNALS
No 99 Electrical Engineering 2019

DOI 10.21008/j.1897-0737.2019.99.0013

Przemysław STRZELCZYK*, Krzysztof TOMCZEWSKI*

IMPLEMENTACJA MODUŁU STEROWANIA NAPĘDEM ZŁĄCZA MANIPULATORA W ADAPTACYJNYM SYSTEMIE WYMIANY INFORMACJI

W artykule omówiono sposób implementacji elementów systemu sterowania napędem złącza kinematycznego. Do napędu złącza zastosowano silnik BLDC z dedykowanym sterownikiem MAXON EPOS2 70/10, włączonym do systemu sterowania za pomocą modułu abstrakcji sprzętowej adaptacyjnego systemu wymiany informacji. W pracy pokazano strukturę budowy napędu oraz systemu sterowania. Przedstawiony sposób implementacji abstrakcji sprzętowej pozwala na oddzielenie kodu oprogramowania sterującego napędem od implementacji sterowników dostarczanych przez producentów napędów oraz na odpowiednią synchronizację odwołań sprzętowych w obrębie modułu sterującego. Prezentowany układ stanowi napęd przegubu łokciowego prototypu aktywnego egzoszkieletu ręki.

SŁOWA KLUCZOWE: sterowanie, napęd BLDC, HAL, abstrakcja sprzętowa.

1. WSTĘP

Obecnie duży nacisk w projektowaniu systemów robotycznych kładzie się na wprowadzanie rozwiązań ułatwiających przyszłą produkcję. Głównym efektem tych działań jest skrócenie czasu przygotowania nowych wersji urządzeń. Jednym ze sposobów jest modularyzacja oprogramowania robotycznego oraz stosowanie oprogramowania pośredniego. [1]

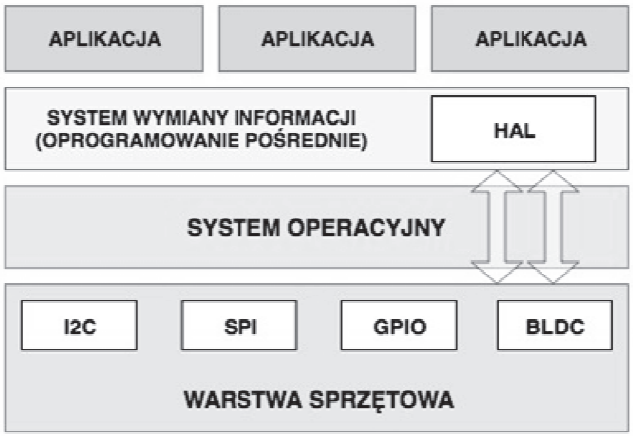
Obecnie na rynku istnieje szereg systemów oferujących mechanizmy wspomagające budowę skomplikowanych systemów robotycznych. [2] Najbardziej znanymi produktami dostępnymi na rynku są: ROS, RT-Middleware oraz Player/Stage. [3, 4, 5] Niektóre z nich w zestawie oferowanych funkcjonalności posiadają warstwę abstrakcji sprzętowej (*ang. Hardware Abstraction Layer*), która jest niezbędna do tworzenia modułowego oraz generycznego oprogramowania robotycznego. [6] Jednak większość oprogramowania robotycznego jest obecnie opracowywana bez jakiejkolwiek separacji wywołań sprzętowych oraz specyficznych odwołań bazujących na elementach wykorzystywanych modułów. Taki sposób tworzenia oprogramowania powoduje poważne konsekwencje na przy-

* Politechnika Opolska

szłość. W przypadku wymiany podzespołów elektronicznych lub systemu operacyjnego, konieczna jest zmiana specyficznych odwołań systemowych bądź odwołań funkcji API sterowników lub nawet poważniejsze modyfikacje oprogramowania. Często konieczna jest całkowita zmiana architektury oprogramowania sterującego. Jest to czynność długotrwała wiążąca się z dużymi dodatkowymi nakładami finansowymi i opóźnieniami przy wdrażaniu nowych wyrobów. Prezentowane w niniejszej pracy rozwiązanie wymaga od dostawcy oprogramowania lub osoby zamierzającej wykorzystać sterownik dedykowany do nowego sprzętu jedynie dostosowanie kodu do struktury narzuconej przez abstrakcyjną warstwę sprzętową.

