Politechnika Wrocławska
Wydział Informatyki i Telekomunikacji

Kierunek: **Informatyka Stosowana (IST)**
Specjalność: **Projektowanie Systemów Informatycznych (PSI)**

PRACA DYPLOMOWA MAGISTERSKA

Zastosowanie kamery wizyjnej i lidaru do percepcji otoczenia bolidu autonomicznego

Kacper Synator

Opiekun pracy
dr inż. Robert Muszyński

Słowa kluczowe: Formuła Student, lidar, kamera wizyjna, percepcja otoczenia, fuzja danych

WROCŁAW 2025

Spis treści

Rozdział 1

Wstęp

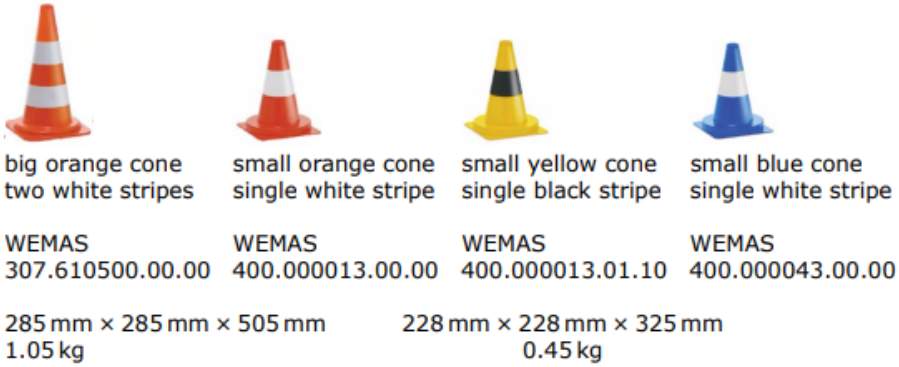
Samochody autonomiczne stają się coraz bardziej popularne w ostatnich latach dzięki dynamicznemu rozwojowi technologii percepcji otoczenia zarówno pod kątem sprzętowym, jak i programowym. Choć technologia autonomiczna znajduje szerokie zastosowanie przede wszystkim w sektorze komercyjnym, na przykład w transporcie pasażerskim i logistyce, coraz częściej wykorzystywana jest również w sporcie motorowym czego przykładem są zawody Formula Student Driverless organizowane od 2017 roku [1]. Są to międzynarodowe zawody studenckie, w których zespoły projektują i budują bolidy wyścigowe z systemem jazdy autonomicznej, zgodne z regulaminem [2]. Następnie biorą udział w konkurencjach autonomicznych: Acceleration, Skidpad, Autocross i Trackdrive, które sprawdzają ogólne osiągi samochodu oraz jego zdolności jazdy autonomicznej takie jak percepcja otoczenia czy planowanie trasy.

Tory konkurencji autonomicznych Formuły Student są oznaczone przedstawionymi na rysunku 1.2 pachołkami. Pomarańczowe pachołki oznaczają początek i koniec toru, niebieskie pachołki wyznaczają lewą krawędź toru, natomiast pachołki żółte jego prawą krawędź. Do wykrywania pachołków systemy percepcji wykorzystują pojedyncze sensory, takie jak kamera wizyjna [3] lub lidar [4]. Jednakże, aby uzyskać lepszą dokładność i niezawodność, często stosuje się fuzję danych z wielu sensorów, wykorzystując w tym celu algorytmy dokonujące projekcji danych z przestrzeni pomiarowej jednego sensora do drugiego. Powszechnie stosowanym podejściem jest projekcja 3D na 2D, gdzie dane z lidaru są przekształcane do obrazu kamery wizyjnej, a następnie jest dokonywana detekcja np. poprzez sieci neuronowe [5] czy ograniczenie projekcji danych z lidaru trójkątem [6] lub prostokątem [7] i dopasowanie ich do detekcji uzyskanych bezpośrednio z obrazu kamery wizyjnej.

Celem pracy jest zaprojektowanie i implementacja systemu percepcji otoczenia bolidu autonomicznego, który wykorzystuje kamerę wizyjną i lidar oraz porównanie efektywności zaimplementowanego systemu do metod i systemów powszechnie stosowanych w literaturze. Analiza porównawcza efektywności obejmuje zbadanie dokładności, szybkości działania oraz wykorzystania zasobów obliczeniowych systemu.

Nowatorski algorytm fuzji danych zaprezentowany w pracy jest oparty na projekcji 2D na 3D, w której dane z kamery wizyjnej są przekształcane do przestrzeni lidaru. Porównano go z algorytmem używającym powszechnie stosowaną projekcję 3D na 2D. Obydwa algorytmy zostały zaimplementowane i przetestowane na bolidzie RT14e 1.1 zespołu PWR Racing Team, który jest piętnastym bolidem zespołu i trzecim bolidem elektrycznym, brał on udział w pięciu edycjach zawodów Formuły Student w roku 2024.

