AKADEMIA GÓRNICZO-HUTNICZA IM. STANISŁAWA STASZICA W KRAKOWIE
Wydział Elektrotechniki, Automatyki, Informatyki i Inżynierii Biomedycznej

Raport z wykonania projektu

Robot minisumo

Autorzy: *Mateusz Jurczak*
Paulina Marchut
Kierunek studiów: Mikroelektronika w Technice i Medycynie

Kraków, 2023

1. Cel Pracy

Celem projektu było zbudowanie robota minisumo, napisanie oprogramowanie z wykorzystaniem systemu FreeRTOS oraz refaktor kodu komponentu CommManager.

1.1. Założenia pracy

Głównym założeniem pracy jest wykonanie i zaprojektowanie robota minisumo w oparciu o system FreeRTOS, zdolnego do autonomicznej walki. Oprogramowanie ma pozwalać na obsługę wielu konfiguracji w łatwy sposób oraz możliwości budowania oprogramowania dla różnych robotów wykorzystujących jedną płytkę.

2. Struktura oprogramowania

2.1. Opis działania

Program aktualnie posiada 3 wątki systemu FreeRTOS:

- Obsługa czujników ToF
- Obsługa i diagnostyka silników
- Obsługa algorytmu sterowania

W dalszej części opisane zostaną poszczególne wątki systemu.

2.1.1 Czujniki ToF

Nasz robot wykorzystuje czujniki ToF VL53L5CX, niestety dokumentacja do nich jest wyjątkowo uboga, ponieważ zdaniem producenta pełna specyfikacja byłaby „zbyt skomplikowana". W zamian za dokumentację dostarczone jest blokujące „Ultra Light API". Za każdym włączeniem czujnika potrzeba wgrać do niego oprogramowanie układowe, ponieważ przechowywane jest ono w pamięci RAM. Proces wgrywania tego oprogramowania posiada wiele momentów wymuszających oczekiwanie na odpowiedź, co powoduje blokowanie programu i jest nieefektywne. Dzięki zastosowanie systemu operacyjnego udało nam się ten problem częściowo obejść.

Działanie klasy czujników ToFSensors można podzielić na następujące kroki:

- Rejestracja czujników (Stworzenie instancji klasy VL53LC5X)
- Inicjalizacja czujników
- Diagnostyka i obsługa błędów

Po uruchomieniu i inicjalizacji dane z czujników powinny być odbierane z wykorzystaniem klasy CommManager i przerwań sprzętowych, jednakże ze względu na usterki powstałe podczas walki robota, bądź z przyczyn losowych, nie zawsze jesteśmy w stanie odebrać przerwanie (Np. ze względu na uszkodzone połączenie). Jeżeli sensor nie wygeneruje przerwania przez pewien czas, to zacznie być odpytywany w trybie pollingu. Jeżeli w dalszym ciągu nie nastąpi odpowiedź to obiekt czujnika przejdzie w stan failure, co zostanie zakomunikowane algorytmowi sterującemu.

Aby obejść blokującą naturę API inicjalizacja wygląda w następujący sposób:

1. W czasie konstrukcji obiektów ich wskaźniki są zbierane w klasie bazowej dla VL53L5CX czyli ToFSensor
2. Po wywołaniu statycznej metody ToFSensor::StartSensorTask() wyłączone zostają interfejsy komunikacyjne dla każdego sensora
3. Po kolei są one włączane i ustawiane są im adresy na magistrali
4. Uruchomione zostaje kilka wątków (w zależności od zdefiniowanej liczby), które pobierają wskaźniki do obiektów VL53L5CX i rozpoczynają inicjalizację dla każdego z nich.
5. Po zakończeniu inicjalizacji wszystkich czujników wątki i używane semafory zostają usunięte z systemu.

Pozwoliło nam to na zmniejszenie czasu inicjalizacji dla wszystkich czujników z około 6s do 2.1s.

2.1.2 Sterowniki silników

Do obsługi silników wykorzystujemy sterowniki L9960T od firmy ST. Pozwalają one na diagnostykę i zmianę ustawień poprzez SPI. Tak jak w przypadku czujników ToF działanie wątku zajmującego się obsługą tych sterowników możemy podzielić na 3 etapy:

- Rejestrację sterowników (Stworzenie instancji klasy L9960T)
- Inicjalizację sterowników
- Diagnostykę i obsługę błędów

Po uruchomieniu programu następuje inicjalizacja sterowników, w pierwszym etapie wykorzystywana jest do tego klasa CommManager która zapewnia wywołanie funkcji callback każdego obiektu klasy L9960T po wysłaniu przez niego wiadomości. Po zakończeniu tej części następuje uruchomienie wątku zajmującego się diagnostyką i obsługą błędów które pojawiają się w czasie działania programu/walki robota. Co 5ms odczytywane są rejestry diagnostyczne, a ich zawartość zapisywana do zmiennej obiektu. Dzięki temu możemy w programie stwierdzić jakie występują problemy. Każdy odczyt danych diagnostycznych powoduje też automatyczne wyczyszczenie błędów. Dzieje się tak ponieważ w czasie inicjalizacji sterownika ustawiamy tą opcję, wiedząc że jest ona niebezpieczna i może spowodować uszkodzenie silników.

