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 2](#_Toc435889441)

[1.1 Cel projektu 2](#_Toc435889442)

[1.2 Założenie projektowe 2](#_Toc435889443)

[2. Kontrukcja sprzętowa 2](#_Toc435889444)

[2.1 Platforma mechaniczna 2](#_Toc435889445)

[2.2 Konstrukcja eletroniczna 2](#_Toc435889446)

[2.2.1 Moduł STM32 NUCLEO-F401RE 2](#_Toc435889447)

[2.2.2 Bazowy obwód – schemat 2](#_Toc435889448)

[2.2.3 Bazowy obwód – obwód drukowany 2](#_Toc435889449)

[2.2.4 Moduł Bluetooth – HC-05 2](#_Toc435889450)

[2.3 Podsumowanie 2](#_Toc435889451)

[3. Oprogramowanie robota mobilnego 2](#_Toc435889452)

[3.1 Struktura oprogramowania 2](#_Toc435889453)

[3.2 Konfiguracja mikrokontrolera STM32F401RE 2](#_Toc435889454)

[3.3 Realizacja komunikacji Bluetooth 2](#_Toc435889455)

[3.4 Proces przetwarzania wiadomosci 2](#_Toc435889456)

[3.5 Zabezpieczenia robota 2](#_Toc435889457)

[3.6 Realizacja odczytu predkosci 2](#_Toc435889458)

[3.7 Układ regulacji 2](#_Toc435889459)

[4. Oprogramowanie aplikacji sterującej 2](#_Toc435889460)

[4.1 Struktura oprogramowania 2](#_Toc435889461)

[4.2 Moduł kontroli komunikacji bluetooth 2](#_Toc435889462)

[4.3 Moduł sterowania tekstowego 2](#_Toc435889463)

[4.3.1 Użycie silnika Javascript w aplikacji 2](#_Toc435889464)

[4.4 Moduł sterowania graficznego 2](#_Toc435889465)

[4.4.1 Bazowy blok sterujący 2](#_Toc435889466)

[4.4.2 Rodzaje bloków sterujących 2](#_Toc435889467)

[4.4.3 Synchronizacja działania bloków 2](#_Toc435889468)

[4.5 Okno główne aplikacji i podsumowanie 2](#_Toc435889469)

[5. Protokół komunikacyjny 2](#_Toc435889470)

[5.1 Konwersje liczb całkowitych i rzeczywistych na ciąg bitów 2](#_Toc435889471)

[5.2 Spis wiadomości kontrolnych 2](#_Toc435889472)

[6. Możliwości wykanego projektu 2](#_Toc435889473)

[7. Podsumowanie projektu 2](#_Toc435889474)

# Wprowadzenie

Robot mobilny klasy 2,0 jest robotem posiadającym dwukołowy napęd różnicowy.

Przemiaszczenia robota umożliwia sterowanie prędkością obrotu koła przymocowanego do danego napędu bez możliwości zmiany kieruku ułożenia koła w stosunku do platformy robota.

## Cel projektu

Celem projetu było stworznie fizycznego robota mobilnego 2.0 wraz ze stworzeniem oprogramowania na miktrokontroler na nim umieszczony, jak również aplikacji, która umożliwia zdalne sterowania robotem. Komunikacja pomiędzy platformą mobilną a programem sterujący opaty jest na standardzie bluetooth. Celem było umożliwianie w miare możłiwości jak niajwiększej interakcji z robotem w zrożnicowany sposób.