2. WARSTWA ABSTRAKCJI SPRZĘTOWEJ

Warstwa abstrakcji sprzętowej została wykorzystana do odseparowania algorytmów kodu aplikacji wyższego rzędu od oprogramowania dostarczanego przez producentów poszczególnych rozwiązań sprzętowych, np. sterowników. Posiada ona interfejs, który pozwala na dołączenie kompatybilnych z nią sterowników dla wykorzystywanej bazy sprzętowej. Rysunek 1. przedstawia umiejscowienie warstwy abstrakcji sprzętowej w architekturze oprogramowania sterującego. W przedstawionym przykładzie, abstrakcyjna warstwa sprzętowa udostępnia funkcjonalność oferowaną przez dostępne rozwiązania sprzętowe: I2C (*ang. Inter-Integrated Circuit*), SPI (*ang. Serial Peripheral Interface*), GPIO (*ang. General-purpose input/output*) oraz BLDC (*ang. Brushless DC motor*).



Rys. 1. Prezentacja graficzna architektury oprogramowania przedstawiająca umiejscowienie warstwy abstrakcji sprzętowej

Warstwa wspomagająca komunikację ze sprzętem znajduje się pomiędzy systemem operacyjnym a aplikacjami korzystającymi z jego funkcjonalności. W pre-

zentowanym rozwiązaniu HAL jest częścią adaptacyjnego systemu wymiany informacji, pełniącego rolę robotycznego oprogramowania pośredniczącego. Opracowana warstwa abstrakcji sprzętowej została zaprojektowana na potrzeby oprogramowania pośredniego i posiada kilka podstawowych cech.

1. Separacja odwołań sterujących sprzętem od ich faktycznych implementacji. Odwołanie takie następuje poprzez wykorzystanie interfejsu HAL.
2. Określa strukturę budowy sterownika w przypadku wykorzystania go w ramach HAL.
3. Synchronizuje odwołania do sprzętu w ramach systemu operacyjnego.

Warstwa abstrakcji sprzętowej daje możliwość konfiguracji systemu w czasie działania programu za pomocą funkcji *loadConfiguration*. Funkcja ta pozwala użytkownikowi na konfigurację urządzeń poprzez odpowiednio przygotowany plik konfiguracyjny. Użytkownik ma możliwość definiowania urządzeń aktywnych, deaktywowania niepożądanych oraz wprowadzenia dodatkowych aliasów. Przykładowy plik konfiguracyjny pokazano na rys. 2.

```
[motor] [Enable=on] // zdefiniowanie urządzenia „motor” jako aktywnego
[motor] [maxon] // wprowadzenie aliasu nazewniczego
[gyro]  [Enable=off] // zdefiniowanie urządzenia „gyro” jako nieaktywnego
```

Rys. 2. Plik konfigurujący abstrakcyjną warstwę sprzętową

W przedstawionym przykładzie założono, że w systemie zostały zainstalowane dwa sterowniki: „motor” – służący do kontroli pracy napędów BLDC oraz sterownik „gyro” – służący do obsługi żyroskopu. W przypadku próby dostępu do urządzenia plik konfiguracyjny pozwala na odwoływanie do niego oraz przeszukiwanie bazy dostępnych urządzeń według identyfikatora „motor” lub „maxon”. Możliwość tworzenia aliasów pozwala na ułatwioną identyfikację urządzenia oraz zdefiniowanie własnej, unikalnej nazwy. Zmiana konfiguracji umożliwia również aktywowanie oraz deaktywowanie wybranych urządzeń. Możliwość konfiguracji systemu w czasie jego działania pozwala na szybkie jego dostosowanie do obecnej konfiguracji sprzętowej, bez konieczności ponownej, długotrwałej kompilacji oprogramowania.