2.1.3 Algorytm sterowania

Algorytm w przeciwieństwie do poprzednich elementów składa się z dwóch etapów:

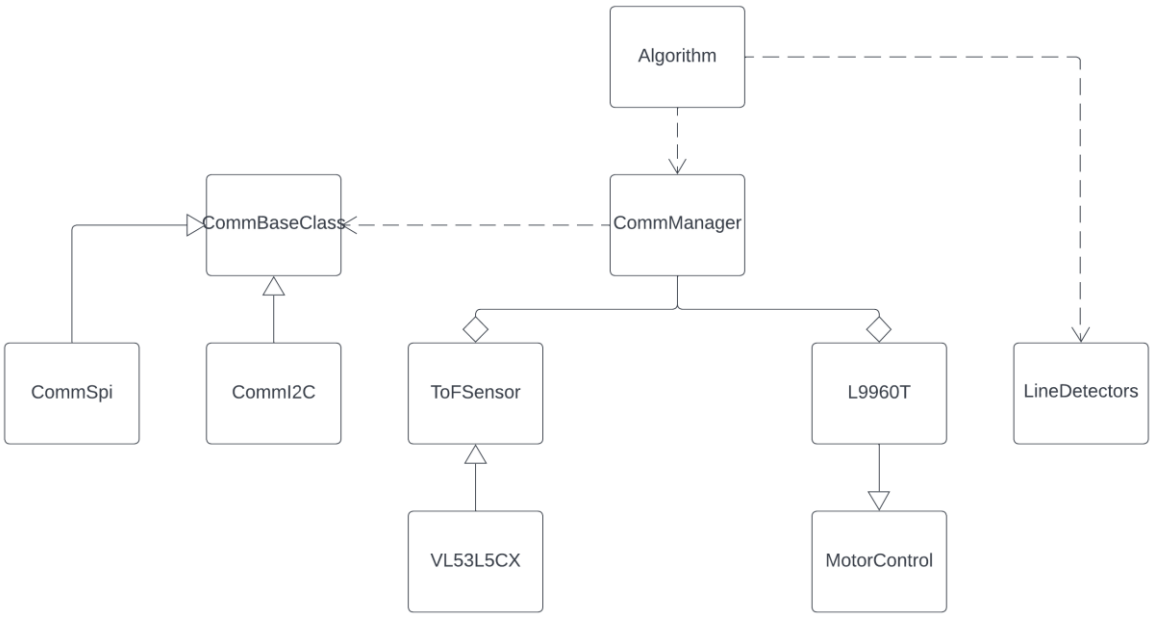
- Inicjalizacja
- Algorytm główny

W pierwszej części robot czeka na inicjalizację czujników ToF, czujników linii, oraz sterowników silników. Po jej zakończeniu silniki uruchamiają się z pełną mocą do przodu na 0,5sek w celu sygnalizacji gotowości. W kolejnej części pętla sterowania wywoływana jest co jeden tick systemowy.

2.2. Diagramy

2.2.1 Diagram systemowy

Poniższy diagram klas przedstawia przykładową konfigurację robota minisumo, wykorzystującego czujniki odległości ToF VL53L5CX z którymi komunikacja odbywa się po magistrali I2C, analogowe czujniki linii oraz podwójny sterownik silników L9960T sterowany poprzez magistralę SPI.



