Marek DŁUGOSZ[1], Michał ROMAN[1], Paweł WĘGRZYN[1]

AGH Akademia Górniczo-Hutnicza (1)

doi:10.15199/48.2021.08.08

System sterowania autonomicznego pojazdu A-EVE

Streszczenie. *W artykule opisano koncepcję układu sterowania prototypowego autonomicznego pojazdu elektrycznego A-EVE. Opracowano koncepcję dwupoziomowego układu sterowania który składa się z układu sterowania niskopoziomowego i wysokopoziomowego. W pracy opisano kluczowe elementy autonomicznego pojazdu A-EVE takie jak sensory, układ sterowania, układy bezpieczeństwa. Przeanalizowano wpływ flagi kompilacji programu sterownika niskiego poziomu na czas wykonywania programu sterowania.*

Abstract. *The paper presented the concept of the control system of the prototype electric autonomous vehicle A-EVE. The proposed control system is a two-level of control system which contain a low level and a high level control system. The paper describes the key elements of the autonomous A-EVE vehicle, such as sensors, control system, and safety systems. Additionally, the impact of the program compilation flag for the low-level control program execution time was analyzed.**(Control system of the autonomous vehicle A-EVE)***

Słowa kluczowe: autonomia, autonomiczne pojazdy, system sterowania, elektryczny pojazd
Keywords: autonomous, autonomous vehicle, control system, electrical vehicle

Wstęp

Autonomiczne pojazdy stają się coraz bardziej popularne i coraz częściej spotykamy je w życiu codziennym. Trwające od lat 90-tych intensywne badania naukowe prowadzone w tym obszarze a także ogromny postęp technologiczny sprawia iż pojazdy autonomiczne nie są już tylko konstrukcjami laboratoryjnymi, ale są oferowane jako produkty na rynku [1, 2]. Sercem każdej konstrukcji autonomicznego pojazdu (czy też robota autonomicznego) jest jego układ sterowania. Układy sterowania w pojazdach (lub robotach) autonomicznych są systemami bardzo złożonymi. Pojazd autonomiczny oprócz realizacji typowych zadań sterowania jak np. stabilizacja zadanej prędkości musi podejmować szereg decyzji związanych z samą jazdą i zachowaniem się pojazdu w ruchu drogowym (np. sposób zachowania się na skrzyżowaniu, podjęcie decyzji o zmianie pasa ruchu itp.) [3]. Analizując istniejące architektury układów sterowania prototypowych pojazdów autonomicznych można zauważyć iż, w większości ich układ sterowania podzielony jest na dwa podsystemy: sterowanie nadrzędne i sterowanie podrzędne, zob. np [4, 5, 6, 7, 8]. Tego typu architektury sterowania zastosowano między innymi w pojazdach autonomicznych biorących udział w zawodach DARP-a [9] takich jak: Stanley Stanford [10], BOSS [11], TALOS-Mit [12]. Wielopoziomowa architektura układu sterowania umożliwia efektywne implementowanie złożonych układów sterowania oraz ich łatwą modyfikację i testowanie. Możliwość dekompozycji układu sterowania na oddzielne podsystemy umożliwia niezależne weryfikowanie zastosowanych algorytmów sterowania [13] lub testowania nowych typów regulatorów zob. np. [14, 15].

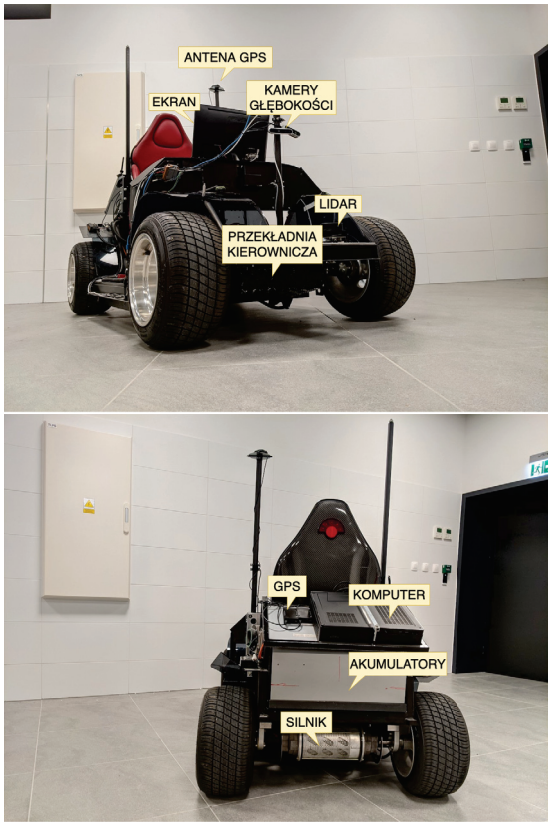
W artykule opisano projekt i sposób zrealizowania dwupoziomowego układu sterowania prototypowego pojazdu autonomicznego o nazwie A-EVE. Założeniem projektu było zaprojektowanie i wykonanie w pełni funkcjonalnego systemu sterowania pojazdem autonomicznym który umożliwi dalszy jego rozwój oraz testy różnych aspektów związanych z sterowaniem i kontrolą pojazdów autonomicznych. Istotną częścią opisywanego układu sterowania jest sterownik podrzędny który został od początku zaprojektowany i zrealizowany w ramach prowadzonych prac badawczych.