W celu wykorzystania funkcjonalności abstrakcji sprzętowej w kodzie oprogramowania robotycznego należy utworzyć obiekt typu HAL. Kod przedstawiający tworzenie obiektu oraz odwołanie do sterownika silnika BLDC firmy MAXON pokazano na rys. 3.

```
#include <iostream>
#include <memory>
#include <HAL.hpp>

int main()
{
    std::unique_ptr<HAL> hal(new HAL(EHALMode_Master));

    auto maxonMotor = dynamic_cast<MaxonMotor*>(hal->device("maxon"));

    return 0;
}
```

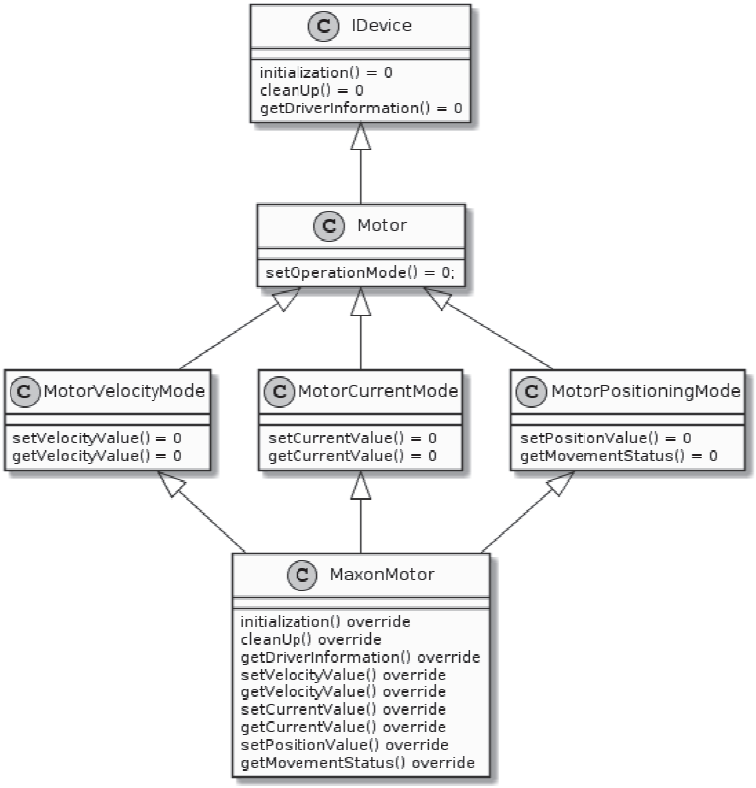
Rys. 3. Kod tworzący obiekt typu HAL oraz odwołanie do sterownika napędu firmy MAXON

Mechanizm HAL został wyposażony w metodę *device* służącą do przeszukiwania wewnętrznej bazy sterowników. Może zostać uruchomiony w trybie master lub slave. Tryby pracy są bezpośrednio związane z mechanizmem synchronizacji. W obrębie jednego systemu operacyjnego tylko jedna aplikacja może utworzyć obiekt HAL w trybie master. Każde ponowne wykorzystanie musi zostać zdefiniowane jako slave. Różnica w tworzeniu takich obiektów HAL polega na tym, że w przypadku trybu master tworzona i zainicjalizowana zostaje pamięć współdzielona, przechowująca informacje na temat aktualnie wykorzystywanych sterowników oraz ich użytkowników. W momencie, w którym metoda znajdzie sterownik zgodny z danym kluczem wyszukiwania, zwraca wskaźnik typu *IDevice* do znalezionego obiektu. Typ ten pełni rolę podstawowego interfejsu każdego sterownika będącego częścią HAL. Abstrakcyjna warstwa sprzętowa może zarejestrować w swojej bazie sterowniki dziedziczące pośrednio lub bezpośrednio po klasie *IDevice*. Interfejs klasy *IDevice* posiada następujące metody:

1. *initialization* – metoda czysto wirtualna, wymuszająca inicjalizację sterownika. Za pośrednictwem tej metody sterownik wykonuje wszystkie czynności, które pozwalają na pełną inicjalizację urządzenia. Po wykonaniu tej metody urządzenie jest gotowe do wykorzystania.
2. *cleanUp* – metoda czysto wirtualna wymuszająca implementację czyszczącą zasoby po odłączeniu sterownika. Za pośrednictwem tej metody sterownik wykonuje wszystkie czynności, które pozwalają na pełną finalizację i zwolnienie zasobów wykorzystywanych podczas działania sterownika.
3. *getDriverInformation* – metoda czysto wirtualna wymuszająca implementację metody zwracającej informacje na temat wersji oraz rodzaju sterownika.

