POLITECHNIKA WROCŁAWSKA

WYDZIAŁ ELEKTRONIKI

KIERUNEK: Automatyka i Robotyka

SPECJALNOŚĆ: Komputerowe Systemy Sterowania

* 1. PRACA DYPLOMOWA

INŻYNIERSKA

Sterowanie przepustnicą, układem kierowniczym i układem hamulcowym w samochodzie autonomicznym

Steering with throttle, steering wheel and breaking system in autonomous car

AUTOR:

Imię i nazwisko

PROWADZĄCY PRACĘ:

Dr inż. Łukasz Korus   
Instytut Informatyki, Automatyki i Robotyki

OCENA PRACY:

WROCŁAW 2013

[1. Projekt budowy Pierwszego Polskiego Pojazdu Autonomicznego 6](#_Toc374571029)

[1.1. Ogólnie o projekcie 6](#_Toc374571030)

[1.2. Cele projektu – kamienie milowe 7](#_Toc374571031)

[1.3. Założenia projektu 7](#_Toc374571032)

[2. Sterowanie pojazdem autonomicznym 9](#_Toc374571034)

[2.1. Cele 9](#_Toc374571035)

[2.2. Zakres 9](#_Toc374571036)

[2.3. Przepływ danych 11](#_Toc374571037)

[3. Akwizycja danych 12](#_Toc374571039)

[3.1. Pomiar skrętu kół 12](#_Toc374571040)

[3.2. Pomiar prędkości 13](#_Toc374571043)

[3.3. Pomiar kąta pedału hamulca 13](#_Toc374571046)

[4. Sterowanie systemami pojazdu z poziomu komputera 14](#_Toc374571049)

[4.1. Sterowanie skrętem kół 14](#_Toc374571050)

[4.2. Sterowanie przepustnicą 14](#_Toc374571053)

[4.3. Sterowanie układem hamulcowym 14](#_Toc374571054)

[4.4. Sterowanie skrzynią biegów 15](#_Toc374571057)

[4.5. Sterowanie rozrusznikiem i zapłonem 15](#_Toc374571060)

[4.6. Przycisk bezpieczeństwa 16](#_Toc374571063)

[5. Algorytm sterowania 17](#_Toc374571066)

[5.1. Sterowanie kierownicą 17](#_Toc374571067)

[5.2. Sterowanie prędkością 17](#_Toc374571069)

[5.3. Sterowanie przepustnicą 18](#_Toc374571071)

[5.4. Sterowanie układem hamulcowym 18](#_Toc374571072)

[5.5. Diagram całego układu sterowania 20](#_Toc374571074)

[6. Wybór środowiska 20](#_Toc374571075)

[6.1. Wybór środowiska programistycznego 20](#_Toc374571076)

[6.1.1. Wybór systemu operacyjnego 21](#_Toc374571077)

[6.1.2. Wybór języka programowania 22](#_Toc374571078)

[7. Implementacja 23](#_Toc374571079)

[7.1. Założenia 23](#_Toc374571080)

[7.2. Architektura 23](#_Toc374571081)

[7.3. Definicje zdarzeń 24](#_Toc374571082)

[7.4. Logger 26](#_Toc374571083)

[7.5. Baza danych 27](#_Toc374571084)

[7.6. Komunikator 28](#_Toc374571085)

[7.6.1. Opis interfejsu 28](#_Toc374571086)

[7.6.2. Implementacja komunikatora 30](#_Toc374571088)

[7.7. Kontrolery urządzeń 31](#_Toc374571089)

[7.7.1. RS232Controller 31](#_Toc374571090)

[7.7.2. Servo driver 33](#_Toc374571091)

[7.7.3. Kontroler USB4702 33](#_Toc374571092)

[7.8. Menadżer urządzeń 34](#_Toc374571093)

[7.9. Regulatory 38](#_Toc374571095)

[7.9.1. Regulator PID 38](#_Toc374571096)

[7.9.2. Regulator skrętu kierownicy 39](#_Toc374571097)

[7.9.3. Regulator prędkości 41](#_Toc374571098)

[7.9.4. Regulator położenia pedału hamulca 42](#_Toc374571099)

[7.10. Symulator 44](#_Toc374571100)

[7.11. Graficzny interfejs użytkownika (GUI) 44](#_Toc374571101)

[7.12. Sterowanie w przypadku awarii (to z managera urządzeń) 45](#_Toc374571102)

[7.13. Diagram klas 46](#_Toc374571103)

[8. Zakończenie 48](#_Toc374571104)

[9. Bibliografia 48](#_Toc374571105)

[Schemat 1 Diagram przepływu danych 9](#_Toc373098063)

[Schemat 2 Cykl sterowania 10](#_Toc373098064)

[Schemat 3 Pętla sterowania kierownicą 14](#_Toc373098065)

[Schemat 4 Pętla sterowania prędkością 15](#_Toc373098066)

[Schemat 5 Pętla sterowania układem hamulcowym 16](#_Toc373098067)

[Schemat 6 Pełny diagram sterowania 17](#_Toc373098068)

[Schemat 7 Interfejsy 21](#_Toc373098069)

[Schemat 8 Definicje eventów 22](#_Toc373098070)

[Schemat 9 Schemat bazy danych 24](#_Toc373098071)

[Schemat 10 Interfejs komunikatora 24](#_Toc373098072)

[Schemat 11 Diagram klasy FakeCarCommunicator 25](#_Toc373098073)

[Schemat 12 Diagram klasy RealCarCommunicator 26](#_Toc373098074)

[Schemat 13 Diagram klas SafeRS232Controller i SafeRS232Communicator 27](#_Toc373098075)

[Schemat 14 Diagram sekwencji komunikacji po RS232 27](#_Toc373098076)

[Schemat 15 Diagram klasy ServoDriver 28](#_Toc373098077)

[Schemat 16 Diagram klas Speedometer i USB4702 29](#_Toc373098078)

[Schemat 17 Interfejs użytkownika managera urządzeń 30](#_Toc373098079)

[Schemat 18 Diagram możliwych stanów obieków klasy Device 30](#_Toc373098080)

[Schemat 19 Diagram maszyny stanów obiektów klasy Device 31](#_Toc373098081)

[Schemat 20 Diagram maszyny ogólnych stanów obiektów klasy Device 32](#_Toc373098082)

[Schemat 21 Diagram klas PIDRegulator i PIDSettings 33](#_Toc373098083)

[Schemat 22 Diagram interfejsu ISteeringWheelAngleRegulator 33](#_Toc373098084)

[Schemat 23 DIagram klasy PIDSteeringWHeelAngleRegulator 34](#_Toc373098085)

[Schemat 24 Diagram interfejsu ISpeedRegulator 35](#_Toc373098086)

[Schemat 25 Diagram klasy PIDSpeedRegulator 35](#_Toc373098087)

[Schemat 26 Diagram interfejsu IBrakeRegulator 36](#_Toc373098088)

[Schemat 27 Diagram klasy PIDBrakeRegulator 36](#_Toc373098089)

[Schemat 28 Diagram klasy CarModel 37](#_Toc373098090)

[Schemat 29 Interfejs użytkownika systemu sterowania samochodem 38](#_Toc373098091)

[Schemat 30 Pełny diagram klas systemu sterowania samochodem 39](#_Toc373098092)

# Projekt budowy Pierwszego Polskiego Pojazdu Autonomicznego

## Ogólnie o projekcie

Idea budowy Pierwszego Polskiego Pojazdu Autonomicznego narodziła się na początku 2012 roku w Kole Naukowym Pojazdów i Robotów Mobilnych. Pomysł bardzo spodobał się członkom koła (bardzo mocno powiązanego z Wydziałem Mechanicznym Politechniki Wrocławskiej) i jego opiekunowi – profesorowi Piotrowi Wrzecionarzowi. W ciągu kilku miesięcy koło zebrało ponad 60 osób z wielu wydziałów Politechniki Wrocławskiej, które zadeklarowały chęć pracy nad projektem. Właśnie wtedy dołączyłem do projektu – na początku, jako członek grupy sterowania, po krótkim czasie, jako jej wice-leader, a potem leader. Poprzez niezliczone kontakty profesora Piotra Wrzecionarza i osób zaangażowanych w projekt w ciągu 6 miesięcy udało nam się pozyskać poprzez sponsorów praktycznie nową Toyotę Yaris (wcześniej używaną przez Toyotę do testów). Zebraliśmy też pierwsze środki finansowe. Podczas wakacji członkowie grup mechanicznej i elektronicznej stworzyli podstawy pod możliwość sterowania samochodem za pomocą komputera. Mechanicy zamontowali w samochodzie automatyczną skrzynię biegów (a warto wspomnieć, że ten model Toyoty Yaris nie został stworzony do współpracy z automatyczną skrzynią biegów), zaprojektowalii zamontowali układy sterowania przepustnicą, skrzynią biegów i hamulcem. Elektronicy w tym czasie stworzyli układy pozwalające na sterowanie tymi częściami mechanicznymi, a do tego rozpracowali działanie sygnałów sygnały potrzebne do sterowania układem wspomagania kierownicy (przez co ten układ stał się tak naprawdę układem sterowania kierownicą), a do tego zamontowali czujniki kąta skrętu kierownicy, położenia pedału hamulca, prędkości pojazdu. Dzięki temu mogłem rozpocząć pracę na prawdziwym obiekcie. W późniejszej części prac doprowadziłem do możliwości stabilnego sterowania pojazdem z poziomu komputera oraz wraz z Piotrem Gródkiem integrowałem w module odpowiedzialnym za inteligentne sterowanie pojazdem mój system sterowania z jego kodem (odpowiedzialnym za przetwarzanie obrazu z kamer).



Zdjęcie Toyota Yaris

## Cele projektu – kamienie milowe

Aby kontrolować postęp projektu zaplanowaliśmy cele w postaci następujących kamieni milowych:

1. Pozyskanie samochodu.
2. Przystosowanie samochodu do technologii drive-by-wire, czyli stanu, w którym możemy.  
   w pełni sterować samochodem za pomocą komputera
3. Przejechanie przez samochód zadanej trasy przy wykorzystaniu systemów wizyjnych.
4. Autonomiczne parkowanie.
5. Rozpoznawanie przeszkód na drodze i reagowanie na nie.
6. Rozpoznawanie znaków i sygnałów drogowych.
7. Bezpieczne poruszanie sie po drodze bez obecności innych uczestników ruchu.
8. Bezpieczne poruszanie się po drodze przy obecności innych uczestników ruchu.

Tematem mojej pracy jest stworzenie systemu komputerowego sterującego samochodem, co jest bardzo mocno powiązane z 2 kamieniem milowym.

## Założenia projektu

Na samym początku projektu zdefiniowaliśmy wiele założeń, którymi mieliśmy się kierować. Wiele z nich nie przetrwało próby czasu, a część wyklarowała się dopiero po pewnym czasie.

Oto założenia, którymi kierujemy się w tej chwili:

* pojazd, który tworzymy jest przystosowany do jazdy w normalnych warunkach – unikamy rozwiązań, które pomagają pojazdowi radzić sobie w ściśle określonych, testowych warunkach (np. markery na drodze),
* rozwiązania zastosowane w obecnym samochodzie są na tyle uniwersalne, żeby dało się je przenieść do dowolnego innego pojazdu w krótkim czasie,
* nie tworzymy własnych konstrukcji urządzeń pomiarowych i efektorów, jeśli nie ma takiej potrzeby i dysponujemy wystarczającymi środkami na istniejące rozwiązania,
* oszczędzamy środki finansowe nawet, jeśli w danym momencie mamy ich sporo - brak zabezpieczenia finansowego może zamrozić projekt w przyszłości,
* za podejmowanie decyzji jest odpowiedzialna 1 osoba – szef projektu (Zbigniew Żelazny) – jesteśmy przez to w stanie szybko reagować na problemy ,
* podział na grupy nie jest w żadnej kwestii wiążący, istnieje głównie po to, żeby ludzie wiedzieli, z kim powinni rozmawiać o problemach danego typu,
* preferujemy komunikację face-to-face i zarządzanie przez leaderów (bardziej naturalnych niż wyznaczonych) nad sformalizowanymi metodami zarządzania projektem,
* wyciągając wnioski z technik scrumowych rozwijamy projekt metodą małych kroków, nie tworzymy nierealnych planów projektów, których pierwszy efekt działania będzie widoczny dopiero po długim czasie ( > 2 tygodnie).

