Politechnika Śląska

Wydział Mechaniczny Technologiczny

Rozprawa doktorska

Opracowanie metodyki tworzenia zautomatyzowanych testów systemów wbudowanych w pojazdach samochodowych, umożliwiającej weryfikację poprawności wyników z wykorzystaniem algorytmów rozszerzonej inteligencji

Autor: mgr inż. Anna Gnacy-Gajdzik
Promotor: dr hab. inż. Piotr Przystałka, prof. PŚ
Opiekun ze strony przedsiębiorcy: dr inż. Wojciech Sebzda

Gliwice, wrzesień 2024

"Człowiek, który nigdy nie popełnił błędu,
nigdy nie próbował niczego nowego."
Albert Einstein

Pracę tą dedykuję Małgosi, Karolowi i Jasiowi

Spis treści

Spis rysunków

Spis tabel

Podziękowania

Szczególne podziękowania kieruję do promotora mojej rozprawy Pana Profesora Piotra Przystałki za cenne uwagi, przekazaną wiedzę i poświęcony czas. Dziękuję za ciągłe udoskonalanie wyników badań i osiągnięte sukcesy.

Wyrażam wdzięczność firmie DRÄXLMAIER za wsparcie, a w szczególności dr inż. Wojciechowi Sebzda, mojemu opiekunowi z ramienia firmy za jego nieocenione wsparcie i profesjonalizm. Chciałabym też serdecznie podziękować dr inż. Markowi Sznura za okazane zaufanie, wsparcie i motywację do podjęcia się tego wyzwania.

Mężowi i Dzieciom dziękuję za cierpliwość, wyrozumiałość i okazane wsparcie.

Studia doktoranckie były finansowane ze środków Ministerstwa Nauki i Szkolnictwa Wyższego w ramach umowy nr DWD/4/55/2020. Badania opisane w rozprawie doktorskiej zostały zrealizowane we współpracy z firmą DRÄXLMAIER.

Wykaz skrótów i oznaczeń

Wykaz wybranych skrótów i akronimów

ABS	Anti-lock braking system – System przeciwblokujący hamulce
ADAS	Advanced Driver Assistance System – Zaawansowany System Wspomagania Kierowcy
AE	Autoencoder – Autoenkoder
AI	Artificial Intelligence – Sztuczna Inteligencja
ALAD	Adversial Learning for Time Series – Algorytm oparty o uczenie antagonistyczne dla szeregów czasowych
ASCII	American Standard Code for Information Interchange – Amerykański Standardowy Kod do Wymiany Informacji
ASPICE	Automotive Software Process Improvement and Capability Determination – Standard oceny procesów wytwarzania oprogramowania w branży motoryzacyjnej
ASIL	Automotive Safety Integrity Level – Poziom integralności bezpieczeństwa samochodowego
AUC	Receiver Operating Characteristic Area Under the Curve – Miara używana do oceny skuteczności modeli klasyfikacyjnych
AuI	Augmented Intelligence – Rozszerzona Inteligencja
BA	Balanced Accuracy – Zrównoważona dokładność
BCU	Battery Control Unit – Układ sterowania baterią
BEV	Battery Electric Vehicle – Pojazd elektryczny
BMS	Battery Management Systems – System Zarządzania Akumulatorem
CAN	Controller Area Network – Szeregowa magistrala komunikacyjna stosowana w przemyśle samochodowym
CAPL	Communication Access Programming Language – Język programowania dostępu do komunikacji
CatB+	Unbiased Categorical Boosting with categorical features – Algorytm oparty o wzmacnianie gradientowe wykorzystujące drzewa binarne
CI	Continuous Integration – Mechanizm ciągłej integracji
CI/CD	Continuous integration and continuous deployment – Mechanizm ciągłej integracji i wdrażania
CV	Combustion Vehicle – Pojazd spalinowy
dIUS	Czujniki prądu i napięcia DRÄXLMAIER
DLL	Dynamic Link Library – Dynamiczna Biblioteka Łączona
DTC	Decision Tree Classifier – Klasyfikator drzew losowych
DNN	Deep Neural Network – Głęboka Sieć Neuronowa
ECOD	Empirical cumulative distribution functions for outlier detection

	– Algorytm oparty o empiryczne funkcje dystrybuanty rozkładu skumulowanego do wykrywania wartości odstających
ECM	Electronic Control Module – Elektroniczny Moduł Sterujący
ECU	Electronic Control Unit – Elektroniczna Jednostka Sterująca
EFB	Exclusive Feature Bundling – Wyłączające się wiązanie cech
ERG	Exhaust Gas Recirculation – Recyrkulacja Spalin
ESP	Electronic Stability Program – Elektroniczny Program Stabilizacji
FCEV	Fuel Cell Electric Vehicle – Pojazd zasilany wodorem
Fn	False Negative - Fałszywy negatyw
Fp	False Positive – Fałszywy pozytyw
GAN	Generative Adversarial Networks – Generatywne sieci przeciwstawne
GOSS	Gradient-based One-Side Sampling – Próbkowanie jednostronne oparte na gradiencie
GRU	Gated Recurrent Unit – Jednostka rekurencyjna z bramkami
HEV	Hybrid Electric Vehicle – Hybrydowy Pojazd Elektryczny
HI	Human Inteliigence – Ludzka Inteligencja
HIL	Hardware in the Loop – Sprzęt w pętli
HTML	HyperText Markup Language – Hipertekstowy język znaczników
IEEE	Institute of Electrical and Electronics Engineers – Instytut Inżynierów Elektryków i Elektroników
IForest	Isolation Forest – Algorytm oparty o las izolacyjny
INNE	Nearest Neighbor Ensemble – Algorytm bazujący na mechanizmie izolacji, opartym na metodzie najbliższych sąsiadów
IoT	Internet of Things – Internet Rzeczy
ISO	International Organization for Standardization – Międzynarodowa Organizacja Normalizacyjna
KNN	k–nearest neighbours – Algorytm oparty o k– najbliższych sąsiadów
LGB	LightGBM Light Gradient Boosting – Algorytm boostingu zoptymalizowany pod kątem dużych zbiorów danych
LIN	Local Interconnect Network – Lokalna Sieć Komunikacyjna
LSTM	Long Short-Term Memory – Długotrwała Pamięć Krótkoterminowa
MBE	Model-Based Engineering – Inżynieria oparta na modelach
MCAS	Maneuvering Characteristics Augmentation System – System Ulepszania Charakterystyk Manewrowych
MCC	Matthews Correlation Coefficient – Współczynnik korelacji Matthewsa
MIDS	Measurement Intrusion Detection System – System wykrywania włamań pomiarowych
MIL	Model in the Loop – Model w pętli
ML	Machine Learning – Uczenie maszynowe
MLP	Multilayer Perceptron – Algorytm oparty o perceptron wielowarstwowy
MSE	Mean Squared Error – Średni Błąd Kwadratowy
NLP	Natural Language Processing – Przetwarzania języka naturalnego
NM	Network Management – Zarządzanie Siecią
ODDS	Outlier Detection Datasets – Zestawy danych do wykrywania wartości odstających

OEM	Original Equipment Manufacturer – Producent Oryginalnego Wyposażenia
PCA	Principal Component Analysis – Algorytm oparty o analizę głównych składowych
PCB	Printed Circuit Board – Płytka Obwodu Drukowanego
PHEV	Plug-in Hybrid Electric Vehicle – Hybrydowy Pojazd Elektryczny typu Plug-in
PIL	Processor in the Loop – Procesor w pętli
PLM	Project Lifecycle Management – Zarządzanie Cyklem Życia Projektu
PRE	Precyzja - Jedna z miar skuteczności algorytmów klasyfikacyjnych
PSM	Permanent magnet synchronous – Silnik synchronicznych wzbudzanych magnesami trwałymi
PyOD	Python Outlier Detection – Wykrywanie wartości odstających w Pythonie
RAM	Random Access Memory – Pamięć o Dostępie Swobodnym
RF	Random Forest – Las losowy
RMSE	Root Mean Squared Error – Pierwiastek Średniokwadratowego Błędu
RNN	Recurrent Neural Network – Rekurencyjna sieć neuronowa
ROC AUC	Area Under the Curve of the Receiver Operating Characteristic – Pole pod krzywą charakterystyki operacyjnej odbiornika
QM	Quality Management – Zarządzanie Jakością
SAE	Society of Automotive Engineers – Stowarzyszenie Inżynierów Motoryzacji
SBW	Steer-by-Wire – System Sterowania Elektronicznego (bez mechanicznego połączenia kierownicy z kołami)
SCR	Selective Catalytic Reduction – Selektywna Redukcja Katalityczna (technologia redukcji emisji spalin)
SIL	Software in the Loop – Oprogramowanie w pętli
SMOTE	Synthetic Minority Oversampling Technique – Syntetyczna technika nadpróbkowania mniejszości
SOC	State of Charge – Stan naładowania baterii
SOH	State of Health – Stan zdrowia baterii
SOL	State of Life – Przewidywana żywotność baterii
SPI	Serial Peripheral Interface – Interfejs szeregowy
STLC	Software Testing Life Cycle – Cykl życia testowania oprogramowania
SVM	Support Vector Machine – Algorytm oparty o maszynę wektorów nośnych
Tp	True Positive – Prawdziwy pozytyw
Tn	True Negative – Prawdziwy negatyw
UML	Unified Modeling Language – Zunifikowany język modelowania
XCP	Universal Measurement and Calibration Protocol – Uniwersalny Protokół Pomiaru i Kalibracji
XGB+	Scalable end-to-end tree boosting system – Algorytm oparty o skalowalny system wzmacniania drzew od początku do końca

TBP	Total Body Photography – Fotografia całego ciała
VCU	Vehicle Control Unit – Jednostka sterująca pojazdem
VIL	Vehicle in the Loop – Pojazd w pętli

Wykaz wybranych oznaczeń

A – powierzchnia czołowa pojazdu
α – kąt nachylenia terenu
C_d – współczynnik oporów powietrza
d_w – średnica koła
ΔM – moment wprawiający w ruch przekładni
e_n – siła elektromotoryczna indukowana w uzwojeniu n-tej fazy
E – siła elektromotoryczna indukcji
ε_M – częstość migotania testu
F_T – siła trakcyjna pojazdu
F_{DF} – siła związana z oporem aerodynamicznym (siła oporu powietrza)
f_r – współczynnik oporu toczenia
F_r – siła oporu toczenia
F_g – siła oporu wzniesienia
g – przyspieszenie ziemskie
i – przełożenie przekładni mechanicznej
i_n – natężenie prądu n-tej fazy
i_z – natężenie prądu płynącego przez uzwojenia wirnika
I_p – natężenia prądów cewki przekaźnika
k_m – stała mechaniczna silnika
K_e – stała elektryczna silnika
L_{neg} – liczba negatywnych rezultatów
L_n – indukcyjność zastępcza fazy uwzględniająca indukcyjność własną i wzajemną
L_t – liczba wykonanych testów
L_p – liczba pozytywnych rezultatów
m – masa pojazdu
M_e – moment obrotowy silnika
p – ciśnienie
r – indywidualna stała gazowa dla powietrza
R_n – rezystancja fazy stojana
ρ – gęstość powietrza
σ – funkcja sigmoidalna
tanh – funkcja tangens hiperboliczny
T – temperatura w skali bezwzględnej
u_n – napięcie prądu n-tej fazy
U_p – napięcia sterujące przekaźnikami
v – prędkość pojazdu
v_w – prędkość wiatru

ω_m – to prędkość obrotowa silnika
$\odot$ – iloczyn skalarny

1. Wstęp

Na świecie wykorzystywanych jest około miliard samochodów, co sprawia, że przemysł motoryzacyjny odgrywa znaczącą rolę w generowaniu emisji dwutlenku węgla oraz innych negatywnych wpływów na środowisko [1]. Proces dekarbonizacji transportu ukierunkowany jest na tworzenie czystszych i bardziej zrównoważonych alternatyw transportowych [1]. Koncerny samochodowe transformują w kierunku eMobilności (ang. eMobility) [2], co skutkuje tym, że systemy wbudowane w pojazdach (ang. embedded systems), ich liczba oraz złożoność realizowanych przez nie funkcji, ciągle rośnie [3]. Zmiana charakteru systemów wbudowanych z zapewniających rozrywkę i podnoszących wygodę oraz komfort pasażerów w kierunku wspomagania układów mechanicznych pojazdu, oraz zapewnienia bezpieczeństwa kierowcy, przyczyniła się do rozwoju tej gałęzi przemysłu samochodowego. Najważniejszymi trendami, które obecnie kształtują rynek nowo projektowanych samochodów są mobilność elektryczna [4], digitalizacja [5] i pojazdy autonomiczne [6].

Kluczowym zadaniem w procesie projektowania i produkcji samochodów jest zapewnienie bezpieczeństwa, jakości i niezawodności działania zarówno poszczególnych komponentów, jak i całego systemu [7]. Zapewnienie bezpieczeństwa funkcjonalnego w branży motoryzacyjnej napotyka liczne wyzwania, zaczynając od złożoności architektury systemów, przez ciągłe aktualizacje międzynarodowych standardów (ISO26262 [8], ASPICE [9] itd.), po konieczność wprowadzania innowacyjnych rozwiązań, aby nadążać za konkurentami [10]. Testowanie systemów wbudowanych, które realizują zadania inżynierii mechanicznej, stanowi ogromne wyzwanie w porównaniu do testowania aplikacji desktopowych. Wymaga to spełnienia znacznie wyższych standardów niezawodności, które są absolutnie kluczowe dla tego rodzaju oprogramowania. Dodatkowo pewne aspekty, takie jak obsługa przerwań w kontekście przetwarzania równoległego, mogą być testowane tylko w czasie rzeczywistym. Aby sprostać tym wyzwaniom, producenci systemów wbudowanych kładą duży nacisk na automatyzację procesów testowych, co pozwala na zredukowanie kosztów i czasu potrzebnego na przeprowadzenie testów [11].

Rozwój sztucznej inteligencji (ang. artificial intelligence) i jej coraz śmielsze zastosowania we wszystkich dziedzinach przemysłu spowodował, że również przemysł samochodowy jest zainteresowany wprowadzeniem innowacji oraz integracji przełomowych technologii w swoich produktach [12]. Systemy wbudowane oparte na sztucznej inteligencji są zazwyczaj opisywane jako otwierające nowe ścieżki do eksploracji wiedzy, jednak producenci komponentów samochodowych muszą stawić czoła wyzwaniom związanym z włączeniem tych innowacyjnych technologii do swoich dobrze zoptymalizowanych procesów tworzenia produktów [13].

W Europie, jeśli producent samochodów chce wprowadzić swój produkt na rynek, musi on spełnić wymagania Europejskiej Komisji Gospodarczej ONZ w ramach tzw. procesu homologacji [14]. Obejmuje on m.in. zgodność produkcji, w której producent zaświadcza, że każdy pojazd, wyposażenie lub części wprowadzone na rynek zostały wyprodukowane tak, aby były identyczne z homologowanym produktem. W związku z tym wszystkie komponenty pojazdu muszą być przetestowane i zatwierdzone zarówno pod względem bezpieczeństwa, jak i jakości zgodnie z obowiązującymi standardami [15]. Proces homologacji jest przeprowadzany przez niezależną stronę trzecią (np. TÜV, Dekra), a homologacja jest udzielana przez organy rządowe [16]. Ogromnym wyzwaniem dla producentów pojazdów jest w szczególności udana integracja nowych aspektów, takich jak bezpieczna autonomiczna jazda bez ludzkiego kierowcy w pętli sterowania [17], [18], cyberbezpieczeństwo motoryzacyjne [19], [20], [21] i sterowanie systemami oparte na sztucznej

inteligencji [22], z obowiązującymi przepisami.

Proces homologacyjny pojazdu wymaga, aby narzędzia i oprogramowanie używane w czasie rozwoju, walidacji i weryfikacji jego komponentów (np. systemów wbudowanych) spełniały obowiązujące normy i przepisy, w związku z tym powinny przejść testy zgodności z nimi i posiadać odpowiednie certyfikaty. Fakt ten oraz to, że brakuje przepisów prawnych regulujących zastosowanie sztucznej inteligencji w dziedzinach przemysłu oddziałujących na zdrowie czy życie ludzkie powoduje lukę na rynku narzędzi dla dostawców systemów wbudowanych.

Testowanie tego, co zostało zaprojektowane, zaimplementowane czy wyprodukowane jest niezbędne na wszystkich wymienionych etapach tworzenia produktu, aby jego eksploatacja była bezpieczna. Co za tym idzie, testowanie oprogramowania jest kluczowym czynnikiem w zwiększaniu wiarygodności systemów wbudowanych, szczególnie jeśli dotyczy bezpieczeństwa, zdrowia i życia ludzkiego. Wzrastająca złożoność projektowanych systemów wbudowanych i coraz wyższe standardy dotyczące jakości oraz niezawodności, często powiązane z ograniczonym czasem na wprowadzenie produktu na rynek, wymuszają stosowanie efektywnych metod rozwoju oprogramowania w systemach motoryzacyjnych [23]. Pomimo że w budżetach projektów środki finansowe i zasoby ludzkie przeznaczone na testowanie stanowią coraz większy procent, proces ten jest zawsze kompromisem między budżetem, czasem i jakością. Redukcja kosztów testowania systemów wbudowanych może skutkować poważnymi awariami, a nawet katastrofami, szczególnie w przypadku systemów o znaczeniu krytycznym, takich jak stosowane w lotnictwie, motoryzacji czy przemyśle energetycznym. Poniżej przedstawiono przykłady, które ukazują, jak poważne mogą być szkody wynikające z ograniczenia procesu testowania:

- Katastrofa rakiety Ariane 5 (1996), która eksplodowała 37 sekund po starcie. Przyczyną była awaria oprogramowania wbudowanego odpowiadającego za określenie pozycji rakiety. Błąd wynikał z niewłaściwej konwersji liczby zmiennoprzecinkowej na liczbę całkowitą, co spowodowało przepełnienie bufora. Brak odpowiednich testów oprogramowania doprowadził do strat wynoszących około 400 milionów dolarów [24].
- Wypadek śmiertelny Tesli Model S z włączonym autopilotem (2016) spowodowany był niewystarczającym testowaniem w warunkach rzeczywistych. System wbudowany nie rozpoznał białej naczepy ciężarówki, ponieważ czujniki i algorytmy nie były w stanie prawidłowo zidentyfikować przeszkody na tle jasnego nieba, czego skutkiem był brak reakcji - nie zostało włączone automatyczne hamowanie [25].
- Awaria oprogramowania Boeinga 737 MAX (2018), która doprowadziła do dwóch tragicznych wypadków samolotów Boeing 737 MAX (Lion Air i Ethiopian Airlines). Zostały one spowodowane przez wadliwe działanie systemu MCAS (ang. Maneuvering Characteristics Augmentation System). Błędy w projekcie i brak odpowiednich testów oprogramowania oraz systemów redundancji przyczyniły się do śmierci 346 osób. Minimalizacja kosztów związana z ograniczonymi testami bezpieczeństwa systemów wbudowanych i dążenie do szybszego wprowadzenia produktu na rynek były częścią problemu [26].
- Wypadki związane z systemami ABS oraz ESP w pojazdach - wadliwe działanie oprogramowania ABS było powodem licznych awarii w systemach hamowania w niektórych modelach Forda (m.in. Explorer) oraz Toyoty (m.in. Yaris), co doprowadziło do licznych wypadków i kosztownych akcji serwisowych [27].

Zgodnie ze wspomnianymi wcześniej standardami (ISO 26262 oraz ASPICE), testowanie oprogramowania wbudowanego odbywa się na trzech poziomach: testów jednostkowych, testów integracyjnych, testów kwalifikacyjnych. W dwóch pierwszych przypadkach inżynier jakości oprogramowania (często nazywany również testerem) projektuje przypadki testowe oraz implementuje automatyczne skrypty, wykonywane w środowiskach testowych Software in the Loop oraz Processor in the Loop. Włączenie gotowych skryptów testowych, w praktykę ciągłej integracji (ang. Continuous Integration – CI), pozwala na automatyczne wykonywanie testów

regresyjnych po każdym uaktualnieniu plików w repozytorium przez inżyniera systemów wbudowanych. W przypadku negatywnego wyniku serwer ciągłej integracji automatycznie przekazuje informacje o niepowodzeniu testu.

Z kolei testy kwalifikacyjne zgodnie z normą ISO26262 powinny być wykonywane w środowisku docelowym, czyli pojeździe samochodowym. Ze względu na koszty, problemy z fizycznym dostępem do badanego systemu wbudowanego czy symulacją błędów, jest to niezwykle trudne środowisko testowe. Zadaniem zespołu inżynierów jakości oprogramowania wbudowanego jest wykrycie wszystkich defektów oprogramowania podczas przeglądów wymagań, analizy kodu oraz przeprowadzania sesji testowych [28]. Norma dopuszcza wykonywanie testów kwalifikacyjnych w środowisku Hardware in the Loop, gdzie tylko pewne elementy środowiska są rzeczywiste, natomiast pozostała część pojazdu jest symulowana [29]. Taka możliwość niesie ze sobą konkretne niebezpieczeństwo — producenci systemów wbudowanych, aby zmniejszyć koszty oraz czas potrzebny na przygotowanie i utrzymywanie środowiska testowego, dążą do jego maksymalnego uproszczenia. Skutkiem takich działań jest to, że podczas testowania takich systemów wbudowanych jak na przykład sterownik baterii wysokonapięciowej, nie są uwzględniane zjawiska fizyczne oddziałujące na pojazd oraz mechanika pojazdu i z tego powodu część defektów oprogramowania może nie zostać wykryta. Sytuacja, gdy defekt zostanie zidentyfikowany dopiero przez klienta, jest niekorzystna, nie tylko ze względów finansowych. Wydłuża ona proces rozwoju systemu wbudowanego i powoduje opóźnienie kolejnych wydań oprogramowania, przede wszystkim ze względu na konieczność reprodukcji defektu w innym środowisku testowym niż został on znaleziony. W przypadku defektów zidentyfikowanych w docelowym pojeździe często jest to wręcz niemożliwe do zrobienia, ponieważ zespół projektowy nie dysponuje pojazdem testowym [30]. Wyzwaniem na drodze do pełnej automatyzacji testów kwalifikacyjnych są anomalie pojawiające się w środowisku testowym złożonym z rzeczywistych elementów mechatronicznych, które zaburzają wyniki testów, powodując zjawisko migotania rezultatów (ang. flickering results) [31]. Pojawiające się wyniki negatywne, niezależne od testowanej funkcjonalności oprogramowania, prowadzą do wydłużenia czasu potrzebnego na testowanie oraz zwiększenia kosztów, a także podważają wiarygodność całego procesu.

W związku z tym, zastosowanie algorytmów rozszerzonej inteligencji (ang. augmented intelligence) może oferować znaczące usprawnienia w procesach weryfikacji poprzez umożliwienie głębszej analizy i interpretacji danych środowiskowych, co w konsekwencji może przyczynić się do zwiększenia precyzji i wiarygodności wyników [32]. Można oczekiwać, że automatyzacja procesu testowania systemów wbudowanych wspierana algorytmami rozszerzonej inteligencji pozwoli na optymalizację całego procesu, a przede wszystkim poprawę jakości oprogramowania wbudowanego. W dobie rozwoju motoryzacji w kierunku eMobilności i autonomii jazdy jest to bardzo istotne z punktu widzenia bezpieczeństwa życia ludzkiego [13]. Chociaż algorytmy sztucznej inteligencji są szeroko stosowane w wielu dziedzinach [33], [34], ich zastosowanie w kontekście zautomatyzowanych testów systemów wbudowanych w pojazdach samochodowych pozostaje stosunkowo mało zbadane [35]. Skuteczna integracja sztucznej inteligencji z procesem testowania może zwiększyć niezawodność i efektywność procesów testowych, jak również zapewnić bardziej adaptacyjne i skalowalne podejścia do testowania [36]. Synergia sztucznej inteligencji z ludzką pozwala na wykorzystanie możliwości szybkiego przetwarzania ogromnych ilości danych, głębokiego uczenia maszynowego z zastrzeżeniem, że to człowiek jest odpowiedzialny za cały proces i jego wyniki. Zautomatyzowane testy są niezbędne do oceny funkcjonalności, zapewnienia niezawodności i bezpiecznego działania systemów wbudowanych w pojazdach samochodowych, jednak tradycyjne metody testowania wydają się nie być wystarczająco efektywne ani adekwatne do nowych wyzwań technologicznych, jakie niesie zaawansowana elektronika i oprogramowanie [37].

W obliczu wyzwań stojących przed producentami branży motoryzacyjnej istotne jest opracowanie nowych metodyk testowania rozproszonych funkcji oprogramowania, aby osiągnąć odpowiednią wydajność na etapie integracji, zgodnie z wymaganiami specyfikacji. W praktyce

motoryzacyjnej rozwijane są różnorodne strategie zarządzania wielopoziomowym testowaniem oprogramowania, które mają na celu obniżenie kosztów oraz zwiększenie efektywności i odporności systemów. Jednakże nadal istnieje potrzeba opracowania metodyk, które umożliwią lepszą strukturyzację, optymalizację oraz planowanie testów, dostosowanych do konkretnych wymogów branży motoryzacyjnej [38]. Aktualizacja, optymalizacja i rozwój metodyk testowania systemów wbudowanych w kierunku zaangażowania metod sztucznej inteligencji bezpośrednio przełoży się na większe bezpieczeństwo i niezawodność pojazdów, co jest kluczowe w kontekście rosnących wymagań regulacyjnych oraz oczekiwań konsumentów, a także umożliwi producentom pojazdów implementację bardziej zaawansowanych i niezawodnych technologii [39].

Można stwierdzić, że tematyka podejmowana w pracy, a więc opracowanie innowacyjnej metodyki testowej umożliwiającej integrację i efektywne wykorzystanie zaawansowanych technologii AI do analizy i weryfikacji testów systemów wbudowanych ma kluczowe znaczenie w kontekście inżynierii mechanicznej, obejmującej budowę i eksploatację nowoczesnych pojazdów samochodowych, które funkcjonują jako złożone mega układy mechatroniczne.

Poniższa rozprawa koncentruje się na badaniu i ocenie zdolności algorytmów rozszerzonej inteligencji do rozpoznawania anomalii występujących w środowisku testowym opartym na metodyce HIL (ang. Hardware in The Loop) i modelu numerycznym pojazdu. Wprowadzenie modelu numerycznego pozwala na symulację rzeczywistych warunków pracy pojazdu, co umożliwia dokładną ocenę wpływu różnych czynników mechanicznych i środowiskowych na testowane systemy wbudowane. Zastosowanie algorytmów AI nie tylko pozwala na precyzyjne wykrywanie nieprawidłowości, ale także optymalizuje cały proces testowania, zwiększając efektywność i niezawodność, co jest kluczowe dla dalszego rozwoju technologii pojazdów samochodowych oraz spełnienia rosnących wymagań bezpieczeństwa i regulacji w branży motoryzacyjnej.

Dodatkowo praca ta przyczynia się do rozwoju inżynierii mechanicznej poprzez wprowadzenie innowacyjnych rozwiązań w zakresie automatyzacji testów systemów mechatronicznych, co wspiera szybsze i bardziej skuteczne wdrażanie nowych technologii w pojazdach, a także rozwija narzędzia, które będą miały zastosowanie w przyszłych generacjach pojazdów, takich jak samochody autonomiczne.

1.1 Problem badawczy

Testowanie oprogramowania wbudowanego polega między innymi na badaniu reakcji systemu na negatywne sytuacje, które testowany system powinien poprawnie obsłużyć zgodnie z określonymi wymaganiami [40]. Średniej wielkości system wbudowany jest opisywany przez około 10000 wymagań, dla których należy przygotować przypadki testowe. Testy powstają w miarę rozwoju produktu. Należy podkreślić, że przed wydaniem każdej wersji oprogramowania powinny zostać przetestowane nie tylko nowo zaimplementowane funkcjonalności, ale również przeprowadzone testy regresji [41]. Oznacza to wielokrotne wykonywanie setek przypadków testowych. Priorytetem dla zespołów testerskich jest automatyzacja testów, głównie z powodu możliwości skrócenia czasu ich przeprowadzania i związanej z tym redukcji kosztów. Ważna jest także łatwość wykonania złożonych scenariuszy testowych, które obejmują sprawdzanie zależności czasowych [42]. Wyeliminowanie błędów ludzkich takich jak pomyłki, przeoczenia czy zmęczenie przyczynia się do zwiększenia dokładności i niezawodności testów. Jednakże jednym z problemów związanych z automatyzacją testów jest pojawienie się tak zwanych testów migoczących (ang. flickering tests). Testy zakończone wynikiem negatywnym powinny skutkować zgłoszeniem defektu. Natomiast testy migoczące to testy, które bez widocznego powodu dają zmienny, czasami pozytywny a innym razem negatywny wynik, nawet jeżeli uruchamiane są na tej samej wersji oprogramowania, sprzętu i w tym samym (na pierwszy rzut oka) środowisku testowym [31]. Jeżeli w czasie wykonywania procedury testowej w testowanym systemie wystąpi błąd, czyli niewłaściwe działanie człowieka lub środowiska, w którym działa system, może to prowadzić

do negatywnego wyniku testu pomimo prawidłowej reakcji, właściwego funkcjonowania testowanego systemu wbudowanego. Migotanie rezultatów testów jest zjawiskiem niepożądanym, które podważa wiarygodność procesu testowania. Występowanie testów migoczących obserwuje się nie tylko w przypadku układów wbudowanych, ale tu są one szczególnie trudne do wyeliminowania. Jest to sytuacja wymagająca gruntownej analizy, zwłaszcza że przed oficjalnym wykonaniem każdy test jest poddawany formalnym przeglądom na etapach specyfikacji i implementacji, sam skrypt testowy nie powinien zawierać błędów [9]. Pojawienie się niestabilnych wyników testów zwiększa koszty procesu testowania, gdyż każdy taki przypadek wymaga dogłębnego zbadania i diagnozy przyczyn migotania. Przykładowo, jeśli 10% wykonywanych testów uzyska rezultaty migoczące, w przypadku średniego systemu wbudowanego oznacza to ok. 50 raportów z wykonania testów do przeanalizowania, powtórnego wykonania, zgromadzenia logów i ich analizy. Prowadzi to do opóźnień w harmonogramie projektu i generuje dodatkowe koszty. Dlatego też eliminacja niestabilnych wyników testów staje się priorytetem, aby ograniczyć wydatki i zwiększyć efektywność oraz niezawodność procesu testowania systemów wbudowanych [43].

Podsumowując genezę pracy, można zauważyć, że zidentyfikowany problem technologiczny dotyczy migotania rezultatów testów systemów wbudowanych, szczególnie w przemyśle motoryzacyjnym. Migotanie rezultatów testów, czyli niestabilność wyników uzyskiwanych w identycznych warunkach, prowadzi do zwiększonych kosztów, wydłużenia czasu testowania i obniżenia zaufania do wyników. W praktyce wymusza to wielokrotne powtarzanie testów, analizę logów oraz diagnozowanie przyczyn błędów, co znacząco opóźnia wprowadzanie nowych systemów wbudowanych na rynek. Jest to kluczowe wyzwanie technologiczne, które wpływa na proces testowania oprogramowania wbudowanego i wymaga skutecznego rozwiązania.

Problem naukowy wynikający z tej sytuacji dotyczy identyfikacji przyczyn oraz opracowania skutecznych metod eliminacji migotania rezultatów testów. Główne pytania badawcze obejmują:

- Jakie są kluczowe przyczyny migotania rezultatów testów systemów wbudowanych? Czy wynikają one z ograniczeń technicznych, zakłóceń środowiskowych, czy złożoności układów mechatronicznych?
- Jakie algorytmy rozszerzonej inteligencji oraz techniki testowe mogą być zastosowane do identyfikacji i eliminacji zjawiska migotania rezultatów? Jak można wykorzystać te narzędzia do zwiększenia stabilności i niezawodności procesu testowania?
- Jak można wykorzystać model numeryczny pojazdu samochodowego w procesie testowania systemów wbudowanych? Jakie są kluczowe aspekty integracji modelu numerycznego pojazdu z procesem testowym, aby poprawić dokładność symulacji i wyników testów?
- W jaki sposób model numeryczny pojazdu może wspierać proces uczenia maszynowego oraz algorytmy AI w testowaniu systemów wbudowanych? Jak wykorzystać dane z symulacji pojazdu do trenowania modeli AI, aby poprawić detekcję anomalii i zwiększyć niezawodność wyników?
- Jakie elementy powinna zawierać metodyka testowania, aby skutecznie wspierać zespół inżynierów w procesie identyfikacji i eliminacji migotania rezultatów testów? Jakie narzędzia automatyzacji, analizy danych i raportowania są niezbędne, aby tester mógł skutecznie monitorować i diagnozować przyczyny niestabilnych wyników testów?

W związku z powyższym zaplanowano następujące zadania badawcze:

- Rozpoznanie zjawiska migotania rezultatów testów.
- Analiza możliwych przyczyn migotania testów.
- Zaprojektowanie stanowiska testowego, którego symulacyjny model pojazdu jest adaptowalną, modularną częścią.
- Weryfikacja przydatności zaimplementowanego symulacyjnego modelu pojazdu do reprodukcji defektów klienta.

- Sprawdzenie wpływu zastosowania w środowisku testowym symulacyjnego modelu pojazdu na występowanie zjawiska migotania rezultatów testów.
- Opracowanie koncepcji i implementacja algorytmów rozszerzonej inteligencji w procesie weryfikacji wyników testów.
- Ocena skuteczności zaimplementowanych algorytmów w poprawnym wskazywaniu anomalii występujących w środowisku testowym.
- Opracowanie metodyki umożliwiającej integrację algorytmów rozszerzonej inteligencji w proces testowania oprogramowania wbudowanego.
- Ocena zaproponowanej metodyki.

Założono, że badania będą prowadzone w rzeczywistym układzie wbudowanym przeznaczonym do produkcji, na stanowisku testowym dedykowanym do przeprowadzania testów kwalifikacyjnych. Badania będą ograniczone do określonego typu pojazdu samochodowego, co może nie oddawać pełnego spektrum możliwości i wyzwań związanych z różnorodnością systemów wbudowanych w innych typach pojazdów. Środowisko testowe, w którym przeprowadzane będą eksperymenty, jest kontrolowane i stabilne, co zapewnia powtarzalność i wiarygodność wyników. W czasie prac badawczych na podstawie istniejącego rzeczywistego stanowiska, z zastosowaniem sprzętu i narzędzi programowych stosowanych w czasie testowania układów wbudowanych, zostanie zaprojektowane stanowisko badawcze, rozszerzone o model symulacyjny pojazdu. Należy pamiętać, że proponowana metodyka powinna być wydajna i skalowalna przy wdrażaniu w różnorodnych środowiskach produkcyjnych. Praca opiera się na założeniu, że istnieją algorytmy rozszerzonej inteligencji, wystarczająco zaawansowane i skuteczne, aby wspierać proces weryfikacji testów automatycznych. W trakcie prowadzonych badań może się okazać, że istnieje potrzeba modyfikacji lub dostosowania algorytmów do specyficznych potrzeb testowanych systemów wbudowanych.

1.2 Cel rozprawy

Celem rozprawy było opracowanie skutecznej metodyki tworzenia zautomatyzowanych testów systemów wbudowanych w pojazdach samochodowych. Opracowana metodyka powinna umożliwiać weryfikację poprawności wyników testów z wykorzystaniem algorytmów rozszerzonej inteligencji. W pracy zaproponowano zastosowanie modelu pojazdu jako elementu środowiska testowego, a następnie zweryfikowano wpływ tej modyfikacji na zjawisko migotania rezultatów testów oraz potwierdzono, że zaproponowane rozwiązanie wpływa pozytywnie na jakość procesu testowania. Model pojazdu umożliwia reprodukcję defektów wykrytych przez klienta w pojeździe samochodowym. W pracy podjęto próbę zastosowania opracowanej metodyki w rzeczywistych projektach oraz empirycznego potwierdzenia jej skuteczności i przydatności w praktycznych zastosowaniach w przemyśle samochodowym, co może znacząco wpłynąć na podnoszenie standardów jakości i bezpieczeństwa nowoczesnych pojazdów.

1.3 Zakres rozprawy

Niniejsza praca doktorska koncentruje się na rozwoju i weryfikacji metodyki tworzenia zautomatyzowanych testów dla systemów wbudowanych w pojazdach samochodowych, ze szczególnym uwzględnieniem wykorzystania algorytmów rozszerzonej inteligencji do weryfikacji poprawności wyników testów. Rozprawa doktorska porusza wiele zagadnień z dziedziny inżynierii mechanicznej, w tym zastosowanie modelu pojazdu w procesie testowania systemów wbudowanych, diagnostyki anomalii w środowisku testowym z zastosowaniem algorytmów rozszerzonej inteligencji,

bezpieczeństwa i niezawodności układów wbudowanych stosowanych w przemyśle motoryzacyjnym.

Rozdział 1 ułatwia zrozumienie i identyfikację problematyki, przedstawiając genezę rozprawy, podkreślając gwałtowny wzrost zainteresowania zastosowaniem sztucznej inteligencji we wszystkich dziedzinach przemysłu, również w tematyce testowania układów wbudowanych w przemyśle samochodowym. Ponadto, w rozdziale tym przedstawiono problem badawczy, wraz z jasnym kontekstem i szkicem planowanych prac badawczych.

Rozdział 2 opisuje rozwój układów wbudowanych w przemyśle samochodowym. Rozpoczyna się od przedstawienia kompleksowego przeglądu źródeł napędu pojazdów samochodowych i ich wpływu na elementy mechatroniczne stosowne w ich produkcji. Następnie opisano wybrane normy i standardy, które mają wpływ na proces projektowania i testowania układów wbudowanych w przemyśle motoryzacyjnym. W rozdziale tym przedstawiono również zagadnienia związane z inżynierią opartą na modelach, zaczynając od projektowania, a na testowaniu kończąc. Poruszono tutaj również temat modelu pojazdu, którego implementacja została wykorzystana później w projektowanym środowisku testowym oraz eksperymentach badawczych. W ostatniej części tego rozdziału omówiono najważniejsze zagadnienia związane z testowaniem jako narzędziem weryfikacji i walidacji bezpieczeństwa funkcjonalnego układów mechatronicznych w samochodzie, zwracając szczególną uwagę na środowiska testowe, metodykę testowania i problem automatyzacji testów.

Rozdział 3 poświęcony jest rozszerzonej inteligencji. W ramach wprowadzenia w tematykę przedstawiono w nim podstawowe pojęcia związane ze sztuczną inteligencją oraz wytłumaczono koncept rozszerzonej inteligencji. Przedstawiono tutaj wyniki analizy literatury na temat obszarów zastosowań rozszerzonej inteligencji, z wyszczególnieniem dziedzin związanych z tematyką rozprawy jak diagnostyka i testowanie systemów wbudowanych.

Rozdział 4 wprowadza koncepcję metodyki testowania systemów wbudowanych uwzględniającą weryfikację wyników testów z zastosowaniem rozszerzonej inteligencji. Przedstawiono w nim elementy metodyki wymagane przez standardy obowiązujące w przemyśle motoryzacyjnym oraz szczegółowo omówiono wprowadzone innowacje. W rozdziale przedstawiona została koncepcja zaawansowanego stanowiska testowego, które integruje model pojazdu jako kluczowy jego element. Rozwiązanie to ma na celu symulację realnych warunków pracy systemów wbudowanych, umożliwiając precyzyjne testowanie ich działania. Opisano również metody eksploracji i przetwarzania danych środowiskowych wykorzystywanych do trenowania algorytmów rozszerzonej inteligencji, omawiając proces selekcji zmiennych zależnych, analizę ryzyka i opracowanie reguł oraz wstępne przetwarzanie zgromadzonych danych. Omówiono proces walidacji rezultatów testów z zastosowaniem algorytmów rozszerzonej inteligencji. W dalszej części rozdziału szczegółowo omówiono algorytmy wybrane do badań oraz miary skuteczności algorytmów używane w pracy.

W rozdziale 5 omówione zostały badania weryfikacyjne. Przedstawiony został plan badań. Poza tym szczegółowo opisano obiekt badań, omawiając budowę i zasadę działania systemu wbudowanego wybranego do badań oraz zaprojektowane i zaimplementowane stanowisko badawcze. Omówiono metodykę prowadzonych eksperymentów badawczych, opisując kolejne eksperymenty oraz przedstawiono uzyskane wyniki, a także sformułowano wnioski.

Ostatni rozdział niniejszego opracowania podsumowuje wyniki badań, przedstawiając główne wnioski. Nakreślono w nim kierunki dalszych analiz i badań w celu pełnego wykorzystania opracowanej metodyki w czasie testowania szerokiego spektrum systemów wbudowanych projektowanych przez firmę Dräxlmaier dla pojazdów samochodowych. Omówiono elementy pracy wykorzystywane w projektach systemów wbudowanych realizowanych w firmie Dräxlemaier oraz kolejne etapy wdrożenia zaproponowanych rozwiązań.

1.4 Podstawowe terminy

W celu uniknięcia niejasności związanych ze stosowanymi w rozprawie pojęciami, poniżej zamieszczono przyjęte definicje najważniejszych ze stosowanych terminów.

System wbudowany (ang. embedded system) to specjalizowany system mikroprocesorowy, zaprojektowany do wykonywania jednej lub kilku dedykowanych funkcji, często w ramach większego systemu. Charakteryzuje się on tym, że jest zazwyczaj wbudowany w urządzenie, którego funkcjonalność wspomaga lub kontroluje. Jest zoptymalizowany pod kątem określonych zadań, co oznacza, że może być bardziej efektywny energetycznie i wydajniejszy w swojej działalności niż ogólnego przeznaczenia komputery. Systemy wbudowane znajdują zastosowanie w bardzo szerokiej gamie urządzeń, od mikrokontrolerów w urządzeniach gospodarstwa domowego, przez systemy kontrolne w samochodach, aż po kontrolery w przemysłowych systemach automatyzacji. W przeciwieństwie do tradycyjnych komputerów, które mogą uruchamiać wiele różnych aplikacji, układy wbudowane zazwyczaj realizują wąski zakres zadań, przewidziany już na etapie projektowania urządzenia [44].

Sztuczna inteligencja (ang. artificial intelligence — AI) to dziedzina informatyki, która zajmuje się tworzeniem systemów zdolnych do wykonywania zadań wymagających ludzkiej inteligencji. Obejmuje to takie zdolności jak rozumowanie, uczenie się, planowanie, rozpoznawanie wzorców, przetwarzanie języka naturalnego oraz percepcję. Sztuczna inteligencja wykorzystuje algorytmy i modele matematyczne do naśladowania i realizacji procesów myślowych oraz zachowań, które tradycyjnie były uważane za domenę ludzką [45].

Rozszerzona inteligencja (ang. augmented intelligence — AuI) to nowatorskie podejście do sztucznej inteligencji mające na celu wzmocnienie zdolności ludzkiej inteligencji poprzez zastosowanie sztucznej inteligencji. W odróżnieniu od pełnej autonomii sztucznych systemów, rozszerzona inteligencja zakłada współpracę między człowiekiem a maszyną, gdzie interwencja człowieka jest niezbędna do efektywnego funkcjonowania systemu. To wzajemne przenikanie się ludzkich i sztucznych zdolności umożliwia osiągnięcie większej efektywności i precyzji, wzmacniając możliwości obu typów inteligencji. [46].

Anomalia oznacza odchylenie od normy, reguły czy oczekiwanych wzorców zachowań [47]. W kontekście środowiska testowego oznacza nieprzewidziane zmiany w warunkach fizycznych, zachowaniu elementów mechatronicznych, które prowadzą do niepoprawnej interpretacji wyników testów i mogą utrudniać prawidłową ocenę niezawodności i bezpieczeństwa systemu.

Automatyzacja to proces stosowania technologii do wykonania zadań lub kontrolowania operacji bez bezpośredniego udziału człowieka. W praktyce oznacza to wykorzystanie maszyn, systemów komputerowych, oprogramowania i innych narzędzi technologicznych do zwiększenia efektywności, precyzji i szybkości wykonania pracy, która wcześniej była wykonywana przez ludzi [31].

Testowanie układów wbudowanych jest kluczowym procesem zapewniającym, że te specjalizowane systemy mikrokomputerowe działają poprawnie i spełniają określone wymagania funkcjonalne oraz niezawodnościowe [48].

Środowisko testowe jest to specjalnie przygotowane otoczenie, które pozwala na przeprowadzenie testów oprogramowania, systemów lub produktów w warunkach kontrolowanych i odseparowanych od środowiska produkcyjnego. Jest to kluczowy element procesu testowania,

pozwalający na identyfikację błędów, weryfikację funkcjonalności i zapewnienie jakości przed wprowadzeniem produktu do użytku końcowego [48].

Wynik migoczący testu to termin używany w inżynierii oprogramowania, który opisuje zachowanie testu, które jest niestabilne lub niekonsystentne, czyli test, który w różnych przebiegach daje różne wyniki (czasami pozytywne, czasami negatywne) mimo braku zmian w kodzie testowanym. Wyniki migoczące są szczególnie problematyczne, ponieważ utrudniają diagnozę problemów i mogą wprowadzać niepewność co do stabilności systemu. Są one znacznym wyzwaniem w automatyzacji testów i wymagają systematycznego podejścia do identyfikacji, zarządzania i eliminacji ich przyczyn w celu zapewnienia stabilności i wiarygodności procesu testowania [31].

Częstość migotania rezultatów testów ε_M można zdefiniować jako proporcję negatywnych rezultatów względem wszystkich przeprowadzonych testów, co stanowi wskaźnik stabilności i niezawodności wyników testowania systemów wbudowanych. Informuje o tym jak stabilne są wyniki testów wykonanych na identycznym oprogramowaniu i sprzęcie w systemie wbudowanym w danym kontekście testowym. Częstość migotania wyrażamy wzorem:

$$\varepsilon_M = \frac{L_n}{L_t} \tag{1.1}$$

gdzie:

- L_n - liczba negatywnych rezultatów,
- $L_t = L_n + L_p$ - liczba wykonanych testów,
- L_p - liczba pozytywnych rezultatów.

Defekt - w kontekście inżynierii oprogramowania i systemów jest to błąd lub usterka w kodzie, projekcie, sprzęcie lub dokumentacji, która może prowadzić do nieprawidłowego działania systemu, aplikacji lub urządzenia. Defekty mogą powstać na różnych etapach procesu tworzenia oprogramowania lub produkcji urządzeń, a ich identyfikacja i usunięcie jest kluczowym elementem zapewnienia jakości [49].

2. Rozwój układów wbudowanych w przemyśle samochodowym

Inżynieria pojazdów samochodowych jest m.in. częścią inżynierii mechanicznej i jednocześnie obszarem o rosnącym znaczeniu w erze nowoczesnej mobilności. Współczesne pojazdy samochodowe nie tylko spełniają podstawową funkcję środka transportu, ale stają się coraz bardziej zaawansowanymi i złożonymi systemami, łączącymi technologie mechaniczne, elektroniczne oraz informatyczne. Rozwój inżynierii pojazdów nie tylko zmienia sposób, w jaki postrzegamy pojazdy, ale również wpływa na kształt współczesnych miast, ekologię oraz zachowania kierowców i pasażerów. Rozdział poświęcony jest zagadnieniom związanym z mechatroniką pojazdów samochodowych, normom i standardom obowiązującym producentów systemów wbudowanych dla przemysłu motoryzacyjnego, procesowi rozwoju systemów wbudowanych, a w szczególności inżynierii opartej na modelach oraz testowaniu systemów mechatronicznych przeznaczonych do zastosowania w samochodach.

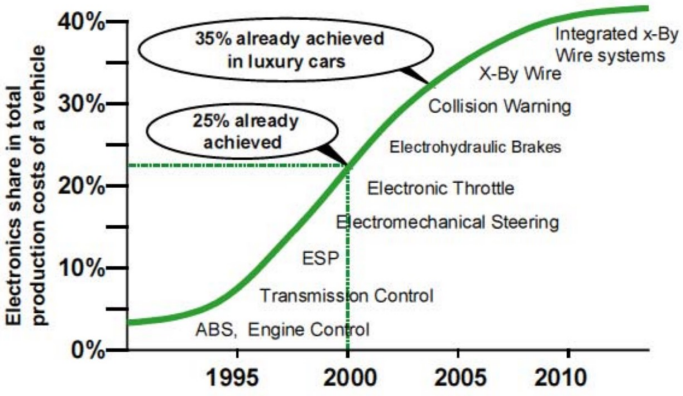
2.1 Mechatronika pojazdów samochodowych

Początki motoryzacji sięgają końca XVII wieku, kiedy to Nicolas Cugnot zamontował silnik parowy do trójkołowego podwozia, tworząc tym samym pierwszy samochód na świecie. Francuska firma De Dion-Bouton w 1884 roku rozpoczęła produkcję praktycznych samochodów parowych, które były stosunkowo łatwe w obsłudze i osiągały znaczną prędkość (średnio 20 km/h). De Dion-Bouton stał się w XIX wieku największym producentem samochodów na świecie. Pierwszy projekt czterokołowego samochodu osobowego o napędzie elektrycznym opracował w 1888 roku niemiecki przedsiębiorca Andreas Flocken. Napędzany był przy pomocy akumulatora kwasowo-ołowiowego. Najprawdopodobniej był to pierwszy elektryczny samochód osobowy na świecie. To jednak nazwisko Carla Benza, konstruktora dwusuwowego silnika spalinowego, który opatentował pierwszy pojazd spalinowy w 1886 roku, jest kojarzone z przemysłem samochodowym. Na długie lata pojazdy spalinowe zdominowały rynek, jednak w dzisiejszych czasach rozwój technologii i dążenie do zrównoważonej mobilności wywarły znaczący wpływ na źródła napędu oraz zastosowanie mechatroniki w pojazdach samochodowych. Tradycyjne silniki spalinowe ustępują miejsca innowacyjnym rozwiązaniom, takim jak napędy elektryczne, hybrydowe i wodorowe. Zapotrzebowanie na czyste źródła energii w technologii pojazdów wzrasta ze względu na negatywne skutki toksycznych emisji z silników spalinowych. Pojazdy elektryczne i wodorowe to dwie zasadnicze rozwijane technologie samochodowe przyjazne dla środowiska [50]. W połączeniu z coraz bardziej zaawansowanymi systemami mechatronicznymi tworzą one nową erę w dziedzinie inżynierii pojazdów samochodowych, wpływając na rozwój różnych dyscyplin naukowych, w tym inżynierii mechanicznej.

Na początku ery motoryzacji zastosowanie elektroniki było ograniczone do systemów rozrywkowych. Jeszcze około 60 lat temu samochód był zasadniczo maszyną sterowaną wyłącznie mechanicznie, z radiem AM na lampy próżniowe i ograniczoną ilością okablowania, potrzebną do zapewnienia funkcjonalności oświetlenia, klaksonu i rozrusznika [51]. Początek rozwoju mechatroniki w motoryzacji sięga lat 70. Pierwotnie nadzorowała ona podstawowe zadania, takie jak regulacja czasu zapłonu. W miarę upływu dekad, jej rola rozszerzała się, teraz zarządza wieloma systemami samochodowymi z ponad setką wejść i wyjść. Od swoich skromnych pierwszych

kroków po przyszłe perspektywy, ewolucja zastosowań mechatroniki podkreśla jej kluczową rolę we współczesnej motoryzacji. Obecnie w zwykłym samochodzie można znaleźć ponad 40 ECU (ang. Electronic Control Unit), a w luksusowym nawet ponad 150.

Tak duża liczba systemów wbudowanych w pojeździe jest spowodowana faktem, że dla prawie każdej nowej funkcji stworzono oddzielną jednostkę sterującą. Jest to wynik powszechnie stosowanych procesów rozwojowych, które w celu zarządzania odpowiedzialnościami, testowalnością i różnorodnością dostawców, segmentują sprzęt zgodnie z różnymi funkcjami [52].
Na rysunku 2.1 pokazano kierunki rozwoju zastosowań systemów mechatronicznych w motoryzacji.



Rys. 2.1. Kierunki rozwoju mechatroniki w motoryzacji [51]

Jednym z pierwszych zastosowań mechatronicznych systemów w pojazdach było wprowadzenie systemu przeciwblokującego hamulce (ang. anti-lock braking system — ABS) na początku lat 70. XX wieku. Następnie zaczęły się pojawiać systemy regulujące czas zapłonu, optymalizujące mieszanko paliwowo — powietrzne dla efektywnego spalania. W latach 90-tych systemy mechatroniczne rozszerzyły swoją rolę o funkcjonalności związane z bezpieczeństwem pojazdu. Początek XXI wieku przyniósł znaczne zmiany przede wszystkim w zarządzaniu przepustnicą przez wprowadzenie systemu elektronicznej przepustnicy. Do dnia dzisiejszego obserwujemy ciągłe rozszerzanie zastosowań systemów wbudowanych, zarządzających szeroką gamą wejść i wyjść w różnorodnych systemach samochodowych. Zaawansowane funkcje, takie jak zarządzanie silnikiem, kontrola trakcji i aktywna dynamika pojazdu, mogą być dziś realizowane jedynie dzięki umiejętnemu połączeniu technologii mechatronicznych [53]. Mechatronika wyróżnia się jako integralny element napędzający innowacje w branży motoryzacyjnej. W rozwoju zastosowań systemów mechatronicznych uwidaczniają się następujące trendy [54]:

- Zwiększona łączność (ang. Enhanced Connectivity) - trend, który stawia na poprawę integracji systemów pojazdu oraz bezpieczeństwo kierowcy dzięki zaawansowanym funkcjom łączności.
- Sztuczna inteligencja i uczenie maszynowe — systemy wbudowane będą wykorzystywać AI i ML do adaptacji wydajności pojazdu, zapewniając bezpieczeństwo i efektywność paliwową.
- Cyberbezpieczeństwo (ang. Cybersecurity) - jest to funkcjonalność bezpośrednio związana ze zwiększoną łącznością, priorytetowo nowoczesne pojazdy będą musiały być wyposażone w zaawansowane funkcje bezpieczeństwa w celu ochrony przed cyberatakami.
- Jazda autonomiczna (ang. Autonomous Driving) - systemy wbudowane przetwarzające

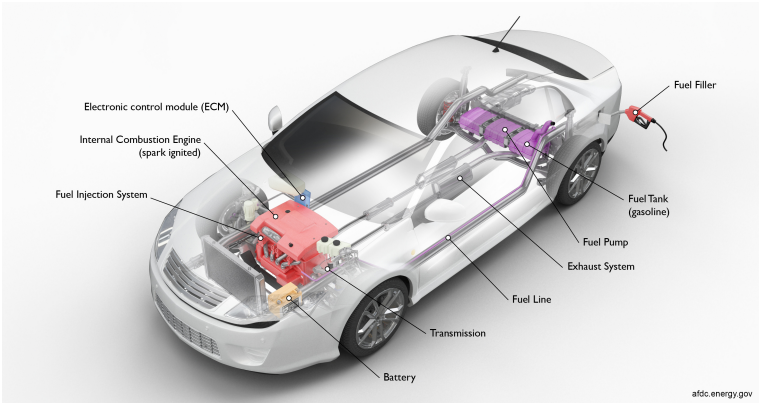
dane z czujników będą kluczowe, aby zapewnić bezpieczeństwo operacyjne pojazdów autonomicznych.

- Inteligentna infrastruktura (ang. Smart Infrastructure) - systemy wbudowane w pojeździe będą współpracowały z infrastrukturą inteligentnych miast, optymalizując przepływ ruchu i redukując emisję.
- Spersonalizowane doświadczenie (ang. Personalized Experience) - systemy wbudowane będą zapewniały dostosowanie funkcjonalności do indywidualnych preferencji użytkowników.

Podczas każdej jazdy ECU przetwarzają ogromną ilość informacji lub danych [55]. W miarę wzrostu złożoności tych jednostek, zarządzanie nimi staje się strategicznym wyzwaniem dla producentów OEM i twórców motoryzacyjnych systemów wbudowanych.

2.1.1 Pojazdy spalinowe

Historia pojazdów spalinowych (ang. Combustion Vehicles - CV) zaczęła się od patentu Carla Benza w 1886 roku [56], pierwszego samochodu montowanego na taśmie produkcyjnej od 1901 roku - Oldsmobile Curved Dash [57]. Gdy w 1908 roku firmę Oldsmobile przejęło General Motors, a w przedsiębiorstwie Henrego Forda z taśmy produkcyjnej zjechał pierwszy Model T [58] zaczął się wyścig o to, kto będzie produkował tańsze i niezawodne samochody. Od tego momentu liczy się moc silnika, zużycie paliwa - konsumenci zawsze chcieli jeździć szybciej i taniej. Światowy kryzys paliwowy i świadomość konieczności zmniejszania emisji CO2 napędzają zmiany w przemyśle motoryzacyjnym. Budowa pojazdów zasilanych benzyną i olejem napędowym jest podobna. Obydwa są napędzane silnikiem spalinowym, różniącymi się sposobem zapłonu. W pojazdach zasilanych benzyną paliwo jest wtryskiwane do komory spalania, a tam mieszane z powietrzem. Zapalenie mieszanki następuje od iskry ze świecy zapłonowej (zapłon iskrowy). Główne elementy budowy pojazdu spalinowego pokazano na rysunku 2.2:

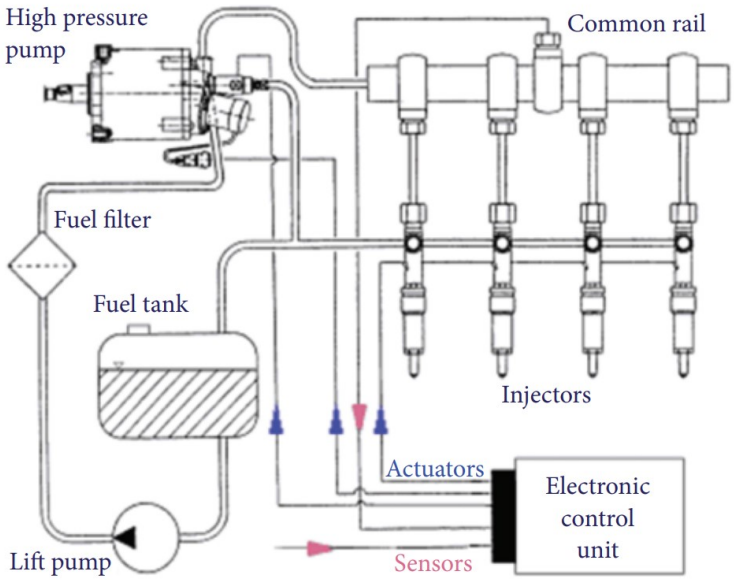


Rys. 2.2. Główne elementy budowy pojazdu spalinowego [59]

- Akumulator (ang. Battery) — zapewnia energię elektryczną niezbędną do uruchomienia silnika i zasilania elektroniki / akcesoriów pojazdu.
- Silnik spalinowy (ang. Combustion Engine)
- Skrzynia biegów (ang. Transmission) - przekładnia przenosząca moc mechaniczną z silnika lub / i elektrycznego silnika trakcyjnego do kół.
- Elektroniczny Moduł Sterujący (ang. Electronic Control Module - ECM) - reguluje skład mieszanki paliwowej, steruje czasem zapłonu i układem emisji, wykrywa i rozwiązuje problemy, chroni silnik i monitoruje pracę pojazdu.

- Wlew paliwa (ang. Fuel filler)
- Zbiornik paliwa (ang. Fuel tank)
- Pompa paliwa (ang. Fuel pump) - tłoczy paliwo przewodem paliwowym ze zbiornika do układu wtrysku paliwa silnika.
- Przewód paliwowy (ang. Fuel line) - metalowa rurka lub elastyczny wąż (lub ich kombinacja) do transportu paliwa ze zbiornika do układu wtrysku silnika.
- Układ wtrysku paliwa (ang. Fuel injection system) - układ ten wprowadza paliwo do komór spalania silnika w celu zapłonu.
- Układ wydechowy (ang. Exhaust system) - odprowadza spaliny z silnika przez rurę wydechową. Katalizator trójdrożny ma na celu redukcję emisji gazów cieplarnianych w układzie wydechowym.

Pod koniec lat 90-tych XX wieku rozpoczął się gwałtowny rozwój silników wysokoprężnych z technologią wtrysku Common Rail pokazany na 2.3. Zapewnia on poprawę mocy, hałasu i charakterystyki emisji spalin w systemach spalania silników diesla. Wspólny wysokociśnieniowy zbiornik paliwa („common rail") dostarcza paliwo do wtryskiwaczy pod ciśnieniem do 160 MPa. Generowanie ciśnienia i wtrysk są od siebie oddzielone. Skoordynowaną i dokładną współpracę różnych części systemu (np. wysokociśnieniowej pompy paliwa, elektrozaworu, dyszy itp.) przez cały cykl życia samochodu zapewniają zaawansowane algorytmy do sterowania i diagnozowania systemu, realizowane przez ECU [60].



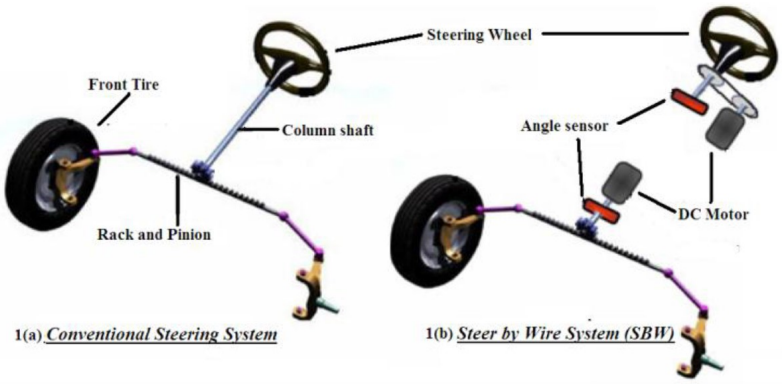
Rys. 2.3. Schemat systemu wtrysku Common Rail [61]

Oprócz systemów opartych na elektrozaworach stosowane są również systemy wtrysku paliwa sterowane piezoelektrycznie, które zapewniają jeszcze wyższe poziomy ciśnienia (do 180 MPa) i większą elastyczność cyklu wtrysku. Równolegle prowadzono próby wprowadzenia wtrysku bezpośredniego w silnikach benzynowych. Główną zaletą silników benzynowych ze wtryskiem bezpośrednim są bardzo czyste spaliny, a ponadto większa wydajność oraz korzystniejszy przebieg momentu obrotowego. Wtrysk bezpośredni stał się standardem, chociaż w mniejszych, wolnossących silnikach napędzających małe samochody nadal nie zawsze był stosowany. Niemniej jednak planuje się wprowadzenie go do wszystkich silników benzynowych w najbliższych latach. Można

się spodziewać dalszych prac nad zwiększeniem wydajności silników benzynowych poprzez podniesienie ciśnienia doładowania, ciśnienia wtrysku i zwiększenia stopnia sprężania.

Na wydajność silnika wpływają także straty generowane przez wszystkie elementy osprzętu, każde łożysko, każda ruchoma część. Również w obszarze ograniczania tarcia oraz oporów ruchu inżynierowie dokonali znacznych postępów. Łożyska są projektowane tak, aby były jak najlżejsze i generowały jak najmniejsze straty. W układzie korbowo — tłokowym powszechnie stosuje się materiały o niskim tarciu współpracujące z olejami silnikowymi o niskiej lepkości. Podczas hamowania alternatory odzyskują energię, ładując akumulator (tzw. technologia mikrohybrydowa). Pompy, które kiedyś były napędzane przez silnik za pośrednictwem paska lub prostej przekładni, stają się teraz elektryczne, podobnie jak wspomaganie kierownicy, którego wersja hydrauliczna przechodzi już do historii.

W przypadku funkcji kierowania pojazdem dążenie do poprawy bezpieczeństwa, manewrowości i komfortu samochodów, przemysły doprowadziło do zastąpienia konwencjonalnych mechanicznych połączeń w pojazdach elektrycznymi czujnikami, siłownikami i sieciami komunikacyjnymi znanymi jako technologia „steer by wire" (SBW). System ten, jak wskazuje jego nazwa, łączy kierowcę z siłownikami w samochodzie za pomocą przewodów elektrycznych, co pozwala na zmniejszenie wagi samochodu oraz eliminację wibracji przenoszonych na kierowcę przez konwencjonalne połączenia mechaniczne [62]. Na Rys. 2.4 pokazano porównanie konwencjonalnego systemu kierowniczego z systemem "steer by wire"(SBW).



Rys. 2.4. Porównanie konwencjonalnego systemu kierowania pojazdem z mechatronicznym [62]

W przypadku funkcji kierowania, na rynku wprowadzane są systemy, które zmieniają przełożenie układu kierowniczego w zależności od prędkości jazdy [63]. Obecnie badane są bardziej zaawansowane systemy kierowania, które automatycznie kompensują wpływ bocznego wiatru oraz pomagają w stabilizacji pojazdu podczas ekstremalnych manewrów, kompensując podsterowność lub nadsterowność. Stosowanie termostatów fazowych, zbliżanie do siebie elementów czy stosowanie aluminium jako materiału do budowy głowic i kadłubów silników to sposoby na ograniczenie strat energii podczas ogrzewania silnika, który pracuje wydajnie i ekologicznie tylko w określonym, wąskim zakresie temperatur. Standardem w silnikach stały się tzw. zawory ERG, czyli urządzenia pozwalające na recyrkulację spalin, czyli kilkukrotne ich przetwarzanie w silniku. Zawory ERG nie potrafią wyeliminować bardzo szkodliwych cząstek stałych oraz tlenków azotu dlatego w samochodach z silnikiem Diesla stosuje się filtry cząstek stałych. Wprowadzenie normy Euro 6 spowodowało, że konieczne jest stosowanie technologii jeszcze lepiej oczyszczającej spaliny z tlenków azotu zwanej selektywną redukcją katalityczną (SCR). Silniki z technologią SCR wykorzystują wodny roztwór mocznika AdBlue, który jest magazynowany w oddzielnym zbiorniku i wtryskiwany do układu wydechowego pojazdu. W katalizatorze mocznik reaguje chemicznie i zamienia szkodliwe gazy w azot i parę wodną.

Dzisiejsze silniki są silnikami niskoobrotowymi niezależnie od tego czy są zasilane benzyną,

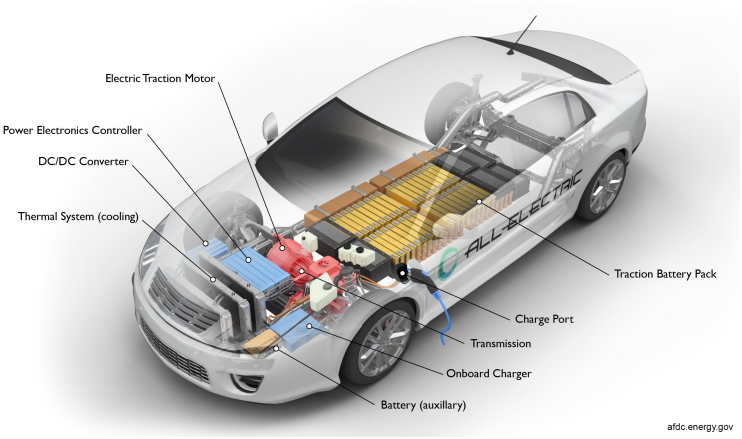
czy olejem napędowym. Niedobór momentu obrotowego w silniku jest eliminowany dzięki małym turbosprężarkom ze zmienną geometrią łopatek turbiny, sterowaną elektronicznie. Zastosowanie koła dwumasowego współpracującego z tłumikami na kołach pasowych eliminuje wibracje wynikające z pracy na obrotach rzędu 1200 - 1600. Ewoluowały również przekładnie, zarówno manuale, jak i automatyczne. Duży wybór dostępnych przełożeń można zawsze jechać w optymalnym zakresie obrotów silnika, niezależnie od prędkości.