Opis prototypowego pojazdu A-EVE

Pojazd EVE (skrót ***E****lectrical* ***VE****hicle*) jest niewielkim, prototypowym elektrycznym pojazdem skonstruowanym przez firmę Aptiv (wcześniej Delphi). Na podstawie umowy o współpracy pomiędzy AGH w Krakowie a firmą Aptiv, został on przekazany zespołowi badawczemu w celu zaprojektowania i zbudowania prototypowego pojazdu autonomicznego. W celu podkreślenia faktu, że prototyp będzie pojazdem autonomicznym akronim EVE został zmieniony na A-EVE (***A****utonomous* ***E****lectrical* ***VE****hicle*).

A-EVE to jednoosobowy pojazd napędzany trójfazowym silnikiem elektrycznym o mocy 750W który jest zamontowany poprzecznie na tylnej osi pojazdu [16]. Zasilany jest z trzech szeregowo połączonych standardowych akumulatorów samochodowych. Skręcanie przednich kół realizowane jest w systemie *drive by wire* tj. odczyty zmian obrotu kierownicy powodują obrót silnika z przekładnią liniową która to przekładnia, skręca przednie koła o kąt proporcjonalny do kąta obrotu kierownicy (kinematyka ackermana). W układzie sterowania kołem kierownicy zaimplementowany jest algorytm *forece feedback* dający wrażenie rzeczywistego układu skręcania oraz dzięki któremu położenie kierownicy oraz kół przednich, wraca do położenia zerowego. Pojazd A-EVE wyposażony jest w dwa niezależne układy hamulcowe. Pierwszy z nich to układ elektryczny (silnik elektryczny zaciska szczęki hamulcowe na tylniej osi). Drugi z nich to układ hydrauliczny który hamuje tylko przednie koła. Obydwa układy są sterowanie za pomocą pedału hamulca z tą różnicą, że sygnał sterowania dla elektrycznego układu hamowania może być generowany niezależnie przez algorytmy sterowania (jest to sygnał typu PWM). Masa pojazdu to ok. 350 kg, maksymalna prędkość 20 km/h (ograniczona elektronicznie ze względu na brak amortyzacji zawieszenia). Na rysunku 1 przedstawiono zdjęcie pojazdu A-EVE wraz z zaznaczonymi kluczowymi elementami.

Pojazd A-EVE wyposażony jest w różnego typu sensory za pomocą których postrzega i interpretuje otocznie w którym się znajduje. Na podstawie informacji uzyskanych z sensorów, autonomiczne pojazdy wyznaczają swoją pozycje geograficzną na mapie, planują trasy, realizują jazdę po zaplanowanej trasie czy też unikają różnych niebezpiecznych sytuacji. Pojazd A-EVE wyposażony jest w następujący zestaw sensorów: dwa enkodery do pomiaru prędkości obrotowej kół, dwa odbiorniki GPS do ustalania pozycji geograficznej pojazdu, dwie kamery głębokości do wykrywania potencjalnych przeszkód na trasie pojazdu, trzy sensory IMU do wykrywania przyśpieszeń liniowych i kątowych, oraz opcjonalnie lidar. W zależności od zwracanych danych sensory pojazdu A-EVE podłączone są do ECU1 lub ECU2, zob. rys. 2.

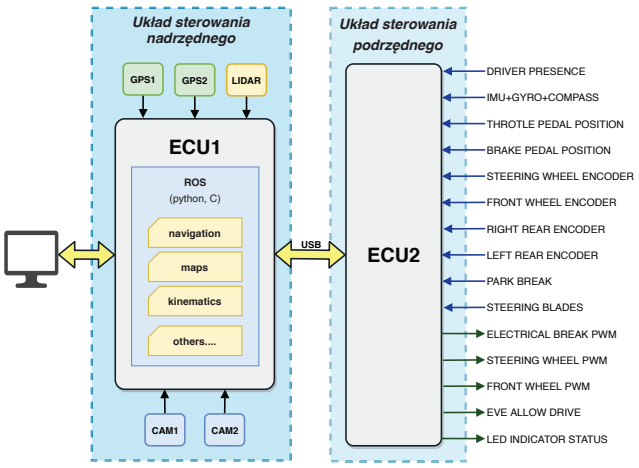


Rys. 1. Zmodernizowany pojazd A-EVE z zaznaczonymi kluczowymi podzespołami.

Układ sterowania

W pojeździe A-EVE zastosowano dwupoziomowy układ sterowania który zawiera sterownik nadrzędny oraz sterownik podrzędny. Strukturę układu sterowania wraz z sygnałami wejściowymi, wyjściowymi oraz sensorami zaprezentowano na rysunku 2. Taka struktura układu sterowania pozwala na optymalny podział zadań realizowanych przez poszczególne sterowniki (sterownik nadrzędny ECU1 i sterownik podrzędny ECU2), a także zapewnia dużą elastyczność w przypadku konieczności zmian w całym układzie sterowania [17].

Sterownik nadrzędny ECU1 komunikuje się z sterownikiem podrzędnym ECU2 poprzez magistralę USB. Do sterownika nadrzędnego ECU1 podłączony jest również monitor LCD z ekranem dotykowym który służy do prezentowania aktualnego stanu pojazdu A-EVE oraz łatwego zarządzania funkcjami pojazdu.



