POLITECHNIKA WROCŁAWSKA

WYDZIAŁ ELEKTRONIKI

KIERUNEK: Automatyka i Robotyka (AiR)

SPECJALNOŚĆ: Systemy informatyczne w automatyce (ASI)

PROJEKT INŻYNIERSKI

Budowa i oprogramowanie robota klasy (2,0) ze zdalnym interfejsem sterowania

Hardware and software design of remotely controlled (2,0) class robot

AUTOR:

Kornel Mrozek

PROWADZĄCY PROJEKT:

Dr inż. Janusz Jakubiak

OCENA PROJEKTU:

Spis treści

[1. Wprowadzenie 3](#_Toc436583494)

[1.1 Cel projektu 3](#_Toc436583495)

[1.2 Założenie projektowe 3](#_Toc436583496)

[2. Kontrukcja sprzętowa 4](#_Toc436583497)

[2.1 Platforma mechaniczna 4](#_Toc436583498)

[2.2 Konstrukcja eletroniczna 4](#_Toc436583499)

[2.2.1 Moduł STM32 NUCLEO-F401RE 4](#_Toc436583500)

[2.2.2 Bazowy obwód – schemat 4](#_Toc436583501)

[2.2.3 Bazowy obwód – obwód drukowany 4](#_Toc436583502)

[2.2.4 Moduł Bluetooth – HC-06 4](#_Toc436583503)

[2.3 Podsumowanie 4](#_Toc436583504)

[3. Oprogramowanie robota mobilnego 4](#_Toc436583505)

[3.1 Struktura oprogramowania 4](#_Toc436583506)

[3.2 Realizacja komunikacji Bluetooth 5](#_Toc436583507)

[3.3 Proces przetwarzania wiadomosci 6](#_Toc436583508)

[3.4 Zabezpieczenia robota 8](#_Toc436583509)

[3.5 Realizacja odczytu prędkości 9](#_Toc436583510)

[3.6 Układ regulacji 11](#_Toc436583511)

[4. Oprogramowanie aplikacji sterującej 12](#_Toc436583512)

[4.1 Struktura oprogramowania 12](#_Toc436583513)

[4.2 Moduł kontroli komunikacji Bluetooth 12](#_Toc436583514)

[4.3 Moduł sterowania tekstowego 13](#_Toc436583515)

[4.3.1 Użycie silnika JavaScript w aplikacji 15](#_Toc436583516)

[4.4 Moduł sterowania graficznego 16](#_Toc436583517)

[4.4.1 Bazowy blok sterujący 18](#_Toc436583518)

[4.4.2 Bloki sterujące dostępne w aplikacji 18](#_Toc436583519)

[4.4.3 Synchronizacja aktywności bloków 21](#_Toc436583520)

[5. Protokół komunikacyjny 22](#_Toc436583521)

[5.1 Spis wiadomości kontrolnych 23](#_Toc436583522)

[6. Możliwości wykonanego projektu 26](#_Toc436583523)

[7. Podsumowanie projektu 26](#_Toc436583524)

# Wprowadzenie

Robot mobilny klasy 2,0 jest robotem posiadającym dwukołowy napęd różnicowy. Przemieszczania robota umożliwia sterowanie prędkością obrotu koła przymocowanego do danego napędu bez możliwości zmiany kierunku ułożenia koła w stosunku do platformy robota.

## Cel projektu

Celem projektu było zbudowanie fizycznego robota mobilnego 2.0 wraz z napisaniem oprogramowania na mikrokontroler oraz aplikacji, która umożliwia zdalne sterowanie robotem. Komunikacja pomiędzy platformą mobilną a programem sterującym oparta jest na standardzie Bluetooth. Celem pracy było umożliwianie jak największej interakcji z robotem zarówno w czasie rzeczywistym, jak i za pomocą komend tekstowych, z możliwością pisania skryptów.

## Założenie projektowe

Projekt zakłada przygotowanie modelu mechanicznego zawierającego dwukołowy napęd różnicowy wraz z układem elektronicznym umożliwiającym sterowanie silnikiem prądu stałego, jak również odczytem prędkości obrotu wału silnika. W układzie elektronicznym konieczny jest moduł Bluetooth umożliwiający komunikację szeregową z mikrokontrolerem. Oprogramowanie mikrokontrolera osadzonego na robocie zawiera:

* komunikację dwukierunkową interfejsu szeregowego
* asynchroniczną reakcję na otrzymane wiadomości z zewnątrz
* obsługę sterowania sygnału PWM (wymagane przy sterowaniu silnikiem)
* obsługę zliczania impulsów (wymagane przy odczycie prędkości)
* układ regulacji prędkości
* zabezpieczenie przed utratą transmisji

Aby możliwa była komunikacja między robotem a otoczeniem z zewnątrz wymagane jest przygotowanie protokołu komunikacyjnego dającego aplikacji interfejs na którym może operować. Protokół zawiera wiadomości typu pobierz/zapisz które pozwalają na pobranie z robota wartości i ich nadpisanie. Aplikacja sterująca, zgodnie z celem projektu, ma umożliwiać szeroki zakres form sterowania, opierając się na wyżej wymienionym protokole. Oprogramowanie sterujące zakłada:

* możliwość zapisu i odczytu danych w standardzie Bluetooth poprzez wirtualny port COM
* możliwość sterowania robotem za pomocą komend tekstowych. Punkt ten zakłada przygotowanie modułu interpretera do wprowadzania komend/skryptów oraz wykorzystanie silnika języka JavaScript
* możliwość sterowania w trybie graficznym za pomocą myszki, wizualizacja prędkości oraz konfiguracja robota

# Konstrukcja sprzętowa

## Platforma mechaniczna

Platforma mechaniczna robota oparta jest o model Zumo Chassis Kit firmy Pololu. Podstawa mechaniczna zaprojektowana została do konstrukcji robotów o napędzie różnicowym wyposażonym w miniaturowe silniki elektryczne prądu stałego tej samej firmy. Podstawa wyposażona jest w gąsienicowy układ bieżny. Główną zaletą tego rozwiązania jest zwiększenia powierzchni styku z podłożem, co zwiększa przyczepność i zmniejsza prawdopodobieństwo poślizgu poprzecznego. Jest szczególnie istotne w przypadku użycia odometrii przy pomiarze prędkości robota. Model posiada miejsca na cztery baterie AA, jednakże projekt zakładał użycie innego źródła zasilania. Niewielkie modyfikacje umożliwiły zastosowanie akumulatora typu LiPol. Omówienie układu zasilania znajduję się podrodziale 2.2. Wybór platformy podyktowany wynikała głównie z konstrukcji napędu, którego zależy zostały wymienione wcześniej , jak również z

|  |
| --- |
|  |
| C:\Users\Kornel\Desktop\pracka\model_mechaniczny.jpg |

Wymiary modelu mechanicznego:

* wymiary pełnego modelu: 86mm x 98mm
* odległość pomiędzy kołami (pomiar od środka gąsienicy):
* przestrzeń na podstawę elektroniczną:
* średnica koła:

Napędami są miniaturowe silniki Pololu HP z dwustronna osi. Silnik posiada przekładnie 50:1. Obustronny wał pozwala na wygodny montaż enkoderów, czyli czujników wykorzystywanych przy pomiarze prędkości obrotowej silnika. Wybór silników wynikał z jego dużych możliwości w stosunku do rozmiaru oraz dużego doświadczenia w pracy z tym napędem.

|  |
| --- |
|  |
| C:\Users\Kornel\Desktop\pracka\silnik.jpg |

Parametry silnika:

* Maksymalne napięcie pracy: 9[V]
* Prędkość obrotowa dla napięcia 6[V]: 625 obrotów na minutę
* Moment obrotowy dla napięcia 6[V]: 1.1 [kg\*cm]
* Maksymalny prąd dla napięcia 6[V]: 1600[mA]