Rysunek 1.1 Bolid RT14e podczas przejazdu autonomicznego (FS Czech 2024)



Rysunek 1.2 Pachołki wykorzystywane w konkurencjach autonomicznych Formuły Student [8]

Układ pracy jest następujący. W rozdziale drugim opisano percepcję otoczenia bolidu, wraz z niezbędnym aparatem matematycznym do implementacji algorytmów fuzji danych sensorycznych. W rozdziale trzecim przedstawiono system autonomiczny bolidu RT14e od strony oprogramowania i sprzętu. Rozdział czwarty zawiera opis implementacji algorytmów fuzji danych a rozdział piąty ich testy porównawcze. W rozdziale szóstym znajduje się podsumowanie.

Rozdział 2

Percepcja otoczenia bolidu

Aby poprawnie i bezpiecznie poruszać się autonomiczne pojazdy muszą być w stanie rozpoznawać swoje otoczenie. W tym celu wykorzystuje się różne sensory, takie jak kamery, lidary czy radary. Dla zwiększenia dokładności i niezawodności percepcji stosowana jest fuzja pochodzących z różnych źródeł sygnałów. Każdy sensor posiada swój własny układ współrzędnych, w odniesieniu do którego zwracane są dane. Żeby poprawnie zintegrować dane z różnych sensorów, konieczne jest przekształcenie ich do wspólnego układu współrzędnych. W tym rozdziale przedstawiono dwa różne algorytmy fuzji danych z kamery i lidaru, wraz z metodami w nich wykorzystywanymi. Omówiono również przekształcenia pomiędzy układami współrzędnych różnych sensorów.

2.1 Projekcja 3D na 2D

Projekcja punktów z przestrzeni 3D, na płaszczyznę dwuwymiarowego obrazu kamery polega na przekształceniu ich z układu współrzędnych świata do punktów (pikseli) na obrazie z kamery. Rysunek 2.1 ilustruję ową projekcję. Przyjmuję się, że układ współrzędnych świata jest statycznym układem globalnym, w którym znajdują się wszystkie obiekty. Projekcję punktów można również dokonać z układu współrzędnych innego sensora, wtedy układ współrzędnych świata widoczny na rysunku 2.1 jest układem współrzędnych tego sensora.

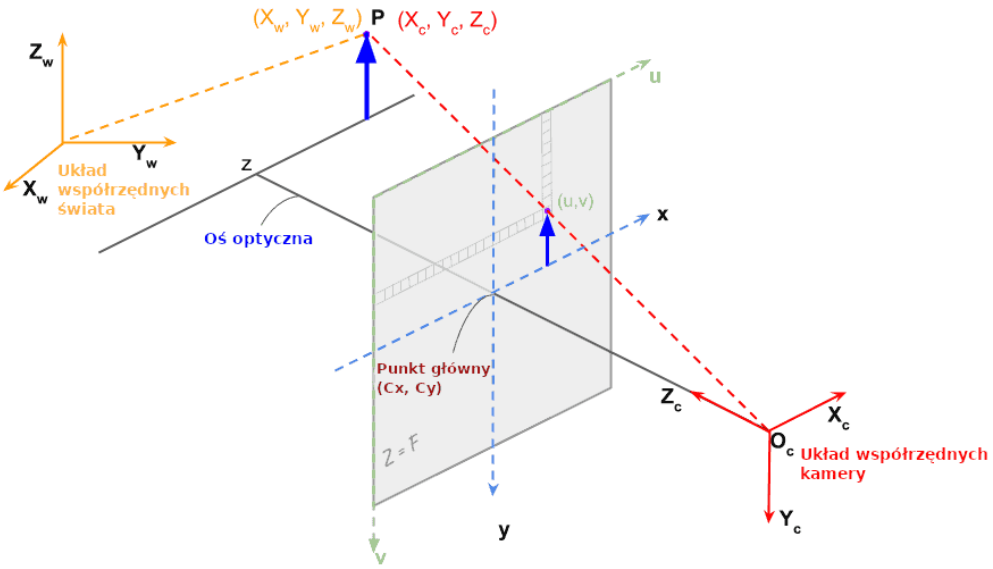
Transformacja z układu współrzędnych świata do układu współrzędnych kamery jest opisana równaniem [10]

$$\begin{bmatrix} x_c \\ y_c \\ z_c \\ 1 \end{bmatrix} = \begin{bmatrix} R & T \\ 0^T & 1 \end{bmatrix} \begin{bmatrix} x_w \\ y_w \\ z_w \\ 1 \end{bmatrix},$$

gdzie R to macierz rotacji a T wektor translacji. Projekcja współrzędnych z układu współrzędnych kamery na obraz jest opisana równaniem [10]

$$\begin{bmatrix} x_i \\ y_i \\ z_i \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_c \\ y_c \\ z_c \end{bmatrix} = K \begin{bmatrix} x_c \\ y_c \\ z_c \end{bmatrix},$$

gdzie K to macierz kalibracji kamery, f_x i f_y ogniskowe kamery a c_x i c_y współrzędne punktu głównego kamery. Sumarycznie, całą operację projekcji można opisać równaniem [10]



Rysunek 2.1 Projekcja punktów przestrzeni 3D na płaszczyznę obrazu kamery na podstawie [9]

$$\begin{bmatrix} x_i \\ y_i \\ z_i \end{bmatrix} = K \begin{bmatrix} R & T \end{bmatrix} \begin{bmatrix} x_w \\ y_w \\ z_w \\ 1 \end{bmatrix} = P \begin{bmatrix} x_w \\ y_w \\ z_w \\ 1 \end{bmatrix},$$

gdzie P to macierz projekcji zwana również macierzą kamery. Eliminując trzeci wymiar za pomocą transformacji do współrzędnych niejednorodnych uzyskane zostają piksele na obrazie kamery [10]