Rys. 2. Architektura układu sterowania pojazdu A-EVE, [16].

Sterowanie podrzędne - sterownik ECU2

Algorytmy sterowania podrzędnego realizowane są przez dedykowany mikrokomputer wraz z odpowiednimi urządzeniami peryferyjnymi, które zostały zaprojektowane i wykonane w ramach projektu inżynierskiego [18]. W sterowniku ECU2 wykorzystano procesor STM32 NUCLEOF767 ARM Cortex-M7 taktowany zegarem o częstotliwości 216 MHz [19]. Sterownik ECU2 realizuje podstawowe algorytmy sterowania jak np. regulator prędkości liniowej, regulator kąta skręcenia kół przednich, regulator obrotu kierownicy (system *drive-by-wire* oraz *force-feedback*). Sterownik ECU2 odpowiedzialny jest również z prawidłowy odczyt i przeliczanie danych z urządzeń takich jak: enkodery kół tylnich, enkoder kąta kierownicy, enkoder do pomiaru kąta skręcenia kół przednich, odczyt pozycji pedałów przyśpieszenia i hamowania, odczyt danych z czujnika obecności kierowcy, odczyt danych z sensorów IMU. Do zadań sterownika ECU2 należy również generowanie poprawnych sygnałów sterowania typu PWM (np. do silnika głównego lub silnika kierownicy) oraz wysyłanie odczytanych danych do sterownika nadrzędnego ECU1.

Sterownik ECU2 może pracować w jednym z czterech stanów pracy: sterowanie ręczne (stan *ManualMode*), sterowanie automatyczne za pomocą algorytmów jazdy autonomicznej (stan *AutomaticMode*), sterowanie za pomocą mainpulatora zewnętrznego (stan *PadSteering*), postój pojazdu A-EVE (stan *StopMode*). Warunki wyjścia z danego stanu sprawdzane są zgodnie z numeracją i jeśli dany warunek nie zostaje spełniony, sprawdzany jest kolejny, natomiast gdy zostanie spełniony kolejne warunki nie są już sprawdzane. Podczas uruchomienia aktywowany jest stan *ManualMode* odpowiedzialny za sterowanie manualne.

Warunki wyjścia ze stanu *ManualMode*:

1. *AutomaticMode* – przejście do stanu jazdy autonomicznej następuje gdy zarówno pedał przepustnicy jak i hamulec nie są naciśnięte oraz ECU1 wysyła sterowanie z odpowiednią częstotliwością.
2. *StopMode* – stan zatrzymania pojazdu zostaje wywołany gdy wystąpi żądanie zatrzymania wysłane przez ECU1 lub oprogramowanie wykryje brak kierowcy.
3. *PadSteering* – tryb jazdy za pomocą pada aktywowany jest na żądanie ECU1 pod warunkiem wysyłania sterowania z odpowiednią częstotliwością.

Warunki wyjścia ze stanu *AutomaticMode*

1. *ManualMode* – przejście do stanu sterowania manualnego następuje gdy któryś z pedałów zostanie naciśnięty lub ECU1 wyśle żądanie zmiany stanu lub sterowanie z ECU1 przestanie być wysyłane z określoną częstotliwością.
2. *StopMode* – przejście do stanu zatrzymania analogicznie jak w stanie *ManualMode*.
3. *PadSteering* – przejście do stanu sterowania padem analogicznie jak w stanie *ManualMode*.

Warunki wyjścia ze stanu *PadSteering*

1. *AutomaticMode* – przejście do trybu jazdy autonomicznej następuje gdy ECU1 zażąda zmiany stanu pod warunkiem wysyłania sterowania z odpowiednią częstotliwością.
2. *ManualMode* – przejście do stanu sterowania manualnego analogicznie jak w stanie *AutomaticMode*.
3. *StopMode* – przejście do stanu zatrzymania analogicznie jak w poprzednio opisanych stanach.

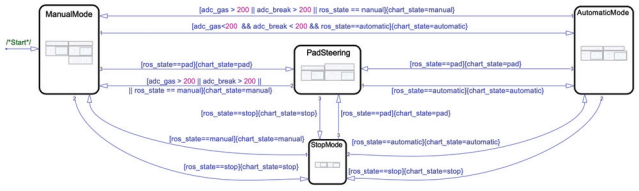
Warunki wyjścia ze stanu *StopMode*

1. *ManualMode* – przejście do stanu sterowania manualnego następuje gdy spełnione zostaną wszystkie wymogi dotyczące bezpieczeństwa.
2. *AutomaticMode* – przejście do stanu jazdy autonomicznej następuje po spełnieniu powyższych warunków przy jednoczesnym żądaniu trybu autonomicznego przez ECU1.
3. *PadSteering* – przejście do stanu jazdy autonomicznej następuje po spełnieniu warunków przejścia do stanu *ManualMode* przy jednoczesnym żądaniu trybu sterowania padem przez ECU1.