## Konstrukcja elektroniczna

## Moduł STM32 NUCLEO-F401RE

Elektronika robota opera się o układ Nucleo F401RE. Jest to modu

## Bazowy obwód

## Moduł Bluetooth – HC-06

## Podsumowanie

# Oprogramowanie robota mobilnego

## Struktura oprogramowania

Oprogramowanie napisane jest w języku C korzystając z bibliotek HAL (Hardware Abstraction Layer) udostępnionych przez firmę STMicroelectronics . Konfiguracja mikrokontrolera wykonana jest przy użyciu oprogramowania STMCubeMX

|  |
| --- |
| C:\Users\Kornel\Desktop\pracka\cubemx1.png |
| C:\Users\Kornel\Desktop\pracka\clock.PNG |

Mikrokontroler jest taktowany z częstotliwością 84 MHz. Źródłem taktowania jest rezonator kwarcowy 8 MHz, którego częstotliwość jest powielana przez wewnętrzną pętlę PLL.

|  |
| --- |
| C:\Users\Kornel\Desktop\pracka\uC_config.PNG |

Oprogramowanie wykorzystuje 7 sprzętowych timerów, obsługę transmisji szeregowej UART, magistrale DMA oraz kontroler przerwań NVIC. Oprogramowanie nie zawiera żadnej instrukcji w pętli głównej programu, zachowanie robota jest całkowicie sterowane przerwaniami. Aktualny stan robota utrzymywany jest w zmiennych globalnych.

## Realizacja komunikacji Bluetooth

Komunikacja Bluetooth wykorzystuje obsługę dwóch komponentów mikrokontrolera, interfejsu szeregowo UART oraz technika DMA. UART (*Universal Asynchronous Receiver and Transmitter)* umożliwia asynchroniczne wysyłanie i odbieranie danych przez port szeregowy. Wykorzystanie modułu Bluetooth ogranicza się do połączenia linii danych TX (wysyłanie) mikrokontrolera z linią danych RX (odbiór) układu HC-06 oraz analogicznie linii RX mikrokontrolera z linią TX modułu. Transmisja bezprzewodowa jest przezroczysta z poziomu mikrokontrolera. Interfejs UART konfigurowany jest przed podanie 4 wartości określających transmisję:

* prędkość transmisji (ang. Baud rate)
* długość słowa
* ilość bitów stopu
* bit parzystości (kontrola błędów odbiory danych )

Zdecydowałem się na użycie standardowej konfiguracji prędkość: 9600, 8-bitowe słowo, jeden bit stopu, brak bitów parzystości.

DMA jest modułem który umożliwia bezpośredni dostęp do pamięci RAM i układów peryferyjnych. Układ DMA umożliwia transfer danych pomiędzy urządzeniem peryferyjnym a pamięcią RAM bez angażowania procesora, którego zadanie ogranicza się do konfiguracji urządzenia. Komponent został użyty do przesyłu i odbioru danych pomiędzy układem UART i dziesięciobajtowym obszarem pamięci RAM. Włączenie odbioru danych szeregowych z użyciem DMA:

|  |  |  |
| --- | --- | --- |
|  | int8\_t command[10];  HAL\_UART\_Receive\_DMA(&huart2, command, sizeof(command)); |  |

Argumentami tej funkcji są adres obiektu obsługującego interfejs UART, dziesięciobajtowy obszar pamięci ( tablica dziesięciu bajtów utworzona w globalnej przestrzeni pamięci ) oraz długość tego obszaru.

Asynchroniczność transmisji możliwa jest poprzez obsługę przerwania które generuje UART po zakończeniu transmisji. Biblioteki HAL zapewniają niskopoziomową obsługę przerwań oraz deklaracje zestawu funkcji zwrotnych które zostają wykonane w odpowiedzi na zarejestrowane przerwanie. Obsługa zakończenia transmisji:

|  |  |  |
| --- | --- | --- |
|  | voidHAL\_UART\_RxCpltCallback(UART\_HandleTypeDef \*huart)  {  int8\_t response[10];  if(getTimerTimeout(&htim9))  stopBluetoothTimer();  commandHandler(command,response);  HAL\_UART\_Transmit\_DMA(&huart2,response,10);  if(getTimerTimeout(&htim9))  startBluetoothTimer();  } |  |

Funkcja jest przez bibliotekę HAL wywoływana dla zarejestrowanego przerwania od dowolnego interfejsu UART. Obiekt UART komponentu który wywołał przerwanie jest przekazywany jako argument funkcji. Z uwagi na to, że używany jest jeden interfejs szeregowy nie ma potrzeby sprawdzania czy to on jest źródłem przerwania. Funkcja tworzy tablicę bajtów - response, która wraz z kopią odebranej komendu przekazywana jest do funkcji obsługującej daną wiadomość. Wypełniona odpowiedź wysyłana jest tą samą techniką. Obsługa wiadomości oraz zastosowany timer opisany jest w dalszej części tego rozdziału.

Problem, jaki należy rozważyć, w przypadku komunikacji asynchronicznej, jest sytuacja, w której wiadomość jest odbierana zanim wcześniejsza zostanie obsłużona. Dodatkową komplikacja, w tej sytuacji jest użycie DMA, które wypełnia wartości w pamięci z pominięcie pracy, a tym samym kontroli procesora. Praca, na tym samym obszarze pamięci, które wypełnia DMA, podczas obsługi, mogłaby skutkować zmianą wartości tablicy podczas odczytu z niej wiadomości. Z tego powodu, pierwszym etapem w obsłudze przerwania, jest utworzenie kopii tablicy *command* i praca na duplikacie. To rozwiązanie, zwiększa stabilność przetwarzania wiadomości, ale nie rozwiązuje problemu całkowicie. Nowa wiadomość mogłaby pojawić się podczas tworzenia duplikatu, co znów generuje ten sam problem. Z pomocą przychodzi tutaj sprzętowa obsługa przerwań w mikrokontrolerach STM. Przetworzenie wiadomości wywoływana jest w obsłudze przerwania, które kończy się dopiero po zaaplikowaniu komendy. Kontroler przerwań NVIC, w przypadku takiego samego priorytetu, tworzy kolejkę LIFO przerwań i a następnie kolejno przekazuje sterowanie, po zakończeniu aktualnie wywołanej funkcji obsługi. Z tego powodu handler nie wywoła się przed zakończeniem przetwarzania bieżącej komendy. Poza wyżej wymienionymi zabezpieczeniami omawiany błąd eliminowany jest przez konstrukcję protokołu komunikacyjnego z aplikacją sterującą. Zgodnie z nim, aplikacja nie wysyła, następnej wiadomości bez otrzymania odpowiedzi na poprzednią, w przypadku prawidłowej wymiany danych, rozwiązuje ten problem.