## Grupy w projekcie

Centralne zarządzanie projektem, w którym bierze udział kilkadziesiąt osób jest praktycznie niemożliwe, więc wraz z trwaniem projektu wyklarował się podział na kilka grup:

* Grupa koncepcyjna:
  + prowadzona przez Zbigniewa Żelaznego – szefa całego projektu,
  + członkami tej grupy są leaderzy innych grup,
  + zajmuje się definiowaniem celów prac innych grup;
* Grupa mechaniki:
  + prowadzona przez Filipa Godlewskiego,
  + zajmuje się przeróbkami mechanicznymi pojazdu oraz wszelkimi problemami związanymi z obsługą pojazdu, także elektryką;
* Grupa elektroniki:
  + prowadzona przez Daniela Dudzika i Konrada Zawadę,
  + zajmuje się ogólnopojętymi częściami elektronicznymi;
* Grupa sterowania:
  + prowadzona przez Macieja Oziębły (przeze mnie),
  + zajmuje się oprogramowaniem sterującym samochodem i akwizycją danych z czujników;
* Grupa oprogramowania:
  + prowadzona początkowo przez Dobromiła Budzianowskiego, a w tej chwili przez Piotra Gródka,
  + zajmuje się przetwarzaniem obrazów z kamer i wysokopoziomowym kierowaniem pojazdem;
* Grupa marketingu:
  + prowadzona przez Macieja Chełmiskiego - wice-szefa projektu,
  + zajmuje się ogólnie pojętą propagandą, a także rozmowami ze sponsorami, kontaktem z mediami, finansami oraz wyjazdami;
* Grupa komunikacji z otoczeniem:
  + prowadzona przez Pawła Sawickiego,
  + miała zajmować sie oczujnikowaniem pojazdu,
  + została rozwiązana przez brak kompetencji członków zespołu   
    i nieefektywność działań podejmowanych przez nich.

W trakcie moich prac nad projektem zajmowałem się głównie sterowaniem, ale miałem też ścisły kontakt z elektronikami (Danielem Dudzikiem i Konradem Zawadą)   
i mechanikami (Filip Godlewski, Michał Lochter i Zbigniew Żelazny), a także byłem częścią grupy koncepcyjnej i grupy oprogramowania.

# Sterowanie pojazdem autonomicznym

## Cele

Główne cele mojej pracy nad sterowaniem Pojazdem Autonomicznym:

* ustalenie interfejsów komunikacji elektroniki z systemem sterowania,
* zapewnienie możliwie prostego interfejsu do sterowania pojazdem warstwom nadrzędnym,
* stworzenie systemu kontroli nad urządzeniami elektronicznymi w pojeździe,
* stworzenie systemu zapisującego stany podsystemów tak, aby umożliwić prostą naprawę błędów,
* stworzenie systemu sprawującego kontrolę nad pojazdem w czasie rzeczywistym,
* ustabilizowanie systemu tak, aby nie sprawiał problemów w trakcie ciągłej pracy  
  w niesprzyjających warunkach.

## Zakres

Zagadnienia sterowania pojazdem obejmują tylko część problemów, z którymi spotkaliśmy się w trakcie całego projektu. W tej pracy skupię się na części projektu, która była stworzona przeze mnie.

W zakres moich prac nad pojazdem wchodziło:

* stworzenie modelu urządzeń mechanicznych,
* ustalenie interfejsów urządzeń elektronicznych służących do zbierania informacji  
  o samochodzie dla:
  + kąta kierownicy,
  + szybkości chwilowej samochodu,
  + kąta wciśnięcia hamulca,
* stworzenie programu zbierającego dane o samochodzie w czasie rzeczywistym (data aquisition),
* stworzenie programu kontrolującego pracę czujników i wysyłającego odpowiednie błędy i reagującego na nie (fault management),
* ustalenie interfejsów komunikacji z efektorami dla:
  + przepustnicy,
  + układu wspomagania kierownicy,
  + skrzyni biegów,
  + zapłonu,
  + rozrusznika,
* stworzenie programu kontrolującego pracę całych zespołów urządzeń   
  i reagującego na ich błędy,
* zdefiniowanie interfejsu wyjściowego z mojej części programu dla warstw nadrzędnych,
* stworzenie regulatorów pozwalających sterować parametrami samochodu:
  + skrętem kół,
  + szybkością,
  + kątem wciśnięcia hamulca,
* stworzenie fake’ów klas służących do komunikacji ze wszystkimi urządzeniami   
  w celu możliwości testowania stabilności systemu i jego odporności na zakłócenia,
* stworzenie interfejsu użytkownika pozwalającego na manualne sterowanie wartościami zadanymi,
* stworzenie systemu logowania komunikatów aplikacji sterującej obsługującego różne priorytety wiadomości,
* stworzenie systemu bazodanowego pozwalającego na przechowywanie dużych ilości informacji na temat samochodu napływających w czasie rzeczywistym,
* stworzenie modułu pozwalającego na logowanie maksymalnej ilości dostępnych informacji, o samochodzie i jego sterowaniu do bazy danych,
* zapewnienie stabilności całego systemu,
* stworzenie sekwencji startowej pozwalającej na szybki i bezpieczny start samochodu z jednoczesnym testowaniem komunikacji z urządzeniami zewnętrznymi,
* integracja programu z modułem nadrzędnym.

## Przepływ danych

W celu pokazania dróg, jakimi przepływają dane w samochodzie autonomicznym stworzyłem schemat przepływu danych (dataflow chart):



Schemat Diagram przepływu danych

## Schemat działania systemu



Schemat Cykl sterowania

# Akwizycja danych

## Pomiar skrętu kół

### Opis działania

Pomiar skrętu kół odbywa się pośrednio, poprzez pomiar skrętu kierownicy. Kierownica jest połączona za pomocą paska z urządzeniem pomiarowym, które rozpoznaje skręt walca przymocowanego do płytki pomiarowej. Dane z urządzenia są zbierane poprzez płytkę zaprojektowaną i wykonaną przez Konrada Zawadę i Daniela Dudzika, która wysyła te dane do komputera sterującego poprzez wirtualny port RS232 oparty o USB.

### Znane problemy

Przy szybkich skrętach kierownicy pasek przekazujący obroty na wałek pomiarowy przesuwa się względem kierownicy, co skutkuje dodaniem stałego błędu do pomiarów. Błąd ten można zniwelować poprzez kalibrację układu (w programie do sterowania pojazdem widać, w jakiej pozycji są według niego koła – wystarczy ustawić koła samochodu prosto, a następnie przesuwać pasek względem kierownicy tak, aby program komputerowy pokazał, że koła są idealnie proso). Problem ten jest dla nas bardzo dotkliwy – przy długiej pracy bez kalibracji samochód stawał się niestabilny nawet przy jeździe po prostym torze, ponieważ błąd pomiaru dochodził nawet do . Aby wyeliminować ten problem planujemy zastąpić pasek małym łańcuchem i zębatkami (wtedy poślizg nie będzie możliwy). Dokładność urządzenia pomiarowego to ok .

Walec pomiarowy można obracać o dowolny kąt, lecz w chwili przekręcenia (obrotu ponad w dowolną stronę mierząc od poziomu 0) wartość zmierzona zmienia się   
z maksymalnejna minimalną. W przypadku dużej dekalibracji czujnika i dużego wychylenia kierownicy zdarzało się, że walec przekręcał się ponad wartość maksymalną, a program uznawał, że koła są skręcone maksymalnie w lewo, kiedy tak na prawdę były skręcone maksymalnie w prawo. Doprowadzało to do niestabilności pracy systemu. Rozwiązanie problemu z dekalibracją kierownicy powinno rozwiązać także ten problem. Możliwe jest także stworzenie dokładniejszego odwzorowania urządzenia pomiarowego w oprogramowaniu, jednak wydaje się, że takie rozwiązanie nie jest konieczne, ponieważ grupa elektroniki pracuje już nad sprzętowym rozwiązaniem tego problemu.

## Pomiar prędkości

### Opis działania

Pomiar prędkości samochodu odbywa się poprzez pomiar prędkości na lewym przednim kole. System działa tak, jak mierniki rowerowe – na kole znajduje się 5 magnesów, które podczas każdego obrotu koła są wykrywane przez czujnik. Miernik zlicza impulsy wywołane przez magnesy, a następnie dane te są przesyłane do komputera poprzez urządzanie Advantech USB4702 interfejsem USB.

### Znane problemy

Podczas pracy zdarza się, że komputer bardzo rzadko (czasem nawet raz na 2-3s zamiast 50 razy na sekundę) odbiera dane z czujnika prędkości. Problem został częściowo rozwiązany przez zmianę sposobu działania systemu obrabiającego dane z czujników (jest to dokładniej opisane w dalszej części pracy).

## Pomiar kąta pedału hamulca

### Opis działania

Pomiar odbywa się niemal identycznie jak pomiar skrętu kół. Wciskanie pedału hamulca powoduje skręt wałka na czujniku obrotu. Dane z czujnika są przekazywane   
do komputera poprzez wirtualny port RS232 oparty o USB. Wciskanie hamulca obraca wałkiem bez żadnej przekładni, co sprawia, że wałek obraca się tylko w niewielkim zakresie, przez co niedokładność pomiarowa układu jest dosyć duża i wynosi ok 3%. Układ ten jest w pełni sztywny, przez co nie istnieją tutaj problemy takie jak przy pomiarze skrętu kół (dekalibracja i przekraczanie zakresu).

### Znane problemy

Mała dokładność pomiaru (aczkolwiek na chwilę obecną wystarczająca). Przy manualnym wciśnięciu pedału hamulca system nie jest w stanie rozpoznać sytuacji, która miała miejsce i próbuje przeciwdziałać sile, która wciska hamulec (a warto dodać, że silnik sterujący hamulcem jest bardzo silny). Stwarza to bardzo niebezpieczną sytuację, w której podczas autonomicznej jazdy samochodu ciężko go w szybki sposób zatrzymać (trzeba jednocześnie użyć przycisku awaryjnego i samemu wcisnąć hamulec). Z tego powodu podczas pokazów często wyłączaliśmy obsługę układu hamulcowego, żeby można było szybciej reagować na pojawiające się problemy (wtedy wystarczyło samemu wcisnąć hamulec).

# Sterowanie systemami pojazdu z poziomu komputera

## Sterowanie skrętem kół

### Opis działania

Do skręcania kołami wykorzystujemy oryginalny układ wspomagania kierownicy Toyoty Yaris. Oszukujemy układ wspomagania tak, aby reagował ruszając kierownicą   
w odpowiedni sposób. Ze strony sterowania wygląda to tak, że aby sterować kierownicą trzeba wysyłać parą przewodów symetryczne sygnały z zakresu 0-5V (tak, aby suma obu sygnałów była równa 5V). Przy sygnale 2,5V na obu przewodach układ wspomagania jest bezczynny, przy sygnałach 5V i 0V układ wspomagania będzie przykładał maksymalną siłę do kierownicy w kierunku zależnym od podanego napięcia.  
Z naszych doświadczeń wynika, że siła przyłożona do kierownicy jest mniej więcej proporcjonalna do zmiany napięcia. Sterowanie układem odbywa sie poprzez port analogowy urządzenia Advantech USB4702, który jest sterowany z komputera przez interfejs USB.

### Znane problemy

Z nieznanych powodów czasem zdarza się, że po włączeniu silnika pojazdu lub podłączeniu kabla USB do komputera (nawet bez przesyłania z komputera jakiegokolwiek sterowania) kierownica wpada w bardzo mocne drgania. Podejrzewamy, że odpowiada za to układ przełączający wspomaganie pomiędzy trybem manualnym prowadzenia pojazdu i trybem prowadzenia pojazdu przez komputer (który nota bene nie działa prawidłowo – wspomaganie działa obecnie jedynie w trybie sterowania przez komputer).

## Sterowanie przepustnicą

Przepustnica jest sterowana za pomocą serwomechanizmu, który bezpośrednio nawija linkę podłączoną do przepustnicy. Podczas testów okazało się, że moc pojazdu jest bardzo nieliniowym odwzorowaniem ustawienia serwa – przy początkowym przesuwaniu serwa samochód praktycznie nie reagował, dopiero przy połowie zakresu obrotu serwomechanizmu samochód zaczyna reagować. Ciężko dokładnie określić charakterystykę mocy silnika od pozycji przepustnicy, ale wydaje się, że jest ona  
w przybliżeniu liniowa, gdy zaczynamy mierzyć mniej więcej w środku (dokładny punkt widoczny jest w kodzie) zakresu obrotu serwa. Serwomechanizm jest sterowany przez servodriver, który jest połączony z komputerem poprzez interfejs USB, a do sterowania układem trzeba wysyłać do niego wartości z zakresu 0-8000 (gdzie 0 to wyłączenie serwa, 1 to minimalny obrót, a 8000 to maksymalny obrót).  
Sterowanie przepustnicą to najbardziej niezawodny element naszego pojazdu.