Przedstawiona abstrakcyjna warstwa sprzętowa jest częścią opracowanego oprogramowania pośredniego. Dzięki mechanizmowi HAL dane urządzenie może być wykorzystywane przez kilka aplikacji sterujących jednocześnie.

Współdzielone korzystanie z urządzeń odbywa się tylko w pewnych przypadkach. Każdy z nich jest rozpatrywany indywidualnie z uwzględnieniem specyfiki danego urządzenia. Rysunek 4. przedstawia zależności pomiędzy klasami. Klasa *MaxonMotor* odpowiada za sterowanie pracą napędu BLDC. Implementuje ona wszystkie metody wirtualne dostarczone wraz z dziedziczeniem po klasach *MotorVelocityMode*, *MotorCurrentMode*, *MotorPositioningMode* oraz *Motor*.



Rys. 4. Diagram klas przedstawiający relacje pomiędzy klasą obsługującą urządzenie Maxon (MaxonMotor) a klasami abstrakcyjnymi

Każda z klas abstrakcyjnych została przygotowana tak, aby określała interfejs danego trybu. Przykładowo klasa *MotorVelocityMode* oferuje zestaw metod, które muszą być zaimplementowane w sterowniku oferującym możliwość sterowania prędkością silnika. Analogiczna sytuacja ma miejsce w trybie sterowania prądowego oraz pozycjonowania. Rysunek 5. przedstawia wywołanie metody inicjalizującej sterownik.

```
#include <iostream>
#include <memory>
#include <HAL.hpp>

int main()
{
    std::unique_ptr<HAL> hal(new HAL(EHALMode_Master));

    auto maxonMotor = dynamic_cast<MaxonMotor*>(hal->device("maxon"));

    maxonMotor->initialization();

    maxonMotor->setOperationMode(EMotorOperationMode_VelocityControl);

    maxonMotor->setVelocityValue(100);

    return 0;
}
```

Rys. 5. Kod prezentujący inicjalizację sterownika, zmianę trybu pracy oraz uruchomienie silnika

Następnie określony zostaje tryb pracy jako tryb kontroli prędkości. Kolejno, za pomocą metody *setVelocityValue* określa się zadaną prędkość obrotową silnika. W przykładzie przedstawionym na rys. 5. zdefiniowano typ zastosowanego napędu. Może jednak zdarzyć się sytuacja, w której trzeba skorzystać jedynie z ogólnodostępnych funkcjonalności oferowanych przez sterownik silnika. Nie jest wówczas konieczna znajomość typu i nazwy napędu. Wystarczy zdefiniować w systemie dowolny silnik pod kluczem *motor*. HAL umożliwi wyszukanie odpowiedniego urządzenia oraz wykonanie zadań z zakresu podstawowego interfejsu.

Przykład pokazany na rys. 6. prezentuje sposób wyszukiwania napędu dowolnego typu oraz uruchomienie go za pomocą podstawowego interfejsu zdefiniowanego przez klasę Motor. Użytkownik może wykorzystywać interfejs podstawowy do momentu pojawienia się potrzeby wykorzystania funkcjonalności, które są specyficzne dla danego modelu napędu i nie zostały określone w interfejsie podstawowym. W przypadku potrzeby zdefiniowania konkretnego rozwiązania sprzętowego oraz specyficznych funkcjonalności można postępować zgodnie z przykładem przedstawionym na rys. 5.