## Proces przetwarzania wiadomosci

W poprzednim rozdziale zaprezentowana jest obsługa przerwania od zakończenia transmisji szeregowej. Obecna jest funkcja „commandHandler” odpowiedzialna za obsługę wiadomości i uformowanie odpowiedzi. Struktura funkcji „commandHandler” ( dla czytelności pominąłem fragment funkcji,n gdyż nie wpływa to na zrozumienie jej działania):

|  |  |  |
| --- | --- | --- |
|  | int8\_t\* commandHandler(int8\_t\* command,int8\_t\* response)  {  if(command[0] != ‘B’ || command[4] != ‘M’ || command[9] != ‘E’)  returninvalidMessage(response);  if(command[1] == 1)  {  //set speed  returnsetVelocityResponse(command,response);  }  if(command[1] == 2)  {  //get speed  returngetVelocityResponse(command,response);  }  //(…) pominęta część funkcji  if(command[1] == 14)  {  //set direction pins  returnsetMotorsDirection(command,response);  }  if(command[1] == 15)  {  //get direction pins  returngetMotorsDirection(command,response);  }  returninvalidMessage(response);  } |  |

Działanie prezentowanej funkcji jest ściśle związane z protokołem komunikacyjnym pomiędzy robotem a aplikacją sterującą. Zagadnienie to jest szczegółowo opisane w rozdziale piątym. Początek funkcji sprawdza czy na odpowiednich pozycjach występują wartości stałe i charakterystyczne dla każdej wiadomości, w przeciwnym wypadku wysyłana jest wiadomość informująca o błędzie. Następnie na podstawie identyfikatora wiadomości wywoływana jest konkretna funkcja obsługująca daną wiadomość, a jeżeli podany identyfikator nie jest znany zwracany jest błąd.

Obsługa wiadomości zaprezentowana będzie na przykładzie komendy ustalające prędkość zadaną w układzie regulacji:

|  |  |  |
| --- | --- | --- |
|  | int8\_t\* setVelocityResponse(int8\_t\* command,int8\_t\* response)  {  uint8\_tindex = 0;  for(;index<10; ++index)  response[index] = 0;  response[0] = ‘B’;  response[1] = command[1];  response[2] = command[2];  response[3] = command[3];  response[4] = command[5];  response[5] = command[6];  if(command[2] == 0 && command[3] == 0 && command[5] == 0 && command[6] == 0)  {  stopMotors();  return response;  }  int8\_t data[2];  data[0] = command[2];  data[1] = command[3];  setLeftSpeed = byte2Float(data);  data[0] = command[5];  data[1] = command[6];  setRightSpeed = byte2Float(data);  if(command[7] != 0 && command[8] != 0)  {  data[0] = command[7];  data[1] = command[8];  uint32\_t timer = byte2int(data);  startSpeedTimer(timer);  }  return response;  } |  |

Każda funkcja początkowo wypełnia odpowiedź zerami i wartościami stałymi. Dla wiadomości typu „*ustaw”* odpowiedź powinna być identyczna jak komenda przychodząca. Dzięki temu aplikacja sterująca może mieć pewność, że ustawione zostały podane wartości. Pobór fatycznych wartości ustawionych na robocie odbywa się za pomocą wiadomości typu „*pobierz”*. Zaprezentowana wiadomość ustawia 3 pozycje: prędkość lewego koła, prędkość prawego koła (w cm/s ) oraz opcjonalnie czas trwania zadanego ruchu. Wiadomości ustawiające prędkość charakteryzują się jeszcze jedną właściwością. Mając wszystkie pola wartości wypełnione zerami wywołuje się funkcja zatrzymania silników robota. Jest to jedna z form zabezpieczenia. Warunek ten jest sprawdzany od razu po uformowaniu odpowiedzi. Kolejnym etapem jest konwersja odpowiednich bajtów na liczbę zmiennoprzecinkową i zapis wartości do globalnej zmiennej określającej zadaną prędkość dla danego koła. Ostatnim etapem obsługi wiadomości jest ustawienie czasu trwania długości danego ruchu. Jeżeli pola nie są wypełnione zerami wystartuje timer z przesłanymi danymi ilościami milisekund, i po tym czasie zostaną zatrzymane silniki. W przeciwnym razie zadana prędkość będzie się utrzymywała aż do jej zmiany. Po zakończeniu procedury zwracany jest adres z odpowiedzią.

## Zabezpieczenia robota

Podstawową funkcją wykorzystywaną we każdej formie zabezpieczeń robota jest *stopMotors*:

|  |  |  |
| --- | --- | --- |
|  | voidstopMotors(void)  {  setLeftSpeed = 0.0;  setRightSpeed = 0.0;  LeftPWM(0);  RightPWM(0);  stopSpeedTimer();  } |  |

Funkcja wyzerowuje wartość zadaną oraz ustawiony PWM, jak również zatrzymuje timer odpowiedzialny za czas trwania ustawionej prędkości. Robot posiada 3 mechanizmy zabezpieczeń. Pierwszym jest wymienione w poprzednim rozdziale wysłanie wiadomości zadającej prędkość z wartościami równymi zero. Kolejnym mechanizmem jest wysłanie niepoprawnej wiadomości. Funkcja przygotowująca odpowiedź informującą o błędzie również zatrzymuje silniki. Ostatnim zabezpieczeniem jest możliwość ustawienia timera Bluetooth ( widoczny jest we fragmencie kodu w podrozdziale 3.2 ). Timer zaczyna odliczanie po otrzymaniu dowolnej wiadomości i gdy następna wiadomość nie pojawi się przed zakończeniem odliczania, silniki są zatrzymywane. Jest to forma zabezpieczenia przed utratą łączności. Domyślnie timer ten jest wyłączony. Włączenie oraz ustalenie czasu odliczania jest możliwe za pomocą wiadomości kontrolnej.

## Realizacja odczytu prędkości

Aby możliwa była regulacja prędkości wymagane jest sprzężenie zwrotne od faktycznej prędkości osiągniętej przez robota. W tym celu kluczowe jest niezawodne przetwarzanie informacji uzyskanych za pomocą enkoderów. Enkodery zamontowane na wale silników podłączone są to pinów skonfigurowanych jako wejścia kanałów timerów obsługujących ten czujniki. Mikrokontroler STM umożliwia skonfigurowanie timera do trybu *Encoder Mode*, który wiąże wejście sygnału taktującego licznika z impulsem generowanym przez enkoder. Tryb ten również pozwala na sprzętową analizę kolejności generowanych impulsów wyjściowych, co w wygody sposób pozwala na detekcje kierunku obrotu koła. (…). Do obliczania prędkości użyty został jeszcze jeden niezależny timer, który o ustawiony interwał aktualizuje globalne zmienne zawierające aktualne prędkości kół robota.

|  |  |  |
| --- | --- | --- |
|  | voidHAL\_TIM\_PeriodElapsedCallback(TIM\_HandleTypeDef \*htim)  {  if(htim == &htim2)  {  leftVelocity = LeftVelocity();  rightVelocity = RightVelocity();  htim2.Instance->CNT = encoderTimer - 1;  HAL\_TIM\_Base\_Start\_IT(&htim2);  }  //( … )  else if(htim == &htim3)  {  onEncoderOverload(&htim3,&leftTotalTicks);  }  Else if(htim == &htim1)  {  onEncoderOverload(&htim1,&rightTotalTicks);  }  } |  |

Jest to uniwersalny handler dla wszystkich timerów zarejestrowanych na przerwanie od końca odliczania. Biblioteka HAL obsługuje niskopoziomowo przerwanie timera i wywołuje funkcje obsługi przekazują w argumencie adres obiektu kontrolującego dany licznik. Timer (htim2) odpowiedzialny za aktualizację prędkości jest obsługiwany na początku zapisuje wyniki funkcji obliczających prędkość oraz zapisuje je zmiennych globalnych. Interwał czasowy dla licznika może być ustalony przez wiadomość kontrolną. Obliczona wartość jest więc średnią prędkością w ustawionej ramie czasowej. Z powodu, że interwał aktualizacji prędkości może być ustalany dowolną szesnastobitową wartością oraz maksymalna prędkość jaką mogą uzyskać koła nie jest stała (zależy od zużycia silników, poziomu naładowania baterii, użytej przekładni i innych czynników ) wymagane było zabezpieczenie przed przekroczeniem maksymalnej wartości timera zanim nastąpi aktualizacja prędkości. Obsługę tej sytuacji zapewnia funkcja:

|  |  |  |
| --- | --- | --- |
|  | void onEncoderOverload(TIM\_HandleTypeDef \*htim,int32\_t \*totalTickCounter)  {  if(htim->Instance->CNT >64000 && htim->Instance->CNT <65000)  \*totalTickCounter += (ENCODER\_INITIAL + 65000 - htim->Instance->CNT);  else if (htim->Instance->CNT >0 && htim->Instance->CNT <1000)  \*totalTickCounter -= (ENCODER\_INITIAL + htim->Instance->CNT);  htim->Instance->CNT = ENCODER\_INITIAL;  HAL\_TIM\_Base\_Start\_IT(htim);  } |  |