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} x_i/z_i \\ y_i/z_i \\ z_i/z_i \end{bmatrix}.$$

2.2 Projekcja 2D na 3D

Projekcja 2D na 3D jest operacją odwrotną do projekcji opisanej w poprzednim podrozdziale. Polega ona na przekształceniu dwuwymiarowych współrzędnych z obrazu kamery do współrzędnych trójwymiarowych w układzie współrzędnych świata. Do wykonania odwrotnej projekcji potrzebna jest dodatkowa informacja, utracona podczas przekształcenia z przestrzeni trójwymiarowej do dwuwymiarowej, o odległości obiektu od kamery, pochodząca np. z innego sensora.

Mnożąc współrzędne pikseli z obrazu przez macierz odwrotną do macierzy K uzyskuje się wektor kierunkowy prostej przechodzącej przez początek układu współrzędnych kamery i dany piksel obrazu kamery, co opisuje równanie

$$\begin{bmatrix} x_{cn} \\ y_{cn} \\ 1 \end{bmatrix} = K^{-1} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} 1/f_x & 0 & -c_x/f_x \\ 0 & 1/f_y & -c_y/f_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} (u-c_x)/f_x \\ (v-c_y)/f_y \\ 1 \end{bmatrix},$$

gdzie K^{-1} to macierz odwrotna do macierzy kalibracji kamery.

Dodatkowa informacja o odległości obiektu od kamery jest wykorzystana do przeskalowania wektora kierunkowego prostej, co prowadzi do uzyskania współrzędnych trójwymiarowych w układzie współrzędnych kamery. Równanie

$$\begin{bmatrix} x_c \\ y_c \\ z_c \end{bmatrix} = \frac{d}{\sqrt{x_{cn}^2 + y_{cn}^2 + 1}} \begin{bmatrix} x_{cn} \\ y_{cn} \\ 1 \end{bmatrix}$$

opisuje powyższy proces, gdzie d to odległość obiektu od kamery.

Po wyliczeniu współrzędnych w układzie kamery i wykorzystaniu własności ortogalności macierzy rotacji punkty są przekształcane do układu współrzędnych świata za pomocą równania

$$\begin{bmatrix} x_w \\ y_w \\ z_w \\ 1 \end{bmatrix} = \begin{bmatrix} R^T & -R^T T \\ 0^T & 1 \end{bmatrix} \begin{bmatrix} x_c \\ y_c \\ z_c \\ 1 \end{bmatrix}.$$

Całą operację odwrotnej projekcji można sumarycznie opisać równaniem

$$\begin{bmatrix} x_w \\ y_w \\ z_w \\ 1 \end{bmatrix} = \begin{bmatrix} R^T & -R^T T \\ 0^T & 1 \end{bmatrix} sK^{-1} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix},$$

gdzie $s = \frac{d}{\sqrt{x_{cn}^2 + y_{cn}^2 + 1}}$.

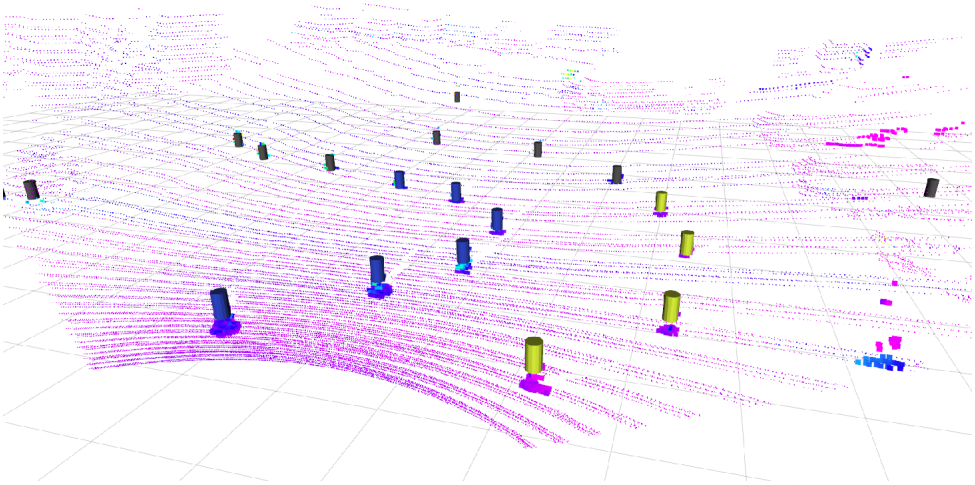
2.3 Fuzja danych sensorycznych

Do fuzji danych zostały wykorzystane sygnały pochodzące z dwóch sensorów, lidaru i kamery. Lidar zwraca chmurę punktów, gdzie każdy punkt jest opisany przez jego współrzędne w przestrzeni 3D i intensywność[1]. Po odfiltrowaniu gruntu, klasteryzacji i detekcji poszczególnych klastrów zostają uzyskane wykrycia obiektów. Na rysunku 2.2 przedstawiono przykładową chmurę punktów z lidaru, gdzie wykrytymi obiektami są pachołki.