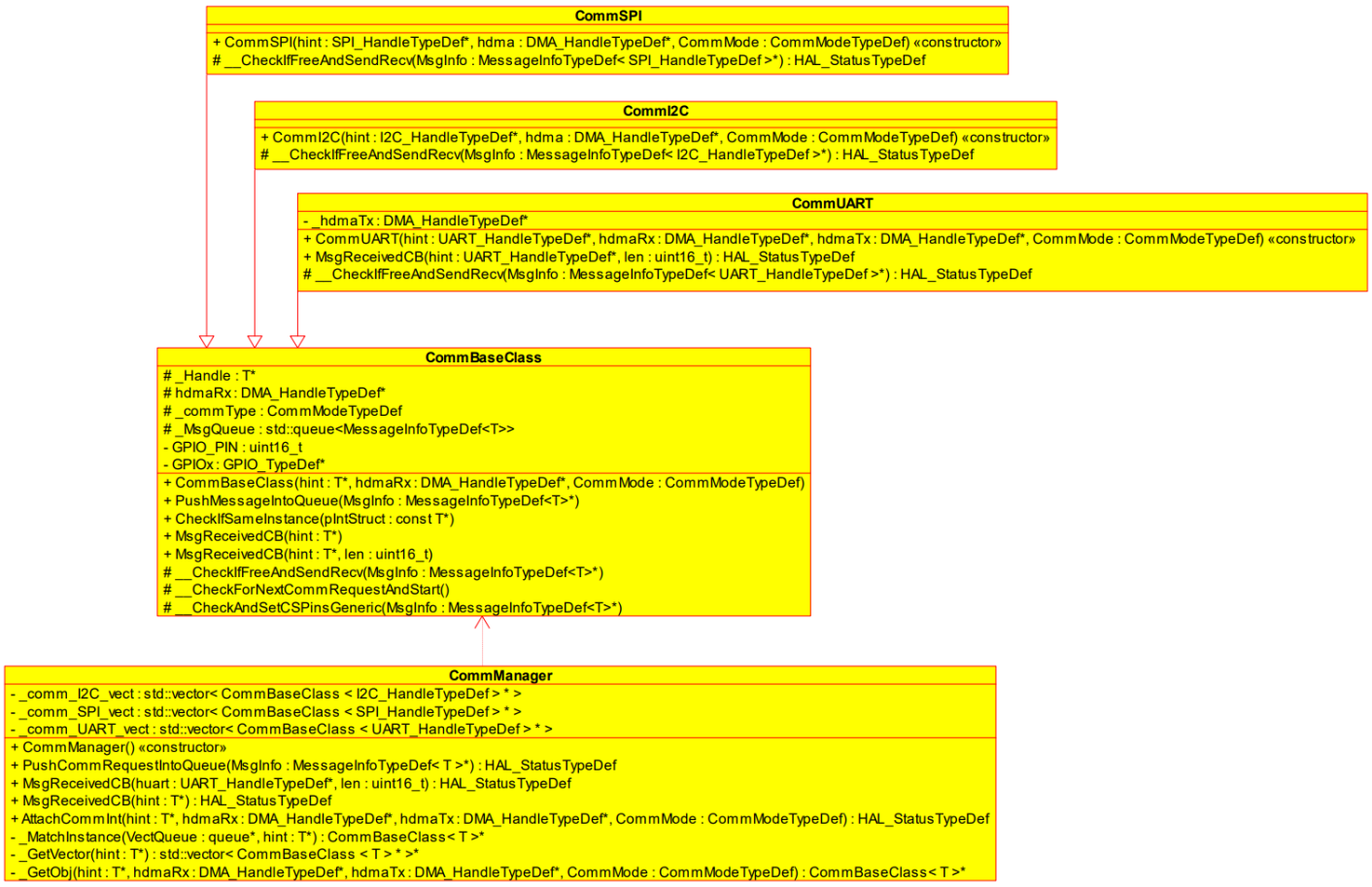
2.2.2 Diagram klas CommManager-a

Głównym zadaniem CommManager-a jest ujednolicenie i ułatwienie komunikacji mikrokontrolera z urządzeniami zewnętrznymi. Aby uzyskać jak największą szybkość działania oraz nadać uniwersalny charakter tego komponentu, CommManager został zaprojektowany tak, aby nie używał systemu operacyjnego a tylko niskopoziomowych funkcji. Z założenia, w programie ma działać jedna instancja CommManager-a zarządzająca całą komunikacją dostępnych na mikrokontrolerze magistral. Obiekt ten posiada w swoich atrybutach wektory handlerów dla interfejsów SPI, I2C i UART.

W poprzedniej wersji CommManager był pojedynczą klasą i zawierał w sobie kod niskopoziomowy obsługujący magistrale. Z tego względu jego kod był mocno nieczytelny i zawiły (zawierał dużą ilość ifdef-ów), pozwalał na tylko jeden sposób komunikacji na magistralę (z wykorzystaniem przerwań, DMA lub pollingu). Dodatkowo wykorzystywał przeładowania, co dodatkowo zmniejszało czytelność i powodowało dużą powtarzalność kodu.