Ze względów bezpieczeństwa przyjęto, że w stanie sterowania automatycznego *AutomaticMode* jakakolwiek reakcja kierowcy np. skręcenie kierownicy, naciśnięcie pedału hamulca itp. powoduje wyjście sterownika ECU2 z tego stanu i przejścia do stanu sterownia ręcznego *ManualMode*. Innymi słowy sterowanie ręczne ma najwyższy priorytet dzięki temu możliwe jest natychmiastowe przerwanie sterowania automatycznego jeśli kierowca stwierdzi, że jest ono nieprawidłowe. W przypadku kiedy sterownik wyższego poziomu ECU1 nie pracuje, sterownik ECU2 może znajdować jedynie w dwóch stanach *ManualMode* lub *StopMode*, realizując funkcję zwykłego pojazdu bez żadnych elementów autonomicznej jazdy.

Programowanie sterownika ECU2

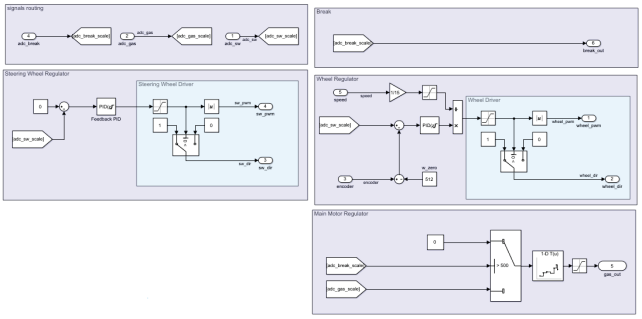
Główny program mikrokontrolera został napisany w języku C z wykorzystaniem biblioteki HAL dostarczonej przez firmę STMicroelectronics. W celu szybkiego i łatwego implementowania różnych typów regulatorów postanowiono skorzystać z możliwości automatycznego generowania kodu oferowanego przez pakiet Matlab© za pomocą narzędzia Simulink Coder©. Kod wygenerowany przez pakiet Matlab© jest cyklicznie wykonywany w pętli głównej programu z częstotliwością 1kHz. Dodatkowo w sterowniku ECU2 wykorzystano mechanizm przerwań do realizacji innych zadań (np. zliczanie impulsów z enkoderów, odczyt stanu różnych przełączników podłączonych do wejść). Logika maszyny stanów przełączająca sterownik ECU2 w jeden z czterech stanów została zaimplementowana za pomocą przybornika StateFlow© programu Matlab©, rys. 3.



Rys. 3. Maszyna stanów sterownika ECU2 pojazdu A-EVE.

Każdy ze stanów w jakim może się znajdować sterownik ECU2 zawiera w sobie podprogram odpowiedzialny za sterowanie w aktualnym stanie. Na rysunku 4 przedstawiono przykładowy podprogram sterowania realizowany w trybie pracy automatycznej, stan *AutomaticMode*.

Wykorzystanie graficznych narzędzi pakietu Matlab© do tworzenie oprogramowania sterującego bardzo upraszcza i przyśpiesza proces projektowania i implementowania różnych rodzajów algorytmów sterowania wykonywanych w sterowniku ECU2. Dodatkowo mamy pewność, że automatycznie wygenerowany kod jest poprawny w sensie formalnym i wolny od błędów [20].



Rys. 4. Podprogram sterowania w stanie pracy *AutomaticMode* pojazdu A-EVE.

Sterowanie nadrzędne - sterownik ECU1

Algorytmy sterowania nadrzędnego realizowane są z wykorzystaniem komputera wyposażonego w procesor Ryzen 3800X, posiadający 8 rdzeni, możliwość obsługi 16 wątków, 32GB RAM oraz 250 GB SSD. Sterownik ECU1, realizuje zadania sterowania wyższego poziomu takie jak: wyznaczenie tras globalnych, wyznaczenie tras lokalnych, analiza danych z podłączonych sensorów takich jak kamery głębokości, LIDAR, GPS-y, implementacje różnego rodzaju algorytmów bezpieczeństwa np. awaryjne zatrzymywanie, omijanie przeszkód. W sterowniku ECU1 zaimplementowane są również wszelkiego rodzaju zachowania "behawioralne" pojazdu A-EVE [21].

Sterowanie nadrzędnie implementuje również funkcjonalność komunikacji z użytkownikiem. W tym celu wykorzystano monitor z ekranem dotykowym. Oprogramowanie interfejsu użytkownika pozwala na zarządzaniem sposobem jazdy pojazdu A-EVE oraz podglądem i monitorowaniem różnych parametrów pojazdu. Dodatkowo w warstwie sterowania nadrzędnego implementowane są różnego typu systemy ADAS (***A**dvanced **D**river **A**ssistance **S**ystem*) [22, 23, 24].

Wszystkie dane sterujące wygenerowane przez sterownik ECU1 są przesyłane przez protokół USB-CDC do sterownika ECU2 np. zadana prędkość liniowa, zadany kąt skręcenia kół. Sterownik ECU2 odsyła do sterownika ECU1 dane niezbędne do działania algorytmów sterowania wyższych poziomów np. prędkość liniową, kąt skręcenia kierownicy.