Kamera zwraca kolorowy obraz, gdzie każdy piksel jest opisany kolorem w formacie RGB. Stosując konwolucyjne sieci neuronowe uzyskane zostają wykrycia obiektów w formie ramek ograniczających[2]. Na rysunku 2.3 przedstawiono przykładowe zdjęcie z kamery z naniesionymi wykryciami obiektów, którymi są pachołki.

W dalszej części rozdziału omówiono kalibrację sensorów percepcji i przedstawiono dwa algorytmy fuzji danych. Pierwszy wykorzystuje wykrycia pachołków z obu sensorów i projekcję 3D na 2D. Drugi algorytm wykorzystuje wykrycia tylko z kamery i nieprzetworzoną chmurę punktów z lidaru oraz projekcję 2D na 3D.

[1] ang. intensity
[2] ang. bounding box



Rysunek 2.2 Wizualizacja skanu z lidaru z nałożonymi wykryciami pachołków [11]

Rysunek 2.3 Obraz z kamery z nałożonymi wykryciami pachołków

2.3.1 Kalibracja

Do projekcji punktów z lidaru na obraz kamery, jak również odwrotnej projekcji potrzebna jest macierz projekcji P, która jest złożeniem macierzy kalibracji kamery K i macierzy transformacji z układu współrzędnych lidaru do układu współrzędnych kamery. Kalibracja polega na określeniu powyższych macierzy. Przykładowe metody kalibracji są następujące:

- wykorzystujące wzorce kalibracyjne[3] takie jak szachownice czy ArUco [12],
- oparte na ruchu[4] wykorzystujące odometrię platformy, na jakiej znajdują się sensory,
- oparte na scenie[5], w których wykorzystywane są punkty charakterystyczne sceny takie jak krawędzie czy ściany,
- wykorzystujące głębokie uczenie[6] np. sieci uczone od początku do końca[7] lub sieci, które wykrywają cechy charakterystyczne sceny, wykorzystywane później do kalibracji inną metodą.

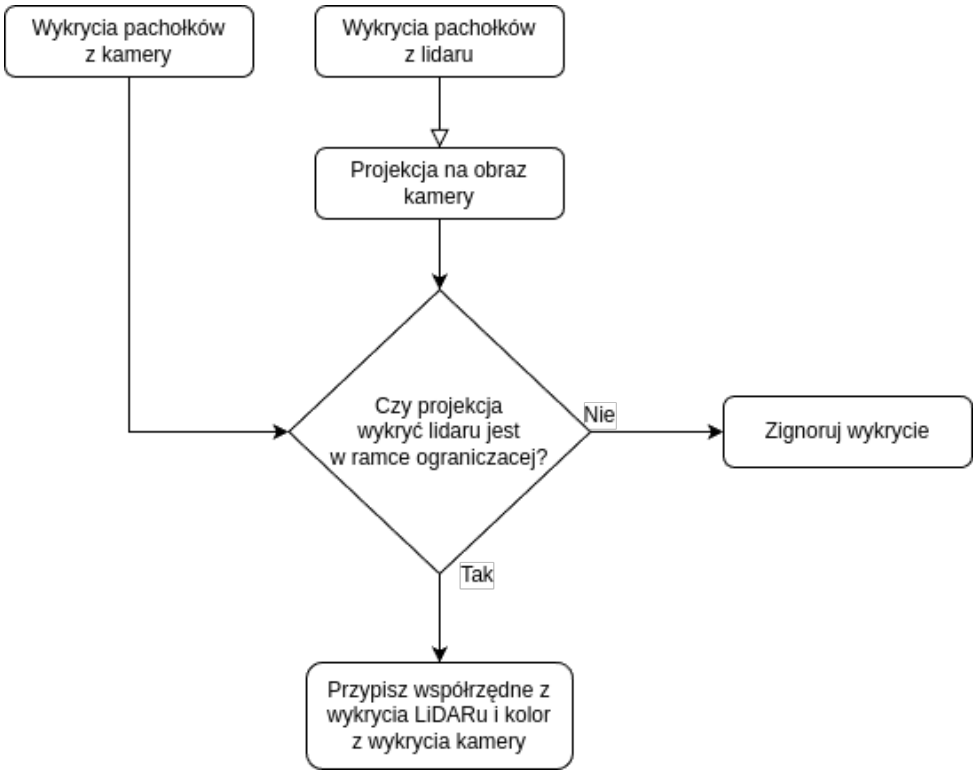
2.3.2 Algorytm fuzji z projekcją 3D na 2D

Algorytm fuzji z projekcją 3D na 2D (zwany algorytmem 3D->2D) polega na połączeniu wykryć pachołków z lidaru i kamery. Na rysunku 2.4 przedstawiono jego szczegóły. Wejściem algorytmu są zsynchronizowane wykrycia pachołków pochodzące z danych kamery i lidaru. W pierwszej kolejności dokonywana jest projekcja wykryć uzyskanych na podstawie danych lidaru na obraz kamery. Kolejnym krokiem jest sprawdzenie, czy projekcje znajdują się wewnątrz ramek ograniczających, jeśli tak to wykrycia są uznawane za poprawne. Ostatnim krokiem jest przypisanie poprawnym wykryciom pozycji z wykrycia pochodzącego z danych lidaru i koloru, który został uzyskany na podstawie wykryć z obrazu kamery.