## Sterowanie układem hamulcowym

### Opis działania

Aby sterować układem hamulcowym podłączyliśmy do pedału hamulca w samochodzie linkę, która jest nawijana przez silnik prądu stałego.

### Znane problemy

Przy złym sterowaniu układ może próbować nawijać linkę, kiedy jest ona nawinięta.   
W takim przypadku układ zasilający silnik (Advantech USB4702) i sam silnik bardzo się grzeją, co doprowadza do czasowego wyłączenie zasilania silnika (na ok 5 minut). Taki stan już dawno się nam nie przytrafił, jednak trzeba o tym pamiętać.

Ostatnim razem, kiedy silnik został źle wysterowany (ręcznie z komputera) linka zahaczyła o kable i silnik zaczął nawijać w swoją stronę elektronikę z naszego samochodu (płytki, kable). Straty były niewielkie, ale trzeba bardzo uważać, żeby nic nie było zahaczone o linkę do hamowania. Nie można zapominać o tym, że całe obecne sterowanie układem hamulcowym może działać albo w trybie manualnym albo w trybie automatycznym (manualne wciśnięcie hamulca podczas pracy w trybie automatycznym doprowadza do błędnego działania układu sterującego, który próbuje odwinąć już odwiniętą linkę, przez co linka zaczyna nawijać się w drugą stronę, co całkowicie zmienia model układu hamulca – przez co, gdy sterowanie próbuje ‘odcisnąć’ hamulec, tak naprawdę go zaciska, co doprowadza do niestabilności całego układu). Ze względów bezpieczeństwa podczas pokazów zawsze korzystamy z trybu manualnego, dzięki czemu cały układ automatycznego sterowania hamulcem jest wyłączony.

W najbliższym czasie zastąpimy silnik nawijający linkę układem hydraulicznym, którym będziemy mogli dokładnie sterować. Nowy układ został zaprojektowany tak, żeby można było bezproblemowo hamować manualnie w trakcie jazdy autonomicznej.

## Sterowanie skrzynią biegów

### Opis działania

Układ sterowania skrzynią biegów został ostatnio zmodyfikowany. W tej chwili serwo (takie same jak na przepustnicy) steruje zębatkami zmieniającymi pozycję skrzyni biegów. Układ jest sterowany z komputera poprzez servodriver (specjalną płytkę PCB) połączony z komputerem poprzez interfejs USB.

### Znane problemy

Układ jest obecnie źle skalibrowany (ale dopiero powstał, więc można mu to wybaczyć)   
i nie jest możliwe włączenie wstecznego biegu. Kalibracja układu wymaga jego rozkręcenia. Wydaje się, że po skalibrowaniu, układ powinien działać bezawaryjnie przez długi czas.

## Sterowanie rozrusznikiem i zapłonem

### Opis działania

Układ sterowania rozrusznikiem i zapłonem to jeden z najmłodszych układów   
w samochodzie (podziękowania dla Konrada Zawady). Sterowanie nim odbywa się poprzez porty cyfrowe urządzenia Advantech USB4702 połączonego z komputerem interfejsem USB. Ustawienie 4 portów cyfrowych  
na odpowiednie wartości pozwala na włączenie lub wyłączenie zapłonu i rozrusznika. Wyłączenie zapłonu podczas pracy samochodu powoduje wyłączenie silnika, więc możemy tą metodą gasić silnik z poziomu komputera.

### Znane problemy

Układ nie zabezpiecza przed uruchomieniem rozrusznika podczas pracy samochodu (co doprowadza do głośnych zgrzytów, ale po konsultacjach z profesjonalnymi mechanikami dowiedzieliśmy się, że nie jest to duży problem o ile nie zdarza się to zbyt często). W tej chwili komputer nie ma żadnej informacji o tym, czy silnik jest włączony, więc operator programu do automatycznego sterowania powinien pamiętać, że nie powinno się włączać efektorów w menadżerze urządzeń, kiedy silnik jest włączony.

## Przycisk bezpieczeństwa

### Opis działania

Układ zabezpieczający z założenia miał wyłączać wszystkie elementy, które dołożyliśmy do samochodu, tak, aby samochód działał jak oryginalna Toyota Yaris. Układ ten nie wyłącza, ani nie hamuje samochodu, lecz zmienia go w maszynę, którą można bez problemu sterować ręcznie tak jak zwykłą, seryjnie produkowaną Toyotą Yaris.

### Znane problemy

Problem z wariującą kierownicą (opisany w dziale „Sterowanie skrętem kół”) nie jest hamowany po wciśnięciu przycisku bezpieczeństwa – pozwala to sądzić, że układ zabezpieczający nie działa prawidłowo. Konsultacje z elektronikami i testy układu pokazały, że rzeczywiście nawet po wciśnięciu przycisku bezpieczeństwa pojawiają się zakłócenia w układzie wspomagania kierownicy. Elektronicy pracują nad rozwiązaniem tego problemu.

# Algorytm sterowania

## Sterowanie kierownicą

Przy sterowania kierownicą mamy możliwość odczytu aktualnego kąta kierownicy. Samo sterowanie odbywa się poprzez układ wspomagania kierownicy, za pomocą którego można przykładać siłę do kierownicy. Wartością zadaną jest kąt położenia kierownicy, ponieważ jest on wprost proporcjonalny do skrętu kół. Dla takiego modelu kierownica ma charakter całkujący, więc zdecydowałem się na zastosowanie regulatora proporcjonalno-różniczkującego (PD).

Warto zauważyć, że model kierownicy zmienia się podczas jazdy. W zależności od prędkości jazdy, rodzaju nawierzchni i temperatury opon zmienia się siła potrzebna do skrętu kierownicy. W obecnym układzie nie bierzemy tych zmian pod uwagę, ponieważ obecna pętla sterowania bardzo dobrze radzi sobie z kontrolą kierownicy i jej implementacja nie wymaga poprawek, a prosta pętla sterowania gwarantuje bardzo szybkie działanie układu sterującego.

## Diagram pętli sterowania



Schemat Pętla sterowania kierownicą

## Sterowanie prędkością

Przy sterowaniu prędkością odczytujemy aktualną prędkość z sensora na kole,   
a sterujemy za pomocą przepustnicy i hamulca. Przyjmując, że siła napędzająca samochód jest proporcjonalna do stopnia otwarcia przepustnicy, a siła hamująca jest proporcjonalna do wciśnięcia pedału hamulca można stworzyć przybliżony model sterowania prędkością. Prędkość samochodu zależy od bardzo wielu czynników, ale do zagadnień sterowania wystarczające jest, że do rozpędzenia dużej masy samochodu trzeba przez długo czas napędzać go z dużą siłą, co oznacza, że prędkość ma charakter inercyjny. Do sterowania obiektem tego typu świetnie nadaje się regulator PID. Sterowanie przepustnicą jest trywialne, jednak do sterowania hamulcem trzeba zastosować osobny regulator. Zdecydowałem się na zastosowanie kaskadowego układu regulacji.

## Diagram pętli sterowania



Schemat Pętla sterowania prędkością

## Sterowanie przepustnicą

Do sterowania przepustnicą używamy serwomechanizmu, który bezpośrednio zmienia jej otwarcie. Rozwiązanie to okazuje się niezawodne, więc zdecydowałem się sterować przepustnicą układem bez sprzężenia zwrotnego (open-loop). W rzeczywistości istnieje zamknięta pętla sterowania sterująca serwem przepustnicy – pętla sterowania prędkością.

## Sterowanie układem hamulcowym

Korzystając z założenia, że siła hamowania układu hamulcowego jest proporcjonalna do wciśnięcia pedału hamulca zdecydowaliśmy się na sterowanie układem   
z wykorzystaniem pomiarów z czujnika kąta wciśnięcia hamulca i siłowym działaniem na pedał hamulca. Wciśnięcie pedału hamulca ma charakter całkujący, lecz na hamulec działa ciągła, zmienna siła, która ma zwrot przeciwstawny do wciskania hamulca, więc przy sterowaniu potrzebny jest człon całkujący. Silnik sterujący hamulcem ma także funkcję „stop”, więc zastosowałem dodatkowy blok stopujący silnik, gdy wartość sterowania jest bardzo niska w celu ograniczenia czasu działania silnika (który musi walczyć  
z przeciwstawną siłą odciskającą hamulec). Rozwiązanie takie sprawiło, że nie mamy problemów  
z przegrzewaniem się silnika do sterowania pedałem hamulca. Sposób zamontowania linki sterującej pedałem hamulca pozwala na ręczne sterowanie hamulcem   
w awaryjnych sytuacjach. Niestety pętla sterowania odbiera takie zachowanie hamulca, jako zmianę stanu pedału hamulca spowodowaną przez warunki otoczenia samochodu, przez co układ sterowania próbuje odcisnąć hamulec, co doprowadza do niebezpiecznych sytuacji.

### Diagram pętli sterowania



Schemat Pętla sterowania układem hamulcowym

## Diagram całego układu sterowania



Schemat Pełny diagram sterowania

# Wybór środowiska

## Wybór środowiska programistycznego

Sterowanie samochodem autonomicznym jest zadaniem sterowania w czasie rzeczywistym, podczas którego nawet małe błędy systemu mogą powodować duże problemy. Od oprogramowania obsługującego systemy samochodu wymagana jest przede wszystkim niezawodność i zgodność  
z urządzeniami pokładowymi. Ważnymi parametrami systemu są też maksymalny   
i średni czas reakcji. Nie można zapominać też, że systemy tego typu tworzą ludzie, więc trzeba wziąć pod uwagę ich umiejętności operowania w systemach i językach danego typu. Od wyboru języka programowania i dostępnych narzędzi takich jak IDE, debuggery, narzędzia do statycznej i dynamicznej analizy kodu, frameworki do testowania czy biblioteki graficzne w bardzo dużym stopniu zależy też prędkość pisania kodu. Trzeba też pamiętać, że przy ograniczonych zasobach czasu narzędzia przyspieszające pracę programisty mogą zadecydować o sukcesie całego projektu. W moim przypadku w grę wchodziły 2 systemy operacyjne – Windows 7 i Arch Linux.

### Wybór systemu operacyjnego

Przed wyborem systemu operacyjnego warto zastanowić się, jak dany system sprawdzi się w różnych kategoriach:

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Windows 7** | **Arch Linux** | Komentarz |
| Niezawodność | Po wielu poprawkach uznawany za stabilny system operacyjny | Jeden z najbardziej stabilnych systemów operacyjnych, jakie istnieją |  |
| Zgodność z peryferiami samochodu | Pełna zgodność | Brak sterowników do USB4702 | Istnieje możliwość reverse-engineering’u i prób komunikacji z USB4702 pod Linuxem |
| Prostota użycia | Bardzo prosty | Dla obytych ludzi nie sprawia problemów |  |
| Zużywane zasoby jednostki centralnego | Nawet do 1GB pamięci RAM, przy odpowiedniej konfiguracji niewielkie zużycie procesora | Do 200MB pamięci RAM, minimalne zużycie procesora |  |
| Obsługa dużej ilości wątków | Standardowe „Windows’owe” podejście | Podejście „UNIX’owe” | System wątków zastosowany w Windowsie 7 jest w praktyce dużo wydajniejszy, niż ten stosowany pod Linuxem |
| Możliwości konfiguracji | Możliwość zmian wielu opcji systemowych | Możliwość pełnej rekonfiguracji systemu – łącznie ze zmianami w jądrze systemu operacyjnego |  |
| Cena | Do celów badawczych możliwość zastosowania darmowej akademickiej wersji | Darmowy |  |

Z konfrontacji systemów operacyjnych ciężko wyłonić zwycięzcę. Główną przewagą systemu „Windows 7” jest pełna zgodność ze wszystkimi urządzeniami peryferyjnymi samochodu, co pozwala oszczędzić dużą ilość czasu na dostosowywaniu USB4702 do działania pod systemem Arch Linux. Jednocześnie wydaje się, że do sterowania systemem czasu rzeczywistego, jakim jest samochód autonomiczny, o wiele bardziej nadają się systemy UNIX’owe ze względu na bardzo duże możliwości konfiguracji, ich stabilność i małe zużycie zasobów jednostki centralnych. Po rozważeniu wszystkich za   
i przeciw doszedłem do wniosku, że nie można zadecydować o wyborze systemu bez zwrócenia uwagi na środowisko programistyczne, którego chce się użyć.