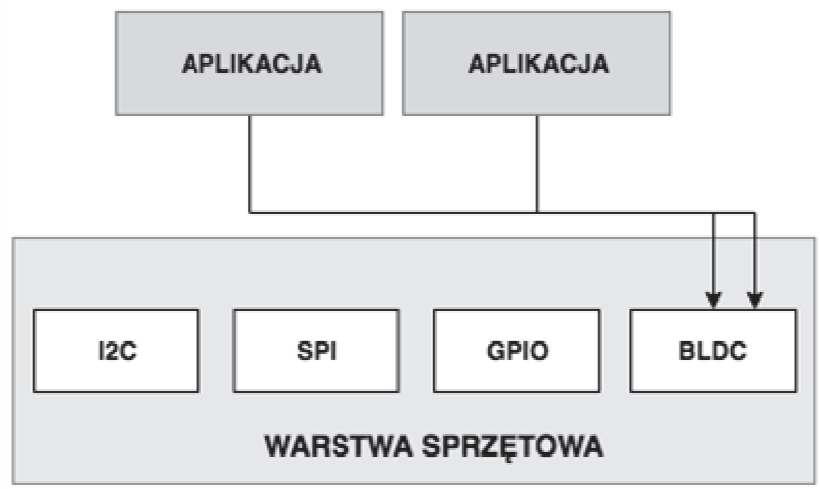
```
#include <iostream>
#include <memory>
#include <HAL.hpp>

int main()
{
    std::unique_ptr<HAL> hal(new HAL(EHALMode_Master));

    auto defaultMotor = dynamic_cast<Motor*>(hal->device("motor"));

    defaultMotor->initialization();

    defaultMotor->setOperationMode(EMotorOperationMode_CurrentControl);

    defaultMotor->setCurrentValue(100);

    return 0;
}
```

Rys. 6. Kod prezentujący inicjalizację sterownika, zmianę trybu pracy oraz uruchomienie silnika bez zdefiniowania modelu napędu

3. SYNCHRONIZACJA ODWOŁAŃ SPRZĘTOWYCH

Głównym problemem występującym podczas udostępnienia możliwości asynchronicznych odwołań do sprzętu jest kwestia ich poprawnej synchronizacji po stronie warstwy udostępniającej abstrakcję sprzętową. Synchronizacja w obrębie systemu operacyjnego odwołań sprzętowych w opracowanym adaptacyjnym systemie wymiany informacji, będącym oprogramowaniem pośrednim, została zrealizowana za pomocą umieszczenia odpowiednich struktur danych w pamięci współdzielonej.

Na rys. 7. przedstawiono sytuację, w której dwie aplikacje sterujące próbują uzyskać dostęp do tego samego urządzenia jednocześnie. Ze względu na uniwersalność opracowanego systemu wymiany informacji, jednym z zadań abstrakcyjnej warstwy sprzętowej jest odpowiednia synchronizacja niniejszych odwołań sprzętowych. Zakładając, że dostęp do pojedynczego urządzenia może być żądany przez więcej niż jedną aplikację sterującą przygotowano strukturę danych reprezentującą poszczególne urządzenia przyłączone do HAL. Struktura ta została pokazana na rys. 8.



Rys. 7. Rysunek poglądowy prezentujący przypadek próby dostępu do napędu BLDC z dwóch aplikacji jednocześnie

```
struct SharedDeviceArrayElement
{
    pthread_mutexattr_t mutexAttr;
    pthread_mutex_t mutex;
    char driverName[255];
    int lastUserID;
};

struct SUserIdArray
{
    pthread_mutexattr_t mutexAttr;
    pthread_mutex_t mutex;
    int index;
    int users[USERS_LIMIT];
};
static constexpr unsigned int USERS_LIMIT{128};

SharedDeviceArrayElement sharedArray[DRIVERS_LIMIT];
SUserIdArray userIDSharedArray[DRIVERS  LIMIT];
```

Rys. 8. Struktury danych wykorzystane w mechanizmie synchronizacji odwołań sprzętowych

Tablice struktur *sharedArray* oraz *userIDSharedArray* zostały umieszczone w pamięci współdzielonej. Pierwsza ze struktur odpowiedzialna jest za przetrzymywanie podstawowych informacji identyfikujących poszczególne sterowniki. Zarejestrowanie sterownika w systemie HAL wymaga umieszczenia wpisu