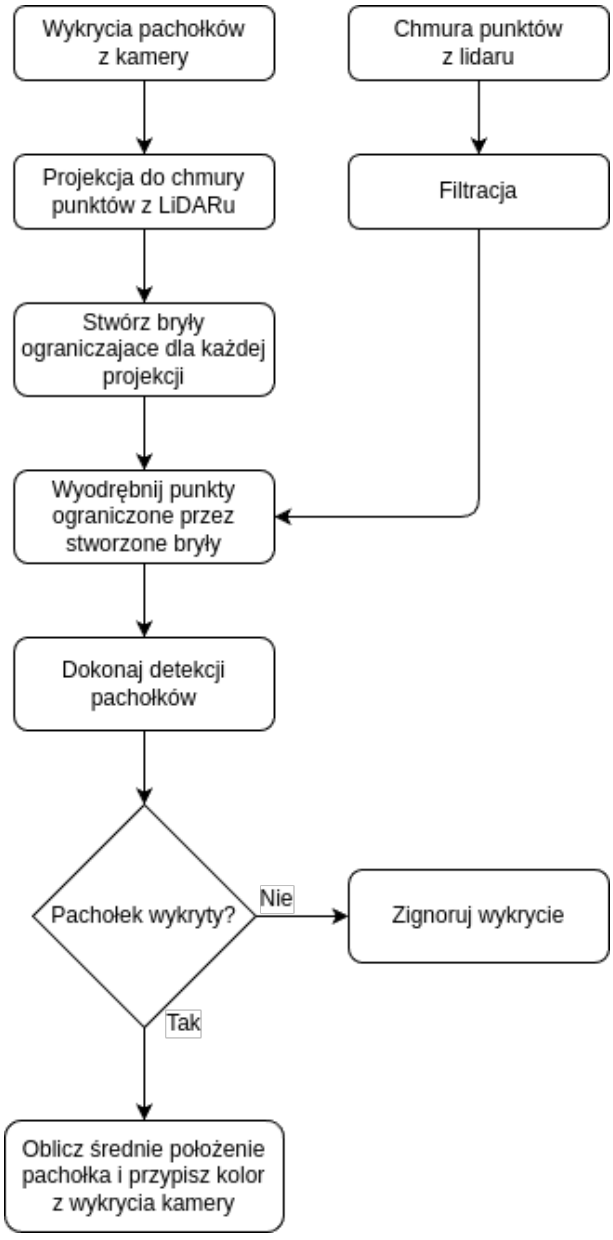
2.3.3 Algorytm fuzji z projekcją 2D na 3D

Algorytm fuzji z projekcją 2D na 3D (zwany algorytmem 2D->3D) polega na fuzji wykryć pachołków z kamery i nieprzetworzonej chmury punktów z lidaru. Na rysunku 2.5 przedstawiono szczegóły algorytmu. Wejściem algorytmu są zsynchronizowane wykrycia pachołków pochodzące z kamery oraz nieprzetworzona chmura punktów z lidaru. W pierwszej kolejności dokonywana jest projekcja wykryć pochodzących z obrazu kamery do układu współrzędnych lidaru. Uzyskane projekcje są następnie wykorzystane jako środki prostopadłościanów, którymi zostaje ograniczona przefiltrowana chmura punktów. Uzyskane w ten sposób klastry punktów są poddane detekcji i jeśli zostanie wykryty pachołek, to zostaje wyliczona średnia pozycja klastra, która jest następnie używana jako pozycja wykrycia. Finalnie kolor zostaje przypisany na podstawie wykrycia uzyskanego z obrazu kamery.

[3] ang. Target-based
[4] ang. Motion-based
[5] ang. Scene-based
[6] ang. Deep-learning
[7] ang. end-to-end



Rysunek 2.4 Algorytm fuzji z projekcją 3D na 2D



Rysunek 2.5 Algorytm fuzji z projekcją 2D na 3D

Rozdział 3

System autonomiczny bolidu RT14e

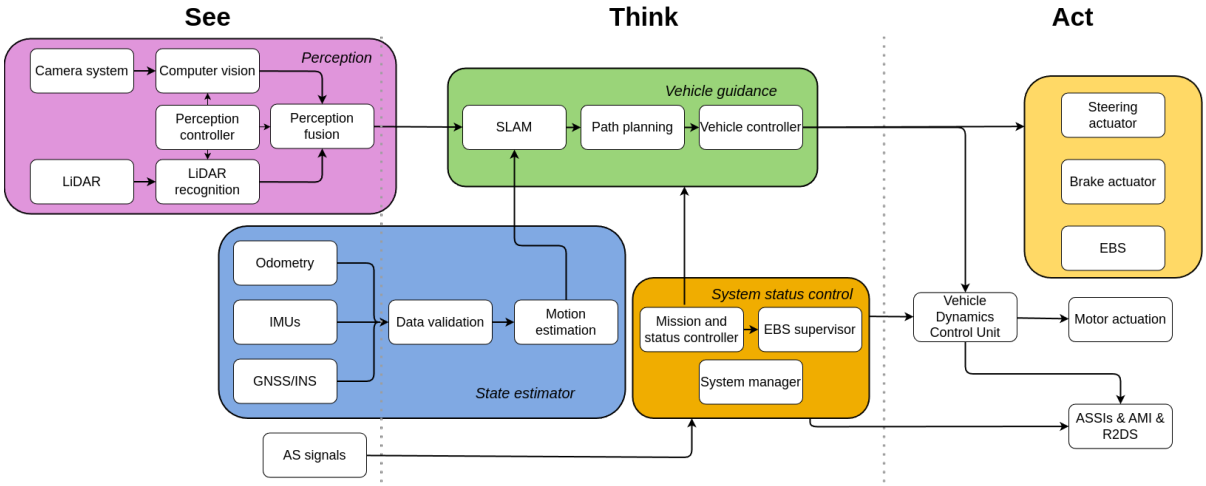
W rozdziale przedstawiono środowisko pracy, jakim jest bolid RT14e i jego system autonomiczny. Opisano architekturę systemu autonomicznego, jej implementację oraz sprzęt.