### Wybór języka programowania

Po wstępnej analizie możliwości szybko wyłonili się 2 kandydaci na środowisko programistyczne. Pierwszy to C++0x11 z kompilatorami Clang (do debuggu) i gcc (do standardowej pracy systemu) pod Linuxem lub Visual C++ pod Windowsem oraz wykorzystanie biblioteki Boost. Drugim kandydatem został język C# z platformą .NET4.5 pod Windowsem lub .NET3.0 z użyciem Mono pod Linuxem. Stworzyłem też porównanie obu środowisk programistycznych:

|  |  |  |
| --- | --- | --- |
|  | **C++** | **C#** |
| Wydajność | Zdecydowany zwycięzca | Użycie maszyny wirtualnej spowalnia pracę programów |
| Popularność | Bardzo wysoka – szczególnie wśród elektroników | Wysoka wśród programistów w naszym projekcie |
| Stabilność | Bardzo stabilna platforma, programy na bazie C++ stworzone przez niedoświadczonych programistów często polegają jednak na niezdefiniowanych zachowaniach języka (tak zwanych UB), co znacznie pogarsza stabilność programów | Stabilna platforma, na której łatwo pisać stabilne programy |
| Szybkość tworzenia kodu | Stosunkowo niska dla ludzi bez dużego doświadczenia | Bardzo wysoka |
| Debugger | gdb – jeden z najlepszych istniejących debuggerów, jego wadą jest trudność obsługi | Visual Studio 2012 zapewnia bardzo dobry i przystępny debugger, który pozwala na proste debuggowanie wielowątkowych programów |
| Dostępność bibliotek | Pakiet boost zapewnia większość potrzebnych narzędzi, Qt jest uznaną biblioteką graficzną | Bardzo duża ilość bibliotek, dostępne wbudowane biblioteki graficzne (WPF, Windows Forms) pozwalające na bardzo szybkie tworzenie GUI |
| Wycieki pamięci | Odwieczny problem języka, łatwo go uniknąć stosując wyspecjalizowane narzędzia do dynamicznej analizy kodu (Valgrind) | Wbudowany garbage collector praktycznie eliminuje problem wycieków pamięci, ale tworzy problem z gwarantowanym czasem odpowiedzi programów (garbage collector może w dowolnym momencie przerwać program na pewien okres) |
| IDE | Pod Linuxem edytory tekstowe (Vim + ctags) lub Qt Creator, pod Windowsem Visual Studio 2008 (nowsze wersje nie zapewniają pełnego wsparcia czystego C++ (bez CLI)) | Visual Studio 2012 – prawdopodobnie najlepsze istniejące IDE, do tego doskonale współpracujące z C# i wspomagające tworzenie programów z GUI, debug. Posiada wbudowane narzędzia wysokiej klasy do statycznej analizy kodu. Bardzo lubiane przez członków projektu |
| Biblioteka do obróbki obrazu | OpenCV – prawdopodobnie najlepsza darmowa biblioteka tego typu | EmguCV – biblioteka OpenCV przeportowana do C#, oparta na skompilowanych w C++ narzędziach, przez co jest niemal równie wydajna jak natywny OpenCV |

Przy zagadnieniach sterowania, wydajność programu nie jest kluczową cechą, ponieważ ilość obliczeń potrzebnych do wykonania jest mała. W przypadku przetwarzania obrazu, które też jest częścią projektu problem mniejszej wydajności platformy .NET jest rozwiązany przez zastosowanie biblioteki EmguCV, która korzysta z natywnego kodu OpenCV napisanego w C++. Najważniejszym czynnikiem okazuje się być szybkość rozwijania projektu. Analiza tego zagadnienia nie pozostawiła wątpliwości. Język C#   
i platforma .NET pozwalają na znacznie szybsze tworzenie oprogramowania przez wbudowanie wielu mechanizmów w język oraz bardzo dobre IDE - Visual Studio, które rozwiązuje większość problemów, stawianych przed programistami C++.

Język C# znacznie lepiej współpracuje z Windowsem (na którym jest rozwijany), niż   
z Linuxem, a do tego w przypadku wyboru systemu Windows można zastosować nowszą wersję platformy .NET (4.5 zamiast 3.0 podLinuxem). Ostatecznie zdecydowaliśmy się na wybór języka C# pod platformą .NET4.5 i systemu operacyjnego Windows 7   
w najnowszej możliwej wersji.

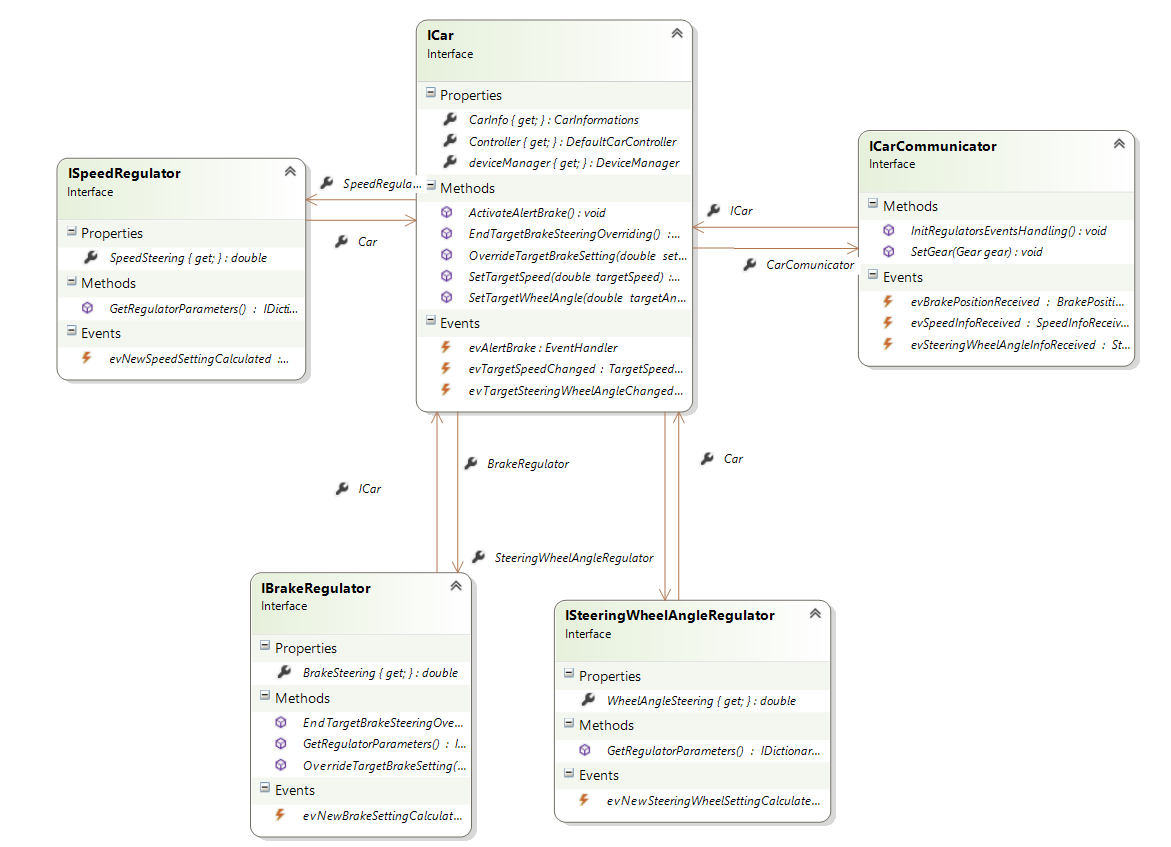
# Implementacja

## Założenia

System zajmuje się przetwarzaniem danych w czasie rzeczywistym. Jego głównymi zadaniami są pobieranie danych z czujników, regulacja procesów i sterowanie urządzeniami wyjściowymi. Bodźcami do działania większości podsystemów jest przeważnie otrzymanie nowych danych z czujnika. Z tego powodu zdecydowałem się na zastosowanie wzorca projektowego obserwatora. Zastosowałem typowe dla C++ podejście z wykorzystaniem sygnałów i slotów (znanego głównie z Qt i biblioteki boost::signals2), lecz w .NETowej formie – z wykorzystałem eventów. Ważnym założeniem całego systemu jest też to, że we wszystkich miejscach używane są dane  
w jednostkach układu SI.

## Architektura

Wydzieliłem 2 główne bloki programu – regulatory i komunikatory. W przypadku regulatorów łatwo rozdzielić je na regulatory prędkości, kąta pedału hamulca i skrętu kierownicy. W przypadku komunikacji z urządzeniami zewnętrznymi prosty podział kodu nie jest możliwy, ponieważ wiele systemów korzysta fizycznie z tych samych systemów i ciężko stworzyć dla nich abstrakcje rozdzielające ich zadania, ponieważ podsystemy mogłyby mocno ze sobą interferować i zaburzać swoją pracę. Zdecydowałem się też na stworzenie interfejsu dla całego samochodu udostępniającego na zewnątrz możliwie najprostszy interfejs. W końcowej wersji poza zadawaniem prędkości i kąta skrętu kół samochodu zdecydowałem się też umieścić możliwość przejęcia sterowania nad wartością zadaną położenia hamulca, aby było możliwe szybsze reagowanie   
w sytuacjach zagrożenia.



Schemat Interfejsy

## Definicje zdarzeń

W systemie opartym na zdarzeniach często używa się predefiniowanych zdarzeń, które przenoszą informację takiego samego typu jak ten, którego potrzebujemy. W przypadku mojego systemu zdecydowałem się na zdefiniowanie wszystkich eventów, żeby uniknąć problemów z użyciem niewłaściwego eventa. Definicje te okazały się też bardzo pomocne przy logowaniu danych systemowych. Każda z definicji eventa składa się   
z 2 części. Pierwszą z nich jest event handler – jest to C++’owy slot, do którego można dokładać akcje, które mają się wydarzyć, kiedy event zostanie wywołany. Drugą częścią jest argument eventu. To w nim przekazywane są informację dotyczące zdarzenia  
(na przykład wynik pomiaru pochodzący z systemu pomiaru prędkości). Ważnym założeniem przy definiowaniem argumentów było też tworzenie ich w taki sposób, żeby użycie danych z eventa nie mogło powodować ich modyfikacji, co mogłoby być źródłem trudnych do wykrycia błędów.

Kod przykładowej definicji zdarzenia:

public delegate void TargetSpeedChangedEventHandler(object sender,  
 TargetSpeedChangedEventArgs args);

public class TargetSpeedChangedEventArgs : EventArgs

{

private double targetSpeed;

public TargetSpeedChangedEventArgs(double speed)

{

targetSpeed = speed;

}

public double GetTargetSpeed()

{

return targetSpeed;

}

}



Schemat Definicje eventów

## Logger

Ważnym, często niedocenianym elementem każdego systemu czasu rzeczywistego jest system logujący zdarzenia systemowe, tak, żeby w przypadku awarii można było zobaczyć, co spowodowało problem. W moim przypadku nie skorzystałem z żadnego   
z gotowych rozwiązań, lecz stworzyłem własną prostą, statyczną klasę do logowania.

Jej interfejs jest bardzo prosty:

public static void Log(Object loggingObj, string msg, int priority = 0)

Klasa logująca na podstawie prostych wywołań typu:

Logger.Log(this, “Some test message with no priority”);

Logger.Log(this, “Some important test message”, 1);

Potrafi wygenerować wpis dający dużą ilość informacji o tym co się stało:

[CarController.TestClass]:<<’00:00:21’>>: Some test message with no priority

<<<PRIORITY:1>>>[CarController.TestClass]:<<’00:00:22’>>: Some important test message

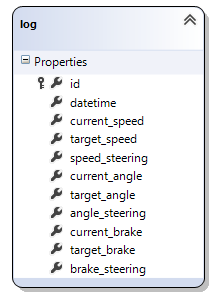
Poza czasem wywołania każdej z metod logujących (mierzonym od włączenia programu) widać też, jaka klasa wywołała sygnał do logowania. Istotny jest również system priorytetów pozwalający łatwo odfiltrować mniej potrzebne informacje.