Rada Uni Europejskiej przyjęła rozporządzenie Euro 7, ustanawiające nowe przepisy o limitach transmisji dla samochodów. Oznacza to, że prawdopodobnie od 2035 roku na rynku europejskim nie będą już sprzedawane nowe auta osobowe i dostawcze z silnikami spalinowymi. Wszystkie nowe pojazdy na terenie Europy mają być zeroemisyjne, co oznacza, że nie będzie można także kupować ani rejestrować samochodów hybrydowych.

Norma Euro 7, wprowadzając limity emisji tlenków azotu, narzuca również nowe warunki dotyczące emisji cząstek stałych, monitorowania emisji oraz notyfikowania producenta o potencjalnych przekroczeniach i awariach. To wszystko oznacza zwiększone koszty produkcji, co skutkuje wyższymi cenami samochodów spalinowych. Według szacunków Morgan Stanley, wiodący producenci motoryzacyjni, tak jak Volkswagen czy Stellantis, mogą ponieść koszty redukcji emisji oraz dostosowania się do nowych wymagań sięgające nawet 350-450 milionów euro. To może skłonić wielu producentów do wcześniejszego wycofania się z oferty samochodów spalinowych niż wymaga tego Unia Europejska [64].

2.1.2 Pojazdy elektryczne

Pojazdy w pełni elektryczne (ang. battery electric vehicles - BEV) są napędzane silnikiem elektrycznym, który jest zasilany z dużego zestawu akumulatorów wysokiego napięcia. Baterie te muszą być podłączane do gniazdka ściennego lub urządzenia ładującego w celu uzupełnienia energii elektrycznej w akumulatorze. Pojazdy elektryczne nie emitują spalin i nie zawierają typowych elementów dla paliwa płynnego.



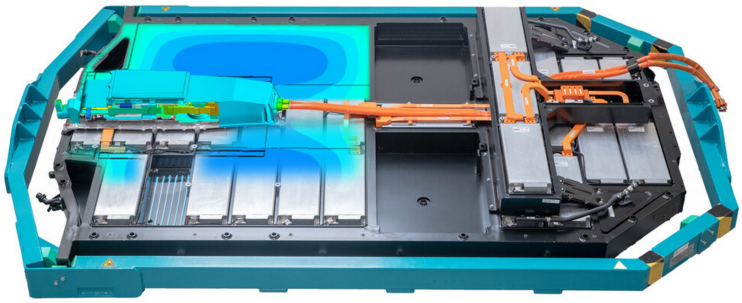
Rys. 2.5. Główne elementy budowy pojazdu elektrycznego [65]

Główne elementy budowy pojazdu elektrycznego pokazano na Rys. 2.5:

- Akumulator (ang. Battery) - to akumulator pomocniczy zapewniający energię elektryczną dla akcesoriów pojazdu (niskonapięciowy).
- Elektryczny silnik trakcyjny (ang. electric traction motor) - wykorzystuje energię elektryczną z baterii wysokonapięciowej do napędzania kół pojazdu. Często elementem powiązanym z silnikiem są agregaty prądotwórcze pełniące funkcję regeneracyjną (rekuperacja).

- Elektryczna skrzynia biegów (ang. Electric transmission) - przekładnia przenosząca moc mechaniczną z elektrycznego silnika trakcyjnego w celu napędzania kół.
- Port ładowania (ang. Charge port) - umożliwia podłączenie pojazdu do zewnętrznego źródła zasilania w celu naładowania pakietu akumulatorów trakcyjnych.
- Ładowarka pokładowa (ang. Onboard charger) - przekształca energię prądu przemiennego dostarczaną przez port ładowania w prąd stały w celu ładowania wysokonapięciowego akumulatora trakcyjnego. Komunikuje się również ze sprzętem ładującym i monitoruje właściwości akumulatora, takie jak napięcie, prąd, temperatura i stan naładowania.
- Wysokonapięciowy akumulator trakcyjny - (ang. High voltage battery pack) - magazynuje energię do zasilania silnika trakcyjnego.
- Sterownik elektroniki mocy (ang. Power electronics controller) - zarządza przepływem energii elektrycznej dostarczanej przez akumulator trakcyjny, kontrolując prędkość elektrycznego silnika trakcyjnego i wytwarzany przez niego moment obrotowy.
- Przetwornica DC/DC (ang. DC/DC converter) - przekształca prąd stały o wysokim napięciu z akumulatora trakcyjnego na prąd stały o niższym napięciu potrzebny do zasilania akcesoriów pojazdu i ładowania akumulatora pomocniczego.
- Układ termiczny chłodzący (ang. Thermal cooling system) - układ odpowiedzialny za utrzymanie właściwego zakresu temperatur pracy silnika, układu zarządzającego mocą i innych podzespołów.

Układy napędowe pojazdów elektrycznych cechują się dużym poziomem złożoności, która wynika z zastosowania różnych wariantów zasilania, elementów elektronicznych, elektrycznych czy mechanicznych [66]. Systemy zarządzania bateriami (ang. Battery Management Systems - BMS) to mechatroniczne systemy wbudowane, których zadaniem jest monitorowanie stanu akumulatorów, utrzymywaniem ich w bezpiecznym obszarze pracy i zapewnianiem optymalnego wykorzystania pozostałej energii w akumulatorach, maksymalizując tym samym ich bezpieczeństwo, niezawodność, trwałość i wydajność.



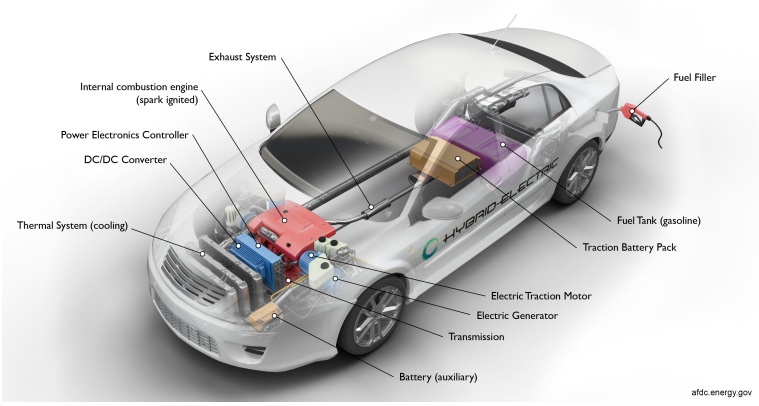
Rys. 2.6. Akumulator wysokonapięciowy w pojeździe elektrycznym

Stan akumulatora wysokiego napięcia pojazdu elektrycznego jest monitorowany przez system BMS za pomocą różnych bezpośrednich i pośrednich pomiarów. Informacje z czujników prądu, napięcia oraz temperatury wewnętrznej ogniw akumulatora są przetwarzane przez ECU i wykorzystywane do oszacowania stanu naładowania akumulatora (SOC), który wskazuje poziom naładowania akumulatora; stanu zdrowia (SOH), który reprezentuje pozostałą pojemność akumulatora i wskazuje, kiedy należy go wymienić. Szczególnie istotne są różnice temperatur między najzimniejszą a najcieplejszą komórką. Nawet niewielkie różnice w oporze wewnętrznym mogą powodować różne rozkłady prądu, w zależności od połączenia i typu komórek. To z kolei

wpływa na cykliczne starzenie się komórek akumulatora, które jest związane z użytkowaniem, a tym samym na żywotność akumulatora. Podobnie ważne jest dokładne monitorowanie napięcia, ponieważ odchylenia napięcia spowodowane przeładowaniem, nadmiernym rozładowaniem lub impulsami o dużej mocy mogą prowadzić do znacznie skróconej żywotności akumulatora i problemów z bezpieczeństwem.

2.1.3 Pojazdy hybrydowe

Hybrydowe pojazdy elektryczne (ang. hybrid electric vehicles — HEVs) są napędzane są silnikiem spalinowym, wspomaganym przez jeden lub większą liczbę silników elektrycznych wykorzystujących energię zgromadzoną w akumulatorach. Hybrydowego pojazdu elektrycznego nie można podłączyć do zewnętrznej ładowarki w celu ładowania akumulatora, natomiast jest ładowany w czasie hamowania regeneracyjnego oraz z silnika spalinowego. Dodatkowa moc zapewniana przez silnik elektryczny pozwala na zastosowanie mniejszego silnika spalinowego. Dzięki akumulatorowi możliwe jest ograniczenie pracy silnika spalinowego na biegu jałowym po zatrzymaniu pojazdu. Pozwala to na ograniczenie zużycia paliwa bez utraty wydajności.



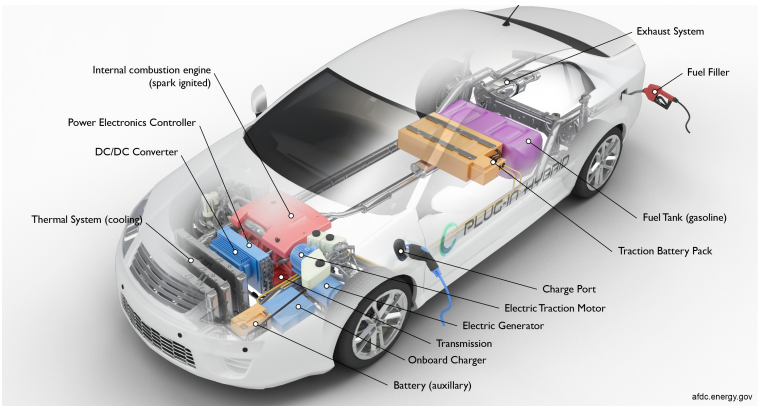
Rys. 2.7. Główne elementy budowy pojazdu hybrydowego [67]

Główne elementy budowy pojazdu hybrydowego pokazano na Rys. 2.7:

- Akumulator (ang. Battery) - to akumulator pomocniczy niskiego napięcia zapewniający energię elektryczną niezbędną do uruchomienia samochodu przed włączeniem akumulatora trakcyjnego, zasila również akcesoria pojazdu (niskonapięciowy).
- Elektryczny silnik trakcyjny (ang. electric traction motor) - wykorzystuje energię elektryczną z pakietu akumulatorów trakcyjnych do napędzania kół pojazdu. Często elementem powiązanym z silnikiem są agregaty prądotwórcze pełniące funkcję regeneracyjną (rekuperacja).
- Silnik spalinowy (ang. Combustion Engine)
- Skrzynia biegów (ang. Transmission) - przekładnia przenosząca moc mechaniczną z silnika lub / i elektrycznego silnika trakcyjnego do kół.
- Pakiet akumulatorów trakcyjnych - (ang. Traction battery pack) - magazynuje energię do zasilania silnika trakcyjnego.
- Sterownik elektroniki mocy (ang. Power electronics controller) - zarządza przepływem energii elektrycznej dostarczanej przez akumulator trakcyjny, kontrolując prędkość elektrycznego silnika trakcyjnego i wytwarzany przez niego moment obrotowy.
- Przetwornica DC/DC (ang. DC/DC converter) - przekształca prąd stały o wysokim napięciu z akumulatora trakcyjnego na prąd stały o niższym napięciu potrzebny do zasilania akcesoriów pojazdu i ładowania akumulatora pomocniczego.

- Generator elektryczny (ang. Electric genereator) - podczas hamowania wytwarza energię elektryczną z obracających się kół i przekazuje ją z powrotem do akumulatora trakcyjnego. W niektórych pojazdach zastosowano agregaty prądotwórcze, które pełnią zarówno funkcję napędową, jak i regeneracyjną.
- Układ termiczny chłodzący (ang. Thermal cooling system) - układ odpowiedzialny za utrzymanie właściwego zakresu temperatur pracy silnika elektrycznego i spalinowego, układu zarządzającego mocą i innych podzespołów.
- Wlew paliwa (ang. Fuel filler)
- Zbiornik paliwa (ang. Fuel tank)
- Układ wydechowy (ang. Exhaust system) - odprowadza spaliny z silnika przez rurę wydechową. Katalizator trójdrożny ma na celu redukcję emisji gazów cieplarnianych w układzie wydechowym.

Hybrydowe pojazdy elektryczne typu plug-in (PHEV) łączą w sobie dwa źródła zasilania: wykorzystują akumulatory do zasilania silnika elektrycznego, a także inne paliwo, takie jak benzyna lub olej napędowy, do zasilania silnika spalinowego. W odróżnieniu od HEV pojazdy PHEV mogą być ładowane z zewnętrznych ładowarek. Zaletą pojazdów PHEV jest generowanie niższego poziomu emisji, w zależności od źródła energii elektrycznej i częstotliwości użytkowania pojazdu w trybie całkowicie elektrycznym.



Rys. 2.8. Główne elementy budowy pojazdu hybrydowego Plug - In [68]

Główne elementy budowy pojazdu hybrydowego pokazano na rysunku 2.8:

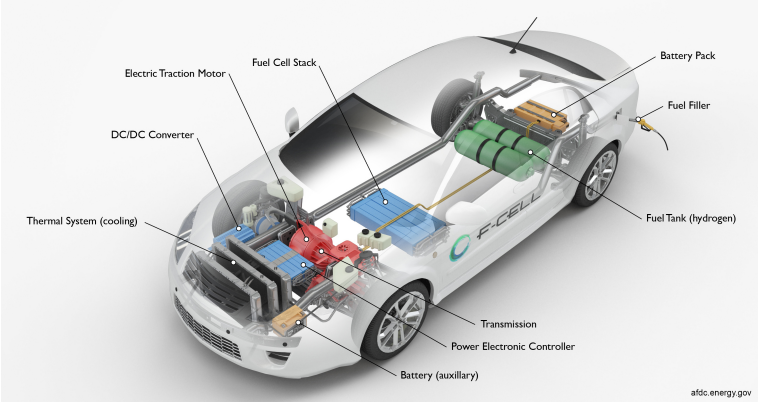
- Akumulator (ang. Battery) - to akumulator pomocniczy niskiego napięcia zapewniający energię elektryczną niezbędną do uruchomienia samochodu przed włączeniem akumulatora trakcyjnego, zasila również akcesoria pojazdu (niskonapięciowy).
- Elektryczny silnik trakcyjny (ang. electric traction motor) - wykorzystuje energię elektryczną z pakietu akumulatorów trakcyjnych do napędzania kół pojazdu. Często elementem powiązanym z silnikiem są agregaty prądotwórcze pełniące funkcję regeneracyjną (rekuperacja).
- Silnik spalinowy (ang. Combustion Engine)
- Skrzynia biegów (ang. Transmission) - przekładnia przenosząca moc mechaniczną z silnika lub / i elektrycznego silnika trakcyjnego do kół.
- Pakiet akumulatorów trakcyjnych - (ang. Traction battery pack) - magazynuje energię do zasilania silnika trakcyjnego.

- Sterownik elektroniki mocy (ang. Power electronics controller) - zarządza przepływem energii elektrycznej dostarczanej przez akumulator trakcyjny, kontrolując prędkość elektrycznego silnika trakcyjnego i wytwarzany przez niego moment obrotowy.
- Przetwornica DC/DC (ang. DC/DC converter) - przekształca prąd stały o wysokim napięciu z akumulatora trakcyjnego na prąd stały o niższym napięciu potrzebny do zasilania akcesoriów pojazdu i ładowania akumulatora pomocniczego.
- Generator elektryczny (ang. Electric genereator) - podczas hamowania wytwarza energię elektryczną z obracających się kół i przekazuje ją z powrotem do akumulatora trakcyjnego. W niektórych pojazdach zastosowano agregaty prądotwórcze, które pełnią zarówno funkcję napędową, jak i regeneracyjną.
- Układ termiczny chłodzący (ang. Thermal cooling system) - układ odpowiedzialny za utrzymanie właściwego zakresu temperatur pracy silnika elektrycznego i spalinowego, układu zarządzającego mocą i innych podzespołów.
- Wlew paliwa (ang. Fuel filler)
- Zbiornik paliwa (ang. Fuel tank)
- Układ wydechowy (ang. Exhaust system) - odprowadza spaliny z silnika przez rurę wydechową. Katalizator trójdrożny ma na celu redukcję emisji gazów cieplarnianych w układzie wydechowym.
- Port ładowania (ang. Charge port) - umożliwia podłączenie pojazdu do zewnętrznego źródła zasilania w celu naładowania pakietu akumulatorów trakcyjnych.
- Ładowarka pokładowa (ang. Onboard charger) - przekształca energię prądu przemiennego dostarczaną przez port ładowania w prąd stały w celu ładowania wysokonapięciowego akumulatora trakcyjnego. Komunikuje się również ze sprzętem ładującym i monitoruje właściwości akumulatora, takie jak napięcie, prąd, temperatura i stan naładowania.

Pojazdy hybrydowe wymagają nowych układów mechatronicznych systemów sterowania, które muszą obejmować nie tylko mieszany napęd silnikiem spalinowym i elektrycznym, ale także uwzględniać aspekty hamowania rekuperacyjnego z uruchamianiem niezależnym od pedału hamulca [69].

2.1.4 Pojazdy wodorowe

Pojazdy zasilane wodorem (Fuel Cell Electric Vehicles - FCEV) wykorzystują układ napędowy podobny do pojazdów elektrycznych, w których energia zmagazynowana w postaci wodoru jest przekształcana w energię elektryczną przez ogniwo paliwowe. W przeciwieństwie do innych pojazdów elektrycznych, pojazdy FCEV wytwarzają energię elektryczną za pomocą ogniwa paliwowego zasilanego wodorem, a nie pobierają energię elektryczną wyłącznie z akumulatora. Większość dzisiejszych pojazdów FCEV wykorzystuje akumulator gromadzenia energii odzyskanej z hamowania. Dzięki temu zapewnia dodatkową moc podczas krótkich przyspieszań oraz do wygładzania mocy dostarczanej z ogniwa paliwowego w czasie pracy na biegu jałowym lub w przypadku niskiego zapotrzebowania na moc. O ilości energii zmagazynowanej na pokładzie decyduje wielkość zbiornika paliwa wodorowego. Rozważając zasilanie pojazdów wodorem jako zielonym, ekologicznym źródłem energii należy mieć na uwadze, że odnawialne metody produkcji wodoru, takie jak elektroliza wody zasilana energią wiatrową lub słoneczną, są dostępne, ale droższe niż parowy reforming metanowy. Szacuje się, że produkcja wodoru ze źródeł odnawialnych jest od dwóch do trzech razy droższa niż wodór wytwarzany z gazu ziemnego. Chociaż wodór może być niskoemisyjną alternatywą dla paliw kopalnych, jego produkcja wymaga znacznych ilości energii i infrastruktury, a większość produkowanego obecnie wodoru pochodzi z paliw kopalnych, co wiąże się z emisjami CO2.



Rys. 2.9. Główne elementy budowy pojazdu wodorowego [70]

Główne elementy budowy pojazdu hybrydowego pokazanego na Rys. 2.9:

- Akumulator (ang. Battery) - to akumulator pomocniczy zapewniający energię elektryczną dla akcesoriów pojazdu (niskonapięciowy).
- Elektryczny silnik trakcyjny (ang. electric traction motor) - wykorzystuje energię z ogniwa paliwowego i pakietu akumulatorów trakcyjnych do napędzania kół pojazdu. Często elementem powiązanym z silnikiem są agregaty prądotwórcze pełniące funkcję regeneracyjną (rekuperacja).
- Elektryczna skrzynia biegów (ang. Electric transmission) - przekładnia przenosząca moc mechaniczną z elektrycznego silnika trakcyjnego w celu napędzania kół.
- Pakiet akumulatorów (ang. Battery pack) - akumulator wysokiego napięcia przechowuje energię wytwarzaną podczas hamowania regeneracyjnego i zapewnia dodatkową moc elektrycznemu silnikowi trakcyjnemu.
- Sterownik elektroniki mocy (ang. Power electronics controller) - zarządza przepływem energii elektrycznej dostarczanej przez akumulator ogniwo paliwowe i akumulator trakcyjny, kontrolując prędkość elektrycznego silnika trakcyjnego i wytwarzany przez niego moment obrotowy.
- Przetwornica DC/DC (ang. DC/DC converter) - przekształca prąd stały o wysokim napięciu z akumulatora trakcyjnego na prąd stały o niższym napięciu potrzebny do zasilania akcesoriów pojazdu i ładowania akumulatora pomocniczego.
- Układ termiczny chłodzący (ang. Thermal cooling system) - układ odpowiedzialny za utrzymanie właściwego zakresu temperatur pracy ogniwa paliwowego, silnika elektrycznego, układu zarządzającego mocą i innych podzespołów.
- Zbiór ogniw paliwowych (ang. Fuel Cell Stack) - zespół pojedynczych elektrod membranowych, które wykorzystują wodór i tlen do wytwarzania energii elektrycznej.
- Zbiornik paliwa (ang. Fuel tank) - zbiornik służący do przechowywania wodoru do czasu, aż będzie potrzebny do doładowania ogniwa paliwowego.
- Wlew paliwa (ang. Fuel filler) - dysza dystrybutora paliwa.

2.1.5 Podsumowanie

W branży motoryzacyjnej nie osiągnięto konsensusu co do najlepszego podejścia do ograniczenia emisji gazów cieplarnianych. Wielu producentów samochodów koncentruje się na pojazdach elektrycznych zasilanych akumulatorami. Jednak równolegle trwają prace nad technologią wodorowych ogniw paliwowych, która może zapewnić jazdę bezemisyjną, ale jest mniej wydajna

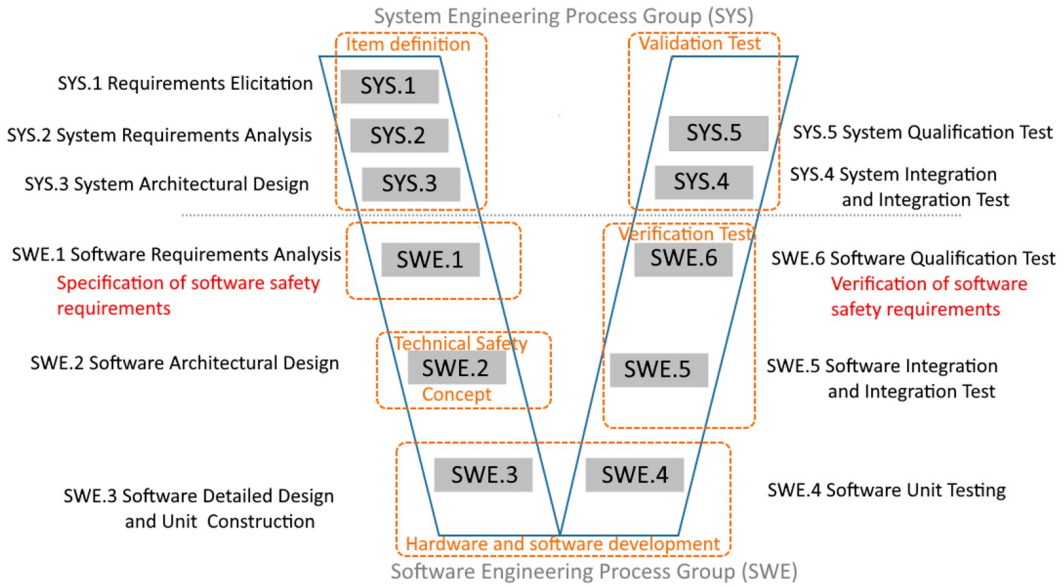
niż pojazdy elektryczne zasilane akumulatorowo. Ze względu na brak infrastruktury i niższą wydajność, reakcja społeczeństwa na pojazdy elektryczne napędzane ogniwami paliwowymi jest słaba. Podczas gdy ładowanie samochodów elektrycznych jest możliwe w nocy, w domu kierowcy, nie dotyczy to pojazdów z wodorowymi ogniwami paliwowymi [50].

2.2 Wybrane normy i standardy

Ewolucja, jaką przechodzą współczesne samochody, nowe funkcjonalności związane z rozrywką, e-mobilnością i autonomią prowadzą do wzrostu skomplikowania systemów wbudowanych realizujących te zadania [71]. Nowoczesne pojazdy zawierają ponad 100 jednostek ECU połączonych ze sobą magistralami danych, tworząc w ten sposób zaawansowany system mechatroniczny. Te systemy muszą nie tylko spełniać wymagania dotyczące funkcjonalności i wydajności, ale także skutecznie współdziałać z innymi komponentami, pracującymi w czasie rzeczywistym. W dodatku muszą spełniać rygorystyczne normy dotyczące bezpieczeństwa funkcjonalnego oraz niezawodności [72]. Priorytetowym zadaniem w procesie produkcji samochodów jest nie tylko zapewnienie jakości, ale przede wszystkim bezpieczeństwa i niezawodności działania zarówno poszczególnych komponentów, jak i całego systemu. Inżynierowie pracujący nad systemami wbudowanymi w pojazdach są zobligowani do przestrzegania określonych procesów i standardów. Współczesne normy, takie jak ISO26262 [8], ASPICE [9] czy AUTOSAR [73], stanowią wytyczne dla tworzenia tych systemów, zapewniając spójność oraz bezpieczeństwo. Jednocześnie, istnieją także bardziej tradycyjne standardy, jak AEC [74], OBD-II [75] i MISRA [76], które nadal mają znaczenie i wpływ na praktyki inżynierskie [77]. Przestrzeganie tych standardów jest kluczowe do projektowania i budowy złożonych, niezawodnych systemów mechatronicznych odpornych, na błędy i uszkodzenia.

2.2.1 Proces rozwoju układów wbudowanych w przemyśle samochodowym

Norma ISO26262 oraz standard ASPICE definiują kolejne fazy rozwoju systemów wbudowanych, które schematycznie przedstawia V-model. ASPICE (ang. Automotive Software Process Improvement and Capability dEtermination) jest modelem oceny procesów w przemyśle motoryzacyjnym, który koncentruje się na doskonaleniu procesów inżynierii oprogramowania wbudowanego. W tym kontekście V-model jest używany jako struktura, która ułatwia organizację i zarządzanie procesami oraz integrację poszczególnych faz projektowania i testowania. Z kolei norma ISO 26262 to międzynarodowa norma dotycząca bezpieczeństwa funkcjonalnego pojazdów drogowych, która szczegółowo opisuje wymagania dotyczące zapewnienia bezpieczeństwa systemów wbudowanych w samochodach. W kontekście ISO 26262 V-model pomaga w realizacji procesu rozwoju oprogramowania, zapewniając, że każdy etap projektowania (lewa strona V-modelu) jest powiązany z odpowiednim etapem testowania (prawa strona V-modelu). Pozornie wygląda on jak klasyczna metoda wodospadu (ang. waterfall), która zakłada, że produkt wytwarzany jest „w jednym przebiegu", czyli prace implementacyjne zaczynają się dopiero po opracowaniu ostatecznej specyfikacji. Należy podkreślić, że w przypadku systemów wbudowanych, gdy producenci pojazdów korzystają z wielu dostawców podzespołów, metoda wodospadu jest niepraktyczna. Nowy model pojazdu czy wyposażenia powstaje, rozwijając równocześnie wiele funkcjonalności. Ze względu na tę równoległość i konieczność integracji w pojeździe wielu elementów, w rzeczywistości rozwój danego systemu wbudowanego obejmuje wiele cykli V-modelu. Powstaje architektura wysokiego poziomu, a następnie dekompozycja i podział projektowanych systemów na komponenty. Systemy powstają iteracyjnie, realizując metodykę V-Modelu - najpierw powstaje dokumentacja, a następnie implementacja. Kolejne etapy rozwoju systemów wbudowanych przedstawiono na rysunku 2.10.



Rys. 2.10. Model etapów rozwoju systemów wbudowanych w motoryzacji na podstawie normy ISO26262 oraz ASPICE v3.1 [72]

Pierwszy etap rozwoju systemów wbudowanych, oznaczony jako SYS.1 to pozyskiwanie wymagań od klienta, analiza dokumentacji, obowiązujących norm i przepisów (ang. Requirements Elicitation). W kolejnym etapie - SYS.2 następuje specyfikacja wymagań systemowych, czyli identyfikacja i dokumentacja wymagań funkcjonalnych i niefunkcjonalnych, które z kolei są informacjami wejściowymi dla poziomu kwalifikacyjnych testów systemowych - SYS.5. Identyfikowane i analizowane są ryzyka związane z bezpieczeństwem funkcjonalnym oraz określane systemowe wymagania bezpieczeństwa. Kolejny etap — SYS.3 to opracowanie ogólnej struktury systemu — specyfikacja wymagań architektury systemowej, z uwzględnieniem wymagań bezpieczeństwa. Na podstawie wymagań architektury systemowej powstają testy integracyjne systemu - SYS.4. Z kolei SWE.1 to analiza wymagań systemowych i dekompozycja ich do szczegółowych specyfikacji technicznych komponentów oprogramowania. Na podstawie wymagań oprograwmowania powstają testy kwalifikacyjne (opisane na schemacie jako SWE.6), które zostaną wykonane w systemie wbudowanym, w środowisku Hardware in the Loop po zakończeniu fazy implementacji oprogramowania. Ich rola polega na weryfikacji działania systemu w kontekście zgodności z wymaganiami bezpieczeństwa. Na podstawie wymagań oprogramowania tworzona jest architektura oprogramowania (SWE.2), która jest źródłem informacji dla testów integracyjnych, sprawdzających integrację poszczególnych modułów oprogramowania. Zostaną one wykonane w środowisku Processor in the Loop. Ostatnim etapem projektowania oprogramowania jest projektowanie szczegółów implementacji modułów kodu (ang. detail design) - SWE.3. Są one punktem wyjścia dla zaprojektowania testów jednostkowych (SWE.4), które będą wykonywane w środowisku Software in the Loop lub Processor in the Loop po zaimplementowaniu poszczególnych modułów kodu.

2.2.2 ISO 26262: 2018 Road vehicles — Functional safety

Pojazdy samochodowe powinny być wolne od błędów systemowych i przypadkowych awarii - w takich przypadkach zagrożone jest zdrowie i życie ludzkie, jednak ryzyko z tym związane rośnie wraz z rozbudowywaniem funkcjonalności realizowanych przez układy mechatroniczne w pojazdach i systemy wbudowane [78]. Norma ISO 26262 jest międzynarodowym standardem zapewniającym, że systemy wbudowane zostały zaprojektowane, zbudowane i przetestowane z właściwym

poziomem rygoru i spełniają wszystkie konieczne wymagania bezpieczeństwa wdrożone w różnych technologiach budowy układu (mechanicznej, hydraulicznej, pneumatycznej, elektrycznej, elektronicznej) i na różnych etapach rozwojowych [8]. Norma skupia się nie tylko na zagadnieniach związanych z bezpieczeństwem funkcjonalnym, zapewniając, że błędnie działający system nie doprowadzi do zagrożenia zdrowia i życia ludzkiego, ale również na wewnętrznej jakości systemu poprzez właściwe zaprojektowanie, wytrzymałość, łatwość konserwacji i testowalność.

Zgodnie z normą zadaniem bezpiecznego systemu jest zapobiegać zagrożeniom dla ludzkiego zdrowia. ISO stosuje podejście oparte na ryzyku do zarządzania potencjalnymi szkodami na podstawie trzech czynników:

- dotkliwość (ang. severity) - określa jak poważna może być potencjalna szkoda,
- narażenie na szkodę (ang. exposure) - określa jak prawdopodobne jest wystąpienie szkody,
- sterowalność (ang. controllability) - określa czy system jest w stanie uniknąć określonej szkody.

Norma ISO 26262 organizuje ryzyko, przypisując je do czterech poziomów integralności bezpieczeństwa samochodowego (ang. Automotive Safety Integrity Levels) popularnie określanymi jako ASIL. Poziom A jest poziomem najniższym, a poziom D poziomem najwyższym. Sam dokument normy składa się z dwunastu części, zorganizowanych hierarchicznie. Rozdział 1 definiuje język normy, określa terminy i skróty w niej użyte, rozdział 2 dotyczy aspektów zarządzania bezpieczeństwem funkcjonalnym, natomiast kolejne części (3-7) dedykowane są poszczególnym fazom rozwoju produktu, tj. koncepcji, rozwoju na poziomie systemowym, sprzętu, oprogramowania oraz produkcji i eksploatacji. Rozdział 8 skupia się na procesach wspierających, które odgrywają rolę w całym cyklu życia projektu. Część 9 dotyczy analizy ukierunkowanej poziom integralności bezpieczeństwa samochodowego (ASIL). Z kolei rozdziały 10 i 11 określają wytyczne dotyczące stosowania normy ISO 26262 oraz zastosowania jej w kontekście elementów półprzewodnikowych. Ostatni, dwunasty rozdział, odnosi się do motocykli w kontekście normy.

Norma ISO 26262 wyraźnie rozgranicza inżynierię systemową, rozwoju oprogramowania i sprzętu. W kontekście oprogramowania wbudowanego najbardziej interesujący jest rozdział 6. Rozdział ten precyzuje proces tworzenia oprogramowania w ramach projektu, umieszczając go w tzw. V-modelu przedstawionego na rysunku 2.10. Pokazuje w sposób strukturalny kolejne etapy tworzenia oprogramowania oraz ich relacje z pozostałymi poziomami. W kontekście omawianej normy zostały wyodrębnione i wyróżnione specyficzne etapy procesu testowania, które stanowią kluczowe elementy analizowanego obszaru:

- Testy jednostkowe oprogramowania (ang. software unit testing) - SWE.4,
- Testy integracji oprogramowania (ang. software integration testing) - SWE.5,
- Weryfikacja wymogów bezpieczeństwa oprogramowania - w standardzie ASPICE zwane testami kwalifikacyjnymi (ang. qualificaiton testing) - SWE.6.

W zależności od poziomu ASIL norma zaleca stosowanie określonych metod testowania, metod tworzenia przypadków testowych oraz technik pomiaru pokrycia testami.

2.2.3 ISO 21448:2022 Road vehicles — Safety of the intended functionality

Dla branży motoryzacyjnej, zapewnienie bezpieczeństwa podczas użytkowania pojazdów jest priorytetem. Zainteresowanie producentów pojazdów elementami autonomicznej jazdy spowodowało istotny postęp w implementacji zaawansowanych funkcji opierających się na analizie danych z różnych sensorów, kamer i radarów. Dane w ogromnych ilościach są przetwarzane za pomocą złożonych algorytmów, w wyniku których sterowanie jest realizowane przez zaawansowane układy mechatroniczne. Implementacja systemów ADAS (Advanced Driver Assistance System), zaklasyfikowanych jako poziom 2 w terminologii SAE [79], w masowej produkcji spowodowała potrzebę zbadania kwestii bezpieczeństwa w szerszym kontekście niż bezpieczeństwo

funkcjonalne definiowane przez normę ISO 26262. Do tej pory, w ramach wypełniania wytycznych normy ISO 26262, analiza i ocena ryzyka związane z nieprawidłowym działaniem elementów i systematycznymi awariami nie obejmowały analizy potencjalnie ryzykownego zachowania systemu, wynikającego z odpowiedzi na pojawiające się zagrożenia środowiskowe. Norma ISO 26262 nie wymagała od producentów systemów wbudowanych analizy ryzykownych zachowań systemu w odpowiedzi na zagrożenia występujące w środowisku zewnętrznym. Rosnące możliwości funkcjonalne pojazdów, klasyfikowanych na poziomach jazdy autonomicznej wyższych niż poziom pierwszy, ujawniły lukę w regulacjach związaną z bezpieczeństwem pojazdów wyposażonych we wspomniane funkcje, a których stabilność może być narażona na nieoczekiwane zmiany w otoczeniu. W związku z tym, w roku 2019, wprowadzono normę ISO 21448, zatytułowaną "Safety of the Intended Functionality"(SOTIF), która stanowi komplementarny zestaw regulacji dążących do zapewnienia bezpiecznego funkcjonowania pojazdów, niezależnie od kontekstu, w którym są używane [80]. ISO 21448 obejmuje takie aspekty jak ograniczenia wydajności podzespołów samochodu oraz nieoczekiwane zmiany w środowisku. Aby sprostać wymaganiom normy, producenci muszą przeprowadzić bardzo dużą liczbę symulacji oraz wykorzystać uczenie maszynowe i sztuczną inteligencję do przetwarzania danych, które pomagają im przewidzieć, jak zareagują pojazdy na złożone, rzeczywiste scenariusze ryzykownych sytuacji [81].

2.2.4 Standard ASPICE

ASPICE (ang. Automotive Software Process Improvement and Capability Determination) jest to międzynarodowy standard oceny procesów wytwarzania oprogramowania w branży motoryzacyjnej. Jego istota leży w usprawnianiu procesów oraz ocenie zdolności organizacji w zakresie tworzenia oprogramowania. Dokument ten opisuje wzorcowy model procesów stosowanych podczas wytwarzania oprogramowania wraz z powiązanymi procesami biznesowymi oraz wytyczne dotyczące oceny zgodności procesów stosowanych w firmie z opisanymi w standardzie. ASPICE umożliwia standaryzowaną ocenę zdolności organizacji z branży motoryzacyjnej do efektywnego dostarczania niezawodnego oprogramowania. ASPICE obejmuje kompletny cykl życia oprogramowania, począwszy od zarządzania wymaganiami, poprzez procesy tworzenia oprogramowania, jego weryfikacji i testowania, aż po konserwację. Poprzez identyfikowanie i eliminowanie nieefektywności, poprawianie komunikacji oraz redukcję kosztów, ASPICE wspiera ciągłe doskonalenie procesów i produktywność organizacji. Główni producenci branży motoryzacyjnej, tacy jak Audi, BMW, Daimler czy Ford, oceniają swoich dostawców systemów wbudowanych na podstawie osiągniętego poziomu ASPICE i bardzo często żądają oceny procesów przez zewnętrznych ekspertów przed odebraniem produktu od producenta (ang. assesment). W takim przypadku dojrzałość procesów (ang. Capability Levels) w przedsiębiorstwie czy projekcie oceniana jest w skali 6 poziomowej - od 0 do 5 dla każdego procesu spośród trzech kategorii: podstawowych, organizacyjnych, wspomagających. W kategorii procesów podstawowych zawierają się procesy nabywcze (ang. acquisition), dostarczania (ang. supply), inżynierii systemu (ang. systems engineering) oraz grupa procesów inżynierii oprogramowania (ang. software engineering). Każda z grup zawiera kilka do kilkunastu procesów, z których każdy jest oceniany. Norma definiuje również zestaw atrybutów procesów, tak zwanych „dobrych praktyk" (ang. best practices) dla każdego procesu [82]. Atrybuty procesów są oceniane w skali osiągnięcia:

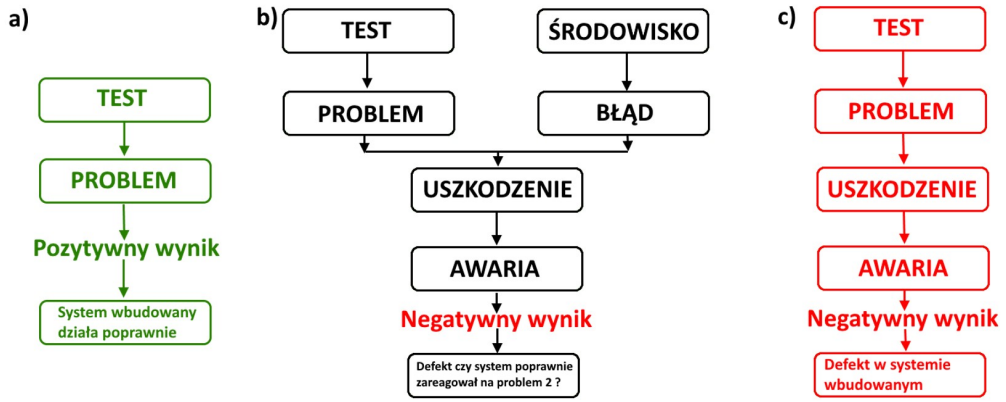
- **N** (ang. not achieved) - nieosiągnięty (0 - 15%),
- **P** (ang. partialy achieved) - częściowo osiągnięty (15 - 50%),
- **L** (ang. largely achieved) - osiągnięty w znacznej części (50 - 85%),
- **F** (ang. fully achieved) - w pełni osiągnięty (85 - 100%).

Ocena ogólnej zdolności organizacyjnej do wykonywania procesu i osiągania z niego dobrych rezultatów opiera się o łączną ocenę poszczególnych aspektów wszystkich procesów. Przykła-

dowo poziom 2 oznacza, że proces w przedsiębiorstwie jest zarządzany — czyli planowany, monitorowany i dostosowywany, a produkty jego pracy są prawidłowo wykonane, kontrolowane i utrzymywane [9].

2.2.5 IEEE Std 1044-2009 IEEE Standard Classification for Software Anomalies

W celu ustanowienia spójnego nazewnictwa, umożliwiającego skuteczną komunikację na temat odchyleń w oprogramowaniu oraz analizy danych dotyczących defektów i awarii oprogramowania organizacja IEEE ogłosiła standard klasyfikacji anomalii programowych. Zdefiniowany został podstawowy zestaw atrybutów służących do klasyfikacji awarii oraz wad w oprogramowaniu. Podkreślono również, że w kontekście konkretnych zastosowań biznesowych, oprogramowania testowego, systemów wbudowanych i innych specyficznych obszarów znaczenie mogą mieć inne cechy. Poza tym każda faza cyklu życia projektu, produktu lub systemu wymaga zaadoptowania atrybutów klasyfikacji [49]. Standard precyzuje następujące pojęcia: defekt, błąd, awaria, usterka, problem. Należy podkreślić, że w kontekście anomalii programowych nie są one synonimami i nie mogą być używane zamiennie. Defektem (ang. defect) jest niedoskonałość lub wada w produkcie (systemie, oprogramowaniu), jeśli jest wynikiem niespełnionych wymagań lub specyfikacji. Objawia się on w postaci uszkodzenia (ang. fault), czyli niewłaściwego, błędnego działania systemu, co może prowadzić do awarii (ang. failure) nie tylko danego systemu wbudowanego, ale też innych komponentów pojazdu. W procesie testowania oprogramowania wbudowanego sprawdzana jest jego reakcja na negatywną sytuację, trudność, z którą testowany system powinien sobie poradzić (ang. problem). Negatywny wynik testu świadczy o defekcie w testowanym systemie wbudowanym. Jeżeli w testowanym systemie pojawi się błąd (ang. error), czyli niewłaściwe działanie człowieka lub środowiska, w którym funkcjonuje system, może on doprowadzić do negatywnego wyniku testu, mimo iż system wbudowany na problem zareagował poprawnie [83]. Na Rys. 2.11 przedstawiono relacje między terminami odnoszącymi się do nieprawidłowości w działaniu oprogramowania a rezultatami przeprowadzanych testów.



Rys. 2.11. Zależności między pojęciami opisującymi nieprawidłowości w działaniu oprogramowania [83]

Na rysunku 2.11a pokazano sytuację, gdzie analizowany system wbudowany działa poprawnie, a problem został skutecznie rozwiązany zgodnie z wymaganiami umieszczonymi w specyfikacji systemu wbudowanego — rezultat testu jest pozytywny. Z kolei na rysunku 2.11c pokazano sytuację odwrotną — testowany system wbudowany nie zareagował na sytuację problematyczną zgodnie z wymaganiami, w wyniku czego w systemie pojawiło się uszkodzenie, co objawiło się niewłaściwym zachowaniem systemu i w prawdziwym pojeździe mogłoby doprowadzić do awa-

rii. Rezultat testu w takim przypadku jest negatywny i powinien zostać zaraportowany defekt. Jednak doświadczenie pokazuje, że negatywny rezultat testu może być też wynikiem reakcji systemu wbudowanego na błąd występujący w środowisku testowym, nakładającym się na problem, którego rozwiązanie miało być testowane 2.11b. Błąd ten może wynikać zarówno z ludzkiej interakcji, jak i zjawisk występujących w środowisku testowym, niewykazujących bezpośrednich relacji z testowaną funkcjonalnością. System wbudowany, reagując na błąd, nie zachowa się tak, jak zaplanowano w scenariuszu testowym jako reakcja na problem. W przypadku kolejnego wykonania tego testu błąd w środowisku może już być nieobecny — wynik testu tym razem będzie pozytywny. Opisane zjawisko nazywa się migoczącymi rezultatami testów. Wymaga ono dogłębnej, kosztownej i czasochłonnej analizy.

W ostatnich latach naukowcy zwiększyli swoje wysiłki w zbadania problemu niedeterministycznych awarii testów, czyli testów migoczących. Badacze zidentyfikowali trzy główne przyczyny błędów: błędy semantyczne, nieudane testy i błędy środowiskowe [84]. Luo i inni [85] przedstawili dziesięć głównych przyczyn nieregularności testów (np. asynchroniczne oczekiwanie, współbieżność, zależność od kolejności testów). Natomiast Gao i inni [86] przeprowadzili badanie, które wykazało, że odtwarzanie testów migoczących może być trudne i czasochłonne.

Wykrywanie migoczących testów jest istotnym wyzwaniem, ponieważ wiążą się one z wieloma problemami:

- Zwiększają koszty debugowania. Załóżmy, że inżynier nie wie, że test jest migoczący. Może on spędzić mnóstwo czasu na debugowaniu tylko po to, aby odkryć, że zaobserwowane niepowodzenie testu nie jest spowodowane ostatnimi zmianami w oprogramowaniu. Jak wyjaśniają Gruber i inni [87], potrzeba około 170 uruchomień przypadku testowego, aby z całą pewnością stwierdzić, czy test jest wadliwy. Z kolei Parry i inni wskazują [88], że ponowne uruchamianie wykrywa migoczące testy tylko w 40% przypadków.
- Podważają zaufanie do testów. Niespójności w wynikach testów mogą spowodować, utratę wiary w testy i sens procesu testowania.
- Mogą spowodować ukrywanie prawdziwych błędów. Jeśli migoczący test zawodzi losowo, testerzy mają tendencję do ignorowania jego niepowodzeń, a tym samym mogą przeoczyć prawdziwe defekty w oprogramowaniu [85].

Zapobieganie migotaniu jest skomplikowane, a czasami nawet niemożliwe, ponieważ nie zawsze zależy od przypadku testowego, ale od zachowania środowiska, które nie jest przewidywalne i nie można kontrolować jego wszystkich aspektów. Pomimo faktu, że debugowanie testów migoczących jest uważane za czasochłonne, większość inżynierów uważa, że znalezienie pierwotnej przyczyny jest istotne, szczególnie jeżeli możliwe jest jej naprawienie [89].

2.3 Inżynieria oparta na modelach

Rynek motoryzacyjny przechodzi dynamiczne zmiany. Klienci oczekują innowacji, nowoczesności i szybko wprowadzanych zmian. Podejście do projektowania i testowania systemów wbudowanych wyposażonych w nowatorskie funkcje ewoluowało wraz ze zmianami na rynku motoryzacyjnym. Inżynieria oparta na modelach oferuje możliwość większej elastyczności w projektowaniu i testowaniu, już na wczesnych etapach życia projektu, jeszcze zanim powstaną prototypy produktów. Konieczność przedstawiania zjawisk i obiektów za pomocą modeli jest powszechnie akceptowana i stanowi istotny obszar zarówno w dziedzinie nauki, jak i inżynierii. Analiza układów mechanicznych, procesów technologicznych oraz innych zjawisk występujących w rzeczywistym świecie staje się możliwa dzięki wykorzystaniu odpowiedniego aparatu matematycznego opartego na wcześniej zdefiniowanych modelach matematycznych tych procesów [90]. Inżynieria oparta na modelach (ang. Model-Based Engineering - MBE) dąży do zwiększenia wydajności

procesu inżynieryjnego poprzez wykorzystanie modeli jako kluczowych elementów w procesie rozwoju[91].

2.3.1 Projektowanie oparte na modelach

Rywalizacja w przemyśle samochodowym doprowadziła do skrócenia czasu rozwoju nowych modeli pojazdów. Z kolei złożoność funkcji realizowanych przez systemy wbudowane oraz wysokie wymagania bezpieczeństwa powodują, że na rozwój systemów wbudowanych potrzeba relatywnie dużo czasu. Aby sprostać tym wyzwaniom, producenci i dostawcy samochodów dokonują zmiany paradygmatu rozwoju oprogramowania z ręcznego kodowania na rozwój oparty na modelach (ang. Model Based Design) [92]. Jest to proces napędzany przez dwie siły. Z jednej strony należy połączyć ewolucyjny rozwój systemów motoryzacyjnych z ich integracją ze znaczną liczbą funkcji z poprzednich wersji systemów, a z drugiej rozwój ten powinien być niezależny od platformy, co znacznie zmniejsza liczbę przeprojektowań / konserwacji spowodowanych szybko zmieniającymi się generacjami sprzętu. Stosowanie podejścia opartego na modelach umożliwia przesunięcie punktu ciężkości procesu rozwoju na wczesne fazy, wspierając inżynierię systemów motoryzacyjnych opartą na funkcjach, a nie na kodzie. Podejście takie obiecuje znaczny wzrost wydajności, poprawę jakości i oszczędność kosztów. Z drugiej strony, niesie to ze sobą wyzwania, w postaci konieczności przeprojektowania procesu, co wpływa na wymagane zasoby i strukturę organizacyjną. Konieczne również są inwestycje w narzędzia i szkolenia pracowników [93]. Modele umożliwiają zrozumienie charakterystyki efektywności potencjalnych opcji implementacji na poziomie systemu oraz są skutecznym narzędziem dokumentacji i komunikacji. W rezultacie prowadzą do skrócenia i bardziej przewidywalnego czasu rozwoju oraz lepiej kontrolowanej jakości produktu [94].

Modelowanie w procesie rozwoju systemu wbudowanego może być stosowane na wielu etapach i w różnym celu [95]:

- Zrozumienie domeny aplikacji: analiza wymagań klienta, dokładna identyfikacja głównych działań i ich relacji pomagają wyjaśnić wiele kwestii, które na początku mogą nie być od razu rozpoznane.
- Projektowanie architektury systemu — tworzenie diagramów klas, diagramów komponentów czy diagramów strukturalnych umożliwia określenie struktury systemu oraz relacji pomiędzy jego różnymi elementami.
- Projektowanie detali (ang. detail design) - modelowanie na tym etapie pozwala na uszczegółowienie diagramów klas, komponentów, stanów czy sekwencji, co ułatwia implementację konkretnych funkcjonalności systemu.
- Weryfikacja i testowanie — symulacje i testy wydajnościowe mogą być przeprowadzane na modelach przed przystąpieniem do implementacji, co pozwala wykryć błędy na wczesnym etapie. Na etapie testów akceptacyjnych modelowanie pozwala na zbudowanie środowiska testowego możliwie jak najbardziej zbliżonego do rzeczywistości.
- Dokumentacja — modele są doskonałą formą dokumentacji projektowej. Dzięki temu, że są precyzyjne i czytelne, ułatwiają komunikację między członkami zespołu projektowego i mogą być punktem odniesienia dla przyszłych modyfikacji i rozbudowy systemów.

2.3.2 Testowanie oparte na modelach

Z pojęciem projektowania opartego na modelu nierozłącznie wiąże się temat testowania na poziomie modelu (ang. Model Based Testing). Tak zwane podejście Model-in-the-Loop for Embedded System Test (MiLEST) wykorzystuje przede wszystkim systematyczną, ustrukturyzowaną, powtarzalną i abstrakcyjną specyfikację testową i koncentruje się na automatyzacji procesu testowania. Paradygmat zorientowany na cechy sygnału pozwala na abstrakcyjny opis sygnału

i rozwiązuje problemy brakujących przepływów sygnału referencyjnego, a także systematycznego wyboru danych testowych. Liczne cechy sygnału są identyfikowane, podczas gdy predefiniowane wzorce testowe pomagają zbudować specyfikację testową. Testowanie rozpoczyna się w fazie wymagań i schodzi do poziomu wykonania testu [96]. Dzięki możliwości lepszej integracji rozwoju systemu i testów redukuje się koszty i niweluje ryzyka. Testowanie oparte na modelach, szczególnie w przypadkach heterogeniczności, polegającej na tym, że komponenty i podsystemy, są częściowo opracowywane w oprogramowaniu, a częściowo w sprzęcie, często przez różnych dostawców, daje możliwość rozpoczęcia testowania na wczesnych etapach projektowania i przyczynia się do wysokiego pokrycia testami [97].

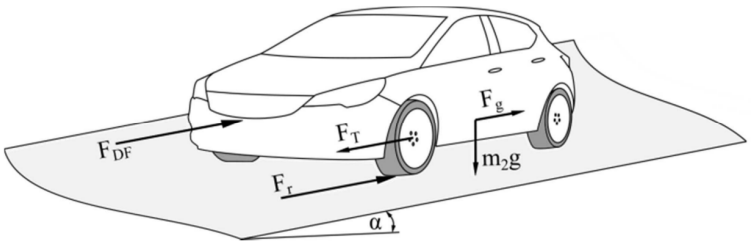
2.3.3 Model pojazdu samochodowego

Tworzenie modelu umożliwia lepsze zrozumienie, przewidywanie, optymalizację i projektowanie systemów oraz procesów, co przyczynia się do efektywności i skuteczności działań w wielu dziedzinach nauki i inżynierii.

W pierwszym kroku opisuje się istotę zjawiska – powstaje model fenomenologiczny. Następnie opisuje się zjawisko za pomocą praw fizyki i ustala parametry mające wpływ na jego przebieg – definiuje się model fizyczny. Stworzenie modelu matematycznego, czyli dokładne opisanie rzeczywistych zjawisk i procesów zachodzących w obiekcie przy pomocy równań matematycznych (równań różnicowych, równań różniczkowych, równań całkowych) umożliwia implementację modelu w postaci programu komputerowego. Bardzo trudne jest idealne odwzorowanie rzeczywistego świata w modelu matematycznym. Zgodnie z zasadami modelowania, model powinien w dostateczny sposób odzwierciedlać zachowania rzeczywistego zjawiska lub obiektu. Projektując model, należy mieć świadomość, że stopień szczegółowości modelu wpływa na jego dokładność oraz wiarygodność opisu danego zjawiska. Równocześnie nadmierna rozbudowa modelu jest niemożliwa oraz ekonomicznie nieuzasadniona. Z tego powodu każdy model jest swego rodzaju kompromisem pomiędzy stopniem złożoności a precyzją opisu.

W modelu matematycznym użytym w niniejszej rozprawie uwzględnione zostaną siły oporu i strat na drodze przekazywania energii, a więc składowe sił napędowych i oporów ruchu działających w kierunku osi wzdłużnej pojazdu. Teoria dotycząca ruchu pojazdów jest szeroko znana i wielokrotnie omawiana w literaturze [98–102]. Poniżej, przedstawiono propozycję modelu matematycznego pojazdu wraz z zależnościami i oznaczeniami przyjętymi w rozważaniach. Szczegółowy opis omawianych zagadnień można znaleźć między innymi w cytowanych pozycjach literatury.

Na rysunku 2.12 przedstawiono najistotniejsze siły oddziałujące pojazd poruszający się po drodze o kącie nachylenia α.



Rys. 2.12. Siły oddziałujące na pojazd [101]

Równanie ruchu pojazdu, wynikające z drugiej zasady dynamiki Newtona, stanowi punkt wyjścia dla proponowanego modelu pojazdu. Można je zapisać jako sumę sił napędowych i sił

oporu równoległych do nawierzchni jezdni jako:

$$m\frac{d\upsilon(t)}{dt} = F_T(t) - F_{DF}(t) - F_r(t) - F_g(t) \tag{2.1}$$

gdzie:

F_T - siła trakcyjna pojazdu

F_{DF} - siła związana z oporem aerodynamicznym (siła oporu powietrza)

F_r - siła oporu toczenia

F_g - siła oporu wzniesienia

Od siły trakcyjnej pojazdu F_T należy odjąć siłę związaną z oporem aerodynamicznym F_{DF}, siłę związaną z całkowitymi oporami tarcia F_r oraz siłę oporu wzniesienia F_g. Przyjęto założenie upraszczające, że pojazd porusza się ruchem prostoliniowym, więc opory skrętu w rozważaniach zostały pominięte. Siły oporów, które występujące w powyższych równaniach są uzależnione od licznych czynników i parametrów, z których wiele charakteryzuje się nieliniowością.

Siła oporu aerodynamicznego (siła oporu powietrza)

Środowisko, w którym porusza się pojazd, jest źródłem siły oporu zależnej od gęstości środowiska, wynikającej z sił tarcia i oddziaływań środowiska (np. wiatru).

$$F_{DF} = \frac{1}{2}\rho C_d A(\upsilon - \upsilon_w)^2 \tag{2.2}$$

Siła F_{DF}, związana z oporem aerodynamicznym, uwzględnia prędkość pojazdu v, prędkość wiatru v_w, współczynnik oporów powietrza C_d, powierzchnię czołową pojazdu A oraz gęstość powietrza ρ wyrażoną wzorem:

$$\rho = \frac{p}{r \cdot T} \tag{2.3}$$

gdzie:

p — ciśnienie,

T — temperatura w skali bezwzględnej,

r — indywidualna stała gazowa dla powietrza.

Dla suchego powietrza:

$$r = 287,05\frac{J}{kg \cdot K} \tag{2.4}$$

a zatem w standardowych warunkach przyjmowanych w aerostatyce i aerodynamice (15°C i 101 325,15 Pa) suche powietrze ma gęstość $\rho_{ISA} = 1,225kg/m^3$.

Siła oporu toczenia

Całkowity opór toczenia w ruchu prostoliniowym stanowi sumę oporów toczenia wszystkich kół pojazdu:

$$F_r = \sum_i F_{r_i} \tag{2.5}$$

gdzie indeks i oznacza siłę oporów toczenia i-tej osi pojazdu.

Zważywszy na podobną postać konstrukcyjną kół osi przedniej i tylnej, identyczne ogumienie

i ciśnienie w oponach przyjmujemy identyczny współczynnik oporu toczenia dla każdego koła. Należy pamiętać, że zależy on od rodzaju nawierzchni, rodzaju ogumienia i ciśnienia w oponie. Zależy również od prędkości jazdy i rośnie wraz z jej zwiększaniem wskutek odkształcenia opony. Do modelowania pojazdu na potrzeby testowania systemów wbudowanych można przyjąć, że ma on stałą wartość. Uwzględniając powyższe założenia, można całkowity opór toczenia przedstawić jako:

$$F_r = f_r m_2 g cos(\alpha) \tag{2.6}$$

gdzie:

m_2 – masa pojazdu,

g – przyspieszenie ziemskie,

f_r – współczynnik oporu toczenia.

Siła oporu wzniesienia

Siła oporu wzniesienia wynika z siły grawitacji, w której uwzględniono kąt nachylenia terenu:

$$F_g = m_2 g sin(\alpha) \tag{2.7}$$

gdzie:

m_2 – masa pojazdu,

g – przyspieszenie ziemskie,

α – kąt nachylenia terenu [101].

Siła trakcyjna pojazdu

$$F_T = (M_e(\omega_m) - \Delta M)\frac{2i}{d_w} \tag{2.8}$$

gdzie:

M_e – to moment obrotowy silnika,

ΔM – to moment wprawiający w ruch przekładnię,

d_w – to średnica koła,

i – to przełożenie przekładni mechanicznej,

ω_m – to prędkość obrotowa silnika.

W modelu pojazdu rozważanego w niniejszej rozprawie wykorzystuje się system silników synchronicznych wzbudzanych magnesami trwałymi (ang. permanent magnet synchronous motor - PSM), zasilany i sterowany przez układ energoelektroniczny trójfazowym napięciem przemiennym. Napięcie przemienne o częstotliwości oscylującej wokół punktu zerowego umożliwia kontrolę obrotów silnika. Kluczowym komponentem regulującym prędkość obrotową wirnika jest falownik impulsowy, który przekształca prąd stały z akumulatora o napięciu 800 woltów na prąd przemienny i doprowadza go do silnika. Układ energoelektroniczny jest zintegrowany bezpośrednio z napędem, co zapewnia krótkie, wydajne i lżejsze połączenie silnika z układem czujników. Warto również zwrócić uwagę na przekładnię planetarną napędu przedniej osi, która posiada jedno przełożenie wynoszące 1:8. Dzięki temu moment obrotowy kół osiąga wartość do 3000 Nm. W konstrukcji silnika PSM, w stojanie znajdują się aktywne elektromagnesy, podczas gdy w wirniku zlokalizowane są pasywne magnesy trwałe wykonane ze stopów neodymu, żelaza i boru. W procesie produkcji poddawane są one trwałemu namagnesowaniu przy użyciu silnego,

kierunkowego pola magnetycznego. Magnesy te odgrywają istotną rolę w procesie rekuperacji energii, gdyż podczas hamowania rekuperacyjnego silniki elektryczne działają jako generatory. Magnesy indukują wówczas napięcie i prąd w uzwojeniach stojana, co pozwala na efektywny odzysk energii.

Zjawiska zachodzące w silniku nie są istotne z punktu widzenia niniejszej rozprawy. Dla realizacji pracy istotne są takie wielkości jak moc mechaniczna (czyli moment - M_e i prędkość obrotowa na wale silnika - ω_m).
Równanie obwodu elektrycznego n-tej fazy dla rozważanego silnika ma postać:

$$R_n i_n + L_n \frac{di_n}{dt} + e_n = u_n \tag{2.9}$$

gdzie:

R_n – rezystancja fazy stojana,

L_n – indukcyjność zastępcza fazy uwzględniająca indukcyjność własną i wzajemną,

u_n i i_n – napięcie i natężenie prądu n-tej fazy,

e_n – siła elektromotoryczna indukowana w uzwojeniu n-tej fazy

Zakładając, że prąd płynie zawsze przez dwa, spośród trzech uzwojeń stojana uzyskujemy:

$$R = 2R_n \tag{2.10}$$

$$L = 2L_n \tag{2.11}$$

$$E = 2e_n \tag{2.12}$$

Równanie obwodu przyjmuje postać [103]:

$$Ri_z + L\frac{di_z}{dt} + E = U_z \tag{2.13}$$

Zakładając, że strumień magnetyczny stojana ma stałą wartość, moment obrotowy M_e jest proporcjonalny do natężenia prądu płynącego przez uzwojenia wirnika i tzw. stałej mechanicznej k_m zależnej m.in. od liczby zwojów w uzwojeniach wirnika oraz strumienia magnetycznego wirnika.

$$M_e = k_m i_z \tag{2.14}$$

Gdy wirnik wykonuje ruch obrotowy, w uzwojeniu jest indukowana siła elektromotoryczna indukcji E, która jest wprost proporcjonalna do prędkości obrotowej silnika ω_m oraz stałej elektrycznej K_e, zależnej m.in. od liczby zwojów w uzwojeniach wirnika.

$$E = K_e \omega_m \tag{2.15}$$

2.4 Testowanie układów mechatronicznych w samochodzie

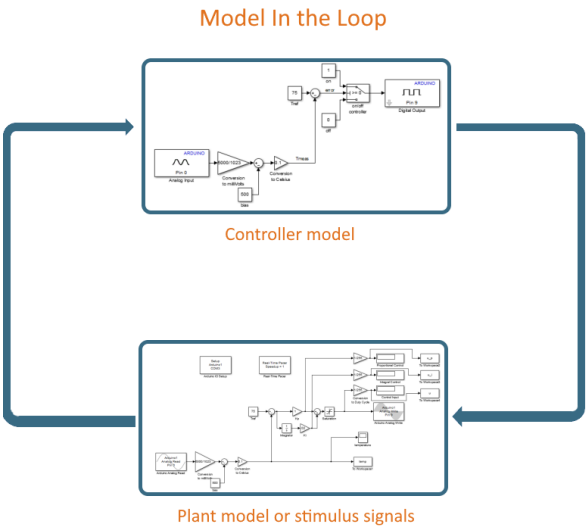
Wadliwy samochodowy system wbudowany może skutkować zagrożeniem bezpieczeństwa ludzi. Dlatego tak ważne jest wykrywanie wszelkich defektów w oprogramowaniu wbudowanym za pomocą różnych procesów testowych [11]. W procesie tworzenia oprogramowania wbudowanego weryfikacja i walidacja wszystkiego, co zostało zaprojektowane i zaimplementowane tworzy prawą

gałąź V-modelu (Rys. 2.10). Każdy etap projektowania i tworzenia systemu wbudowanego powinien zostać przetestowany w celu weryfikacji poprawności i niezawodności realizacji założonych funkcji oprogramowania. Testowanie jest najbardziej czasochłonną i kosztowną częścią procesu rozwoju systemu wbudowanego [104]. Zaczynając od testów jednostkowych, a kończąc na systemowych, wykrywanie defektów bądź błędów w założeniach, jest kluczowe dla zapewnienia jakości oprogramowania, a co za tym idzie bezpieczeństwa jego użytkowników. Zgodnie ze standardem ASPICE i normą ISO26262 strategie, metody i praktyki testowania powinny być precyzyjnie określone dla wszystkich etapów procesu inżynierii systemów. Właściwe i dokładne zaplanowanie, a później przeprowadzenie procesów testowych ma fundamentalne znaczenie w kontekście zagwarantowania, że oprogramowanie wbudowane działa bezbłędnie, zgodnie z oczekiwaniami oraz spełnia ściśle określone wymagania. Fundamentalnym celem procesu testowania jest rozpoznanie jak największej liczby defektów oprogramowania na możliwie wczesnym etapie rozwoju. Im wcześniej defekt zostaje wykryty, tym mniejsze są koszty jego analizy i naprawy. Ogromne znaczenie w poprawie efektywności i jakości procesu testowania ma automatyzacja i implementacja procesu tzw. Testowania Ciągłego (ang. Continous Testing), które na poziomach SWE.4 i SWE.5 jest stosunkowo prosto zaimplementować, również uruchamiając skrypty testowe na docelowym mikrokontrolerze. Jednak na poziomie SWE.6, który koncentruje się na sprawdzeniu integracji sprzętu z oprogramowaniem wbudowanym, niestabilne środowisko testowe może negatywnie wpłynąć na wyniki testów. Na tym etapie niemożliwe jest przetestowanie pojedynczych, odizolowanych funkcjonalności systemu.

2.4.1 Środowiska testowe

Zagwarantowanie prawidłowego działania urządzeń mechatronicznych składających się na pojazd jest bardzo złożone, a ich oczekiwane działanie musi być zapewnione w bardzo trudnych warunkach środowiskowych. W rzeczywistości pojazdy są narażone na wibracje, hałas, ekstremalne temperatury i pola elektromagnetyczne, które mogą wpływać na komponenty elektroniczne i powodować ich degradację [105]. W celu skutecznego, efektywnego i kompleksowego przetestowania oprogramowanie wbudowanego zalecaną praktyką jest, aby weryfikacja oraz walidacja jego funkcjonalności i jakości rozpoczęła się na jak najwcześniejszym etapie. Zgodnie z postanowieniami normy ISO26262 implementacja automatycznego testowania regresji na całym każdym etapie procesu tworzenia oprogramowania wbudowanego jest zalecana. Wspomniana norma rekomenduje środowiska testowe właściwe dla poszczególnych poziomów testowania. W początkowych etapach rozwoju zaleca stosowanie czystych środowisk symulacyjnych (MIL: Model in the Loop, SIL: Software in the Loop). Rzeczywisty system wbudowany na tym etapie testowania zazwyczaj nie jest jeszcze dostępny. Natomiast środowisko testowe dla testów weryfikujących wymogi bezpieczeństwa oprogramowania (zgodnie z ASPICE — testy kwalifikacyjne oprogramowania - SWE.6) powinno być jak najbardziej zbliżone do rzeczywistego [8]. Symulacja Vehicle in the Loop generuje wysokie koszty sprzętowe, a także utrudnione w niej jest wyzwalanie błędów w celu sprawdzenia reakcji oprogramowania wbudowanego na nie. W związku z tym, na poziomie SWE.6 powszechnie stosowana jest symulacja Hardware in the Loop (HIL), która umożliwia testowanie oprogramowania wbudowanego poprzez symulację rzeczywistej jednostki sterującej w interakcji ze zdefiniowanym środowiskiem symulacyjnym. Dzięki temu możliwe jest bardziej precyzyjne i realistyczne sprawdzanie funkcjonalności oraz reakcji oprogramowania na różne scenariusze testowe [106].