3.1 Architektura

Architektura systemu autonomicznego bolidu RT14e bazuję na zasadzie Patrz-Myśl-Działaj[1]. W systemie można wyróżnić pięć głównych modułów (zobacz rysunek 3.1):

- moduł percepcji[2], którego zadaniem jest zbieranie informacji o otoczeniu za pomocą kamery i lidaru;
- moduł estymatora stanu[3], służący do estymacji stanu pojazdu na podstawie danych czujników prędkości kół, IMU[4] i GPS[5];
- moduł prowadzenia pojazdu[6], który lokalizuje pojazd w otoczeniu i wyznacza ścieżkę jazdy. Służy również do kontroli prędkości i przyspieszenia pojazdu;
- moduł kontroli statusu systemu[7], służący do obserwacji, kontroli i monitorowania stanu pozostałych komponentów systemu autonomicznego, dokonuje również wyboru misji autonomicznej oraz określa stan systemu zgodnie z regulaminem Formuły Student;
- aktuatory, wykonują polecenia systemu autonomicznego, ich głównymi zadaniami są: kontrola silników elektrycznych, hamulców i układu kierowniczego.

[1] ang. See-Think-Act
[2] ang. Perception
[3] ang. State estimator
[4] ang. Inertial Measurement Unit
[5] ang. Global Positioning System
[6] ang. Vehicle guidance
[7] ang. System status control



Rysunek 3.1 Architektura systemu autonomicznego bolidu RT14e

3.2 ROS2

System autonomiczny bolidu RT14e bazuje na platformie programistycznej ROS2 [13]. Dostarcza ona zestaw narzędzi i bibliotek do tworzenia aplikacji lub całego systemu. W odróżnieniu od pierwszej wersji, jądro ROS2 jest napisane za pomocą nowoczesnego języka C++[8]. Interfejs programistyczny ROS2 w języku Python również został zaktualizowany z wersji 2 do 3.5.

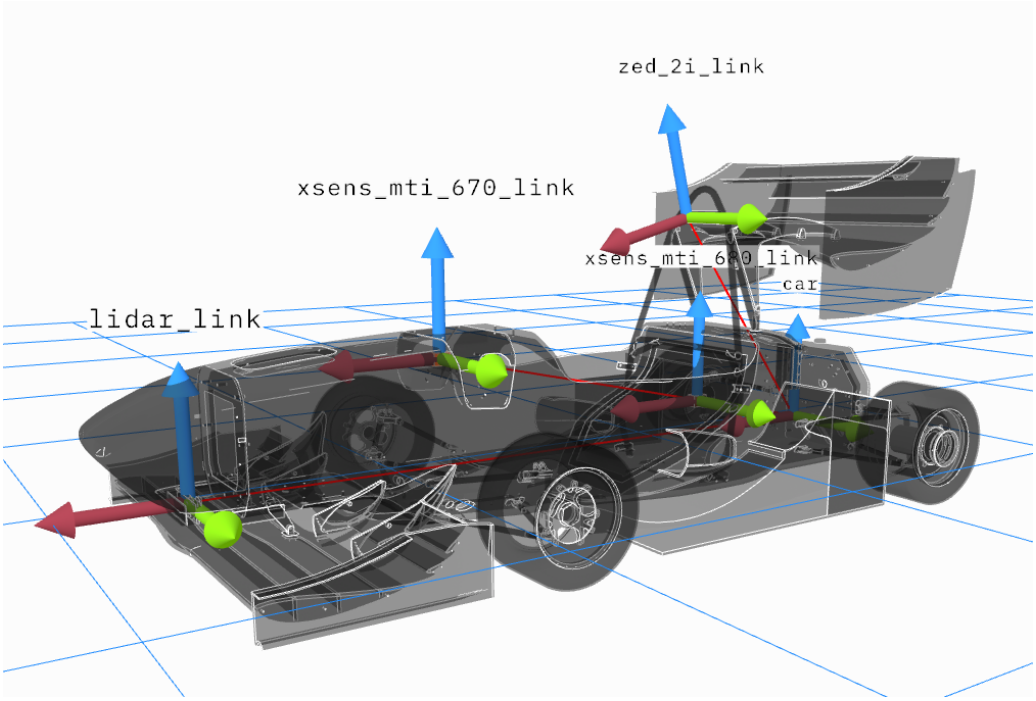
Warstwa komunikacji ROS2 oparta jest na standardzie DDS[9], co pozwala na tworzenie zdecentralizowanych systemów rozproszonych. DDS oferuje również różne profile QoS[1], które pozwalają na dostosowanie komunikacji do wymagań aplikacji i sieci.