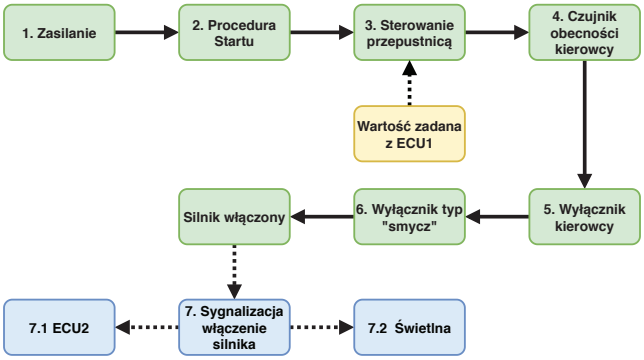
Programowanie sterownika ECU1

Oprogramowanie sterownika ECU1 stworzone zostało z wykorzystaniem frameworka ROS (***R**obot **O**peration **S**ystem*) w wersji Kinect [25, 26, 27]. Oprogramowanie ROS to w zasadzie bardzo obszerne repozytorium najróżniejszych pakietów które realizują różne funkcjonalności np. tworzenie map, wyznaczanie tras, odczyt sensorów itp., aktualną listę pakietów można znaleźć pod adresem `https://index.ros.org/packages/`. W chwili obecnej jest to jeden z najpopularniejszych frameworków wykorzystywanych do tworzenie programów sterowania dla różnego rodzaju robotów. Bardzo dużą zaletą framework-a ROS jest aktywna i duża społeczność dzięki której jest on ciągle rozwijany. Framework ROS daje możliwość programowania z wykorzystaniem różnych języków programowania np. C/C++ oraz Python. Ze względu na wygodę i szybkość tworzenia oprogramowania w projekcie układu sterowania pojazdu A-EVE w pierwszej kolejności wykorzystano język Python. Części oprogramowania których działanie jest krytyczne ze względu na czas stworzono z wykorzystaniem języka C/C++.

Funkcjonalność komunikacji z użytkownikiem (HMI) została zaimplementowana z wykorzystaniem frameworka ReactJS wraz z niezbędnymi dodatkowymi modułami [28]. Framework ReactJS umożliwia tworzenie zaawansowanych programów GUI i jest on powszechnie wykorzystywany w tego typu aplikacjach.

Systemy bezpieczeństwa

Ze względu na rozmiar pojazdu istnieje spore ryzyko uszkodzeń, powstałych w trakcie niezamierzonej jazdy, spowodowanej błędem w kodzie testowanych algorytmów. Aby zwiększyć poziom bezpieczeństwa zastosowano kilka zabezpieczeń połączonych ze sobą szeregowo, które w razie wystąpienia określonych sytuacji rozłączają przekaźnik głównego silnika na najniższym możliwym poziomie (odcięcie obwodu sterowania przekaźnika). Aby możliwa była jazda pojazdem muszą zostać spełnione wszystkie założone warunki. Wszystkie z przestawionych zabezpieczeń znajdują się w obwodzie cewki przekaźnika głównego. Schemat blokowy systemu bezpieczeństwa przedstawia rysunek 5.



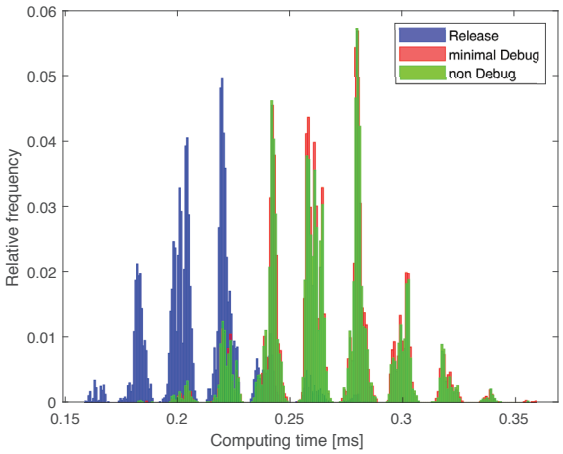
Rys. 5. Schemat blokowy układu bezpieczeństwa pojazdu A-EVE.

Poszczególne elementy systemu zabezpieczeń to:

1. Zasilanie – cewka przekaźnika głównego silnika zasilana jest z przetwornicy 24V, dzięki czemu pojazd A-EVE nie może jechać jeśli przetwornica nie jest zasilana.
2. Procedura startu – realizowana jest bezpośrednio przez sterownik silnika głównego, wymaga ona, aby pedał przepustnicy nie był naciśnięty podczas włączania zasilania sterownika silnika głównego. W przeciwnym przypadku pomimo włączenia zasilania sterownika silnika głównego nie będzie możliwe ruszenie silnikiem. Kolejnym zabezpieczeniem jest stacyjka samochodowa. W przypadku braku lub użycia niewłaściwego kluczyka wysterowanie cewki silnika głównego nie jest możliwe.
3. Sterowanie przepustnicy z ECU2 – logika zawarta w ECU2 odczytuje położenie pedału przepustnicy, zadaną prędkość, sygnał włączenia przekaźnika głównego oraz czujnika nacisku w siedzeniu kierowcy. Na podstawie tych sygnałów steruje sterownikiem silnika głównego. Jeśli wszystkie wymienione warunki nie są spełnione do sterownika silnika nie jest wysyłany żaden sygnał.
4. Czujnik obecności kierowcy – jest to czujnik nacisku znajdujący się w siedzisku fotela kierowcy. W przypadku stwierdzenie braku obecności kierowcy jazda pojazdu A-EVE nie jest możliwa.
5. Wyłącznik kierowcy – zabezpieczenie to wykorzystuje znajdujący się po lewej stronie kierownicy wyłącznik, który podobnie jak pozostałe zabezpieczenia, znajduje się w obwodzie sterującym cewką przekaźnika głównego.
6. Wyłącznik typu "smycz" – wyłącznik ten zrealizowany jest za pomocą zwory wykonanej z cienkiego giętkiego przewodu, który rozpięty jest między dwoma stykami. Jeden ze styków stałych jest magnesem utrzymującym zworę na miejscu podczas jazdy pojazdu. Zwora ta również znajduje się bezpośrednio w obwodzie cewki głównego przekaźnika.
7. Sygnalizacja włączenia silnika – wykorzystuje sygnał cewki przekaźnika głównego.
   (a) ECU2 – sygnał wysterowania cewki przekaźnika głównego odczytywany jest przez ECU2 i zapisywany w bicie `EVE_INFO_ENGINE_ERR` oraz jest on wysyłany do sterownika nadrzędnego ECU1.
   (b) Świetlna – migające pole ostrzegawcze znajdujące się na słupku po lewej stronie pojazdu, podłączone równolegle z cewką przekaźnika głównego. Pole sygnalizuje, że pojazd jest w stanie gotowości do jazdy.