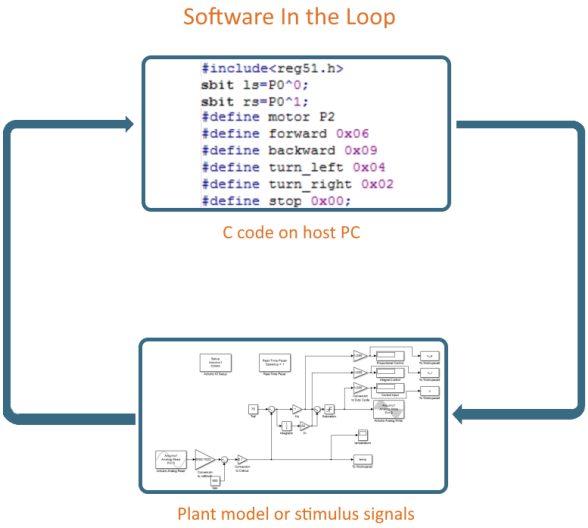
Model in the Loop (MIL) - Model w pętli



Rys. 2.13. Środowisko Model in the Loop [107]

Jest to środowisko, w którym zarówno testowany kontroler, jak i środowisko są symulowane w celu zweryfikowania funkcjonalności w ramach modelowania bez żadnych fizycznych komponentów sprzętowych. Testowanie MiL odbywa się na początkowym poziomie integracji systemu w procesie projektowania [108]. Część wirtualna środowiska jest realizowana poprzez symulację komputerową w czasie rzeczywistym, a testowany obiekt jest dostępny jako model [109].

Software in the Loop (SIL) – Oprogramowanie w pętli

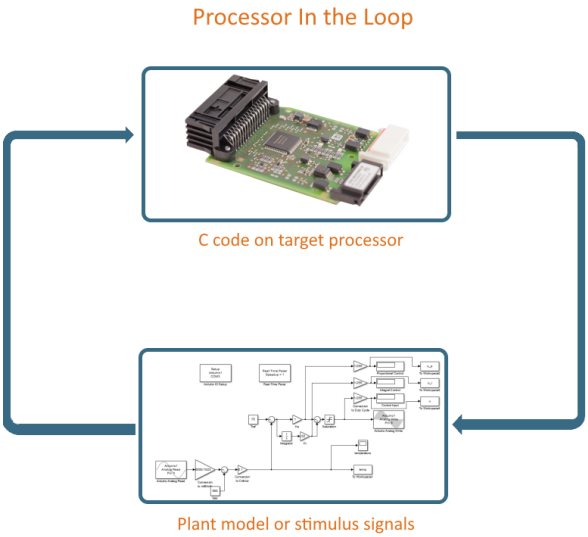


Rys. 2.14. Środowisko Software in the Loop [107]

Symulacja ta opiera się na integracji zoptymalizowanego, skompilowanego kodu źródłowego docelowego kontrolera wbudowanego z matematycznym modelem symulacyjnym. Tego rodzaju

podejście umożliwia weryfikację oprogramowania jeszcze przed etapem tworzenia prototypu sprzętowego, co równocześnie pozwala na wczesne wykrycie ewentualnych defektów na poziomie systemowym. Przed konfiguracją sprzętu i systemu, wirtualne środowisko jest symulowane na komputerze PC, co umożliwia testowanie kodu źródłowego. Przez symulację warstwy wirtualnej możliwe jest obserwowanie zachowań oraz interakcji oprogramowania bez konieczności angażowania docelowego układu sterującego (ECU) [110].

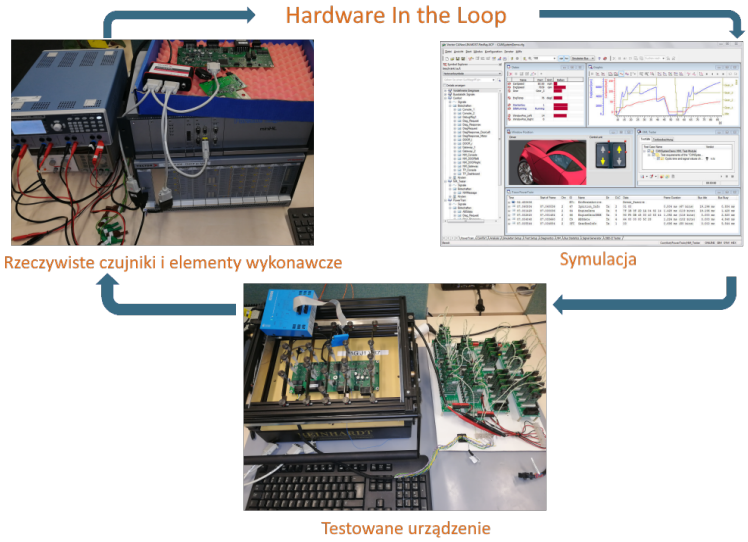
Processor in the Loop (PIL) – Procesor w pętli



Rys. 2.15. Środowisko Processor in the Loop [107]

Technika testowania w środowisku PIL stanowi efektywną metodę umożliwiającą projektantom ocenę działania oprogramowania na dedykowanym kontrolerze. Jedną z kluczowych cech charakteryzujących tę technikę jest możliwość debugowania zarówno samego kontrolera, jak i elementów składowych testowanego systemu, co pozwala na identyfikację i korektę ewentualnych defektów dotyczących ich funkcjonowania. Należy jednak zaznaczyć, że nie jest to środowisko działające w czasie rzeczywistym [111]. Wartości, które w rzeczywistej implementacji byłyby odczytane przez kontroler z fizycznych czujników, są zamiast tego wyznaczane przez narzędzie symulacyjne i dostarczane jako wejście do wewnętrznych algorytmów kontrolujących. Następnie wyniki działania tych algorytmów są ponownie wprowadzane do symulacji, co napędza wirtualne środowisko. Umożliwia to symulację zachowania systemu, eliminując konieczność używania rzeczywistego sprzętu, co pozwala na bardziej elastyczne, powtarzalne i kontrolowane testowanie oprogramowania wbudowanego. Dzięki symulacji w środowisku PIL, testerzy mogą eksplorować różne scenariusze działania systemu, w tym te, które mogą prowadzić do błędów charakterystycznych dla konkretnej platformy sprzętowej (błędy rzutowania danych, przekroczenie zakresu wartości). Dodatkowo możliwość testowania algorytmów sterujących w symulowanym środowisku pozwala na identyfikację potencjalnych problemów związanych z ich wielowątkowym wykonaniem, co przyczynia się do poprawy stabilności oraz wydajności oprogramowania wbudowanego.

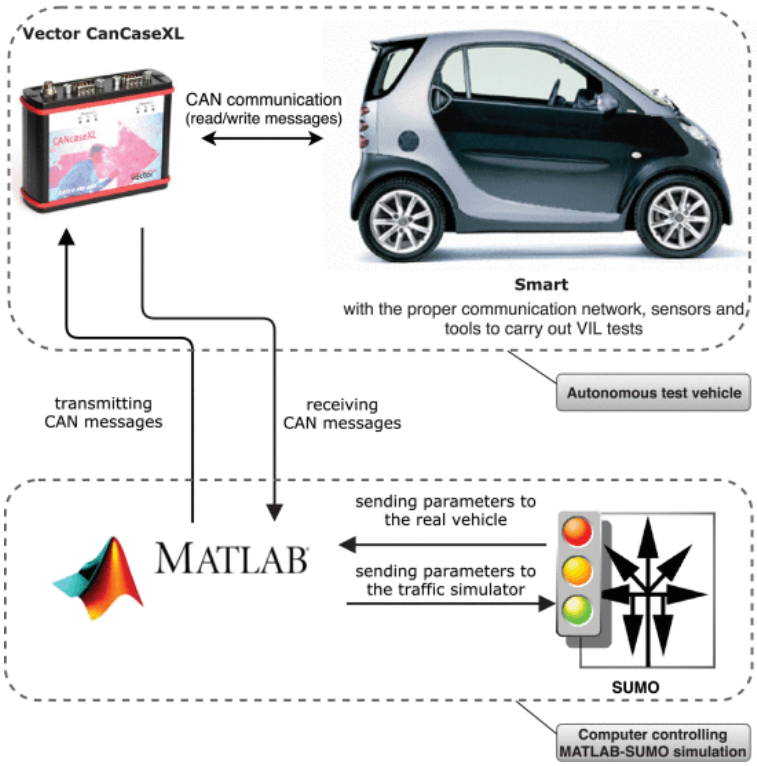
Hardware in the Loop (HIL) – Sprzęt w pętli



Rys. 2.16. Środowisko Hardware in the Loop [72]

Symulacja w środowisku HIL (Hardware in the Loop) umożliwia projektowanie i testowania kompleksowych systemów wbudowanych w czasie rzeczywistym oraz w kontrolowanych i bezpiecznych warunkach, dokładnie oddających środowisko rzeczywistej eksploatacji systemu. Ponieważ symulowanie ekstremalnych warunków pracy na drodze jest bardzo kłopotliwe i może prowadzić do uszkodzenia systemu, funkcjonalność opracowanych systemów wbudowanych jest najpierw weryfikowana i walidowana w laboratorium, zanim zostaną poddane rzeczywistym warunkom. Jest to możliwe dzięki symulacji HIL, nie narażając przy tym bezpieczeństwa i funkcjonalności systemu [112]. W ramach tego podejścia istotne jest dokładne odwzorowanie elektrycznych sygnałów pochodzących zarówno z czujników, jak i elementów wykonawczych. W praktyce, sygnały generowane przez kontroler są przekazywane do systemu testowego, który symuluje rzeczywiste warunki pracy [113]. Zastosowanie środowiska HIL pozwala na przeprowadzanie tysięcy różnych scenariuszy testowych bez konieczności ponoszenia kosztów i czasu związanego z fizycznymi testami. Niewątpliwą zaletą tego środowiska jest możliwość symulowania skomplikowanych układów oraz zastępowania komponentów pojazdu, takich jak silnik, za pomocą symulacji opartej na sprzęcie i oprogramowaniu. Dzięki temu możliwe jest interaktywne oddziaływanie z rzeczywistymi wejściami i wyjściami, co prowadzi do stworzenia wirtualnego odwzorowania, gdzie cały pojazd jest wirtualnie obecny [114].

Vehicle in the Loop (VIL) – Pojazd w pętli



Rys. 2.17. Środowisko Vehicle in the Loop [115]

Istotą tego podejścia jest stworzenie środowiska, gdzie symulacyjne oprogramowanie ruchu drogowego jest w stanie emulować zachowanie pojazdów autonomicznych w czasie rzeczywistym w ruchu wirtualnym, z wieloma symulowanymi pojazdami i innymi elementami rzeczywistego dynamicznego środowiska drogowego. W ten sposób możliwe jest przetestowanie rzeczywistych pojazdów z ich zdolnościami autonomicznymi poprzez modelowanie autentycznej dynamiki środowiska wewnątrz symulatora. Umożliwia to symulację różnorodnych scenariuszy, takich jak tworzenie innych pojazdów, pieszych, kontrolowanych obiektów ruchu, symulowanie wypadków, trudnych warunków widoczności oraz innych zdarzeń drogowych [115]. Niezwykle istotne włączenie w proces testowania w środowisku VIL algorytmów opartych na rozszerzonej inteligencji do generowania zróżnicowanych kombinacji otaczającego środowiska, przypadków testowych oraz scenariuszy zdarzeń drogowych. Systemy oparte na uczeniu maszynowym lub głębokim uczeniu są w stanie identyfikować elementy otoczenia, włączając w to dynamiczne interakcje z obiektami takimi jak inne pojazdy, piesi, zwierzęta, stan nawierzchni czy też panujące warunki atmosferyczne. Poprzez analizę reakcji pojazdu na różnorodne warunki środowiskowe, takie algorytmy są w stanie generować różnorodne scenariusze testowe. Działanie takiego systemu umożliwia dynamiczne dostosowywanie się do otoczenia i sytuacji drogowych, co prowadzi do skuteczniejszego i bardziej realistycznego testowania [116].

2.4.2 Metodyka testowania

Metodyka testowania stanowi zestaw zasad, procedur i technik stosowanych w procesie testowania oprogramowania wbudowanego w celu zapewnienia jakości, niezawodności i zgodności z wymaganiami. Obejmuje ona planowanie, projektowanie, wykonywanie i ocenę testów, a także zarządzanie nimi w sposób zorganizowany i skoordynowany. Metodyka testowania uwzględnia

różne podejścia i techniki testowania. W firmie Dräxlmaier stosuje się podejście do testowania systemów wbudowanych oparte o V-model, wynikające z obowiązujących w motoryzacji norm i standardów. W związku ze zmieniającym się rynkiem komponentów oraz systemów wbudowanych dla pojazdów, który dynamicznie reaguje na pojawiające się zmiany, konieczna jest zmiana w zarządzaniu projektami i stosowanie metod zwinnych. Chociaż firma integruje narzędzia do automatyzacji testów, które pozwalają na szybkie i skuteczne przeprowadzanie powtarzalnych testów, to okazuje się, że stosowane metody napotykają na pewne ograniczenia. Jednym z przykładów jest skrócenie czasu na przygotowanie przypadków testowych, ich recenzowanie i wykonanie do krótkich, 2 - 4 tygodniowych cykli.

Fazy testowania

Cykl życia testowania oprogramowania (ang. Software Testing Life Cycle - STLC) jest niezbędny do systematycznego i skutecznego zapewnienia jakości produktu. Obejmuje on sekwencję działań i etapów, składającą się z kolejnych faz, z których każda ma swoje zadania i cele, które są ważne dla ogólnego sukcesu projektu.

1. **Analiza wymagań**
   W początkowej fazie zespół testowy składający się z kierownika testów (ang. Test Manager) i doświadczonych inżynierów testów analizuje wymagania funkcjonalne i niefunkcjonalne, aby zrozumieć istotę działania projektowanego systemu oraz oczekiwania wobec niego. Jest to kluczowe dla kolejnego etapu i może wymagać kilkukrotnych spotkań synchronizacyjnych z inżynierami systemowymi, zespołem odpowiedzialnym za zarządzanie bezpieczeństwem funkcjonalnym, cyberbezpieczeństwem lub klientem.
2. **Planowanie testów**
   Definiuje się strategię testowania, cele testów, zakres, potrzebne zasoby (ludzkie i sprzętowe), harmonogram oraz metodyki testowe. Tworzony jest plan testów (ang. master test plan), który jest wytyczną dla kolejnych działań. Dobry plan testów to zadanie, które wymaga dyskusji, negocjacji i perswazji w całej organizacji [48].
3. **Projektowanie testów**
   Najważniejszym etapem tej fazy jest ustalenie, czy podstawa testów jest wystarczającej jakości do pomyślnej specyfikacji i wykonania przypadków testowych. Następnie definiowane są przypadki testowe opisujące scenariusze testowe, zestaw wejść oraz oczekiwane wyniki. Projektuje się również środowisko testowe i przygotowuje dane testowe.
4. **Implementacja testów**
   Przygotowane przypadki testowe są implementowane, najczęściej w postaci automatyzujących skryptów. Należy również określić czy różne przypadki testowe mają być wykonywane niezależnie, czy wpływać na wyniki kolejno wykonywanych. W przypadku testów wykonywanych manualnie ta faza jest pomijana. Podczas tej fazy można wykryć niedociągnięcia w specyfikacji testów.
5. **Wykonanie testów**
   W tej fazie uruchamiane są przygotowane przypadki testowe w rzeczywistym lub wirtualnym środowisku. Oprogramowanie jest testowane zgodnie z ustalonym planem, a wyniki są rejestrowane.
6. **Analiza wyników i raportowanie błędów**
   Ewaluacja testu, zwana również oceną testu to proces, który wykorzystuje wyrocznię testową. Jest to mechanizm analizowania danych wyjściowych i decydowania o wynikach testu. Rzeczywiste wyniki są porównywane z oczekiwanymi i przypisywany jest werdykt. Wyrocznią (ang. oracle) może być istniejący system, specyfikacja testu lub ekspercka wie-

dza inżyniera oprogramowania [96]. Wykryte błędy i nieprawidłowości są dokumentowane, klasyfikowane pod względem krytyczności i raportowane zespołowi programistów.

7. **Naprawa błędów i powtórne testowanie wraz z testami regresji**
   Naprawą błędów zajmują się deweloperzy, natomiast rolę testerów jest weryfikacja oprogramowania po naprawie — testy są wykonywane powtórnie, aby upewnić się, że błąd został naprawiony (ang. defect retesting), oraz że zmiany nie wpłynęły negatywnie na inne części systemu. Jest to tak zwane testowanie regresji (ang. regression testing).
8. **Raportowanie i dokumentacja**
   Całość procesu testowego jest dokumentowana, włącznie z planem testów, przypadkami testowymi, wynikami przeglądów, wynikami testów oraz zgłoszonymi błędami. Raporty te służą jako podstawa do analizy, oceny jakości oraz przekazywania informacji między zespołami.
9. **Zakończenie testów**
   W tej fazie ocenia się, czy cele testowania zostały osiągnięte i czy spełniono warunki zakończenia testów. Jeśli to możliwe, wydaje się zgodę na dalszy proces wdrożenia lub dostarczenia systemu wbudowanego do klienta.

Cykl życia testów jest iteracyjny i może się powtarzać wielokrotnie w miarę wprowadzania zmian w oprogramowaniu, wersji produktu lub dodawania nowych funkcjonalności. Kluczowe jest podejście systematyczne i dokumentacja wszystkich etapów, aby zapewnić skuteczność i jakość procesu testowania oprogramowania wbudowanego. Automatyzacja pozwala na większą efektywność wykonywania testów, jednak doświadczenie pokazuje, że firmy takie jak Dräxlmaier, zainteresowane są automatyzacją także innych etapów, szczególnie tych najbardziej czasochłonnych i wymagających wiedzy eksperckiej.

Poziomy testów

To aspekty organizacyjne są powodem wprowadzenia poziomów testów. Strukturyzuje to proces testowania, stosując zasadę testowania przyrostowego – na wczesnym etapie rozwoju elementy systemu są testowane w izolacji, w środowisku podobnym do deweloperskiego, aby sprawdzić, czy są one zgodne ze specyfikacjami technicznymi. Gdy poszczególne komponenty cechuje zadowalającą jakość, są one integrowane w większe komponenty lub podsystemy. Te z kolei są następnie testowane w celu sprawdzenia, czy są zgodne z wymaganiami wyższego poziomu. Czynności testowe są wykonywane przez różnych testerów i zespoły testowe w różnych momentach projektu i w różnych środowiskach. Poziom testów to grupa czynności, które są zorganizowane i zarządzane jako całość. Ma to odzwierciedlenie w V-Modelu prezentowanym na rysunku 2.10.

Wyróżniamy testy jednostkowe (SWE.4), testy integracyjne (SWE.5) oraz testy kwalifikacyjne (SWE.6). Końcową fazą testowania systemu wbudowanego są testy systemowe (SYS.4 i SYS.5). Testy systemu są końcową fazą zapewniania jakości w celu walidacji, więc błędy oprogramowania powinny być wykrywane na niższych poziomach. Liczba przypadków testowych jest bardzo duża (często przekracza 1000), ponieważ testy integracyjne dotyczą interfejsów między komponentami, biorąc pod uwagę ich kombinacje. Ponieważ skala oprogramowania rośnie, koszty testów integracyjnych również wzrosły. Zwłaszcza w przypadku systemów wbudowanych, cele testowe w testach integracyjnych dotyczą nie tylko interfejsów między komponentami oprogramowania, ale także interfejsów oprogramowanie – sprzęt [117].

Typy testów

W ramach procesu testowania układów wbudowanych stosuje się różnorodne typy testów, które pozwalają na gruntowną ocenę funkcjonalności oraz zachowań systemu. Typ testu określa, co

będzie testowane. Poniżej zaprezentowano kluczowe typy testów oprogramowania wbudowanego, podkreślając ich istotne aspekty:

- ***Testy jednostkowe*** są skoncentrowane na poszczególnych modułach lub komponentach oprogramowania, umożliwiając weryfikację działania poszczególnych funkcji i procedur.
- ***Testy interfejsów*** koncentrują się na weryfikacji poprawności przesyłania danych, zgodności protokołów komunikacyjnych oraz ogólnej spójności interakcji. W kontekście systemów wbudowanych, interakcje z innymi komponentami, urządzeniami lub systemami mogą mieć kluczowe znaczenie.
- ***Testy funkcjonalne*** stanowią kluczowy element procesu weryfikacji oprogramowania wbudowanego. Obejmują one analizę zachowania systemu na podstawie zestawu określonych wejść oraz oczekiwanych wyników. Testy te pozwalają na ocenę, czy system reaguje zgodnie z wymaganiami funkcjonalnymi, identyfikując ewentualne niezgodności i błędy działania.
- ***Testowanie wydajności i obciążenia (ang. performance testing)*** pozwala ocenić, jak system wbudowany reaguje na zwiększone obciążenie zasobów, jak np. dużą ilość danych czy równoczesne żądania. Testy te oceniają czas odpowiedzi systemu, zużycie pamięci, moc obliczeniową i inne krytyczne parametry wydajności, a także identyfikują potencjalne problemy z wydajnością i dostarcza informacji o zachowaniu systemu w warunkach ekstremalnych.
- ***Testy odzyskiwania***, czyli umiejętności systemu do powrotu do normalnego działania po awarii, są kluczowe w kontekście oprogramowania wbudowanego. Analizują one zdolność systemu do szybkiego i niezawodnego przywrócenia poprawnego działania po wystąpieniu nieprawidłowości.
- ***Testy bezpieczeństwa***, których celem jest identyfikacja potencjalnych luk w zabezpieczeniach systemu, analiza reakcji na próby naruszenia oraz ocena zgodności z wymogami bezpieczeństwa. Bezpieczeństwo stanowi priorytetowy aspekt w systemach wbudowanych, dlatego też odgrywają one istotną rolę.
- ***Testowanie regresji*** pozwala zweryfikować, czy wprowadzone zmiany w oprogramowaniu wpłynęły na działanie innych komponentów. Ten typ testów ma na celu wykrycie ewentualnych nowych błędów lub nieprawidłowości w istniejących funkcjonalnościach [48].

Podsumowując, różnorodne typy testów oprogramowania wbudowanego odgrywają istotną rolę w procesie weryfikacji i weryfikacji systemów wbudowanych. Poprzez wnikliwą analizę różnych aspektów, te testy przyczyniają się do zapewnienia wysokiej jakości, niezawodności oraz zgodności z wymaganiami funkcjonalnymi i bezpieczeństwa oprogramowania wbudowanego.

Techniki projektowania testów

Głównym etapem w procesie testowania jest projektowanie tzw. przypadków testowych. Przypadek testowy stanowi zestaw określonych danych wejściowych, warunków wykonania oraz oczekiwanych rezultatów, zdefiniowanych w celu wykonania określonej ścieżki w programie lub modelu, w celu weryfikacji zgodności oprogramowania z postawionymi wymaganiami. Techniki testowania dzielą się na dwie główne grupy:

- Techniki czarnoskrzynkowe (ang. black-box techniques)
  - testowanie klas równoważności (ang. equivalence partitioning) - technika ta polega na podziale danych wejściowych na grupy, które mają podobne zachowanie. Testy są następnie przeprowadzane na reprezentatywnych wartościach z każdej klasy, co redukuje liczbę przypadków testowych.

  - testowanie wartości brzegowych (ang. boundary value analysis) - jest skoncentrowane na tworzeniu przypadków testowych dla wartości na granicach klas równoważności, aby sprawdzić, jak system zachowuje się na krawędziach dopuszczalnych zakresów danych.
  - testowanie tabel decyzyjnych (ang. decision table testing) - różne reguły decyzyjne w przedstawia się w tabeli, w formie warunków i działań, pomagając w identyfikacji przypadków testowych, które obejmują kombinacje tych reguł.

- Techniki białoskrzynkowe (ang. white-box techniques)
  - testowanie ścieżki (ang. path testing) - mające na celu sprawdzenie, czy wszystkie możliwe ścieżki przez dany fragment kodu są wykonywane.
  - testowanie pokrycia instrukcji (ang. statement coverage) - ma na celu sprawdzenie, czy każda instrukcja w kodzie programu została wykonana.
  - testowanie pokrycia gałęzi (ang. branch coverage) - ma na celu sprawdzenie, czy każda możliwa ścieżka wykonania w programie (gałęzie instrukcji warunkowych) została wykonana.

Oprócz tego powszechnie stosowane są również techniki ***oparte na doświadczeniu*** (ang. experience-base techniques) takiej jak testowanie eksploracyjne i testowanie ad-hoc. Różnica między nimi polega na formalizacji procesu – testowanie ad-hoc nie wymaga zdefiniowania przypadku testowego, jest nieformalne i spontaniczne, najczęściej we współpracy z deweloperem w celu szybkiego znalezienia błędu.

Należy wspomnieć również o technice opartej na przypadkach użycia (ang. use-case based technique), która skoncentrowana jest na tworzeniu przypadków testowych na podstawie symulacji specyficznych sposobów użycia systemu. Stosowanie tej techniki, w porównaniu z klasyczną techniką pokrywania testami pojedynczych wymagań, pozwala na ograniczenie ilości przypadków testowych, wymaga jednak projektującego je inżyniera dogłębnej znajomości wszystkich funkcjonalności testowanego systemu.

Kluczowym składnikiem każdego przypadku testowego jest właściwa stymulacja systemu, co obejmuje dobór funkcji sterujących i warunków początkowych. Następnie dokonywany jest pomiar oraz ocena prawidłowości generowanych przez system wyników. Istotnym aspektem jest także budowa odpowiedniego środowiska testowego, za pomocą którego możliwe jest przeprowadzanie zaprojektowanych przypadków testowych [118].

2.4.3 Automatyzacja testów

Kompleksowość funkcjonalności systemów, ewoluujące wymagania klientów oraz zmieniające się przepisy, co obserwujemy szczególnie w przemyśle motoryzacyjnym, prowadzą do konieczności wprowadzania modyfikacji w systemach wbudowanych zarówno w trakcie, jak i po zakończeniu procesu rozwoju. Wymaga to powtarzania procesu testowania przy każdej zmianie oprogramowania lub sprzętu, co podkreśla nieodzowność automatyzacji testów w celu obniżenia kosztów, skrócenia całego procesu i zwiększenia efektywności [114]. Kluczową zaletą automatyzacji jest zwiększenie efektywności testów regresji, które mogą być przeprowadzane szybko i wielokrotnie, co jest istotne dla procesów rozwoju iteracyjnego. Dzięki automatyzacji testy mogą być włączone w mechanizm ciągłej integracji i wdrażania (ang. continuous integration and continuous deployment — CI/CD) jako ciągłe testowanie (ang. continous testing), co z kolei indukuje natychmiastowe wykrywanie problemów. Dzięki szybkiej informacji zwrotnej odnośnie zmian wprowadzanych w oprogramowaniu deweloperzy są w stanie szybciej podejmować decyzje i wprowadzać korekty. Automatyzacja testów jest inwestycją, która finalnie prowadzi do redukcji kosztów długoterminowych, szczególnie pod względem wykorzystania zasobów ludzkich i czasu. Należy

podkreślić, że testowanie automatyczne eliminuje błędy ludzkie podczas wykonywania testów, zapewniając spójne i dokładne wyniki testów. Mimo wymienionych zalet automatyzacja prowadzi do ujawnienia wpływu środowiska testowego na rezultaty testów. Podczas automatycznego wykonywania skryptów testowych pojawiają się wyniki migoczące, które były niezauważalne gdy tester wykonywał test ręcznie.

W standardowym rozumieniu automatyzacja testów obejmuje przygotowanie przez testerów skryptów realizujących scenariusze testowe. Jest to duże uproszczenie całego procesu automatyzacji przypadków testowych. Axelrod [31] wskazuje, że dla skutecznej automatyzacji konieczna jest możliwość kontroli nad wszystkimi danymi wprowadzanymi do systemu, które mogą mieć wpływ na wartości wyjściowe, łącznie z danymi z zewnętrznych systemów, od których jesteśmy zależni. Automatyzacja testów to proces, który wymaga starannego planowania i realizacji na kilku kluczowych etapach. Dotychczasowo stosowana metodyka tworzenia przypadków testowych uwzględniała wprawdzie implementację automatycznego przypadku testowego, jednak nie skupiała się na tematach związanych z wiarygodnością wyników oraz wpływem środowiska testowego czy stosowanych w nim uproszczeń na cały proces automatyzacji. Poniżej przedstawiono etapy automatyzacji przypadków testowych opisywane w literaturze, które mają odzwierciedlenie w metodyce testowania systemów wbudowanych. W pierwszym kroku należy określić zakres automatyzacji, czyli wybrać funkcje i przypadki testowe najbardziej nadające się do automatyzacji [119]. Należy się w tym przypadku kierować takimi wskaźnikami jak czasochłonność ręcznego wykonania testu, powtarzalność czy krytyczność ze względu na testowane funkcje. W przypadku testowania systemów wbudowanych przeznaczonych do zastosowania w motoryzacji bardzo często jako pierwsze są automatyzowane przypadki testowe o najwyższym poziomie ASIL. Brakującym ogniwem procesu planowania zakresu automatyzacji są rozważania na temat zastosowania narzędzi do automatyzacji w odniesieniu do stosowanej technologii oraz wymagań specyficznych dla danego środowiska testowego.

Kolejny krok to projektowanie testów, czyli opracowanie skryptów, które będą wykonywane automatycznie. Dostępne narzędzia umożliwiają ręczne tworzenie przypadków testowych, które następnie można wykonywać automatycznie. Raporty z wykonania testów również mogą być automatycznie generowane [42]. Należy pokreślić, że wymaga to zrozumienia działania testowanego systemu wbudowanego w kontekście całego pojazdu. Należy pamiętać o tym, żeby projektować testy w taki sposób, aby były modułowe (czyli ich rezultat nie zależał od kolejności wykonania ani rezultatu poprzedniego testu), łatwe do utrzymania i ponownego użycia w zbliżonych scenariuszach testowych. W przypadku gdy występują zależności między wyspecyfikowanymi przypadkami testowymi, skrypty im odpowiadające można zebrać w kolekcje, ale należy zapewnić, że nie będzie możliwe wykonanie pojedynczego testu z kolekcji. Równolegle z implementacją skryptów testowych przygotowuje się i konfiguruje środowisko testowe, w którym będą uruchamiane testy. Zaimplementowane testy i skonfigurowane środowisko powinny przejść walidację — czyli sprawdzenie, czy wszystko działa zgodnie z założeniami oraz weryfikację — czyli sprawdzenie przez drugiego eksperta pod względem poprawności i zgodności ze specyfikacją przypadku testowego. Bardzo ważnym aspektem, nieujętym w metodyce i często pomijanym, jest takie projektowanie środowisko testowego, aby możliwe było szybkie i łatwe zgromadzenie informacji na temat zmian zachodzących w środowisku w czasie wykonywania testów automatycznych np. w postaci plików dziennika.

Po zaimplementowaniu skrypty testowe są uruchamiane na wybranej konfiguracji sprzętowo – programowej, wyniki testów są zbierane, przygotowywane są, najczęściej również automatycznie [42], raporty z wykonania testów. Jeżeli skrypty testowe zostały włączone do procesów CI/CD są one uruchamiane automatycznie przy każdej zmianie kodu i zapisaniu go w repozytorium, a raporty automatycznie przesyłane do deweloperów.

Po wykonaniu testów otrzymane wyniki są analizowane, błędy i problemy muszą zostać zidentyfikowane, a defekty w oprogramowaniu odpowiednio opisane i zgłoszone. Należy pamiętać

o zaplanowaniu czasu na debugowanie systemu i rozpoznanie przyczyn negatywnych wyników testów [120]. Jest to proces czasochłonny i wymagający uwagi ekspertów dobrze znających testowany system oraz kontekst przypadku testowego. Bardzo często w przypadku problemów ze zidentyfikowaniem przyczyny negatywnego testu, wykonywany jest on powtórnie manualnie. Zautomatyzowanie tego procesu i umożliwienie zastosowania algorytmów rozszerzonej inteligencji do weryfikacji wyników testów stanowi jeden z problemów badawczych niniejszej dysertacji. Ostatnim etapem dotychczas stosowanej metodyki jest modyfikacja skryptów testowych na podstawie wyników i informacji zwrotnych od zespołu deweloperów. Ma to na celu poprawienie ich skuteczności, optymalizację procesu. Dokładne informacje zwrotne uzyskane w wyniku zastosowania algorytmów rozszerzonej inteligencji do weryfikacji wyników i wykrywania anomalii w środowisku testowym mogłyby być wskazówką do optymalizacji i poprawy samego środowiska.

W przypadku zmian w oprogramowaniu może zachodzić potrzeba modyfikacji skryptów testowych — powinny one być na bieżąco aktualizowane. Oprócz automatycznego wykonywania przypadków testowych interesującymi obszarami rozwoju w dziedzinie automatyzacji testów jest generowanie przypadków testowych [11]. Biorąc pod uwagę generowanie testów na przykład na poziomie SWE.4, podczas którego tworzone są sekwencje danych wejściowych programu oraz wyrocznie oceniające poprawność wynikowego wykonania dla testowanego systemu efektywna automatyzacja mogłaby prowadzić do znacznych oszczędności czasu i kosztów. Jak wskazuje Fontes i Gay [121] generowanie automatycznych testów jest popularnym tematem badawczym, w którym osiągnięto znaczące sukcesy. Wskazują oni na krytyczne ograniczenia istniejących rozwiązań, do których głównie należy fakt, że proponowane metody są bardzo ogólne, a stosowane techniki celują w proste, uniwersalne heurystyki, stosowane statycznie do wszystkich systemów w jednakowy sposób. Rozważają oni również zastosowanie uczenia maszynowego, które usprawnia proces generowania przypadków testowych. Są to jednak wszystko prace w fazie badań podstawowych, na rynku nie są jeszcze dostępne kwalifikowane narzędzia umożliwiające zastosowanie ich w rzeczywistych projektach w branży motoryzacyjnej.

3. Rozszerzona inteligencja

Rozdział dotyczy rozszerzonej inteligencji (ang. Augmented Intelligence - AuI) – zaawansowanej technologii, która łączy elementy sztucznej inteligencji (ang. Artificial Intelligence - AI) z interaktywnymi systemami wspierającymi człowieka w podejmowaniu decyzji oraz wykonywaniu złożonych zadań. Wprowadzenie do tego rozdziału obejmuje genezę sztucznej inteligencji oraz omówienie podstawowych pojęć z nią związanych, takich jak uczenie maszynowe, uczenie nadzorowane i nienadzorowane, sieci neuronowych oraz uczenia głębokiego. Następnie przeanalizowano różnice między sztuczną a rozszerzoną inteligencją oraz przedstawiono zastosowania rozszerzonej inteligencji w takich dziedzinach jak diagnostyka medyczna i przemysł. Szczególną uwagę poświęcono zastosowaniu algorytmów rozszerzonej inteligencji w diagnostyce oraz testowaniu systemów wbudowanych, co stanowi kluczowy element w kontekście tematu niniejszej dysertacji, czyli zastosowaniu tego podejścia w procesie automatyzacji testów systemów wbudowanych stosowanych w pojazdach samochodowych.

3.1 Wprowadzenie

Geneza sztucznej inteligencji datuje się na wczesne lata pięćdziesiąte ubiegłego wieku, kiedy Alan Turing, pionier współczesnej algorytmiki sformułował pytanie, znane jako test Turinga: Czy program komputerowy może się komunikować z człowiekiem w taki sposób, aby nie było możliwe zidentyfikowanie, że prowadzi się rozmowę z maszyną. Termin sztuczna inteligencja został wprowadzony przez Johna McCarthy'ego i in. w 1955 roku, który zdefiniował go jako dziedzinę nauki i inżynierii, zajmującą się tworzeniem inteligentnych maszyn, a w szczególności inteligentnych programów komputerowych, zdolnych symulować swoim działaniem ludzką inteligencję [122]. Andreas Kaplan i Michael Haenlein określili tym mianem zdolność systemu do prawidłowego interpretowania danych pochodzących z zewnętrznych źródeł, nauki na ich podstawie oraz wykorzystywania tej wiedzy do realizacji konkretnych zadań i celi poprzez elastyczne dostosowanie [123].

Z dziedziną sztucznej inteligencji nierozerwalnie łączy się termin uczenie maszynowe (ang. machine learning - ML), który został użyty po raz pierwszy przez Arthura Samuela w 1959 roku jako umiejętność komputerów uczenia się bez bycia zaprogramowanym do danego zadania. Najogólniej mówiąc, maszyny uczą się wykonywać zadania, rozpoznając pewien wzorzec i próbując go pośrednio lub bezpośrednio naśladować. Istotą uczenia maszynowego jest poprawne analizowanie zależności występujących w danych.

Algorytmy sztucznej inteligencji są stosowane do rozwiązywania różnorodnych zadań, m.in. takich jak regresja, klasyfikacja czy grupowanie. W literaturze często pojawiają się takie metody AI, jak: drzewa decyzyjne, które tworzą modele oparte na sekwencji warunków; sieci neuronowe, inspirowane działaniem ludzkiego mózgu, znajdujące zastosowanie w rozpoznawaniu obrazów, klasyfikacji i przetwarzaniu języka naturalnego; systemy rozmyte, umożliwiające modelowanie niepewności i wieloznaczności w procesie podejmowania decyzji; oraz sieci bayesowskie, oparte na prawdopodobieństwie warunkowym, które wykorzystywane są do przewidywań oraz analizy niepewności [45]. Wszystkie te metody są integralną częścią uczenia maszynowego (ML), które polega na automatycznym doskonaleniu się algorytmów na podstawie danych. Drzewa decyzyjne oraz sieci neuronowe są klasycznymi przykładami algorytmów uczenia nadzorowanego, gdzie modele są trenowane na oznakowanych danych. Z kolei systemy rozmyte i sieci bayesowskie

często znajdują zastosowanie w bardziej zaawansowanych modelach, które potrafią dostosować się do niepewności i braków w danych, stanowiąc fundament dla nowoczesnych technik ML [124].

Istnieje wiele typów oraz podziałów uczenia maszynowego. Wyróżniamy uczenie nadzorowane (ang. supervised learning), czyli metodę uczenia, w której zbiór danych treningowych, na którym uczony jest algorytm, zawiera informacje zwrotne dotyczące rozwiązania problemu, znane jako etykiety lub klasy. Uczenie nadzorowane, pomimo swoich licznych zalet, ma również wady, które mogą mieć wpływ na skuteczność i użyteczność modeli uczonych w tym podejściu. Wymaga ono etykietowania danych treningowych, co oznacza, że należy mieć informacje o prawidłowych odpowiedziach dla każdego przypadku. Pozyskanie etykiet może być kosztowne, czasochłonne, a czasami niepraktyczne. Ponadto jakość i dokładność etykiet ma wpływ na wydajność modelu. Ręczne etykietowanie może prowadzić do błędów ludzkich, które są trudne do zdiagnozowania i skorygowania. Model uczenia nadzorowanego jest trenowany na określonym zbiorze danych treningowych, co powoduje ograniczoną zdolność do generalizacji [125]. Kolejnym aspektem, który należy rozważyć, wdrażając model uczenia nadzorowanego, jest zależność od reprezentatywności danych treningowych. Jeżeli zbiór treningowy nie odzwierciedla pełnego zakresu możliwych przypadków w rzeczywistym środowisku, model może być niewystarczająco przygotowany do obsługi różnorodnych sytuacji. Również wrażliwość na nadmierne dopasowanie jest wadą uczenia nadzorowanego. Jeżeli model nadmiernie dopasuje się do danych treningowych (ang. overfitting), może nie radzić sobie dobrze na nowych danych. Oznacza to, że nadmiernie dopasowany model traci zdolność do uogólniania [126]. Wad tych pozbawione jest uczenie nienadzorowane (ang. unsupervised learning), czyli proces uczenia modelu, w którym dane treningowe nie są oznakowane, czyli nie posiadają etykiet. Innymi słowy, są to surowe dane, które są przekazywane do modelu, a algorytm jest odpowiedzialny za odkrywanie wzorców lub związków między nimi [127]. Znakowanie danych etykietami często jest czasochłonne i kosztowne. W większości przypadków dane składają się głównie z nieoznakowanych obserwacji, z tylko niewielką liczbą oznakowanych danych, dlatego też powstały algorytmy, radzące sobie z tym problemem — mówimy w takiej sytuacji o uczeniu półnadzorowanym (ang. semisupervised learning). Większość algorytmów półnadzorowanych opiera się na kombinacji technik uczenia nadzorowanego i nienadzorowanego, często wykorzystujących sieci neuronowe [128].

Sztuczna inteligencja rozwija się w kilku kluczowych kierunkach, które wyznaczają przyszłość tej technologii. Najważniejsze z nich to:

- Autonomia systemów – AI staje się coraz bardziej niezależna, rozwijając zdolność do podejmowania decyzji i realizacji zadań bez udziału człowieka. Pojazdy autonomiczne i roboty przemysłowe są przykładami takich technologii. W tym kierunku AI zmierza ku pełnej automatyzacji procesów, gdzie człowiek pełni jedynie rolę nadzorczą [129].
- Rozwój głębokiego uczenia – Jednym z kluczowych aspektów rozwoju AI jest wykorzystanie głębokich sieci neuronowych. Głębokie uczenie pozwala na samodzielne odkrywanie złożonych wzorców w danych, co znajduje zastosowanie w zadaniach takich jak rozpoznawanie obrazów, analiza języka naturalnego czy przewidywania oparte na dużych zbiorach danych [130], [131].
- Sztuczna inteligencja ogólna (ang. Artificial General Intelligence, AGI) – Chociaż AGI pozostaje w sferze teorii, badania nad tą formą AI mają na celu stworzenie systemu, który mógłby rozwiązywać szeroki wachlarz problemów, podobnie jak ludzki umysł. Dążenie do AGI to kierunek rozwoju, który wymaga połączenia różnych technik AI i ML, z elastycznym podejściem do uczenia się [132].
- Współpraca człowiek-AI (ang. Human-AI collaboration) – W tym kontekście rozszerzona inteligencja (AuI) odgrywa kluczową rolę. Zamiast dążyć do pełnej autonomii, AI ma wspierać człowieka w podejmowaniu lepszych decyzji. Rozwój AI w tym kierunku koncentruje się na poprawie interakcji z człowiekiem, na przykład w systemach wspomagania

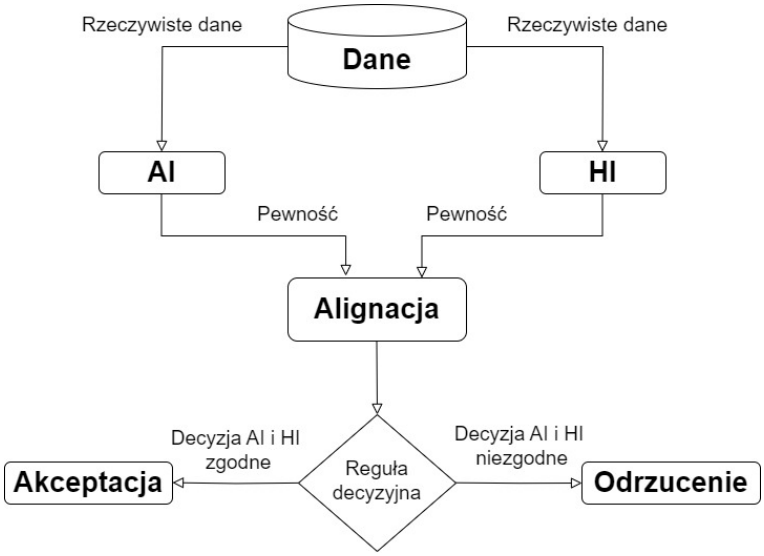
decyzji w medycynie, produkcji czy zarządzaniu [133], [134].

- Etyka i transparentność – Wraz z rozwojem AI rośnie również potrzeba odpowiedzialnego rozwoju tej technologii. W szczególności istotne stają się badania nad etycznym zastosowaniem AI oraz zapewnieniem transparentności algorytmów, aby zapobiegać dyskryminacji i zapewniać zaufanie do systemów AI [135], [136].

3.2 Sztuczna inteligencja vs. rozszerzona inteligencja

Społeczne postrzeganie terminu *sztuczna inteligencja* ukierunkowanego przez media na fantastyczną wizję robotów i systemów komputerowych, które stworzone przez człowieka, ale zimne i nieludzkie, wskutek logicznego działania oprogramowania przejmują kontrolę nad planetą i rodzajem ludzkim, zapoczątkowało wiele etycznych dyskusji nad tematem zastąpienia ludzkiej inteligencji (ang. HI - human intelligence) [137]. Wszystkie te etyczne rozważania spowodowały, że sztuczna inteligencja staje się narzędziem do rozszerzenia, a nie zastępowania zdolności człowieka [138]. Zmiana perspektywy postrzegania tematu AI z inteligencji autonomicznej, na rozszerzoną wynika również z faktu, że człowiek wspierany sztuczną inteligencją może wykonywać pewne zadania lepiej niż autonomiczna sztuczna inteligencja [32].

W.R. Ashby w 1956 r. we *„Wprowadzeniu do cybernetyki”* przedstawił pojęcie wzmocnienia inteligencji [139] rozumianego jako udoskonalenie ludzkiego procesu decyzyjnego. Tradycyjnie AI dąży do naśladowania HI w wąsko zdefiniowanych zadaniach, podczas gdy AuI ma na celu poprawę i wzmocnienie ludzkiej mocy poznawczej, a nie jej zastąpienie [140]. Rozszerzona inteligencja wykorzystuje algorytmy sztucznej inteligencji, uczenie maszynowe, w tym głębokie uczenie, aby zapewnić człowiekowi informacje i podpowiedzi do dalszego działania. Podejście bazujące na AuI, wykorzystując techniki AI i operując na bardzo dużych zbiorach danych, jest w stanie je analizować, dostrzegać wzorce i informować o nich człowieka w czasie nieosiągalnym dla istoty ludzkiej. Rozszerzona inteligencja umożliwia synergiczną współpracę między ludźmi a maszynami, gdzie obydwie strony wzajemnie się uczą i wykorzystują swoje indywidualne mocne strony w celu osiągnięcia wspólnych celów, co zostało schematycznie pokazane na rysunku 3.1.



Rys. 3.1. Koncepcja rozszerzonej inteligencji [140]

Przedstawione na rysunku 3.1 algorytmy AI potrafią analizować duże ilości danych, szybciej i dokładniej niż ludzie, dzięki czemu wspomagają w ramach rozszerzonej inteligencji ludzką

zdolność do podejmowania decyzji na podstawie faktów i wniosków. Wykrywanie wzorców i zależności w danych, niewidocznych dla ludzkiego oka pozwala ludziom podejmować bardziej świadome decyzje. Zaznaczone na rysunku przetwarzanie tych samych danych przez człowieka ma nieco inny wymiar. Ludzka inteligencja służy przede wszystkim do określenia celów i oczekiwań względem analizy danych przez algorytmy sztucznej inteligencji. Ludzka intuicja i doświadczenie są kluczowe w procesie selekcji i przygotowania danych, identyfikowaniu istotnych informacji, eliminowaniu szumów oraz błędów. W etapie alignacji, czyli dopasowania, następuje harmonizacja działań sztucznej inteligencji z celami i intencjami ludzkimi. Ludzka inteligencja jest niezbędna do zrozumienia subtelnych niuansów wyników, które mogą być pominięte przez algorytmy sztucznej inteligencji. Ograniczenia technologii AI w niektórych ważnych zastosowaniach, takich jak kontrola ryzyka w przemyśle, diagnostyka medyczna, diagnostyka przemysłowa oraz system wymiaru sprawiedliwości powodują, że w celach weryfikacyjnych muszą zostać wprowadzone nadzór, interakcja i udział człowieka. To ludzie podejmują decyzje, oceniając ryzyko oraz biorąc pod uwagę czynniki, które są poza zasięgiem sztucznej inteligencji, na podstawie swojej wiedzy, doświadczenia i intuicji. Z jednej strony poprawia to poziom zaufania do systemów inteligentnych, z drugiej strony, optymalnie wykorzystana zostanie wiedza ekspercka i wysoko rozwinięta technologia sztucznej inteligencji [141]. Integracja ludzkiej inteligencji w ramach rozszerzonej inteligencji pomaga również uwzględnić wartości etyczne, prawne i społeczne podczas podejmowania decyzji, takie jak uczciwość, odpowiedzialność i przejrzystość [136].

Zaprezentowana powyżej zaawansowana technologia ma niezliczone zastosowania, w tym automatyczne rozpoznawanie mowy, przetwarzanie języka naturalnego, analizę obrazów, systemy rekomendacyjne i wiele innych [46]. W ramach niniejszej dysertacji zidentyfikowano potencjał rozszerzonej inteligencji do automatyzacji weryfikacji wyników testów systemów wbudowanych, której celem jest zwiększenie efektywności i dokładności tego procesu, co przekłada się na wyższą jakość i niezawodność systemów wbudowanych w pojazdach samochodowych.

3.3 Obszary zastosowań rozszerzonej inteligencji

W ostatniej dekadzie zaobserwowano dynamiczny wzrost wykorzystania rozszerzonej inteligencji w różnorodnych dziedzinach. Sektor medyczny, przemysłowy i finansowy doświadczyły znacznego postępu, dzięki zastosowaniu tej zaawansowanej technologii. Implementacja algorytmów sztucznej inteligencji oraz uczenia maszynowego przyczyniło się do precyzyjnego prognozowania na rynkach finansowych, rozwoju autonomicznych pojazdów czy identyfikacji komórek rakowych w organizmach ludzkich [46]. Należy podkreślić, że we wszystkich wymienionych niżej przykładach zaawansowane techniki sztucznej inteligencji służą wzmocnieniu ludzkiej mocy poznawczej, natomiast udział inteligencji ludzkiej w procesie podejmowania decyzji czy to odnośnie leczenia pacjenta, czy na przykład toru jazdy samochodu jest nieodzowny.

Potencjał zastosowań rozszerzonej inteligencji w sektorze medycznym i zdrowotnym jest szeroki. Dzięki analizie historii klinicznych pacjentów i wsparciu uczenia maszynowego możliwe jest bardziej efektywne diagnozowanie schorzeń, precyzyjną analizę obrazów medycznych, przewidywanie wyników terapeutycznych oraz rozwinięcie spersonalizowanych planów terapii. Ponadto, zastosowanie rozszerzonej inteligencji może usprawnić monitorowanie stanu zdrowia pacjentów, wczesne wykrywanie potencjalnych zagrożeń oraz szybką interwencję medyczną w celu maksymalizacji rezultatów terapeutycznych. Potencjał rozszerzonej inteligencji jest również istotny dla przewidywania i zapobiegania chorobom lub epidemiom na dużą skalę [137].

Początek nowej ery wykrywania czerniaka dzięki zastosowaniu rozszerzonej inteligencji zapowiada Cerminara [142]. Badania przesiewowe w kierunku raka skóry pozostają ciągle wyzwaniem dla dermatologów ze względu na czasochłonny proces śledzenia wszystkich zmian melanocytowych. Europejskie wytyczne dotyczące czerniaka zalecają stosowanie fotografii całego ciała (ang. total body photography - TBP) oprócz sekwencyjnej dermatoskopii cyfrowej. Zgodne z tymi zale-

ceniami zautomatyzowane, oparte na sztucznej inteligencji trójwymiarowe (3D) i dwuwymiarowe (2D) urządzenia TBP wykorzystują głęboko uczące się konwolucyjne sieci neuronowe do oceny ryzyka złośliwości. Dzięki temu możliwe jest wczesne wykrywanie zmian de novo i minimalizacja liczby niepotrzebnych biopsji. Również Amit [143] proponuje nowatorskie podejście do klasyfikacji i przewidywania raka skóry przy użyciu rozszerzonej inteligencji. Natomiast Cheema stworzył i wdrożył model uczenia maszynowego [144], który miał za zadanie wspomagać lekarzy pierwszego kontaktu w rozpoznawaniu ciężkiej postaci niewydolności serca. Zadanie identyfikacji pacjentów z ciężką postacią niewydolności serca jest trudne z kilku powodów. Symptomy zbliżającej się choroby są najczęściej bardzo subtelne i rozłożone w czasie. Do trenowania modelu wykorzystał on ustrukturyzowane dane generowane w ramach rutynowej opieki klinicznej.

Przewiduje się, że rozszerzona inteligencja będzie stanowić wsparcie dla początkujących lekarzy, bez specjalizacji. Umożliwi im wykonywanie zadań, które obecnie są zarezerwowane dla specjalistów, a także przesiewanie prostych przypadków, co z kolei pozwoli specjalistom skupić się na bardziej złożonych problemach. Algorytmy rozszerzonej inteligencji wydają się idealnym narzędziem do badań przesiewowych oraz oceny w obszarach, gdzie dostęp do specjalistycznej wiedzy medycznej jest ograniczony [145].

W sektorze przemysłowym można zauważyć zastosowanie rozszerzonej inteligencji do optymalizacji procesów produkcyjnych, prognozowania popytu, monitorowania jakości, planowania i zarządzania łańcuchem dostaw. Dodatkowo rozszerzona inteligencja może wspierać w utrzymaniu i diagnozowaniu maszyn oraz przewidywaniu awarii. Implementacja metod i algorytmów rozszerzonej inteligencji w inżynierii mechanicznej otwiera nowe możliwości rozwoju w zakresie utrzymania ruchu, planowania konserwacji oraz optymalizacji procesów. To nowatorskie podejście przyczynia się do podniesienia poziomu niezawodności, efektywności i wydajności systemów mechanicznych z akcentem na decyzyjność po stronie człowieka odpowiedzialnego za dany proces. Predykcyjna analiza danych z czujników i systemów monitorujących za pomocą technik uczenia maszynowego jest w stanie identyfikować wzorce i sygnały alarmowe, które wskazują na nadchodzące problemy, identyfikując wzorce i sygnały alarmowe. Możliwe jest dzięki temu podjęcie przez człowieka odpowiednich działań zapobiegawczych, planowanie konserwacji, aby zminimalizować czas przestoju i koszty napraw.

Jednym z najbardziej rozpoznawalnych zastosowań rozszerzonej inteligencji w sektorze przemysłowym są autonomiczne pojazdy [137]. Korzystamy z praktycznych aplikacji rozszerzonej inteligencji w formie systemów wspierających kierowcę takich jak adaptacyjny tempomat czy systemy utrzymywania pasa ruchu. W momencie osiągnięcia poziomu 3 autonomicznej jazdy zgodnie z klasyfikacją SAE (Society of Automotive Engineers), możemy odnieść się do zastosowania sztucznej inteligencji w roli zastępującej kierowcę [146]. W obszarze projektowania nowych produktów rozszerzona inteligencja zapewnia kreatywne i rozmyte rozumowanie bez konieczności stosowania kompleksowych modeli inżynieryjnych w połączeniu z szybkim i efektywnym wykonywaniem skomplikowanych, powtarzalnych i proceduralnych zadań.

Dzięki zastosowaniu rozszerzonej inteligencji do wykrywania usterek i diagnostyki ich pierwotnych przyczyn, w dużych zakładach chemicznych i produkcyjnych, zmniejszone jest opóźnienie w wykrywaniu błędów, co jest ważne dla zmniejszenia strat. Zastosowanie rozszerzonej inteligencji w procesie szkolenia pracowników pozwala na włączenie do programu szkolenia wiedzy empirycznej, niejawnej. Zastosowanie systemu szkoleniowego opartego tylko na sztucznej inteligencji jest praktycznie niemożliwe, ponieważ należy modelować różne zachowania i osobowości [140].

W odniesieniu do sektora finansowego i bankowego badania wykazały [137], że rozszerzona inteligencja, dzięki implementacji w usługach do analizy rynku, prognozowania trendów, zarządzania ryzykiem, wykrywaniu oszustw czy bionicznym doradztwie zmieniła sposób, w jaki wchodzimy w interakcje z bankami i zarządzamy kapitałem. Dodatkowo może ona wspomagać automatyzację procesów transakcyjnych i decyzyjnych, zwiększający tym samym precyzję

i efektywność działań w tym sektorze. Dzięki zastosowaniu zaawansowanych algorytmów i technik uczenia maszynowego, systemy oparte na rozszerzonej inteligencji są w stanie analizować ogromne ilości danych finansowych oraz dostarczać cennych informacji i rekomendacji dla podmiotów działających w branży finansowej.

3.4 Zastosowanie metod rozszerzonej inteligencji w testowaniu układów wbudowanych

Przewiduje się, że sztuczna inteligencja wpłynie na testowanie w szerokim zakresie domen oprogramowania od aplikacji mobilnych i internetowych, IoT, aplikacje bazodanowe, przemysł gier aż po systemy wbudowane, aplikacje czasu rzeczywistego i aplikacje krytyczne [147]. W tych trzech ostatnich przypadkach, aby podkreślić rolę człowieka w procesie zapewnienia jakości i bezpieczeństwa testowanych systemów, należy używać terminu rozszerzona inteligencja.

W pracy [148] przedstawiono uczenie maszynowe jako technikę automatycznego uczenia się poprawnego zachowania systemu, które musi spełniać daną specyfikację. Wraz z rosnącą złożonością systemu wbudowanego nie wszystkie możliwe interakcje mogą być ręcznie określone jako przypadki testowe. Należy pamiętać, że dodatkowe dane wejściowe, jeżeli wprowadzane automatycznie, muszą być zgodne ze wzorcami zachowania komponentów systemu, aby przeprowadzany test odpowiadał rzeczywistemu zachowaniu systemu. Bielefeldt zaproponował zastosowanie uczenia głębokiego do uzyskania nowych, realistycznych danych testowych oraz do identyfikacji nowych wzorców zachowania systemu wbudowanego [149]. Weryfikacja tych aspektów to zadanie dla ludzkiej inteligencji (Rys. 3.1). Rozszerzona inteligencja zastosowana w procesie testowania systemów wbudowanych to narzędzie, które może działać jako kompetentny i kompleksowy system ekspercki. Biorąc pod uwagę dane wejściowe dotyczące systemu wbudowanego oraz jego komponentów sprzętowych, będzie on w stanie zasugerować najbardziej odpowiedni zestaw testów, aby zapewnić pełne pokrycie wymagań dla produktu, ostrzec testerów o niespójnościach czy anomaliach w systemie testowym.

Fontes przeprowadził analizę 124 publikacji związanych z zastosowaniem uczenia maszynowego w automatycznym generowaniu testów oprogramowania. Stwierdził on, że dotychczasowe prace są obiecujące, jednak istnieją wyzwania związane z danymi szkoleniowymi — ich wymaganą ilością, jakością i zawartością. Podkreślił również, że badania prowadzone były na uproszczonych przykładach i nie jest jasne czy wskazane przez badaczy techniki są skalowalne do rzeczywistych systemów [121]. Celowe wydaje się zintegrowanie zaproponowanych rozwiązań w ramach rozszerzonej inteligencji, dzięki czemu można podjąć próbę ich aplikacji w rzeczywistych systemach.

Analizując możliwości zastosowania rozszerzonej inteligencji w procesie testowania systemów wbudowanych, uwzględniono dostępną literaturę przedstawiającą zastosowanie algorytmów sztucznej inteligencji. Hourani [36] wspomina, że sztuczna inteligencja może usprawnić proces testowania oprogramowania między innymi w analizie wymagań funkcjonalnych i przez automatyczne generowanie przypadków testowych. Verma i Beg [150] zaproponowali do tego podejście oparte na technice przetwarzania języka naturalnego (ang. Natural Language Processing — NLP), wymagające jednak narzędzia do automatyzacji i zastosowania bazy danych do przechowywania wygenerowanych grafów. Ansari [151] również zaproponował system oparty na NLP umożliwiający redukcję wysiłku i czasu na ekstrachowanie przypadków testowych ze specyfikacji wymagań oprogramowania. Kikuma i in. również zaproponowali technikę automatycznego wyodrębniania jednorodnych przypadków testowych z dokumentów specyfikacji wymagań, eliminując tym samym zależność od umiejętności i wiedzy inżyniera, którego zadaniem jest tworzenie przypadków testowych [152]. Biorąc pod uwagę wymienione przykłady, w kontekście procesu testowania oprogramowania wbudowanego, połączenie zaproponowanych metod sztucznej inte-

ligencji z ludzką inteligencją w rozszerzoną inteligencję (Rys. 3.1) można zrealizować na etapie analizy wymagań funkcjonalnych. Obejmuje to analizę zależności między wymaganiami oraz niuansów w działaniu systemu wbudowanego. Dodatkowo rozszerzona inteligencja może wspierać recenzję wygenerowanych przypadków testowych pod kątem poprawności i kompletności pokrycia wymagań.

Na podstawie wyników badania Khaliq i in. [147] można stwierdzić, że zastosowanie technik sztucznej inteligencji znacznie usprawniło proces generowania i projektowania przypadków testowych. Jednak autorzy wspominają, że ich badania koncentrowały się głównie na obszarach takich jak generowanie przypadków testowych, priorytetyzacja, generowanie danych i budowa wyroczni [153]. Warto zauważyć, że niektóre czynności związane z testowaniem oprogramowania, w tym implementacja środowiska testowego, wybór technik testowania, utrzymanie testów i analiza podatności, zostały pominięte w ich manuskrypcie ze względu na ograniczoną dostępność badań opartych na sztucznej (czy rozszerzonej) inteligencji w tych obszarach.

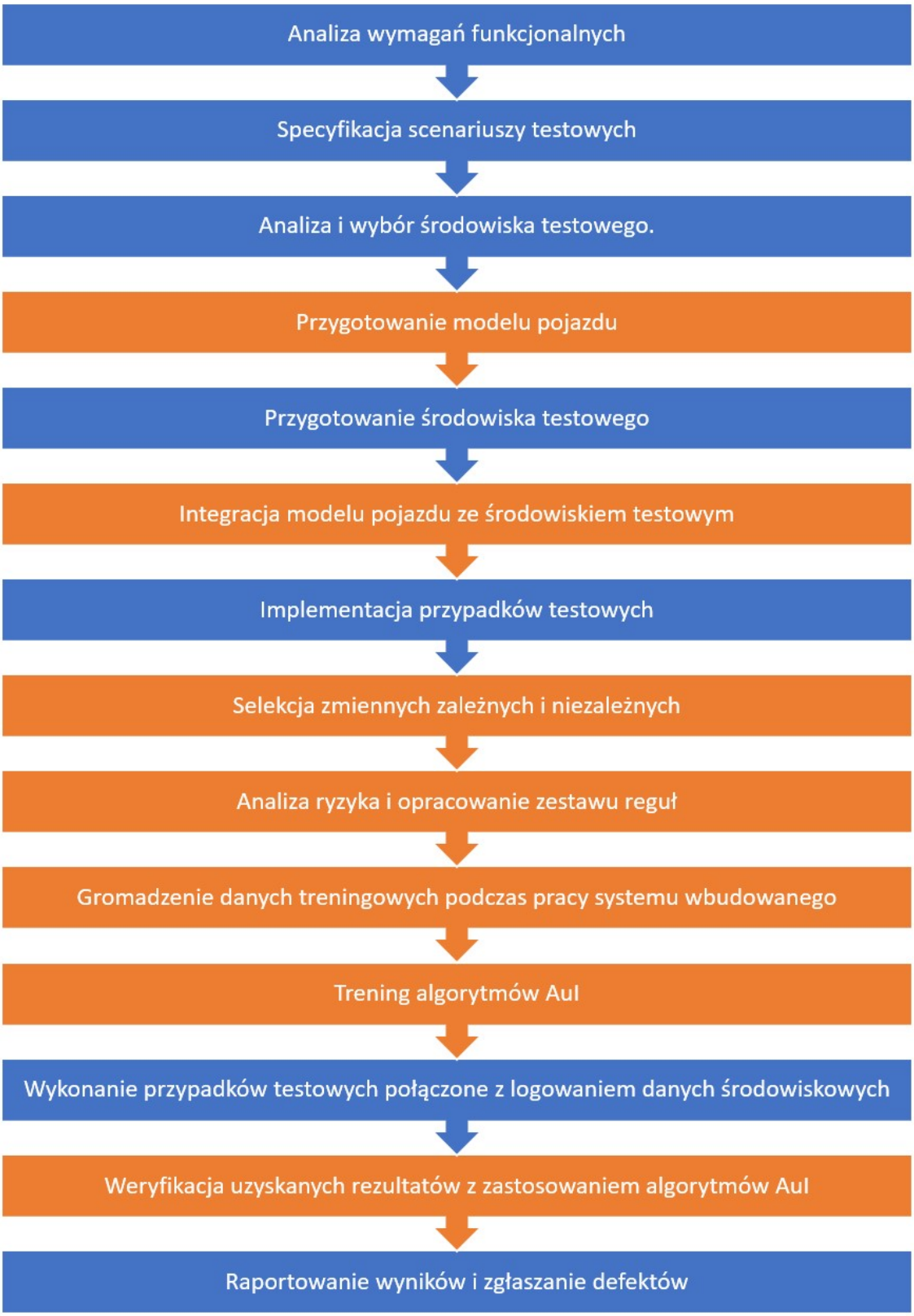
Według najlepszej wiedzy autorki obecnie brakuje wystarczających wyników badań dotyczących efektywnego zastosowania algorytmów AI, które są podstawą działania AuI, do automatyzacji testów systemów wbudowanych w pojazdach samochodowych. Co więcej, brakuje również badań nad zastosowaniem metod wspartych modelowo, które mogłyby znacznie usprawnić procesy testowania poprzez dostosowanie technik do specyficznych wymogów systemów wbudowanych. Metody te powinny odnosić się zarówno do AI, jak i AuI, co pozwoliłoby na bardziej efektywne strukturyzowanie i planowanie testów oraz skuteczniejszą weryfikację wyników. Niemniej jednak ich praktyczne zastosowanie w testach automatycznych systemów wbudowanych w pojazdach pozostaje wciąż nieodkrytym obszarem. Podsumowując, pomimo znaczącego rozwoju algorytmów AI, stosowanych w myśli idei AuI w różnych dziedzinach, istnieje wyraźna luka badawcza dotycząca ich zastosowania w automatyzacji testów systemów wbudowanych w przemyśle motoryzacyjnym.

4. Metodyka testowania systemów wbudowanych uwzględniająca weryfikację wyników z zastosowaniem rozszerzonej inteligencji

Rozdział ten przedstawia zintegrowane podejście do testowania systemów wbudowanych, które łączy tradycyjne metodyki testowania z zaawansowanymi technikami rozszerzonej inteligencji. Składa się on z trzech podrozdziałów. Pierwszy z nich koncentruje się na kluczowych elementach metodyki testowania, które są niezbędne do skutecznej oceny jakości systemów wbudowanych oraz wprowadza nowe, innowacyjne elementy, zwiększające dokładność i efektywność procesu testowania. Poprzez całościowe omówienie kluczowych elementów metodyki testowania, zastosowania modelu pojazdu, eksploracji danych środowiskowych oraz weryfikacji wyników za pomocą AuI, rozdział ten dostarcza kompleksowych wytycznych dla projektowania i wdrażania skutecznych systemów testowych. Natomiast drugi podrozdział skupia się na integracji poszczególnych elementów metodyki w procesie testowania systemów wbudowanych. Z kolei trzeci opisuje algorytmy sztucznej inteligencji wybrane do badań w celu zastosowania w ramach rozszerzonej inteligencji.

4.1 Elementy metodyki stosowane w procesie testowania systemów wbudowanych

Podstawą zaproponowanej metodyki jest proces testowania oprogramowania wbudowanego opierający się o normę ISO26262 [8] oraz standard ASPICE [9]. Na schemacie 4.1 kolorem niebieskim oznaczono kroki wymagane przez wspomniane standardy. Nowatorskie podejście do procesu testowania systemów wbudowanych uwzględnia zastosowanie modelu pojazdu oraz algorytmów rozszerzonej inteligencji do weryfikacji rezultatów wykonywanych testów. Innowacyjne kroki metodyki zostały zaakcentowane kolorem pomarańczowym.



Rys. 4.1. Ogólny zarys metodyki testowania systemów wbudowanych uwzględniającej weryfikację rezultatów z zastosowaniem rozszerzonej inteligencji.