## Założenie projektowe

Projekt zakłada przygotawanie modelu mechanicznego zawierającego dwukołowy napęd różnicowy wraz 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 robota ( mikrokontrolera osadzonego na platformie ) pokrywa zakres:

* Komunikacja dwukierunkowa intefejsem szeregowym
* Asynchroniczna rekacja na otrzymane wiadomości z zewnątrz
* Obsługa sterowania synagłu PWM ( wymagane przy sterowaniu silnikiem )
* Obsługa 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 komukacyjnego dający aplikacji intefejs na którym może operować. Protoła zakłada wiadomości typu pobierz/zapisz które pozwalają za pobranie z robota danej wartości i nadpisanie jej nową wartością. Aplikacja sterująca, zgodnie z celem projektu, ma umożliwiac szeki 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 pomacą komend tekstowych. Punkt ten zakłada przygotowanie modułu interpretera do wprowadzania komend/skryptów oraz wykożystanie silnika języka javascript
* Możliwość sterowania w trybie graficznym za pomocą myszki, wizualizacja prędkości oraz konfiguracja robota

# Kontrukcja sprzętowa

## Platforma mechaniczna

## Konstrukcja eletroniczna

## Moduł STM32 NUCLEO-F401RE

## Bazowy obwód – schemat

## Bazowy obwód – obwód drukowany

## Moduł Bluetooth – HC-06

## Podsumowanie

# Oprogramowanie robota mobilnego

## Struktura oprogramowania

Oprogramowanie robota wymagało obsługi sterowania silnikami

## Konfiguracja mikrokontrolera STM32F401RE

## Realizacja komunikacji Bluetooth

Komunikacja Bluetooth wykożystuje obsługe dwóch komponentów mikrokontrolera, interfejsu szeroegoweo 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 wyprowadzenia linii danych TX (wysyłanie) mikrokontrolera z linią danych RX (odbiór) układu HC-06 oraz analogicznie linii RX mikrokontrolera z linią TX modułu.Transmisje bezprzewodowa jest przeroczysta z punktu widzenia mikrokontrolra. Interfejs UART konfigurowany jest przed podanie 4 wartości określającyhc transmisję:

* Pręskość transmisji (ang. Baud rate)
* Długość slowa
* 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 ) gdyż całkowicie wystarczała moim wymaganiom.

DMA jest modułem który umożliwia bezpośreni dostęp do pamięci RAM i układów perfyferyjnych. Układ DMA umożliwia teransfer danych pomiędzy urządeniem peryferyjnym a pamięcią RAM bez angażowania procesora, którego zaganie 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.

Asychroniczność transmisji możliwa jest poprzez obsuge przerwania które generuje UART po zakończeniu transmisji. Biblioteki HAL zapeniają niskopoziomową obsługe przerwań oraz deklaracje zestawu funkcji zwrotnych które zostają wykone w odpowiedzi na zarejestronawe przerwanie. Obsługa zakończenia transmisji:

|  |  |  |
| --- | --- | --- |
|  | void HAL\_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 biblioteke HAL wywoływana dla zarejstrowanego przerwania od dowolnego interfejsu UART. Obiekt UART komponentu który wywołał przerwanie jest przekazywany jako argument funkcji. Z powodu, że używany jest jeden interfejs szeregowy ni ma potrzeby sprawdzanie czy to on jest źródłem przerwania. Funkcja tworzy tablicę bajtów - response, która wraz z nowowypełnioną tablicą-command przekazywana jest do funkcji obsługującej daną wiadomość. Wypełniona odpowiedź wysyłana jest tą samą techniką. Obsługa wiadomości oraz zastosowant timer opisany jest w dalszej części tego rozdziału.

## Proces przetwarzania wiadomosci

W poprzednim podrozdziale zaprezentowana była obsługa przerwania od zakończenia transmisji szeregowej, obecna jest funkcja „commandHandler” odpowiedzialan za obsługe wiadomości i uformowanie odpowiedzi. Struktura funkcji „commandHandler” ( dle czytelności pominąłem fragment funcji, nie wpływa to na zrozumienie jej działąnia):

|  |  |  |
| --- | --- | --- |
|  | int8\_t\* commandHandler(int8\_t\* command,int8\_t\* response)  {  if(command[0] != ‘B’ || command[4] != ‘M’ || command[9] != ‘E’)  return invalidMessage(response);  if(command[1] == 1)  {  //set speed  return setVelocityResponse(command,response);  }  if(command[1] == 2)  {  //get speed  return getVelocityResponse(command,response);  }  //(…) pominęta część funkcji  if(command[1] == 14)  {  //set direction pins  return setMotorsDirection(command,response);  }  if(command[1] == 15)  {  //get direction pins  return getMotorsDirection(command,response);  }  return invalidMessage(response);  } |  |