Wszystkie logi (powyżej zdefiniowanego priorytetu) są zapisywane do pliku i wyrzucane na konsole. Logger ma z założenia pracować z systemami wielowątkowymi,   
a zapisywanie danych do pliku nie jest możliwe z wielu wątków jednocześnie. Problem ten można było rozwiązać przez zastosowanie prostego mutexa, lecz czekanie przez sytem na otwarcie pliku przez loger jest działaniem niepożądanym , więc to rozwiązanie zostało odrzucone. Ciekawym rozwiązaniem wydaje się też asynchroniczne zapisywanie danych do pliku przez kontroler logów, jednak mogłoby sprawić ono problem przy crash’ach systemu – ostatnie logi prawdopodobnie nie zostałyby zapisane do pliku.   
Z tego powodu zdecydowałem się na proste rozwiązanie polegające na próbie otwarcia pliku przy tworzeniu loga, a jeśli taki plik jest już otwarty, porzucaniem takiego loga   
i wypisaniem komunikatu o niezapisaniu wiadomości do pliku na konsolę. Rozwiązanie to w praktyce sprawdza się bardzo dobrze, a informacja o takim błędzie pojawiła się tylko kilka razy (a podczas każdej godziny pracy systemu zapisywane jest około 70MB logów).

Z perspektywy czasu widać, że taka konstrukcja klasy loggującej jest bardzo pomocna   
w dalszej części projektu i pomogła rozwiązać wiele problemów podczas wielu godzin debuggowania.

## Baza danych

Do analizowania zachowań systemu poza systemem prostych logów powstał system bazodanowy, do którego w czasie pracy systemu zapisywane są najważniejsze dane   
o stanie samochodu, sterowaniu i wartościach zadanych. Skorzystałem z serwera Microsoft SQL Serwer 2012, ponieważ jest on dostępny za darmo dla studentów Elektroniki na Politechnice Wrocławskiej oraz bardzo stabilny i łatwo korzysta się z niego w C# (przez bibliotekę System.Linq). Za dodawanie danych do bazy odpowiada osobny wątek o niskim priorytecie. Struktura bazy danych jest bardzo prosta. Baza zawiera tylko 1 tabelę:



Schemat Schemat bazy danych

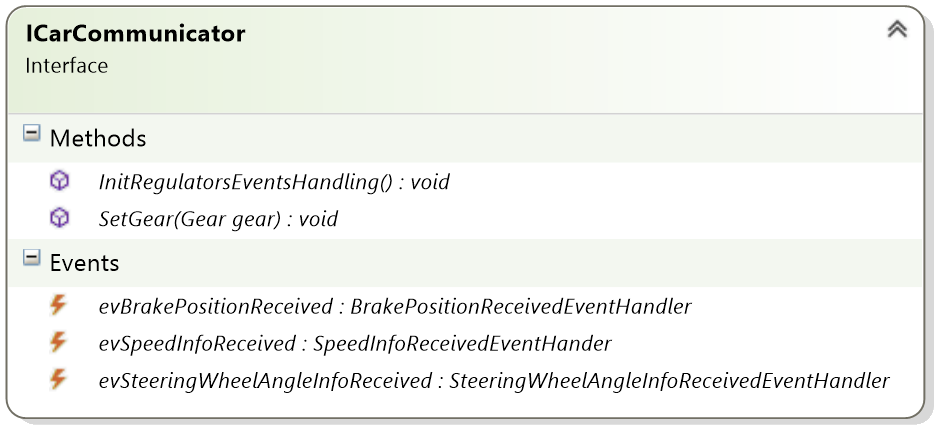
Wątek do logowania co około 100ms tworzy nowy rekord w bazie danych, po czym usypia się. W przyszłości dane z tej bazy mogą posłużyć do prowadzenia badań nad modelem pojazdu, co z pomoże w stworzeniu lepszych regulatorów do wszystkich podsystemów. Z prostej analizy zawartości bazy po wielogodzinnych przejazdach wynika, że regulatory może nie są idealne, ale sprawują się bardzo dobrze, a całe systemy sterowania zachowują się bardzo stabilnie.

## Komunikator

### Opis interfejsu

Interfejs komunikatora powstał w celu stworzenia abstrakcji nad urządzeniami wejścia/wyjścia w całym systemie. Rozwiązanie takie sprawia, że w bardzo prosty sposób mogłem stworzyć symulator udający pracę urządzeń komunikujących się z moim system, co pozwoliło na proste testowanie regulatorów i warunkach zbliżonych   
do rzeczywistych.

Interfejs komunikatora:



Schemat Interfejs komunikatora

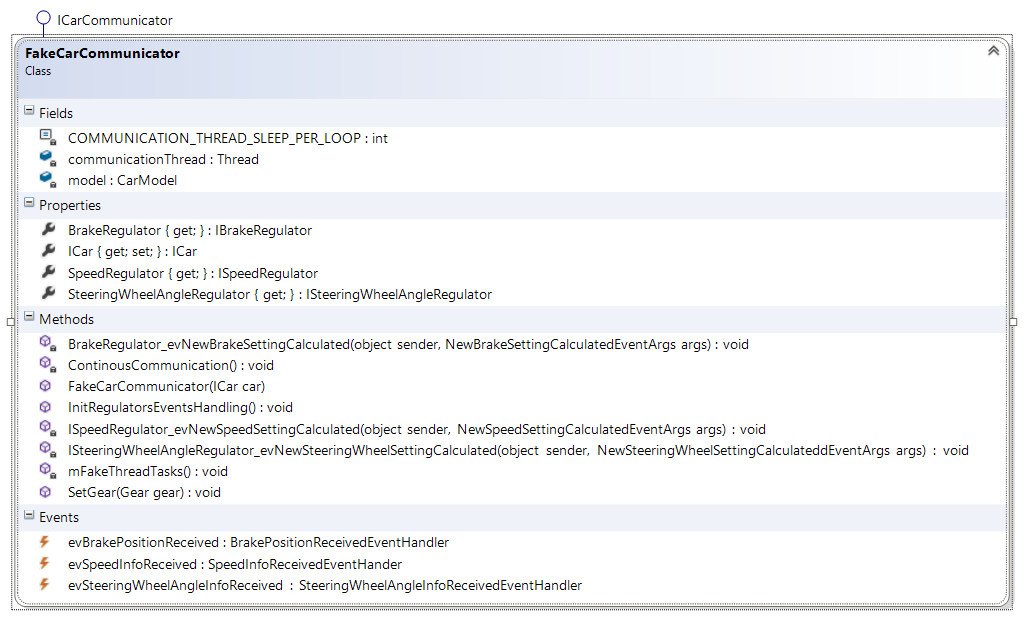
System sterowania opiera się na wzorcu projektowym obserwatora, co jest bardzo dobrze widoczne na przykładzie interfejsu. Komunikator udostępnia 3 event handlery. Każdy z nich jest wywoływany, kiedy komunikatory otrzymają pewną informację   
z sensorów – o położeniu pedału hamulca, prędkości pojazdu lub skręcie kół. Każdy obiekt mający dostęp do obiektu komunikatora może zarejestrować się na otrzymywanie danych z czujników i sprecyzować akcje, jakie ma wykonać po ich otrzymaniu. Poza urządzeniami wejściowymi komunikator obsługuje też urządzenia wyjściowe i za pomocą metody public void InitRegulatorsEventsHandling() subskrybuje się na eventyz zewnątrz dotyczące otrzymania sterowań. Rozwiązanie takie łamie zasadę Demeter (zwaną też „zasadą minimalnej wiedzy”), ponieważ klasa implementująca ten interfejs musi mieć wiedzę o reszcie systemu, ale zdecydowałem się na ten krok, ponieważ w moim systemie interfejsy są twardo zdefiniowane i odnoszenie się do nich w innych interfejsach nie sprawia problemu (a poza tym nikt nie obalił zasady „Worse is Better”, która sprzyja mojemu rozwiązaniu, bo przeplanowanie całej architektury byłoby bardzo czasochłonne, a nie przyniosłoby widocznych rezultatów). Sterowanie biegami nie podlega regulacji (ustawianie biegów odbywa się za pomocą serwomechanizmu), więc metoda zmieniająca bieg została tutaj wystawiona wprost:

void SetGear(Gear gear).

### 7.7.1. Implementacja udawanego komunikatora

Podczas tworzenia systemu zdecydowałem się na implementację fake’owego komunikatora w pierwszej kolejności. Rozwiązanie takie pozwoliło mi na pisanie pozostałych części systemu bez fizycznego dostępu do samochodu (a wiadomo, że   
w domowym zaciszu pracuje się lepiej niżw samochodzie). Dodatkowo takie posiadanie symulatora pozwala na bardzo szybkie debugowanie regulatorów, ponieważ łatwo w ten sposób dostarczać do nich praktycznie dowolne spreparowane dane.

Na diagramie klasy widać, że poza zaimplementowaniem wprost interfejsu IcarCommunicator, komunikator ten nie wykonuje wielu innych czynności:

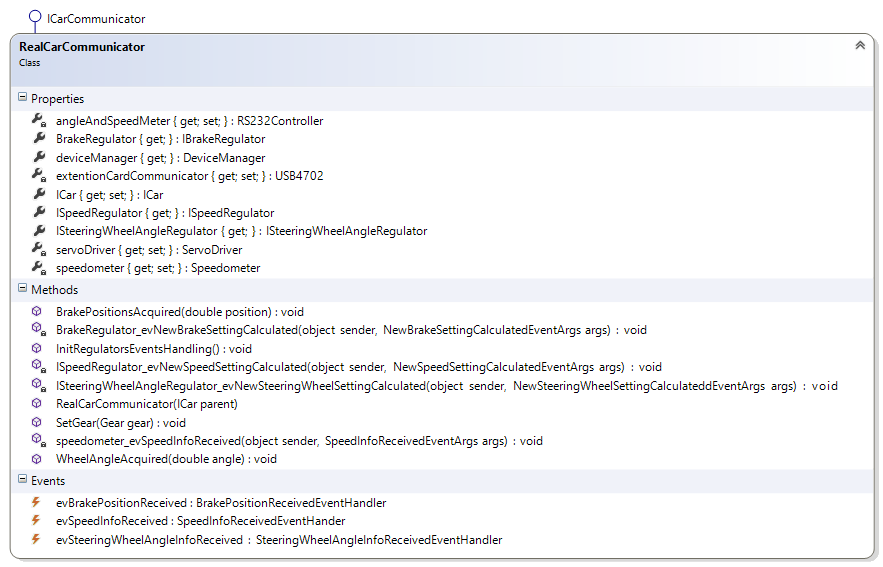


Schemat Diagram klasy FakeCarCommunicator

Praca fake’owego komunikatora opiera się głównie na przekazywaniu informacji   
z i do symulatora samochodu. Działanie symulatora zostanie opisane w dalszej części pracy.

### Implementacja prawdziwego komunikatora

Niestety, do działania prawdziwego systemu, poza udawanym komunikatorem, musiałem napisać też taki, który komunikuje się z rzeczywistymi urządzeniami. Zdecydowałem się na wydzielenie odpowiedzialności za komunikację z konkretnymi do osobnych klas, a w komunikatorze pozostawienie tylko sterowania ich zachowaniami.