Analiza wymagań funkcjonalnych

Analiza wymagań funkcjonalnych to kluczowy etap w procesie inżynierii oprogramowania, zgodnie ze standardem ASPICE [9] zlokalizowany na lewym ramieniu V-modelu, co nie oznacza, że zespół inżynierów jakości oprogramowania nie powinien być zaangażowany w ten proces. Wręcz przeciwnie, zaangażowanie zespołu testującego oprogramowania w proces analizy i definiowania wymagań pozwala uniknąć wielu nieścisłości i defektów w oprogramowaniu wykrywanych na późniejszych etapach. W pierwszej kolejności zespoły inżynieryjne identyfikują i zbierają wymagania od interesariuszy, w tym przypadku inżynierów systemowych. Często na tym etapie wymagane są również spotkania z klientem w celu doprecyzowania wymagań systemowych. Niezwykle istotne jest uwzględnienie aspektów związanych z bezpieczeństwem, zarówno w kontekście zamierzonej funkcjonalności, jak i zagrożeń zewnętrznych, tak pod względem samego oprogramowania wbudowanego, a także w odniesieniu do całego pojazdu. Kolejnym etapem jest weryfikacja wymagań pod kątem ich kompletności, spójności oraz zgodności z normami branżowymi. Analiza obejmuje również identyfikację ewentualnych konfliktów między różnymi wymaganiami oraz ich wpływ na zagadnienia niefunkcjonalne, takie jak np. wydajność systemu czy zużycie zasobów. Na tym etapie zespół inżynierów jakości oprogramowania podejmuje decyzje o zastosowaniu poszczególnych metod testowania oraz technik weryfikacyjnych.

Doboru metod testowania należy dokonać na podstawie poziomu ASIL zdefiniowanego dla danego wymagania. Dla poziomu QM wystarczającą metodą jest testowanie oparte na wymaganiach (ang. Requirement based test), które polega na weryfikacji, czy system spełnia określone wymaganie funkcjonalne. Z kolei dla poziomu ASIL A metodą rekomendowaną, a dla poziomów wyższych wymaganą jest testowanie przez wstrzykiwanie błędów (ang. fault injection test). Jest to metoda, która polega na celowym wprowadzaniu systemu w błąd w celu oceny jego niezawodności i odporności na awarie. Symulowane są błędy sprzętowe oraz środowiskowe. W przypadku wymagań związanych z interfejsami komunikacyjnymi stosowane są metody:

- testowanie interfejsów zewnętrznych (ang. test of external interfaces) - testowane są magistrale komunikacyjne (np. LIN, CAN) oraz linie zasilające,
- testowanie interfejsów wewnętrznych (ang. test of internal interfaces) - testowaniu podlegają interfejsy wewnętrzne systemu (np. magistrala SPI),
- sprawdzanie spójności interfejsów (ang. interface consistency check) - jest to metoda sprawdzająca, czy poszczególne elementy systemu komunikują się w sposób zgodny z oczekiwaniami (np. konfiguracja układu i funkcji mikrokontrolera, przetworników analogowo - cyfrowych).

W przypadku gdy inżynier jakości oprogramowania podejmie decyzję o zastosowaniu innej metody testowej niż rekomendowane w standardzie ISO 26262 [8] powinien to uzgodnić z inżynierem bezpieczeństwa funkcjonalnego oraz uzasadnić w komentarzu do specyfikacji przypadku testowego.

Zespół inżynierów jakości oprogramowania współpracuje również z zespołem programistów nad opracowaniem wymaganych przez standard kryteriów weryfikacyjnych [8]. Powinny one zawierać informacje uzupełniające niezbędne do zweryfikowania wymagania:

- kryteria sukcesu,
- znane ograniczenia funkcjonalne,
- znane kwestie techniczne,
- specyficzne, istotne aspekty wymagania, które powinny być pokryte przez test,
- warunki operacyjne,
- warunki wstępne,
- określone wartości tolerancji.

Przykładowo dla następującego wymagania o poziomie bezpieczeństwa funkcjonalnego QM:

„Oprogramowanie musi regulować natężenie prądu cewki do 600mA".

metody testowe określono na „testowanie oparte o wymagania" oraz „testowanie interfejsów zewnętrznych".

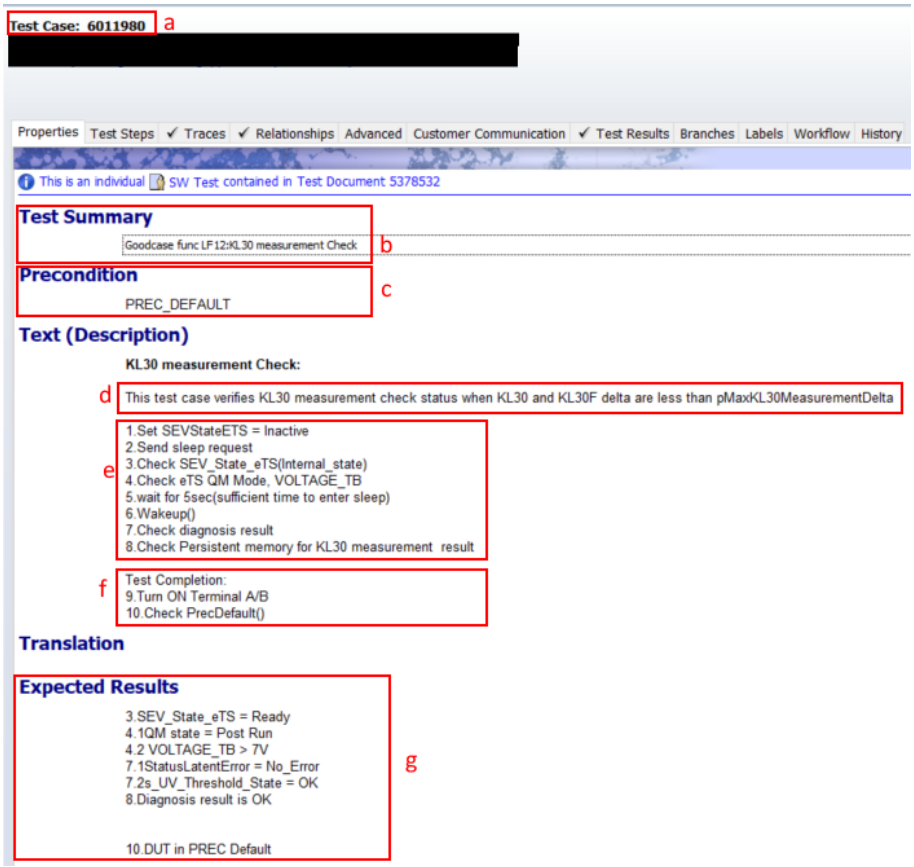
Natomiast kryteria weryfikacyjne zdefiniowano w następujący sposób:

„Pomiar prądu cewki odbywa się przy napięciu KL30C wynoszącym 6V, 12V i 16V oraz w temperaturze otoczenia 20° C. Natężenie prądu nie powinno odbiegać o więcej niż $\pm 5\%$ *od wymaganej wartości".*

Poprawność, kompletność i spójność wymagań są kluczowe dla sukcesu projektu, zwłaszcza w dziedzinie takiej jak branża motoryzacyjna, gdzie bezpieczeństwo i niezawodność odgrywają kluczową rolę. Kluczowym elementem procesu analizy wymagań przed jego zakończeniem jest recenzja. Jest to kontrola przeprowadzana wieloaspektowo, w której biorą udział wszyscy interesariusze (inżynierowie systemowi, programiści, architekci oprogramowania, inżynierowie bezpieczeństwa funkcjonalnego oraz inżynierowie jakości oprogramowania). Proces ten jest dokumentowany w postaci protokołu, w którym notowane są wszystkie uwagi sprawdzających, z przypisanymi wagami. Następnie ustalane są metody poprawy zauważonych błędów czy nieścisłości. Wprowadzone poprawki są ponownie weryfikowane przez osoby zgłaszające uwagi oraz przez pozostałych sprawdzających.

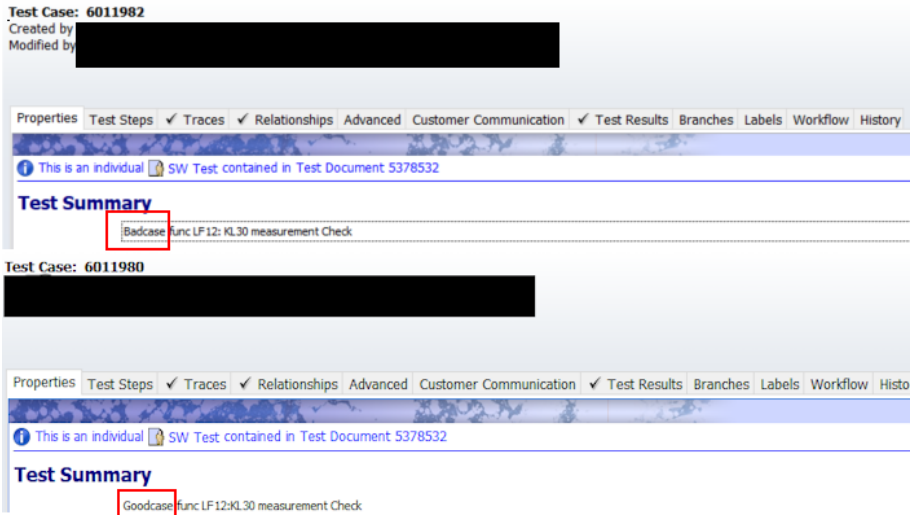
Specyfikacja scenariuszy testowych

Na podstawie zdefiniowanych wymagań inżynierowie jakości oprogramowania wbudowanego opracowują scenariusze testowe, zgromadzone w dokumencie nazywanym „Zestawem testowym" (ang. Test Suite). Dla każdego poziomu testów (SWE.4, SWE.5, SWE.6) powstaje odrębny dokument, ze względu na konieczność zapewnienia dwukierunkowej identyfikowalności (ang. traceability) wymaganej przez standard ASPICE [82]. Jest to etap w procesie testowania oprogramowania wbudowanego, który umożliwia skuteczne planowanie, przeprowadzanie i zarządzanie testami. Celem specyfikacji scenariuszy testowych jest precyzyjne zdefiniowanie procedury testowej, opartej o powiązane ze sobą przypadki testowe. Przykładowy scenariusz i wszystkie jego wymagane elementy zostały przedstawione na rysunku 4.2. Każdy przypadek testowy powinien być jednoznacznie identyfikowalny (4.2a i b). Opis przypadków testowych powinien zawierać informację o celu testu (4.2d), warunkach początkowych (4.2c), kolejnych krokach do wykonania (4.2e), oczekiwanych rezultatach (4.2g) i warunkach końcowych (4.2f).



Rys. 4.2. Przykład scenariusza testowego

Przypadki testowe powinny być zrozumiałe, zdefiniowane w sposób niegenerujący wątpliwości co do weryfikowanych wymagań. Z tego powodu ziarnistość scenariuszy testowych powinna się ograniczać do wymagania lub wymagań opisujących jedną funkcjonalność. Istotne jest jasne zdefiniowanie i rozdzielenie przypadków negatywnych i pozytywnych (na przykład w taki sposób jak pokazano na Rys. 4.3, ponieważ daje to transparentny obraz pokrycia wymagania.



Rys. 4.3. Przykład scenariusza testowego.

Warunki testowe (początkowe i końcowe) powinny być jasno określone (4.2 c i f). Ich spraw-

dzenie jest szczególnie istotne przy automatycznym wykonywaniu przypadków testowych. Właściwe zdefiniowanie warunków testowych umożliwia niezależne wykonywanie przypadków testowych, co ma ogromne znaczenie w sytuacji, gdy w danym cyklu testowym jest wykonywane tylko testowanie różnicowe (tzw. testowanie delty), czyli sprawdzenie wyłącznie zmian wprowadzonych w projektowanym systemie. Dla każdego przypadku testowego powinny zostać zdefiniowane kryteria akceptacji, które określają czy test został zakończony pomyślnie (4.2 g). Mogą one obejmować oczekiwane wyniki poszczególnych kroków, czas wykonania testu oraz inne wymagania jakościowe. Należy podkreślić, że nie spełnienie jednego z kryteriów akceptacji oznacza wynik negatywny dla przypadku testowego oraz całego scenariusza testowego. Scenariusz testowy powinien również zawierać informację o zależnościach między przypadkami testowymi, o ile takie występują. Specyfikacja przypadków testowych stanowi kluczowy dokument w procesie testowania oprogramowania wbudowanego, z tego powodu dobrą praktyką jest zaangażowanie inżynierów oprogramowania wbudowanego (architektów, programistów) w proces przeglądu (ang. review) wyspecyfikowanych scenariuszy testowych.

Analiza i wybór środowiska testowego

Wybór środowiska testowego ma istotny wpływ na skuteczność i efektywność procesu testowania. Norma ISO26262 [8] zaleca wykonywanie testów kwalifikacyjnych w jednym z trzech środowisk: docelowym pojeździe, środowisku Hardware in the Loop lub środowisku Vehicle in the Loop. Inżynier jakości oprogramowania po dokładnej analizie wymagań funkcjonalnych i niefunkcjonalnych określa kryteria wyboru środowiska testowego. Na podstawie analizy wymagań identyfikuje funkcjonalności testowe wymagające przetestowania i na tej podstawie określa narzędzia i techniki testowe niezbędne do pokrycia danej funkcjonalności. Kolejnym etapem jest analiza technologii i platform sprzętowych oraz kwalifikowanego oprogramowania wspierającego testowanie oprogramowania wbudowanego. Ocenie podlega ich zgodność z wymaganiami testowymi oraz ich zdolność do obsługi specyficznych funkcjonalności systemu. W połączeniu z oceną dostępnych zasobów (ludzkich, finansowych i materialnych) dokonuje się wyboru optymalnego środowiska testowego, umożliwiającego rzetelne i kompleksowe przetestowanie oprogramowania wbudowanego. Najczęściej stosowanym środowiskiem ze względu na optymalizację kosztów i stosunkowo proste wstrzykiwanie błędów jest środowisko Hardware in the Loop, gdzie elementy i systemy współpracujące z testowanym układem wbudowanym mogą być elementami rzeczywistymi (jak na przykład przekaźniki wysokonapięciowe dla układu sterownika baterii wysokonapięciowej), symulowanymi elektronicznie (np. układ baterii wysokonapięciowej) lub symulowanymi programowo (np. magistrale komunikacyjne w pojeździe). Inżynier jakości oprogramowania wbudowanego powinien być świadomy ograniczeń, jakie powoduje stosowanie środowiska Hardware in the Loop w porównaniu ze środowiskiem Vehicle in the Loop czy rzeczywistym pojazdem samochodowym i uwzględnić te ograniczenia w specyfikowanych scenariuszach testowych. Optymalne środowisko testowe powinno umożliwiać rzetelne przetestowanie funkcjonalności, wydajności i niezawodności.

Przygotowanie modelu pojazdu

W przypadku, gdy wybranym środowiskiem testowym jest środowisko Hardware in the Loop, należy mieć świadomość, że zjawiska zachodzące w pojeździe samochodowym i wokół niego mogą mieć wpływ na działanie testowanego układu wbudowanego. W pierwszym kroku należy zdefiniować czy i w jakim stopniu ma to znaczenie w kontekście funkcjonalności podlegających walidacji. Bardzo często stosowanym podejściem jest traktowanie pojazdu jak czarnej skrzynki, umieszczonej w środowisku testowym w stanie zamrożonym. Oznacza to, że w przygotowywanej symulacji takie informacje, jak na przykład prędkość pojazdu, natężenie prądu, temperatura czy napięcie baterii wysokonapięciowej są stałe, np. wyznaczone na podstawie średnich wartości.

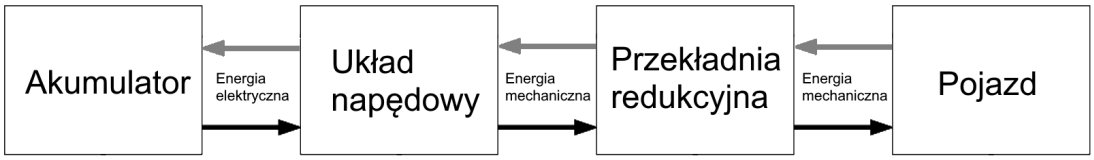
Przykładowo, przyjmuje się prędkość pojazdu 50 km/h, a stan naładowania baterii 85%. Ułatwia to wstrzykiwanie błędów związanych np. z natężeniem prądu baterii, napięciem na jej celach czy temperaturami, ponieważ podczas projektowania scenariuszy testowych zawsze znany jest stan początkowy pojazdu.

Na etapie projektowania scenariuszy testowych należy określić czy taki statyczny stan pojazdu jest poprawny i wystarczający. W przypadku, gdy dynamika pojazdu może mieć wpływ na testowane funkcjonalności, należy ją zasymulować, na przykład implementując w środowisku testowym model pojazdu.

W pracy doktorskiej zdefiniowano następujące wymagania dla modelu pojazdy elektrycznego, aby dostarczał on dynamicznych danych związanych z jazdą (uwzględniających układ napędowy) do środowiska Hardware in the Loop:

- symulacja układu napędowego, akumulatora i rekuperacji elektrycznej,
- parametryzacja napięcia, pojemności i temperatury akumulatora,
- parametryzacja wymiarów fizycznych pojazdu.

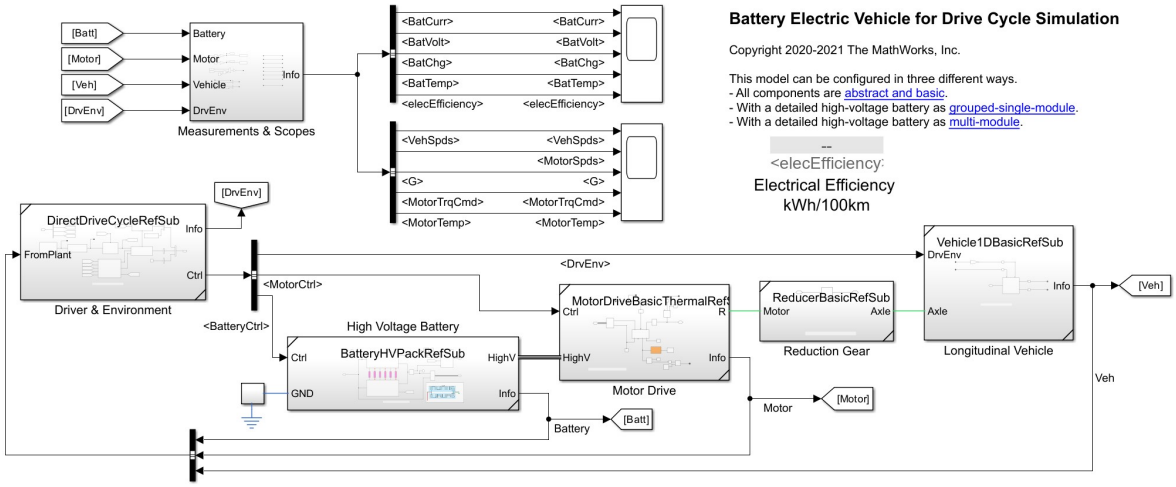
Poniższa ilustracja przedstawia ogólną architekturę modelu teoretycznego pojazdu, na którym wyróżnione zostały podstawowe rodzaje energii oraz kierunki ich przepływu.



Rys. 4.4. Architektura modelu teoretycznego

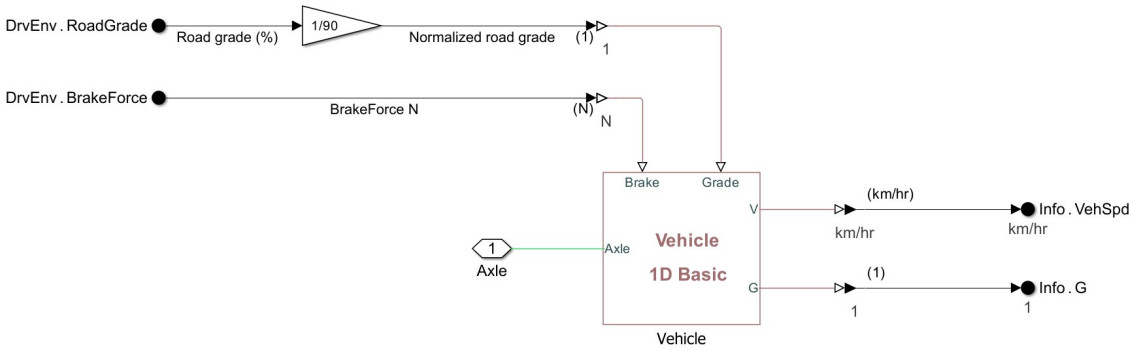
Strzałki na Rys. 4.4 ukazują przepływ energii w kontekście trybu jazdy. Czarne strzałki oznaczają zużycie energii (przyśpieszanie), natomiast szare strzałki odzyskiwanie energii (hamowanie). Warto podkreślić, że jednostka o nazwie „Układ napędowy" pełni istotną funkcję dwukierunkowej konwersji energii, będącej kluczowym aspektem realizacji procesu rekuperacji. Ma on kluczowe znaczenie dla odzwierciedlenia dynamicznego zachowania parametrów fizycznych akumulatora, takich jak natężenie prądu i stan naładowania akumulatora (ang. State of Charge — SoC). Pojęcie „energia mechaniczna" umieszczone na schemacie jest swego rodzaju abstrakcyjnym uproszczeniem, które należy doprecyzować w kontekście sił działających na pojazd, co zostało szczegółowo omówione w rozdziale 2.3.3. Implementację modeli matematycznych do modeli symulacyjnych zazwyczaj przeprowadza się w dedykowanych do tego środowiskach typu MATLAB-Simulink, Amesim czy LabView. Na rynku dostępne są różnorodne programy komputerowe. Warto zwrócić uwagę na platformę ADVISOR (Advanced Vehicle Simulator) [154], dostępną w pakiecie MATLAB-Simulink. Umożliwia ona, dzięki rozbudowanym bibliotekom, analizę różnych rozwiązań układu zasilającego, układu napędowego oraz zachowania pojazdu. Z kolei platforma PSAT (The Powertrain System Analysis Toolkit), również dostępna w środowisku MATLAB-Simulink, zawiera wiele komponentów pozwalających na symulacje pojazdów konwencjonalnych oraz elektrycznych [155].

Możliwe jest wykorzystanie modeli symulacyjnych udostępnionych na zasadach wolnego dostępu (ang. open source) [156]. Umożliwiają one symulację układu sił działających na pojazd w czasie jazdy, ich wpływu na układ napędowy oraz wielkości wyjściowe akumulatora wysokonapięciowego (m.in. natężenie prądu, napięcie, temperatura baterii) oraz układu rekuperacji [157], a nie wymaga to wykorzystania zasobów projektowych do przygotowywania kompletnego modelu. Wybrany ogólnodostępny model należy zmodyfikować, dostosowując parametry fizyczne pojazdu i baterii do rzeczywistych występujących w pojeździe docelowym dla badanego układu wbudowanego. Model symulacyjny wykorzystany do badań pokazano na rysunku 4.5.



Rys. 4.5. Model pojazdu przygotowany na podstawie Simscape-Battery-Electric-Vehicle-Model [157], [156]

W modelu przedstawionym na rysunku 4.5 wprowadzono kilka modyfikacji. Przede wszystkim dostosowano parametry fizyczne pojazdu i baterii do rzeczywistych, związanych z pojazdem, do którego dedykowany jest testowany system wbudowany. W modelu zastosowano blok „Longitudinal Vehicle", który reprezentuje abstrakcyjny pojazd ograniczony do ruchu wzdłużnego. Dzięki niemu można sparametryzować dowolny pojazd.



Rys. 4.6. Blok „Longitudal Vehicle" reprezentujący abstrakcyjny pojazd

Pokazane na schemacie sygnały to:

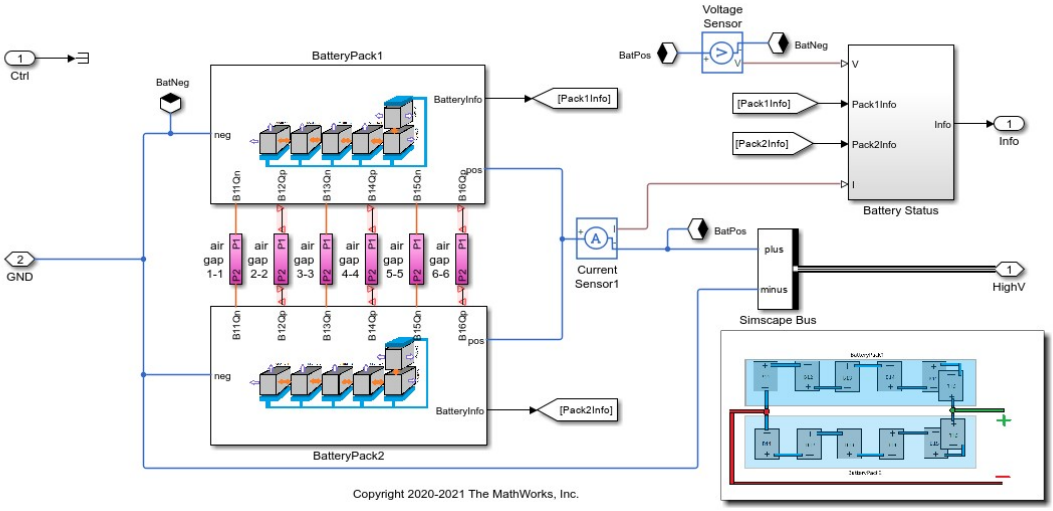
- Brake — port wejściowy sygnału fizycznego powiązanego z hamulcami. Port hamulca ignoruje wartości ujemne.
- Grade — port wejściowy powiązany z procentowym nachyleniem drogi. Procent nachylenia drogi równy 100 odpowiada kątowi $\pi/4$ radianów.
- Axle — połączenie z przekładnią redukcyjną, reprezentuje mechaniczny ruch obrotowy związany z połączeniem osi z piastą koła, związany z momentem obrotowym układu napędowego.
- VehSpd — port wyjściowy sygnału związany z prędkością wzdłużną pojazdu.
- G — port wyjściowy reprezentujący stosunek przyspieszenia wzdłużnego pojazdu do przyspieszenia grawitacyjnego.

Na Rys. 4.7 pokazano przykład, w jaki sposób zmodyfikowano parametry bloku zgodnie ze specyfikacją techniczną pojazdu, do którego dedykowany jest testowany system wbudowany.

```
%% Vehicle
%Vehicle1DBasicParams
%-{
vehicle.vehMass_kg = 2090;
vehicle.tireRollingRadius_cm = 58;
vehicle.roadLoadA_N = 455;
vehicle.roadLoadB_N_per_kph = 0;
vehicle.roadLoadC_N_per_kph2 = 0.607;
vehicle.roadGrade=0;
%}
```

Rys. 4.7. Parametry bloku „Longitudal Vehicle" zmodyfikowane do rzeczywistych wartości [158]

Drugim blokiem, którego parametry dostosowano do rzeczywistych wartości testowanego systemu wbudowanego, jest blok oznaczony na Rys. 4.5 jako „High Voltage Battery". Jest to komponent pojazdu samochodowego służący do symulacji dynamiki zestawu akumulatorów wysokiego napięcia.



Rys. 4.8. Blok „High Voltage Battery" reprezentujący zestaw akumulatorów wysokiego napięcia

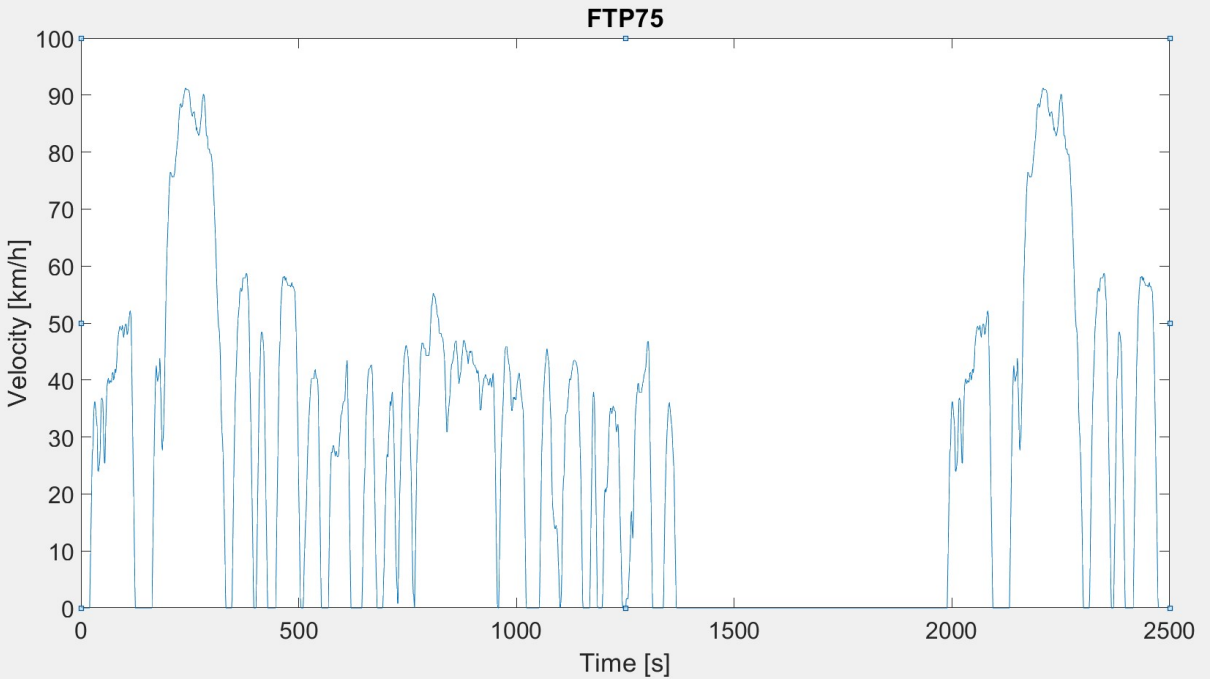
Zastosowany blok można wykorzystać do modelowania napędów hybrydowych i elektrycznych. Nie modeluje on zjawisk zanikania, starzenia, dynamiki ani właściwości baterii zależnych od temperatury. Spełnia swoją funkcję do symulacji procesu ładowania i rozładowywania, pomocny jest również w określeniu, w jaki sposób obniżone napięcie wpływa na wydajność całego systemu wbudowanego. Przykładowe parametry baterii wysokonapięciowej, które zostały dostosowane do parametrów testowanego systemu wbudowanego pokazano na rysunku 4.9.

```
%% High Voltage Battery
batteryHV.nominalCapacity_kWh = 79.2;
batteryHV.voltagePerCell_V = 4;  % Open Circuit Voltage. 3.5V to 3.7V
batteryHV.packVoltage_V = 800;
batteryHV.batteryPackAhr_ini = ...
  batteryHV.nominalCapacity_kWh / batteryHV.packVoltage_V * 1000;
batteryHV.internal_R_Ohm = 0.13;
batteryHV.AmbTemp=293.15; % Kelvin
%% Battery Pack 1
batteryHV1.batteryNp = 12;          % Number of parallel cells in a string
batteryHV1.batteryNs = 33;          % Number of series connected strings
batteryHV1.battery_N_cells = batteryHV1.batteryNp*batteryHV1.batteryNs;
batteryHV1.batteryCell_mass = 0.125; % [kg] Cell mass
batteryHV1.batteryCell_cp = 500; % [J/kg/K] Cell specific heat

batteryHV1.batterySOC_LUT = batteryHV.batterySOC_LUT;
batteryHV1.batteryTemp_LUT = batteryHV.batteryTemp_LUT;
batteryHV1.batteryCapacity_LUT = batteryHV.batteryCapacity_LUT ;
batteryHV1.batteryEm_LUT = [
    3.4966    3.5057    3.5148
    3.5519    3.5660    3.5653
    3.6183    3.6337    3.6402
    3.7066    3.7127    3.7213
    3.9131    3.9259    3.9376
    4.0748    4.0777    4.0821
    4.1923    4.1928    4.1930];
```

Rys. 4.9. Parametry bloku "High Voltage Battery"zmodyfikowane do rzeczywistych wartości [159]

Należy podkreślić, że jako źródło cyklu jazdy (ang. drive cycle) zastosowano rzeczywisty sygnał zarejestrowany na magistrali CAN pojazdu elektrycznego, określający prędkość pojazdu, podczas 40-minutowej jazdy po mieście. Przebieg tego sygnału pokazano na rysunku 4.10.



Rys. 4.10. Sygnał zastosowany jako źródło cyklu jazdy

Przygotowanie środowiska testowego

Środowisko testowe to połączenie wszystkich komponentów sprzętowych i programowych, które są używane do wykonywania testów. Aby zaplanować odpowiednio środowisko testowe, warto uzyskać przegląd planowanych działań testowych i odpowiadających im celów testowych. Niektóre czynności testowe są określone przez normy i standardy oraz wymagania klienta. Wymagane jest przygotowanie listy wszystkich istotnych cech środowiska. Należy wziąć pod uwagę następujące obszary:

- Sprzęt — komputer PC, moduły systemu testowego HIL, PCB dla testowanego urządzenia, rozmiar pamięci zewnętrznej, rodzaj połączeń testowanego urządzenia z modułami HIL (wiązki, adapter igłowy), urządzenia peryferyjne, sieć.
- Oprogramowanie — system operacyjny dla komputera PC, debuger, kompilator, oprogramowanie do fleszowania, licencje, oprogramowanie pośredniczące, sterowniki, biblioteki.
- Dane — podstawowe dane testowe, dane historyczne, aktualność danych, rozmiar danych.
- Bezpieczeństwo — konta i uprawnienia, certyfikaty konieczne do testowania.
- Administracja — wdrożenie i czas konserwacji.
- Zakupy — dostępność, czas dostawy, koszty.

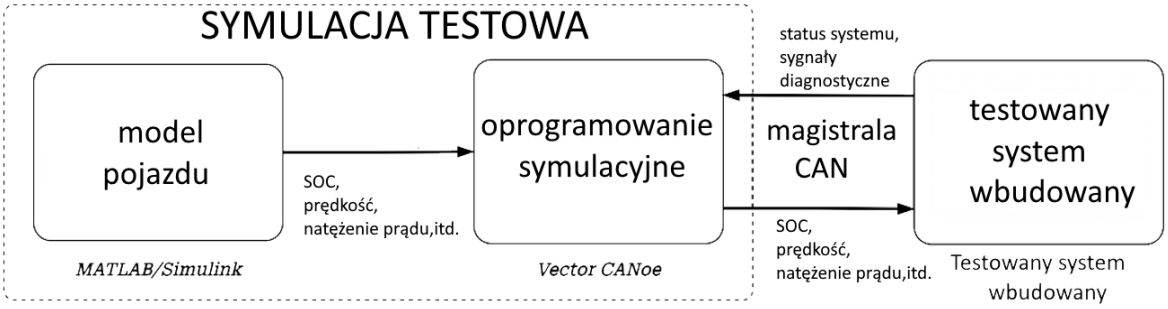
W przygotowaniu środowiska testowego pomocne będą odpowiedzi na następujące pytania:

1. *Ile interfejsów komunikacyjnych posiada testowane urządzenie i jakie to są interfejsy?* Informacja te powinna być dostępna na bardzo wczesnym etapie w postaci schematu urządzenia. Dzięki niej określa się liczbę urządzeń i kanałów komunikacyjnych potrzebnych do testowania, a także potrzebnych narzędzi do analizy protokołów. W przypadku testów tylko związanych z oprogramowaniem urządzenia komunikacyjne również mogą być niezbędne ze względu na to, że urządzenie bez utrzymywanej komunikacji będzie przechodziło w tryb uśpienia.
2. *Ile wejść oraz wyjść posiada testowane urządzenie / mikrokontroler?* Oprócz liczby warto również określić czy są to wejścia / wyjścia cyfrowe, czy analogowe.
3. *Czy funkcjonalność testowanego urządzenia jest już dobrze zdefiniowana, czy może być w późniejszych etapach modyfikowana?* Pozwala to zdecydować czy sprzęt testowy musi być modułowy, czy jego wejścia / wyjścia mogą być wyizolowane do późniejszych zastosowań.
4. *Czy planowane są testy o wysokiej precyzji lub wysokonapięciowe?* Na tej podstawie można określić jakiej klasy sprzęt będzie potrzebny do działań testowych.
5. *Ilu członków zespołu będzie używało środowiska testowego?* Na tej podstawie należy podjąć decyzję o liczbie licencji oraz zdecydować czy będzie budowane jedno, czy kilka stanowisk testowych. Należy zwrócić uwagę, że zwiększa to koszty nie tylko zakupu koniecznych narzędzi i elementów, ale także koszty utrzymania środowiska w cyklu rozwoju produktu. Z kolei współdzielenie środowiska testowego wymaga opracowania harmonogramów pracy i zasad współdzielenia zasobów.
6. *Czy będzie wymagane testowanie równolegle kilku wersji systemów wbudowanych?* Jeżeli istnieje taka konieczność należy zaplanować zarządzanie środowiskiem testowym w zależności od testowanej wersji, bądź rozważyć przygotowanie kilku wariantów środowiska testowego.
7. *Czy są planowane jakieś aktywności testowe wymagające specjalnych zasobów?* Przykładowo jeżeli planowane są testy wydajnościowe, wymagana jest odpowiednia ilość miejsca na dysku komputera PC oraz rozmiar pamięci RAM. Możliwe, że należy zaplanować również specjalny sprzęt peryferyjny np. oscyloskop, logger danych, analizator logiczny itp.

8. *Jakie oprogramowanie oraz sprzęt jest potrzebny do fleszowania / kompilowania / debugowania?* Pozwala to na zaplanowanie zakupów / instalacji koniecznych narzędzi.
9. *Czy są konieczne dodatkowe źródła zasilania?*
10. *Czy jakieś elementy środowiska testowego muszą być sterowane, zarządzane zdalnie?* Pytanie to może dotyczyć zarówno zewnętrznych źródeł zasilania (np. w celu zrestartowania debuggera), jak i elementów samego środowiska testowego (np. modułowego systemu testowego HIL).
11. *Czy istnieje projekt z podobnymi funkcjonalnościami?* Warto wykorzystać rozwiązania stosowane w innych projektach ze względu na oszczędność czasu i zasobów.
12. *Czy konieczne są specjalne konfiguracje środowiska testowego?*
13. *Czy są jakieś znane ograniczenia w środowisku testowym?* Należy określić różnice między rzeczywistym środowiskiem operacyjnym a przygotowywanym środowiskiem testowym i przeanalizować czy mogą one mieć wpływ na jakość procesu testowego.

Integracja modelu pojazdu ze środowiskiem testowym

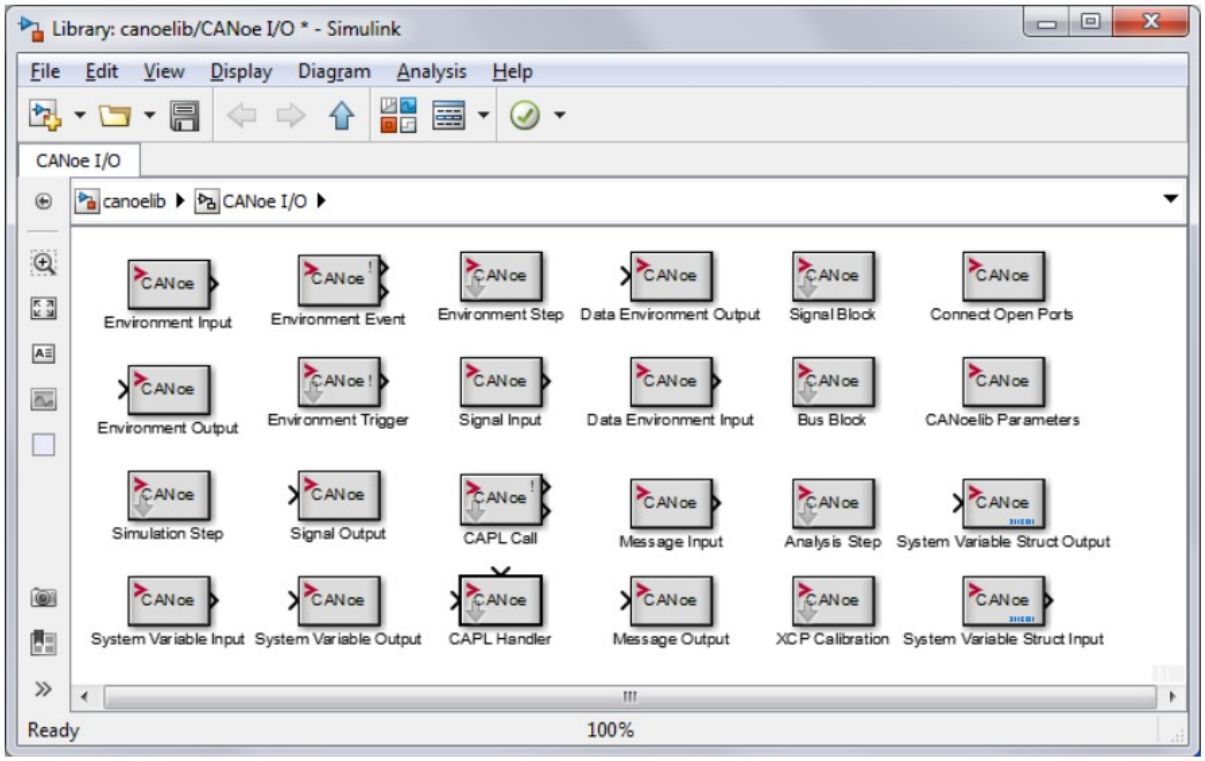
Modyfikacja środowiska testowego HIL przez wprowadzenie do niego modelu pojazdu pozwala zapewnić zbliżoną do rzeczywistej zmienność sygnałów pochodzących z pozostałej części pojazdu, uzależnić ich wartości od cyklu jazdy, oraz umożliwia sterowanie obciążeniem magistral komunikacyjnych, tak jak wyglądałoby to w rzeczywistym samochodzie. Dzięki temu możliwe jest generowanie rzeczywistych opóźnień w reakcjach systemu wbudowanego na zmiany w pojeździe wynikające z rzeczywistej sytuacji na drodze i w pojeździe.
W oprogramowaniu symulacyjnym środowiska testowego zintegrowano przedstawiony model z symulacją testową, co pokazano na rysunku 4.11.



Rys. 4.11. Model pojazdu przygotowany do integracji z oprogramowaniem symulacyjnym [160]

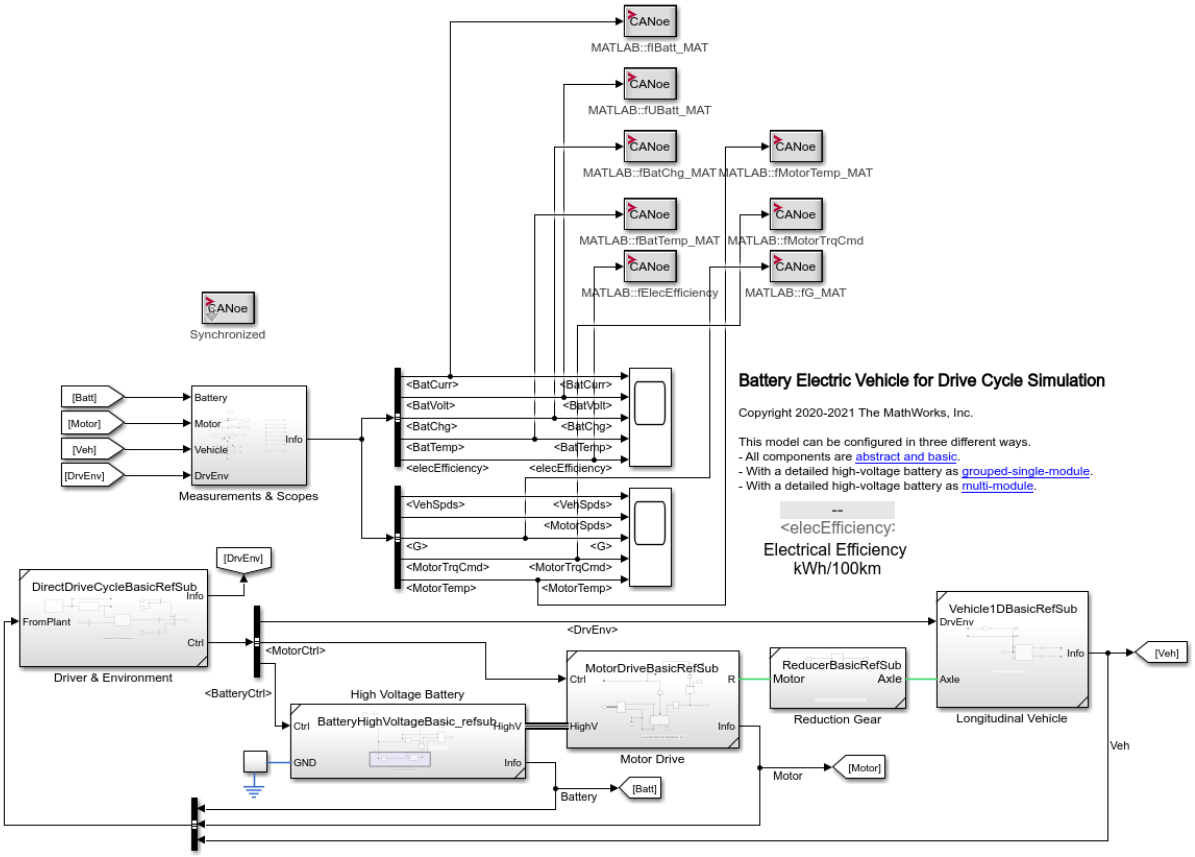
Oprogramowanie symulacyjne CANoe firmy Vector Informatik stosowane w firmie Dräxlmaier obsługuje modele Simulink. Firma Vector dostarcza wraz z oprogramowaniem CANoe bibliotekę CanoeLib, która zapewnia interfejsy do wymiany danych w postaci następujących bloków pokazanych na Rys. 4.12 obsługujących wymianę danych między:

- sygnałami,
- zmiennymi środowiskowymi,
- zmiennymi systemowymi,
- funkcjami CAPL.



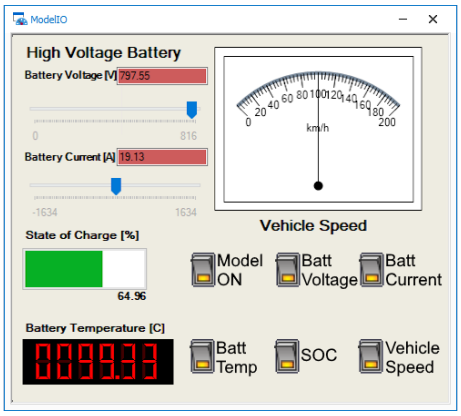
Rys. 4.12. Zbiór bloków służących do wymiany danych między modelem Simulink a CANoe

Interfejs ten dostarcza docelowy plik dla Simulink Coder, który umożliwia kompilację modelu Simulink jako biblioteki DLL, działającej w środowisku CANoe. Poza konfiguracją docelowego pliku CANoe i kompilacją modelu nie są wymagane żadne dodatkowe kroki. Chociaż model jest kompilowany, parametry bloków Simulink mogą być modyfikowane bezpośrednio z poziomu CANoe. Do celów debugowania, sygnały Simulink mogą być analizowane w CANoe. Przeglądanie modelu Simulink jest możliwe przy użyciu wbudowanego eksploratora modeli w CANoe. Co więcej, do uruchamiania lub przeglądania skompilowanego modelu nie jest wymagana licencja MATLAB/Simulink. Szczegóły dotyczące funkcjonalności poszczególnych bloków oraz możliwych sposobów integracji oprogramowania jest szeroko omówiona w dokumentacji technicznej biblioteki [161].



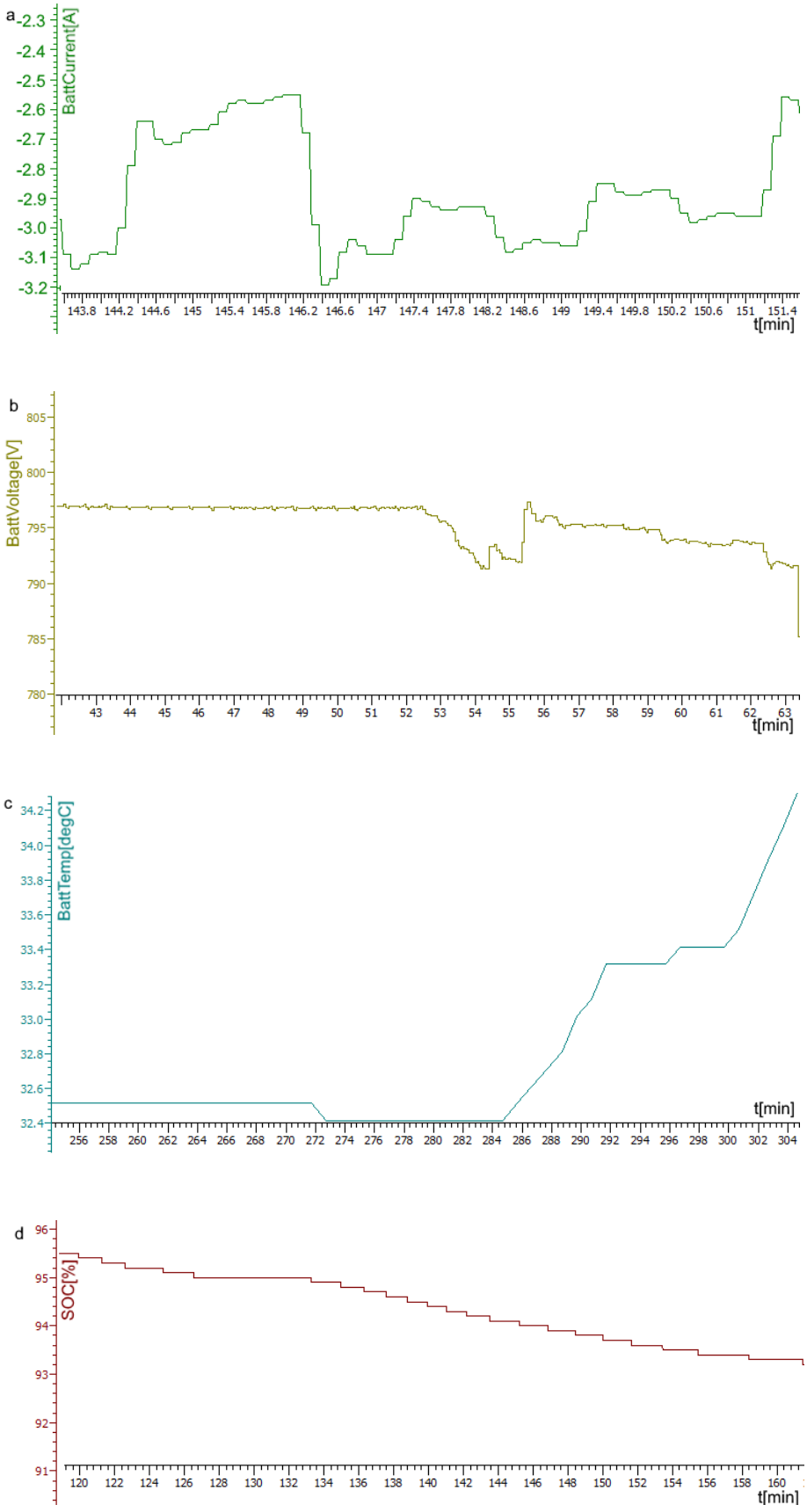
Rys. 4.13. Model pojazdu przygotowany do integracji z oprogramowaniem symulacyjnym [157]

Zaprojektowano również panel sterujący, służący do obsługi modelu pokazany na Rys. 4.14. Model, podobnie jak propagacja poszczególnych sygnałów z modelu do symulacji testowej, może być aktywowany z poziomu panelu.



Rys. 4.14. Panel służący do aktywowania modelu

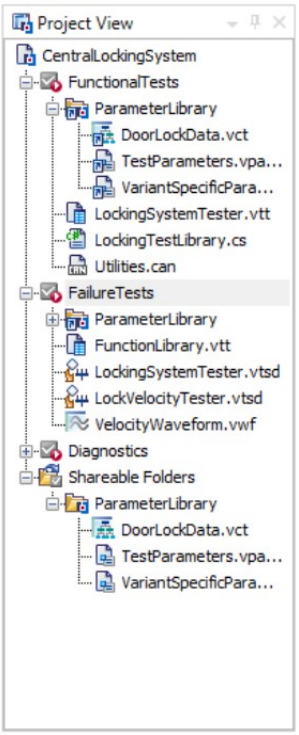
Możliwe jest również uaktywnianie modelu czy jego elementów z poziomu skryptu testowego – pozwala to wykorzystać zaproponowany model w pojedynczym przypadku testowym, a nawet tylko jego fragmencie. Na rysunku 4.15 przedstawiono przykładowe parametry symulacji, uzyskane dzięki zastosowaniu modelu pojazdu: natężenie prądu baterii (a), napięcie baterii (b), temperaturę baterii (c) i stan naładowania baterii (d).



Rys. 4.15. Przykładowe sygnały uzyskane z modelu pojazdu

Implementacja przypadków testowych

Implementacja przypadków testowych zaczyna się od określenia struktury projektu, która powinna umożliwiać łatwe poruszanie się po implementowanych skryptach testowych. Dobrą praktyką jest odzwierciedlenie struktury dokumentu, zawierającego specyfikację przypadków testowych. Podane w kolejnych krokach przykłady dotyczą oprogramowania do implementacji przypadków testowych vTestStudio firmy Vector stosowanego w firmie Dräxlmaier, ale można je przenieść na dowolne tego typu narzędzie. Przykładowy projekt, zawierający trzy jednostki testowe „FunctionalTests", „FailureTests" i „Diagnostics" pokazano na rysunku 4.16. Zalecane jest zdefiniowanie globalnej listy parametrów oprogramowania wbudowanego (zawierającej np. progi wartości fizycznych dla błędów rozpoznawanych przez system) oraz biblioteki funkcji i korzystanie z nich podczas implementacji przypadków testowych. Kolekcja „Shareable Folders" jest standardową lokalizacją, w której należy je przechowywać.



Rys. 4.16. Struktura projektu w oprogramowaniu vTestStudio [162]

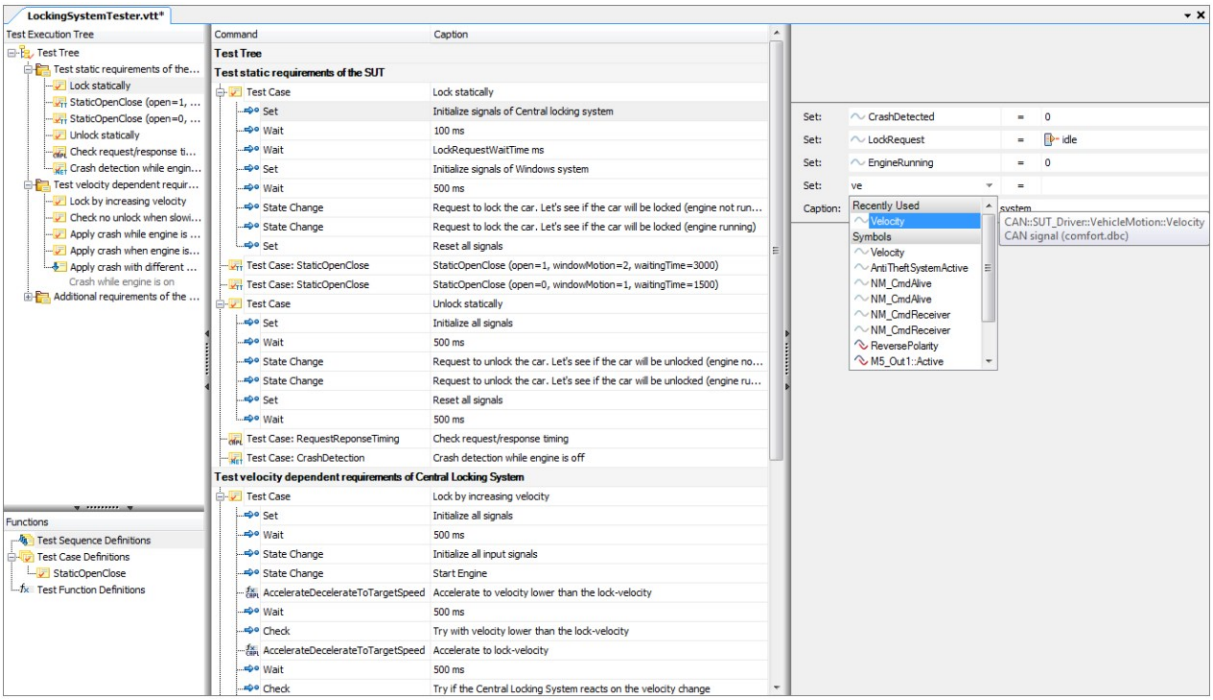
Jednostka testowa (ang. test unit) reprezentuje samodzielną konfigurowalną i wykonywalną kolekcję. Składa się z zestawu plików, które zawierają implementację testu:

- tabele testowe (Rys. 4.17),
- pliki parametrów (Rys. 4.18),
- pliki CAPL (Rys. 4.19),
- pliki C# (Rys. 4.20).

Na podstawie scenariuszy testowych abstrakcyjne przypadki testowe zostają na tym etapie przekształcone w konkretne kroki testowe. Edytor tabel testowych umożliwia użytkownikom łatwe definiowanie sekwencji testowych w formie tabelarycznej bez konieczności posiadania specjalistycznej wiedzy programistycznej. Dostępne są specjalne polecenia do stymulowania i testowania systemu wbudowanego.

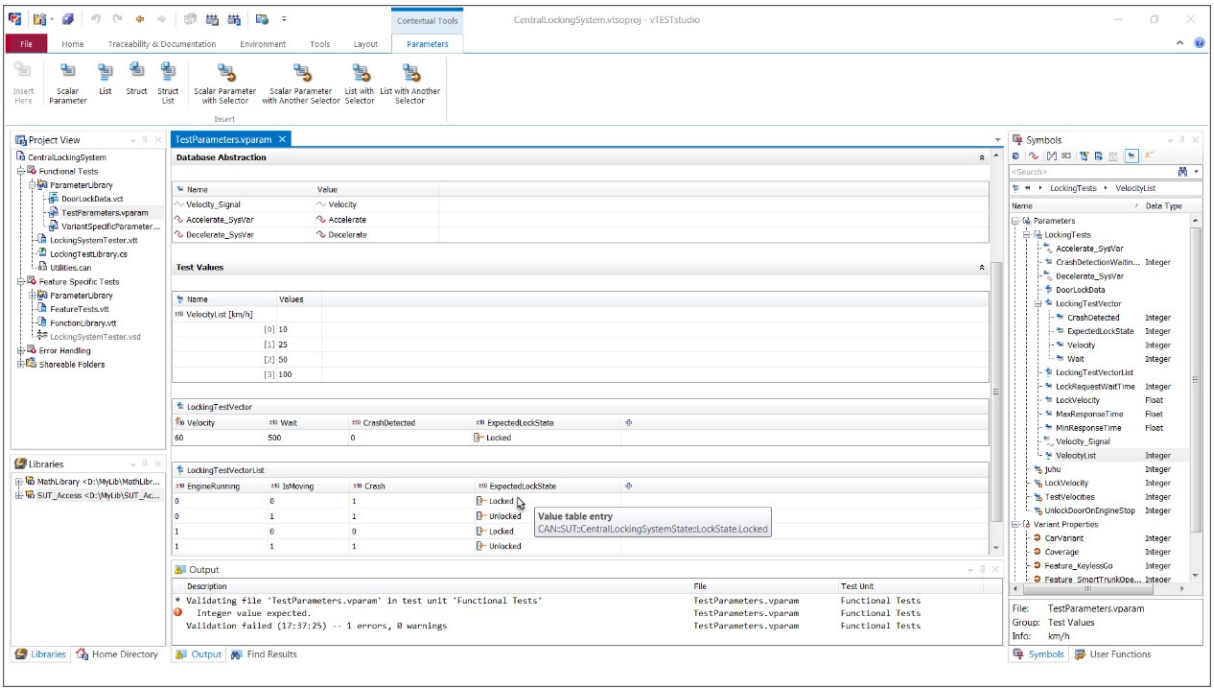
Struktura skryptu testowego powinna pokrywać się ze strukturą scenariusza testowego,

w szczególności mieć wyodrębnione trzy elementy: przygotowanie (ang. preparation), główną część testu oraz zakończenie testu (ang. completion). Przygotowanie oznacza etap wprowadzenia systemu w stan początkowy. Zazwyczaj polega ono na sprawdzeniu, czy badany system wbudowany nie jest w stanie błędu, ewentualnie na wprowadzeniu go w oczekiwany tryb pracy. Natomiast zakończenie testu jest elementem szczególnie istotnym ze względu na pomyślną automatyzację przypadków testowych. Jego zadaniem jest przywrócenie systemu wbudowanego do stanu początkowego. Jest to jedno z głównych źródeł pojawiania się zjawiska migotania testów, jednocześnie najłatwiejsze do wyeliminowania. Dobrą praktyką jest komentowanie tworzonego kodu odnośnikami do kolejnych kroków przypadku testowego oraz weryfikowanego wymagania. Ułatwia to przeprowadzanie przeglądów (ang. review) implementacji, wprowadzanie zmian oraz opisywanie defektów oprogramowania w przypadku negatywnego rezultatu testu.



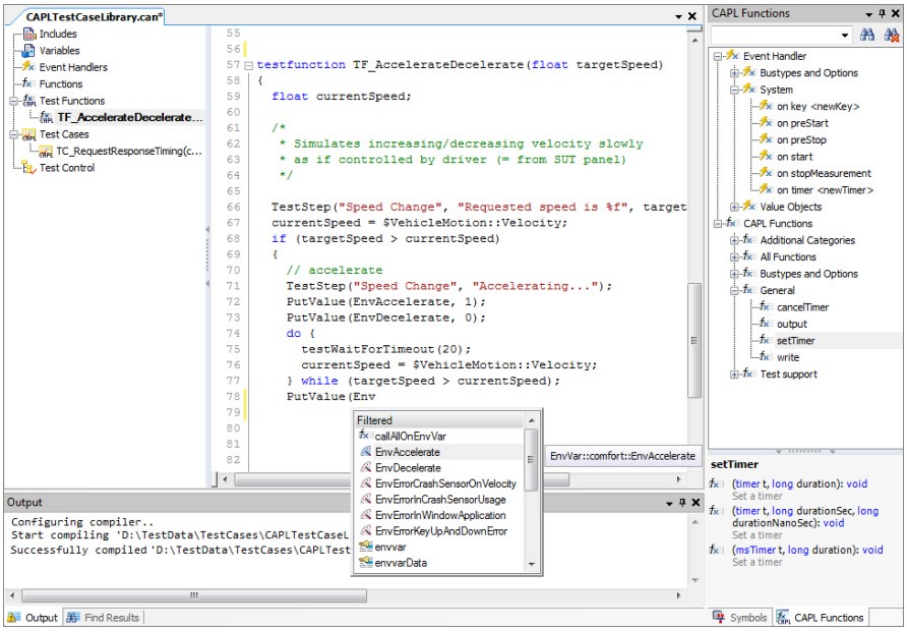
Rys. 4.17. Przykładowa tabela testowa zawierająca przypadki testowe [162]

Implementacja przypadków testowych wymaga również przygotowania danych testowych, które będą wykorzystywane podczas testowania. Dane te powinny być zgodne z założeniami testowymi. Dobrą praktyką jest gromadzenie danych testowych w przejrzystych strukturach czy tabelach, odpowiednio zdefiniowanych i opisanych, aby ułatwić utrzymywanie zaimplementowanych przypadków testowych. Definiuje się je w osobnych plikach, utrzymując hierarchiczną strukturę, dzięki zastosowaniu przestrzeni nazw.

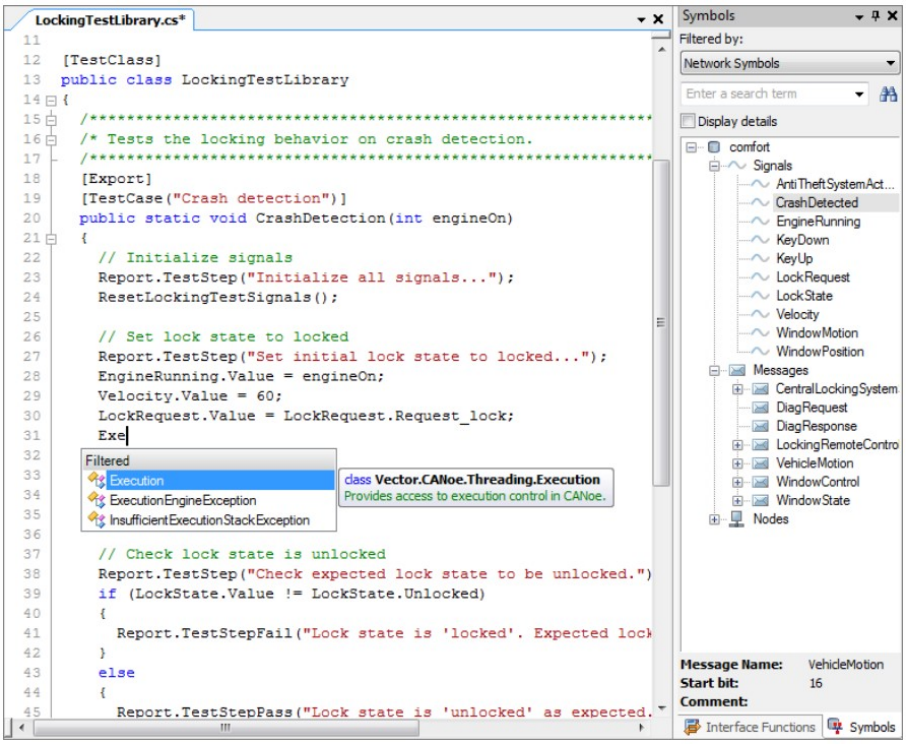


Rys. 4.18. Przykładowy plik parametrów [162]

Oprócz możliwości programowania sekwencji testowych przy użyciu predefiniowanych instrukcji w tabelach testowych vTestStudio udostępnia również edytory m.in. CAPL (Rys. 4.19) i C# (Rys. 4.20). Dzięki temu możliwe jest wykorzystanie przypadków testowych i funkcji przygotowanych w innych narzędziach, ale przede wszystkim, możliwe jest programowanie procedur zdarzeń (tzw. obsługa zdarzeń — ang. event handling). Metody tak zdefiniowane są następnie wywoływane, gdy tylko zdarzenie wystąpi podczas wykonywania testu.