Działanie tej funkcji jest ściśle związane z protokołem komunikacyjnym pomiędzy robotem a aplikacją sterującą. Zagadnienie to jest szczegółowo opisan w rozdziale piątym. Początek funkcji sprawdza czy na odpowiednich pozycjach występują stałe wartości, w przeciwnym wypadku wysyłana jest wiadomość informująca o błędzie. Następnie na podstawie indentyfikatora wiadomości wywoływana jest kontetna funkcja obsługująca daną wiadmość, 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\_t index = 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 komnda przychodząca. Dzięki temu aplikacja sterująca może mieć pewność, że ustawione zostały podane wartości. Pobór fatycznych wartości ustawionych na robocie odbywa sięza pomocą wiadomości typu „pobierz”. Zaprezentowna wiadomość ustawia 3 pozycje: prędkość lewego koła, prędkość prawego koła (w cm/s ) oraz opcjonalnie czas trwania zadanego ruchu. Wadomości ustawiające prędkość charateryzują się jeszcze właściwością. Mają wypełnione wszystkie pola wartości wypełnione zerami wywołuje się funkcja zatrzymania silników robota. Jest to jedna z formy zabezpieczenia. Warunek ten jest sprawdzany od razu po uformowaniu wiadomości. Kolejnym etapem jest konwersja odpowiednich bajtów na liczbę zmiennoprzecinkową i zapis wartości do globalnej zmiennej określającej zadaną prędkość dla danego koła. Konwersja wartości omóiona jest w rozdziale piątym. Ostatnim etapem obsługi wiadomości jest ustawienie czasu trwanai długości danego ruchu. Jeżeli pola nie są wypełnione zerami wystartuje timer z przesłanądaną danąilością milisekund, który po tym czasie zatrzyma silnika. W przeciwnym razie zadana prędkość będzie się utrzymywałą az do jej zmiany. Po zakończeniu procedury zwracany jest adres z odpowiedzią.

## Zabezpieczenia robota

Podstawową funkcją wykorzystywaną we każdej formie zabezbieczeń robota jest „stopMotors”:

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

Funkcja wyzrowuje 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 podrozdziale wysłanie wiadomisci zadającej prędkość z wartościami równymi zero. Kolejnym mechanizem jest wysłanie niepoprawnej wiadomości. Wywołanie funkcji „invalidMessage” powoduje zatrzymanie pracy silnikow. Ostatnim zabezpieczenim jest ustawienie timer bluetooth ( widoczy jest we fragmencie kodu w podrozdziale 3.3 ). Timer zaczyna odliczanie po otrzymaniu wiadomosci i gdy nastena wiadomosc nie pojawi sięp przed zakończeniem odliczania silaniki są zatrzymywanie. Jest to form zabezpieczenia przed utratą łączności. Domyślnie timer ten jest wyłaczony. Włączenie oraz ustalenie czasu odliczania jest możliwe za pomodzą wiadomości kontrolnej.

## Realizacja odczytu predkosci

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 pomacą enkoderów. Enkodery zamontowane na wale silników podłączone są to pinów skonfigurowanych jako wejścia kanałów timerów obsługujących ten czujniki. Mikrokontroler STM umożliwia dla skofigurowanie timera do trybu Encoder Mode. Oznacza to że sygnałem sygnały z czujnika zmienią stan licznika, nie sygnal taktujący. Timery użyte do obsługi enkoderów(htim1 i htim3) skonfugurowane zostały do tryb zliczanai w dół, gdzie maksymalna wartość to 65000. Do wartości rejestru, którega jest zmieniana przez impulsy enkodera można się dostąc za pomocą obiektu timera: htim1/htim3.Instance->CNT. Wartość ta przyjmuje punkt zerowy 32500(stałą = „ENCODER\_INITIAL”) czy w połowie wartości przepełnienia. Do obliczania predkości użyty został jeszcze jeden niezależny timer, który o czas ustalany aktualizuje globalne zmienne zawirajacy aktualna wartość prędkości kół robota. Obsługa timera:

|  |  |  |
| --- | --- | --- |
|  | void HAL\_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 == &htim5)  {  stopMotors();  HAL\_TIM\_Base\_Stop\_IT(&htim5);  }  else if(htim == &htim9)  {  stopMotors();  HAL\_TIM\_Base\_Stop\_IT(&htim9);  }  else if(htim == &htim10)  {  regulation();  HAL\_TIM\_Base\_Start\_IT(&htim10);  }  else if(htim == &htim3)  {  onEncoderOverload(&htim3,&leftTotalTicks);  }  else if(htim == &htim1)  {  onEncoderOverload(&htim1,&rightTotalTicks);  }  } |  |

Jest to uniwerslany timer dla wszystkich timerów zarejestrowanych na przerwanie od końca odliczania. Biblioteka HAL obsługuje niskopoziomwo przerwanie timera i wywołuje funkcje obsługi przekazują w argumencie adres obiektu kontrolującego dany timer. Timer (htim2) odpowiedzialny za aktualizację prędkości jest obsługiwanie na początku zapisuje wyniki funkcji obliczających prędkość oraz zapisuje je zmiennych globalnych, po restartuje się. Interwal czasowy dla tego timera może być ustalony przez wiadomość kontrolną. Wyjaśnienie wymaga również obsługa przerwania wywołanego przez timery htim1 i htim3. Wywołuje ono funkcję „onEncoderOverload”. Z powodu, że interwał aktualizacji prędkości może być ustalany dowolną wartością oraz maksymalna prędkość jaką mogą uzyskać koła nie jest stała ( zależy od użytych silników, poziomu naładowanai baterii, użytej przekładni i innych czynników ) wymagane było zabezpieczenie przed przekroczeniem maksymalnej wartości timera zanim nastąpi aktualizacja prędkości. Wyżej wymieniona funkcja zapewnia obsługę takich sytuacji:

|  |  |  |
| --- | --- | --- |
|  | 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 ostatnij aktualizacji prędkości. Zmienna „totalTickCounter” jest ze znakiem co umożliwia określenie zwrotu prędkości. Prękość zgodna z przyjętą przedniącześcią robota przekracza licznik zmniejszając wartość rejesu 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 lu dekrementowana o obliczoną warość impulsów.

Oblicznie prędkości podczas aktualizacji odbywa się za pomocą funkcji „LeftVelocity” i „RightVelocity”. Obliczanie prędkości kołą na przykładzie funkcji „LeftVelocity”:

|  |  |  |  |
| --- | --- | --- | --- |
|  | |  | | --- | | float LeftVelocity(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 wygnerowancych przez enkoder od ostatnie aktualizacji prędkości. Następnie zeruje ilość impulsów, oblicza prękość postępową w cm/s i zwraca ją. Funkcja obliczająca prędkośC postępową:

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

W pierwszym kroku obicza ilość im 🡨 ogarnąć

## Układ regulacji

Układ regulacji opiera się na regulatorze PD.

# Oprogramowanie aplikacji sterującej

## Struktura oprogramowania

## Moduł kontroli komunikacji bluetooth

## Moduł sterowania tekstowego

## Użycie silnika Javascript w aplikacji

## Moduł sterowania graficznego

## Bazowy blok sterujący

## Rodzaje bloków sterujących

## Synchronizacja działania bloków

## Okno główne aplikacji i podsumowanie

# Protokół komunikacyjny

## Konwersje liczb całkowitych i rzeczywistych na ciąg bitów

## Spis wiadomości kontrolnych

# Możliwości wykanego projektu

# Podsumowanie projektu