Wpływ flag kompilacji na czas obliczeń ECU2

Jednym z istotniejszych parametrów pracy każdego dyskretnego układu sterowania (a takim jest sterownik ECU2) jest czas potrzebny na wyliczenie odpowiednich wartości sygnałów sterowania. W idealnym przypadku czas ten powinien być maksymalnie krótki a także niezmienny. Kompilując program do postaci wykonywalnej na procesorze istnieje możliwość ustawienia różnych opcji kompilatora które mogą mieć wpływ na czas odpowiedzi dyskretnego układu sterowania. W trybie *debug* kompilator generuje dodatkowe informacje w kodzie maszynowym pozwalające na śledzenie przebiegu realizacji programu za pomocą specjalnego oprogramowania. Ilość dodatkowo generowanego kodu odpowiada szczegółowości informacji jakie możemy uzyskać podczas debugowania kodu. Niestety wraz ze wzrostem szczegółowości debugowania wzrasta rozmiar pliku wykonywalnego a co za tym idzie, czas jego realizacji. Wykres 6 przedstawia wyniki zmierzonego czasu wykonywania programu sterownika ECU2 w zależności od ustawienia różnych opcji konfiguracyjnych kompilacji programu.



Rys. 6. Średnie czasy realizacji programu sterującego przez sterownik ECU2 w zależności od różnych wartości flag kompilujących.

Przebadano następujące konfiguracje kompilacji programu:

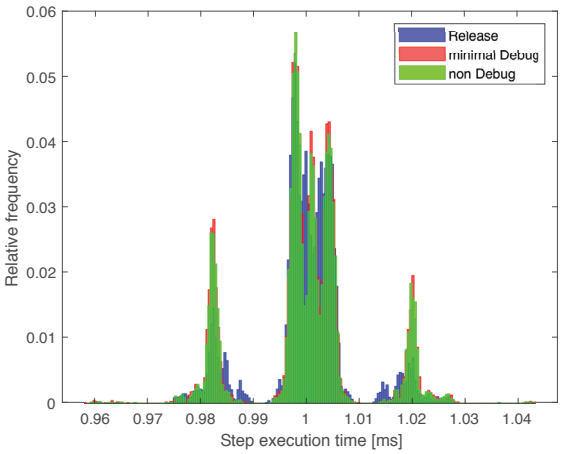
1. tryb *Release* – ustawienia domyślne środowiska SW4STM32,

2. tryb *Debug* – ustawiona flagi `-g1` oraz `-Og` (minimalny poziom informacji debugowych oraz optymalizacja do trybu Debug),
3. ustawiona tylko flaga `-O0` (brak informacji debugowych, wyłączona optymalizacja).

Obliczenia w trybie *Release* trwają miedzy 160μs a 280μs, natomiast zarówno wyłączenie optymalizacji kodu jak włączenie minimalnego trybu debugowego wydłuża zauważalnie czas obliczeń który wynosi miedzy 200μs a 340μs. Różnica ta spowodowana jest w jednym przypadku brakiem optymalizacji a w drugim kodem nadmiarowym będącymi dodatkowymi informacjami o wykonywaniu programu. Rozkład czasu wykonywania programu przypomina rozkład wielomodalny, spowodowane jest to najprawdopodobniej ilością wywołanych przerwań w trakcie wykonywania obliczeń, a także czasem wykonywania obliczeń dla konkretnego zestawu danych. Widać także że, pomiędzy programem skompilowanym z użyciem flag `-g1` oraz `-Og` a programem skompilowanym z flagą `-O0` niema zauważalnych różnic w rozkładzie czasu obliczeń.

Wpływ flag kompilacji na stabilność kroku obliczeń

W odróżnieniu od poprzedniego przypadku w tym teście mierzono czas pomiędzy kolejnymi cyklami rozpoczęcia pętli głównej programu sterownika ECU2. Ustawienia flag kompilacji są identyczny jak w poprzednim podrozdziale ponieważ wykorzystano te same dane wejściowe. Rysunek 7 przedstawiona histogramy zmierzonych wartości kroku obliczeń dla różnych ustawień konfiguracyjnych kompilatora.



Rys. 7. Stabilność kroku realizacji programu sterującego przez sterownik ECU2 w zależności od wartości flag kompilujących.