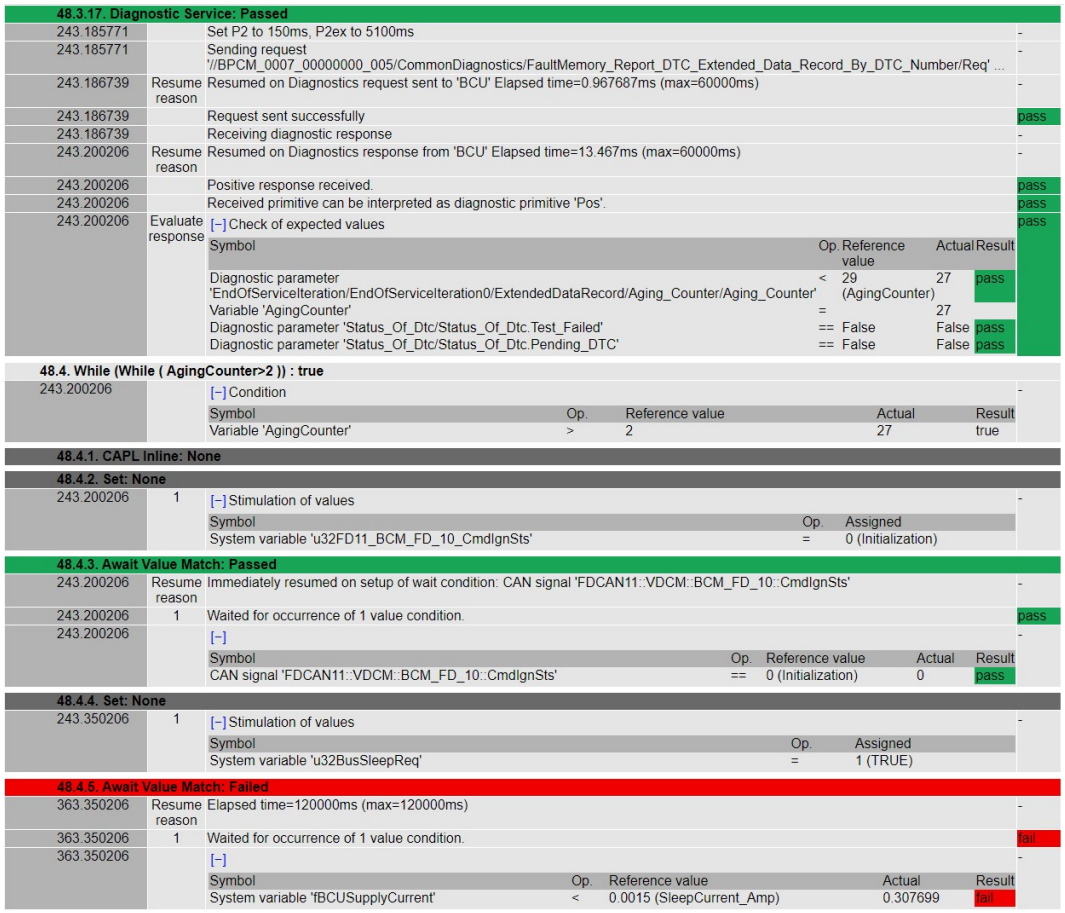
Rys. 4.19. Edytor CAPL [162]



Rys. 4.20. Edytor C# [162]

Selekcja zmiennych zależnych i niezależnych

Selekcja zmiennych zależnych polega na wyborze najbardziej istotnych zmiennych, które będą stanowiły podstawę do budowy modelu predykcyjnego. Przebieg testu systemów wbudowanych raportowany jest w pliku .html, w którym znajdują się wszystkie kroki testowe wraz z punktami czasowymi. Każda asercja występująca w teście opatrzona jest werdyktem: pozytywnym lub negatywnym. Wynik testu jest negatywny, jeżeli chociaż jedna z asercji w nim występujących ma rezultat negatywny. Na Rys. 4.21 przedstawiono fragment przykładowego raportu.



Rys. 4.21. Asercje w przykładowym raporcie

Oprócz informacji bezpośrednio związanych z wykonywanym przypadkiem testowym przedstawionych w raporcie istnieje możliwość zarejestrowania zachowania środowiska testowego w postaci pliku loga, zawierającego informacje o wszystkich zmiennych wielkościach w systemie. Pliki logów można zapisać w kilku różnych formatach:

- *binarnym* – zawierającym informacje o zmiennych wartościach zapisane binarnie, plik taki jest nieczytelny dla człowieka. Zaletą tego formatu jest jego relatywnie niewielki rozmiar.
- *ASCII* – informacje o zmiennych wartościach w środowisku zapisane są postaci znaków ASCII, czego zaletą jest to, że plik taki jest łatwy do interpretacji przez inżyniera systemów wbudowanych. Wadą tego rozwiązania jest rozmiar pliku — osiągający rozmiar do kilku GB przy testach trwających kilkanaście minut.

Na Rys. 4.22 pokazano fragment pliku loga. Należy zaznaczyć, że zawiera on informacje o komunikacji na wszystkich magistralach w systemie, o zmieniających się wartościach zmiennych systemowych wykorzystywanych do sterowania symulacją, a także o zmieniających się wartościach fizycznych w systemie (rzeczywistych i symulowanych) takich jak: natężenia prądów, napięcia, temperatury. W pliku loga widoczne są również sygnały symulowane, uzyskane za pomocą modelu pojazdu.

```
          ::VehicleSim::FD11_CanNm_Enabled = 0
4887107   243.350206    SV: 2 0 0 ::VehicleSim::u32BusSleepReq
          = 1
4887108   243.350265    SV: 6 0 0 ::ComObserver::iCan1_rx_cntr
          = L30811
4887109   243.350302    SV: 6 0 0 ::ComObserver::iCan3_rx_cntr
          = L424b5
4887110   243.350335    SV: 99 0 1 ::q_Dius_6U::IsCurrent = [ 0
          0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 ]
4887111   243.350556    SV: 6 0 0 ::ComObserver::iCan1_rx_cntr
          = L30812
4887112   243.350573    SV: 99 0 1 ::q_Dius_6U::IsVoltage = [ 0
          0 0 0 0 38 88 40 0 0 0 0 0 0 0 0 0 0 0 0 0 22 40 0
          0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
          0 0 0 0 ]
4887113   243.350588    SV: 6 0 0 ::ComObserver::iCan3_rx_cntr
          = L424b6
4887114   243.350766    SV: 6 0 0 ::ComObserver::iCan2_rx_cntr
          = Lbc849
4887115   243.350939    SV: 99 0 1 ::q_Dius_6U::SetVoltage = [
          0 0 0 0 0 38 88 40 0 0 0 0 0 0 0 0 0 0 0 0 0 22 40
          0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 9 0 0
          0 11 0 0 0 ]
4887116   243.350977    SV: 1 0 1 ::VTS::M7_Ch1::Avg = 0
4887117   243.350977    SV: 1 0 1 ::VTS::M7_Ch4::Avg = 0.00488296
4887118   243.351138    SV: 1 0 1 ::VTS::M5_Ch3::Cur = 0
4887119   243.351138    SV: 1 0 1 ::VTS::M5_Ch5::Cur = 0
4887120   243.351138    SV: 1 0 1 ::VTS::M5_Ch7::Cur = 0.136
4887121   243.351138    SV: 1 0 1 ::VTS::M5_Ch9::Cur = 0.096
4887122   243.351138    SV: 1 0 1 ::VTS::M5_Ch11::Cur = -0.02
4887123   243.351138    SV: 1 0 1 ::VTS::M5_Ch13::Cur = -0.04
4887124   243.351138    SV: 1 0 1 ::VTS::M5_Ch14::Avg = 0.096
4887125   243.351303    SV: 1 0 1 ::VTS::M4_Out1::AvgVoltage =
          11.8949
4887126   243.351303    SV: 1 0 1 ::VTS::M4_Out2::AvgVoltage =
          13.8188
4887127   243.351303    SV: 1 0 1 ::HIL::fBCUSupplyVoltage =
          11.8949
4887128   243.351426    SV: 1 0 1 ::VTS::M9_Ch3::AvgVoltage =
          3.34771
```

Rys. 4.22. Zawartość loga zapisana w czasie 1 ms wykonania testu

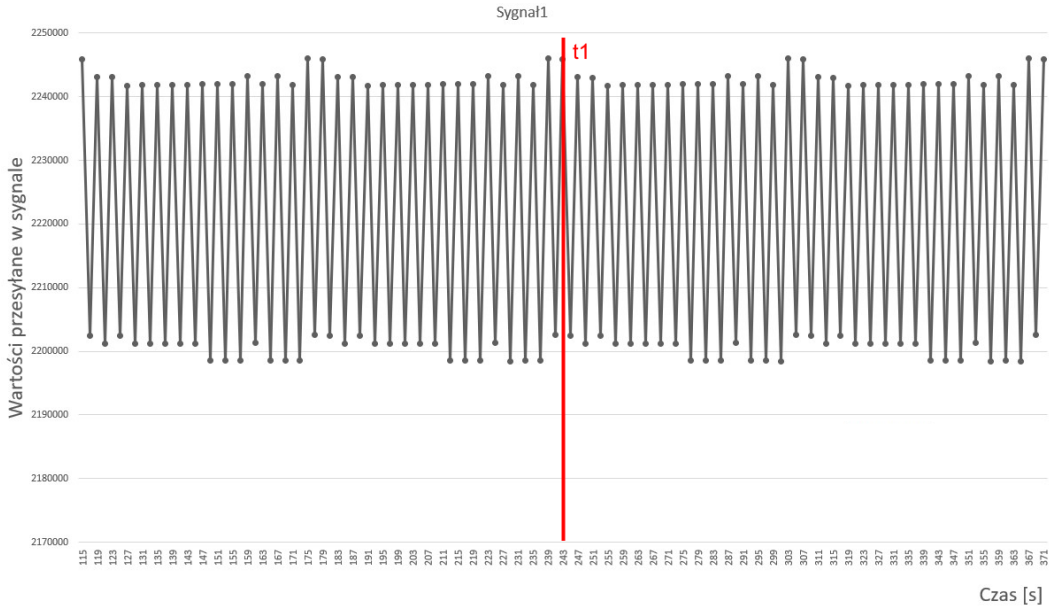
Na podstawie przedstawionego przykładu, z raportu można wyodrębnić jedną zmienną niezależną — jest nią zmienna *u32BusSleepReq*, oraz jedną zmienną zależną *fBCUSupplyCurrent*. Na podstawie tego raportu ustalono również parametr *Timeout*, niezbędny w procesie ekstrakcji cech do wyznaczenia zmiennych wyjściowych niezależnych z pliku loga. Jego wartość wyznaczono jako różnicę czasu t2 - chwili zarejestrowania w raporcie negatywnej asercji (t2=363.35) oraz czasu t1 - momentu, w którym nastąpiła manipulacja wartością zmiennej niezależnej (t1=243.35). Uzyskany w ten sposób przedział czasowy był punktem odniesienia podczas analizy zmian sygnałów i wartości fizycznych w pliku loga. Potrzebna do tego jest korelacja czasowa raportu z wykonania testu i pliku loga. Obydwa te źródła informacji powinny zawierać znaczniki czasowe, aby można było zlokalizować dane środowiskowe powiązane z negatywną asercją. Analizie poddano fragment loga odpowiadający dwom przedziałom czasowym od *t1-Timeout* do *t1* oraz od *t1* do *t2*. W pierwszym kroku wyeliminowano sygnały i wartości fizyczne nieulegające zmianom w obydwu przedziałach. Następnie przeprowadzono analizę statystyczną, której

fragment przedstawiono na Rys. 4.23, która pozwoliła wyeliminować wielkości fizyczne niewpływające na wynik testu. Jako kryterium przyjęto odchylenie standardowe różniące się dla obydwu przedziałów o więcej niż 5%.

Time	fSUP_VCCA_3V3	fBCUSupplyCurrent	fBCUSupplyVoltage	fDI_ISO_SERN_uC	u32BusSleepReq	deltaT	delta_timeout
Średnia do t1	3.6287	0.1772	11.9150	3.3871	0.8810	9.4812	0.1559
Średnia od t1	3.6309	0.3080	11.8937	3.3871	0.9971	59.8309	0.5001
Mediana do t1	3.6287	0.2633	11.9034	3.3866	1.0000	0.4169	0.1268
Mediana od t1	3.6295	0.3075	11.8937	3.3867	1.0000	59.8303	0.5001
Odchylenie standardowe do t1	0.1421	0.1392	0.0214	0.0091	0.3238	13.5802	0.1153
Odchylenie standardowe od t1	0.1473	0.0073	0.0015	0.0091	0.0538	34.7483	0.2887

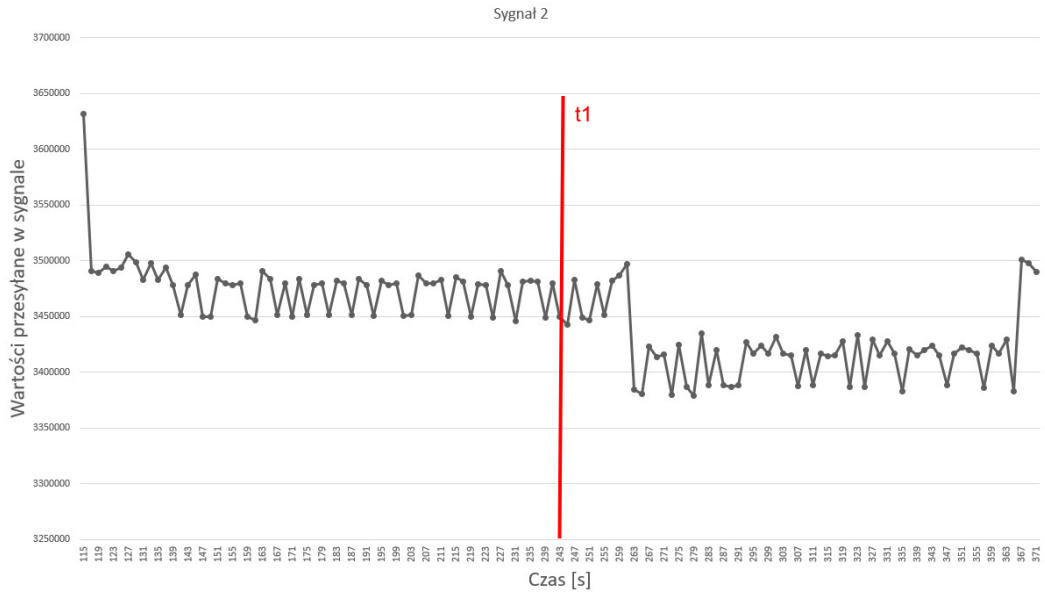
Rys. 4.23. Statystyczna analiza wielkości fizycznych pochodzących z loga

W kolejnym kroku analizie poddano sygnały komunikacyjne, zwracając uwagę na widoczną zmianę wartości w rozważanych przedziałach czasowych czy zmianę częstotliwości przesyłania sygnałów. Na Rys. 4.24 pokazano przykładowy sygnał, którego postać nie zmieniła się w założonych przedziałach czasowych. Zostanie on wyeliminowany z dalszych rozważań.



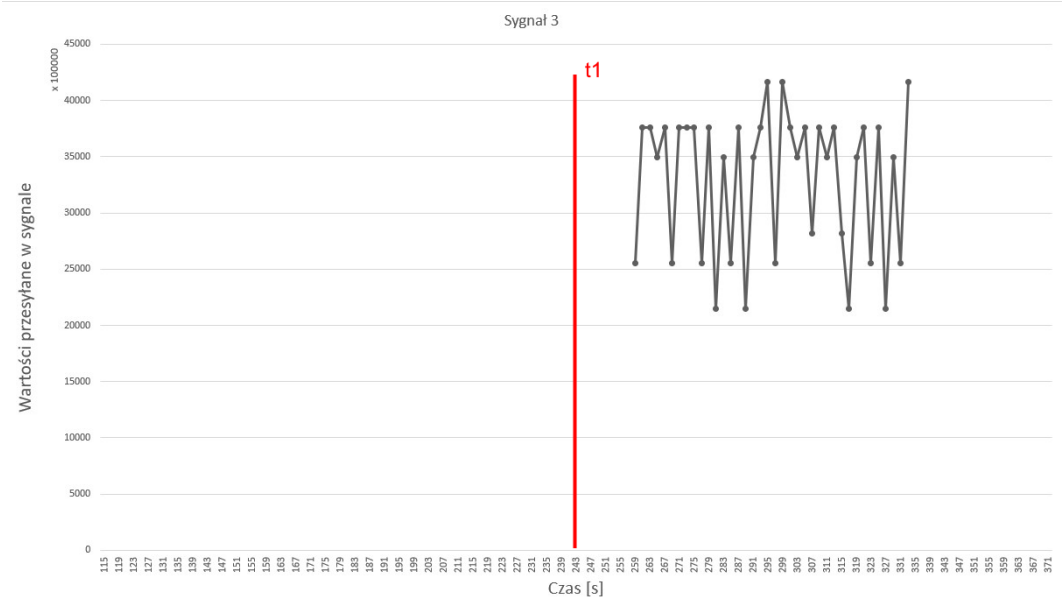
Rys. 4.24. Analiza sygnałów komunikacyjnych pochodzących z loga — sygnał bez widocznych zmian

Z kolei na Rys. 4.25 pokazano przykład, w którym wartości przesyłane w sygnale widocznie się zmieniają w przedziale czasowym od *t1* do *t2*. Sygnał ten będzie rozpatrywany jako zmienna niezależna.



Rys. 4.25. Analiza sygnałów komunikacyjnych pochodzących z loga — zmiana wartości przesyłanych w sygnale

Z kolei na Rys. 4.26 pokazano przykład, w którym w przedziale czasowym od *t1* do *t2* zmienia się częstotliwość przesyłanego sygnału. Sygnał ten będzie rozpatrywany jako kolejna zmienna niezależna.



Rys. 4.26. Analiza sygnałów komunikacyjnych pochodzących z loga — zmiana częstotliwości przesyłanego sygnału

Analiza ryzyka i opracowanie zestawu reguł

Jest to jeden z trudniejszych etapów przetwarzania danych, ponieważ konieczna jest tutaj ekspercka wiedza zarówno z zakresu znajomości funkcjonalności testowanego układu wbudowanego, jak i działania elementów, urządzeń i oprogramowania zastosowanego w środowisku.

Etap ten jest istotny wyłącznie dla przypadków, gdy do weryfikacji negatywnych rezultatów testów planowane jest zastosowanie algorytmów nadzorowanych. Pierwszym krokiem jest identyfikacja potencjalnych zagrożeń, które mogą wpłynąć na proces testowania i wyniki testów. Przykładem mogą być tutaj opóźnienia występujące na magistralach komunikacyjnych, które mogą opóźnić komunikację protokołem XCP i odczytanie wartości zmiennych w założonym czasie czy fluktuacje prądów i napięć w systemach symulacyjnych.

Na przykład dla zmiennej zależnej *fBCUSupplyCurrent* z przypadku przedstawionego w podrozdziale 4.1 w celu identyfikacji potencjalnych zagrożeń zadano następujące pytania:

1. Na jakie elementy środowiska testowego ma wpływ zmienna *u32BusSleepReq*?
2. Jaka jest zależność *fBCUSupplyCurrent* od napięcia zasilającego system wbudowany *fBCUSupplyVoltage*?
3. Czy wartość sygnałów komunikacyjnych, ich częstotliwość, zwiększone obciążenie magistral komunikacyjnych ma wpływ na wartość *fBCUSupplyCurrent*?
4. Czy obciążenie magistral komunikacyjnych ma związek ze zmienną *deltaT* (Rys. 4.23) ?
5. Co może powodować zmianę obciążenia magistral komunikacyjnych?

Następny krok polega na ocenie potencjalnych konsekwencji wystąpienia identyfikowanych ryzyk. Po ocenie ryzyka są priorytetowane według ich znaczenia i wpływu na proces testowania. Priorytetyzacja pozwala na skoncentrowaniu się na najważniejszych i najbardziej krytycznych. Na podstawie odpowiedzi na przedstawione wyżej pytania zdefiniowano ryzyka przedstawione w Tab. 4.1 i określono ich konsekwencje oraz priorytety.

Tab. 4.1. Identyfikacja przykładowych ryzyk negatywnie wpływających na rezultat testu systemu wbudowanego

Lp.	**Ryzyko**	**Wpływ na rezultat testu**	**Priorytet**
1	Utrzymana komunikacja — wiadomości kontrolujące stan sieci (wiadomości NM) są cyklicznie wysyłane.	Obsługa otrzymanych wiadomości nie pozwala na przejście systemu w stan uśpienia.	1
2	Duże fluktuacje napięcia zasilającego system powodują nieoczekiwane zmiany w natężeniu prądu *fBCUSupplyCurrent*	Następuje reset systemu.	2
3	Na magistrali komunikacyjnej pojawiają się nieoczekiwane sygnały sterujące.	Obsługa sygnałów sterujących, wysłanie odpowiedzi powoduje opóźnienie w przejściu systemu w stan uśpienia.	3
4	Zwiększone obciążenie magistral komunikacyjnych powoduje zwiększenie czasu reakcji systemu na żądanie przejścia w stan uśpienia.	Obsługa przychodzących wiadomości powoduje brak możliwości przejścia systemu w stan uśpienia.	3
5	Zmiana parametrów systemu wbudowanego (np. napięć na celach baterii wysokonapięciowej, temperatur) powoduje zwiększenie obciążenia magistral komunikacyjnych.	Dynamicznie zmieniające się parametry baterii wysokonapięciowej, szczególnie w okolicach wartości krytycznych powodują konieczność obsłużenia sytuacji wyjątkowej, ustawienia alarmów oraz wysyłania komunikatów do pozostałych jednostek sterujących pojazdem.	2

Dla zdefiniowanych ryzyk należy określić zestaw wartości zmiennych, bądź sekwencji wartości zmiennych, które posłużą do zdefiniowania reguł pozwalających na etykietowanie danych zebranych z systemów w celu trenowania algorytmów rozszerzonej inteligencji. Przykładowe reguły definiujące etykietowanie danych dla ryzyka 1. przedstawiono w Tab. 4.2.

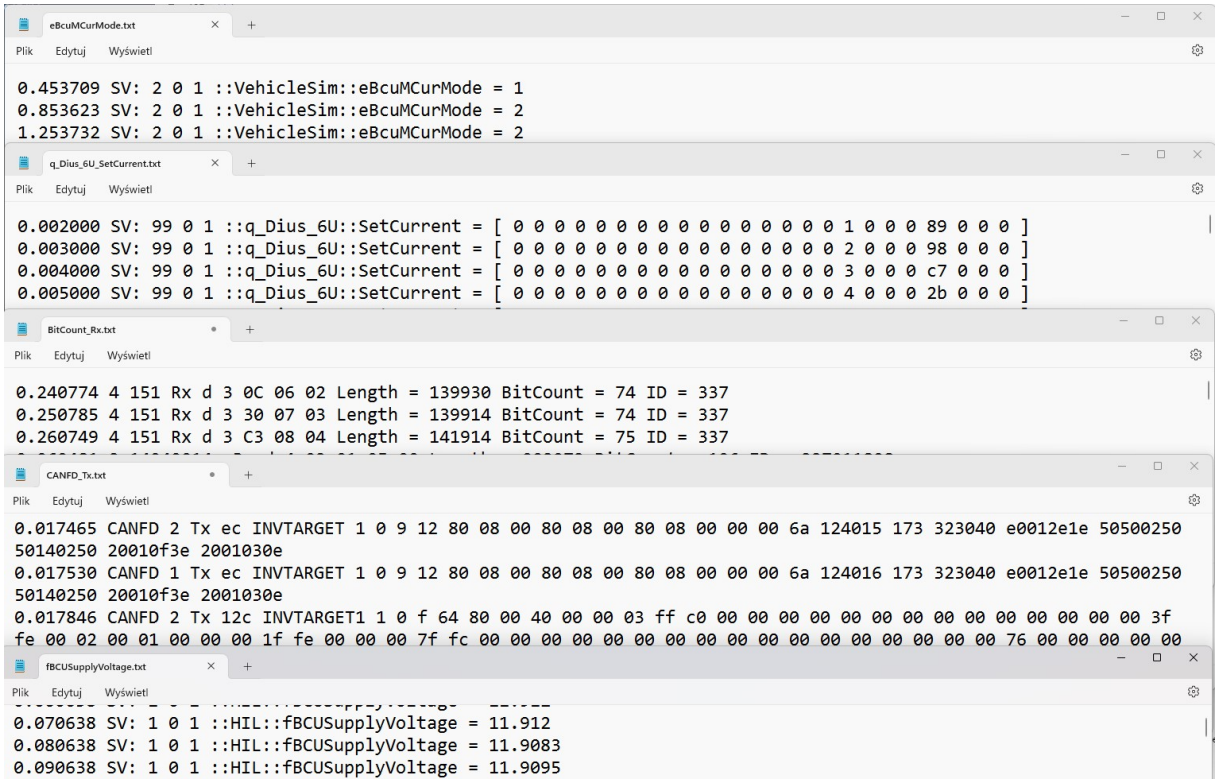
Tab. 4.2. Przykładowe reguły definiujące etykietowanie danych dla ryzyka 1

Lp.	**Reguła**	**Etykieta**
1	Jeżeli u32$BusSleepReq == 0$ i wiadomości NM obecne	0
2	Jeżeli $u32BusSleepReq == 1$ i wiadomości NM obecne i $delta\_timeout < 0.5$	0
3	Jeżeli $u32BusSleepReq == 1$ i wiadomości NM obecne i $delta\_timeout >= 0.5$	1
4	Jeżeli $u32BusSleepReq == 1$ i wiadomości NM nieobecne	0
5	Jeżeli $u32BusSleepReq == 0$ i wiadomości NM nieobecne	1

Gromadzenie danych treningowych podczas pracy systemu wbudowanego

Etap ten ma na celu zebranie i przygotowanie danych służących do trenowania algorytmów AuI wykorzystywanych w przyszłości do weryfikacji rezultatów wyników testów. Powinien on obejmować pracę systemu w dwóch wariantach. Pierwszy z nich, gdy nie są wykonywane żadne przypadki testowe, natomiast drugi to kilkukrotne wykonanie wszystkich dostępnych na danym etapie testów, ze szczególnym uwzględnieniem testów uzyskujących rezultaty migoczące. W drugim etapie istotne jest zgromadzenie danych zarówno dla wyników pozytywnych jak i negatywnych.

Zazwyczaj pliki logowane z systemów wbudowanych zawierają różnorodne informacje — wartości fizyczne parametrów systemu wbudowanego, wartości sterujące środowiskiem czy sygnały z magistral komunikacyjnych (Rys. 4.22). Aby możliwe było zastosowanie algorytmów rozszerzonej inteligencji w zadaniu weryfikacji rezultatów testów, dane zebrane podczas wykonywania przypadku testowego, w postaci raportu z wykonania testu i pliku dziennika zawierającego informacje o wszystkich wartościach i zdarzeniach w systemie testowym, muszą zostać poddane procesowi wstępnego przetwarzania (ang. preprocessing). Pierwszą operacją, którą należy wykonać, jest podział pliku loga na mniejsze pliki, zawierające informacje o takiej samej strukturze. Tworzony jest na przykład osobny plik z danymi transmitowanymi magistralą komunikacyjną CAN, osobny plik zawierający informacje o danych systemowych środowiska testowego, czy osobny plik zawierający wartości wielkości fizycznych zmierzonych w środowisku. Przykłady plików wyekstrahowanych z pliku loga pokazano na Rys. 4.27.



Rys. 4.27. Przykładowy wstępny podział pliku loga

Podział ten jest konieczny nie tylko ze względu na rozmiar pliku loga, ale przede wszystkim z powodu różnorodnej struktury danych w nim zgromadzonych. Wspomniana segregacja informacji umożliwi wyodrębnienie istotnych danych z poszczególnych wpisów logów i przygotowanie jednego pliku z danymi przygotowanymi do pracy z algorytmami rozszerzonej inteligencji. Etap ten należy rozpocząć od zaprojektowania struktury danych, która będzie później wykorzystywana w procesie trenowania algorytmów rozszerzonej inteligencji. Proponowanym rozwiązaniem jest przygotowanie danych w postaci tabelarycznej, gdzie kolumny oznaczają atrybuty. Pierwsza kolumna będzie zawierała informacje o punktach czasowych. Ustalono, jaka będzie podstawa czasu, na podstawie danych z największego utworzonego pliku — największa liczba wierszy świadczy o najczęściej zmieniających się wartościach w danym pliku, więc powinna stanowić podstawę czasu. Natomiast w kolejnych kolumnach, w odpowiednich punktach czasowych, będą gromadzone istotne informacje z poszczególnych plików. Istotne elementy tego etapu to eliminacja brakujących danych — w zależności od kontekstu brakujące dane mogą być usuwane, uzupełniane średnią lub medianą, bądź też zastępowane innymi metodami. Przykładowo dla sygnałów komunikacyjnych przesyłanych magistralą CAN, gdzie istotną informacją jest nie tylko pole danych sygnału, ale także częstotliwość jego wysyłania, brakujące wiersze uzupełniono zerami. Z kolei w przypadku parametrów fizycznych takich jak napięcie, natężenie prądu czy temperatura brakujące próbki ekstrapolowano. Jednym z ostatnich etapów jest oznaczenie danych etykietami na podstawie wcześniej przygotowanych reguł. Istotna jest również normalizacja danych, czyli skalowanie wartości atrybutów do określonego zakresu. Jeżeli zbiór zawiera dane kategoryczne, należy je przekształcić na formę numeryczną, posługując się jedną z technik: kodowanie binarne, kodowanie one — hot czy po prostu mapowanie wartości kategorycznych na liczby.

Trening algorytmów AuI

Zbiór danych zgromadzony podczas pracy systemu i odpowiednio przygotowany powinien zostać podzielony na zbiór treningowy, testowy i walidacyjny. Zbiory treningowy i testowy będą wykorzystane do trenowania algorytmów rozszerzonej inteligencji oraz strojenia parametrów modelu, natomiast zbiór walidacyjny do oceny skuteczności modelu. Przykładowy podział danych zgromadzonych w dwóch plikach logów pokazano w tabeli 4.3.

Tab. 4.3. Przykładowy podział danych uczących na zbiory: treningowy, testowy, walidacyjny

Nazwa pliku	**Zbiór**	**Liczba próbek**	**Liczba atrybutów**	**Liczba anomalii**	**% anomalii**	**Liczba negatywnych werdyktów**
A.csv	dane treningowe	173911	17	18060	10%	6
	dane testowe	57970	17	6019	10%	2
B.csv	dane walidacyjne	146179	17	12041	8%	4

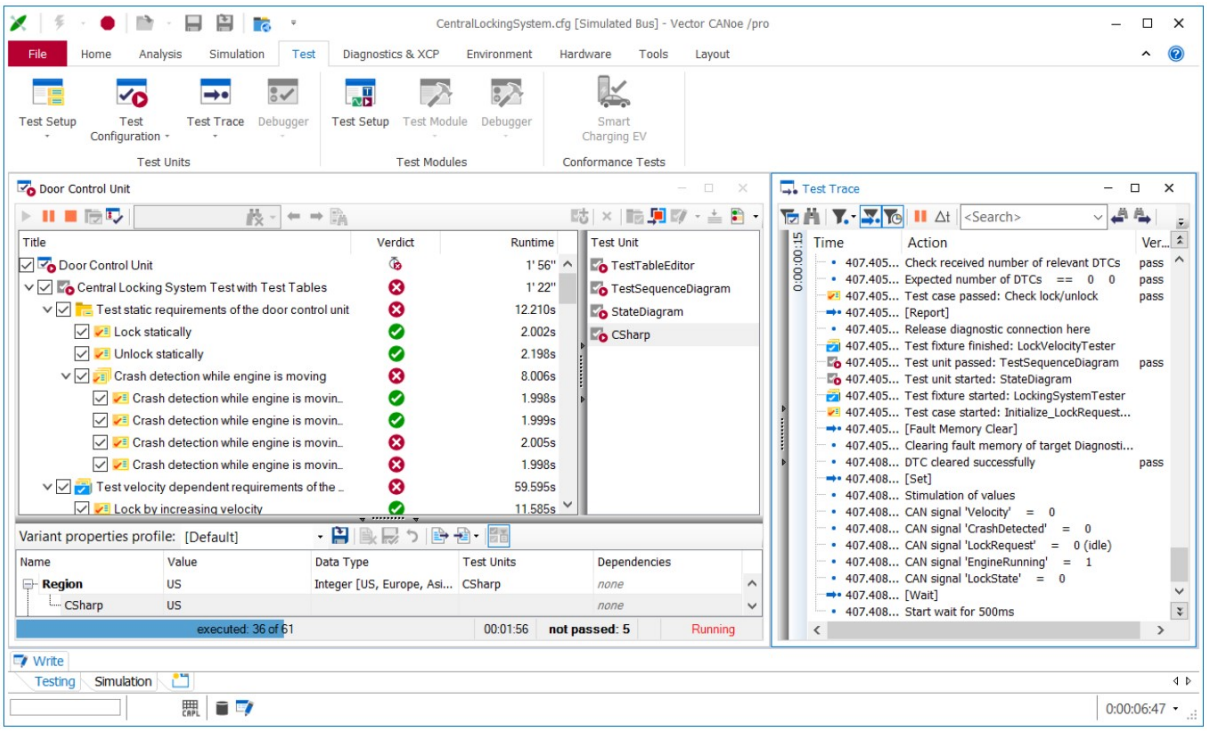
Oceny skuteczności modelu należy dokonać na podstawie różnych metryk takich jak precyzja, czułość czy macierz pomyłek. W tabeli 4.4 pokazano przykład wskaźników modelu wytrenowanego i uznanego za skuteczny do celów walidacji rezultatów uzyskanych testów. Opis metryk przedstawiono w podrozdziale 4.3.3.

Tab. 4.4. Przykładowe wskaźniki skuteczności wytrenowanego algorytmu.

	TP	**TN**	**FP**	**FN**	**PRE**	**MCC**	**BA**	**Liczba potwierdzonych negatywnych werdyktów**
Dane treningowe	4975	50769	1182	1044	0.81	0.80	0.89	2
Dane walidacyjne	10145	129965	4173	1896	0.70	0.75	0.85	3

Wykonanie przypadków testowych połączone z logowaniem danych środowiskowych

Testy wykonywane automatycznie mogę być uruchamiane bezpośrednio z oprogramowania symulacyjnego np. Vector CANoe. Na Rys. 4.28 pokazano przebieg takiego wykonania zestawu testów. W czasie rzeczywistym, w oknie śledzenia testu (ang. Test Trace), widoczne są kolejno wykonywane kroki danego testu wraz z wynikami asercji (jeżeli dany krok taką asercję zawiera).



Rys. 4.28. Wykonanie testów automatycznych w oprogramowaniu symulacyjnym [163]

Wykonywanie przypadków testowych powinno być szczegółowo dokumentowane. W raporcie z wykonania testów powinny się znaleźć następujące informacje:

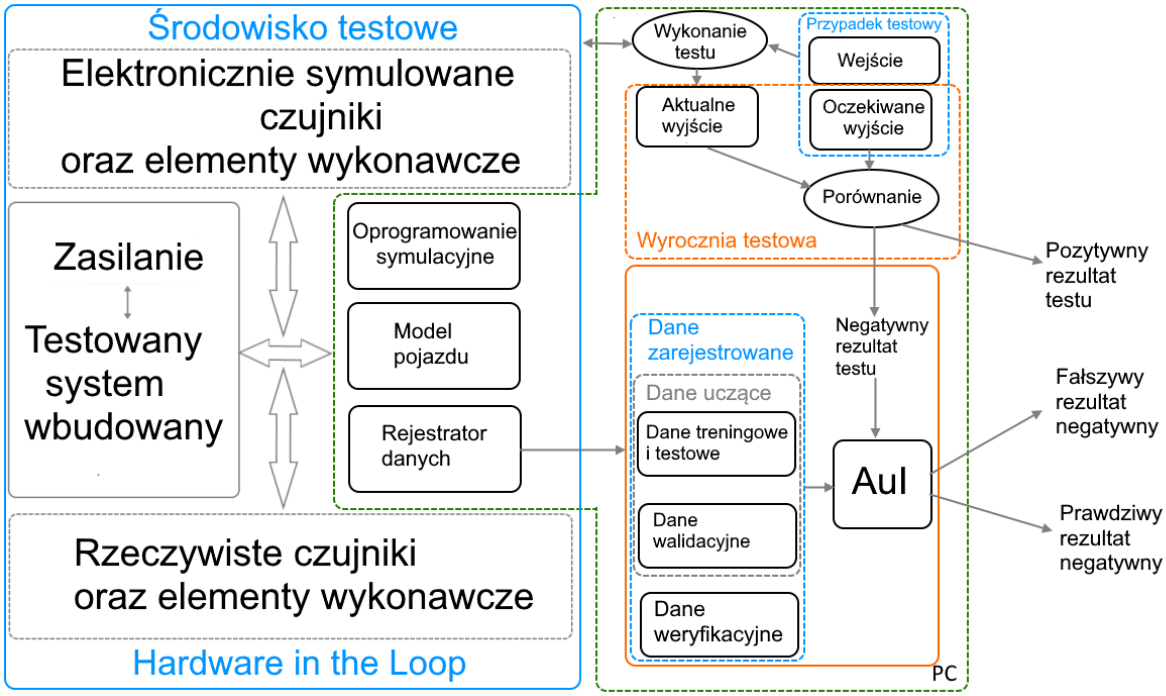
- szczegóły dotyczące środowiska testowego (wersje zastosowanego sprzętu, wersje narzędzi programowych),
- dane pozwalające zidentyfikować osobę odpowiedzialną za wykonanie testu,
- wersję systemu wbudowanego, a w szczególności wersję oprogramowania podlegającego testom,
- czas rozpoczęcia i zakończenia testów,
- informację o szczegółowych kolejnych krokach testu, ustawianych wartościach zmiennych, wstrzykiwanych błędach, poprzedzone informacją o czasie wykonania danej czynności (Rys. 4.21),
- wyniki asercji z podaniem oczekiwanych i otrzymanych rezultatów (Rys. 4.21),
- rezultat wykonanego testu.

Niezależnie od sposobu wykonywania testów, oprócz raportu, kluczowe jest także logowanie danych środowiskowych zmieniających się w czasie testu. Poprzez konsekwentne logowanie danych możliwe jest szybkie wykrywanie błędów, identyfikacja potencjalnych problemów oraz szczegółowe dokumentowanie przebiegu testu do celów raportowania i ewaluacji. Informacje zgromadzone w plikach logów powinny być optymalnie dobrane ze względu na testowane funkcje, ale także zawierać informacje o wszystkich zidentyfikowanych wcześniej zmiennych zależnych i niezależnych (podrozdział 4.1). Przed przystąpieniem do wykonywania testów mechanizm logowania danych powinien być poprawnie skonfigurowany. Oprócz identyfikacji kluczowych zmiennych (mogą to być np. parametry fizyczne, stan urządzeń, dane wejściowe czy dane z magistral komunikacyjnych), powinny zostać określone typy danych, format logów, lokalizacja oraz miejsce ich przechowywania. Należy pamiętać o zapewnieniu odpowiedniej ilości wolnego miejsca na zapisywane dane. Możliwe jest również określenie kroków, w kluczowych punktach testu, w których

dane będą logowane i uruchamianie logowania wprost ze skryptu przypadku testowego. Jest to istotne z punktu widzenia testów trwających długo (np. 12 godzin), aby ograniczyć rozmiar loga, ale zgromadzić wszystkie istotne informacje.

Weryfikacja uzyskanych rezultatów z zastosowaniem algorytmów AuI

Na rysunku 4.29 pokazano schematycznie proces weryfikacji negatywnych rezultatów testów przy pomocy algorytmów rozszerzonej inteligencji.



Rys. 4.29. Zaproponowana platforma badawcza do weryfikacji rezultatów testów

Algorytm rozszerzonej inteligencji, oznaczony na rysunki jako AuI stanowi interfejs między środowiskiem testowym, wyrocznią testową oraz inżynierem systemów wbudowanych. Na podstawie danych ze środowiska testowego zgromadzonych w czasie wykonywania testu oraz informacji z raportu algorytm AuI identyfikuje anomalie występujące w środowisku testowym i wskazuje potencjalnie fałszywe rezultaty negatywne. Przykładowe algorytmy rozszerzonej inteligencji przedstawiono w rozdziale 4.3 (Algorytm 1 i Algorytm 2).

Raportowanie wyników i zgłaszanie defektów

Ostatnim krokiem w pracy inżyniera jakości systemów wbudowanych jest raportowanie wyników. Jeżeli jednak negatywny rezultat testu został wskazany przez algorytmy AuI jako wynikający z nieprawidłowości w działaniu środowiska testowego czy niewłaściwej implementacji przypadku testowego, wprowadza się konieczne zmiany (w środowisku testowym lub skrypcie testowym), które następnie podlegają ewaluacji. Przypadek testowy zostaje wykonany ponownie. Należy podkreślić, że zmiany wprowadzone w środowisku powinny implikować ponowne wykonanie wszystkich scenariuszy testowych, również tych, które wcześniej uzyskały wyniki pozytywne. Na podstawie uzyskanych rezultatów i raportów z wykonania testów przygotowuje się końcowy raport, zawierający szczegółowe informacje o liczbie:

- wykonanych testów wraz ze spisem funkcjonalności, których dotyczyły,

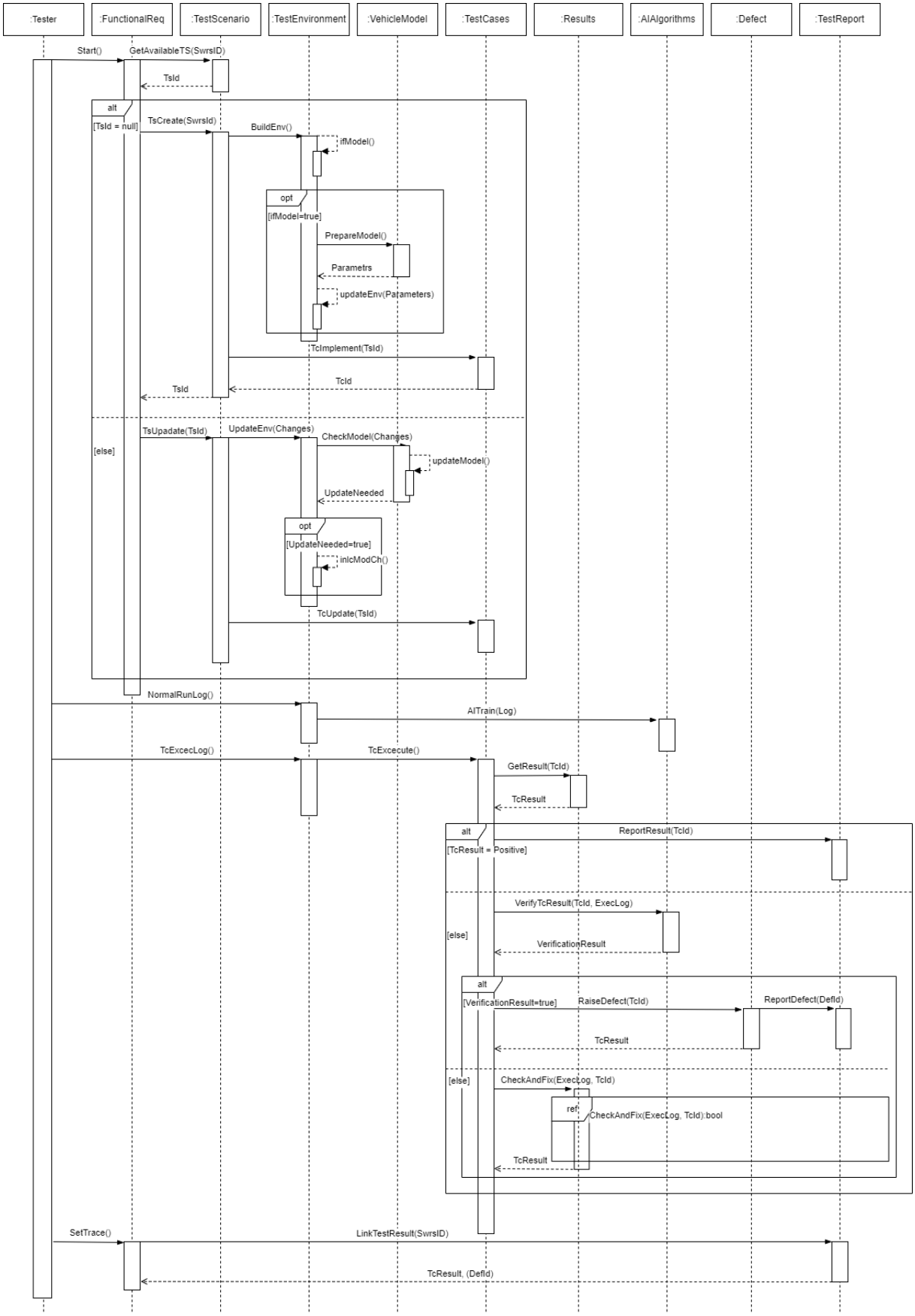
- uzyskanych wyników pozytywnych,
- uzyskanych negatywnych rezultatów wraz z listą zgłoszonych defektów.

Każdy negatywny rezultat zostaje zgłoszony jako defekt, czyli artefakt w systemie PLM (ang. project lifecycle management) opisujący szczegółowo znalezioną usterkę. Każdy defekt musi zawierać:

- dane osoby zgłaszającej,
- krótki opis usterki,
- priorytet i wagę usterki,
- dziedzinę, której dotyczy (np. specyfikacja, sprzęt, oprogramowanie),
- numer wersji oprogramowania i sprzętu, dla którego zidentyfikowano usterkę,
- opis środowiska testowego,
- warunki początkowe,
- kolejne kroki przypadku testowego, opisane w szczegółowy sposób, umożliwiający reprodukcję usterki w sposób manualny (nie wymagający uruchomienia skryptu testowego),
- oczekiwany rezultat,
- rzeczywisty rezultat,
- załączniki (raporty z wykonania testu, pliki logów, zrzuty ekranu obrazujące stan systemu itp.).

4.2 Integracja elementów metodyki testowania

Integracja elementów metodyki testowania systemów wbudowanych stanowi kluczowy aspekt w zapewnieniu spójności, niezawodności oraz efektywności procesu testowania. W niniejszym rozdziale omówiono, w jaki sposób poszczególne komponenty, takie jak środowisko testowe, scenariusze testowe, model pojazdu oraz algorytmy rozszerzonej inteligencji, są integrowane w ramach zdefiniowanej metodyki, co przestawiono schematycznie na Rys. 4.30. Z perspektywy wdrożeniowej, najważniejsze jest, aby w czasie integracji poszczególnych elementów zachować zgodność z obowiązującymi normami i standardami. Należy pamiętać, że proces rozwoju systemów wbudowanych jest cykliczny, co oznacza kilku, a nawet kilkunastokrotne przejście wszystkich procesów V-modelu zanim produkt zostanie przekazanych do produkcji.



Rys. 4.30. Integracja wymaganych elementów metodyki testowania z elementami umożliwiającymi weryfikację rezultatów z zastosowaniem algorytmów rozszerzonej inteligencji

Proces jest inicjalizowany w momencie zakończenia formalnego przeglądu wymagań funkcjonalnych. Inżynier systemów wbudowanych (:Tester) rozpoczyna analizę wymagań. Sprawdza on, czy z wymaganiami powiązane są scenariusze testowe (GetAvailableTS(SwrsID)).

W przypadku, gdy wymagania nie są powiązane z żadnymi scenariuszami testowymi (TsId= =null) przechodzi on do specyfikacji scenariuszy testowych (TsCreate()), pamiętając o tym, aby z każdym testowanym wymaganiem powiązać odpowiedni scenariusz testowy (TsId). Specyfikacja scenariusza testowego jest powiązana z koniecznością analizy i wyboru środowiska testowego (BuildEnv()). Etap ten obejmuje weryfikację, czy należy zintegrować model pojazdu ze środowiskiem testowym. Jeżeli tak, następuje przygotowanie modelu pojazdu (PrepareModel()). Może to z kolei powodować konieczność dostosowania przygotowanego środowiska testowego i uwzględnienie w nim sygnałów integrujących model ze środowiskiem (updateEnv(Parameters)). Zgodnie z obowiązującymi standardami, zarówno wyspecyfikowane scenariusze testowe, jak i samo środowisko testowe podlegają sprawdzeniu przez zespół projektowy w czasie oficjalnego przeglądu. Implementacja przypadków testowych jest ostatnim krokiem formalnego przygotowania do testowania systemu wbudowanego. Zgodnie ze standardem ASPICE, należy zapewnić pełną dwukierunkową możliwość śledzenia od wymagania, przez scenariusz testowy do przypadku testowego i rezultatu testu oraz ewentualnego defektu.

$$SwrsId \leftrightarrow TsId \leftrightarrow TcId \leftrightarrow TcResult \leftrightarrow DefId$$

Alternatywna ścieżka w przypadku, gdy wymaganie funkcjonalne było już powiązane ze scenariuszem testowym, ale uległo zmianie, obejmuje analizę zmian w wymaganiu. Następuje aktualizacja scenariusza testowego (TsUpdate(TsId)) oraz kolejno analiza zmian koniecznych do wprowadzenia w środowisku (UpdateEnv(Changes)), sprawdzenie aktualności modelu (CheckModel(Changes)), jego ewentualna aktualizacja (inclModCh()) oraz aktualizacja przypadku testowego (TcUpdate(TsId)).

Przetwarzanie danych pochodzących z symulacji oraz rzeczywistych testów jest nieodłącznym elementem integracji metodyki. Przed oficjalnym wykonaniem przypadków testowych należy przeanalizować dane opisujące działanie środowiska testowego, dokonać selekcji zmiennych zależnych i niezależnych w zależności od zaplanowanych testów do wykonania, przeanalizować ryzyko oraz opracować zestaw reguł służący do etykietowania danych. Czynności te można robić równolegle z gromadzeniem danych uczących podczas pracy systemu wbudowanego (NormalRunLog()). Trenowanie algorytmów AuI kończy etap przygotowawczy do wykonania testów.

Oficjalną weryfikację funkcjonalności systemu wbudowanego rozpoczyna wykonanie automatycznych przypadków testowych (TcExecute()) połączone z logowaniem danych środowiskowych (TcExecLog()). W przypadku negatywnego rezultatu testu (TcResult≠ Positive) następuje weryfikacja uzyskanych wyników z zastosowaniem algorytmów AuI (VerifyTcResult(TcId, ExecLog)). Jeżeli weryfikacja negatywnego rezultatu potwierdzi, że jego przyczyną jest defekt w systemie wbudowanym, a nie anomalia występująca w środowisku testowym, następuje zgłaszanie defektów (RaiseDefect(TcId) oraz ReportDefect(DefId)). W przeciwnym wypadku konieczna jest analiza przyczyn negatywnego rezultatu i próba zniwelowania wpływu środowiska na rezultat testu (CheckAndFix(ExecLog, TcId)). Ostatnim krokiem koniecznym do wykonania jest przygotowanie raportów oraz ustawienie wspomnianego wcześniej połączenia między rezultatem testu a wymaganiem funkcjonalnym.

4.3 Algorytmy sztucznej inteligencji wybrane do badań w celu zastosowania w rozszerzonej inteligencji

Rozwój systemów wbudowanych w motoryzacji, zwłaszcza w kontekście rosnącej złożoności i znaczenia bezpieczeństwa funkcjonalnego, wymaga coraz bardziej zaawansowanych metod testowania i weryfikacji. Rozszerzona inteligencja, łączy możliwości ludzkiego rozumowania z efektyw-

nością przetwarzania i analizy danych przez algorytmy sztucznej inteligencji, stanowi obiecujące podejście w tej dziedzinie [32, 164].

Celem tego podrozdziału jest przedstawienie kryteriów wyboru algorytmów sztucznej inteligencji oraz zdefiniowanie zbioru, z którego dokonano selekcji do badań nad weryfikacją wyników testów systemów wbudowanych w ramach rozszerzonej inteligencji. Zdefiniowano następujące kryteria wyboru algorytmów:

1. Skuteczność w detekcji anomalii: zadaniem wybranych algorytmów jest detekcja oraz identyfikacja nieprawidłowości w środowisku testowym, algorytmy muszą cechować się wysoką czułością i specyficznością w wykrywaniu anomalii w dużych zbiorach danych.
2. Zdolność do przetwarzania sekwencyjnego: systemy wbudowane często analizują dane o charakterze sekwencyjnym (np. dane czasowe), dlatego algorytmy muszą efektywnie radzić sobie z modelowaniem zależności czasowych.
3. Możliwość uczenia się na ograniczonych zbiorach danych: w wielu przypadkach dostępność dużych, oznaczonych zbiorów danych może być ograniczona, szczególnie w kontekście testów bezpieczeństwa. Algorytmy muszą być zdolne do efektywnego uczenia się na podstawie ograniczonej liczby przykładów.
4. Złożoność obliczeniowa i skalowalność: algorytmy muszą być skalowalne i efektywne pod względem obliczeniowym, aby mogły być stosowane w rzeczywistych warunkach, gdzie przetwarzanie dużych ilości danych w czasie rzeczywistym jest często wymagane.
5. Łatwość implementacji i utrzymania zaproponowanych rozwiązań: integracja rozszerzonej inteligencji z klasycznymi metodami testowania i weryfikacji powinna być możliwa bez konieczności angażowania inżynierów doświadczonych w tematyce sztucznej inteligencji.

W rozważaniach nad zastosowaniem rozszerzonej inteligencji do weryfikacji wyników testów uwzględniano zarówno uczenie maszynowe nadzorowane, jak i nienadzorowane. Kluczową różnicą między tymi dwoma podejściami jest wykorzystanie etykietowanych zbiorów danych. W uczeniu nadzorowanym algorytmy korzystają z danych wejściowych i odpowiadających im etykiet (czyli wzorców wyjściowych wraz z etykietami), które pozwalają określić poprawne odpowiedzi. Modele wyuczane w ten sposób zazwyczaj charakteryzują się wysoką dokładnością predykcji, ale wymagają wcześniejszego przygotowania etykietowanych danych. Wdrożenie tego rozwiązania w praktyce może się okazać nieopłacalne w rzeczywistych projektach, gdyż będzie wymagało zaplanowania odpowiedniej liczby doświadczonych pracowników oraz czasu. Z kolei w przypadku uczenia nienadzorowanego, algorytmy działają na nieoznakowanych danych, nie mając dostępu do odpowiedzi. Ich zadaniem jest samodzielne odkrycie struktury i wzorców w nieoznakowanych danych. W tym przypadku algorytmy muszą polegać na innych technikach, takich jak klastrowanie, redukcja wymiarów czy asocjacje, aby wydobyć wartościowe informacje z danych. Warto zauważyć, że mimo samodzielnego działania, modele uczenia nienadzorowanego również mogą wymagać pewnej interwencji człowieka w procesie walidacji wyników, aby upewnić się, że odkryte struktury są odpowiednie i użyteczne.

4.3.1 Rekurencyjne sieci neuronowe

W badaniach zastosowano rekurencyjne sieci neuronowe (ang. Recurrent Neural Networks - RNN), ponieważ jest to często stosowana w uczeniu głębokim metoda analizy szeregów czasowych. Struktura sieci RNN posiada pętle sprzężenia zwrotnego, czyli połączenie dostarczające sygnały wyjściowe znajdujące się w dalszych warstwach sieci neuronowej do neuronów położonych w warstwach ukrytych lub warstwie wejściowej. W teorii neurony z połączeniami rekurencyjnymi są w stanie modelować relacje długotrwałe, jednak wskutek stopniowego zanikania pierwszych próbek sygnałów po pewnym czasie stan RNN nie zawiera praktycznie żadnych śladów początkowych szeregów czasowych. Jest to związane z zanikającym gradientem dla odległych w czasie

relacji, który odbywa się wykładniczo wraz z czasem [130]. Sieci RNN zmagają się również z niestabilnością uczenia, utożsamianego często z przetrenowaniem, a określanym również jako eksplodujący gradient [165].

Poniżej przedstawiono pseudokod algorytmu rozszerzonej inteligencji opartego o rekurencyjną sieć neuronową, którego zadaniem jest weryfikacja rezultatów testów systemów wbudowanych.

Algorithm 1 Algorytm rozszerzonej inteligencji oparty o rekurencyjną sieć neuronową

Wejście: Zbiór danych treningowych $Z = \{X, Y\}$, zbiór danych testowych $Z' = \{X', Y'\}$.
Wyjście: Zbiór znaczników czasowych fałszywie negatywnych asercji Q.

Zainicjalizuj Rekurencyjną Sieć Neuronową (np. Rys. 4.31 lub 4.33).
Trenuj model Rekurencyjnej Sieci Neuronowej wektorami danych X i Y ze zbioru Z.
$i \leftarrow 0$, $j \leftarrow 0$, $k \leftarrow 0$
for all $X'[i]$ w zbiorze Z' **do**
    Prognozuj przy pomocy RNN wartości wektorów $Y''[i]$ na podstawie danych X'
    Oblicz resuiduum dla każdej predykcji $\delta \leftarrow |(Y''[i]–Y'[i])|$
    if $\delta > 1$ **then**
        if $k = 0$ **then**
            $A[j, 0] \leftarrow i$
            $k \leftarrow k + 1$
        else
            $A[j, 1] \leftarrow k$
        end if
    else
        $k \leftarrow 0$
        $j \leftarrow j + 1$
    end if
    $i \leftarrow i + 1$
$i \leftarrow 0$
$t \leftarrow 0$
for all i w zbiorze $A[i, 2]$ **do**
    $j \leftarrow A[i, 0]$
    $k \leftarrow A[i, 1]$
    if w raporcie, w przedziale czasowym $< j, k >$ występuje negatywna asercja **then**
        $Q[t] \leftarrow$ znacznik czasowy początku asercji
        $t \leftarrow t + 1$
    end if
    $i \leftarrow i + 1$
return Q

Danymi wejściowymi algorytmu rozszerzonej inteligencji (Algorytm 1) są dwa zbiory danych: zbiór Z, czyli zbiór treningowy, oraz zbiór Z', czyli zbiór testowy. Przy ocenie wielu raportów z wykonania testów w danej sesji testowej stosowany jest jeden wspólny zbiór treningowy, przygotowany przed wykonaniem przypadków testowych zgodnie z wytycznymi przedstawionymi w rozdziale 4.1. Natomiast zbiór Z', czyli zbiór testowy jest przygotowywany dla każdego weryfikowanego reportu z wykonania testów, na podstawie pliku loga zarejestrowanego w czasie egzekucji danego skryptu testowego. Zbiór X i odpowiednio X' reprezentują zmienne niezależne, natomiast Y i Y' zmienne zależne. Dla przykładu przedstawionego w rozdziale 4.1 zmienną zależną jest zmienna *fBCUSupplyCurrent*, natomiast zbiór zmiennych niezależnych został wyselekcjonowany zgodnie z zasadami określonymi we wspomnianym rozdziale i zawiera m.in. sygnał *u32BusSleepReq*.

Wynikiem działania algorytmu będzie zbiór znaczników czasowych fałszywie negatywnych asercji Q. Fałszywie negatywnymi asercjami nazywamy te negatywne werdykty zawarte w raporcie z wykonania testu, których wynik spowodowany był anomalią w środowisku testowym wykrytą przez algorytm sztucznej inteligencji. Jest to jednoznaczna informacja dla inżyniera jakości systemów wbudowanych, że otrzymany rezultat testu nie jest wiarygodny, był spowodowany anomalią występującą w środowisku testowym w czasie wykonywania testów i należy przeprowadzić dalsze analizy.

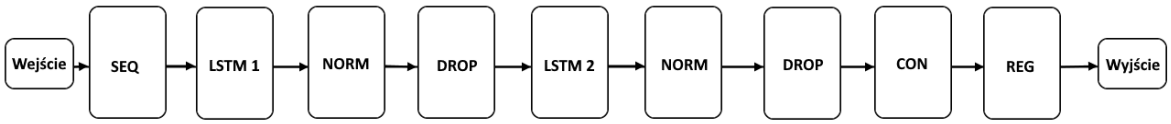
W pierwszym kroku algorytmu należy zainicjalizować zdefiniowany model rekurencyjnej sieci neuronowej, a następnie przeprowadzić trening modelu. Trzecim krokiem jest detekcja anomalii w środowisku testowym na podstawie danych testowych Z'. Polega ona na predykcji wartości Y"na podstawie danych testowych X', następnie obliczeniu residuum δ dla każdej predykcji. W kolejnym kroku (linie 7 - 17), na podstawie wartości residuum ($\delta > 1$) do tablicy A zapisywane są informacje o znaczniku czasowym odpowiadającym pojawieniu się anomalii (i) oraz jej czasie trwania (k).

Ostatni krok (linie 21 - 28) to sprawdzenie, czy wykryte anomalie miały wpływ na rezultaty wykonywanych testów. Jeżeli czas wykrycia anomalii pokrywa się ze znacznikiem czasowym negatywnej asercji w raporcie, informacja o tej asercji jest zapisywana w zbiorze wyjściowym Q.

Rekurencyjna sieć neuronowa z dwoma warstwami LSTM

W badaniach zaproponowano zastosowanie warstwy Long Short Term Memory (LSTM), która radzi sobie z problemem zanikania gradientu, korzystając z komórki pamięci: długo- i krótkotrwałej. Na pamięć długotrwałą nie wpływają bezpośrednio wagi oraz składowa stała, co pozwala na propagację informacji bez niekontrolowanych zmian wartości wewnątrz komórek sieci, nawet dla bardzo długich sekwencji danych.

Na Rys. 4.31 przedstawiono schemat blokowy modelu sieci neuronowej zbudowanego z wykorzystaniem warstwy LSTM, której zadaniem jest wykrywanie anomalii w środowisku testowym układów wbudowanych.



Rys. 4.31. Schemat blokowy modelu sieci neuronowej zawierającej warstwę LSTM

Dane pochodzące z logów zarejestrowanych podczas wykonywania testów systemów wbudowanych, odpowiednio przygotowane (rozdział 4.1) podawane są na wejście modelu, czyli do bloku oznaczonego na schemacie jako SEQ, pełniącego rolę sekwencyjnej warstwy wejściowej (ang. Sequence Input Layer). Jej zadanie polega na przygotowaniu danych do postaci uporządkowanych czasowo sekwencji, podawanych do pierwszej warstwy LSTM. Natomiast kolejna warstwa, oznaczona na schemacie blokowym jako NORM normalizuje każdy kanał wejściowy x_i w danej partii, poprzez odjęcie średniej danej partii μ_B i podzielenie przez jej odchylenie standardowe, gdzie ϵ poprawia stabilność numeryczną gdy wariancja partii σ_B^2 jest bardzo mała.

$$\widehat{x_i} = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} \tag{4.1}$$

Aby zniwelować wpływ danych wejściowych o zerowej średniej i jednostkowej wariancji na kolejne warstwy danych, wykorzystuje się zmienne przesunięcie β i zmienny współczynnik skali γ

zgodnie z zależnością [166]:

$$y_i = \gamma \widehat{x}_i + \beta \tag{4.2}$$

Warstwa oznaczona na schemacie jako DROP umożliwia efektywne łączenie różnych architektur sieci neuronowych, a przede wszystkim zapobiega nadmiernemu dopasowaniu modelu (ang. overfitting). Technika określana jako „dropout" polega na losowym usuwaniu elementów wejściowych lub ukrytych w sieci neuronowej. Usunięcie elementu oznacza tymczasowe wyłączenie jej z sieci wraz z wszystkimi jej połączenia [167]. W kolejnych krokach dane przesłane są przez kolejne warstwy LSTM, NORM i DROP.

Kolejnym blokiem modelu jest blok oznaczony jako CON, reprezentujący warstwę w pełni połączoną (ang. fully connected layer), w której wszystkie neurony łączą się z wszystkimi neuronami warstwy poprzedniej. Oznacza to, że każdy neuron tej warstwy otrzymuje sygnały (wejścia z poprzedniej), mnoży je przez macierz wag, sumuje i dodaje wektor odchylenia, następnie przepuszczając przez funkcję aktywacji, generuje jedno wyjście przekazywane do neuronów kolejnej warstwy.

Ostatnia z warstw — oznaczona na schemacie jako REG — to warstwa regresji wyjściowej, której zadaniem jest przewidywanie wartości liczbowych. Warstwa ta oblicza błąd średniokwadratowy:

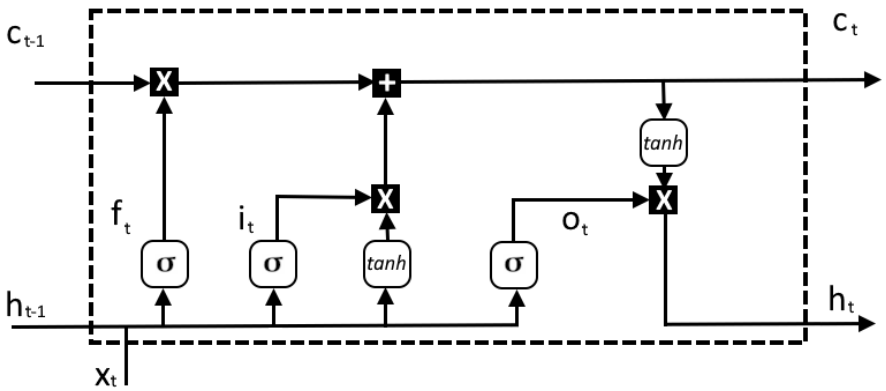
$$MSE = \sum_{i=1}^{R} \frac{(t_i - y_i)^2}{R} \tag{4.3}$$

gdzie R jest liczbą odpowiedzi, t_i docelowym wynikiem, a y_i prognozą sieci dla odpowiedzi i. Funkcja straty warstwy regresji jest określana następująco:

$$loss = \frac{1}{2} \sum_{i=1}^{R} (t_i - y_i)^2 \tag{4.4}$$

Podczas treningu algorytm oblicza średnią stratę z obserwacji w danej partii, minimalizując ją.

Warstwa LSTM jest zbudowana z komórek, które mogą przechowywać informacje w czasie i selektywnie je zapominać lub aktualizować na podstawie wejścia z bieżącego kroku czasowego i poprzednich kroków czasowych. Komórka pamięci pokazana na Rys. 4.32 jest kontrolowana przez trzy bramy: bramkę wejściową, bramkę wyjściową i bramkę zapominającą [168].



Rys. 4.32. Schemat budowy neuronu w warstwie LSTM [168]

Każda jednostka LSTM zawiera cztery bramki *(f,g,i,o)*. Otrzymuje ona wektor wejściowy x_t oraz wektor wyjściowy z poprzedniej komórki h_{t-1}. Natomiast wektorem wyjściowym jest h_t.

Stan komórki w kroku t określa formuła:

$$c_t = f_t \odot c_{t-1} + i_t \odot g_t, \tag{4.5}$$

Natomiast stan ukryty w kroku t:

$$h_t = tanh(c_t) \odot o_t, \tag{4.6}$$

Jako funkcję aktywacyjną bramki zastosowano funkcję sigmoidalną:

$$\sigma(x) = (1 + e^{-x})^{-1}, \tag{4.7}$$

a więc:

$$i_t = \sigma(W_i x_t + R_i h_{t-1} + b_i), \tag{4.8}$$

$$f_t = \sigma(W_f x_t + R_f h_{t-1} + b_f), \tag{4.9}$$

$$g_t = tanh(W_g x_t + R_g h_{t-1} + b_g), \tag{4.10}$$

$$o_t = \sigma(W_o x_t + R_o h_{t-1} + b_o), \tag{4.11}$$

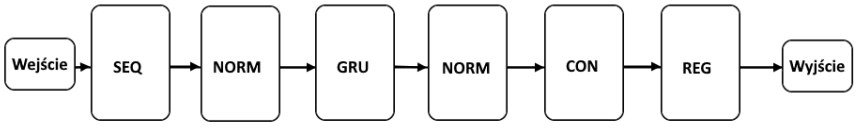
gdzie $\odot$ oznacza iloczyn skalarny, σ to funkcja sigmoidalna, tanh oznacza funkcję tangens hiperboliczny, natomiast W jest macierzą wag wejściowych, R macierzą wag rekurencyjnych, natomiast b wektorem błędu.

$$W = \begin{bmatrix} W_i \\ W_f \\ W_g \\ W_o \end{bmatrix}, R = \begin{bmatrix} R_i \\ R_f \\ R_g \\ R_o \end{bmatrix}, b = \begin{bmatrix} b_i \\ b_f \\ b_g \\ b_o \end{bmatrix}. \tag{4.12}$$

Rekurencyjna sieć neuronowa z warstwą GRU

Drugim modelem zaproponowanym do identyfikacji anomalii w środowisku testowym badanych układów wbudowanych jest rekurencyjna sieć neuronowa zawierająca jednostkę rekurencyjną z bramkami (ang. Gated Recurent Unit - GRU). Wprowadzenie warstwy GRU umożliwia lepszą kontrolę nad przepływem informacji przez sieć w czasie, a więc skuteczne modelowanie sekwencji danych poprzez zastosowanie bramek, które regulują przepływ informacji pomiędzy stanami rekurencyjnymi. W porównaniu z warstwą LSTM warstwa GRU ma mniej parametrów i jest szybsza w treningu [169].