Argumentami jest adres danego timera oraz adres zmiennej pomocniczej określającej ilość impulsów wysłanych przez enkoder od ostatniej aktualizacji prędkości. Zmienna „*totalTickCounter*” jest ze znakiem co umożliwia określenie zwrotu prędkości. Prędkość zgodna z przyjętą przednią częścią robota przekracza licznik zmniejszając wartość rejestru przy wartości 0, w związku z czym pod rejestrem, podczas obsługi przerwania, jest zapisana wartość mniejsza od 65000. Dolne ograniczenie zostało dobrane empirycznie. Analogicznie sytuacja wygląda przy przepełnieniu wartości timera odwrotną prędkością. Zmienna pomocnicza jest inkrementowana lub dekrementowana o obliczoną wartość impulsów.

Obliczenie prędkości podczas aktualizacji odbywa się za pomocą funkcji „*LeftVelocity*” i „*RightVelocity*”. Obliczanie prędkości koła na przykładzie funkcji „*LeftVelocity*”:

|  |  |  |  |
| --- | --- | --- | --- |
|  | |  | | --- | | floatLeftVelocity(void)  {  leftTotalTicks += (ENCODER\_INITIAL -htim3.Instance->CNT);  htim3.Instance->CNT = ENCODER\_INITIAL;  float vel = 0.0f;  vel = calcVelocity(leftTotalTicks);  leftTotalTicks = 0;  return vel;  } | |  |

Funkcja dodaje do zmiennej „*leftTotalTicks*” ilość impulsów wygenerowanych przez enkoder od ostatniej aktualizacji prędkości. Następnie zeruje ilość impulsów, oblicza prędkość liniową w cm/s i zwraca ją. Funkcja obliczająca prędkość liniową:

|  |  |  |
| --- | --- | --- |
|  | floatcalcVelocity(int8\_t encoderTicks)  {  float tmp = encoderTicks/TICK\_PER\_ROUND;  tmp = tmp\*2.0\*SHORT\_PI;  tmp = tmp /(getTimerTimeout(&htim2)\*0.0001);  tmp = tmp\*RADIUS;  return tmp;  } |  |

W pierwszym kroku obliczana jest ilość obrotów jakie wykonało koło biorąc po uwagę przekładnie silnika i rozdzielczość samego czujnika . Następnym krokiem jest zamiana ilości obrotów na prędkość kątową wyrażoną w radianach na sekundę. Ostatnim etapem jest uzyskanie prędkości liniowej mnożąc prędkość kątową przez promień koła i zwrócenie wartości.

Przedstawione rozwiązanie jest odporne na duże prędkości, niemniej jednak może się nie sprawdzać w przypadku małych prędkości. Korzystając z odometrii należy być świadomym niepewności pomiarowych w odczycie ilości impulsów w stałej ramie czasowej, wywołanej brakiem synchronizacji generowanego przez czujniki sygnału i timera wyzwalającego aktualizację. Problem pogłębia fakt, że sama rama czasowa może być ustalona z zewnątrz podczas pracy robota. Bezpieczniejsze byłoby zablokowanie możliwości zmiany ustawień timera, jednakże byłoby to sprzeczne z ideą projektu. Przyjęta więc dobrana empirycznie wartość nominalna 60 [ms], a zamianę tej wartości przejmuję pełną odpowiedzialność użytkownik.

W nominalnym interwale czasowym problem pojawia wariancja odczytu impulsów rośnie wraz ze zmniejszaniem prędkości. Dla stosunkowo dużych wartość wariancja odczytanych impulsów jest na tyle niewielka, że można ją pominąć, natomiast przy niewielkich

## Układ regulacji

Układ regulacji opiera się na regulatorze PD.

|  |
| --- |
| Schemat układu regulacji |
| C:\Users\Kornel\Desktop\pracka\regulacja.png |

Bloki o podwójnym obramowaniu oznaczają stan robota, natomiast o pojedynczym reprezentują operację. Strzałki pokazują przepływ danych podczas regulacji prędkości.

Regulator został zaimplementowany zgodnie z przyjętymi wcześniej założeniami.

|  |  |  |
| --- | --- | --- |
|  | void regulation(void)  {  eLeft = setLeftSpeed - leftVelocity;  eRight = setRightSpeed - rightVelocity;  newPWMLeft = pVal \* eLeft + dVal \* (eLeft - peLeft);  newPWMRight = pVal \* eRight + dVal \* (eRight - peRight);  peLeft = eLeft;  peRight = eRight;  addNewPWM(newPWMLeft,newPWMRight);  } |  |

giug

# Oprogramowanie aplikacji sterującej

## Struktura oprogramowania

Aplikacja do sterowania zdalnego robotem została napisana w języka Java. Kod źródłowy jest kompilowany do kodu bajtowego ( forma pośrednia pomiędzy kodem źródłowym, a instrukcjami maszynowymi ) i może być uruchamiany na każdej platformie posiadającej Maszynę Wirtualną Javy. Wybór języka wynika, z wieloplatformowości powstałego programu oraz osobistym doświadczeniu w tworzeniu graficznego interfejsu użytkownika.

Aplikacja składa się z trzech podstawowych komponentów:

1. modułu kontroli komunikacji Bluetooth
2. modułu sterowania tekstowego
3. modułu sterowania graficznego

Poza tymi, istnieje jeszcze jeden komponent odpowiedzialny ze utworzenie wyżej wymienionych modułów oraz posiada zbiór funkcji używanych przez wszystkie moduły. Wszystkie komponenty w kodzie źródłowym zorganizowane są w formie pakietów Java. Pakiet Java jest mechanizmem języka, który pozwala grupować powiązane ze sobą klasy, nadając im jednocześnie wspólną przestrzeń nazw.

## Moduł kontroli komunikacji Bluetooth

Moduł ten jest komponentem, którego zadaniem jest obsługiwanie transmisji szeregowej Bluetooth z robotem. Moduł definiuje abstrakcyjny interfejs który pozwala na wymianę informacji z robotem, jednocześnie ukrywając techniczne szczegóły realizacji transmisji danych.

|  |  |  |
| --- | --- | --- |
|  | public interface IBluethooth {    public String setVelocity(double translation, double rotation, int time) throws SerialPortException;  public String getVelocity() throws SerialPortException;  public String setConfig(double P, double D) throws SerialPortException;  public String getConfig() throws SerialPortException;  public String setTimeout(int timeout) throws SerialPortException;  public String getTimeout() throws SerialPortException;  public String setPWM(int left, int right, int time) throws SerialPortException;  public String getPWM() throws SerialPortException;  public String setEncoderMeas(int time) throws SerialPortException;  public String getEncoderMeas() throws SerialPortException;  public String setRegulationTimer(int time) throws SerialPortException;  public String getRegulationTimer() throws SerialPortException;  public String setMotorDirection(byte bitmap) throws SerialPortException;  public String getMotorDirection() throws SerialPortException;  public String getSetSpeed() throws SerialPortException;  } |  |