3.2.1 Węzeł

W systemie autonomicznym bolidu RT14e każdy komponent systemu autonomicznego jest reprezentowany przez węzeł[2], który jest podstawowym elementem systemu ROS2. Każdy węzeł realizuje jedną, określoną funkcjonalność. Węzły komunikują się ze sobą za pomocą tematów[3], które wykorzystują model publikacji i subskrypcji, Innym sposobem komunikacji w ROS2 jest model klient-serwer[4], gdzie jeden węzeł pełni rolę serwera odpowiadającego na zapytania klienta, który jest drugim węzłem.

Węzły w systemie ROS2 mogą być uruchamiane jako osobne procesy lub z zastosowaniem kompozycji [14] jako część jednego procesu. Pierwsza opcja zapewnia większe bezpieczeństwo i izolację, natomiast druga pozwala na lepszą wydajność dzięki komunikacji wewnątrzprocesowej.

[8] ang. modern C++ – standard C++11 i nowszy
[9] ang. Data Distribution Service
[1] ang. Quality of Service
[2] ang. Node
[3] ang. Topics
[4] ang. Client-Server



Rysunek 3.2 Wizualizacja bolidu RT14e z naniesionymi układami współrzędnych

3.2.2 Biblioteka tf2

Do zarządzania transformacjami pomiędzy układami współrzędnych w systemie autonomicznym bolidu RT14e wykorzystano bibliotekę tf2 [15]. Umożliwia ona łatwe zarządzanie transformacjami pomiędzy układami współrzędnych nie tylko dla bieżącej chwili, ale i wcześniejszych. Na rysunku 3.2 przedstawiono bolid RT14e z naniesionymi układami współrzędnych. Od lewej widoczne są następujące układy współrzędnych:

- lidar_link – układ współrzędnych lidaru,
- xsens_mti_link_670_link – układ współrzędnych przedniego sensora Xsens,
- zed_2i_link – układ współrzędnych kamery,
- xsens_mti_link_660_link – układ współrzędnych tylnego sensora Xsens,
- car – układ współrzędnych pojazdu, który jest używany do lokalizacji pojazdu w układzie świata.

3.3 Percepcja

Moduł percepcji składa się z następujących komponentów:

- komponent Computer Vision[5] – węzeł służy do akwizycji i przetwarzania obrazu z kamery. Dokonuje detekcji pachołków na obrazie, z użyciem modelu YOLO[6]. W zależności od konfiguracji może to być model YOLOv8 [16] lub YOLOv11 [17];

[5] zwany również węzłem kamery
[6] ang. You Only Look Once

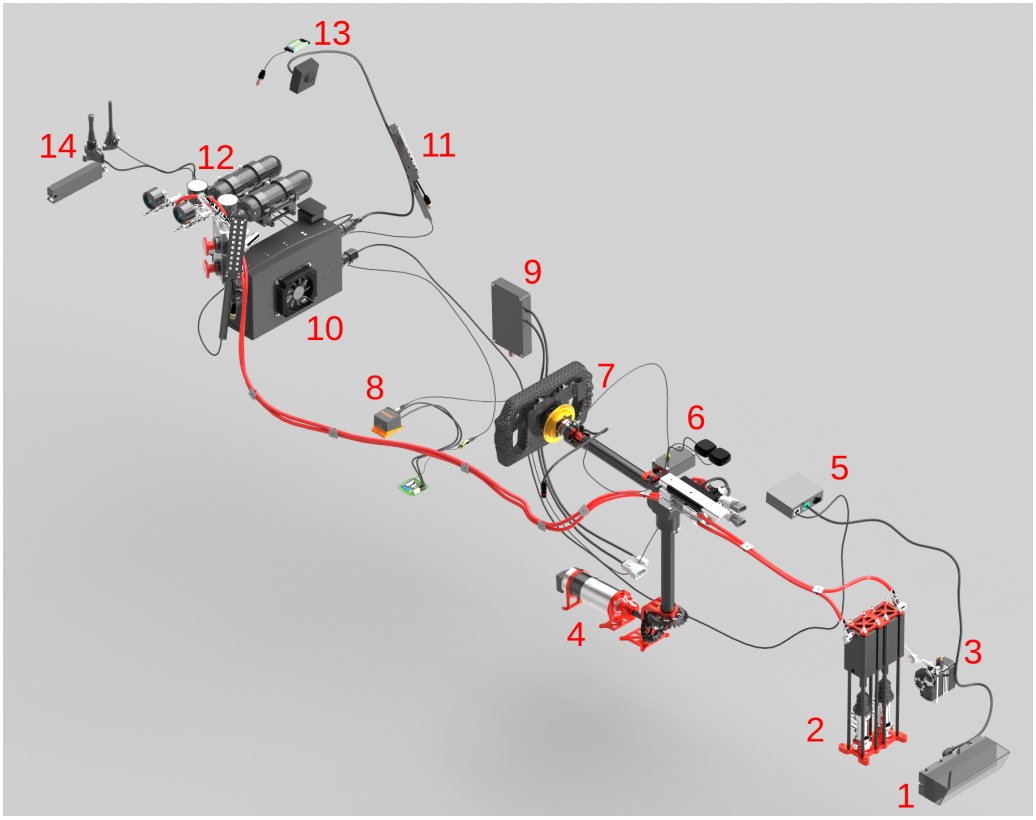
- komponent LiDAR recognition[7] – węzeł jest wykorzystywany do akwizycji i przetwarzania danych z lidaru. Dokonuje detekcji pachołków na podstawie chmury punktów. Opisany dokładniej w pracy [11];
- komponent Perception fusion[8] – węzeł realizuje fuzję danych z kamery i lidaru. Szczegóły implementacji zostały opisane w rozdziale 4;
- komponent Perception controller – węzeł służy do kontroli i monitorowania stanu innych węzłów percepcji. Określa tryb działania percepcji. Dokonuje również synchronizacji danych z lidaru i kamery.