Schemat Diagram klasy RealCarCommunicator

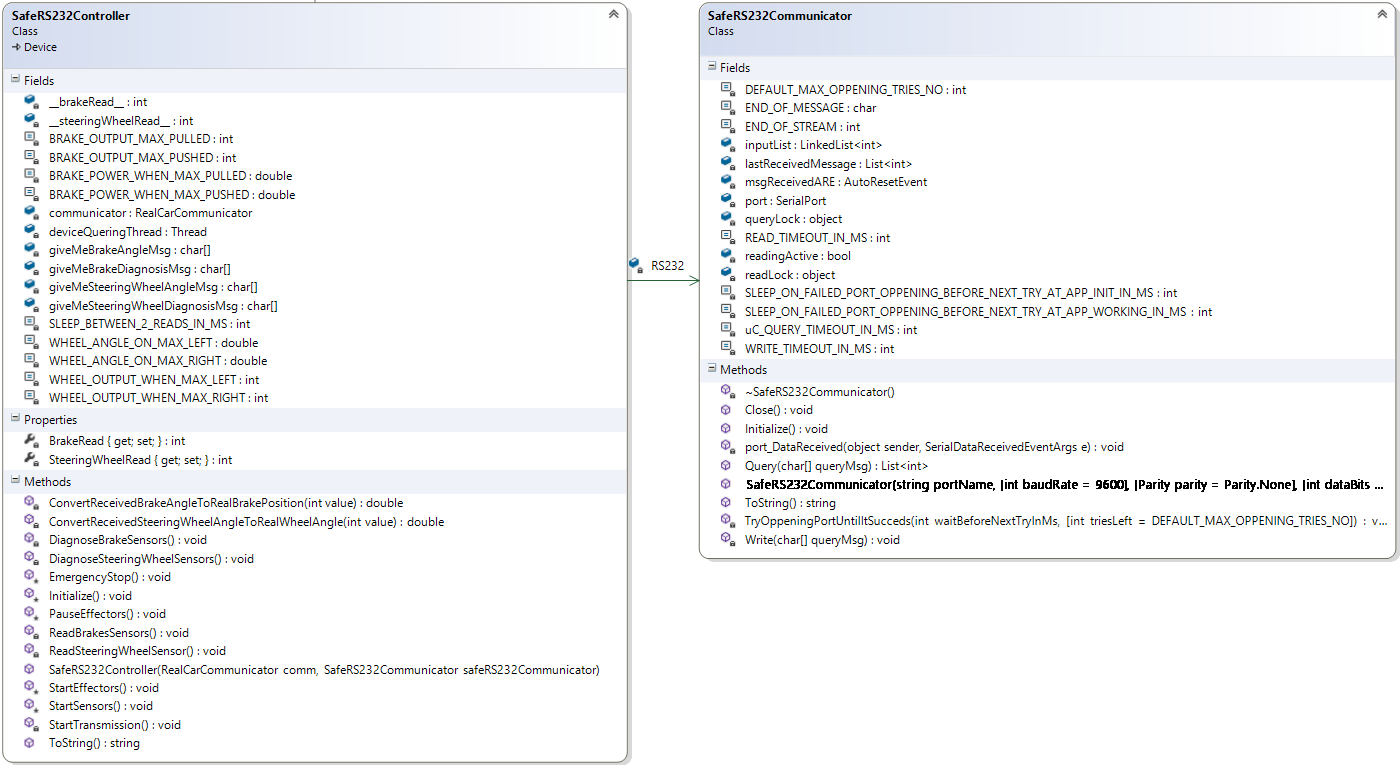
Klasa zawiera odniesienia do obiektów 3 klas sterujących konkretnymi urządzeniami (żeby zrozumieć, jakie dane przepływają przez komunikator warto rzucić okiem na diagram przepływu danych). Klasy te to RS232Controller, USB4702 i ServoDriver. Wywołania przychodzące do obiektu klasy RealCarController są przekazywane do odpowiedniego z kontrolerów urządzeń, a dane zebrane przez sensory są widoczne na zewnątrz przez wywołanie odpowiedniego eventu przez obiekt tej klasy.

## Kontrolery urządzeń

Klasy kontrolujące pracę urządzeń implementują abstrakcję Device, przez co mogą być zarządzane przez manager urządzeń (który jest opisany w dalszej części rozdziału). Same kontrolery powstały w celu stworzenia abstrakcji nad komunikacją z urządzeniami zewnętrznymi.

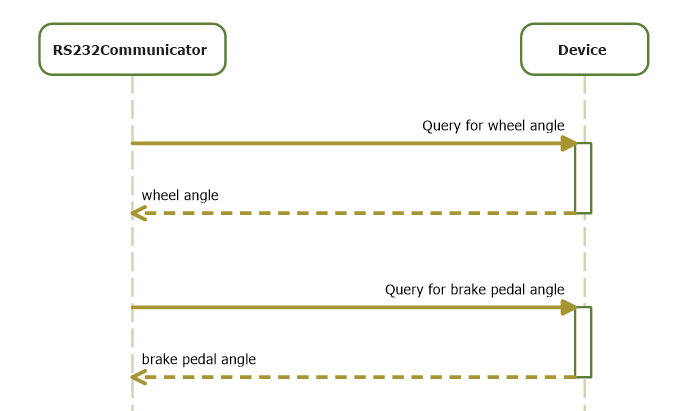
### RS232Controller

Z powodu problemów, które sprawiała kontrola przesyłania danych przez RS232 (które są powodowane głównie przez błędy samego urządzenia i błędy w transmisji) zdecydowałem się na przepisanie starej klasy kontrolującej (obecnej teraz w kodzie jako RS232Controller\_old)  
i rozbicie funkcjonalności na 2 części. Pierwsza z nich to komunikacja z urządzeniem stworzonym przez Konrada Zawadę, drugą zaś kontrola samego procesu komunikacji.   
W ten sposób stworzyłem następujące klasy:



Schemat 13 Diagram klas SafeRS232Controller i SafeRS232Communicator

Klasa SafeRS232Communicator służy tylko i wyłącznie do komunikacji z płytką PCB odpowiedzialną za zbieranie pomiarów. Samo odpytywanie płytki ma charakter poolingowy. Diagram sekwencji pętli odpytywania jest przedstawiony poniżej.

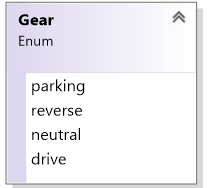
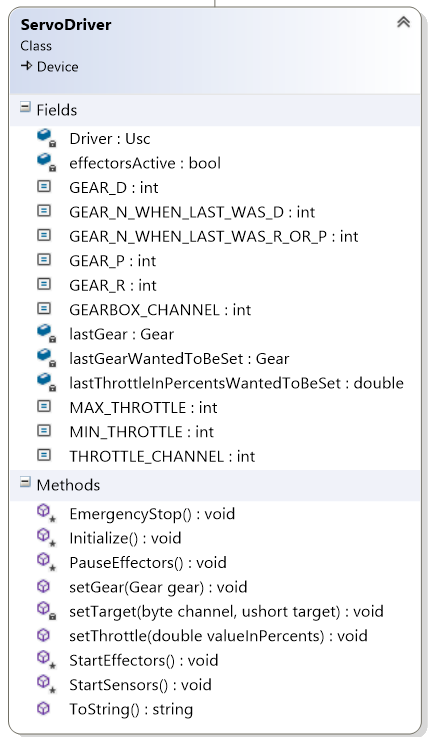


Schemat Diagram sekwencji komunikacji po RS232

Komunikator zajmuje się transmisją, kontrolą błędów i połączenia. W razie problemów zgłasza błąd do kontrolera RS232. Rolą kontrolera jest wywoływanie odpowiednich poleceń na komunikatorze i reagowanie na zmiany stanu urządzenia.

### Servo driver

ServoDriver jest klasą wydającą polecenia do serwomechanizmów obsługujących przepustnicę i skrzynię biegów. Ich obsługa to sterowanie w otwartej pętli sprzężenia zwrotnego (open-loop), więc klasa jest stosunkowa prosta – opiera się tylko   
na wydawaniu rozkazów płytce kontrolującej serwa.

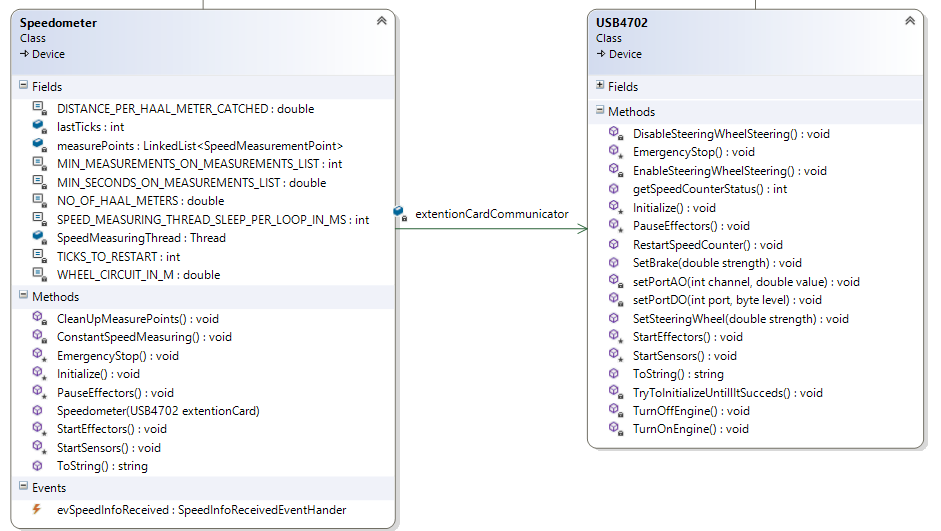


Schemat Diagram klasy ServoDriver

Klasa ma metody pozwalające wprost na sterowanie zmianą biegów i położeniem przepustnicy. Implementuje abstrakcję Device tak, żeby w razie problemów móc reagować awaryjnie.

### Kontroler USB4702

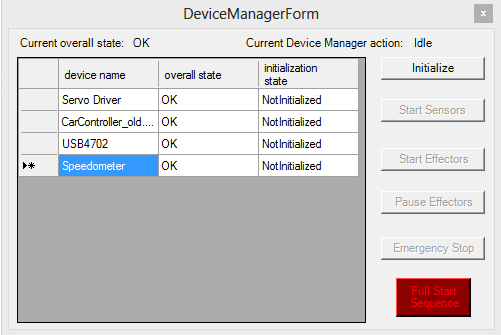
Głównym urządzeniem wejścia/wyjścia podłączonym do systemu jest USB4702. Jest to moduł posiadający wejścia cyfrowe oraz wyjścia cyfrowe i analogowe. Za pomocą USB4702 system może sterować rozrusznikiem i zapłonem pojazdu (przez co może go włączać i wyłączać). To przez niego odbywa się sterowanie układem hamulcowym oraz pomiar prędkości samochodu. Wszystkie zadania poza pomiarem prędkości są dosyć proste, więc nie zdecydowałem się na wydzielanie ich do osobnych klas. Sam pomiar prędkości przysporzył nam sporo problemów, a trzeba pamiętać, że jest to bardzo ważny układ – w przypadku złych wartości pomiarów samochód może zacząć się niebezpiecznie rozpędzać. Z tego powodu stworzyłem klasę Speedometer, której zadaniem jest możliwie dokładny pomiar prędkości samochodu oraz kontrolowanie pomiarów   
i raportowanie awarii (żeby było to możliwe klasa Speedometer dziedziczy po klasie Device i jest kontrolowana przez manager urządzeń). Problemy z pomiarami prędkości wynikały z tego, że z pewnych powodów urządzenie dostarcza zbyt rzadko wyniki pomiarów (przejść magnesów na kole przez czujnik). Algorytm polegający na liczeniu średniej ze 100 ostatnich próbek nie dawał pożądanych rezultatów, gdy czas pomiędzy pomiarami nie był stały (a nie był nawet w przybliżeniu stały – potrafił zmienić się do 200 razy). Stworzyłem strukturę danych trzymającą pomiary z minimalnie 1 ostatniej sekundy  
w taki sposób, żeby na liście były przynajmniej 3 pomiary. Rozwiązało to nasze problemy i teraz nawet przy dziwnym działaniu urządzenia pomiarowego nie mamy problemów   
z pomiarami prędkości.



Schemat Diagram klas Speedometer i USB4702

## Menadżer urządzeń

Manager urządzeń powstał dopiero w późniejszym etapie rozwijania systemu sterowania samochodem autonomicznym. Jego zadaniem jest kontrola stanów urządzeń elektronicznych  
w pojeździe i reagowanie na ich błędy. W przypadku awarii manager urządzeń może doprowadzić wszystkie urządzenia do stanu, w którym doprowadzają one do awaryjnego zatrzymania pojazdu. Inną funkcją managera jest też pokazywanie w stanie rzeczywistym stanów wszystkich urządzeńi kontrolowanie sekwencji startowej pojazdu poprzez interfejs użytkownika (stworzony przy pomocy Windows Forms):



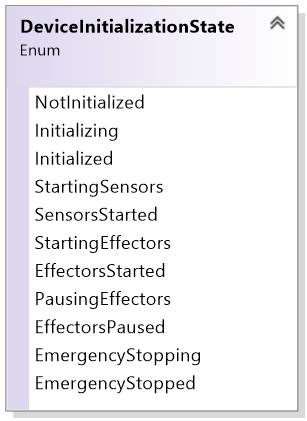
Schemat Interfejs użytkownika managera urządzeń

Co ciekawe, dzięki modułowi napisanemu przez mojego kolegę Piotra Gródka (który jest autorem części kodu odpowiedzialnego za analizę obrazu w czasie rzeczywistym i szefem grupy programistów), manager zadań potrafi rozpoznawać mowę i po rozpoznaniu komend „Start” i „Stop” odpowiednio uruchamiać i zatrzymywać pojazd   
(a po rozpoznaniu komendy „Jurek” opowiada kawał za pomocą syntezatora mowy).

Najważniejszą funkcją managera zadań jest jednak kontrolowanie stanu urządzeń   
i reagowanie na błędy. Każde z urządzeń kontrolowanych przez manager musi implementować abstrakcję Device.