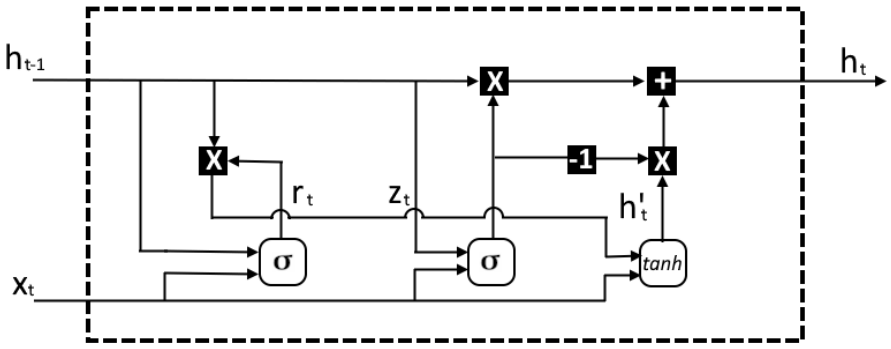
Na Rys. 4.33 przedstawiono schemat blokowy modelu sieci neuronowej zbudowanego z wykorzystaniem warstwy GRU, której zadaniem jest wykrywanie anomalii występujących w środowisku testowym zaburzających wyniki testów systemów wbudowanych.



Rys. 4.33. Schemat blokowy modelu sieci neuronowej zawierającej warstwę GRU

Podobnie jak w przypadku sieci zawierającej warstwę LSTM dane pochodzące z logów zarejestrowanych podczas wykonywania testów systemów wbudowanych podawane są do bloku SEQ. Dzięki temu blokowi dane podawane na wejście sieci neuronowej są uporządkowanymi czasowo danymi sekwencyjnymi. Przed wprowadzeniem danych do warstwy GRU poddawane są one normalizacji w bloku NORM, podobnie jak to zostało opisane dla modelu sieci zawierającego warstwę LSTM. Po przejściu przez warstwę GRU dane poddawane są kolejnej normalizacji w warstwie NORM, a następnie wprowadzane do warstwy CON, również opisanej przy okazji omawiania modelu sieci zawierającej warstwy LSTM.

Komórka pamięci w warstwie GRU składa się z dwóch bramek: resetującej i aktualizującej. Te bramki pozwalają na selektywne zapamiętywanie lub zapominanie informacji z poprzednich kroków czasowych, co jest ważne dla modelowania zależności długoterminowych w danych sekwencyjnych.



Rys. 4.34. Schemat budowy neuronu w warstwie GRU [169]

W każdym kroku czasowym warstwa GRU przyjmuje dwa wejścia: bieżące wejście i wyjście z poprzedniego kroku czasowego. Bieżące wejście jest mnożone przez bramkę aktualizacji, a poprzednie wyjście jest mnożone przez bramkę resetowania. Te dwie wartości są następnie łączone, aby wyprodukować bieżące wyjście warstwy GRU. Wektor wyjściowy h_t jest określony następująco:

$$h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \widehat{h_t} \tag{4.13}$$

Komórka sieci GRU łączy bramkę zapomnienia i bramkę wejściową sieci LSTM w bramkę aktualizacji z_t, która służy do kontrolowania ile informacji historycznych i nowych, ma być zapomnianych w bieżącym stanie. Bramka resetująca kontroluje ilość informacji dostępnych ze stanu kandydującego. Nowe i historyczne informacje wzajemnie się uzupełniają. Bramka aktualizacji kontroluje ile nowych i dawnych informacji, jest zachowywanych. Jeżeli zachowanych zostanie wiele nowych informacji, historycznych będzie mniej i odwrotnie, tak więc stare i nowe informacje mają komplementarną relację [169].

$$\widehat{h_t} = \phi(W_h x_t + U_h(r_t \odot h_{t-1}) + b_h), \tag{4.14}$$

$$z_t = \sigma(W_z x_t + U_z h_{t-1} + b_z), \tag{4.15}$$

$$r_t = \sigma(W_r x_t + U_r h_{t-1} + b_r), \tag{4.16}$$

gdzie $\odot$ oznacza iloczyn skalarny, σ to funkcja sigmoidalna, tanh oznacza funkcję tangens hiperboliczny, natomiast W i U są macierzami wag, natomiast b wektorem błędu.

$$W = \begin{bmatrix} W_z \\ W_r \\ W_h \end{bmatrix}, U = \begin{bmatrix} U_z \\ U_r \\ U_h \end{bmatrix}, b = \begin{bmatrix} b_z \\ b_r \\ b_h \end{bmatrix}. \tag{4.17}$$

4.3.2 Algorytmy do wykrywania anomalii

W wyniku dyskusji nad przydatnością wdrożeniową rozwiązań autorskich (ze względu na konieczność zapewnienia eksperta od sztucznej inteligencji do utrzymywania zaproponowanych rozwiązać) podjęto badania, które miały ocenić przydatność algorytmów sztucznej inteligencji oferowanych przez ogólnodostępne biblioteki Pythona. Decyzja ta wynikała przede wszystkim z szerokiego dostępu do zweryfikowanych algorytmów, których skuteczność została potwierdzona w licznych artykułach naukowych. W związku z tym zdecydowano się na eksplorację i ewaluację przydatności algorytmów służących do detekcji anomalii dostępnych w bibliotece PyOD (ang. Python Outlier Detection)[170]. Han i in. [171] zaprojektowali i przeprowadzili wszechstronne eksperymenty porównawcze na 14 algorytmach nienadzorowanych, 7 półnadzorowanych i 9 nadzorowanych pochodzących z biblioteki PyOD. Zbadali m.in. skuteczność algorytmów przy różnych typach anomalii. Na podstawie tych wyników oraz prac własnych [172] do badań wybranych zostało 7 algorytmów nienadzorowanych i 5 nadzorowanych. Eksperymentowanie z algorytmami z biblioteki PyOD pozwoliło na porównanie ich skuteczności z wcześniej zaproponowanymi modelami, a także na ewentualne dostosowanie wybranych rozwiązań do specyfiki badanego środowiska testowego i właściwości realizowanego projektu.

Poniżej przedstawiono pseudokod algorytmu rozszerzonej inteligencji bazujący na algorytmie sztucznej inteligencji do wykrywania anomalii, którego zadaniem jest weryfikacja rezultatów testów systemów wbudowanych.

Algorithm 2 Algorytm rozszerzonej inteligencji oparty o algorytmy AI do wykrywania wartości odstających

Wejście: Zbiór danych treningowych $Z = \{X, Y\}$, zbiór danych testowych $Z' = \{X', Y'\}$.
Wyjście: Zbiór znaczników czasowych fałszywie negatywnych asercji Q występujących w raporcie.

1: Zdefiniuj hiperparametry algorytmu.
2: Trenuj model na danych X i Y ze zbioru Z, aby nauczył się rozpoznawać wzorce normalnych danych oraz identyfikować wartości odstające $O[k]$.
3: $i \leftarrow 0$, $k \leftarrow 0$
4: $dist \leftarrow 0$
5: $waga \leftarrow 0$
6: **for all** $X'[i]$ w zbiorze Z' **do**
7: Prognozuj wartości odstające w zbiorze przy X' pomocy algorytmu AI:
8: **if** $X'[i]$ zidentyfikowano jako wartość odstającą **then**
9: **if** $dist > 0$ **then**
10: $waga \leftarrow 0$
11: **else**
12: $waga \leftarrow waga + 1$
13: **end if**
14: $O[k] \leftarrow X'[i]$
15: $W[k] \leftarrow waga$
16: $k \leftarrow k + 1$
17: $dist \leftarrow 0$

```
18:     else
19:         dist ← dist + 1
20:     end if
21:     i ← i + 1
22: i ← 0
23: t ← 0
24: for all i w zbiorze O[i] do
25:     if W[i] == 1 then
26:         j ← O[i]
27:     else
28:         if W[i] >W[i+1] then
29:             k ← W[i]
30:             if w raporcie, w przedziale czasowym < j, k > występuje negatywna asercja then
31:                 Q[t] ← znacznik czasowy początku asercji
32:                 t ← t + 1
33:             end if
34:         end if
35:     end if
36:     i ← i + 1
37: return Q
```

Danymi wejściowymi algorytmu rozszerzonej inteligencji (Algorytm 2) są dwa zbiory danych: zbiór Z, czyli zbiór treningowy, oraz zbiór Z', czyli zbiór testowy.

Podobnie jak w przypadku algorytmu opartego o sieci RNN (Algorytm 1) proces trenowania algorytmu przy ocenie wielu raportów z wykonania testów w danej sesji testowej opiera się o jeden wspólny zbiór treningowy. Z kolei zbiór testowy Z' jest przygotowywany dla każdego weryfikowanego reportu z wykonania testów, na podstawie danych zarejestrowanych w pliku loga w czasie egzekucji danego skryptu testowego. W tym przypadku również wynikiem działania algorytmu będzie zbiór znaczników czasowych fałszywie negatywnych asercji Q.

W pierwszym kroku należy zdefiniować hiperparametry stosowanego algorytmu, a następnie przeprowadzić trening modelu danymi X i Y ze zbioru Z. W trzecim kroku dane testowe poddawane są analizie przez algorytm AI, w wyniku której otrzymujemy zbiór wartości odstających $O[k]$ oraz odpowiadających im wag $W[k]$. Wagi są potrzebne, aby w następnym kroku ocenić poziom odstępstwa i dokonać wyboru próbek do dalszej analizy. Ustalane są one w liniach 9 - 20 na podstawie odległości zidentyfikowanych wartości odstających (czy są to kolejne próbki, czy nie) przy pomocy zmiennej pomocniczej *dist*. Otrzymane w ten sposób dwa zbiory: zbiór O zawierający informacje o wartościach odstających oraz zbiór W, w którym zapisano ich wagi, służą w kolejnych krokach do weryfikacji negatywnych asercji w raporcie (linie 24 - 36). W analizie tej pomijane są wartości odstające oznaczone wagą 0, traktowane jako pojedyncze, nieznaczące zaburzenia o charakterze szumu. Jeżeli czas wykrycia anomalii (od chwili wystąpienia próbki j do próbki k) pokrywa się ze znacznikiem czasowym negatywnej asercji w raporcie, informacja o tej asercji jest zapisywana w zbiorze wyjściowym Q.

Algorytmy nienadzorowane

Algorytmy nienadzorowane nie wymagają etykietowania danych treningowych, czyli wcześniejszego przygotowania danych i przypisania im odpowiednich etykiet.

1. Algorytm ECOD (ang. Empirical Cumulative Outlier Detection)

Algorytm zaproponowany przez Li i in. [173] jest oparty o estymatę empirycznej funkcji rozkładu skumulowanego danych wejściowych (ang. Empirical Cumulative Distribution Function - ECDF) w nieparametryczny sposób, co pozwala uniknąć problemu dostrajania. Problem pojawia się podczas pracy z danymi wielowymiarowymi, ze względu na konieczność oszacowania prawdopodobieństwa dla każdego wymiaru i dla każdej próbki danych. W przypadku algorytmu ECOD obliczana jest jednozmienna funkcja ECDF dla każdego wymiaru osobno. Na koniec wynik odstający dla każdej próbki danych jest obliczany przez agregację oszacowanych prawdopodobieństw w różnych wymiarach. Należy podkreślić, że jest on oparty na założeniu, że różne wymiary są niezależne. Wykazano, że wydajność algorytmu pogarsza się, gdy wartości odstające są dobrze wymieszane z normalnymi punktami (nie znajdują się w tzw. ogonach rozkładu) lub ukryte w środku normalnych punktów we wszystkich wymiarach. Jest mało prawdopodobne, aby punkty odstające przypominały normalne punkty we wszystkich wymiarach, dlatego ECOD może konsekwentnie działać dla większości zestawów danych. Podsumowując, jest to wysoce wydajny, skalowalny — pasujący do wysoce wymiarowych zestawów danych — algorytm detekcji wartości odstających, nie wymagający parametrów, o wysokiej interpretowalności, oparty na empirycznych funkcjach rozkładu skumulowanego. Ponadto ma złożoność czasową, która skaluje się liniowo wraz z rozmiarem i wymiarami zbioru danych. Dodatkowo ECOD nie wymaga ponownego szkolenia, aby dopasować się do nowych punktów danych ze stosunkowo dużymi próbkami danych, jeśli nie zakłada się przesunięcia danych. W związku z tym jest wysoce pożądany w zastosowaniach, w których potrzebne są prognozy w czasie rzeczywistym.

2. Algorytm PCA (ang. Principal Component Analysis)

W badaniach uwzględniono algorytm bazujący na analizie głównych składowych (PCA) opisanej przez Shyu i in. [174]. Analiza głównych składowych jest powszechnie stosowana do redukcji wymiarowości danych w celu ułatwienia eksploracji i dalszej analizy. PCA polega na wyjaśnieniu struktury wariancji-kowariancji zbioru zmiennych poprzez zestawienie kilku nowych zmiennych, które są kombinacjami liniowymi oryginalnych zmiennych. Te nowe zmienne, nazywane składowymi głównymi, posiadają trzy istotne właściwości:

- są nieskorelowane,
- pierwsza składowa główna ma najwyższą wariancję, a kolejne składowe mają coraz niższe wariancje,
- łączna wariancja wszystkich składowych głównych jest równa łącznej wariancji oryginalnych zmiennych.

Składowe główne z macierzy kowariancji i macierzy korelacji zwykle nie są takie same, a także nie są one prostymi funkcjami innych. Gdy poszczególne zmienne mają znacznie większą wartość niż inne, otrzymają duże wagi w głównych składowych. Z tego powodu, jeśli zmienne są mierzone w skalach o bardzo różnych zakresach lub jeśli jednostki miary nie są współmierne, lepiej jest koncentrować PCA na macierzy korelacji. Istotne jest, aby dane treningowe były pozbawione wartości odstających przed ich zastosowaniem do analizy metodą głównych składowych, gdyż obecność takich wartości może znacząco zwiększyć wariancję, kowariancję oraz korelację. Względna wielkość tych miar zmienności i zależności ma istotny wpływ na ostateczne wyniki analizy PCA, zwłaszcza dla pierwszych kilku składowych głównych. Dlatego warto rozpocząć analizę PCA od solidnego oszacowania macierzy korelacji danych treningowych. Tylko niewielka liczba składowych głównych musi zostać zachowana do późniejszego wykrywania, więc korzystną cechą jest możliwość

szybkiego obliczania statystyk podczas etapu detekcji, co z kolei umożliwia zastosowanie metody w czasie rzeczywistym. Większość zestawów danych zawiera jedną lub kilka wartości. Gdy wartość różni się od większości danych lub jest wystarczająco mało prawdopodobna w ramach założonego modelu prawdopodobieństwa danych, jest ona uważana za wartość odstającą. W przypadku danych dotyczących pojedynczej cechy wartości odstające to te, które są albo bardzo duże, albo bardzo małe w stosunku do innych. Jeżeli mamy do czynienia z rozkładem normalnym wartości to każda obserwacja, której standaryzowana wartość jest duża w wartości bezwzględnej, jest często identyfikowana jako wartość odstająca. W przypadku wielu cech sytuacja staje się jednak bardziej skomplikowana. W danych wielowymiarowych mogą występować obserwacje odstające, które nie pojawiają się jako obserwacje odstające, gdy analizuje się każdy wymiar osobno i dlatego nie zostaną wykryte na podstawie kryterium jednowymiarowego. Dlatego wszystkie cechy muszą być rozpatrywane razem przy użyciu podejścia wielowymiarowego. Procedurą powszechnie stosowaną do wykrywania wielowymiarowych wartości odstających jest pomiar odległości każdej obserwacji od środka danych. Każda obserwacja, której odległość jest większa niż wartość progowa jest uważana za wartość odstającą. Próg jest zazwyczaj określana na podstawie empirycznego rozkładu odległości. Zakładając, że wielkość próby jest duża, rozkład jest normalny oraz ponieważ główne składowe próbki są nieskorelowane poniższe równanie ma rozkład χ^2 z q stopniami swobody:

$$\sum_{n=1}^{q} \frac{y_i^2}{\lambda_i} = \frac{y_1^2}{\lambda_1} + \frac{y_2^2}{\lambda_2} + ... + \frac{y_q^2}{\lambda_q}, q \leqslant p \tag{4.18}$$

gdzie $y_1, y_2 ... y_p$ to składowe główne obserwacji $\mathbf{x}$, gdzie $\lambda_1, \lambda_2 ... \lambda_p$ to wariancje próby kolejnych składników głównych, p - ilość wymiarów danych.

Biorąc pod uwagę poziom istotności α, kryterium wykrywania wartości odstających jest następujące:

$$\sum_{n=1}^{q} \frac{y_i^2}{\lambda_i} > \chi_q^2(\alpha), \tag{4.19}$$

gdzie:

- χ_q^2 odnosi się do wartości granicznej, powyżej której leży α procent obszaru pod krzywą rozkładu χ^2,
- α to poziom istotności, czyli prawdopodobieństwo, że zostanie odrzucona hipoteza zerowa, gdy jest ona prawdziwa (błąd I rodzaju),
- q to stopnie swobody związane z liczbą zmiennych losowych w rozkładzie.

3. Algorytm AvgkNN (ang. Avarage K-Nearest Neighbors)

Algorytm ten bazuje na definicji wartości odstającej jako średniej odległości punktu p z d-wymiarowego zbioru danych do jego wszystkich k sąsiadów. Waga punktu p jest zdefiniowana jako:

$$\omega_k(p) = \sum_{i=1} k d_t(p, nn_i(p)), \tag{4.20}$$

gdzie $nn_i(p)$ oznacza i-te najbliższe sąsiedztwo punktu p, a d_t to funkcja odległości.
Punkt p należący do zbioru DB n-tym punktem odstającym względem k, oznaczany jako Out_k^n, jeśli istnieje dokładnie $n-1$ punktów q w DB takich, że $\omega_k(q) > \omega_k(p)$.

Waga $\omega_k(p)$ jest wykorzystywana do rankingowania punktów zestawu danych. W celu obliczenia tych wag jest znajdowanych k najbliższych sąsiadów każdego punktu, w szybki i wydajny sposób, poprzez linearyzację przestrzeni wyszukiwania. Dopasowywany jest d-wymiarowy zbiór danych DB do hipersześcianu $D = [0,1]^d$, a następnie D jest odwzorowywany do interwału $I = [0,1]$ za pomocą krzywej wypełnienia przestrzeni Hilberta i uzyskuje się k najbliższych sąsiadów każdego punktu przez sprawdzenie jego poprzedników i następców w I. Odwzorowanie zapewnia, że jeśli dwa punkty są blisko siebie w I, to są również blisko siebie w D. Aby ograniczyć utratę bliskości, zbiór danych jest przesuwany $d+1$ razy wzdłuż głównej przekątnej hipersześcianu $[0,2]^d$. Algorytm przedstawiony przez Angiulli'ego i Pizzuti'ego [175] składa się z dwóch faz: pierwsza faza zapewnia przybliżone rozwiązanie, z niewielkim współczynnikiem, po wykonaniu co najwyżej d+1 skanów zbioru danych (gdzie d jest liczbą wymiarów danych), przy niskim koszcie złożoności czasowej. Podczas każdego skanowania uzyskiwana jest lepsza dolna granica dla wagi k-tej wartości odstającej DB i liczba punktów kandydujących do zbioru rozwiązań jest rozsądnie zmniejszana. Druga faza zwraca dokładne rozwiązanie, wykonując pojedynczą analizę niewielkiej części zbioru danych.

Punkty odstające to punkty o największych wartościach. Algorytm ten może być stosowany w zadaniach klasyfikacji i regresji. Skaluje się on liniowo zarówno w odniesieniu do wymiarowości, jak i rozmiaru zbioru danych.

4. Algorytm IForest (ang. Isolation Forest)

Algorytm bazujący na koncepcji lasu izolacyjnego, czyli izolowania anomalii w mniejszych podziałach za pomocą zestawu drzew izolacyjnych. Metoda zaproponowana przez Liu i in.[176] wykorzystuje dwie ilościowe właściwości anomalii, które czynią je bardziej podatnymi na izolację niż normalne punkty:

- stanowią one mniejszość, składającą się z mniejszej liczby instancji,
- mają wartości atrybutów, które bardzo różnią się od wartości normalnych instancji.

Ze względu na wysoką podatność na izolację anomalie są izolowane bliżej korzenia drzewa. W tej metodzie występują tylko dwie zmienne: liczba drzew do zbudowania i rozmiar próbkowania.

Wykrywanie anomalii przy użyciu algorytmu bazującego na iForest jest procesem dwuetapowym. Pierwszy etap (szkoleniowy) buduje drzewa izolacyjne przy użyciu podprób z zestawu szkoleniowego. Drugi etap (testowanie) przepuszcza instancje testowe przez drzewa izolacyjne w celu uzyskania wyników anomalii dla każdej instancji. W drzewie losowym indukowanym przez dane, partycjonowanie instancji jest powtarzane rekurencyjnie, aż wszystkie instancje zostaną odizolowane. Ten losowy podział daje zauważalnie krótsze ścieżki dla anomalii, ponieważ mniej przypadków anomalii skutkuje mniejszą liczbą partycji (krótszymi ścieżkami w strukturze drzewa) oraz instancje z rozróżnialnymi wartościami atrybutów są bardziej prawdopodobne do oddzielenia we wczesnym partycjonowaniu. Wartości odstające są identyfikowane na podstawie tego jak szybko są izolowane w drzewach. Algorytm jest szczególnie przydatny w scenariuszach, gdzie anomalie są rzadkie i wykazują inne wzorce niż wartości normalne. W przeciwieństwie do innych metod, gdzie duży rozmiar próbki jest bardziej pożądany, metoda izolacji działa najlepiej na niewielkich rozmiarach próbek [176].

Wynik anomalii s instancji x jest zdefiniowany jako:

$$s(x,n) = 2^{-\frac{E(h(x))}{c(n)}}, \tag{4.21}$$

gdzie:

- E(h(x)) - średnia długość ścieżki punktu x jest mierzona przez jako liczba krawędzi, przez które przechodzi ścieżka od węzła głównego do węzła zewnętrznego dla kolekcji drzew izolacyjnych,
- c(n) - średnią długość ścieżki nieudanego wyszukiwania dla zbioru n instancji.

Zgodnie z równaniem 4.21, jeśli:

- $E(h(x)) \rightarrow c(n), s \rightarrow 0.5$;
- $E(h(x)) \rightarrow 0, s \rightarrow 1$;
- $E(h(x)) \rightarrow n, s \rightarrow 0$;

Używając wartości s jesteśmy w stanie wywnioskować:

- jeśli instancje zwracają s bardzo bliskie 1, to są to zdecydowanie anomalie;
- jeśli instancje mają s dużo mniejsze od 0.5, to są to zdecydowanie wartości poprawne;
- jeśli wszystkie instancje zwracają wartość 0,5, to cała wartość zbioru nie ma żadnej wyraźnej anomalii.

Algorytm ma niską, liniową złożoność czasową, z niskim zapotrzebowaniem na pamięć, co czyni go idealnym rozwiązaniem dla dużych zbiorów danych.

5. Algorytm INNE (ang. Isolation using Nearest Neighbours Ensamble)

Algorytm bazuje na alternatywnym mechanizmie izolacji, opartym na metodzie najbliższych sąsiadów, zaproponowanym przez Bandaragoda i in. [177]. INNE, podobnie jak iForest, dzieli przestrzeń danych w celu odizolowania każdej instancji od reszty instancji w podpróbce i określa wynik izolacji dla każdego regionu izolacji. W przeciwieństwie to IForest jednak każdy region jest hipersferą, której środek stanowi instancja podpróby, a granice są wyznaczone przez odległość najbliższego sąsiada (ang. Nearest Neighbour - NN) instancji w centrum. Podsumowując, do przeprowadzenia izolacji jest zastosowana metoda oparta na NN, a nie podziału równoległego do osi. Zalety tej metody są następujące:

- Każdy region izolacji dostosowuje się do lokalnej dystrybucji tworząc mniejsze hipersfery w gęstych obszarach i większe w rzadkich. Promień hipersfery stanowi miarę podatności na izolację.
- Wykorzystuje wszystkie dostępne atrybuty do podziału przestrzeni danych na regiony izolacyjne.
- Proponowany wynik izolacji jest miarą lokalną, która odnosi się do lokalnego sąsiedztwa, umożliwia więc wykrywanie lokalnych anomalii.
- Mechanizm izolacji NN radzi sobie z multimodalnymi zestawami danych lepiej niż osiowo — równoległy.

Algorytm INNE jest zaimplementowany jako 2-etapowy proces:

- faza trenowania - z t losowo wybranych podpróbek o rozmiarze ψ jest tworzonych t zbiorów hipersfer zdefiniowanych następująco:

$$\{\{B(c) : c \in S_i\} : i = 1, ...t\} \tag{4.22}$$

gdzie hipersfera B(c) wyśrodkowana w punkcie c o promieniu $\tau(c) = ||c - \eta_c||$ jest zdefiniowana następująco:

$$\{x : ||x - c|| < \tau(c)\} \tag{4.23}$$

- faza ewaluacji - każda instancja testowa jest oceniana względem t zestawów hipersfer oraz wyników izolacji uzyskanych na podstawie:

$$I(x) = \begin{cases} 1 - \frac{\tau(\eta_{cnn(x)})}{\tau(cnn(x)),} & \text{jeśli } x \in \bigcup_{c \in S} B(c) \\ 1, & \text{w pozostałych przypadkach} \end{cases} \tag{4.24}$$

gdzie:

$$cnn(x) = \arg min_{c \in S}\{\tau(c) \colon x \in B(c)\}. \tag{4.25}$$

które są uśredniane w celu uzyskania wskaźnika anomalii zgodnie z definicją:

$$\bar{I}(x) = \frac{1}{t} \sum_{i=1}^{t} I_i(x) \tag{4.26}$$

W odróżnieniu od innych algorytmów detekcji anomalii opartych na NN algorytm INNE ma liniową złożoność czasową uzyskaną dzięki wykorzystaniu wielu podpróbek, z których każda ma rozmiar znacznie mniejszy niż zestaw danych, dlatego doskonale sobie radzi z dużymi zestawami danych.

6. Autoenkoder

Alternatywnym algorytmem do wykrywania wartości odstających, umożliwiającym redukcję wielowymiarowości i wydobycie istotnych cech reprezentacji wejściowych, jest algorytm oparty o autoenkoder.

Działanie autoenkodera opiera się na zastosowaniu warstwy ukrytej (ang. hidden layer), której liczba węzłów jest znacznie mniejsza niż liczba węzłów warstwy wejściowej. Wyjście z warstwy ukrytej jest zredukowaną reprezentacją danych wejściowych. Architektura sieci neuronowej po obydwu stronach warstwy ukrytej jest zazwyczaj symetryczna. Autoenkoder składa się z kodera, kompresującego dane do mniej wymiarowej postaci oraz dekodera odpowiedzialnego za rekonstrukcję oryginalnych danych wejściowych. Pierwsza część sieci uczy się funkcji kodowania ϕ, a druga część sieci neuronowej uczy się funkcji dekodowania ψ. Dlatego $\phi(D)$ jest skompresowaną reprezentacją zbioru danych D, a zastosowanie funkcji dekodera ψ do $\phi(D)$ daje zrekonstruowane dane D', które mogą się różnić od D. Odstające wartości wejściowe są odporne na kompresję, więc będą wykazywały największą zmienność pomiędzy D i D'. Wartości bezwzględne macierzy rezydualnej $D - D'$ dostarczają informacji o wartościach odstających. Głównym celem zastosowania autoenkodera jest zmniejszenie błędu rekonstrukcji podczas uczenia się na zminimalizowanej, znaczącej reprezentacji wewnętrznej danych. Przy większej liczbie ukrytych jednostek można uzyskać niskowymiarową reprezentację dowolnego rozkładu danych, niezależnie od tego czy jest on liniowy, czy nie. Powodzenie tej strategii wynika z jej umiejętności modelowania skomplikowanych rozkładów nieliniowych, jednak należy pamiętać, że nadmierne zwiększanie liczby warstw ukrytych może prowadzić do nadmiernego dopasowania [178].

Zakłada się, że wszystkie punkty uczące są punktami normalnymi w ustawieniu jednoklasowym, predykcja z poniższego równania powinna wynosić 0:

$$\sum_{i=1}^{d} w_i \cdot x_i \approx 0 \tag{4.27}$$

Każda niezerowa wartość przewidziana przez jednoklasową sieć neuronową jest interpretowana jako wskazanie na wartości odstające, które nie pasują do modelu normalnych

danych. W związku z tym, dla konkretnej instancji X_i, w której sieć neuronowa przewiduje z_i, kwadrat błędu dla tego punktu, wynikający z założenia o jednoklasowym modelu, można wyrazić następująco:

$$J_i = z_i^2 = (\bar{W} \cdot \bar{X}_i)^2 \tag{4.28}$$

Aby ocenić dany punkt danych X_i, używa się wyuczonego modelu do obliczenia jego wyniku odstającego w następujący sposób:

$$S(\bar{X}_i = (\bar{W} \cdot \bar{X}_i)^2 \tag{4.29}$$

Wyniki punktów testowych są standaryzowane do zerowej średniej i wariancji jednostkowej. Proces ten jest powtarzany wielokrotnie na wielu losowych próbkach, a wyniki punktów są uśredniane.

7. Algorytm ALAD (ang. Adversarially Learned Anomaly Detection)

Algorytm bazuje na zaproponowanym przez Zenatii'ego i in. [179] podejścia do wykrywania anomalii opartego o architekturę GAN (ang. Generative Adversial Network) dla danych sekwencyjnych. Wykorzystuje również zaproponowany przez nich sposób pomiaru błędu rekonstrukcji. Algorytm jest dedykowany analizie danych szeregowych, które naturalnie posiadają zarówno krótkoterminowe, jak i długoterminowe zależności. Zaproponowane sieci typu kodującego i generującego opierają się na komórkach LSTM. Aby zapewnić spójność cyklu w trakcie treningu, wykorzystany został mechanizm rekonstrukcji dla treningu kodera i generatora, oraz druga sieć krytyczna do obsługi poprawnych dwukierunkowych mapowań. Dodatkowo zaproponowano, że punktowy błąd rekonstrukcji między oryginalnymi a zrekonstruowanymi punktami szeregu czasowego, który często jest stosowany do wykrywania anomalii, zostanie zastąpiony dwoma miarami podobieństwa, które będą oceniały lokalne podobieństwo między oryginalnymi i zrekonstruowanymi sekwencjami. Następnie połączone zostaną z danymi wyjściowymi sieci krytycznej, tworząc funkcję, która zapewnia solidne wyniki wykrywania anomalii dla szeregów czasowych. W naszym przypadku podobnie staramy się znaleźć odchylenia od normalnego stanu, znajdując wartości odstające w obu wynikach. W tym celu obliczany jest wektor skalarny błędu rekonstrukcji $RE(x)$ oraz wyjścia funkcji krytycznej $C_x(x)$:

$$\alpha(x) = RE(x) \odot C_x(x) \tag{4.30}$$

Pozwala to na zmniejszenie liczby wyników fałszywie pozytywnych, a także może prowadzić do zwiększenia liczby wyników prawdziwie pozytywnych.

Algorytmy nadzorowane

W przeciwieństwie do uczenia maszynowego nienadzorowanego algorytmy te wymagają etykietowania danych treningowych, czyli wcześniejszego przygotowanie danych i przypisanie im odpowiednich etykiet. Przyjęto konwencję, że wartości normalne przypisano do klasy o etykiecie wynoszącej 0, natomiast wartości odstające (anomalie) zostaną zakwalifikowane do klasy o etykiecie równej 1, Dla wszystkich wymienionych algorytmów nadzorowanych, realizujących zadania klasyfikacji badana była skuteczność w prawidłowym kwalifikowaniu zaburzeń w systemie testowym do klasy 1.

1. Algorytm XGB+

Algorytm bazuje na zaproponowanym przez Chen i Guestrin [180] algorytmie XGBBoost (XGB+) wykorzystującym metodę wzmacniania drzewa gradientowego. Zakłada on wykorzystanie algorytmu wzmacniania drzewa gradientowego. Zastosowany dodatkowy obiekt regulacyjny pozwala wygładzić ostateczne wyuczone wagi, aby nie doprowadzić do nadmiernego dopasowania. Model zespołu drzew zastosowany w rozwiązaniu zawiera funkcje jako parametry, w związku z czym nie może być optymalizowany przy użyciu tradycyjnych metod w przestrzeni Euklidesowej. Zamiast tego model jest trenowany w sposób addytywny.

Aby uniknąć nadmiernego dopasowania, zostały zastosowane dodatkowe dwie techniki: tzw. kurczenie (ang. shrinkage), które skaluje nowo dodane wagi o współczynnik η po każdym kroku wzmacniania drzewa [181] oraz podpróbkowanie (ang. subsampling) kolumn (cech). Kluczowym czynnikiem przyczyniającym się do sukcesu XGBoost jest jego zdolność do skalowania w różnych scenariuszach. System działa ponad dziesięć razy szybciej niż istniejące popularne rozwiązania nawet przy pojedynczym przetwarzaniu, oraz jest w stanie obsłużyć miliardy przykładów w środowiskach rozproszonych lub o ograniczonej pamięci. Skalowalność XGBoost wynika z kilku istotnych optymalizacji systemowych i algorytmicznych. Te innowacje obejmują wprowadzenie nowatorskiego algorytmu do równoległego uczenia drzewiastego, który jest skonstruowany specjalnie do obsługi rzadkich danych. Dodatkowo, zastosowanie procedury szkicu kwantylowego pozwala na uwzględnienie wag instancji w przybliżonym uczeniu drzewiastym. Wykorzystanie obliczeń równoległych i rozproszonych dzięki efektywnej strukturze blokowej przyspiesza proces uczenia, co umożliwia szybszą eksplorację modelu [180].

2. Algorytm CATB+

Algorytm opiera się o zaproponowaną przez Prokhorenkovą i in. [182] technikę łączącą zaawansowane uporządkowane wzmacnianie oraz innowacyjne podejście do przetwarzania cech kategorialnych. Obydwie techniki zostały opracowane w celu zwalczania przesunięć predykcji spowodowanych specjalnym rodzajem wycieku docelowego obecnego we wszystkich dotychczasowych implementacjach algorytmów wzmacniania gradientowego.

CatBoost (CATB+) to implementacja wzmacniania gradientowego wykorzystująca binarne drzewa decyzyjne jako podstawowe predyktory. Każdy końcowy region (liść drzewa) jest przypisany do wartości, która jest szacunkową odpowiedzią w regionie dla zadania regresji lub przewidywaną etykietą dla problemu klasyfikacji.

Cecha kategorialna to cecha posiadająca dyskretny zbiór wartości, nazywanych kategoriami, które nie mogą być porównywane między sobą. Jedną z powszechnie stosowanych technik radzenia sobie z cechami kategorialnymi jest technika *one-hot encoding*, polegająca zdefiniowaniu każdej cechy przy pomocy jej binarnego odpowiednika. Jednakże, w przypadku cech o wysokiej kardynalności, takich jak na przykład cecha „identyfikator użytkownika", taka technika prowadzi do tworzenia niezwykle dużej liczby nowych cech, co jest niepraktyczne. Popularną metodą jest grupowanie kategorii według docelowych statystyk (ang. target statistics - TS) przy pomocy szacowanej wartości docelowej dla każdej kategorii. Należy zwrócić uwagę, że liczba możliwych kombinacji rośnie wykładniczo wraz z liczbą cech kategorialnych w zbiorze danych. CatBoost konstruuje te kombinacje w sposób zachłanny. Innymi słowy, dla każdego podziału drzewa, CatBoost łączy (konkatenuje) wszystkie cechy kategorialne (oraz ich kombinacje), które zostały już użyte w poprzednich podziałach w bieżącym drzewie, ze wszystkimi cechami kategorialnymi w zbiorze danych. Kombinacje te są przetwarzane w czasie rzeczywistym.

3. Algorytm LGB

Algorytm bazuje na innowacyjnym podejściu zwanym LightGBM (LGB), opartym na dwóch nowych technikach, jakimi są: próbkowanie jednostronne oparte na gradiencie (ang. Gradient — based One — Side Sampling — GOSS) i ekskluzywne (wyłączające się) wiązanie cech (ang. Exclusive Feature Bundling — EFB). GOSS opiera się na wykluczeniu znacznej części instancji z małymi gradientami i wykorzystaniu pozostałych do oszacowania przyrostu informacji, co pozwala na przetwarzanie informacji o znacznie mniejszym rozmiarze. Takie podejście może prowadzić do dokładniejszego oszacowania wzmocnienia niż równomierne losowe próbkowanie, przy zachowaniu tej samej docelowej częstotliwości próbkowania, zwłaszcza gdy wartość wzmocnienia informacyjnego ma duży zakres. Natomiast EFB łączy wzajemnie wykluczające się cechy, aby zmniejszyć ich liczbę. Realizowane jest to przy pomocy algorytmu zachłannego osiągającego dobry współczynnik aproksymacji, a więc pozwala na skuteczne zmniejszenie liczby cech bez pogorszenia dokładności. W pierwszym etapie cechy są sortowane w porządku malejącym według liczby ich wartości niezerowych. Na koniec sprawdzana jest każda cecha na uporządkowanej liście i jest przypisywana do istniejącego pakietu (z małym, pomijalnym konfliktem), albo tworzony jest nowy pakiet. Kluczowym aspektem tego etapu jest zapewnienie, że wartości oryginalnych cech można zidentyfikować na podstawie pakietów [183].

4. Algorytm SVM (ang. Support Vector Machine)

Algorytm bazuje na klasycznym algorytmie SVM, którego pierwsza koncepcja została przedstawiona w 1963 roku przez Vapnika i Lernera [184]. W 1992 roku został zaprezentowany nieliniowy klasyfikator SVM [185]. Natomiast w 1995 roku pojawił się ulepszony klasyfikator SVM, uwzględniający możliwość występowania zaszumionych danych [186]. Sieci neuronowe typu SVM wykorzystują specyficzną metodologię uczenia, która dąży do maksymalizacji marginesu separacji między danymi należącymi do dwóch różnych klas. Podstawowa koncepcja działania sieci typu SVM polega na traktowaniu zbiorów danych wejściowych jako wektorów, które reprezentują współrzędne punktów w przestrzeni wielowymiarowej [187]. Transformacja jest realizowana w taki sposób, że w nowej przestrzeni obiekty stają się separowalne za pomocą hiperpłaszczyzny $f(x) = 0$. W przypadku danych liniowych hiperpłaszczyzna może być wyznaczona z następującego równania [188]:

$$f(x) = W^T x + b = \sum_{i=1}^{N} W_i x_i + b \tag{4.31}$$

gdzie N to liczba próbek, W jest $N-$wymiarowym wektorem a b skalarem stosowanym do zdefiniowania pozycji hiperpłaszczyzny. Taka separacja jest zazwyczaj niemożliwa w oryginalnej przestrzeni danych. W przypadku klasyfikacji nieliniowej kluczowym składnikiem transformacji jest wybór funkcji jądra (ang. kernel function), która odpowiada za odwzorowanie punktów do nowej przestrzeni [189]. Wybór jądra ma kluczowe znaczenie dla jakości modelu [190]. Hiperpłaszczyzna rozgranicza dwie klasy punktów z pewnym marginesem bezpieczeństwa, wewnątrz którego nie leży żaden punkt z separowanych zbiorów. Najbliższe hiperpłaszczyźnie punkty leżą na krańcach tego marginesu, pozostałe zaś dalej, poza nim [191].

5. Algorytm MLP (ang. Multilayer Perceptron)

Algorytm bazuje na perceptronie wielowarstwowym (MLP), który jest jednym z najczęściej stosowanych modeli w dziedzinie sieci neuronowych do rozwiązywania zadań predykcyjnych. W przeciwieństwie do algorytmu SVM algorytm MLP wykorzystuje sigmoidalną

funkcję aktywacji [192]. Perceptron wielowarstwowy (MLP) to klasa nieliniowych modeli statystycznych, które składają się z wielu warstw węzłów w grafie skierowanym, gdzie każda warstwa jest w pełni połączona z następną. W strukturze MLP wyróżniamy trzy różne typy warstw: warstwę wejściową, ukrytą i wyjściową. Z wyjątkiem węzłów wejściowych, każdy węzeł jest neuronem z nieliniową funkcją aktywacji [193]. Dla danych wejściowych $x_i(i = 1, 2, ..., N)$ dane wyjściowe modelu neuronowego y można przedstawić za pomocą równania:

$$y = f(W^T x) = f(\sum_{i=1}^{N} W_i x_i + b) \tag{4.32}$$

gdzie f to funkcja aktywacji, W to wagi modelu, a b to wektor przesunięcia (ang. bias) [188].

MLP wykorzystuje algorytm wstecznej propagacji błędów do dostosowania wag połączeń między neuronami, minimalizując błąd między predykcją a rzeczywistymi danymi. Wagi są aktualizowane iteracyjnie na podstawie gradientu funkcji kosztu. Istotne jest odpowiednie dostrojenie hiperparametrów takich jak liczba warstw ukrytych, liczba neuronów w każdej warstwie, współczynnik uczenia i funkcja kosztu.

Algorytm MLP wymaga danych znormalizowanych. Wynikiem klasyfikacji binarnej jest wartość 0 lub 1, odpowiadająca przynależności do klasy docelowej.

4.3.3 Miary skuteczności algorytmów

W zależności od rodzaju problemu, danych wejściowych oraz oczekiwanego zachowania modelu do oceny jakości działania algorytmów sztucznej inteligencji służą różne miary skuteczności algorytmów [194]:

- Średnia kwadratowa błędu (ang. Root Mean Squared Error) jest jedną z najpopularniejszych miar do oceny dokładności modelu regresyjnego. RMSE mierzy średnią wielkość błędów prognozowanych przez model, przy czym błędy te są mierzone w tych samych jednostkach co dane wyjściowe.

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \widehat{y}_i)^2} \tag{4.33}$$

gdzie:

n to liczba obserwacji,

y_i to rzeczywiste wartości,

$\widehat{y}_i$ to wartości prognozowane przez model.

- Macierz pomyłek (ang. confusion matrix) - umożliwia porównanie decyzji klasyfikatora z rzeczywistym przypisaniem obiektów do klas. Dzięki temu łatwo zidentyfikować zjawisko nadmiernego dopasowania. Macierz pomyłek przedstawia liczbę przypadków dla każdej kombinacji rzeczywistej klasy i przewidywanej klasy z wyników klasyfikacji. Każda przewidywana etykieta binarna ma cztery możliwości:
  - True Positive (TP): Poprawnie sklasyfikowane pozytywne przypadki.
  - False Negative (FN): Błędnie sklasyfikowane negatywne przypadki.
  - False Positive (FP): Błędnie sklasyfikowane pozytywne przypadki.
  - True Negative (TN): Poprawnie sklasyfikowane negatywne przypadki.

- Precyzja (ang. Precision - PRE) - określa odsetek prawdziwie pozytywnych wyników wśród wszystkich pozytywnych predykcji. Szczególnie przydatny w przypadku kosztów związanych z fałszywą predykcją negatywną.

$$PRE = \frac{TP}{TP + FP} \tag{4.34}$$

- Roc_Auc_Score (ang. Receiver Operating Characteristic Area Under Curve – AUC) - pokazuje on jak dobrze model może generować względne wyniki w celu rozróżnienia między pozytywnymi i negatywnymi przypadkami we wszystkich prograch klasyfikacji. Jego wartość mieści się w zakresie od 0 do 1, gdzie wartość 0,5 oznacza losowe zgadywanie, a 1 doskonałe działanie algorytmu.
- Współczynnik korelacji Matthewsa (ang. Matthews Correlation Coefficient - MCC) - jest to miara wydajności stosowana w zadaniach klasyfikacji binarnej, szczególnie w sytuacjach, gdzie istnieje nierównowaga między klasami. Wartości MCC mieszczą się w przedziale od -1 do 1. Wartość 1 oznacza idealne dopasowanie, wartość 0 oznacza brak korelacji, a wartość -1 oznacza idealne przeciwdziałanie.

$$MCC = \frac{TN \cdot TP - FN \cdot FP}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}} \tag{4.35}$$

- Zrównoważona dokładność (ang. Balanced Accurancy - BA) - jest używane w sytuacjach, gdzie dokładność (ang. accuracy) mogłaby być myląca ze względu na nierównomierny rozkład klas. Ta miara właściwsza, gdy mamy do czynienia z niezbalansowanymi danymi, czyli tam, gdzie jedna klasa występuje znacznie częściej niż druga. BA zwraca wartość z zakresu od 0 do 1, gdzie 1 oznacza idealnie zrównoważoną dokładność, a 0 oznacza brak zrównoważenia, natomiast 0,5 oznacza, że klasyfikator działa jak klasyfikator losowy.

$$BA = 0.5 \cdot (\frac{TP}{TP + FN} + \frac{TN}{TN + FP}) \tag{4.36}$$

5. Badania weryfikacyjne

Rozdział poświęcony jest badaniom weryfikacyjnym, mającym na celu wykazanie skuteczności opracowanej metodyki testowania systemów wbudowanych z wykorzystaniem algorytmów rozszerzonej inteligencji. Rozdział ten zawiera szczegółowy plan badań weryfikacyjnych, opis przeprowadzonych eksperymentów oraz wyniki uzyskane w ich trakcie. W początkowej części przedstawiono plan badań, który obejmuje analizę problemu migotania rezultatów testów systemów wbudowanych oraz identyfikację potencjalnych przyczyn tego zjawiska. Następnie zawarto opis eksperymentów, w tym charakterystykę badanych układów wbudowanych, opis stanowiska testowego oraz zastosowanego modelu symulacyjnego pojazdu, a także zastosowane metody i narzędzia eksploracji danych. W dalszej części rozdziału przedstawiono wyniki przeprowadzonych badań, które obejmują analizę częstości występowania migotania testów, ocenę przydatności modelu pojazdu do generowania aberracji oraz weryfikację skuteczności zaimplementowanych algorytmów rozszerzonej inteligencji w identyfikacji anomalii. Wyniki te zostały uzyskane poprzez wielokrotne wykonanie przypadków testowych i analizę ich rezultatów z wykorzystaniem różnych algorytmów. Podsumowując, rozdział ten stanowi kluczowy element pracy, prezentując kompleksowe podejście do testowania systemów wbudowanych, które może być zastosowane w rzeczywistych projektach inżynieryjnych, w celu poprawy jakości i niezawodności testowanych systemów.

5.1 Plan badań weryfikacyjnych

W celu udowodnienia skuteczności zaproponowanej metodyki przeprowadzono badania weryfikacyjne w następujących etapach.

Etap I — Analiza występowania testów migoczących

Badanie ma na celu potwierdzenie istoty problemu, jakim są rezultaty migoczące występujące w procesie testowania systemów wbudowanych. Przeprowadzone zostanie badanie co najmniej 200 przypadków testowych. Każdy przypadek testowy zostanie wykonany dziesięciokrotnie, spośród wykonanych testów wyodrębnione zostaną testy o zmiennych rezultatach, analizie poddane zostaną ich wyniki oraz wyznaczona zostanie częstość migotania testów.

Etap II — Analiza możliwych przyczyn migotania testów

Spośród uzyskanych rezultatów testów migoczących wybranych zostanie co najmniej 20 przypadków testowych. Zostaną one poddane analizie ze względu na możliwe przyczyny migotania testów. Opracowany zostanie katalog anomalii wraz z częstotliwością ich występowania oraz opisem zależności między nimi.

Etap III — Analiza wpływu zastosowanego modelu pojazdu na występowanie zjawiska migotania rezultatów testów

Zostanie opracowana koncepcja stanowiska testowego, w którym zastosowany będzie model pojazdu. Dzięki temu możliwe będzie zastąpienie uproszczeń związanych z sygnałami komunikacyjnymi oraz wartościami fizycznymi istotnymi z perspektywy badanego systemu wbudowanego,

informacjami zmiennymi w czasie, zależnymi od mechaniki pojazdu oraz dynamiki cyklu jazdy. Stanowisko to dodatkowo będzie umożliwiało generowanie minimum trzech aberracji w badanych systemach. Badaniu będzie podlegał wpływ aberracji generowanych na stanowisku testowym na częstość występowania fałszywie negatywnych wyników testów układów wbudowanych. Weryfikacji poddane zostaną przypadki testowe z poprzedniego eksperymentu.

Etap IV — Sprawdzenie poprawnej identyfikacji fałszywie negatywnych wyników testów przez zaimplementowane algorytmy sztucznej inteligencji

Podczas wykonywania testów w czasie etapów I i II zostaną zarejestrowane dane o wartościach fizycznych zmieniających się w środowisku testowym. Zgromadzone w ten sposób informacje zostaną wykorzystane w procesie treningu, a następnie weryfikacji efektywności działania wybranych i zaimplementowanych algorytmów sztucznej inteligencji. Algorytmy, których skuteczność zostanie potwierdzona w tym etapie, zostaną wybrane do realizacji kolejnych etapów badawczych jako element proponowanych algorytmów rozszerzonej inteligencji.

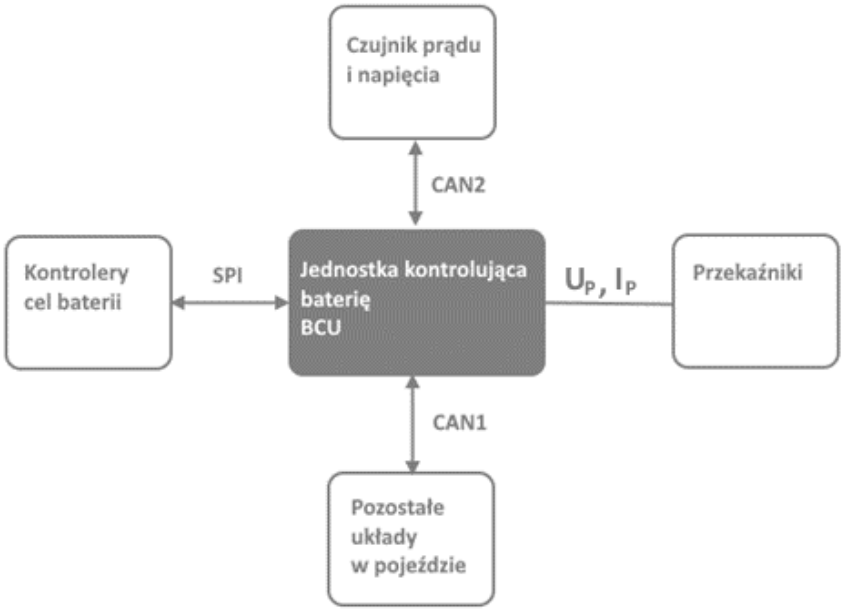
Etap V — Końcowa weryfikacja zaproponowanej metodyki

Prace badawcze w tym etapie mają na celu zbadanie, jaki wpływ na rezultaty wyników testów systemów wbudowanych ma zastosowanie zaproponowanej metodyki podczas specyfikacji i implementacji przypadków testowych. Poddano analizie wpływ zastosowania modelu pojazdu na jakość uzyskanych rezultatów testów systemów wbudowanych wykonanych na stanowisku badawczym dwukrotnie — z modelem pojazdu zintegrowanym ze środowiskiem oraz bez modelu. Podczas wykonywania tych przypadków testowych zostaną zarejestrowane informacje o wartościach fizycznych zmieniających się w środowisku testowym. W przypadku wystąpienia rezultatu negatywnego testu, dane z dzienników zostaną automatycznie przygotowane i wprowadzone na wejście algorytmów rozszerzonej inteligencji. Analizie zostaną poddane rezultaty wykonania zaimplementowanych algorytmów, czyli poprawność wskazania przez nie anomalii występującej w środowisku testowym, zaburzającej wynik prowadzonego testu.

5.2 Obiekt badań

5.2.1 System wbudowany wybrany do badań – budowa i zasada działania

W czasie prac badawczych wykorzystywany był system wbudowany pokazany na Rys. 5.1.



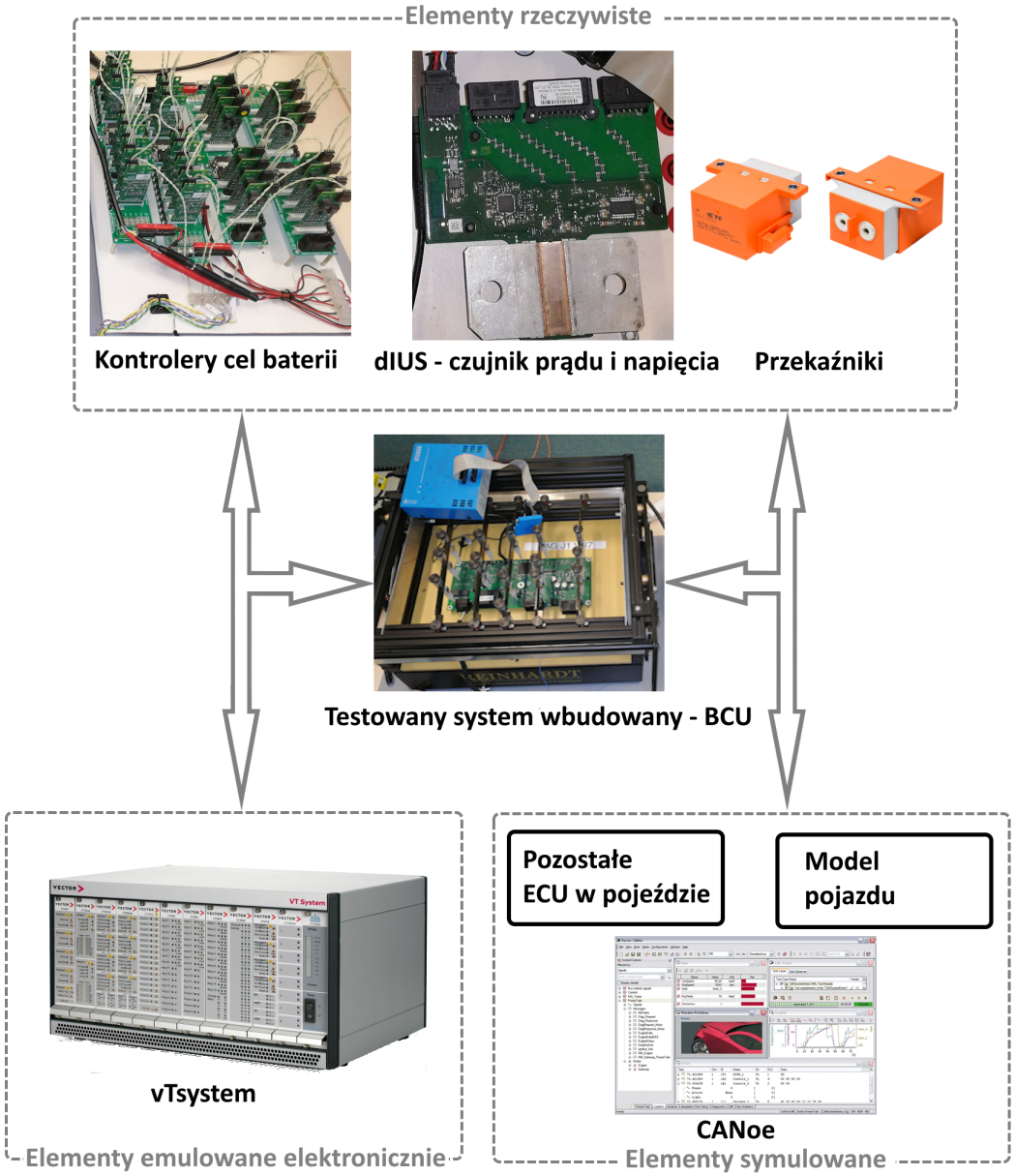
Rys. 5.1. Badany system jako czarna skrzynka [83]

W pojeździe elektrycznym pełni on funkcję jednostki zarządzającej (ang. Battery Control Unit — BCU) stanem naładowania, i funkcjonowaniem akumulatora wysokonapięciowego. BCU pełni kluczową rolę w zapewnieniu bezpieczeństwa, wydajności i trwałości systemu baterii w pojazdach elektrycznych. Dzięki zaawansowanym funkcjom monitorowania, równoważenia i ochrony, BCU pomaga zarządzać złożonymi systemami zasilania, umożliwiając pojazdom elektrycznym bezpieczne i efektywne działanie. BCU stale monitoruje napięcie, natężenie prądu i temperaturę każdej celi w baterii, analizując te dane w czasie rzeczywistym. Na podstawie tych pomiarów ocenia stan naładowania (SOC), stan zdrowia (SOH) oraz przewidywaną żywotność (SOL) baterii. Komunikuje się on z kontrolerami baterii poprzez interfejs szeregowy (ang. Serial Peripheral Interface — SPI), za pośrednictwem którego przekazywane są informacje niezbędne do skutecznego wdrożenia algorytmów sterowania. Ponadto system wbudowany uzyskuje dane z innego systemu wbudowanego pełniącego funkcję sensora mierzącego natężenie prądu i napięcie baterii pojazdu. Dane te przesyłane są magistralą szeregową CAN (ang. Controller Area Network), oznaczoną na schemacie jako CAN2. Dzięki temu BCU kontroluje prąd ładowania i rozładowania, aby zapewnić, że mieści się on w bezpiecznych granicach, co zapobiega uszkodzeniu baterii i optymalizuje jej wydajność. Dodatkowo, aby zapewnić równomierne rozładowywanie i ładowanie poszczególnych cel, BCU aktywnie zarządza funkcjami balansowania, wyrównując napięcia między celami. Proces ten minimalizuje ryzyko przeładowania lub głębokiego rozładowania poszczególnych cel. W przypadku wykrycia anomalii, takich jak nadmierna temperatura, zbyt wysokie napięcie lub zwarcie, BCU automatycznie odłącza baterię od reszty układu, chroniąc ją przed uszkodzeniem i zapobiegając zagrożeniom bezpieczeństwa. Wspomniane na schemacie przekaźniki spełniają funkcję przełączników: w zależności od stanu pracy baterii – czy jest ona rozładowywana (tryb podstawowy), czy też ładowana napięciem zmiennym lub stałym - odpowiedni obwód jest łączony z baterią. W sytuacji wykrycia

incydentu klasy awaryjnej (jak np. nadmiernie wysokie napięcie na komórce baterii), sterownik, oprócz zawiadomienia innych układów pojazdu o zaistniałym błędzie, aktywuje niezwłoczne otwarcie przekaźników i przechodzi w stan bezpieczny. Poprzez magistralę oznaczoną na schemacie jako CAN1, sterownik utrzymuje komunikację z pozostałymi modułami ECU w strukturze pojazdu, przede wszystkim z jednostką sterującą pojazdu (ang. Vehicle Control Unit - VCU) oraz systemami ładowania.

5.2.2 Stanowisko badawcze

Stanowisko badawcze zbudowano zgodnie z zaleceniami normy ISO 26262 dotyczącymi weryfikacji wymagań o poziomie bezpieczeństwa funkcjonalnego określonym na ASIL A lub wyższym. Zastosowano środowisko HIL pokazane na Rys. 5.2, co jest dopuszczalne ze względu na koszty oraz trudności z symulacją błędów w czasie testowania systemów wbudowanych w rzeczywistym pojeździe.



Rys. 5.2. Stanowisko badawcze

W ramach środowiska badawczego wykorzystywane są rzeczywiste sensory (dIUS) oraz elementy wykonawcze (przekaźniki), które wiążą się bezpośrednio z funkcjonalnością testowanego systemu wbudowanego. Oprócz tego na stanowisku badawczym zastosowano rzeczywiste kontrolery cel baterii. Jednak samą baterię wysokonapięciową, a więc jej parametry fizyczne takie jak napięcie na celach, temperatura poszczególnych cel, natężenie prądu, rezystancja izolacji podawane na wejścia tych kontrolerów oraz systemu dIUS, były emulowane elektrycznie w systemie symulacyjnym firmy Vector Informatik - oznaczonym na schemacie jako vTsystem. Jednocześnie, w oprogramowaniu CANoe, poprzez pętlę sprzężenia zwrotnego symulowana jest pozostała część pojazdu, a więc komunikacja poprzez magistralę CAN z innymi ECU w pojeździe (np. status BMU, sygnały diagnostyczne, komendy sterujące przekaźnikami). W wyniku zaimplementowanych algorytmów np. do sterowania przekaźnikami na podstawie informacji odbieranych magistralą CAN możliwa jest realizacja różnych scenariuszy testowych np. szybkiego ładowania pojazdu. Poważnym ograniczeniem prezentowanego rozwiązania jest stosowanie pewnych form uproszczeń. Należy podkreślić, że sygnały (SOC, prędkość pojazdu) oraz parametry fizyczne (napięcie, natężenie prądu baterii, temperatura baterii) pochodzące z pojazdu mechanicznego w toku symulacji są przypisywane do stałej, określonej wartości (np. SOC = 85%, prędkość pojazdu = 50km/h, napięcie baterii = 775V). W toku prac badawczych zaproponowano zastosowanie w tym miejscu zamiast stałych wartości sygnałów pochodzących z modelu pojazdu mechanicznego przygotowanego w środowisku Simulink (rozdział 4.1). Dzięki tej modyfikacji warunki testowe uwzględniają zależności czasowe pomiędzy zmiennymi procesowymi, przez co są bardziej zbliżone do rzeczywistości, a równocześnie możliwe jest zbadanie wpływu zaburzeń występujących w środowisku na wyniki testów dzięki możliwości generowania anomalii w środowisku testowym badanego systemu.

5.3 Opisy oraz wyniki przeprowadzonych badań

Badanie weryfikacyjne rozpoczęto od analizy zjawiska migotania rezultatów testów oraz możliwych przyczyn występowania tego zjawiska. Etap I i II badań posłużył jako podstawa do dalszych prac nad metodyką tworzenia zautomatyzowanych testów systemów wbudowanych w pojazdach samochodowych umożliwiającej weryfikację poprawności wyników z wykorzystaniem algorytmów rozszerzonej inteligencji. Kolejne etapy III i IV stanowiły fazę pośrednią, której celem była weryfikacja poszczególnych innowacyjnych rozwiązań, które zaproponowano w metodyce, takich jak zastosowanie modelu pojazdu w środowisku testowym czy identyfikacja anomalii pojawiających się w środowisku w procesie testowania, wpływających na rezultaty testów przy pomocy algorytmów rozszerzonej inteligencji. Były one punktem odniesienia do kolejnych, bardziej złożonych eksperymentów, które były zwieńczeniem prowadzonych prac badawczych — etapu V, które miał na celu walidację zaproponowanej metodyki. Wyniki uzyskanych badań własnych były szeroko publikowane w czasopismach [172] oraz na konferencjach naukowych [195], [83], [160].

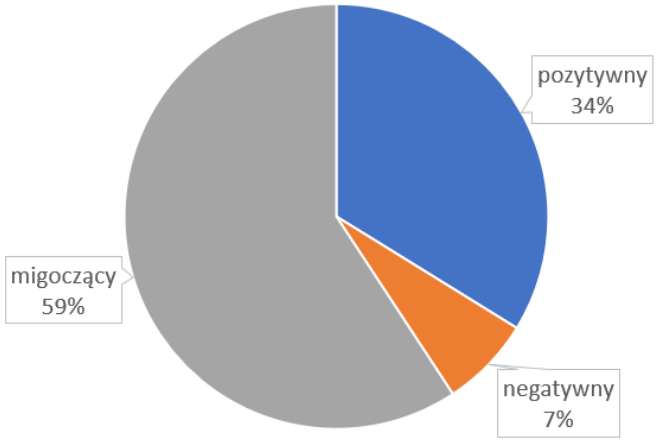
5.3.1 Etap I — Analiza występowania testów migoczących

Etap ten został podzielony na dwie części. W pierwszym eksperymencie wykonano 216 przypadków testowych. Należy podkreślić, że był to kompletny zestaw testów pokrywających wszystkie funkcjonalności testowanego systemu wbudowanego takie jak interfejsy komunikacyjne, diagnostykę, obsługę błędów, a przede wszystkim funkcje związane z obsługą elementów zewnętrznych pokazanych na rysunku 5.2.

Eksperyment polegał na jednokrotnym wykonaniu wszystkich testów i analizie procentowej uzyskanych wyników. Test uznaje się za zakończony rezultatem pozytywnym, jeśli wszystkie asercje zakończyły się wynikiem pozytywnym. Natomiast test jest uznawany za zakończony rezultatem negatywnym, jeżeli przynajmniej jedna asercja w trakcie jego wykonywania dała wynik negatywny. W wyniku przeprowadzonego eksperymentu otrzymano 167 rezultatów pozytywnych oraz 49 negatywnych, czyli 22,69% negatywnych rezultatów. Czas wykonywania tego eksperymentu to 233 minuty.

Zgodnie z metodyką testowania (Rys. 4.1) w przypadku rezultatów negatywnych konieczna jest analiza otrzymanych wyników i oszacowanie przyczyny niepowodzenia. Należy opisać defekt badanego układu lub wykazać, że przyczyną jest czynnik środowiskowy. Taka analiza trwa zazwyczaj kilka do kilkudziesięciu minut w zależności od skomplikowania przypadku testowego. Zakładając czas analizy 5 minut – w przypadku tej sesji testowej analiza negatywnych wyników zajęłaby co najmniej 245 minut.

Drugi eksperyment tego etapu miał na celu określenie, jak poważnym zjawiskiem jest migotanie rezultatów testów. Wybrane wcześniej 216 przypadków testowych zostało wykonanych dziesięciokrotnie w losowej kolejności, w celu uniezależnienia otrzymanych wyników bieżących od poprzednich rezultatów. Otrzymano 128 wyników migoczących (czyli 59%), 73 wyniki stabilnie pozytywne (co stanowi 34%) oraz 15 stabilnie negatywnych (7%), co przedstawiono na Rys. 5.3.



Rys. 5.3. Rezultaty uzyskane w 10 próbach

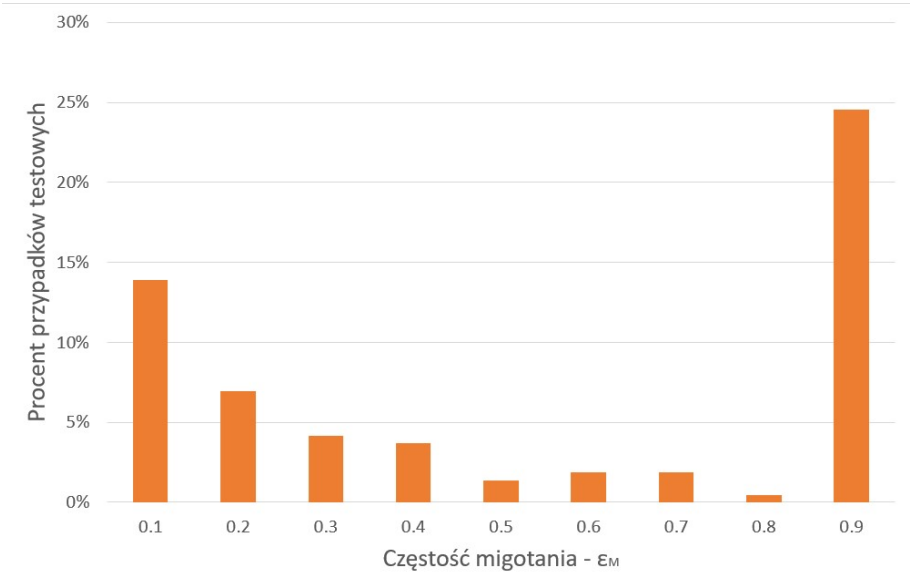
W tabeli pokazano rezultaty dla przykładowych 20 przypadków testowych uzyskane w etapie I. Liczbę rezultatów pozytywnych L_p zestawiono z liczbą rezultatów negatywnych L_{neg} oraz wyznaczono częstość migotania rezultatów ε_M.