Interfejs udostępnia szereg funkcji, do wymiany danych z robotem. Argumenty funkcji posiadają typ danych, który w naturalny sposób odzwierciedla wartość jaką wyraża. Przykładem może być, zadana prędkość, która posiada typ double, czy czas ekspiracji timera, wyrażony w liczbach całkowitych. Odpowiedzialność za przygotowanie odpowiedniej wiadomości, zgodnej z zaprojektowanym protokołem przeniesiona jest na ten komponent. Warto również zauważyć, że funkcja zadająca prędkość (setVelocity) posługuje się argumentami : Translacja, Rotacja. Z poziomu aplikacji użytkownik korzysta z zadawania prędkości translacji ( ruchu postępowego robota w dwuwymiarowym układzie współrzędnych) i rotacji ( ruchu obrotowego ) , a nie prędkościami poszczególnych kół. Z punktu widzenia użytkownika jest to bardziej intuicyjne rozwiązanie. Pobieranie wartości również jest przekształcane z prędkości lewego i prawego koła na prędkość translacji i rotacji. Transformacje za pomocą funkcji:

|  |  |  |
| --- | --- | --- |
|  | public static double[] TranRot2LeftRight(double translation, double rotation)  {  double[] result = new double[2] ;  result[0] = translation + rotation;  result[1] = translation - rotation;  return result;  }  public static double[] LertRight2TranRot(double left,double right)  {  double[] result = new double[2] ;  result[0] = (left + right)/2;  result[1] = (left - right)/2;  return result;  } |  |

Każda funkcja zwraca łańcuch znaków zawierająca sformatowaną odpowiedź robota. Wybór tego typu jest omówiony w podrozdziale 4.3. Interfejs posiada dwie implementacje:

* klasa, która symuluje działanie robota, używana we wstępnej fazie implementacji oraz do testowania działania aplikacji bez połączenia z robotem
* klasa, która obsługuje połączenie szeregowe poprzez wirtualny port COM z modułem Bluetooth na robocie oraz tworzy właściwe wiadomości gotowe do transmisji

Istotną koncepcją przy tworzeniu było zapewnianie, aby tylko jedna instancja obiektu była tworzona podczas pracy programu. Jak w przypadku klasy symulującej nie ma to większego znaczenia, jest niezbędne we właściwej implementacji. Wynika to z faktu, że nawiązywana jest transmisja szeregowa, która blokuje dany port COM. Próba utworzenia kolejnej instancji zakończy się błędem. Rozwiązane to jest poprzez zastosowanie wzorca projektowego singleton.

Sama obsługa portu szeregowego wykorzystuje bibliotekę jSSC udostępnioną na otwartej licencji. Obsługa portów szeregowych jest ściśle związana z systemem operacyjnym i wymagana jest ich natywna implementacja, specyficzna dla danego systemu. Zastosowana biblioteka posiada zaimplementowaną natywną obsługę transmisji szeregowej dla większości znanych platform oraz udostępnia abstrakcyjny interfejs w języku Java, który pozwala wykorzystanie go w programie.

## Moduł sterowania tekstowego

Głównym założeniem przy projektowaniu aplikacji sterującej było umożliwienie użytkownikowi tworzenia skryptów pozwalający implementować dowolny algorytm oraz interakcję z robotem za pomocą komend tekstowych. Wymagało to stworzenie modułu o podstawowej funkcjonalności edytora testowego. Założeniem była również interakcja z poziomu konsoli interpretera, który swoją funkcjonalnością będzie przypominał inne znane interpretery języków skryptowych.

|  |
| --- |
| Okno panelem interpretera z interpreterem |
| C:\Users\Kornel\Desktop\pracka\iterpreter1.png |

Interpreter wykonuje pojedynczą instrukcję, zakończoną średnikiem. Klawisz ENTER rozpoczyna wykonywanie komendy, natomiast klawiszami strzałek możemy przeglądać historie wykonanych instrukcji. Interpreter nie jest związany z panelem sterowania umieszczonym powyżej, interakcja odbywa się tylko za pomocą klawiszy.

|  |
| --- |
| Okno z panelem skryptu |
| C:\Users\Kornel\Desktop\pracka\skryptowanie.png |

Panel edycji skryptów posiada funkcjonalność podstawowego edytora testu. Górne pole tekstowe służy do tworzenia kodu, natomiast dolne jest przekierowanym standardowym wyjściem silnika JavaScript. Powyżej pól tekstowych znajduję się pasek narzędzi oraz menu. Wszystkie funkcje wspomagania edycji dostępne są z poziomu menu, paska narzędzi oraz skrótów klawiszowych. Edytor umożliwia:

* uruchomianie skryptu
* zapis i odczyt plików z dysku
* czyszczenie pola wprowadzania testu i pola wyjściowego
* prosty mechanizm przeszukiwania tekstu
* zakończenie pracy edytora

Ważnym aspektem modułu jest fakt, że zarówno interpreter, jak i edytor korzystają z tej samej instancji silnika JavaScript. Oznacza to, że działania wykonane na jednym panelu widoczne są w drugim. Nie stoi nic na przeszkodzi, aby konfiguracje robota przeprowadzić za pomocą edytora skryptów, natomiast same sterowanie odbyło się za pomocą panelu interpretera.

## Użycie silnika JavaScript w aplikacji

Możliwość pisania skryptów wymagała użycia silnika jednego z języków skryptowych. Wybór padł na JavaScript, który główne zastosowanie ma w stronach internetowych. Język ten pozwala na pisanie pełnoprawnych aplikacji oraz realizacji dowolnego algorytmu. Głównym powodem była obecność silnika *Nashorn* w Java 8 . *Nashorn* jest wydajną implementacją języka JavaScript pozwalającą łączyć funkcjonalność obu języków.

|  |  |  |
| --- | --- | --- |
|  | private ScriptEngine engine;  // (…)  public JavaScriptInterpreter()  {  engine = new ScriptEngineManager().getEngineByName("nashorn");  stringWriterInit();  engine.getContext().setWriter(stringWriter);    loadInitScript();  } |  |

Java 8 udostępnia możliwość stworzenia obiektu klasy *ScriptEngine*, który zarządza silnikiem JavaScript. Konstruktor obiektu ustawia jako strumień wyjściowy obiekt tekstowy *stringWriter* powiązany z oknem aplikacji oraz uruchamia skrypt początkowy, umieszczając tym samym w przestrzeni JavaScript definicję funkcji używane przy komunikacji z robotem.

|  |  |  |
| --- | --- | --- |
| Uruchamianie kodu JavaScript | | |
|  | @Override  public String evaluateCommand(String command) throws ScriptException {  engine.eval(command);  String res = stringWriter.toString();  stringWriter.getBuffer().setLength(0);  return res;  }  (…) |  |
|  | @Override  public void runStript(String text, Boolean isFromFile) throws Exception{  stringWriter.setIsEvent(true);  if(isFromFile)  engine.eval(new FileReader(text));  else  engine.eval(text);  stringWriter.setIsEvent(false);    } |  |

Przedstawione metody używane są przy wywoływaniu skryptów. Pierwsza funkcja używana jest przez panel interpretera. Po wykonaniu fragmentu kodu ( „*engine.eval(command);”)*czyści bufor a następnie umieszcza w nim dane odebrane z standardowego wyjścia silnika. Druga funkcja używana jest w edytorze skryptów. Pozwala na uruchamianie zarówno plików tekstowych jak i ciągów znaków. W tej metodzie przekierowanie strumienia wyjściowego opiera się na obsłudze generowanych prze obiekt *stringWriter* zdarzeń.

|  |  |  |
| --- | --- | --- |
| Fragment inicjalnego skryptu umożliwiającego kontakt z robotem | | |
|  | var BluethoothClass = Java.type('bluethoothCtrl.BluetoothImp');  var btObject = BluethoothClass.getInstance();  //(…)  function RobotSetVel(translate,rotate,time)  {  var result = btObject.setVelocity(translate,rotate,time);  return decode(result);  }  function RobotGetVel()  {  var result = btObject.getVelocity();  return decode(result);  }  //(…) |  |

Sam inicjalny skrypt ładowany podczas konstruktora klasy opakowuje wszystkie metody interfejsu *IBluethooth.* Skrypt uzyskuję dostęp do klasy Java ( jedna z wspomnianych wcześniej możliwości silnika *Nashorn* ) oraz pobiera statyczny obiekt tej klasy. Każda funkcja wywołuje metodę z interfejsu *IBluethoot*, a następnie zwraca zdekodowany do tablicy łańcuch znaków. Powodem, dla którego w interfejsie *IBluethooth* każda metoda zwraca ten typ jest fakt, że z poziomu JavaScript pobieranie zmiennej tekstowej działa zawsze zgodnie z oczekiwaniami, czego nie można powiedzieć o innych typach danych.