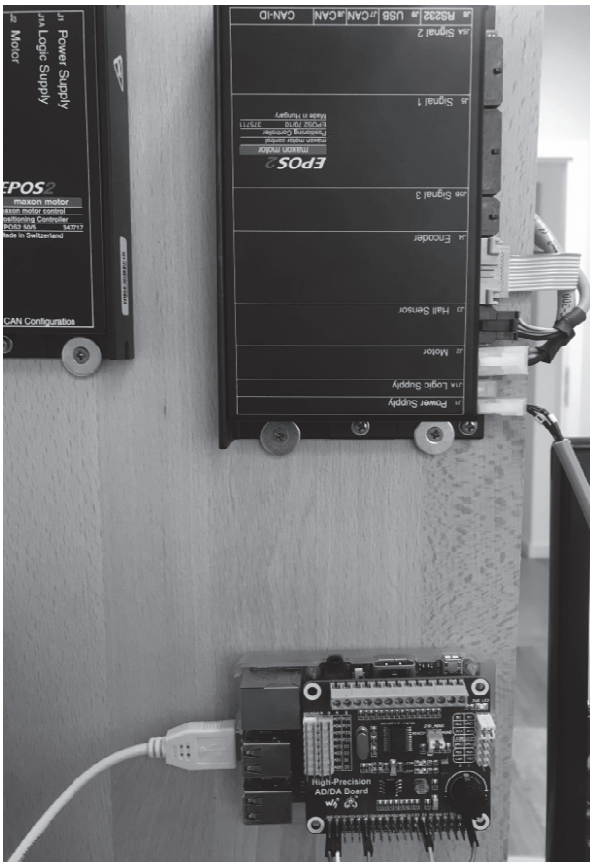
w pamięci współdzielonej. Każdy sterownik posiada swój osobny obiekt typu *pthread_mutex_t*, służący do synchronizacji dostępu do niego. Element struktury przetrzymuje również informacje o nazwie sterownika oraz identyfikator ostatniego użytkownika, który odwoływał się do urządzenia. Podczas próby dostępu do urządzenia, proces żądający odpowiedzialny jest za sprawdzenie stanu mutex'a a następnie ewentualnej blokady do momentu zaprzestania korzystania ze sterownika. Struktura sterownika narzuca założenie blokady wyłącznie na okres wydania pojedynczej komendy do urządzenia. Oznacza to, że system nie zapewnia możliwości zablokowania urządzenia na zdefiniowany czas. Dzięki temu zapewnia możliwość elastycznego współdzielenia urządzeń na zasadzie przyznawania krótkich przedziałów czasowych możliwości dostępu.

Kolejną jest struktura *userIDSharedArray*, która odpowiedzialna jest za przechowywanie podstawowych informacji o użytkownikach wykorzystujących HAL w obrębie systemu operacyjnego. Założono, że jedna instancja obiektu typu HAL odpowiada jednemu użytkownikowi. Podczas tworzenia instancji obiektu tworzony jest nowy użytkownik, dla którego generowany jest unikalny 32-bitowy identyfikator. Wszystkie te informacje zostają zapisane w niniejszej strukturze. Dopuszczalną liczbę urządzeń obsługiwanych przez abstrakcyjną warstwę sprzętową można definiować poprzez modyfikację wartości stałej *USERS_LIMIT*.

4. UKŁAD PROTOTYPOWY

Do testowania abstrakcyjnej warstwy sprzętowej opracowano stanowisko, fragment którego pokazano na rys 9. Układ składa się z modułu Raspberry Pi 2 B połączonego bezpośrednio ze sterownikiem silnika BLDC firmy MAXON EPOS2 70/10. Moduł Raspberry Pi 2 B stanowi część platformy sprzętowej sterującej aktywnym egzoszkieletem ręki.

Elementem wykonawczym w układzie prototypowym jest silnik MAXON EC 45 flat o mocy 50 W. Silnik został wyposażony w czujniki halla, wbudowany enkoder inkrementalny MILE 1024 CPT oraz przekładnię GP42. Został on wykorzystany do napędu złącza łokciowego w prototypie aktywnego egzoszkieletu ręki. Abstrakcja sprzętowa została w tym przypadku wykorzystana do zaimplementowania sterownika napędu BLDC oraz do pobierania informacji o aktualnej pozycji wirnika z wyznaczanej za pomocą enkodera inkrementalnego.