Tab. 5.1. Przykładowe rezultaty uzyskane w I etapie badań

	1 wykonanie		**10 wykonań**		
Przypadek testowy	L_p	L_{neg}	L_p	L_{neg}	ε_M
Test1	1	0	6	4	0.4
Test2	1	0	7	3	0.3
Test3	1	0	6	4	0.4
Test4	0	1	2	8	0.8
Test5	0	1	5	5	0.5
Test6	1	0	5	5	0.5
Test7	1	0	10	0	0
Test8	0	1	2	8	0.8
Test9	1	0	9	1	0.1
Test10	1	0	3	7	0.7
Test11	0	1	8	2	0.2
Test12	1	0	8	2	0.2
Test13	1	0	5	5	0.5
Test14	1	0	9	1	0.1
Test15	0	1	4	6	0.6
Test16	1	0	3	7	0.7
Test17	0	1	9	1	0.1
Test18	0	1	6	4	0.4
Test19	1	0	4	6	0.6
Test20	0	1	7	3	0.3

Oznacza to, że nie wszystkie otrzymane wcześniej wyniki pozytywne są wynikami pewnymi. Rozkład częstości migotania uzyskany w czasie 10 prób został przedstawiony na Rys. 5.4. Przy-

jęto, że częstość migotania testu oznacza liczbę negatywnych rezultatów spośród wszystkich wyników dla danego testu.



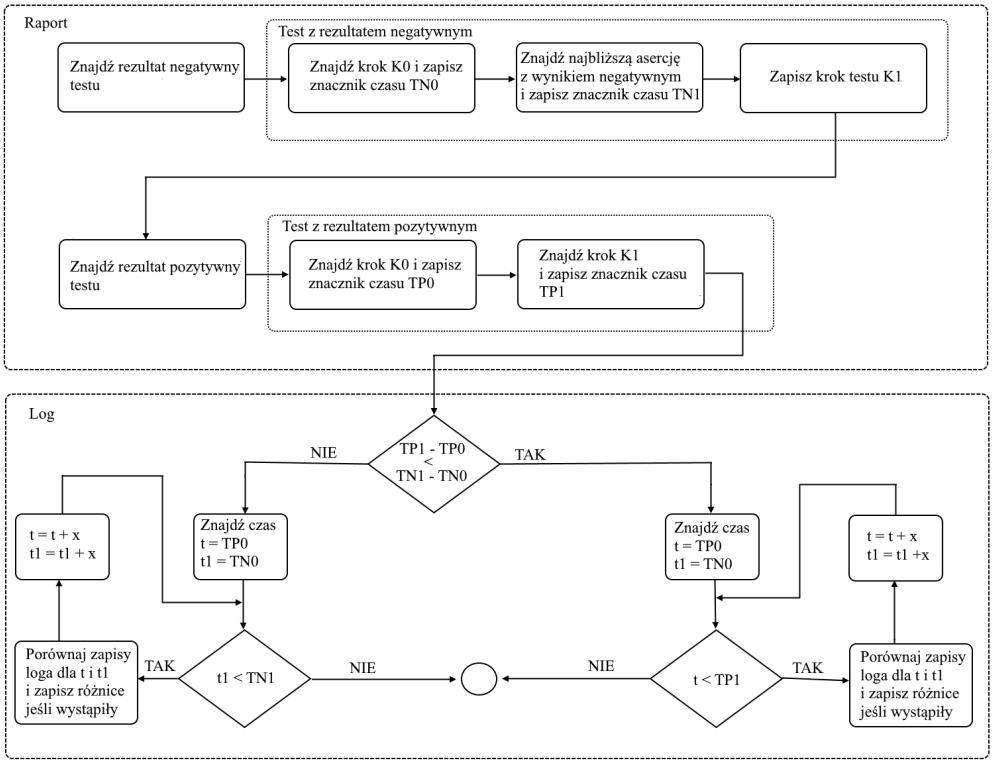
Rys. 5.4. Procentowy rozkład przypadków testowych według częstości migotania rezultatów

Uzyskane rezultaty pokazują 30 przypadków testowych (14%) uzyskało 1 wynik negatywny w 10 próbach, a aż 53 przypadki testowe (25%) jeden wynik pozytywny w 10 próbach. Natomiast 2 rezultaty przeciwne (odpowiednio negatywne lub pozytywne) uzyskało 15 (7%) i 1 (0.4%) przypadek testowy. Podsumowując 46.4% rezultatów migoczących charakteryzuje się tym, że wynik przeciwny pojawił się maksymalnie 2 razy w dziesięciu próbach, aż w przypadku 53.6% rezultatów migoczących zmienność rezultatów jest wyższa. Wartości te pokazują, jak bardzo niekorzystnym zjawiskiem jest migotanie wyników.

Prawidłowe podejście do testów migoczących powinno polegać przede wszystkim na analizie przyczyny ich migotania, a nie kolejnym wykonywaniu testów, aż do uzyskania wyniku pozytywnego. Jednak jest to proces czasochłonny i wymagający eksperckiej wiedzy z zakresu nie tylko testowanego projektu, ale także mechatroniki. Uzyskane wyniki i oszacowany czas, jaki jest potrzebny na eksperckie analizy, potwierdziły konieczność opracowania algorytmów rozszerzonej inteligencji, umożliwiających automatyzację analizy wyników testów. Pozwoli to na optymalizację tego procesu i poprawi wiarygodność procesu testowania systemów wbudowanych stosowanych w pojazdach samochodowych.

5.3.2 Etap II — Analiza możliwych przyczyn migotania testów

Celem prowadzonego etapu badawczego była identyfikacja i kategoryzacja anomalii występujących podczas wykonywania testów tak, aby można było zaproponować metodykę tworzenia przypadków testowych umożliwiających zastosowanie algorytmów rozszerzonej inteligencji do automatycznej weryfikacji negatywnych rezultatów testów oraz identyfikacji testów potencjalnie migoczących. Na potrzeby tego etapu badań spośród 128 testów o wynikach sygnalizowanych migotaniem wybrano losowo 25. Zaproponowano algorytm przedstawiony na Rys. 5.5, w którym pierwszy krok to analiza raportów z wykonania testów, a następnie interpretacja określonych chwil czasowych z plików dzienników i próba wnioskowania na temat kategorii anomalii występujących w środowisku testowym na podstawie zapisów informacji o zmianach wielkości fizycznych i sygnałów przesyłanych na liniach komunikacyjnych.



Rys. 5.5. Algorytm analizy wyników testów pod kątem identyfikacji anomalii wywołujących migotanie rezultatów testów [83]

W pierwszym etapie analiza obejmuje dwa raporty z wykonania tego samego testu: jeden z wynikiem pozytywnym, drugi z wynikiem negatywnym. W raporcie z wynikiem negatywnym pierwszy krok wykonywanego testu jest oznaczony jako K0, a odpowiadający mu znacznik czasowy jako TN0. Pierwsza asercja dająca rezultat negatywny jest oznaczona jako krok K1, a odpowiadający jej znacznik czasowy jako TN1. Następnie w teście z wynikiem pozytywnym oznaczony został znacznik czasowy TP0 odpowiadający krokowi K0 wykonania testu oraz znacznik TP1 odpowiadający krokowi K1. Dane te są podstawą do analizy plików dziennika z wykonania testów. Porównaniu podlegają wszystkie wartości sygnałów i wielkości fizycznych zdefiniowanych dla danego testu jako zmienne zależne i niezależne (zgodnie z opisem w rozdziale 4.1) od znaczników czasowych TN0 i TP0 do kroku K1. Należy zwrócić uwagę na fakt, że czas wykonania testu z wynikiem negatywnym i tego z wynikiem pozytywnym będzie różny, więc analiza powinna obejmować wszystkie wartości aż do osiągnięcia kroku K1 w teście, który zakończył się wcześniej. Wartość iteratora x zaznaczonego na schemacie zależy od czasu trwania cyklu, z jakim zmieniają się wartości zapisywane w logu.

Wybrane zostały następujące atrybuty, pojawiające się w normie IEEE Standard Classification for Software Anomalies [49], do klasyfikacji zidentyfikowanych anomalii:

a) identyfikator (ang. ID)

b) waga (ang. severity)

c) prawdopodobieństwo (ang. probability)

d) tryb (ang. mode)

e) typ (ang. type)

W tabeli 5.2 przedstawiono wartości poszczególnych atrybutów zdefiniowane dla bieżącego etapu badań.

Tab. 5.2. Wartości atrybutów stosowane do kategoryzacji [83]

Atrybut	Wartość	Opis
LP	liczba całkowita	kolejny numer testu
Identyfikator	tekst	krótki, zwięzły opis dotyczący przeprowadzanego testu, np. zawierający nazwę modułu i funkcjonalności, którą reprezentuje
waga	blokująca	przeprowadzenie testu jest niemożliwe
	krytyczna	podstawowe operacje są zakłócone, kolejne testy będą miały negatywne rezultaty, powoduje błędy w środowisku testowym
	główna	jest przyczyną negatywnego wyniku testowanej funkcji, testy mogą być kontynuowane
	drobna	nie ma wpływu na testowaną właściwość systemu, negatywny wynik testu wynika z zakłóconych nieistotnych operacji
	nieznacząca	nie ma wpływu na testowany system, ani na środowisko testowe
prawdopodobieństwo	wysokie	prawdopodobieństwo wystąpienia większe niż 70%
	średnie	prawdopodobieństwo wystąpienia pomiędzy 40% a 70%
	niskie	prawdopodobieństwo wystąpienia mniejsze niż 40%
tryb	nadmiarowy	niewłaściwa implementacja nadmiarowa przypadku testowego (np. uruchomienie procedury usuwania błędów bez sprawdzenia, czy one wystąpiły)
	prawidłowy	prawidłowo zaimplementowany przypadek testowy
	niepełny	niewłaściwa implementacja przypadku testowego np. brakujące elementy w sprawdzające poprawne warunki wstępne testu
typ	komunikacja	negatywny rezultat spowodowany błędami na magistralach komunikacyjnych
	tolerancja	negatywny rezultat spowodowany przekroczeniem założonych wartości tolerancji
	fluktuacja	negatywny rezultat spowodowany fluktuacją prądu, temperatury, itp.
	czas	negatywny rezultat spowodowany przekroczeniem założonych ram czasowych dla zmiany sygnałów
	inne	pozostałe przyczyny

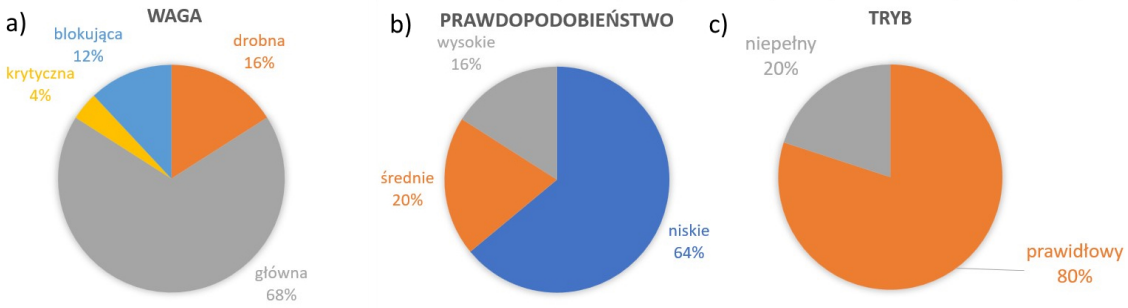
Testom nadano identyfikatory w postaci kodu testowanej funkcji wraz z symboliczną nazwą testu. Na podstawie raportów z dziesięciokrotnego wykonania testów określono następujące atrybuty testów: waga, prawdopodobieństwo oraz tryb. Posługując się algorytmem przedstawionym na Rys. 5.5 oszacowano możliwe przyczyny negatywnego wyniku testu oraz określono typ

anomalii. Wyniki kategoryzacji badanych testów sygnalizowanych migotaniem przedstawiono w Tab. 5.3.

Tab. 5.3. Wyniki analizy testów o rezultatach migoczących [83]

ID	Opis	Waga	Prawdopodobieństwo	Tryb	Typ
1	ASF_006_CalM	drobna	niskie	prawidłowy	czas / komunikacja
2	SF_001_DiagServ	główna	wysokie	prawidłowy	tolerancja
3	SF_014_ErrHandl	krytyczna	średnie	prawidłowy	komunikacja
4	SF_013_Obsrv	drobna	niskie	prawidłowy	komunikacja
5	SF_012_LvSense	główna	niskie	prawidłowy	fluktuacja
6	SF_002_VehCan	blokująca	niskie	niepełny	inne (XCP)
7	SF_003_ConIf	drobna	średnie	prawidłowy	Inne (BusSleep)
8	SF_001_CmcMeas	główna	niskie	prawidłowy	fluktuacja
9	SF_003_PickUp	główna	wysokie	prawidłowy	inne (BusSleep)
10	SF_004_BCK	główna	niskie	prawidłowy	fluktuacja
11	SF_002_Airbag	główna	niskie	prawidłowy	inne (XCP)
12	SF_005_BalanOff	główna	wysokie	prawidłowy	inne (BusSleep)
13	SF_013_overcurr	główna	niskie	prawidłowy	inne (przekaźniki)
14	SF_013_undercurr	główna	niskie	prawidłowy	tolerancja
15	SF_001_BattMode	główna	średnie	niepełny	inne (XCP)
16	SF_001_TempMeas	główna	niskie	niepełny	inne (XCP)
17	SF_001_CalibIdent	drobna	niskie	prawidłowy	inne (XCP)
18	SF_001_VehMeas	główna	niskie	niepełny	inne (BusSleep)
19	SF_001_LAD_Meas	główna	niskie	niepełny	inne (XCP)
20	SF_007_SeccAcc	główna	niskie	prawidłowy	komunikacja
21	SF_007_ProgPrecon	blokująca	średnie	prawidłowy	komunikacja
22	SF_007_EraseFlash	blokująca	niskie	prawidłowy	komunikacja
23	SF_008_DTC_PosRes	główna	niskie	prawidłowy	komunikacja
24	SF_008_DTC_Btl	główna	wysokie	prawidłowy	komunikacja
25	SF_009_IsoMon	główna	średnie	prawidłowy	tolerancja

Wyniki analizy trzech atrybutów: wagi negatywnego rezultatu, prawdopodobieństwa jego wystąpienia oraz trybu zostały przedstawione na Rys. 5.6. W przypadku anomalii zidentyfikowanej jako uniemożliwiającą przeprowadzenie testu wagę *blokującą* przypisano do 12% analizowanych przypadków testowych (Rys. 5.6a). Z kolei anomalia wpływająca nie tylko na bieżący rezultat, ale powodująca błąd w środowisku testowym została zidentyfikowana jako waga *krytyczna* w 4% przypadków testowych. 68% przypadków testowych otrzymało, wagę *główną*, oznaczającą, że anomalia występująca w środowisku jest przyczyną negatywnego wyniku testowanej funkcjonalności, ale nie zaburza pracy całego systemu wbudowanego. Wagę *drobną*, oznaczającą negatywny wynik asercji niezwiązanych bezpośrednio z testowaną funkcjonalnością, przypisano do 16% badanych przypadków testowych. Podsumowując oznacza to, że w przypadku większości testów (84%), które uzyskały wynik negatywny, należy szczegółowo przeanalizować przyczynę niepowodzenia przed zgłoszeniem defektu.

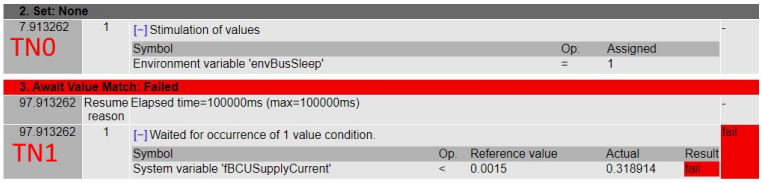


Rys. 5.6. Wyniki analizy testów o rezultatach migoczących [83]

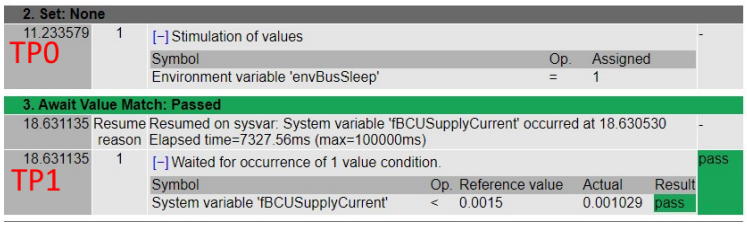
Prawdopodobieństwo wystąpienia anomalii oszacowano na podstawie częstości migotania rezultatów podczas dziesięciokrotnego wykonania (Rys. 5.6b). Określono je jako *niskie* dla 64% analizowanych przypadków testowych, *średnie* dla 20% oraz *wysokie* dla 16%.

Z kolei tryb (Rys. 5.6c), określony jako *niepełny* dla 20% badanych testów oznacza, że wprowadzenie zmian w implementacji przypadku testowego zniweluje wpływ środowiska na pojawiające się wyniki migoczące. Jednak oznacza również to, że 80% przypadków testowych (gdzie tryb oznaczono jako *prawidłowy*) zostało zaimplementowanych poprawnie i inżynier jakości oprogramowania nie jest w stanie zniwelować wpływu środowiska na rezultat testu, zmieniając skrypt testowy.

Określenie typu anomalii prowadzącego do negatywnego rezultatu testu jest najbardziej czasochłonnym oraz wymagającym wiedzy i doświadczenia elementem analizy. Przykładowo, dla testu o ID=12, charakteryzującego się wysokim prawdopodobieństwem wystąpienia migotania oraz określonego jako krytyczny (anomalia uniemożliwiająca przetestowanie funkcji ECU), plik dziennika ma rozmiar prawie 700 MB i zawiera ponad 5 milionów wierszy danych zebranych z symulacji dla 160 różnych wielkości fizycznych. Wstępne przygotowanie polega na wyznaczeniu w pliku raportu chwil czasu TN0=7.91 oraz TN1=97.91 (Rys.5.7), a także TP0=11.23 i TP1=18.63 (Rys. 5.8), oraz wyodrębnieniu odpowiednich fragmentów plików dzienników do dalszej analizy.

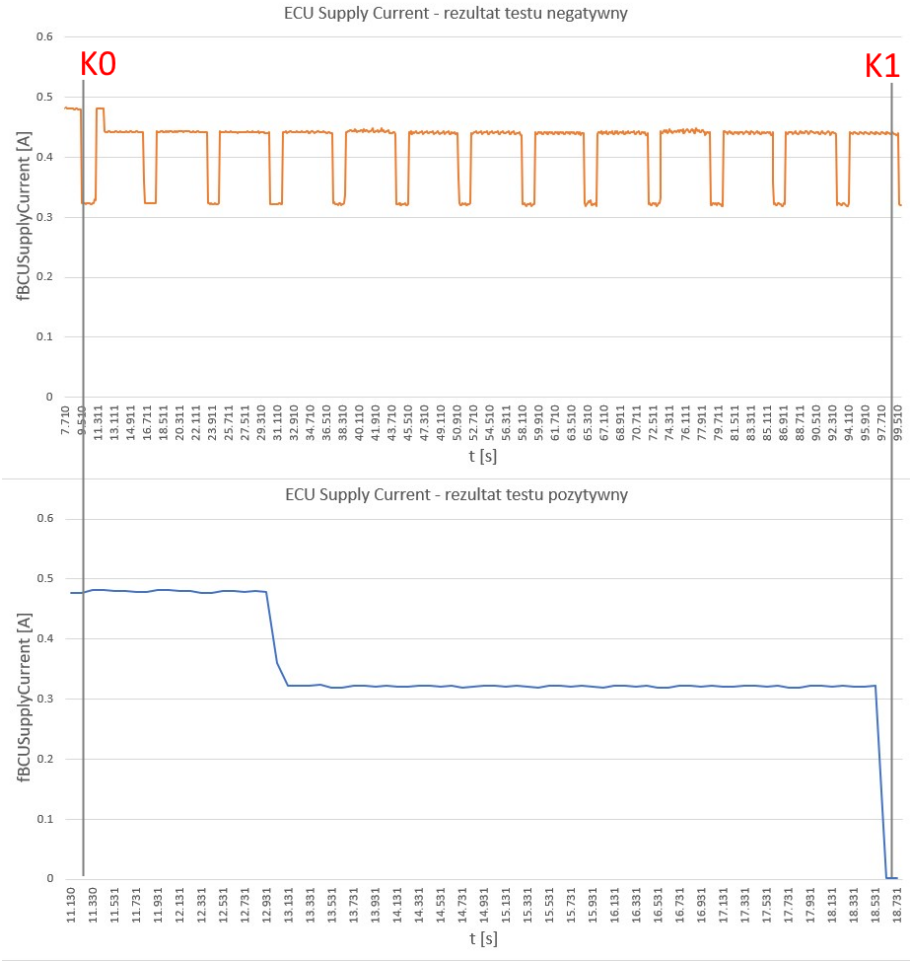


Rys. 5.7. Wyznaczenie TN0 i TN1 na podstawie raportu z wykonania testu



Rys. 5.8. Wyznaczenie TP0 i TP1 na podstawie raportu z wykonania testu

Różnice zauważono m.in. w przebiegu natężenia prądu zasilającego jednostkę sterującą ECU, przedstawionego na Rys. 5.9.



Rys. 5.9. Przebieg prądu zasilania ECU w badanym układzie w czasie od kroku K0 do kroku K1

W kroku K1 zaobserwowano, że natężenie prądu zasilającego ECU w przypadku testu zakończonego pozytywnym wynikiem zbliża się do zera, podczas gdy w przypadku testu z wynikiem negatywnym nie spada poniżej wartości 300 mA. Wynik ten wskazuje, że w przypadku negatywnego wyniku testu system nie przeszedł w oczekiwany stan uśpienia, co można przypisać utrzymaniu dwukierunkowej komunikacji na głównej magistrali łączącej badany system z pozostałą częścią pojazdu. W związku z powyższym ten typ testu został zakwalifikowany do kategorii „Inne (Bus Sleep)".

5.3.3 Etap III — Analiza wpływu zastosowanego modelu pojazdu na występowanie zjawiska migotania rezultatów testów

Jednym z elementów proponowanej metodyki jest modyfikacja środowiska testowego polegająca na integracji z modelem pojazdu (rozdział 4.1). Celem prac badawczych na tym etapie była analiza wpływu modyfikacji środowiska testowego polegająca na integracji z modelem pojazdu na rezultaty testów. Model pojazdu zaimplementowano, aby warunki testowe były bardziej zbliżone do rzeczywistych, oddawały realia związane z dynamiką ruchu pojazdu. Dzięki temu możliwe jest sprawdzenie reakcji systemu wbudowanego w sytuacji rzeczywistego obciążenia magistral komunikacyjnych i zasobów mikrokontrolera sterującego funkcjami układu wbudowanego. Dotychczas np. zjawiska związane z dynamiczną zmianą prędkości pojazdu w wyniku naciśnięcia pedału gazu w czasie testów systemów były pomijane. Model pojazdu zaimplementowany

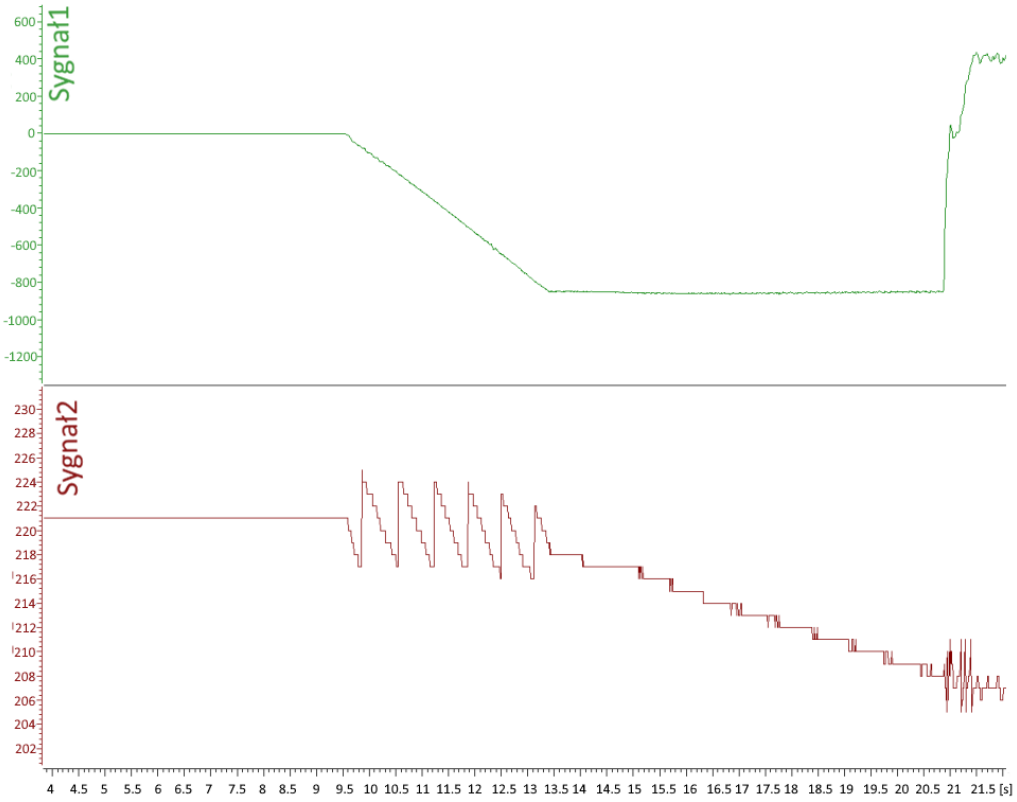
w środowisku testowym zapewnia zbliżoną do rzeczywistej zmienność sygnałów pochodzących z samochodu, związaną z jego cyklem jazdy i dynamiką ruchu. Daje możliwość sterowania obciążeniem magistral komunikacyjnych sygnałami zawierającymi zmienne wartości, odpowiadające rzeczywistej sytuacji na drodze oraz generowanie opóźnień w reakcjach systemu wbudowanego na zmiany.

Pierwszym eksperymentem, przeprowadzonym, aby zweryfikować poprawność zaproponowanego modelu oraz jego przydatność na stanowisku testowym była próba reprodukcji defektu zgłoszonego przez klienta, którego nie wykryły testu w standardowo stosowanym środowisku testowym, a następnie sprawdzenia, czy defekt ten został poprawnie naprawiony. Zaimplementowany przypadek testowy dedykowany do weryfikacji zmian w systemie wbudowanym po naprawieniu defektu został wykonany czterokrotnie:

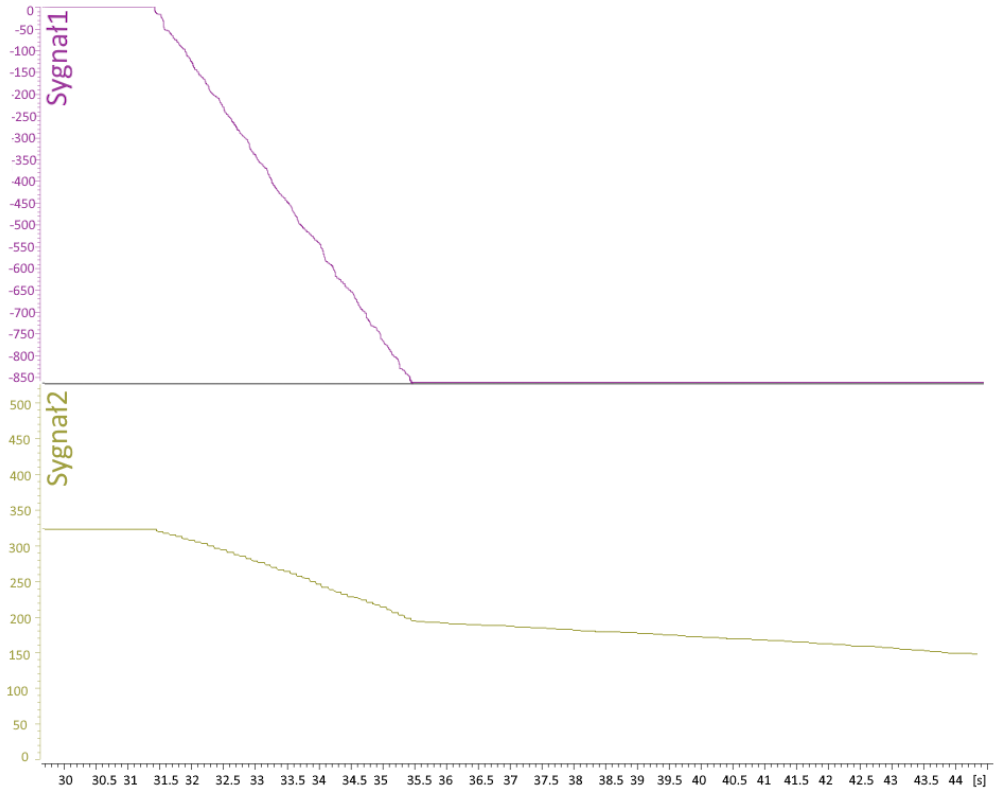
1. z systemem zawierającym defekt, w środowisku testowym niezawierającym modelu pojazdu;
2. z systemem zawierającym defekt, w środowisku testowym z zaimplementowanym modelem pojazdu;
3. w systemie z naprawionym defektem, w środowisku testowym niezawierającym modelu pojazdu;
4. w systemie z naprawionym defektem, w środowisku testowym z zaimplementowanym modelem pojazdu.

Wykonanie testu w systemie zawierającym defekt, w środowisku niezawierającym modelu dało pozytywny rezultat testu, oscylacje sygnału (Sygnał2) związanego z rozładowywaniem baterii w czasie naciskania pedału gazu były niewidoczne. Defekt jest zamaskowany, niemożliwy do zreprodukowania. Wprowadzenie modelu do środowiska testowego dało rezultat negatywny testu — na wykresie widoczne są oscylacje sygnału, świadczące o defekcie w systemie wbudowanym (Rys. 5.10a). Zastosowanie modelu pojazdu przyczyniło się więc do wykrycia niedopuszczalnych oscylacji, co oznacza, że wykonanie tego samego przypadku testowego w systemie wbudowanym z naprawionym oprogramowaniem i uzyskanie rezultatu pozytywnego jest dowodem na usunięcie defektu z systemu (Rys. 5.10b).

Zastosowanie modelu pojazdu w środowisku testowym umożliwia wykrywanie defektów w systemie związanych z dynamiką działania kierowcy (np. odwzorowując dynamikę zmian wciśnięcia pedału gazu w zmienność prędkości pojazdu) oraz reakcji systemu wbudowanego na zjawiska związane bezpośrednio lub pośrednio z mechaniką pojazdu.



a) system wbudowany zawierający defekt



b) system wbudowany z naprawionym defektem

Rys. 5.10. Porównanie wyników testu systemu wbudowanego z zastosowaniem modelu pojazdu

Kolejna modyfikacja środowiska testowego miała na celu umożliwić generowanie aberracji w środowisku testowym. Najprostsze do zasymulowania są anomalie sklasyfikowane jako *komunikacja* oraz *inne (XCP, BusSleep)* (Tab. 5.3). W pierwszym przypadku przygotowano skrypt generujący wzmożony ruch na wszystkich magistralach komunikacyjnych w kierunku sterownika ECU, natomiast w drugim skrypt wysyłający żądania odczytu zmiennych związanych ze stanem naładowania baterii z użyciem protokołu XCP. W celu wygenerowania błędów związanych z *tolerancją* wartości napięć i prądów odczytywanych z systemu zwiększono dynamikę zmian wartości generowanych z cel baterii oraz czujnika natężenia prądu i napięcia baterii (*BattCurrent, BattVoltage, BattTemp* - Rys. 4.15).

Eksperyment badawczy, którego celem była weryfikacja przydatności opracowanej koncepcji stanowiska testowego do dalszych badań, polegał na wykonaniu 20 wcześniej przebadanych przypadków testowych z tabeli 5.3. Przeprowadzono go w dwóch krokach:

- krok 1 - w oryginalnym, projektowym środowisku testowym,
- krok 2 - w środowisku testowym z zaimplementowanym modelem pojazdu oraz zaproponowanymi modyfikacjami umożliwiającymi generowanie anomalii w środowisku.

Zestaw 20 testów wykonano dziesięć razy w losowej kolejności, a następnie określono częstość migotania rezultatów (wystąpienia wyniku negatywnego w 10 próbach). Przykładowe wyniki dla przypadku testowego *ID 9 SF_003_PickUp* przedstawiono w Tab. 5.4 w postaci liczby uzyskanych rezultatów negatywnych L_n oraz częstości migotania ε_M.

Tab. 5.4. Wyniki weryfikacji przydatności modyfikacji środowiska testowego — możliwość generowania aberracji w środowisku — dla przypadku testowego *ID 9 SF_003_PickUp*

	Krok 1		**Krok 2**	
Iteracja	L_n	ε_M	L_n	ε_M
1	6	0.6	10	1.0
2	0	0	10	1.0
3	2	0.2	9	0.9
4	3	0.3	10	1.0
5	2	0.2	9	0.9
6	3	0.3	10	1.0
7	3	0.3	10	1.0
8	4	0.4	10	1.0
9	3	0.3	10	1.0
10	3	0.3	10	1.0

Proces ten został powtórzony dziesięciokrotnie, a w rezultacie eksperymentu określono średnią częstość migotania rezultatów ε_M, jego medianę i odchylenie standardowe. Wyniki eksperymentu przedstawiono w Tab. 5.5. ID w kolumnie Test odpowiada ID testu z tabeli 5.3. Uzyskane rezultaty pokazują, że najtrudniej wywołać w środowisku testowym anomalie związane z tolerancją mierzonych wielkości fizycznych. Zastosowanie zaimplementowanych modyfikacji w środowisku umożliwia zwiększenie liczby negatywnych rezultatów. W przypadku anomalii związanych z komunikacją na magistrali CAN czy innymi przyczynami (XCP, BusSleep) możliwe jest uzyskanie 98% negatywnych wyników testów.

Tab. 5.5. Zestawienie wyników weryfikacji przydatności modyfikacji środowiska testowego – możliwość generowania aberracji w środowisku

	Pierwszy etap			**Drugi etap**		
ID	**Średnia** ε_M	**Mediana** ε_M	**Odchylenie standardowe** ε_M	**Średnia** ε_M	**Mediana** ε_M	**Odchylenie standardowe** ε_M
1	0.08	0.1	0.06	0.14	0.1	0.07
2	0.06	0.05	0.07	0.16	0.1	0.16
3	0.14	0.1	0.08	0.15	0.2	0.07
4	0.36	0.4	0.18	0.42	0.35	0.29
5	0.33	0.35	0.15	0.34	0.4	0.14
6	0.08	0.05	0.12	0.11	0.05	0.13
7	0.07	0.05	0.08	0.08	0.05	0.10
8	0.29	0.3	0.14	0.92	0.9	0.04
9	0.29	0.3	0.14	0.98	1.0	0.04
10	0.27	0.15	0.28	0.94	0.95	0.07
11	0.35	0.3	0.24	0.97	1.0	0.09
12	0.45	0.4	0.27	0.93	0.9	0.06
13	0.47	0.4	0.23	0.97	1.0	0.06
14	0.23	0.15	0.20	0.96	1.0	0.07
15	0.62	0.7	0.22	0.92	1.0	0.11
16	0.46	0.35	0.19	0.95	0.95	0.05
17	0.33	0.3	0.25	0.97	1.0	0.05
18	0.43	0.35	0.30	0.98	1.0	0.06
19	0.15	0.1	0.09	0.98	1.0	0.04
20	0.47	0.4	0.25	0.91	0.95	0.10

5.3.4 Etap IV — Sprawdzenie poprawnej identyfikacji fałszywie negatywnych wyników testów przez zaimplementowane algorytmy sztucznej inteligencji

Tradycyjne podejście do analizowania negatywnych wyników testów wymaga od testerów skrupulatnego przeglądania wyników testów, identyfikowania problemów i określania ich przyczyn źródłowych. Proces ten, aby był skuteczny, jest czasochłonny, wymaga dużych zasobów i jest podatny na błędy ludzkie. Co więcej, wraz ze wzrostem złożoności systemów i oprogramowania, ilość danych testowych rośnie wykładniczo, przez co ręczna analiza staje się coraz bardziej niepraktyczna. Wraz z rozwojem algorytmów sztucznej inteligencji, możliwość automatyzacji tego krytycznego etapu testowania staje się realna, oferując nie tylko optymalizację procesu testowego, lecz także zwiększenie jego ogólnej niezawodności i efektywności.

Eksperyment 1 - Weryfikacja działania zaimplementowanych modeli rekurencyjnych sieci neuronowych

Celem niniejszego eksperymentu jest potwierdzenie skuteczności zaproponowanego procesu identyfikacji fałszywie negatywnych wyników testów z zastosowaniem rekurencyjnych sieci neuronowych. W ramach tego badania wyselekcjonowano cztery przypadki testowe, których rezultaty były obarczone występowaniem zjawiska migotania (Tab. 5.3): Test o ID 7 SF_003_ConI, test o ID 9 SF_003_PickUp, test o ID 12 SF_005_BalanOff, test o ID 18 SF_001_VehMeas. W czasie eksperymentu wykorzystano zaprojektowane stanowisko testowe z zaimplementowanym mode-

lem symulującym zachowanie pojazdu.

Zebrano informacje w postaci raportów oraz dzienników z przebiegu testów. Zgromadzone dane zostały przygotowane zgodnie z wytycznymi przedstawionymi w rozdziale 4.1. W Tab. 5.6 przedstawiono informacje o danych wykorzystanych w IV etapie badań.

Tab. 5.6. Metadane plików przygotowanych w celu przeprowadzenia etapu IV prac badawczych

Test ID (Tab. 5.3)	**Nazwa pliku**	**Liczba próbek**	**Liczba cech**	**Liczba anomalii**	**Liczba negatywnych rezultatów testów**
7	Apassed1.csv	113601	23	0	0
9	Apassed2.csv	257600	23	0	0
12	Apassed3.csv	102701	23	0	0
18	Apassed4.csv	69408	23	0	0
7	Afailed1.csv	231800	23	24081	9
9	Afailed2.csv	146100	23	12040	4
12	Afailed3.csv	169200	23	15050	5
18	Afailed4.csv	124537	23	28083	4

Kolejne kroki eksperymentu wykonano dla algorytmu rozszerzonej inteligencji (pseudokod algorytmu – Algorytm 1) dla dwóch modeli rekurencyjnych sieci neuronowych pokazanych na Rys. 4.31 oraz Rys. 4.33. Oba te modele zostały zaimplementowane w środowisku MATLAB, a następnie wytrenowane. Na Rys. 5.11 przedstawiono szczegóły implementacji modelu oraz parametry trenowania dla sieci z dwoma warstwami LSTM, natomiast na Rys. 5.12 z warstwą GRU.

```
% Define LSTM network layers
layers = [sequenceInputLayer(featureDimension, 'Name', 'in')
   lstmLayer(200,'Name', 'lstm1')
   batchNormalizationLayer
   dropoutLayer(0.2)
   lstmLayer(2,'Name', 'lstm2')
   batchNormalizationLayer
   dropoutLayer(0.2)
   fullyConnectedLayer(featureDimension, 'Name', 'fc')
   regressionLayer('Name', 'out') ];

% Set Training Options
options = trainingOptions('adam', ...
   'Plots', 'training-progress', ...
   'LearnRateSchedule','piecewise',...
   'LearnRateDropPeriod',100,...
    'LearnRateDropFactor',0.95,...
    'InitialLearnRate',0.05, ...
    'L2Regularization',0.01, ...
   'MiniBatchSize', 1E3,...
   'MaxEpochs',250);
```

Rys. 5.11. Szczegóły implementacji oraz parametry treningowe dla RNN z dwoma warstwami LSTM

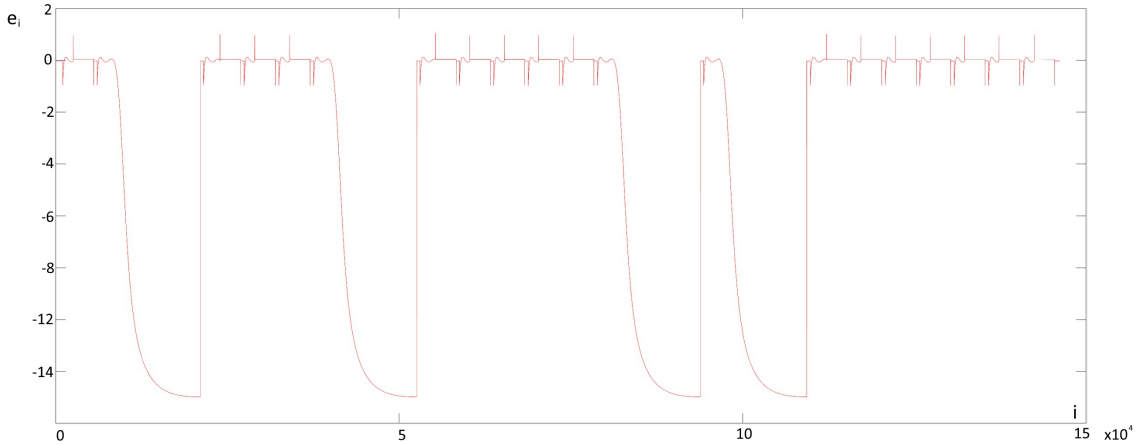
```
featureDimension = 23;

layers = [sequenceInputLayer(featureDimension, 'Name', 'in')
   batchNormalizationLayer
   gruLayer(100,'Name', 'gru1')
   batchNormalizationLayer
   fullyConnectedLayer(1, 'Name', 'fc2')
   regressionLayer('Name', 'out') ];

% Set Training Options
options = trainingOptions('adam', ...
   'Plots', 'training-progress', ...
   'LearnRateSchedule','piecewise',...
   'LearnRateDropPeriod',10,...
    'LearnRateDropFactor',0.95,...
    'InitialLearnRate',1e-2, ...
    'L2Regularization',0.01, ...
   'MiniBatchSize', 10E3,...
   'MaxEpochs',100);
```

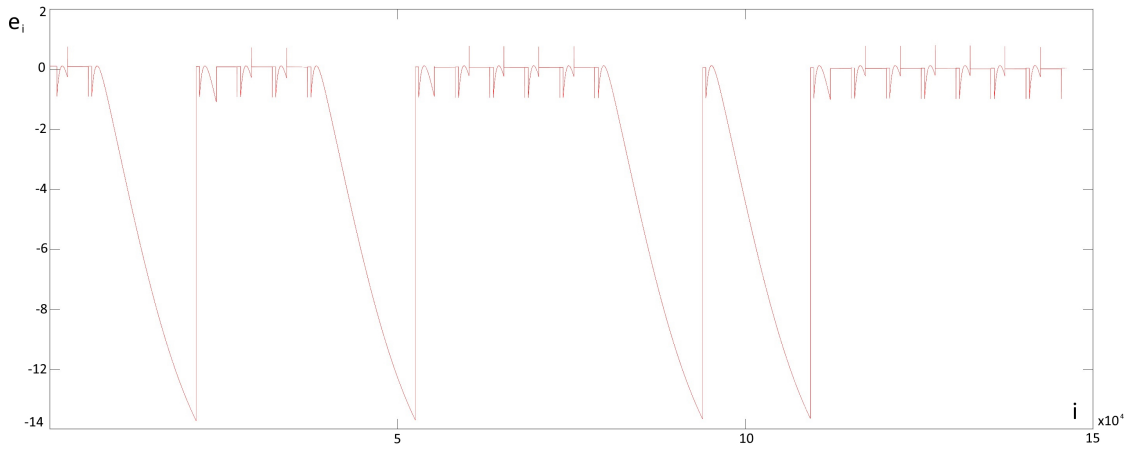
Rys. 5.12. Szczegóły implementacji oraz parametry treningowe dla RNN z warstwą GRU

Do trenowania zastosowano pliki uzyskane podczas wykonania czterech przypadków testowych z rezultatami pozytywnymi (Apassed1.csv, Apassed2.csv, Apassed3.csv, Apassed4.csv). W środowisku testowym nie występowały anomalie wpływające na rezultaty wykonywanych testów. Ocena skuteczności modeli przeprowadzona została za pomocą czterech zestawów danych testowych zgromadzonych podczas dziesięciokrotnego wykonania tych samych przypadków testowych z uzyskanymi rezultatami migoczącymi (Afailed1.csv, Afailed2.csv, Afailed3.csv, Afailed4.csv). Na Rys. 5.13 przedstawiono przykładowy wykres sygnału residuum e_i uzyskanego w modelu zawierającym warstwę LSTM (Rys. 4.31) dla jednego z przypadków testowych. Widoczne są na nim zaburzenia w środowisku testowym, które są przyczyną negatywnego rezultatu testu.



Rys. 5.13. Wykres sygnału residuum dla modelu z warstwą LSTM dla testu ID 9

Podobny rezultat uzyskano dla modelu zawierającego warstwę GRU (Rys. 4.33), co przedstawiono na Rys. 5.14.



Rys. 5.14. Wykres sygnału residuum dla modelu z warstwą GRU dla testu ID 12

RMSE uzyskane w przypadku eksperymentów z warstwą LSTM wynosiło 0.17, natomiast w przypadku modelu z warstwą GRU 0.11. Wyniki tego eksperymentu, przedstawione w Tab. 5.7, pozwoliły na określenie zdolności obu modeli do identyfikacji fałszywie negatywnych wyników testów w kontekście zjawiska migotania rezultatów.

Tab. 5.7. Wyniki identyfikacji anomalii w środowisku testowym przez rekurencyjne sieci neuronowe dla modelu zawierającego warstwę LSTM oraz modelu z warstwą GRU

Test ID (Tab. 5.3)	**Liczba pozytywnych rezultatów**	**Liczba negatywnych rezultatów**	**Liczba wykrytych zaburzeń LSTM**	**Liczba wykrytych zaburzeń GRU**
7	1	9	8	8
9	6	4	4	4
12	5	5	5	4
18	6	4	3	3

Testy wykorzystane w eksperymencie charakteryzowały się różną częstością migotania, nie zaobserwowano jej wpływu na identyfikację fałszywie negatywnych rezultatów przez algorytmy rozszerzonej inteligencji. Skuteczność algorytmów rozszerzonej inteligencji jest wyższa w przypadku algorytmu opartego o sieć RNN z warstwą LSTM tylko w jednym przypadku - dla testu o ID 9 algorytm wskazał, że 5 z negatywnych rezultatów jest spowodowanych anomaliami w środowisku testowym, natomiast algorytm oparty o sieć z warstwą GRU wskazał tylko 4, nie identyfikując jednego z nich.

W wynikach uzyskanych na tym etapie badań stwierdzono, że wykorzystanie wybranych modeli sieci neuronowych do detekcji anomalii w badanym środowisku testowym wymaga zaangażowania specjalisty z zakresu sztucznej inteligencji. W kontekście przemysłowym proponowane rozwiązania powinny być jak najbardziej efektywne pod względem zużycia zasobów, aby zapewniona została pomyślna implementacja projektu w ustalonym czasie.

Z powyższego powodu podjęto decyzję o zbadaniu potencjału algorytmów sztucznej inteligencji przeznaczonych do wykrywania odstających wartości w kontekście rozważanego problemu weryfikacji rezultatów testowania oprogramowania wbudowanego.

Eksperyment 2 - Weryfikacja działania algorytmów dedykowanych do wykrywania anomalii

Do realizacji tego zadania wybrany został język Python ze względu na powszechną dostępność, otwarty charakter oraz bogatą kolekcję narzędzi i bibliotek dedykowanych sztucznej inteligencji. Decyzja poparta jest dostępem do bogatych bibliotek zweryfikowanych algorytmów, których skuteczność została potwierdzona w licznych artykułach naukowych [171]. W kontekście eksploracji i ewaluacji algorytmów dedykowanych wykrywaniu anomalii w środowisku testowym zdecydowano się na wykorzystanie biblioteki Python Outlier Detection (PyOD)[170]. Biblioteka ta wyróżnia się rozbudowanymi implementacjami różnych algorytmów służących do wykrywania anomalii szeroko opisanymi w literaturze. Przed przystąpieniem do tego eksperymentu autorka podjęła próbę oceny skuteczności wykrywania anomalii przez wybrane algorytmy dostępne we wspomnianej bibliotece na danych dostępnych w bibliotece Outlier Detection Dataset ODDS [196] o strukturze zbliżonej do danych rzeczywistych, co opisała w artykule [172].

Algorytmy wybrane do badań zostały szczegółowo opisane w rozdziale 4.3. Pierwszy krok eksperymentu polegał na ewaluacji skuteczności algorytmów nienadzorowanych na rzeczywistym zestawie danych, pochodzącym z logów zarejestrowanych w czasie wykonywania testów o migoczących wynikach. Wykorzystano dane zgromadzone w pliku Afailed.csv (Tab. 5.6), dzieląc dane na treningowe i testowe w stosunku 80% - 20 %. Liczba próbek danych treningowych wynosiła 185 440, liczba anomalii - 21 071, liczba negatywnych rezultatów testów - 7, natomiast dla danych testowych — liczba próbek wynosiła 46 360, liczba anomalii 3 010, liczba negatywnych rezultatów - 2. Wyniki eksperymentu przedstawiono w Tab. 5.8.

Tab. 5.8. Wyniki uzyskane przez algorytmy nienadzorowane z biblioteki PyOD na danych z rzeczywistych logów

	Dane treningowe		**Dane testowe**	
Algorytm	**AUC**	**PRE**	**AUC**	**PRE**
ECOD	0.3177	0.0093	0.3805	0.0141
PCA	0.3105	0.0003	0.3104	0.0003
AvgKNN	0.4666	0.0526	0.3464	0.0
IForest	0.2725	0.0282	0.2751	0.0224
INNE	0.3122	0.0	0.6345	0.0
AutoEncoder	0.3111	0.0057	0.3068	0.0074
ALAD	0.3126	0.0154	0.2765	0.0149

Uzyskane wyniki można określić jako niezadowalające. Modyfikacje hiperparametrów nie zaowocowały poprawą skuteczności wykrywania anomalii w logach z testowanych rzeczywistych systemów wbudowanych.

Postanowiono przeprowadzić taką samą analizę dla algorytmów nadzorowanych. Wyniki oceny skuteczności wykrywania anomalii przez wybrane algorytmy nadzorowane dostępne bibliotece PyOD zostały przedstawione w Tabeli 5.9.

Tab. 5.9. Wyniki uzyskane przez algorytmy nadzorowane z biblioteki PyOD na danych z rzeczywistych logów

	Dane treningowe		**Dane testowe**	
Algorytm	**AUC**	**PRE**	**AUC**	**PRE**
CatB+	0.9998	0.9777	0.7796	0.1314
LGB	0.9997	0.9709	0.7907	0.1203
MLP	0.9983	0.9256	1.0	0.9917
SVM	1.0	0.9944	1.0	1.0
XGB+	1.0	1.0	0.7602	0.0

Podsumowując uzyskane wyniki, niektóre algorytmy, takie jak LGB i CatB+, wykazują się wysoką skutecznością na zbiorze treningowym, jednak istnieje ryzyko nadmiernego dopasowania, co sugerują wyniki na zbiorze testowym. Obserwowany spadek skuteczności kilku modeli przy przejściu na dane testowe sugeruje, że mogą one mieć trudności z generalizacją na nowe dane. Podsumowując, choć niektóre algorytmy wykazują wysoką dokładność na zbiorze treningowym, istotne jest uwzględnienie ich zdolności do generalizacji na dane testowe, co może stanowić kluczowy czynnik przy wyborze algorytmu do konkretnego zastosowania.

Algorytm SVM prezentuje bardzo dobrą skuteczność zarówno pod względem AUC, jak i precyzji, zarówno na zbiorze danych treningowych, jak i testowych. Sugeruje to jego solidne możliwości uogólniania. Podobne wyniki zaobserwowano dla algorytmu MLP, jednak jego niższa skuteczność na zbiorze treningowym może sugerować potencjalne nadmiernie dopasowanie lub wysoką wrażliwość na dane treningowe. Podsumowując, obydwa algorytmy: SVM i MLP, wykazują obiecujące rezultaty, zwłaszcza jeśli chodzi o dokładność predykcji i stabilność wyników na zbiorach danych testowych.

Algorytm XGB+ osiągnął bardzo dobre wyniki na danych treningowych, z AUC oraz precyzją wynoszącymi 1.0. Niemniej jednak, w na danych testowych jego skuteczność znacząco spadła. AUC wyniosło 0.7602, a precyzja spadła do 0.0, co sugeruje, że algorytm ten ma problemy z utrzymaniem stabilności w zmiennych warunkach testowych, co może wskazywać na przetrenowanie.

K-krotna walidacja krzyżowa i wyznaczenie metryk skuteczności algorytmów

W kolejnym kroku eksperymentu 2 zastosowano technikę dziesięciokrotnej walidacji krzyżowej dla wybranych algorytmów nadzorowanych. Polega ona na podziale zbioru danych uczących na K równej wielkości segmentów (w tym przypadku K=10). K-1 podzbiorów danych jest wykorzystywanych do szkolenia modelu (dane treningowe), podczas gdy pozostały podzbiór służy do walidacji (dane testowe). Technika ta obejmuje K iteracji, za każdym razem rezerwując określony podzbiór do walidacji. Wykluczenie próbek treningowych z tych używanych do oceny kandydujących wartości parametrów zmniejsza prawdopodobieństwo nadmiernego dopasowania, zwiększając w ten sposób uogólnienie klasyfikatora [197]. Zbiór danych pochodzący z rzeczywistego loga zarejestrowanego w czasie wykonywania testów migoczących (Afailed.csv - Tab. 5.6) został podzielony na 10 podzbiorów, a procesy uczenia i testowania modelu były powtarzane 10 razy, za każdym razem z inną partią danych traktowaną jako zbiór testowy. Zbiór danych wykorzystany w tym kroku zawierał 231 880 próbek, 24 081 próbek wskazujących na anomalie w środowisku, co odpowiadało 9 negatywnym rezultatom testów. Zastosowano klasę $KFold$ z biblioteki $scikit-learn$, z następującymi parametrami:

- $n\_splits = 10$ - podział danych na 10 części (zwane "foldami"), w każdej iteracji 9 z nich było używanych jako dane treningowe, 1 jako dane testowe.
- $shuffle = True$ - przed podziałem na foldy dane zostaną losowo przetasowane.

- *random_state* = 5 - ustawione zostało deterministyczne ziarno generatora liczb losowych.

Dla badanych algorytmów wyznaczone zostały macierze pomyłek zarówno dla danych treningowych, jak i testowych, obliczane jako suma wartości T_n, F_p, F_n, T_p we wszystkich próbach. Wyniki zostały przedstawione w Tab. 5.10.

Tab. 5.10. Macierz pomyłek uzyskana w wyniku dziesięciokrotnej walidacji krzyżowej

	Dane treningowe				**Dane testowe**			
Algorytm	T_n	F_p	F_n	T_p	T_n	F_p	F_n	T_p
MLP	1 845 825	24 366	7 021	209 708	205 152	2647	758	23 323
SVM	1 869 017	1 174	1 222	215 507	207 668	131	138	23 943
XGB+	1 870 191	0	0	216 729	207 791	7	6	24 075
LGB	1 866 442	3 749	13 464	203 265	207 288	511	1619	22 462
CatB+	1 859 396	10 795	3 043	213 686	206 366	1433	594	23 487

Algorytm SVM i XGB+ osiągnęły wysoką liczbę prawdziwych negatywów (T_n) i prawdziwych pozytywów (T_p) oraz stosunkowo niską liczbę fałszywych pozytywów (F_p) i fałszywych negatywów (F_n) zarówno na danych treningowych, jak i testowych, co wskazuje na skuteczną klasyfikację negatywnych instancji. Podsumowując, algorytmy generalnie wykazują wysoką wydajność w klasyfikowaniu zarówno negatywnych, jak i pozytywnych przypadków, z pewnymi różnicami w równowadze między prawdziwymi i fałszywymi pozytywami.

Badano również skuteczność pięciu różnych algorytmów: MLP, SVM, XGB+, LGB i CatB+ mając na uwadze takie miary oceny jak AUC, PRE, MCC oraz BA (wskaźniki te opisano w rozdziale 4.3.3).

Wyniki walidacji krzyżowej, po obliczeniu średnich wartości wskaźników jakości, przedstawiono w Tab. 5.11.

Tab. 5.11. Wybrane metryki uzyskane w wyniku dziesięciokrotnej walidacji krzyżowej

Dane treningowe				
Algorytm	**AUC**	**PRE**	**MCC**	**BA**
MLP	0.9994	0.9610	0.9271	0.9773
SVM	1.0000	0.9945	0.9938	0.9969
XGB+	1.0000	1.0000	1.0000	1.0000
LGB	0.9995	0.9593	0.9551	0.9679
CatB+	0.9996	0.9668	0.9651	0.9901
Dane testowe				
MLP	0.9994	0.9599	0.9287	0.9779
SVM	1.0000	0.9945	0.9938	0.9968
XGB+	1.0000	0.9997	0.9996	0.9998
LGB	0.9993	0.9511	0.9499	0.9651
CatB+	0.9994	0.9548	0.9539	0.9842

Wszystkie algorytmy wykazują bardzo wysoką moc dyskryminacyjną, z wynikiem AUC bliskim lub równym 1,0000 zarówno dla danych treningowych, jak i testowych. Algorytmy SVM, XGB+ i CatB+ wykazują bardzo dobry wynik AUC, wskazując na optymalną wydajność w rozróżnianiu klas. Dla wszystkich algorytmów wartości PRE są niezmiennie wysokie zarówno dla danych treningowych, jak i testowych, co odzwierciedla zdolność modeli do prawidłowej identyfikacji pozytywnych instancji. Algorytmy SVM, XGB+ i CatB+ wyróżniają się wartościami precyzji zbliżonymi lub równymi 1,0000 na danych testowych. Wartości MCC, które mierzą jakość kla-

syfikacji binarnych, są konsekwentnie wysokie we wszystkich algorytmach i zbiorach danych. Algorytmy XGB+ i SVM osiągnęły szczególnie wysokie wyniki MCC, wskazując na niezawodną wydajność klasyfikacji. Wyniki zrównoważonej dokładności (BA) są konsekwentnie wysokie, co wskazuje na dobrze zrównoważoną wydajność między czułością a swoistością dla wszystkich algorytmów. Algorytmy XGB+ i SVM wykazują najwyższe wartości zrównoważonej dokładności (BA) na danych testowych.

Podsumowując, oceniane algorytmy konsekwentnie osiągają dobre wyniki w szerokim zakresie wskaźników, co pokazuje ich skuteczność w realizacji zadania klasyfikacji. Wybór najbardziej odpowiedniego algorytmu może zależeć od konkretnych priorytetów, takich jak precyzja, interpretowalność lub wydajność obliczeniowa, na co wskazują różne mocne strony zaobserwowane w metrykach.

Weryfikacja algorytmów na niezależnych danych

Po procesie walidacji krzyżowej konieczne jest przeprowadzenie ostatecznej oceny skuteczności na nowym, nieznanym zbiorze testowym. Niezbędne jest potwierdzenie, że model jest dobrze zaznajomiony ze wzorcami w danych i nie wykazuje nadmiernego dopasowania [198].

Podczas tego kroku badawczego cały zbiór danych wykorzystanych w poprzednich badaniach został zastosowany jako zbiór danych treningowych (dane zgromadzone w pliku Afailed1.csv — Tab.5.6), natomiast jako dane testowe wykorzystane zostały nowe, nieznane dla algorytmów dane zebrane podczas wykonywania odrębnego od poprzednich przypadku testowego (dane zgromadzone w pliku Afailed2.csv — Tab.5.6). W tym przypadku treningowy zbiór danych zawiera 231 880 próbek (w tym 24 081 próbek wskazujących na anomalię w środowisku, co stanowi 10%), podczas gdy testowy zbiór danych zawiera 146 100 próbek (w tym 12 040 anomalii, co również stanowi 10%). Wyniki oceny algorytmów przedstawiono w postaci wskaźników jakości w tabeli 5.12, a macierze pomyłek w tabeli 5.13.

Tab. 5.12. Wybrane metryki skuteczności algorytmów uzyskane w wyniku weryfikacji na niezależnych danych

Dane treningowe				
Algorytm	**AUC**	**PRE**	**MCC**	**BA**
MLP	0.9998	0.9787	0.981	0.984
SVM	1.0000	0.9949	0.994	0.997
XGB+	1.0000	1.0000	1.0000	1.0000
LGB	0.9996	0.9603	0.958	0.964
CatB+	0.9997	0.9734	0.969	0.993
Dane testowe				
MLP	0.9997	0.9661	0.970	0.974
SVM	0.9999	0.9949	0.980	0.982
XGB+	0.7693	0.2514	0.140	0.577
LGB	0.7921	0.3668	0.203	0.589
CatB+	0.7300	0.2846	0.490	0.629

Algorytmy MLP, SVM utrzymały wysoką skuteczność na danych testowych, osiągając wysokie wyniki w AUC, precyzji, MCC i zrównoważonej dokładności (BA). Chociaż zauważalny jest niewielki spadek skuteczności w porównaniu do zestawu treningowego. Skuteczność XGB+, LGB i CatB+ jest nieoptymalna na zestawie testowym, podkreślając wyzwania związane z uogólnianiem.

Tab. 5.13. Macierze pomyłek uzyskane w wyniku weryfikacji na niezależnych danych

	Dane treningowe				Dane testowe			
Algorytm	T_n	F_p	F_n	T_p	T_n	F_p	F_n	T_p
MLP	207 750	49	754	23 327	134 114	24	635	11 405
SVM	207 677	122	125	23 956	134 121	17	428	11 612
XGB	207 799	0	0	24 081	122 044	12 094	9 083	2 957
LGB	207 736	63	1 747	22 334	127 877	6 261	9 342	2 698
CatB+	206 760	1 039	149	23 932	134 138	0	8 933	3 107

Algorytmy MLP, SVM, XGB, LGB i CatB+ wykazały wysoką skuteczność na danych treningowych, z wysoką liczbą wyników prawdziwie pozytywnych (Tp) i stosunkowo niską liczbą wyników fałszywie pozytywnych (Fp) i fałszywie negatywnych (Fn). SVM i MLP wyróżniały się bardzo wysoką liczbą prawdziwych negatywów (Tn), co wskazuje na precyzyjną klasyfikację negatywnych instancji. XGB i LGB niezbyt dobrze klasyfikowały pozytywne instancje w porównaniu z pozostałymi algorytmami, co skutkowało niską wartością prawdziwie pozytywnych wyników i wysoką fałszywie negatywnych. XGB+ wykazywał najwyższą liczbę fałszywych pozytywów, wskazując na wyzwania w rozróżnianiu negatywnych instancji. Wybór najbardziej odpowiedniego algorytmu powinien być dostosowany do konkretnych potrzeb i ograniczeń zadania klasyfikacji. Wyniki tego eksperymentu, przedstawione w Tab. 5.14, pozwoliły na określenie zdolności algorytmu rozszerzonej inteligencji (Algorytm 2) do identyfikacji fałszywie negatywnych wyników testów w kontekście zjawiska migotania rezultatów w zależności od zastosowanego algorytmu do wykrywania anomalii.

Tab. 5.14. Wyniki działania rozszerzonej inteligencji w zadaniu identyfikacji anomalii w środowisku testowym

	Dane treningowe			Dane testowe		
Algorytm	**Liczba wyników testów**	**Liczba negatywnych rezultatów**	**Liczba wykrytych zaburzeń**	**Liczba wyników testów**	**Liczba negatywnych rezultatów**	**Liczba wykrytych zaburzeń**
MLP	10	9	9	10	4	4
SVM		9	9		4	4
XGB+		9	9		4	1
LGB		9	7		4	1
CatB+		9	9		4	2

Podsumowując, podczas gdy niektóre algorytmy wykazały wysoką skuteczność w różnych metrykach, zdolność do uogólniania na nowe dane była różna. Algorytmy SVM i MLP wyróżniały się jako wysoce niezawodne modele zarówno w zestawach danych treningowych, jak i testowych, co czyni je silnymi kandydatami do praktycznych zastosowań.

Wnioski z eksperymentu stanowić będą istotny wkład w rozwijanie sposobu poprawy dokładności testów oraz weryfikacji skuteczności zaproponowanych modeli w warunkach diagnostycznych.

5.3.5 Etap V — Końcowa weryfikacja zastosowanej metodyki

Celem tego badania jest weryfikacja zaproponowanej metodyki tworzenia automatycznych przypadków testowych pod kątem skuteczności walidacji wyników testów systemów wbudowanych

przez zaimplementowane algorytmy rozszerzonej inteligencji. Do prac badawczych wybrano dwa przypadki testowe z Tab. 5.3 - Test o ID 7 SF_003_ConI, test o ID 9 SF_003_PickUp (w dalszej części pracy, identyfikowane jako TC_1 oraz TC_2), których specyfikacja oraz implementacja została przeprowadzona zgodnie z zaproponowaną metodyką (Rozdział 4.1). Obydwa przypadki podczas wielokrotnego wykonywania uzyskiwały wyniki migoczące.

W ramach tego etapu badań wykonano dwa eksperymenty, których celem była ocena skuteczności działania algorytmów rozszerzonej inteligencji podczas identyfikacji fałszywie negatywnych rezultatów spowodowanych anomaliami w środowisku. Zbadano skuteczność rozszerzonej inteligencji (Algorytm 2) w zależności od zastosowanego algorytmu AI, aby potwierdzić, że zastosowana podczas procesu specyfikacji, implementacji oraz wykonania przypadku testowego metodyka jest skuteczna. Kryterium oceny było poprawne wskazanie wszystkich fałszywie negatywnych rezultatów testów. Eksperyment 1 polegał na zastosowaniu danych treningowych i testowych zebranych podczas wykonywania tego samego przypadku testowego (TC_1), ale w różnych sesjach testowych. Natomiast Eksperyment 2 obejmował trening danymi treningowymi z Eksperymentu 1, ale jako dane testowe zostały zastosowane dane zebrane podczas wykonania przypadku testowego TC_2. Celem tego eksperymentu było sprawdzenie, czy niewielka ilość danych zebranych ze środowiska testowego podczas wykonania jednego przypadku testowego jest wystarczająca, aby wytrenować algorytmy sztucznej inteligencji tak, aby były zdolne wykrywać anomalie pojawiające się w środowisku testowym podczas wykonywania różnych przypadków testowych. Każdy eksperyment przeprowadzono w dwóch krokach:

- Krok 1 - wykonanie przypadków testowych w środowisku bez zaimplementowanego modelu pojazdu, gdzie symulowane sygnały pochodzące z samochodu miały wartości statyczne.
- Krok 2 - wykonanie przypadków testowych w środowisku z zaimplementowanym modelem pojazdu, w którym dynamika ruchu pojazdu odzwierciedlona była w sygnałach zasilających symulację.

Dane treningowe przygotowane do tego etapu zadań zebrano podczas pięciokrotnego wykonania przypadku TC_1. Natomiast dane testowe zgromadzono podczas jednokrotnego wykonania przypadków TC_1 i TC_2. Dane te zostały przygotowane zgodnie z wytycznymi przedstawionymi w rozdziale 4.1, a informacje o nich przedstawiono w Tab. 5.15.

Tab. 5.15. Metadane plików przygotowanych w celu przeprowadzenia etapu V

Test ID (Tab. 5.3)	**Model pojazdu**	**Nazwa pliku**	**Liczba próbek**	**Liczba cech**	**Liczba anomalii**	**Liczba negatywnych rezultatów testów**	**Ogólna liczba rezultatów testów**
TC_1	NIE	A_II.csv	231645	68	39528	2	5
	TAK	B_II.csv	398590	68	50962	2	5
	TAK	C_II.csv	52247	68	6995	1	1
	NIE	D_II.csc	119030	68	8209	1	1
TC_2	NIE	A_I.csv	833853	68	216828	1	1
	TAK	B_I.csv	812785	68	214896	1	1

Należy podkreślić, że liczba próbek w danych treningowych w przypadku wykonywania tego samego testu w środowisku z modelem pojazdu jest prawie dwukrotnie większa (231645 próbek dla środowiska bez modelu pojazdu, 398590 próbek dla środowiska z modelem pojazdu),

natomiast liczba próbek sygnalizujących anomalię w środowisku spadła z 17% (39528 próbek) do 12.8% (50964 próbki) w przypadku zastosowania modelu pojazdu.