## Moduł sterowania graficznego

Wykorzystanie potencjału sterowanie tekstowego wymaga podstawowej wiedzy programistycznej i poświęcenia czasu aby w pełni poprawnie przygotować skrypt. Poza tym moduł ten nadaje się do implementacji algorytmu poruszania się robota, niekoniecznie natomiast sprawdza się w sytuacji gdy chcemy , aby robot reagował na nasze działania w czasie rzeczywistym. Odpowiedzią na ten problem jest moduł sterowania graficznego. Idea tego komponentu polega na wykorzystaniu bloków sterujących, które pozwalająca komunikację z robotem za pomocą interfejsu graficznego. Bloki sterujące pozwalają na zmianę konfiguracji robota, kontrolę i wizualizację prędkości bez konieczności używanie komend tekstowych.

|  |
| --- |
| Okno modułu graficznego |
|  |

Główne okno modułu graficznego pozwala wybierać bloki, które chcemy aktualnie użyć przy sterowaniu robotem, spośród wszystkich dostępnych. Dodatkowo w menu umieszczone są dwie pomocnicze opcje pozwalające na dodawanie bądź usunięcie wszystkich bloków. Bloki sterujące są szczegółowo opisane w następnych podrozdziałach

Samo okno aplikacji pozwala dynamicznie zarządzać blokami na ekranie. W zależności od rozmiaru okna bloki układają się w sposób, aby jak najefektywniej wykorzystać dostępną przestrzeń. Rozkład elementów można kontrolować poprzez kolejność dodawania bloków do głównego okna. Minimalizacja i zamknięcia okna wysyłają komendę *stop* do robota.

## Bazowy blok sterujący

Podstawą każdego bloku sterującego jest klasa bazowa *BaseTile* (nazwa oznacza „bazowy kafelek”), która definiuje podstawowe parametry i zachowanie każdego elementu. Klasa *BaseTile* jest odpowiedzialna również a za określenie rozmiaru bloku oraz podstawowe elementy wizualne i funkcjonalne bloków sterujących. Każdy blok w górnej części posiada obszar aktywacji:

|  |
| --- |
| C:\Users\Kornel\Desktop\pracka\bazowy.PNG |

Przycisk z krzyżykiem odpowiada za wyłączenie elementu i jest jednoznaczny z odznaczeniem elementu w głównym oknie modułowym. Przycisk aktywacji określa włączenie działania bloku i jest podstawowym mechanizmem w synchronizacji pracy. Współpraca bloków opisana jest w podrozdziale 4.4.3. Ich wymiary są ustalone i nie podlegają modyfikacji. Blok może modyfikować udostępnione przez klasę BaseTile ciało elementu.

Właściwości wspólne dla każdego elementu :

* nazwa – każdy blok musi wprowadzić swoją unikalną nazwę
* typ - blok może by jednego z trzech typów
* STEROWANIE – element kontrolujący ( zadający) prędkość robota
* WIZUALIZACJA – element wizualizujący faktyczną prędkość robota
* KONFIGURACJA – element, który nie steruje prędkością, może natomiast zmieniać konfiguracje robota np. nastawy regulatora
* definiuje podstawową funkcję aktywacji i dezaktywacji elementu. Element może rozszerzyć tą funkcjonalność, niemniej jednak musi wtedy wywołać funkcje klasy bazowej
* definiuje funkcje wyłączenia elementu

## Bloki sterujące dostępne w aplikacji

W aplikacji stworzone zostały 3 bloki sterujące, po jednym w każdym typie.

|  |  |
| --- | --- |
| Nazwa | **Sterowanie Myszką** |
| Typ | **STEROWANIE** |
| C:\Users\Kornel\Desktop\pracka\mause1.png | C:\Users\Kornel\Desktop\pracka\metro2.PNG |
| Blok umożliwia zadawanie prędkości za pomocą myszki. W centralnej części elementu znajduje się obszar obramowany czarowną linią i główny panel sterowania. Naprowadzenie kursora spowoduje zmianę koloru z szarego na biały, natomiast kliknięcie myszką rozpocznie sterowanie. Po rozpoczęciu sterowania nad kursorem pojawi się napis określający zadaną prędkość translacji i rotacji robota wyrażony w cm/s. Punkt zerowy jest w miejscu kliknięcia, które rozpoczyna sterowanie robotem. Na głównym panelem znajduje się informator aktualnego stanu bloku. Pod głównym panelem jest możliwość ustawienia przeskalowania ruchu myszki ( zmiana o ilości pikseli ) na zadaną prędkość. Praktyka pokazuje, że lepiej steruje się robotem, gdy przeskalowanie translacji jest większe niż rotacji. Przesunięcie myszki w pionie zmienia rotację, natomiast przesunięcie w poziomie zmienia zadaną rotację. Można zadawać prędkości zarówno dodatnie jak i ujemne. Ponowne kliknięcie, bądź wyjście kursora poza panel główny powoduje zakończenie sterowania. | |

|  |  |
| --- | --- |
| Nazwa | **Prędkościomierz** |
| Typ | **WIZUALIZACJA** |
| C:\Users\Kornel\Desktop\pracka\speed1.png | C:\Users\Kornel\Desktop\pracka\speed2.png |
| Prędkościomierz pokazuje aktualną prędkość robota. Lewy panel odpowiada za wyświetlenie prędkości natomiast prawy za formę w jakiej prędkość ma być wyświetlana. Za pomocą prędkościomierza może ustawić jednostkę w jakiej chcemy prezentować prędkość:   * centymetry na sekundę(domyślnie) * metry na sekundę * kilometry na godzinę   Kolejną opcją jest ustawienie formy wyświetlenia prędkości:   * prędkość translacji i rotacji robota ( domyślnie) * prędkość lewego i prawego koła robota | |

|  |  |
| --- | --- |
| Nazwa | **Konfiguracja** |
| Typ | **KONFIGURACJA** |
| C:\Users\Kornel\Desktop\pracka\conf1.png | C:\Users\Kornel\Desktop\pracka\conf2.png |
| Dzięki temu blokowi możemy ustalić wszystkie dane konfiguracyjne robota. Blok składa się z tabeli wartości oraz przycisków służących do konfiguracji. Tabel posiada cztery kolumny:   * parametr (pierwsza kolumna) : określa ustawionyparametr * komponent (druga kolumna) : określa czy dany parametr dotyczy konfiguracji robota czy aplikacji * aktualna wartość (trzecia kolumna):pokazuje aktualnie ustawiony parametr * wartość do ustawienie(czwarta kolumna):jedyna gdzie pola można modyfikować. Pozwala na wpisanie wartości, którą będziemy mieli możliwość ustawić   Blok pozwala na modyfikację:   * nastawy regulatora PD na robocie * ustawianie timera zabezpieczającego przed utratą łączności Bluetooth ( *Timeout*) * ustawianie timera aktualizacji prędkości(*Encoder*) oraz interwału regulatora(*Regulation*) * ustawienie timera modułu graficznego (*Timer*) , omówiony szczegółowo w podrozdziale 4.4.3 | |

## Synchronizacja aktywności bloków

Architektura blokowa modułu graficznego pozwala na tworzenie nieograniczonej ilości bloków sterujących dowolnie wybranego typu. Kontrolowanie prędkości robota może być, oprócz sterowania myszką realizowane przez klawiaturę, joystick czy nawet komendy głosowe. Konfigurowanie robota również może przyjmować wiele różnych form. Z tego powodu wymagane było wprowadzenie mechanizmu synchronizacji pracy bloków. Klasa, która odpowiada za zarządzanie blokami sterującymi jest *BTHandler*. Oprócz synchronizacji klasa ta bierze odpowiedzialność za kontrolowanie dostępu do komponentu Bluetooth dla bloków funkcyjnych.