### Device

Device jest abstrakcją urządzenia, która dodaje do klasy urządzenia maszynę stanów obsługującą zdarzenia przy wejściu do poszczególnych stanów i przejścia pomiędzy stanami. Abstrakcja loguje wszystkie przejścia pomiędzy stanami, przez co w logach bardzo łatwo odnaleźć pojawiające się problemy. Gdy urządzenie zmienia swój stan wywoływany jest event evDeviceStateHasChanged, na który zasubskrybowany jest DeviceManager, przez co w prosty sposób może on śledzić stany wszystkich urządzeń.



Schemat Diagram możliwych stanów obieków klasy Device

Maszyna stanów inicjalizacji każdego urządzenia zdefiniowana jest w następujący sposób:

Schemat Diagram maszyny stanów obiektów klasy Device

Każde z urządzeń musi mieć zdefiniowane akcje na wejściu do każdego ze stanów. Gdy urządzenie ulegnie awarii powinno zgłosić swój błąd do managera urządzeń.

Poza maszyną stanów inicjalizacji, każde urządzenie ma też swój ogólny stan. Jeśli awaria jest krytyczna dla systemu, urządzenie powinno ustawić swój stan na „Error”, wtedy manager urządzeń wywoła na wszystkich urządzeniach metodę „EmergencyStop”, co doprowadzi samochód do awaryjnego zatrzymania. Tymczasowo, z powodu przykładania dużej uwagi do bezpieczeństwa projektu, ustawiłem także akcję awaryjnego stopu po wejściu dowolnego urządzenia do stanu „Warning”.

Przejścia pomiędzy ogólnymi stanami urządzenia:



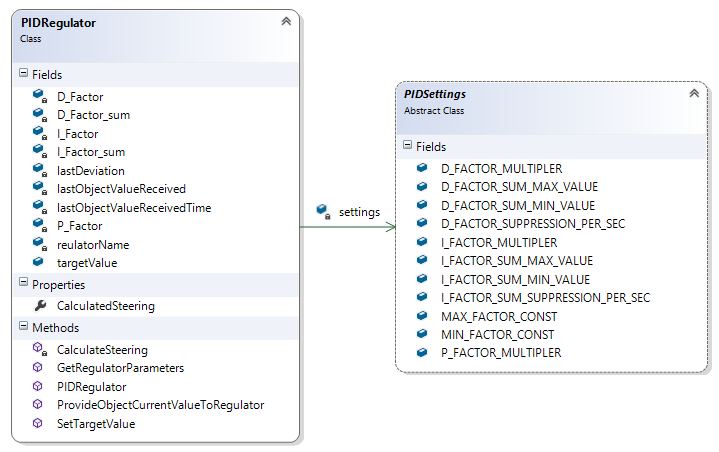
Schemat Diagram maszyny ogólnych stanów obiektów klasy Device

## Regulatory

Zadaniem stworzonego przeze mnie systemu jest sterowanie pojazdem, co sprowadza się do regulacji jego procesów. Bazując na wiedzy o modelu urządzeń stworzyłem klasy regulujące skręt kierownicy, prędkość i położenie pedału hamulca. Zdecydowałem się na zastosowanie regulatorów PID ze względu na ich uniwersalność i prostotę.

### Regulator PID

Regulator PID jest stosowany do regulacji 3 różnych procesów, więc zdecydowałem się na stworzenie klasy będącej czystym regulatorem PID.

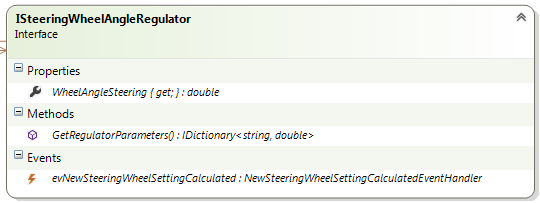


Schemat Diagram klas PIDRegulator i PIDSettings

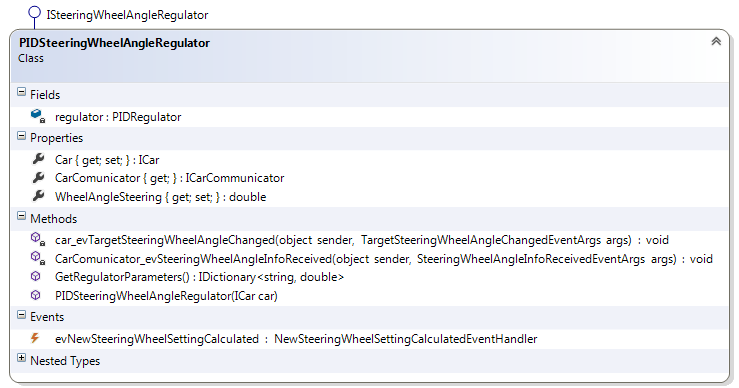
Jak widać po ustawieniach nie zastosowałem typowego regulatora PID, lecz wprowadziłem do niego pewne modyfikacje (znane mi z eksperymentów przeprowadzonych podczas budowy robota klasy line-follower o wdzięcznej nazwie „Piesek Leszek”). W moim regulatorze człon różniczkujący tak samo jak człon całkujący sumuje kolejne wartości, ale nie są to uchyby, lecz zmiany uchybów. Drugą modyfikacją jest procentowe zmniejszanie wartości sum modułów różniczkującego i całkującego na jednostkę czasu. W przypadku członu różniczkującego takie zmniejszanie wartości jest bardzo szybkie (ok 35% na sekundę). Analiza regulatorów nie jest tematem mojej pracy inżynierskiej, jednak modyfikacje te znacząco poprawiły regulację parametrów samochodu. Automatycy, z którymi rozmawiałem poddawali pod wątpliwość sens takich działań, jednak na przykładzie robota klasy line-follower i obserwacji poprawy działania samochodu autonomicznego pokazałem, że rozwiązania te pomagają w sterowaniu systemami tego typu.

### Regulator skrętu kierownicy

Regulator skrętu kierownicy oparty jest na bazowym regulatorze PID (poprzez kompozycję). Implementuje interfejs ISteeringWheelAngleRegulator.



Schemat Diagram interfejsu ISteeringWheelAngleRegulator



Schemat DIagram klasy PIDSteeringWHeelAngleRegulator

W praktyce nie jest to regulator PID, lecz regulator PD. Na poniższym listeningu można zobaczyć jak wyglądają ustawienia takiego regulatora.

private class Settings : PIDSettings

{

public Settings()

{

//P part settings

P\_FACTOR\_MULTIPLER = 5;

//I part settings

I\_FACTOR\_MULTIPLER = 0;

I\_FACTOR\_SUM\_MAX\_VALUE = 0;

I\_FACTOR\_SUM\_MIN\_VALUE = 0;

I\_FACTOR\_SUM\_SUPPRESSION\_PER\_SEC = 1; //1.0 = suppresing disabled

//D part settings

D\_FACTOR\_MULTIPLER = 4;

D\_FACTOR\_SUPPRESSION\_PER\_SEC = 0.3;

D\_FACTOR\_SUM\_MIN\_VALUE = -1000;

D\_FACTOR\_SUM\_MAX\_VALUE = 1000;

//steering limits

MAX\_FACTOR\_CONST = 100;

MIN\_FACTOR\_CONST = -100;

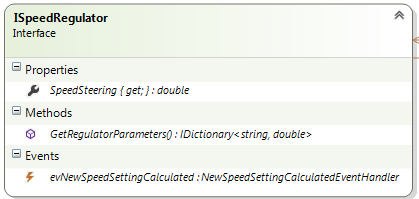
}

}

Ustawienia te pokazują, jak dokładnie skonfigurowany jest regulator PD do sterowania kierownicą. Wartości te zostały dobrane eksperymentalnie stosując metodę Zieglera-Nicolsa. D\_FACTOR\_SUPPRESSION\_PER\_SEC = 0.3; oznacza tłumienie sumy członu różniczkującego o 30% na sekundę – dokładne działanie tego członu zostało opisane   
w poprzednim rozdziale.

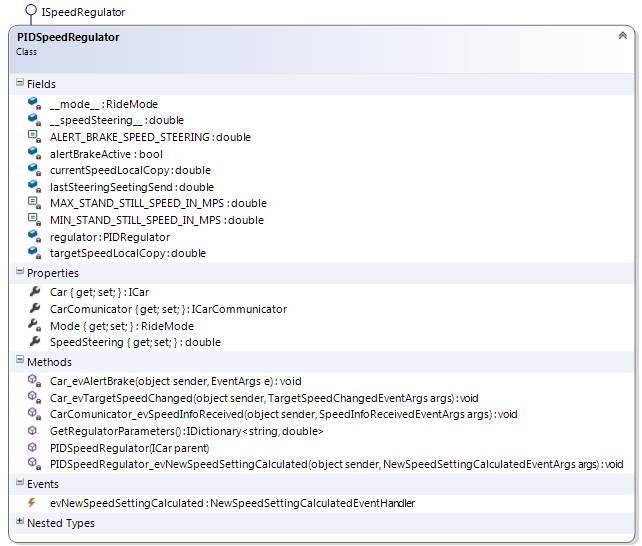
### Regulator prędkości

Abstrakcja regulatora prędkości jest zdefiniowana jako interfejs ISpeedRegulator.



Schemat Diagram interfejsu ISpeedRegulator

Implementacja tego interfejsu poprzez regulator PI widoczna jest w klasie PIDSpeedRegulator.

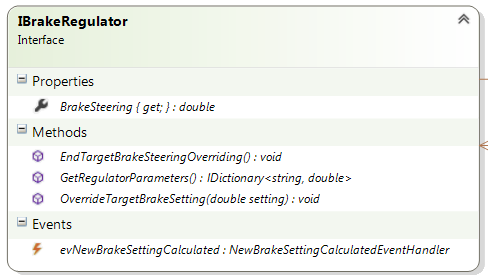


Schemat Diagram klasy PIDSpeedRegulator

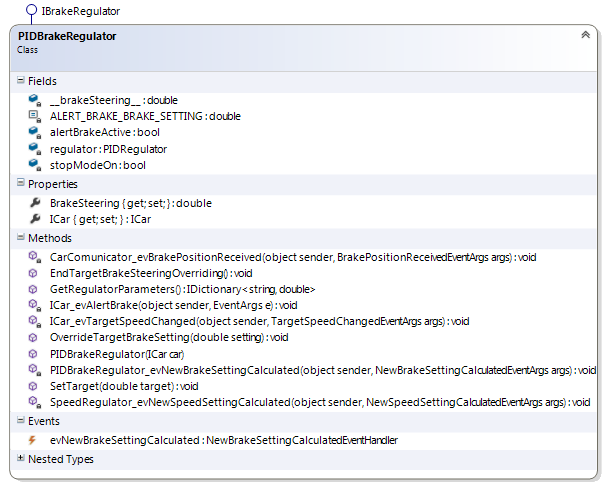
Poza regulacją prędkości regulator ten zarządza też zmianami biegów. Poniżej pewnej prędkości zadanej samochodu zmienia bieg na neutralny, żeby silnik nie przeszkadzał   
w wyhamowywaniu pojazdu, a hamulec nie zatrzymał silnika. Sterowanie biegami   
w samochodzie tego typu nie jest skomplikowanym zadaniem, ponieważ jest   
on wyposażony w automatyczną skrzynię biegów.

### Regulator położenia pedału hamulca

Ze względu na potrzebę możliwości manualnego hamowania samochodu nie mogliśmy zastosować serwomechanizmu do wykonania tego zadania. Z powodu zastosowania silnika prądu zmiennego nawijającego linkę hamulcową powstała konieczność regulowania położenia pedału hamulca. Stworzyłem interfejs odpowiedzialny   
za regulację położenia pedału hamulca i jego implementacje za pomocą pokazanego już wcześniej regulatora PID.



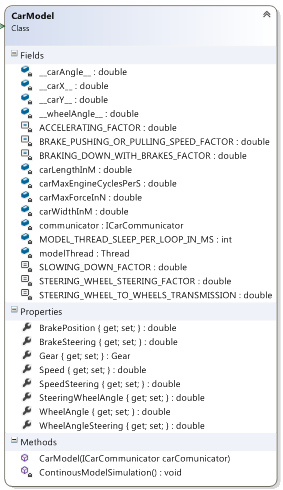
Schemat Diagram interfejsu IBrakeRegulator



Schemat Diagram klasy PIDBrakeRegulator

Poza samym sterowaniem pedałem hamulca implementacja ta pozwala wprowadzić się w stan awaryjnego hamowania, w którym jedyną możliwą akcją jest wciskanie hamulca w taki sposób, żeby był dociśnięty do maksymalnej wartości. Rozwiązanie takie okazuje się bezpieczne, ponieważ nawet w przypadku przegrzania silnika odpowiedzialnego   
za hamowanie ustawia się on w stan „Stop”, co doprowadza do zatrzymania hamulca   
w pozycji zaciśniętej.