Nowa wersja wykorzystuje bardziej obiektowe podejście do programowania. Wszelkie niskopoziomowe funkcje zostały oddelegowane do klas dziedziczących po klasie CommBase, w której znajduje się cały wspólny kod dla magistral. Wykorzystanie templat-ów umożliwiło przerzucenie dużej części kodu do CommBase. Klasy poniżej realizują już tylko funkcje specyficzne dla danego interfejsu. Klasa CommManager na ten moment zajmuje się przechowywaniem informacji o dołączonych do niego interfejsach oraz wybieraniu poprawnej instancji obiektu CommInterface do obsługi callbacku. Proces kolejkowania i wysyłania wiadomości poprzez CommManagera sprowadza się do stworzenia zmiennej typu MessageInfoTypeDef<Interface> (np. SPI, I2C, UART) oraz jej uzupełnieniu. CommManager pozwala na ustawienie pinu ChipSelect funkcji callback, konkretnego interfejsu oraz kontekstu,

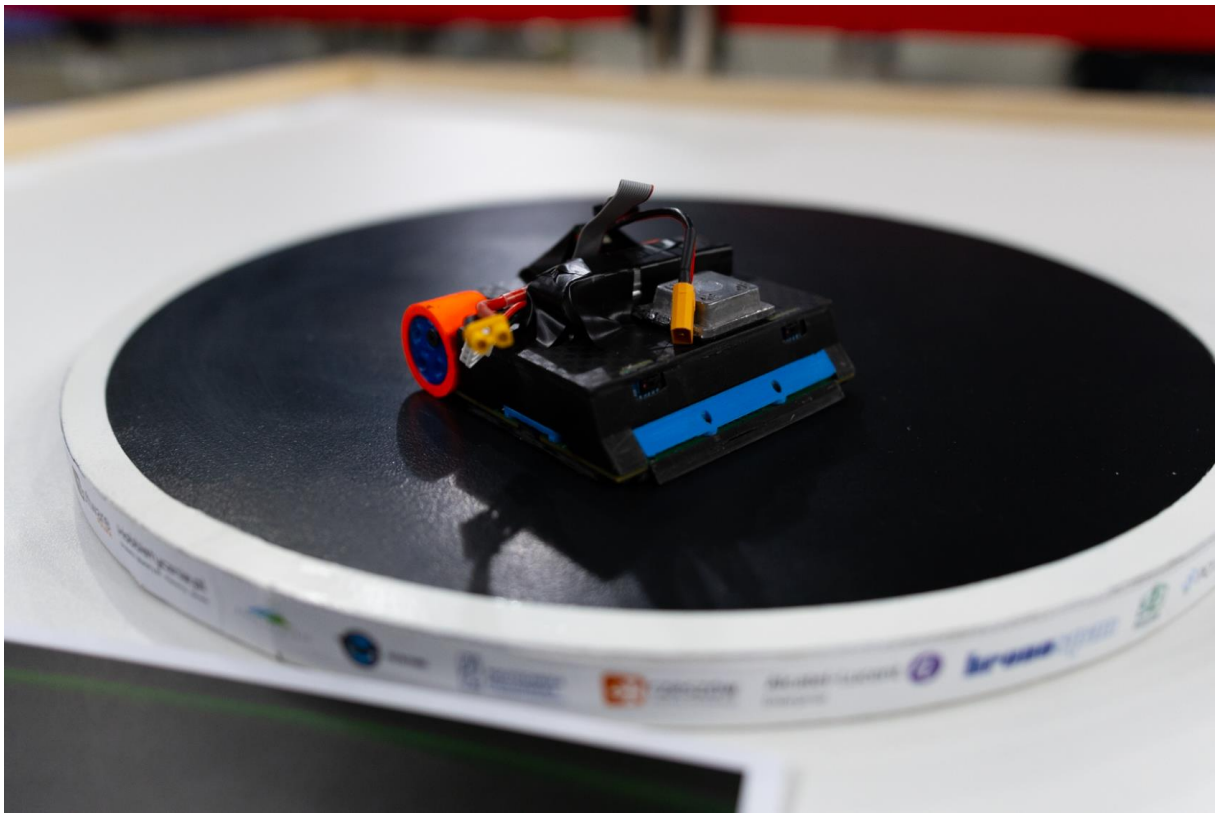
szesnastobitowej wartości która zostanie zwrócona przy zakończeniu transakcji wraz z całą zmienną typu MessageInfoTypeDef, co pozwala na identyfikację wiadomości.



3. Podsumowanie

W aktualnym stanie robot spełnia wszystkie założenia projektowe tzn. jeździ, jest w stanie znaleźć przeciwnika, wszystkie funkcje działają, a nawet udało mu się wygrać dwie walki na zawodach robotycznych w Rzeszowie (wtedy jeszcze w trakcie tworzenia kodu i przed refaktoringiem CommManagera). Aktualnie największej poprawy wymaga mechanika. W planach jest jeszcze udoskonalenie algorytmu sterującego.

Repozytorium: https://github.com/MJurczak-PMarchut/MiniSumo



Ilustracja 3-1 Nasz robot na zawodach, Rzeszów, Listopad 2022