Podstawowym elementem, służącym synchronizacji jest przytoczony wcześniej mechanizm aktywacji. Aby działanie komponentu było możliwe, musi być on aktywowany. Uruchomienie elementu następuje w dwóch etapach. Pierwszy jest w obszarze obiektu klasy *BTHandler*. Obiekt posiada informacje o wszystkich blokach w aplikacji i aktywowanie komponentów podlega niżej wymienionym zasadom:

* aktywny może być tylko jeden komponent typu STEROWANIE lub KONFIGURACJA. Uruchomienie więcej niż jednego komponentu tego typu powoduje dezaktywacje bloku aktualnie pracującego
* aktywnych może być dowolna ilość bloków typu WIZUALIZACJA
* timer modułu graficznego rozpoczyna odliczanie gdy przynajmniej jeden komponent typu STEROWANIE lub WIZUALIZACJA jest aktywny, a wyłącza się gdy nie jest aktywny żaden z tych elementów
* wyłączenie elementu powoduje jego automatyczną dezaktywację
* minimalizacja lub wyłączenie okna powoduje dezaktywację wszystkich elementów

Drugi etap aktywacji realizowany jest w przestrzeni danego bloku i przygotowuje go do pracy. Wymiana danych bloków typu STEROWANIE i WIZUALIZACJE jest cyklicznie uruchamiana przez, wymieniony już wcześniej, timer aplikacyjny. Licznik ten co podany okres uruchamia swoją funkcję obsługi:

|  |
| --- |
| private class Task implements ActionListener  {  private int currentVal = 0;  private const int typeNumber = 1;  double[] velocity;    private void iterateCurrentVal()  {  if(currentVal != typeNumber)  currentVal++;  else  currentVal = 0;  }    @Override  public void actionPerformed(ActionEvent e) {  try{  if(setSpeedObs != null && currentVal == 0)  {  velocity = setSpeedObs.update(null);  bluethooth.setVelocity(velocity[0], velocity[1], 0);  System.out.println("SET Tr: "+velocity[0]+" Rot: "+ velocity[1]);  }  if(!getSpeedObs.isEmpty() && currentVal == 1)  {  String tmp = bluethooth.getVelocity();  velocity = GeneralConverter.deserializeStr2Dbl(tmp);  System.out.println("GET Tr: "+velocity[0]+" Rot: "+ velocity[1]);  for(BTObserver obs : getSpeedObs)  obs.update(velocity);  }    }catch(SerialPortException serialE) {}  System.err.println("Bluetooth transmission error!!!");  }    } |

Obsługa przepełnienia timera naprzemienne aktualizuje włączony blok sterujący i wszystkie bloki pobierające aktualną prędkość.

# Protokół komunikacyjny

Protokół komunikacyjny używany do wymiany danych pomiędzy robotem, a aplikacją składa się z 15 wiadomości kontrolnych. Wiadomości pozwalają pobierać i ustawiać dane na robocie. W większości występują w parach ustaw/pobierz. Protokół nie zakłada istnienia wiadomości jednocześnie zmieniających stan robota i pobierających aktualny stan. Prawidłowa odpowiedź na komendę typu ustaw jest identyczna do odebranej komendy, natomiast obsługa wiadomości typu pobierz nie ujmuje pod uwagę pól danych odebranej komendy.

Wiadomość składa się z 10 bajtów:

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Nr bajtu | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| Rodzaj wiadomości: | „B” | ID | Dane | Dane | „M” | Dane | Dane | Dane | Dane | „E” |

Bajty 0, 4 i 9 w każdej wiadomości są stałe i są równe wartości danej litery w kodzie ASCII. Służą one do kontroli poprawności przesłanej wiadomości.

Bajt 1 określa identyfikator danej wiadomości i na jego podstawie oprogramowanie robota jest w stanie uruchomić odpowiednią funkcję obsługi przesłanej wiadomości. Zostało to zaprezentowane w rozdziale 3.4.

Reszta bajtów przenosi dane konkretnej wiadomości. Wiadomości pozwalają na kodowanie liczby zmiennoprzecinkowej, jak również większych niż 1 bajt liczb całkowitych.

## Spis wiadomości kontrolnych

Każda wiadomość kontrolna przedstawiona jest według poniższego schematu:

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| NAZWA WIADOMOŚCI | | | | | TYP | | | IDENTYFIKATOR | | |
| BAJT 0 | BAJT 1 | BAJT 2 | BAJT 3 | BAJT 4 | BAJT 5 | BAJT 6 | BAJT 7 | | BAJT 8 | BAJT 9 |
| CHARAKTERYSTKA WIADMOSCI | | | | | | | | | | |