Można zauważyć, że w odróżnieniu od czasu obliczeń flagi kompilacji nie wypływają znacząco na stabilność kroku obliczeniowego. Dzieje się tak dlatego że, obliczenia wyzwalane są za pomocą sprzętowego zegara znajdującego się wewnątrz układu które wywołuje przerwanie `HAL_SYSTICK_Callback()` z częstotliwością 1kHz. Ze względu na to że, zajmuje się tym osobny układ, sposób generowania kodu programu nie wpływa na stabilność tego przerwania.

Stabilność wywoływania kroku zaburzyć może natomiast inne przerwanie o wyższym priorytecie, które zawiesi działanie przerwania `HAL_SYSTICK_Callback()` na czas wykonania własnego kodu. Dlatego tak ważne jest aby przerwania wykonywały się możliwie jak najkrócej.

Wnioski

W pracy zaprezentowano opis układu sterowania jaki został zaprojektowany i wykorzystany w eksperymentalnym pojeździe autonomiczny A-EVE. Zdecydowano się na wykorzystanie dwupoziomowego układu sterowania który składa się z sterownika nadrzędnego ECU1 i sterownika podrzędnego ECU2. Zadaniem sterownika nadrzędnego ECU1 jest realizacja zadań sterowania związanych z zachowaniem i reakcjami pojazdu A-EVE. Wykonuje on między innymi takie zadania jak: lokalizacja, wyznaczanie ścieżki poruszania, detekcja przeszkód, omijanie przeszkód, nadzorowanie pracy wszystkich podzespołów pojazdu A-EVE. Zadaniem drugiego sterownika ECU2 (sterowanie podrzędne) jest bezpośrednie sterowanie poszczególnymi komponentami pojazdu A-EVE (np. stabilizacja prędkości podłużnej czy kąta skręcenia kół przednich), odczytywanie, skalowanie i kalibracja danych z wejściowych sensorów np. (enkondery). Bardzo ważną funkcjonalnością realizowaną również przez ECU2 jest algorytm zabezpieczeń dzięki czemu korzystanie z pojazdu A-EVE jest bezpieczne. Zaproponowana architektura układu sterowania umożliwia jego łatwą adaptację lub zmianę. Wykorzystanie narzędzi automatycznego generowania kodu oraz wybór frameworka ROS oraz popularnych języków programowania C i Python umożliwia szybką i skuteczną implementację różnych algorytmów sterowania i działania pojazdu A-EVE.

Skuteczność wykorzystanego systemu sterowania została zweryfikowana w trakcie wielu eksperymentów praktycznych. Ich wyniki potwierdziły prawidłowe działanie układu sterowania w tym przede wszystkim sterownika ECU2 wraz zaimplementowaną funkcjonalnością bezpieczeństwa. W następnych etapach prac planowany jest dalszy rozwój projektu, w tym opracowanie algorytmów fuzji danych do ustalania dokładnej pozycji pojazdu A-EVE oraz kolejne testy algorytmów jazdy autonomicznej.

Autorzy: *dr inż. Marek Długosz,*
mgr inż. Michał Roman,
mgr inż. Paweł Węgrzyn
AGH Akademia Górniczo-Hutnicza
Wydział Elektrotechniki, Automatyki, Informatyki i Inżynierii Biomedycznej
a. A. Mickiewicza 30, 30-059 Kraków, Polska
mail: mdlugosz@agh.edu.pl

LITERATURA

[1] Paul Gao, Hans-Werner Kaas, Det Mohr, and Dominik Wee. Automotive revolution–perspective towards 2030 how the convergence of disruptive technology-driven trends could transform the auto industry. *Advanced Industries, McKinsey & Company*, 2016.
[2] Peter Davidson and Anabelle Spinoulas. Autonomous vehicles: what could this mean for the future of transport. In *Australian Institute of Traffic Planning and Management (AITPM) National Conference, Brisbane, Queensland*, 2015.
[3] Hong Cheng. *Autonomous intelligent vehicles: theory, algorithms, and implementation*. Advances in computer vision and pattern recognition. Springer, London, 2011.
[4] S M Veres, L Molnar, N K Lincoln, and C P Morice. Autonomous vehicle control systems — a review of decision making. *Proceedings of the Institution of Mechanical Engineers, Part I: Journal of Systems and Control Engineering*, 225(2):155–195, Mar 2011.
[5] L. Xu, Y. Wang, H. Sun, J. Xin, and N. Zheng. Design and implementation of driving control system for autonomous vehicle. In *17th International IEEE Conference on Intelligent Transportation Systems (ITSC)*, pages 22–28, Oct 2014.
[6] K. Jo, J. Kim, D. Kim, C. Jang, and M. Sunwoo. Development of autonomous car—part 1: Distributed system architecture and development process. *IEEE Transactions on Industrial*