## Symulator



Schemat Diagram klasy CarModel

Do użycia fake’owego komunikatora potrzebowałem symulatora samochodu. Pierwsza   
i obecnie używana do testów wersja zakłada bardzo prosty model samochodu. W tym modelu kierownica jest członem całkującym wobec siły przyłożonej do niej, a prędkość członem inercyjnym wobec różnicy stopnia otwarcia przepustnicy i stopnia wciśnięcia pedału hamulca. Oczywiście do modelu dołożone są pewne proste ograniczenia, przez które wciśnięcie hamulca nie może przyspieszać samochodu „do tyłu”, a zmiany biegów powodują zmiany modelu prędkości względem stopnia otwarcia przepustnicy.   
W ostatnim czasie stworzyłem o wiele dokładniejszy model samochodu, który uwzględnia działanie silnika i oddziaływanie samochodu z otoczeniem, jednak system sterujący jest już w tak zaawansowanym stadium, że nie potrzebuję w tej chwili korzystać z symulatora, więc nie zdecydowałem się na połączenie nowego modelu   
z fake’owym komunikatorem.

## Graficzny interfejs użytkownika (GUI)

W celu zapewnienia możliwości kontrolowania parametrów samochodu w trakcie jazdy i sterowania pojazdem stworzyłem prosty interfejs użytkownika w technologii Windows Presentation Foundation (WPF). W oknie, poza obecnym stanem pojazdu, widoczne są też wartości zadane dla każdego z parametrów oraz obecne sterowanie, jakie wysyłane jest do efektorów.



Schemat Interfejs użytkownika systemu sterowania samochodem

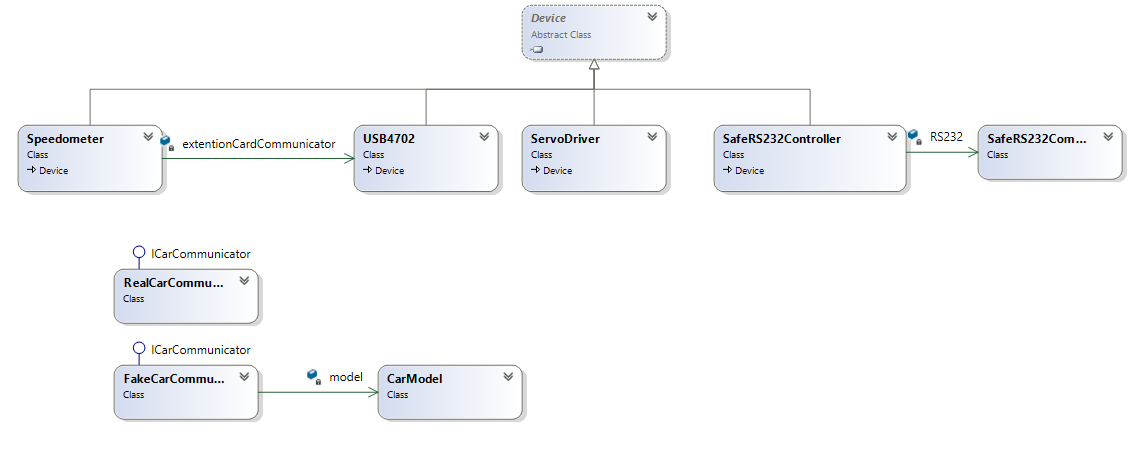
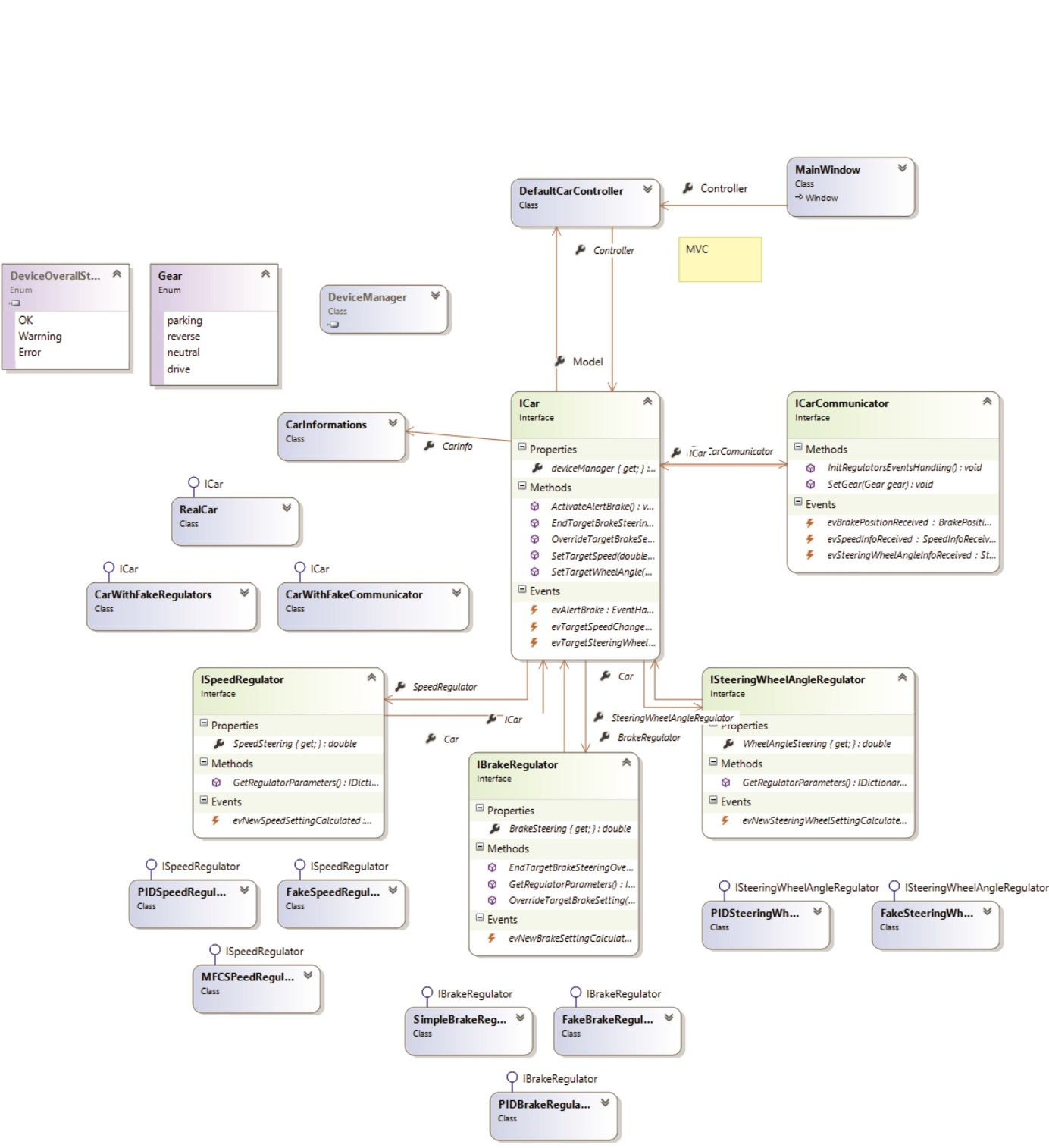
Interfejs użytkownika pozwala też na sterowanie pojazdem za pomocą strzałek. Jazda za pomocą mojego programu różni się znacznie od sterowania samochodem w grach komputerowych, ponieważ sterowanie zmienia tutaj wartości zadane prędkości i skrętu kół zamiast wpływać bezpośrednio na przyspieszenie pojazdu i położenie kierownicy. Takie sterowanie okazuje się dosyć intuicyjne – niejednokrotnie pozwalałem nawet kilkunastoletnim dzieciom na prowadzenie pojazdu.

## Sterowanie w przypadku awarii (to z managera urządzeń)

Manager urządzeń pozwala na prostą implementację awaryjnego działania każdego  
 z urządzeń. Kiedy dowolne z urządzeń koniecznych do poprawnego funkcjonowania pojazdu zgłosi awarię, na każdym z urządzeń zarejestrowanych pod managerem zadań wywoływana jest metoda EmergencyStop. Implementacja konkretnych akcji na każdym z urządzeń okazała się być bardzo prosta - wystarczyło zdefiniować zadania urządzeń korzystając z wystawionych przez nie publicznych metod.  
W przypadku jakiejkolwiek awarii najważniejsze jest zatrzymanie samochodu. Biorąc pod uwagę sytuację, w której to właśnie układ hamulcowy uległ awarii zdefiniowałem reakcję układu kierowniczego na wyłączenie sterowania, a reakcję obsługi skrzyni biegów na ustawienie biegu neutralnego (samochód po krótkim czasie sam się zatrzyma).

## Diagram klas

Implementacja najważniejszej części kodu opiera się na 5 interfejsach, które zostały zaimplementowane w kodzie na wiele sposobów. Powstała przez to duża ilość klas, co pozornie pokazuje duży stopień skomplikowania kodu – jednak warto pamiętać, że podczas działania programu tylko 1 wybrana implementacja każdego interfejsu jest używana.



Schemat Pełny diagram klas systemu sterowania samochodem

# Zakończenie

Moja praca nad tworzeniem, rozwijaniem i utrzymaniem systemu sterowania samochodem autonomicznym trwa nieprzerwanie od ponad 1.5 roku. W tym czasie udało mi się stworzyć stabilny system, który potrafi regulować procesy związane ze sterowaniem. Zastosowane przeze mnie regulatory okazują się szybko reagować na wymuszenia i działać stabilnie nawet w przypadku dużych zmian otoczenia (jazdy na wzniesieniach, w nierównym terenie, złej nawierzchni). Komunikatory, do których zostało oddelegowane sterowanie sygnałami elektronicznymi potrafią reagować na błędy komunikacji próbując naprawić połączenie, a w przypadku awarii zgłosić błąd do managera urządzeń, który potrafi w prosty sposób zarządzać błędami. Najsłabszą częścią projektu wydaje się graficzny interfejs użytkownika, który nie zachwyca elastycznością, ani wyglądem. Celem mojego projektu nie było jednak tworzenie oprogramowania, za pomocą którego człowiek będzie mógł sterować samochodem. Integracja mojego systemu z systemem przetwarzania obrazów pokazała, że wystawiony interfejs sterowania samochodem pozwala na proste sterowania procesami samochodu przez systemy nadrzędne. System sterowania udało mi się doprowadzić do stanu,  
w którym nie wymaga on większych modyfikacji, kiedy urządzenia pokładowe samochodu nie ulegają zmianom. Z tego powodu w obecnej fazie rozwoju projektu skupiam się na wspomaganiu zespołów pracujących nad elementami sztucznej inteligencji samochodu.

Od ponad 8 miesięcy samochód był pokazywany na wielu targach, w programach telewizyjnych („Teleexpres”, „Fakty”, „Dzień dobry TVN”) i internetowych serwisach informacyjnych (onet.pl, wp.pl). Ilość systemów zamontowanych w samochodzie zawsze wzbudza zainteresowanie gapiów, a to właśnie system sterowania, który jest opisany w tej pracy, zarządza wszystkimi urządzeniami zamontowanymi w samochodzie. Z powodu ilości zadań, które musi spełniać system sterowania wydaje się, że powinien on mieć wysoki stopień komplikacji, co zawsze przeszkadza w rozwoju projektów informatycznych. Wydaje się jednak, że z powodu zastosowanej architektury   
i wykorzystania systemu sygnałów i slotów udało mi się stworzyć system, który jest prosty w utrzymaniu i modyfikacji.

# Bibliografia

Andrew Troelsen, “Pro C# 5.0 and the .NET 4.5 Framework”, ISBN13: 978-1-4302-4233-8

Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein, “Introduction to Algorithms”, ISBN-13: 978-0262033848

Andrew Hunt, David Thomas: Pragmatyczny programista. Od czeladnika do mistrza, ISBN 83-204-2672-3

Ben-Ari, M., "Principles of Concurrent and Distributed Programming", ISBN 0-13-711821-X

Buehler, M., Iagnemma, K., and Singh, S., “The 2005 DARPA Grand Challenge: The Great Robot Race”, ISBN 978-3-540-73429-1

Stanford Artificial Intelligence Lab, Stanford University, “The Stanford Entry in the Urban Challenge”

Sebastian Thrun, Mike Montemerlo, Hendrik Dahlkamp, “Stanley: The Robot that Won the DARPA Grand Challenge”