Dla wielkości USTAW w drugim wierszu zaprezentowana jest komenda którą odbiera robot, dla wiadomości typu POBIERZ jest to odpowiedź otrzymywana przez aplikację sterującą.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Ustawienie prędkości zadanej | | | | | USTAW | | | 1 | | |
| „B” | 1 | L1 | L2 | „M” | R1 | R2 | T1 | | T2 | „E” |
| Wiadomość ustawia prędkość zadaną dla regulatora.  Wartości L1 i L2 są zserializowaną zmienną zmiennoprzecinkową oznaczającą zadaną prędkość dla lewego koła  Wartości R1 i R2 są zserializowaną zmienną zmiennoprzecinkową oznaczającą zadaną prędkość dla lewego koła  Wartości T1 i T2 są rozłożoną 16-bitową zmienną całkowitą oznaczającą ilość milisekund po jakim zadana prędkość będzie wyzerowana. Jeżeli wartość T1 i T2 wynoszą zero timer odpowiedzialny za zatrzymanie silników po ustalonej wartości nie będzie uruchomiony a zadana prędkość nie będzie zmieniana.  Dane zawierające prędkości są deserializowane i zapisywane do odpowiadających im globalnych zmiennych robota. Jeżeli te dane zawierają wszystkie wartości równe zero silniki są zatrzymywane | | | | | | | | | | |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Pobranie faktycznej prędkości | | | | | POBIERZ | | | 2 | | |
| „B” | 2 | L1 | L2 | „M” | R1 | R2 | 0 | | 0 | „E” |
| Wiadomość zwraca faktyczną prędkość, jaką posiada robot.  Wartości L1 i L2 są zserializowaną zmienną zmiennoprzecinkową oznaczającą faktyczną prędkość dla lewego koła  Wartości R1 i R2 są zserializowaną zmienną zmiennoprzecinkową oznaczającą faktyczną prędkość dla lewego koła  Odsyłane prędkości są zserializowanymi zmiennymi globalnymi, których wartość obliczana jest w cyklu aktualizacji prędkości. Opis procedury jest w rozdziale 3.5 | | | | | | | | | | |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Ustawienie nastaw regulatora | | | | | USTAW | | | 3 | | |
| „B” | 3 | P1 | P2 | „M” | D1 | D2 | 0 | | 0 | „E” |
| Wiadomość ustawia nastawy regulatora.  Wartości P1 i P2 są zserializowaną zmienną zmiennoprzecinkową oznaczającą wartość wzmocnienia członu proporcjonalnego regulatora PD  Wartości P1 i P2 są zserializowaną zmienną zmiennoprzecinkową oznaczającą wartość wzmocnienia członu różniczkującego regulatora PD  Dane są deserializowane i zapisywane do odpowiadających im globalnych zmiennych robota. | | | | | | | | | | |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Pobranie nastaw regulatora | | | | | POBIERZ | | | 4 | | |
| „B” | 4 | P1 | P2 | „M” | D1 | D2 | 0 | | 0 | „E” |
| Wiadomość zwraca aktualne nastawy regulatora.  Wartości P1 i P2 są zserializowaną zmienną zmiennoprzecinkową oznaczającą wartość wzmocnienia członu proporcjonalnego regulatora PD  Wartości P1 i P2 są zserializowaną zmienną zmiennoprzecinkową oznaczającą wartość wzmocnienia członu różniczkującego regulatora PD  Odsyłane nastawy są zserializowanymi zmiennymi globalnymi. | | | | | | | | | | |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Ustawienie timera bezpieczeństwa | | | | | USTAW | | | 5 | | |
| „B” | 5 | T1 | T2 | „M” | 0 | 0 | 0 | | 0 | „E” |
| Wiadomość ustawia timer bezpieczeństwa. Timer jest startuje wraz z odebraniem wiadomości przez robota. Jeżeli nie otrzyma następnej wiadomości przed końcem odliczania robot zatrzyma silniki  Wartości T1 i T2 są rozłożoną 16-bitową zmienną całkowitą oznaczającą ilość milisekund co ile generowane jest przerwanie. | | | | | | | | | | |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Pobranie interwału timera bezpieczeństwa | | | | | POBIERZ | | | 6 | | |
| „B” | 6 | T1 | T2 | „M” | 0 | 0 | 0 | | 0 | „E” |
| Wiadomość zwraca ustawioną wartość milisekund po którym uruchamiamy jest timer bezpieczeństwa.  Wartości T1 i T2 są rozłożoną 16-bitową zmienną całkowitą oznaczającą ilość milisekund co ile generowane jest przerwanie.  Dane są składane w 16-bitową wartość całkowitą, która jest wykorzystywana jest w ustawieniach timera bezpieczeństwa. Timer został opisany w rozdziale 3.5. | | | | | | | | | | |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Ustawienie wartosci wypelnienia PWM | | | | | USTAW | | | 7 | | |
| „B” | 7 | L1 | L2 | „M” | R1 | R2 | T1 | | T2 | „E” |
| Wiadomość ustawia wartość sygnału PWM dla silników  Wartości L1 i L2 są rozłożoną 16-bitową zmienną całkowitą oznaczającą wartość sygnału PWM, który steruje lewym silnikiem.  Wartości R1 i R2 są rozłożoną 16-bitową zmienną całkowitą oznaczającą PWM który, steruje prawym silnikiem.  Wartości T1 i T2 są rozłożoną 16-bitową zmienną całkowitą oznaczającą ilość milisekund po jakim zadana prędkość będzie wyzerowana. Jeżeli wartość T1 i T2 wynoszą zero timer odpowiedzialny za zatrzymanie silników po ustalonej wartości nie będzie uruchomiony i a zadana prędkość nie będzie zmieniana.  Dane zawierające PWM są składane do zmiennych 16-bitowych i zapisywane ustawiana na timery generujące sygnały PWM Maksymalne wypełnienie wynosi 4200 i wysłanie prędkości większej spowoduję ustawienie tej wartości. Jeżeli te dane zawierają wszystkie wartości równe zero silniki są zatrzymywane  Ustawienie wartości sygnałów PWM jest niemożliwe, gdy uruchomiony jest timer odpowiedzialny za działanie regulatora prędkości. Nie można nastawiać bezpośrednio wartości wypełnienia sygnału PWM, gdy kontroluje tę wartość regulator. | | | | | | | | | | |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Pobranie wartości wypełnienia PWM | | | | | POBIERZ | | | 8 | | |
| „B” | 8 | L1 | L2 | „M” | R1 | R2 | 0 | | 0 | „E” |
| Wiadomość zwraca wartość sygnału PWM dla silników  Wartości L1 i L2 są rozłożoną 16-bitową zmienną całkowitą oznaczającą wartość sygnału PWM, który steruje lewym silnikiem.  Wartości R1 i R2 są rozłożoną 16-bitową zmienną całkowitą oznaczającą PWM który, steruje prawym silnikiem. | | | | | | | | | | |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Ustawienie timera aktualizacji prędkości | | | | | USTAW | | | 9 | | |
| „B” | 9 | T1 | T2 | „M” | 0 | 0 | 0 | | 0 | „E” |
| Wiadomość ustawia timer który, co określony interwał czasu aktualizuje prękość robota  Wartości T1 i T2 są rozłożoną 16-bitową zmienną całkowitą oznaczającą ilość milisekund co ile generowane jest przerwanie.  Działanie timera i opis aktualizacji prędkości opisany został w rozdziale 3.6 | | | | | | | | | | |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Pobranie interwału timera akt. prędkości | | | | | POBIERZ | | | 10 | | |
| „B” | 10 | T1 | T2 | „M” | 0 | 0 | 0 | | 0 | „E” |
| Wiadomość zwraca ustawioną wartość milisekund interwału aktualizacji prędkości robota  Wartości T1 i T2 są rozłożoną 16-bitową zmienną całkowitą oznaczającą ilość milisekund interwału czasowego timera | | | | | | | | | | |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Pobranie nastawionej prędkości regulatora | | | | | POBIERZ | | | 11 | | |
| „B” | 11 | L1 | L2 | „M” | R1 | R2 | 0 | | 0 | „E” |
| Wiadomość zwraca prędkość która jest zadana na wejście regulatora.  Wartości L1 i L2 są zserializowaną zmienną zmiennoprzecinkową oznaczającą zadaną prędkość dla lewego koła  Wartości R1 i R2 są zserializowaną zmienną zmiennoprzecinkową oznaczającą zadaną prędkość dla lewego koła | | | | | | | | | | |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Ustawienie timera aktywacji regulatora | | | | | USTAW | | | 12 | | |
| „B” | 12 | T1 | T2 | „M” | 0 | 0 | 0 | | 0 | „E” |
| Wiadomość ustawia timer który, co określony interwał czasu uruchamia regulator.  Wartości T1 i T2 są rozłożoną 16-bitową zmienną całkowitą oznaczającą ilość milisekund co ile generowane jest przerwanie.  Ustawienie wartości T1 i T2 na zero oznacza wyłączenia regulatora i umożliwia nastawianie wartości wypełnienia PWM bezpośrednio na silniki. W przypadku wyłączonego timera, ustawienie wartości większej od zera włącza regulację prędkości.  Działanie timera i opis regulacji prędkości opisany został w rozdziale 3.7 | | | | | | | | | | |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Pobranie interwału Ustawienie tim. aktywacji reg. | | | | | POBIERZ | | | 13 | | |
| „B” | 13 | T1 | T2 | „M” | 0 | 0 | 0 | | 0 | „E” |
| Wiadomość zwraca ustawioną wartość milisekund interwału regulatora prędkości  Wartości T1 i T2 są rozłożoną 16-bitową zmienną całkowitą oznaczającą ilość milisekund interwału czasowego timera | | | | | | | | | | |

# Możliwości wykonanego projektu

# Podsumowanie projektu