Electronics, 61(12):7131–7140, Dec 2014.
[7] K. Jo, J. Kim, D. Kim, C. Jang, and M. Sunwoo. Development of autonomous car—part 2: A case study on the implementation of an autonomous driving system based on distributed architecture. *IEEE Transactions on Industrial Electronics*, 62(8):5119–5132, Aug 2015.
[8] J. Pérez, V. Milanés, and E. Onieva. Cascade architecture for lateral control in autonomous vehicles. *IEEE Transactions on Intelligent Transportation Systems*, 12(1):73–82, March 2011.
[9] R. Behringer, S. Sundareswaran, B. Gregory, R. Elsley, B. Addison, W. Guthmiller, R. Daily, and D. Bevly. The darpa grand challenge - development of an autonomous vehicle. In *IEEE Intelligent Vehicles Symposium, 2004*, pages 226–231, June 2004.
[10] Sebastian Thrun, Mike Montemerlo, Hendrik Dahlkamp, David Stavens, Andrei Aron, James Diebel, Philip Fong, John Gale, Morgan Halpenny, Gabriel Hoffmann, Kenny Lau, Celia Oakley, Mark Palatucci, Vaughan Pratt, Pascal Stang, Sven Strohband, Cedric Dupont, Lars-Erik Jendrossek, Christian Koelen, Charles Markey, Carlo Rummel, Joe van Niekerk, Eric Jensen, Philippe Alessandrini, Gary Bradski, Bob Davies, Scott Ettinger, Adrian Kaehler, Ara Nefian, and Pamela Mahoney. Stanley: The robot that won the darpa grand challenge. *Journal of Field Robotics*, 23(9):661–692, 2006.
[11] Chris Urmson, Chris Baker, John Dolan, Paul Rybski, Bryan Salesky, William Whittaker, Dave Ferguson, and Michael Darms. Autonomous driving in traffic: Boss and the urban challenge. *AI Magazine*, 30(2):17, Jun. 2009.
[12] Luke Fletcher, Seth Teller, Edwin Olson, David Moore, Yoshiaki Kuwata, Jonathan How, John Leonard, Isaac Miller, Mark Campbell, Dan Huttenlocher, Aaron Nathan, and Frank-Robert Kline. The mit–cornell collision and why it happened. *Journal of Field Robotics*, 25(10):775–807, 2008.
[13] P. Skruch, M. Długosz, and W. Mitkowski. Mathematical methods for verification of microprocessor-based pid controllers for improving their reliability. *Eksploatacja i Niezawodność*, Vol. 17, no. 3:327–333, 2015.
[14] Wojciech Mitkowski, Marta Zagórowska, and Waldemar Bauer. Comparative analysis of dc motor control system. *Applied Mechanics and Materials*, 817:111–121, Jan 2016.
[15] Krzysztof Oprzedkiewcz, Wojciech Mitkowski, and Edyta Gawin. The plc implementation of fractional-order operator using cfe approximation. *Advances in Intelligent Systems and Computing*, pages 22–33, 2017.
[16] Marek Długosz, Michał Roman, and Paweł Węgrzyn. Autonomouse electrical vehicle a-eve. http://shorturl.at/vxAI9, June 2020. [Online; accessed 04-May-2021].
[17] W. Zong, C. Zhang, Z. Wang, J. Zhu, and Q. Chen. Architecture design and implementation of an autonomous vehicle. *IEEE Access*, 6:21956–21970, 2018.
[18] Michał Roman. Design and implementation of autonomous vehicle controller compatible with ros. Master's thesis, AGH Unievrsity of Science and Technology, 2019.
[19] Camine Noviello. *Masterig STM32*. Leanpub, 2018.
[20] Loubna BELHAMEL, Arturo BUSCARINO, Antonio CUCUCCIO, Luigi FORTUNA, and Gaetano RASCONA. Model-based design streamlines for stm32 motor control embedded software system. In *2020 7th International Conference on Control, Decision and Information Technologies (CoDIT)*, volume 1, pages 223–228. IEEE, 2020.
[21] Junqing Wei, Jarrod M Snider, Tianyu Gu, John M Dolan, and Bakhtiar Litkouhi. A behavioral planning framework for autonomous driving. In *2014 IEEE Intelligent Vehicles Symposium Proceedings*, pages 458–464. IEEE, 2014.
[22] M. Komorkiewicz, K. Turek, P. Skruch, T. Kryjak, and M. Gorgon. Fpga-based hardware-in-the-loop environment using video injection concept for camera-based systems in automotive applications. In *2016 Conference on Design and Architectures for Signal and Image Processing (DASIP)*, pages 183–190, Oct 2016.
[23] K.Martin Sagayam, Andrew J, Abhishikt Kadam, and Dang Hien. Vehicle automation and car-following models for accident avoidance. *Przegląd elektrotechniczny*, 2020(1):120–125, 2020.
[24] Safat B Wali, Mohammad A Hannan, Aini Hussain, and Salina A Samad. Comparative survey on traffic sign detection and recognition: a review. *Przeglad Elektrotechniczny*, 1(12):40–44, 2015.
[25] M. Quigley, B. Gerkey, and W.D. Smart. *Programming Robots with ROS: A Practical Introduction to the Robot Operating System*. O'Reilly Media, 2015.
[26] T. Straszheim, B. Gerkey, and S. Cousins. The ros build system. *IEEE Robotics Automation Magazine*, 18(2):18–122, June 2011.
[27] André-Marcel Hellmund, Sascha Wirges, Ömer Şahin Taş, Claudio Bandera, and Niels Ole Salscheider. Robot operating system: A modular software framework for automated driving. In *2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC)*, pages 1564–1570. IEEE, 2016.
[28] Artemij Fedosejev. *React. js essentials*. Packt Publishing Ltd, 2015.