Rys. 9. Fragment stanowiska testowego zawierający sterownik silnika BLDC MAXON EPOS 70/10 oraz modułu Raspberry Pi

Zaprezentowana warstwa sprzętowa umożliwiła synchronizację, pozwalającą na dostęp do urządzenia z kilku aplikacji jednocześnie. Założeniem projektu egzoszkieletu jest decentralizacja sterowania. Umożliwia ona swobodną konfigurację egzoszkieletu, dostosowaną do konkretnych wymagań. Maksymalna liczba złącz kinematycznych uwzględnionych w systemie sterowania opracowywanego egzoszkieletu ręki wynosi pięć. System automatycznie rozpoznaje dołączane napędy i uwzględnia tylko złącza aktywne w danej konfiguracji. W prototypie zainstalowano 3 złącza aktywne. Złącze łokciowe prototypu wraz z napędem pokazano na rys. 10.

Rys. 10. Napęd BLDC MAXON EC 45 flat 50 W przegubu łokciowego

5. PODSUMOWANIE

Jednym z głównych problemów przy opracowywaniu nowych wersji robotów jest konieczność znacznych modyfikacji oprogramowania. Zmiana podzespołów elektronicznych z dużym prawdopodobieństwem wiąże się z koniecznością zmiany znacznej części oprogramowania, odpowiadającej za logikę aplikacji sterującej. Powoduje to znaczny wzrost kosztu wdrażania nowych rozwiązań. Zastosowanie opracowanej warstwy HAL umożliwia separację programu sterującego od warstwy sprzętowej. Wykorzystanie abstrakcji sprzętowej wpływa w znacznym stopniu na skrócenie czasu realizacji nowych wersji programów ze względu na brak konieczności modyfikacji oprogramowania.

Oprogramowanie o takiej strukturze umożliwia tworzenie aplikacji sterujących różnymi urządzeniami lub grupami urządzeń robotycznych. Może być wykorzystywane do realizacji systemów sterowania w urządzeniach o konstrukcji modułowej do ich automatycznej konfiguracji lub w systemach rozproszonych do komunikacji i wymiany informacji pomiędzy urządzeniami.

LITERATURA

[1] Nesnas Issa A., Simmons R., Gaines D, Kunz C., Diaz-Calderon A., Estlin T., Madison R., Guineau J., McHenry M., Shu I., Apfelbaum D., CLARAty: Challenges and Steps Toward Reusable Robotic Software, International Journal of Advanced Robotic Systems, ISSN 1729-8806.
[2] Mohamed N., Al-Jaroodi J., Jawhar I., Middleware for Robotics: A Survey, RAM 2008 978-1-4244-1676-9/08 IEEE.
[3] Cervera E., Try to Start It! The Challenge of Reusing Code in Robotics Research, IEEE Robotics and Automation Letters, Volume 4, January 2019.
[4] Einhorn E., Langner T., Stricker R., Martin Ch., Gross H-M., 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems, 2012 Vilamoura, Portugal.
[5] Elkady A., Sobh T., Robotics Middleware: A Comprehensive Literature Survey and Attribute-Based Bibliography, Hindawi Publishing Corporation Journal of Robotics, Volume 2012, DOI: 10.1155/2012/959013.
[6] Muratore L., Laurenzi A., Hoffman E., Rocchi A., Caldwell D., Tsagarakis N., XBotCore: A Real-Time Cross-Robot Software Platform, 2017 First IEEE International Conference on Robotic Computing, DOI 10.1109/IRC.2017.45.

IMPLEMENTATION OF THE DRIVE CONTROL MODULE OF MANIPULATOR CONNECTOR IN THE ADAPTIVE EXCHANGE INFORMATION SYSTEM

The article presents the way of drive control module implementation. A BLDC motor was used to drive the connector with usage of MAXON EPOS2 70/10. All references to the hardware layer have been made using hardware abstraction layer which is a part of adaptive exchange information system. The paper presents the way in which hardware abstraction layer was implemented. The presented abstraction layer provides high level separation between the hardware and software.

(Received: 27.01.2019, revised: 12.03.2019)