W poniższych tabelach przedstawiono wyniki przeprowadzonych eksperymentów. Porównano skuteczność badanych wcześniej algorytmów w identyfikacji anomalii w środowisku testowym powodujących negatywny rezultat testu (a w konsekwencji migotanie rezultatów) na podstawie macierzy pomyłek przedstawionej w Tab 5.16 oraz wybranych wskaźników pokazanych w tabeli 5.17.

Tab. 5.16. Wyniki badań weryfikacyjnych: Macierz pomyłek

	Bez modelu pojazdu				**Z modelem pojazdu**			
Algorytm	T_n	F_p	F_n	T_p	T_n	F_p	F_n	T_p
Dane treningowe								
MLP	192117	0	3	39525	347626	0	2	50962
SVM	191993	127	93	39432	347613	13	7	50957
XGB+	191603	514	187	39341	347595	31	17	50947
LGB	191315	802	113	39415	347598	28	45	50919
CatB+	191298	819	33	39495	347567	59	47	50917
Dane testowe - Eksperyment 1								
MLP	110821	0	30	8179	45252	0	1	6994
SVM	110719	102	96	8113	45225	27	23	6972
XGB+	110689	132	33	8176	45211	41	27	6968
LGB	109806	1015	50	8159	45167	85	10	6985
CatB+	109951	870	75	8134	45159	93	19	6976
Dane testowe - Eksperyment 2								
MLP	617025	0	216828	0	591887	6002	3639	211257
SVM	617025	0	216828	0	597607	282	192517	22379
XGB+	616476	549	10	216818	591053	6836	14	214882
LGB	326935	290090	0	216828	591036	6853	0	214896
CatB+	616499	526	6	216822	591054	6835	214896	0

Tab. 5.17. Wyniki badań weryfikacyjnych: wybrane wskaźniki skuteczności algorytmów

	Bez modelu pojazdu				**Z modelem pojazdu**			
Algorytm	AUC	PRE	MCC	BA	AUC	PRE	MCC	BA
Dane treningowe								
MLP	1	0.9999	0.9999	0.9999	1	1	0.9999	0.9999
SVM	0.9985	0.9968	0.9966	0.9985	0.9999	0.9997	0.9998	0.9999
XGB+	0.9963	0.9871	0.9894	0.9963	0.9998	0.9994	0.9995	0.9998
LGB	0.9965	0.9800	0.9862	0.9965	0.9995	0.9995	0.9992	0.9995
CatB+	0.9975	0.9797	0.9871	0.9975	0.9995	0.9988	0.9988	0.9995
Dane testowe — Eksperyment 1								
MLP	0.9982	0.9963	0.9980	0.9981	1	0.9999	0.9999	0.9999
SVM	0.9937	0.9876	0.9870	0.9937	0.9981	0.9961	0.9959	0.9980
XGB+	0.9974	0.9841	0.9892	0.9974	0.9976	0.9941	0.9944	0.9976
LGB	0.9924	0.8894	0.9356	0.9924	0.9983	0.9880	0.9922	0.9984
CatB+	0.9915	0.9034	0.9420	0.9915	0.9976	0.9868	0.9908	0.9976
Dane testowe - Eksperyment 2								
MLP	0.9715	0.8145	0	0.5	0.9865	0.9724	0.9696	0.9865
SVM	0.7207	0.23	0	0.5	0.5518	0.9876	0.2777	0.5518
XGB+	0.9974	0.9841	0.9982	0.9995	0.9943	0.9692	0.9788	0.9942
LGB	0.9996	0.4277	0.4760	0.7649	0.9943	0.9691	0.9787	0.9943
CatB+	0.9994	0.9977	0.9983	0.9996	0.4943	0	-0.5521	0.4942

Skuteczność treningu algorytmów na danych zebranych w środowisku zawierającym model pojazdu jest widocznie lepsza niż w przypadku danych ze środowiska bez modelu. W pierwszym przypadku wszystkie algorytmy uzyskały podobne wyniki, z pomijalną liczbą fałszywych pozytywów F_p oraz fałszywych negatywów F_n. W drugim przypadku najlepsze rezultaty uzyskały algorytmy MLP oraz SVM, a najgorsze LGB oraz CATB+, co przełożyło się na minimalne różnice w wyznaczonych wskaźnikach skuteczności algorytmów przedstawionych w tabeli 5.17. Natomiast algorytmy AuI poprawnie wskazały fałszywie negatywne rezultaty testów we wszystkich przypadkach.

Wyniki uzyskane w Eksperymencie 1 potwierdziły skuteczność zastosowanej metodyki. Algorytm MLP wykazał wysoką skuteczność, osiągając zerową liczbę fałszywych pozytywów F_p zarówno dla danych uzyskanych z modelem pojazdu, jak i bez niego. W przypadku uwzględnienia modelu pojazdu wszystkie algorytmy wykazały lepszą skuteczność, osiągając wyższe wartości prawdziwych pozytywów T_p i zredukowaną liczbę fałszywych negatywów F_n. W tym eksperymencie potwierdzona została skuteczność algorytmu rozszerzonej inteligencji w identyfikacji fałszywie negatywnych rezultatów testów i wszystkie fałszywie negatywne rezultaty zostały poprawnie zidentyfikowane.

Najlepsze wyniki w Eksperymencie 2 uzyskał algorytm XGB+, uzyskując małą liczbę fałszywych pozytywów F_p (549) dla danych bez modelu, nieco gorzej było w przypadku danych z modelem pojazdu F_p (6836), natomiast w obydwu przypadkach ilość fałszywych negatywów F_n była znikoma. Wyznaczone wskaźniki skuteczności algorytmu oscylują wokół wartości 0.98, co świadczy o jego wiarygodności i użyteczności w praktycznych zastosowaniach. Wyniki uzyskane przez algorytmy MLP oraz SVM na danych zgromadzonych bez udziału modelu pojazdu ($T_p = 0$) wskazują na problemy z klasyfikacją pozytywnych instancji. Wskaźniki MCC = 0 oraz BA = 0.5 dla tych algorytmów oznaczają, że model nie ma zdolności przewidywania i jego wyniki są równoważne do losowego zgadywania. Na danych zawierających model pojazdu skuteczność algorytmu MLP jest bardzo dobra (wszystkie wyznaczone wskaźniki skuteczności algorytmu oscylują wokół

wartości 0.97). Natomiast SVM mimo wysokiej precyzji (PRE), uzyskał niskie pozostałe metryki (AUC, MCC, BA), co wskazuje na słabą zdolność modelu do poprawnego przewidywania klas. Jest to prawdopodobnie spowodowane nierównowagą między klasami, gdzie model może być mocno przesunięty w stronę przewidywania negatywów, co sztucznie podnosi precyzję. Wskazuje to, że wydajność modelu jest nieco gorsza niż losowe zgadywanie, szczególnie ze względu na jego niezdolność do prawidłowej identyfikacji pozytywnych przypadków F_p. Przekłada się to na wyniki uzyskane w działaniu rozszerzonej inteligencji — na danych bez modelu pojazdu ani algorytm MLP, ani SVM nie wskazały negatywnego rezultatu testu jako fałszywego, spowodowanego anomaliami w środowisku.

Wyniki uzyskane przez algorytm LGB na danych bez modelu pojazdu (AUC = 0.9996, PRE=0.4277, MCC=0.4760, BA=0.7649) wskazują, że model ma bardzo dobrą zdolność rozróżniania klas (bardzo wysokie AUC), ale ma poważne problemy z przewidywaniem poprawnych przypadków pozytywnych (precyzja). Mimo umiarkowanego MCC i stosunkowo wysokiej wartości BA, precyzja wynosząca 0.4277 jest krytyczna i sugeruje, że model nie generuje poprawnie przewidywań pozytywnych — większość przewidywań pozytywnych wyników jest błędna. To nie przełożyło się na rezultat działania rozszerzonej inteligencji, ponieważ jednym z jego kroków jest weryfikacja czy wskazane rezultaty są rzeczywiście negatywne. Z kolei wyniki uzyskane przez CatB+ dla danych z modelem pojazdu (AUC=0.4943 PRE= 0 MCC=-0.5521 BA=0.4942) wskazują, że model jest nieefektywny i może nawet działać szkodliwie, przewidując odwrotnie do rzeczywistych wyników.

Algorytmy muszą być szczególnie dobrze dopasowane do specyfiki danych testowych, aby zachować wysoką skuteczność, co zostało osiągnięte w największym stopniu przez algorytm XGB+. Uwzględnienie modelu pojazdu znacząco poprawiło skuteczność klasyfikacji, co podkreśla istotność dodatkowych cech w zadaniach związanych z analizą danych. Algorytmy trenowane na danych ze środowiska wykorzystującego model pojazdu wykazały lepszą skuteczność, co sugeruje, że integracja dodatkowych cech może prowadzić do bardziej precyzyjnych modeli predykcyjnych. Porównanie wyników obydwu eksperymentów wskazuje na to, że dane treningowe powinny zostać zebrane podczas wykonywania przypadków testowych dotyczących różnych funkcjonalności testowanego systemu wbudowanego.

6. Podsumowanie

Przedstawiona praca jest wynikiem badań autorki związanych z opracowaniem i weryfikacją metodyki tworzenia zautomatyzowanych testów systemów wbudowanych w pojazdach samochodowych, umożliwiających weryfikację poprawności wyników z wykorzystaniem algorytmów rozszerzonej inteligencji.

Tematyka tych badań zainspirowana została potrzebą optymalizacji procesu testowania. Potrzeba ta wynika z dążenia do zmiany podejścia do zarządzania projektami na zwinne, co skutkuje skróceniem czasu dostępnego dla tego etapu projektowania. Jednocześnie konieczne jest zachowanie jakości oraz przestrzeganie wszystkich norm i standardów wymaganych przez przemysł motoryzacyjny. W dobie rozwoju sztucznej inteligencji pojawiła się potrzeba identyfikacji możliwości rozszerzenia istniejących procesów o implementację rozwiązań proponowanych przez tę dziedzinę. Tematyka rozprawy wypływa więc ściśle z potrzeby rozwiązania praktycznych problemów pojawiających się na etapie testowania systemów wbudowanych w przemyśle samochodowym.

Głównym osiągnięciem autorki było włączenie do zaproponowanej metodyki modelu pojazdu, dzięki czemu możliwe jest uwzględnienie dynamiki ruchu pojazdu w testowaniu systemów wbudowanych. Jest to niezwykle istotne ze względu na to, że pojazd jako całość ewoluował z obiektu mechanicznego do mechatronicznego, sterowanego przez rozproszone systemy wbudowane, coraz częściej mające wpływ na zdrowie i bezpieczeństwo uczestników ruchu drogowego.

Podstawą do działań o charakterze teoretycznym była wnikliwa analiza norm i standardów obowiązujących projektantów i producentów systemów wbudowanych w przemyśle samochodowym oraz możliwości, jakie daje inżynieria oparta na modelach. Rozważania nad siłami mechanicznymi działającymi na pojazd samochodowy, ich wpływem na wielkości fizyczne związane z elektrycznym układem napędowym doprowadziły do opracowania modelu symulacyjnego pojazdu, który jako element środowiska testowego służy podniesieniu jakości i niezawodności procesu testowania systemów wbudowanych. Przeprowadzona analiza literatury związanej z testowaniem, w szczególności systemów wbudowanych dała podstawy zaproponowanej metodyce. Rozwiązania związane z automatyzacją testów, opisane w dostępnej literaturze bazują na tworzeniu skryptów testowych i ograniczają się do testowania w środowiskach SIL i PIL. Zastosowanie uczenia maszynowego wspomniane jest w kontekście generowania automatycznych przypadków testowych, jednak pozostałe czynności związane z testowaniem oprogramowania wbudowanego takie jak np. implementacja środowiska testowe, utrzymywanie testów czy analiza rezultatów są w literaturze wspomniane marginalnie ze względu na ograniczoną dostępność badań w tych obszarach.

Przeprowadzone liczne eksperymenty badawcze w celu zgłębienia zjawiska migotania rezultatów testów systemów wbudowanych. Efektem końcowym przeprowadzonych badań jest zapis formalny metodyki testowania systemów wbudowanych, zawierającej innowacyjne elementy takie jak model pojazdu jako element środowiska testowego czy walidacja uzyskanych rezultatów testów z zastosowaniem algorytmów rozszerzonej inteligencji. Zastosowanie zaproponowanej metodyki w procesie testowania systemów wbudowanych w pojazdach samochodowych może przyczynić się do optymalizacji całego procesu, umożliwić zwinne zarządzanie projektami, a przede wszystkim poprawić jakość i niezawodność procesu testowania, a co za tym idzie bezpieczeństwo uczestników ruchu drogowego.

W celu weryfikacji i wykazania przydatności praktycznej opracowanej metodyki autorka

opracowała plan badań weryfikacyjnych obejmujący wykorzystanie dwóch środowisk testowych – środowiska pierwotnego oraz skonfigurowanego z wykorzystaniem zaproponowanego modelu pojazdu. Zaplanowano eksperymenty sprawdzające efektywność działania algorytmów rozszerzonej inteligencji w zakresie weryfikacji rezultatów testów. Obejmowały one badania porównawcze dla dwóch zaimplementowanych sieci neuronowych: zawierającej warstwę GRU oraz LSTM, a także dla ogólnodostępnych algorytmów wykrywania anomalii bazujących na algorytmach sztucznej inteligencji [170]. Następnie przeprowadzono zaplanowane eksperymenty oraz dokonano szczegółowej analizy i oceny uzyskanych wyników.

Szczególny nacisk położono na praktyczne zastosowanie przeprowadzonych badań. Ze względu na to, że badania realizowane były w warunkach rzeczywistego projektu, powodowało to nie tylko konieczność merytorycznego przygotowania zakresu i planu badań, ale także ogromnej ilości pracy organizacyjnej, dostosowania harmonogramu eksperymentów do terminarza projektu.

6.1 Wnioski końcowe

Wyniki badań, potwierdzają, że zaproponowana przez autorkę metodyka tworzenia zautomatyzowanych testów systemów wbudowanych w pojazdach samochodowych umożliwia weryfikację poprawności wyników testów z wykorzystaniem algorytmów rozszerzonej inteligencji. Potwierdzono, że zastosowanie modelu pojazdu jako modularnego oraz konfigurowalnego elementu środowiska testowego pozytywnie wpływa na jakość procesu testowania, zwiększając jego niezawodność i wiarygodność. Na podstawie wyników pracy można wyciągnąć następujące wnioski szczegółowe:

- Zastosowanie modelu pojazdu jako elementu środowiska testowego poprawia niezawodność i wiarygodność procesu testowania. Środowisko testowe, a w szczególności wielkości dynamiczne związane z ruchem pojazdu są kontrolowane, stabilne i odpowiadające rzeczywistej sytuacji na drodze.
- Model numeryczny pojazdu zastosowany jako element symulujący pojazd w kontekście testowanego systemu wbudowanego wpływa pozytywnie na zjawisko migotania rezultatów testów, czego efektem są bardziej powtarzalne i wiarygodne wyniki.
- Zaproponowana innowacyjna metodyka umożliwia integrację zaawansowanych technologii sztucznej inteligencji w procesie testowania systemów wbudowanych w przemyśle samochodowym.
- Zdefiniowana metodyka jest wydajna i skalowalna, co umożliwia wdrożenie jej w różnorodnych środowiskach produkcyjnych.
- Potwierdzono, że zaproponowane dwa modele sieci neuronowych skutecznie wskazywały anomalie występujące w środowisku testowym, zaburzające wyniki testów.
- Zidentyfikowano konieczność uczestnictwa specjalisty z dziedziny sztucznej inteligencji w procesie weryfikacji wyników testów w przypadku zaistnienia potrzeby modyfikacji lub dostosowania zaproponowanych modeli sieci neuronowych do specyficznych potrzeb testowanych systemów wbudowanych.
- Zweryfikowano, że istnieją ogólnodostępne algorytmy sztucznej inteligencji zaimplementowane w języku Python (np. biblioteka PyOD), wystarczająco zaawansowane i skuteczne, aby wspierać proces weryfikacji rezultatów testów automatycznych w ramach rozszerzonej inteligencji.
- Potwierdzono, że zastosowanie opracowanej metodyki w rzeczywistych projektach wpływa pozytywnie na jakość procesu testowania oraz umożliwia efektywną analizę negatywnych rezultatów testów z zastosowaniem rozszerzonej inteligencji, argumentując za zasadnością implementacji proponowanych rozwiązań w przedsiębiorstwie.

6.2 Kierunki dalszych prac

Podjęta przez autorkę próba opracowania innowacyjnej metodyki tworzenia automatycznych testów systemów wbudowanych umożliwiająca zastosowanie algorytmów rozszerzonej inteligencji do weryfikacji wyników testów pozwoliła zidentyfikować wiele ciekawych obszarów, w których celowe jest przeprowadzenie dodatkowych badań.

Istotnym ze względu na praktyczne zastosowanie jest temat zoptymalizowania procesu etykietowania danych wykorzystywanych w procesie trenowania algorytmów rozszerzonej inteligencji. Interesującym podejściem, wartym przebadania jest zastosowanie algorytmów półnadzorowanych jako etapu pośredniego w zadaniu etykietowania danych. W przypadku systemów wbudowanych często dysponuje się dużą ilością nieetykietowanych danych, natomiast dane oznaczone, związane z anomaliami są kosztowne w przygotowaniu. Automatyczne etykietowanie z użyciem półnadzorowanego podejścia pozwoliłoby na dynamiczne aktualizowanie modeli wykrywania anomalii, co mogłoby zwiększyć ich precyzję i skuteczność w długim okresie.

Kolejne prace będą dotyczyły definiowania zadania optymalizacji przetwarzania danych zgromadzonych ze środowiska testowego w procesie testowania systemów wbudowanych. Badania będą się koncentrowały na skutecznych sposobach identyfikacji kluczowych cech zebranych podczas testów, które mają największy wpływ na wykrywanie anomalii. Proces ten ukierunkowany na rozwój metod automatycznego wyboru i ważenia cech w celu zmniejszenia wymiarowości danych poprawiłby wydajność zaproponowanych algorytmów. Warto również opracować techniki usuwania szumów i zakłóceń z danych zbieranych w środowisku testowym, co miałoby na celu poprawę jakości danych.

Multidyscyplinarne podejście do rozwiązywania skomplikowanych wyzwań, zwieńczone sukcesem badań prowadzonych w ramach niniejszej pracy, wskazują na potencjalne inicjatywy badawcze, które mogą obejmować takie dziedziny jak: inżynieria mechaniczna, elektrotechnika, informatyka oraz sztuczna inteligencja. W odniesieniu do inżynierii mechanicznej ciekawym kierunkiem rozwoju zaproponowanych w dysertacji rozwiązań jest integracja różnorodnych technik, co może dodatkowo zoptymalizować przetwarzanie danych w czasie rzeczywistym. Wpływ tej rozprawy rozciąga się na różne dyscypliny naukowe, stanowiąc cenne źródło inspiracji dla przyszłych badań.

6.3 Implementacja w przemyśle

Badania prowadzone przez autorkę w ramach dysertacji oparte były o rzeczywiste projekty systemów wbudowanych projektowanych i produkowanych w firmie Dräxlmaier. Spostrzeżenia i wnioski z uzyskanych rezultatów były implementowane na bieżąco we wspomnianym projekcie.

Zaproponowany modelu pojazdu, zaimplementowany w środowisku testowym był z powodzeniem stosowany do reprodukcji defektów zgłaszanych przez klienta, zidentyfikowanych przez niego podczas testów w pojeździe. Dzięki integracji modelu numerycznego pojazdu ze środowiskiem testowym jest możliwe uwzględnienie takich aspektów jak dynamika sił działających na elementy pojazdu w procesie testowania systemów wbudowanych. Wpłynęło to na podniesienie jakości i niezawodności procesu testowania poprzez odzwierciedlenie pełni interakcji między systemami wbudowanymi a dynamiką pojazdu w ocenie poszczególnych funkcjonalności. Podczas prac nad dysertacją powstały elementy środowiska testowego ułatwiające testerom korzystanie z zaimplementowanego modelu - graficzny panel oraz skrypty umożliwiające automatyczną interakcję z modelem pojazdu z poziomu przypadku testowego.

Ponadto ze względu na duże zainteresowanie firmy zaangażowaniem sztucznej inteligencji w procesy projektowania oraz testowania systemów wbudowanych rezultaty uzyskane w ramach dysertacji zostaną uwzględnione podczas prac nad aktualizacją i modernizacją procesów obowiązujących w firmie, a także wprowadzeniem innowacji w codziennej pracy projektantów,

programistów oraz testerów. Przygotowana została koncepcja przetwarzania danych zebranych ze środowiska testowego i przygotowania ich do wykorzystania przez algorytmy rozszerzonej inteligencji w procesie weryfikacji rezultatów testów. Następnie na tej podstawie zaimplementowano modułowe, łatwe do modyfikacji i adaptacji w innych projektach skrypty. Opracowana została również biblioteka algorytmów rozszerzonej inteligencji weryfikujących wyniki testów, zaimplementowana w języku Python, co daje możliwość efektywnego nadzorowania i weryfikacji wyników testów przez inżynierów.

Prace przedstawione w dysertacji umożliwiły pokazanie potencjału multidyscyplinarnych rozwiązań ze szczególnym uwzględnieniem obszarów inżynierii mechanicznej, tj. zastosowania numerycznego modelu pojazdu oraz algorytmów rozszerzonej inteligencji w procesie rozwoju, a przede wszystkim testowania systemów wbudowanych. Daje to gotowość do zastosowania w świecie rzeczywistym, ale również zapewniło silną podstawę dla wdrożenia nowej technologii. Wysiłki badawczo-rozwojowe odegrały kluczową rolę w wypełnieniu luki między badaniami akademickimi a wdrożeniem przemysłowym.

Bibliografia

[1] K. Shahzad and I. Iqbal Cheema, "Low-carbon technologies in automotive industry and decarbonizing transport," *Journal of Power Sources*, vol. 591, 2024.

[2] M. Rísquez Ramos and M. E. Ruiz-Gálvez, "The transformation of the automotive industry toward electrification and its impact on global value chains: Inter-plant competition, employment, and supply chains," *European Research on Management and Business Economics*, vol. 30, no. 1, 2024.

[3] S. Siegl, K.-S. Hielscher, R. German, and C. Berger, "Automated testing of embedded automotive systems from requirement specification models," in *2011 12th Latin American Test Workshop (LATW)*, pp. 1–6, 2011.

[4] F. E. Alarcón, A. M. Cawley, and E. Sauma, "Electric mobility toward sustainable cities and road-freight logistics: A systematic review and future research directions," *Journal of Cleaner Production*, vol. 430, 2023.

[5] C. Llopis-Albert, F. Rubio, and F. Valero, "Impact of digital transformation on the automotive industry," *Technological Forecasting and Social Change*, vol. 162, 2021.

[6] P. Salvini, L. Kunze, and M. Jirotka, "On self-driving cars and its (broken?) promises. a case study analysis of the german act on autonomous driving," *Technology in Society*, vol. 78, 2024.

[7] H. Martin, Z. Ma, C. Schmittner, B. Winkler, M. Krammer, D. Schneider, T. Amorim, G. Macher, and C. Kreiner, "Combined automotive safety and security pattern engineering approach," *Reliability Engineering System Safety*, vol. 198, 2020.

[8] "ISO 26262:2018 Road vehicles, Functional Safety," standard, International Organization for Standardization, 2018.

[9] "Automotive Spice version 3.1 (ASPICE): Process reference model and process assessment," standard, VDA QMC Working Group 13 / Automotive SIG, 2017.

[10] G. Xie, Y. Li, Y. Han, Y. Xie, G. Zeng, and R. Li, "Recent advances and future trends for automotive functional safety design methodologies," *IEEE Transactions on Industrial Informatics*, vol. 16, no. 9, pp. 5629–5642, 2020.

[11] D. Kum, J. Son, S. Lee, and I. Wilson, "Automated testing for automotive embedded systems," *2006 SICE-ICASE International Joint Conference*, pp. 4414–4418, 2006.

[12] J. Gumiel, J. Mabe, J. Jiménez, and J. Barruetabeña, "Introducing the electronic knowledge framework into the traditional automotive suppliers' industry: From mechanical engineering to mechatronics," *Businesses*, vol. 2, no. 2, pp. 273–289, 2022.

[13] Q. Plantec, M.-A. Deval, S. Hooge, and B. Weil, "Big data as an exploration trigger or problem-solving patch: Design and integration of ai-embedded systems in the automotive industry," *Technovation*, vol. 124, 2023.

[14] M. Zöldy, "Investigation of autonomous vehicles fit into traditional type approval process," *Proceedings of ICCTE*, pp. 517–521, 2018.

[15] H. Martins, "Overview of type approval homologation and self-certification," *Ford Motor Company, Dearborn, MI, USA, Tech. Rep*, 2010.

[16] Z. Szalay, "Next generation x-in-the-loop validation methodology for automated vehicle systems," *IEEE Access*, vol. 9, pp. 35616–35632, 2021.

[17] P. Junietz, W. Wachenfeld, K. Klonecki, and H. Winner, "Evaluation of different approaches to address safety validation of automated driving," in *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, pp. 491–496, IEEE, 2018.

[18] M. Zöldy, "Legal barriers of utilization of autonomous vehicles as part of green mobility," in *Proceedings of the 4th International Congress of Automotive and Transport Engineering (AMMA 2018) IV*, pp. 243–248, Springer, 2019.

[19] T. Bécsi, S. Aradi, and P. Gáspár, "Security issues and vulnerabilities in connected car systems," in *2015 International Conference on Models and Technologies for Intelligent Transportation Systems (MT-ITS)*, pp. 477–482, IEEE, 2015.

[20] Á. Török, Z. Szalay, and B. Sághi, "New aspects of integrity levels in automotive industry-cybersecurity of automated vehicles," *IEEE transactions on intelligent transportation systems*, vol. 23, no. 1, pp. 383–391, 2020.

[21] Á. Török and Z. Pethő, "Introducing safety and security co-engineering related research orientations in the field of automotive security," *Periodica Polytechnica Transportation Engineering*, vol. 48, no. 4, pp. 349–356, 2020.

[22] M. Zöldy, Z. Szalay, and V. Tihanyi, "Challenges in homologation process of vehicles with artificial intelligence," *Transport*, vol. 35, no. 4, pp. 447–453, 2020.

[23] P. Skruch, M. Panek, and B. Kowalczyk, "Model-based testing in embedded automotive systems," in *Model-Based Testing for Embedded Systems* (J. Zander, I. Schieferdecker, and P. J. Mosterman, eds.), Computational Analysis, Synthesis, Design Dynamic Systems, CRC Press, 2011.

[24] A. Gaafar, "Najdroższy bug w historii? przez błąd w kodzie rozpadła się rakieta." https://www.dobreprogramy.pl/najdrozszy-bug-w-historii-przez-blad-w-kodzie-rozpadla-sie-rakieta,6676587891968832a. Accessed: 2024-09-20.

[25] K. Byzdra, "Kolejne problemy techniczne w teslach. mogą stanowić śmiertelne niebezpieczeństwo." https://pch24.pl/blad-w-oprogramowaniu-przyczyna-wypadkow-tesla-wycofuje-363-tys-samochodow/. Accessed: 2024-09-20.

[26] D. Długosz, "Boeing 737 max z nowymi błędami w oprogramowaniu. "może prowadzić do utraty kontroli"." https://www.komputerswiat.pl/aktualnosci/nauka-i-technika/boeing-737-max-z-nowymi-bledami-w-oprogramowaniu-moze-prowadzic-do-utraty-kontroli/xsyt9km. Accessed: 2024-09-20.

[27] M. Łubiński, "Duże akcje serwisowe. kolejne auta wzywane do aso." https://www.wyborkierowcow.pl/duze-akcje-serwisowe-kolejne-auta-wzywane-do-aso/. Accessed: 2024-09-20.

[28] S. Kochanthara, N. Rood, A. K. Saberi, L. Cleophas, Y. Dajsuren, and M. van den Brand, "A functional safety assessment method for cooperative automotive architecture," *Journal of Systems and Software*, vol. 179, 2021.

[29] A. Nadeem, M. Aidong, and S. Myrna, "Applications of hardware-in-the-loop simulation in automotive embedded systems," *SAE Technical Paper*, 2020.

[30] V. Garousi, M. Felderer, Çağrı Murat Karapıçak, and U. Yılmaz, "Testing embedded software: A survey of the literature," *Information and Software Technology*, vol. 104, pp. 14–45, 2018.

[31] A. Axelrod, *Automatyzacja testów. Kompletny przewodnik dla testerów oprogramowania.* Warszawa: PWN, 2019.

[32] A. M. Madni, "Exploiting augmented intelligence in systems engineering and engineered systems," *INSIGHT*, vol. 23, pp. 31–36, 03 2020.

[33] S. M. Ammal, M. Kathiresh, and R. Neelaveni, *Artificial Intelligence and Sensor Technology in the Automotive Industry: An Overview*, pp. 145–164. Springer International Publishing, 2021.

[34] T. H. Selim and M. Gad-El-Rab, *Artificial Intelligence and the Global Automotive Industry*, pp. 31–53. Springer Nature Switzerland, 2024.

[35] R. Lima, A. M. R. da Cruz, and J. Ribeiro, "Artificial intelligence applied to software testing: A literature review," in *2020 15th Iberian Conference on Information Systems and Technologies (CISTI)*, pp. 1–6, 2020.

[36] H. Hourani, A. Hammad, and M. Lafi, "The impact of artificial intelligence on software testing," in *2019 IEEE Jordan International Joint Conference on Electrical Engineering and Information Technology (JEEIT)*, pp. 565–570, 2019.

[37] N. Navet and F. Simonot-Lion, *Automotive Embedded Systems Handbook.* CRC Press, 2009.

[38] K. N. Hodel, J. Reinaldo Da Silva, L. R. Yoshioka, J. F. Justo, and M. M. D. Santos, "Fataes: Systematic methodology of functional testing for automotive embedded software," *IEEE Access*, vol. 10, pp. 74259–74279, 2022.

[39] J. Á. Gumiel, *Improving the Reliability of Automotive Systems*, pp. 151–195. Cham: Springer International Publishing, 2024.

[40] A. Buczacki and P. Piątek, "Proposal for an integrated framework for electronic control unit design in the automotive industry," *Energies*, vol. 14, no. 13, 2021.

[41] E. Dustin, *Efective Software Testing. 50 Ways to Improve Your Testing.* Addisson-Wesley, 2002.

[42] M. Bajer, M. Szlagor, and M. Wrzesniak, "Embedded software testing in research environment. a practical guide for non-experts," in *2015 4th Mediterranean Conference on Embedded Computing (MECO)*, pp. 100–105, 2015.

[43] P. Duvall, S. M. Matyas, and A. Glover, "Continuous integration: Improving software quality and reducing risk (the addison-wesley signature series)," 2007.

[44] S. Bhunia and M. Tehranipoor, "Chapter 2 - a quick overview of electronic hardware," in *Hardware Security* (S. Bhunia and M. Tehranipoor, eds.), pp. 23–45, Morgan Kaufmann, 2019.

[45] L. Rutkowski, *Metody i techniki sztucznej inteligencji*. Warszawa: PWN, 2021.

[46] A. Hebbar, "Augmented intelligence: Enhancing human capabilities," in *2017 Third International Conference on Research in Computational Intelligence and Communication Networks (ICRCICN)*, pp. 251–254, 2017.

[47] C. Aggarwal, *Outlier Analysis*. 11 2013.

[48] B. Broekman and E. Notenboom, *Testing Embedded Software*. Pearson Education, 2003.

[49] "IEEE Std 1044:2009 Standard Classification for Software Anomalies," standard, International Organization for Standardization, 2009.

[50] A. Albatayneh, A. Juaidi, M. Jaradat, and F. Manzano-Agugliaro, "Future of electric and hydrogen cars and trucks: An overview," *Energies*, vol. 16, no. 7, 2023.

[51] F. Assadian, "Mechatronics and its role in the automotive domain," in *UKACC International Conference on Control 2010*, pp. 1–5, 2010.

[52] H.-P. Schöner, "Automotive mechatronics," *Control Engineering Practice*, vol. 12, no. 11, pp. 1343–1351, 2004. Mechatronic Systems.

[53] M. Hiller, R. Bardini, T. Bertram, M. Torlo, and D. Ward, "Mechatronic design in automotive systems," *IFAC Proceedings Volumes*, vol. 33, no. 26, pp. 89–94, 2000. IFAC Conference on Mechatronic Systems, Darmstadt, Germany, 18-20 September 2000.

[54] "What is electronic control unit definition?." https://www.autopi.io/blog/what-is-electronic-control-unit-definition/. Accessed: 2024-07-21.

[55] S. Thomas, "Automotive ECU an inevitable part of the automobiles?." https://www.einfochips.com/blog/automotive-ecu-an-inevitable-part-of-the-automobiles/. Accessed: 2024-07-21.

[56] "Benz Patent Motor Car, 1886 - 1894." https://mercedes-benz-publicarchive.com/marsClassic/en/instance/ko/Benz-Patent-Motor-Car-1886—1894.xhtml?oid=4373. Accessed: 2024-05-02.

[57] L. Tony, "Nov. 3, 1900: The Grandmother of All Auto Shows." https://www.wired.com/2010/11/1103first-us-car-show-new-york/l. Accessed: 2024-05-02.

[58] "The model T is Ford's universal car that put the word on wheels.." https://corporate.ford.com/articles/history/the-model-t.html. Accessed: 2024-05-02.

[59] "How Do Gasoline Cars Work?." https://afdc.energy.gov/vehicles/how-do-gasoline-cars-work. Accessed: 2024-05-03.

[60] W. Dieterle, "Mechatronic systems: Automotive applications and modern design methodologies," *Annual Reviews in Control*, vol. 29, no. 2, pp. 273–277, 2005.

[61] R. Farooqi, B. Snyder, and S. Anwar, "Real time monitoring of diesel engine injector waveforms for accurate fuel metering and control," *Journal of Control Science and Engineering*, vol. 2013, 01 2013.

[62] H. Fahami, H. Zamzuri, S. Mazlan, and N. Zulkarnain, "The design of vehicle active front steering based on steer by wire system," *Advanced Science Letters*, vol. 19, pp. 61–65, 01 2013.

[63] W. Rieger, "Active steering for active safety," *ATZ/MTZ Special Edition*, vol. 105, 2003.

[64] "Co stanie się z samochodami z silnikiem spalinowym po 2025 roku?." https://knaufautomotive.com/pl/co-stanie-sie-z-samochodami-z-silnikiem-spalinowym-po-2025-roku. Accessed: 2024-05-02.

[65] "How Do All-Electric Cars Work?." https://afdc.energy.gov/vehicles/how-do-all-electric-cars-work. Accessed: 2024-05-03.

[66] P. Nyberg, E. Frisk, and L. Nielsen, "Generation of equivalent driving cycles using markov chains and mean tractive force components," *IFAC Proceedings Volumes*, vol. 47, no. 3, pp. 8787–8792, 2014. 19th IFAC World Congress.

[67] "How Do Hybrid Electric Cars Work?." https://afdc.energy.gov/vehicles/how-do-hybrid-electric-cars-work. Accessed: 2024-05-03.

[68] "How Do Plug-In Hybrid Electric Cars Work?." https://afdc.energy.gov/vehicles/how-do-plug-in-hybrid-electric-cars-work. Accessed: 2024-05-03.

[69] H. Leffler and M. Schnabel, "Automotive application of mechatronic systems: State of the art and future prospects," *IFAC Proceedings Volumes*, vol. 39, no. 16, p. 6, 2006. 4th IFAC Symposium on Mechatronic Systems.

[70] "How Do Fuel Cell Electric Vehicles Work Using Hydrogen." https://afdc.energy.gov/vehicles/how-do-fuel-cell-electric-cars-work. Accessed: 2024-05-03.

[71] A. Bień, J. Rzeszutko, and T. Szymański, "Embedded systems dedicated to measurement and control in the automotive industry. the design process and the initial diagnosis," *Pomiary Automatyka Kontrola*, pp. 655–659, 2014.

[72] A. Gnacy-Gajdzik and P. Przystałka, "Ewolucja testowania systemów wbudowanych w motoryzacji wraz z wprowadzeniem nomy ISO 21448:2019 Pojazdy drogowe - Bezpieczeństwo zamierzonej funkcjonalności. ," *Utrzymanie Ruchu*, pp. 7–16, 2021.

[73] "AUTOSAR: AUTOMOTIVE OPEN SYSTEM ARCHITECTURE," standard, 2023.

[74] "AEC," standard, Automotive Electronics Council, 2023.

[75] "SAE OBD-II," standard, SAE International, 2018.

[76] "MISRA C," standard, Motor Industry Software Reliability Association, 2023.

[77] I. Harris, "Chapter 22 - embedded software for automotive applications," in *Software Engineering for Embedded Systems* (R. Oshana and M. Kraeling, eds.), pp. 767–816, Oxford: Newnes, 2013.

[78] P. Kafka, "The Automotive Standard ISO 26262, the Innovative Driver for Enhanced Safety Assessment Technology for Motor Cars," *Procedia Engineering*, vol. 45, pp. 2–10, 2012. 2012 International Symposium on Safety Science and Technology.

[79] "SAE J3016: Taxonomy and Definitions for Terms Related to Driving Automation Systems for On-Road Motor Vehicles," standard, SAE International, 2018.

[80] "ISO 21448:2022 Safety of intended functionality," standard, International Organization for Standardization, 2022.

[81] S. Riedmaier, D. Schneider, D. Watzenig, F. Diermeyer, and B. Schick, "Model validation and scenario selection for virtual-based homologation of automated vehicles," *Applied Sciences*, vol. 11, no. 1, 2021.

[82] M. Hirz, "An approach supporting integrated modeling and design of complex mechatronics products by the example of automotive applications," 07 2018.

[83] A. Gnacy-Gajdzik and P. Przystałka, "Identyfikacja anomalii wywołujących fałszywie negatywne wyniki w procesie testowania oprogramowania wbudowanego," in *Projektowanie, budowa i eksploatacja maszyn*, vol. 558, pp. 61–72, 2021.

[84] Y. Pei, S. Habchi, R. Rwemalika, J. Sohn, and M. Papadakis, "An empirical study of async wait flakiness in front-end testing," in *BElgian-NEtherlands software eVOLution symposium*, 2022.

[85] Q. Luo, F. Hariri, L. Eloussi, and D. Marinov, "An empirical analysis of flaky tests," in *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, FSE 2014, (New York, NY, USA), p. 643–653, Association for Computing Machinery, 2014.

[86] Z. Gao, Y. Liang, M. B. Cohen, A. M. Memon, and Z. Wang, "Making system user interactive tests repeatable: When and what should we control?," in *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, vol. 1, pp. 55–65, 2015.

[87] M. Gruber, S. Lukasczyk, F. Kroiß, and G. Fraser, "An empirical study of flaky tests in python," in *2021 14th IEEE Conference on Software Testing, Verification and Validation (ICST)*, pp. 148–158, 2021.

[88] O. Parry, G. M. Kapfhammer, M. Hilton, and P. McMinn, "What do developer-repaired flaky tests tell us about the effectiveness of automated flaky test detection?," in *Proceedings of the 3rd ACM/IEEE International Conference on Automation of Software Test*, AST '22, (New York, NY, USA), p. 160–164, Association for Computing Machinery, 2022.

[89] J. Morán, C. Augusto, A. Bertolino, C. D. L. Riva, and J. Tuya, "Flakyloc: Flakiness localization for reliable test suites in web applications," *Journal of Web Engineering*, vol. 19, p. 267–296., Jun. 2020.

[90] J. Awrejcewicz, *Matematyczne modelowanie systemów.* WNT, Fundacja "Książka Naukowo-Techniczna", 2007.

[91] G. Liebel, N. Marko, M. Tichy, A. Leitner, and J. Hansson, "Model-based engineering in the embedded systems domain: an industrial survey on the state-of-practice," *Software Systems Modeling*, 2018.

[92] A. Shaout and S. Pattela, "Model based approach for automotive embedded systems," in *2021 22nd International Arab Conference on Information Technology (ACIT)*, pp. 1–7, 2021.

[93] M. Broy, S. Kirstan, H. Krcmar, and B. Schätz, *Evolution and Maintenance of Software Models*, ch. What is the Benefit of a Model-Based Design of Embedded Software Systems in the Car Industry?, pp. 343–369. IGI Global, 2012.

[94] T. Basten, M. Hendriks, N. Trčka, L. Somers, M. Geilen, Y. Yang, G. Igna, S. de Smet, M. Voorhoeve, W. van der Aalst, H. Corporaal, and F. Vaandrager, *Model-Driven Design-Space Exploration for Software-Intensive Embedded Systems*, pp. 189–244. New York, NY: Springer New York, 2013.

[95] F. Paternò, *Model-Based Approaches*, pp. 11–30. London: Springer London, 2000.

[96] J. Zander, "Model-based testing of real-time embedded systems in the automotive domain," *Fraunhofer FOKUS*, 01 2008.

[97] J. Zander, I. Schieferdecker, and P. Mosterman, *Model-Based Testing for Embedded Systems*. CRC Press, 09 2011.

[98] S. Arczyński, *Mechanika ruchu samochodu*. WNT, 1994.

[99] M. Mitschke, *Dynamika samochodu*. WKL, 1997.

[100] L. Prochowski, *Pojazdy samochodowe. Mechanika ruchu*. WKŁ, 2005.

[101] G. Sieklucki, A. Bień, J. Gromba, and S. Sobieraj, "Określenie parametrów ruchu pojazdu na podstawie różnych metod przetwarzania sygnałów przyspieszenia," *Zeszyty Naukowe Wydziału ELektrotechniki i Automatyki Politechniki Gdańskiej*, 2017.

[102] W. Siłka, *Teoria ruchu samochodu*. WNT, 2002.

[103] M. Targosz, *Optymalizacja energochłonnosci całkowitej pojazdu elektrycznego*. Rozprawa doktorska, Politechnika Śląska Wydział Mechaniczny Technologiczny Instytut Podstaw Konstrukcji Maszyn, 2014.

[104] S. P. Karmore and A. R. Mahajan, "New approach for testing and providing security mechanism for embedded systems," *Procedia Computer Science*, vol. 78, pp. 851–858, 2016.

[105] T. Placho, C. Schmittner, A. Bonitz, and O. Wana, "Management of automotive software updates," *Microprocess. Microsystems*, vol. 78, p. 103257, 2020.

[106] A. Krupp and W. Müller, "A systematic approach to the test of combined hw/sw systems," *2010 Design, Automation & Test in Europe Conference & Exhibition (DATE 2010)*, pp. 323–326, 2010.

[107] A. Pellacani, P. Kicman, and M. Suatoni, "Design, development, validation and verification of gnc technologies," in *2019 8TH EUROPEAN CONFERENCE FOR AERONAUTICS AND SPACE SCIENCES (EUCASS)*, 2019.

[108] A. Vidanapathirana, S. D. Dewasurendra, and S. G. Abeyaratne, "Model in the loop testing of complex reactive systems," in *2013 IEEE 8th International Conference on Industrial and Information Systems*, pp. 30–35, 2013.

[109] A. Plummer, "Model-in-the-loop testing," *Proceedings of The Institution of Mechanical Engineers Part I-journal of Systems and Control Engineering - PROC INST MECH ENG I-J SYST C*, vol. 220, pp. 183–199, 05 2006.

[110] J. Sooyong, K. Yongsub, and W. Lee, "Software-in-the-loop simulation for early-stage testing of autosar software component," in *Eighth International Conference on Ubiquitous and Future Networks, ICUFN 2016, Vienna, Austria, July 5-8, 2016*, pp. 59–63, IEEE, 2016.

[111] J. Mina, Z. Flores, E. López, A. Pérez, and J.-H. Calleja, "Processor-in-the-loop and hardware-in-the-loop simulation of electric systems based in fpga," in *2016 13th International Conference on Power Electronics (CIEP)*, pp. 172–177, 2016.

[112] A. Joshi, "Hardware-in-the-Loop (HIL) Implementation and Validation of SAE Level 2 Automated Vehicle with Subsystem Fault Tolerant Fallback Performance for Takeover Scenarios," *SAE Int*, pp. 13–32, 2018.

[113] P.Nikończuk and S. Jaszczak, "Stanowisko do testowania układów sterowania temperaturą w kabinie lakierniczej w trybie hardware in the loop," *Autobusy – Technika, Eksploatacja, Systemy Transportowe*, vol. 12, pp. 1244–1247, 2016.

[114] S. Raikwar, L. Jijyabhau Wani, S. Arun Kumar, and M. Sreenivasulu Rao, "Hardware-in-the-loop test automation of embedded systems for agricultural tractors," *Measurement*, vol. 133, pp. 271–280, 2019.

[115] T. Tettamanti, M. Szalai, S. Vass, and V. Tihanyi, "Vehicle-in-the-loop test environment for autonomous driving with microscopic traffic simulation," in *2018 IEEE International Conference on Vehicular Electronics and Safety (ICVES)*, pp. 1–6, 2018.

[116] H. J. Vishnukumar, B. Butting, C. Müller, and E. Sax, "Machine learning and deep neural network — artificial intelligence core for lab and real-world test and validation for adas and autonomous vehicles: Ai for efficient and quality test and validation," in *2017 Intelligent Systems Conference (IntelliSys)*, pp. 714–721, 2017.

[117] M. Mizoguchi, T. Iida, and T. Irie, "Optimization of automated executions based on integration test configurations of embedded software," *IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, pp. 358–363, 2020.

[118] P. Skruch, "Wprowadzenie do testowania układów automatyki." XI International PhD Workshop, 2009.

[119] V. Garousi and M. V. Mäntylä, "When and what to automate in software testing?," *Information and Software Technology*, vol. 76, 2016.

[120] D. S. Battina, "Artificial intelligence in software test automation: A systematic literature review (december 12, 2019)," *International Journal of Emerging Technologies and Innovative Research (www.jetir.org — UGC and issn Approved)*.

[121] A. Fontes and G. Gay, "The integration of machine learning into automated test generation: A systematic mapping study," *Software Testing, Verification and Reliability*, vol. 33, no. 4, p. e1845, 2023.

[122] J. McCarthy, M. Minsky, N. Rochester, and C. Shannon, "A proposal for the dartmouth summer research project on artificial intelligence, august 31, 1955.," *AI Magazine*, vol. 27, pp. 12–14, 01 2006.

[123] M. Haenlein and A. Kaplan, "A brief history of artificial intelligence: On the past, present, and future of artificial intelligence," *California Management Review*, vol. 61, p. 000812561986492, 07 2019.

[124] S. J. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach (4th Edition)*. Pearson, 2020.

[125] Z. Michalewicz and D. Fogel, *How to Solve It: Modern Heuristics*. Springer Berlin Heidelberg, 2013.

[126] X. Ying, "An overview of overfitting and its solutions," *Journal of Physics: Conference Series*, vol. 1168, 02 2019.

[127] D. S. Watson, "On the philosophy of unsupervised learning," vol. 36, 04 2023.

[128] X. Zhu and A. B. Goldberg, *Introduction to semi-supervised learning*. Springer Nature, 2022.

[129] A. Herrmann, W. Brenner, and R. Stadler, *Autonomous Driving: How the Driverless Revolution Will Change the World*. 04 2018.

[130] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016.

[131] M. Nielsen, *Neural Networks and Deep Learning*. Determination Press, 2015.

[132] R. Hanson, *The Age of Em: Work, Love, and Life When Robots Rule the Earth*. Oxford University Press, 2016.

[133] P. Daugherty and H. Wilson, *Human + Machine: Reimagining Work in the Age of AI*. Harvard Business Review Press, 2018.

[134] H. Morris, D. Kirsch, and J. Hurwitz, *Augmented Intelligence: The Business Power of Human-Machine Collaboration*. Auerbach Publications, 2021.

[135] N. Bostrom, *Superintelligence: Paths, Dangers, Strategies*. Oxford University Press, 2014.

[136] V. Müller, "Ethics of artificial intelligence and robotics," vol. 2020, pp. 1–31, 04 2020.

[137] F. De Felice, A. Petrillo, C. De Luca, and I. Baffo, "Artificial intelligence or augmented intelligence? impact on our lives, rights and ethics," *Procedia Computer Science*, vol. 200, pp. 1846–1856, 2022. 3rd International Conference on Industry 4.0 and Smart Manufacturing.

[138] H. Hassani, E. S. Silva, S. Unger, M. TajMazinani, and S. Mac Feely, "Artificial intelligence (ai) or intelligence augmentation (ia): What is the future?," *AI*, vol. 1, no. 2, pp. 143–155, 2020.

[139] W. Ashby, *Wprowadzenie do cybernetyki*. PWN, 1963.

[140] K.-L. A. Yau, H. J. Lee, Y.-W. Chong, M. H. Ling, A. R. Syed, C. Wu, and H. G. Goh, "Augmented intelligence: Surveys of literature and expert opinion to understand relations between human intelligence and artificial intelligence," *IEEE Access*, vol. 9, pp. 136744–136761, 2021.

[141] N.-n. Zheng, Z.-y. Liu, P.-j. Ren, Y.-q. Ma, S.-t. Chen, S.-y. Yu, J.-r. Xue, B.-d. Chen, and F.-y. Wang, "Hybrid-augmented intelligence: collaboration and cognition," *Frontiers of Information Technology Electronic Engineering*, pp. 153 – 179, 2017.

[142] S. E. Cerminara, P. Cheng, L. Kostner, S. Huber, M. Kunz, J.-T. Maul, J. S. Böhm, C. F. Dettwiler, A. Geser, C. Jakopović, L. M. Stoffel, J. K. Peter, M. Levesque, A. A. Navarini, and L. V. Maul, "Diagnostic performance of augmented intelligence with 2d and 3d total body photography and convolutional neural networks in a high-risk population for melanoma under real-world conditions: A new era of skin cancer screening?," *European Journal of Cancer*, p. 112954, 2023.

[143] A. Kumar K, T. Y. Satheesha, B. B. L. Salvador, S. Mithileysh, and S. T. Ahmed, "Augmented intelligence enabled deep neural networking (audnn) framework for skin cancer classification and prediction using multi-dimensional datasets on industrial iot standards," *Microprocessors and Microsystems*, vol. 97, p. 104755, 2023.

[144] B. Cheema, R. K. Mutharasan, A. Sharma, M. Jacobs, K. Powers, S. Lehrer, F. H. Wehbe, J. Ronald, L. Pifer, J. D. Rich, K. Ghafourian, A. Tibrewala, P. McCarthy, Y. Luo, D. T. Pham, J. E. Wilcox, and F. S. Ahmad, "Augmented intelligence to identify patients with advanced heart failure in an integrated health system," *JACC: Advances*, vol. 1, no. 4, p. 100123, 2022.

[145] G. Bazoukis, J. Hall, J. Loscalzo, E. M. Antman, V. Fuster, and A. A. Armoundas, "The inclusion of augmented intelligence in medicine: A framework for successful implementation," *Cell Reports Medicine*, vol. 3, no. 1, p. 100485, 2022.

[146] J. S. Devagiri, S. Paheding, Q. Niyaz, X. Yang, and S. Smith, "Augmented reality and artificial intelligence in industry: Trends, tools, and future challenges," *Expert Systems with Applications*, vol. 207, p. 118002, 2022.

[147] Z. Khaliq, S. U. Farooq, and D. A. Khan, "Artificial intelligence in software testing : Impact, problems, challenges and prospect," 2022.

[148] L. C. Cordeiro, "Automated verification and synthesis of embedded systems using machine learning," *CoRR*, vol. abs/1702.07847, 2017.

[149] J. Bielefeldt, B. Kai-Uwe, S. Reza Khan, M. Massah, W. Hans-Werner, S. Scharoba, and M. Hübner, "Deeptest: How machine learning can improve the test of embedded systems," in *2021 10th Mediterranean Conference on Embedded Computing (MECO)*, pp. 1–6, 2021.

[150] R. P. Verma and M. R. Beg, "Generation of test cases from software requirements using natural language processing," 2013.

[151] A. Ansari, M. Shagufta, A. Fatima, and S. Tehreem, "Constructing test cases using natural language processing," 2017.

[152] K. Kikuma, T. Yamada, K. Sato, and K. Ueda, "Preparation method in automated test case generation using machine learning," pp. 393–398, 12 2019.

[153] S. R. Shahamiri, W. M. N. W. Kadir, S. Ibrahim, and S. Z. M. Hashim, "An automated framework for software test oracle," *Information and Software Technology*, vol. 53, no. 7, pp. 774–788, 2011.

[154] T. Markel, A. Brooker, T. Hendricks, V. Johnson, K. Kelly, B. Kramer, M. O'Keefe, S. Sprik, and K. Wipke, "Advisor: A systems analysis tool for advanced vehicle modeling.," *Journal of Power Sources*, p. 110:255–266, 2002.

[155] F. Milano, "Power System Analysis Toolbox ." https://eecs.wsu.edu/ ee521/Material/ 20120927/psat-20080214.pdf, 2008. Accessed: 2024-04-13.

[156] I. Ito, "Battery Electric Vehicle Model in Simscape ." https://github.com/mathworks/Simscape-Battery-Electric-Vehicle-Model/releases/tag/v2.2.2, 2024. Accessed: 2024-04-13.

[157] A. Gnacy-Gajdzik, P. Przystałka, and W. Sebzda, "Zastosowanie modelu pojazdu elektrycznego do testów układów wbudowanych," in *Metody komputerowe - 2022. Studencka konferencja naukowa*, pp. 25–28, 2022.

[158] "Aerodynamics: The best value of all current Porsche models." https://newsroom.porsche.com/en/products/taycan/aerodynamics-18554.html. Accessed: 2024-07-25.

[159] "THE BATTERY. Sophisticated thermal management, up to 800-volt system voltage." https://media.porsche.com/mediakit/taycan/en/porsche-taycan/die-batterie. Accessed: 2024-07-25.

[160] A. Gnacy-Gajdzik, P. Przystałka, M. Gajdzik, and K. Sternal, "A model-based approach for testing automotive embedded systems – a preliminary study," in *Intelligent and Safe Computer Systems in Control and Diagnostics*, pp. 340–351, 2023.

[161] "Using MATLAB with CANoe." https://cdn.vector.com/cms/content/know-how/_application-notes/AN-IND-1-007_Using_MATLAB_with_CANoe.pdf. Accessed: 2024-07-25.

[162] "vTest at a Glance." https://www.vector.com/at/en/products/products-a-z/software/vteststudio/c354652. Accessed: 2024-07-25.

[163] "CANoe. Product information." https://cdn.vector.com/cms/content/products/canoe/canoe/docs/Product Accessed: 2024-07-25.

[164] W. B. Rouse, "AI as Systems Engineering Augmented Intelligence for Systems Engineers," *Insight*, vol. 23, pp. 52–54, 2020.

[165] A. Rehmer and A. Kroll, "On the vanishing and exploding gradient problem in gated recurrent units," *IFAC-PapersOnLine*, vol. 53, no. 2, pp. 1243–1248, 2020. 21st IFAC World Congress.

[166] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift." https://arxiv.org/abs/1502.03167, 2015.

[167] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *J. Mach. Learn. Res.*, vol. 15, pp. 1929–1958, 2014.

[168] N. Gruber and A. Jockisch, "Are gru cells more specific and lstm cells more sensitive in motive classification of text?," *Frontiers in Artificial Intelligence*, vol. 3, 2020.

[169] Y.-W. Lu, C.-Y. Hsu, and K.-C. Huang, "An autoencoder gated recurrent unit for remaining useful life prediction," *Processes*, vol. 8, no. 9, 2020.

[170] Y. Zhao, Z. Nasrullah, and Z. Li, "Pyod: A python toolbox for scalable outlier detection," *Journal of Machine Learning Research*, vol. 20, no. 96, pp. 1–7, 2019.

[171] S. Han, X. Hu, H. Huang, M. Jiang, and Y. Zhao, "Adbench: Anomaly detection benchmark," *Advances in Neural Information Processing Systems*, vol. 35, pp. 32142–32159, 2022.

[172] A. Gnacy-Gajdzik and P. Przystałka, "Automating the analysis of negative test verdicts: A future-forward approach supported by augmented intelligence algorithms," *Applied Sciences*, vol. 14, no. 6, 2024.

[173] Z. Li, Y. Zhao, X. Hu, N. Botta, C. Ionescu, and G. H. Chen, "Ecod: Unsupervised outlier detection using empirical cumulative distribution functions," *IEEE Transactions on Knowledge and Data Engineering*, vol. 35, p. 12181–12193, dec 2023.

[174] M.-L. Shyu, S.-C. Chen, K. Sarinnapakorn, and L. Chang, "A novel anomaly detection scheme based on principal component classifier," in *Proceedings of International Conference on Data Mining*, 01 2003.

[175] F. Angiulli and C. Pizzuti, "Fast outlier detection in high dimensional spaces," in *European Conference on Principles of Data Mining and Knowledge Discovery*, 2002.

[176] F. T. Liu, K. M. Ting, and Z.-H. Zhou, "Isolation forest," in *2008 Eighth IEEE International Conference on Data Mining*, pp. 413–422, 2008.

[177] T. R. Bandaragoda, K. M. Ting, D. Albrecht, F. T. Liu, Y. Zhu, and J. R. Wells, "Isolation-based anomaly detection using nearest-neighbor ensembles," *Computational Intelligence*, vol. 34, no. 4, pp. 968–998, 2018.

[178] C. C. Aggarwal, *Outlier Analysis*. Springer, 2015.

[179] H. Zenati, M. Romain, C. S. Foo, B. Lecouat, and V. R. Chandrasekhar, "Adversarially learned anomaly detection," 2018.

[180] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, ACM, aug 2016.

[181] J. H. Friedman, "Stochastic gradient boosting," *Computational Statistics Data Analysis*, vol. 38, no. 4, pp. 367–378, 2002. Nonlinear Methods and Data Mining.

[182] L. Prokhorenkova, G. Gusev, A. Vorobev, A. V. Dorogush, and A. Gulin, "Catboost: unbiased boosting with categorical features," 2019.

[183] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu, "Lightgbm: A highly efficient gradient boosting decision tree," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, (Red Hook, NY, USA), p. 3149–3157, Curran Associates Inc., 2017.

[184] V. N. Vapnik, "Pattern recognition using generalized portrait method," *Automation and Remote Control*, vol. 24, pp. 774–780, 1963.

[185] B. E. Boser, I. M. Guyon, and V. N. Vapnik, "A training algorithm for optimal margin classifiers," in *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, COLT '92, (New York, NY, USA), p. 144–152, Association for Computing Machinery, 1992.

[186] C. Cortes and V. Vapnik, "Support-vector networks," *Machine Learning*, vol. 20, no. 3, pp. 273–297, 1995.

[187] P. Walendowski, *Zastosowanie sieci neuronowych typu SVM do rozpoznawania mowy*. PhD thesis, Politechnika Wrocławska. Wydział Elektroniki, 2008.

[188] P. F. Orrù, A. Zoccheddu, L. Sassu, C. Mattia, R. Cozza, and S. Arena, "Machine learning approach using mlp and svm algorithms for the fault prediction of a centrifugal pump in the oil and gas industry," *Sustainability*, vol. 12, no. 11, 2020.

[189] M. Michał, "Eksploracja danych – przegląd dostępnych metod i dziedzin zastosowań," in *VI edycja "Hurtownie danych i business intelligence", Centrum Promocji Informatyki, Warszawa, 11 kwietnia 2006*, 2006.

[190] M. Ay, D. Stenger, M. Schwenzer, D. Abel, and T. Bergs, "Kernel selection for support vector machines for system identification of a cnc machining center," *IFAC-PapersOnLine*, vol. 52, no. 29, pp. 192–198, 2019. 13th IFAC Workshop on Adaptive and Learning Control Systems ALCOS 2019.

[191] A. P. Dobrowolski, M. Wierzbowski, and T. K., "Analiza wielorozdzielcza i sieć svm w zastosowaniu do klasyfikacji potencjałów czynnościowych," *Biuletyn Wojskowej Akademii Technicznej*, vol. Vol. 58, nr 3, pp. 275–302, 2009.

[192] T. Ciechulski and O. Stanisław, "Analiza wielorozdzielcza i sieć svm w zastosowaniu do klasyfikacji potencjałów czynnościowych," *Przegląd Elektrotechniczny*, vol. 8, pp. 148–151, 2014.

[193] T. Hastie, R. Tibshirani, J. Friedman, and J. Franklin, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer, 2004.

[194] O. Rainio, J. Teuho, and R. Klén, "Evaluation metrics and statistical tests for machine learning," *Scientific Reports*, vol. 14, 2024.

[195] A. Gnacy-Gajdzik, P. Przystałka, and W. Sebzda, "Analiza częstości występowania fałszywie negatywnych wyników automatycznych testów układów wbudowanych," in *Metody komputerowe - 2021 : Studencka konferencja naukowa*, pp. 41–44, 2021.

[196] "Outlier detection datasets." https://odds.cs.stonybrook.edu/. Accessed: 2024-01-21.

[197] C. A. Ramezan, T. A. Warner, and A. E. Maxwell, "Evaluation of sampling and cross-validation tuning strategies for regional-scale machine learning classification," *Remote Sensing*, vol. 11, no. 2, 2019.

[198] D. Berrar, "Cross-validation," in *Encyclopedia of Bioinformatics and Computational Biology*, Elsevier, 2018.

Streszczenie

Systemy wbudowane stosowane w motoryzacji stają się coraz bardziej złożone, a przede wszystkim realizują funkcje wpływające na bezpieczeństwo uczestników ruchu drogowego. Weryfikacja zaprojektowanych systemów wbudowanych, w aspekcie spełnianych wymagań, szczególnie dotyczących bezpieczeństwa funkcjonalnego, jest istotnym etapem prac inżynieryjnych. Jednym z głównych problemów pojawiającym się na drodze do automatyzacji procesu testowania są anomalie pojawiające się w środowiskach testowych, które mogą zakłócać wyniki i prowadzić do tzw. migotania rezultatów. To z kolei jest przyczyną zwiększenia kosztów i czasu trwania testów ze względu na konieczność przeprowadzenia dogłębnej analizy każdego uzyskanego rezultatu negatywnego.

Rozprawa doktorska proponuje innowacyjną metodykę tworzenia zautomatyzowanych testów systemów wbudowanych umożliwiającą zastosowanie rozszerzonej inteligencji na etapie weryfikacji poprawności wyników testów. W tym celu w pracy zaproponowano dwa algorytmy oparte na synergii sztucznej inteligencji z ludzką, dzięki czemu możliwe jest szybkie przetwarzanie bardzo dużych ilości danych zebranych ze środowiska testowego oraz wykorzystanie uczenia maszynowego. Jednocześnie to człowiek nadzorujący cały proces jest odpowiedzialny za rezultaty, czyli potwierdzenie, że testowany system wbudowany spełnia wszystkie wymagania i nie jest potencjalnym zagrożeniem dla życia i zdrowia ludzkiego.

Sformułowanie metodyki wymagało obszernych, interdyscyplinarnych prac badawczo — rozwojowych, ze względu na konieczność identyfikacji zjawisk zachodzących w środowisku testowym, ich wpływu na elementy mechatroniczne systemu wbudowanego, opracowania algorytmów rozszerzonej inteligencji opartych o algorytmy uczenia maszynowego. Wychodząc naprzeciw wyzwaniom związanym z podniesieniem jakości procesu testowania, w ramach przedstawionej metodyki zaproponowano modyfikację środowiska testowego. Polega ona na zastosowaniu w symulacji modelu pojazdu, co pozwala uwzględnić tym samym wpływ mechaniki pojazdu na testowane funkcje systemu wbudowanego. W pracy zawarto opis stanowiska badawczego, zawierającego model symulacyjny pojazdu, a także w ramach III etapu badań potwierdzono jego pozytywny wpływ na skuteczność identyfikacji defektów systemu wbudowanego powiązanych z dynamiką ruchu pojazdu. Oceniono skuteczność dwóch zaproponowanych algorytmów rozszerzonej inteligencji: pierwszego — opartego o rekurencyjne sieci neuronowe (z warstwami LSTM, GRU) oraz drugiego — wykorzystującego popularne algorytmy służące do wykrywania anomalii. Potwierdzono, że obydwa rozwiązania skutecznie identyfikują anomalie w środowisku testowym będące przyczyną występowania rezultatów migoczących, wskazując na fałszywie negatywne rezultaty testów. Weryfikację zaproponowanej metodyki przeprowadzono na podstawie danych zarejestrowanych podczas wykonywania przypadków testowych przygotowanych zgodnie z jej wytycznymi, zarówno w środowisku testowym pierwotnym, jak i zawierającym model pojazdu.

Wyniki rozprawy doktorskiej stanowią istotny wkład w bezpieczeństwo i niezawodność systemów wbudowanych projektowanych w przemyśle motoryzacyjnym, a w szczególności w optymalizację procesu testowania. Zastosowanie modelu pojazdu jako modularnego oraz konfigurowalnego elementu środowiska testowego pozytywnie wpływa na jakość systemów wbudowanych, zwiększając efektywność i wiarygodność procesu testowania. Zaproponowana wydajna i skalowalna metodyka tworzenia zautomatyzowanych testów systemów wbudowanych w pojazdach samochodowych umożliwia efektywną weryfikację poprawności wyników testów z wykorzystaniem algorytmów rozszerzonej inteligencji.

Abstract

Embedded systems in automotive applications are becoming increasingly complex and, above all, they implement functions that affect the safety of road users. Verification of designed embedded systems, in terms of fulfillment of requirements, especially for functional safety, is an essential stage of product development. One of the main problems that arise when automating the testing process are anomalies occuring in test environments, which may influence on the results and lead to the so-called flickering results. This subsequently causes an increase in the cost and duration of the testing due to the necessity of analyzing each negative result.

The dissertation proposes an innovative methodology for creating automated tests of embedded systems that enables the application of augmented intelligence at the stage of verification of the test results correctness. For this purpose, the study proposes two algorithms based on the synergy of artificial intelligence with human intelligence, which allows the fast processing of vast data volumes collected from the test environment and the application of machine learning. However, the human being supervises the entire process and is responsible for results and confirmation that the tested embedded system satisfies all requirements and it is not a potential threat to people's life and health.

The formulation of the methodology required extensive, interdisciplinary research and development efforts. This was due to the need for identifying the phenomena occurring in the test environment, their impact on the mechatronic components of the embedded system, developing augmented intelligence algorithms based on machine learning algorithms. To address challenges of improving the quality of testing process, a modification of the test environment was proposed as part of the presented methodology. It involves applying a vehicle model in the simulation, which allows for reflecting the influence of vehicle mechanics on the tested functions of the embedded system. The paper contains a detailed description of the research test bench, which includes the vehicle simulation model. The third stage of the research confirmed its positive impact on the effectiveness of identyfying embedded system defects related to vehicle's motion dynamics. The effectiveness of two proposed augmented intelligence algorithms was evaluated: one based on recurrent neural networks (with LSTM and GRU layers) and second incorporating anomaly detection algorithms. It was confirmed that both solutions successfully identify anomalies in the test environment that cause flickering results and indicate false negative test results. Verification of the proposed methodology was performed on the datasets recorded during the execution of test cases prepared in accordance with its guidelines, both in the original test environment and in the environment enhanced with the vehicle model.

The results of the dissertation provide an important contribution to the safety and reliability of embedded systems designed in the automotive industry, and in particular to the optimization of the testing process. The application of a vehicle model as a modular and configurable element of the test environment positively affects the quality of embedded systems, enhancing the efficiency and credibility of the testing process. The proposed efficient and scalable methodology for creating automated tests of automotive embedded systems enables efficient verification of test results correctness using augmented intelligence algorithms.