3.4 Sprzęt

Na rysunku 3.3 został przedstawiony schematycznie system autonomiczny bolidu RT14e od strony sprzętowej, gdzie odpowiednio zostały oznaczone:

1. lidar – wykorzystywany w percepcji otoczenia,
2. elektrozawory awaryjnego systemu hamowania[9] – aktywują one awaryjny system hamowania, w razie awarii systemu autonomicznego lub po otrzymaniu sygnału ze zdalnego systemu awaryjnego[1],
3. serwo autonomicznego systemu hamulcowego[2] – służy do aktuacji pedału hamulca w pojeździe,
4. silnik autonomicznego układu kierowniczego – umożliwia autonomiczne sterowanie układem kierowniczym pojazdu,
5. konwerter mediów[3] wykorzystywany, do konwersji sygnału z lidaru,
6. przedni czujnik Xsens – służy do estymacji stanu pojazdu,
7. kierownica układu kierowniczego – posiada wbudowany ekran, na którym wyświetlane są informacje o stanie systemu,
8. tylny czujnik Xsens – patrz przedni czujnik Xsens,
9. rejestrator danych[4] – zapisuje wiadomości wysyłane poprzez magistrale CAN[5],
10. moduł obliczeniowy – główny komputer systemu autonomicznego,
11. sygnalizator stanu systemu autonomicznego[6] – sygnalizuje stan systemu autonomicznego za pomocą diod LED,

[7] zwany również węzłem lidaru
[8] fuzja percepcji odnosi się do fuzji danych z czujników percepji
[9] ang. EBS – Emergency Brake System
[1] ang. RES – Remote Emergency System
[2] ang. ASB – Autonomous System Brake
[3] ang. Media Converter Box
[4] ang. Data Logger
[5] ang. Controller Area Network
[6] ang. ASSI – Autonomous System Status Indicator



Rysunek 3.3 Render sprzętu systemu autonomicznego bolidu RT14e

12. zbiorniki sprężonego powietrza – magazynują sprężone powietrze używane do aktywacji elektrozaworów awaryjnego systemu hamowania,
13. kamera – wykorzystywana do percepcji otoczenia,
14. anteny GPS – podłączone do tylnego sensora Xsens.

W dalszej części podrozdziału przedstawiono szczegółowy opis komponentów związanych z tematyką pracy.

3.4.1 Moduł obliczeniowy

Moduł obliczeniowy bolidu RT14e to komputer NVIDIA Jetson AGX Orin [18]. Jest to komputer o wysokiej wydajności, posiadający 12 rdzeni CPU, 2048 rdzeni GPU oraz 64 rdzenie tensorów. Posiada również 64GB pamięci RAM współdzielonej z GPU i wsparcie sprzętowe do kodowania i dekodowania wideo. Zainstalowany na nim system operacyjny to Jetson Linux 36.4.3, który jest częścią JetPack SDK 6.2 [19]. W celu osiągnięcia największej wydajności systemu autonomicznego bolidu RT14e komputer jest skonfigurowany do działania w trybie zasilania 50W.

3.4.2 Lidar

Zastosowany w bolidzie lidar Velodyne Velarray m1600 [20] (rysunek 3.4) to lidar typu solid-state, czyli pozbawiony ruchomych części. Posiada on 16 par laserów i detek-

Rysunek 3.4 Lidar zamontowany na bolidzie RT14e

torów. Lidar jest podłączony za pomocą przewodu 1000BASE-T1 do konwertera mediów, który przetwarza sygnał na Gigabit Ethernet (GbE), który jest podłączony do modułu obliczeniowego (zobacz rysunek 3.6). Dane z lidaru są przesyłane za pomocą protokołu UDP, które są odbierane i przetwarzane, a następnie wysyłane za pomocą tematu systemu ROS2 przez sterownik VellaGo dostarczony przez producenta. Na potrzeby systemu autonomicznego bolidu RT14e, lidar jest skonfigurowany następująco:

- częstotliwość skanowania: 20Hz,
- tryb powrotu lasera[7]: pojedynczy powrót najsilniejszej wiązki[8].

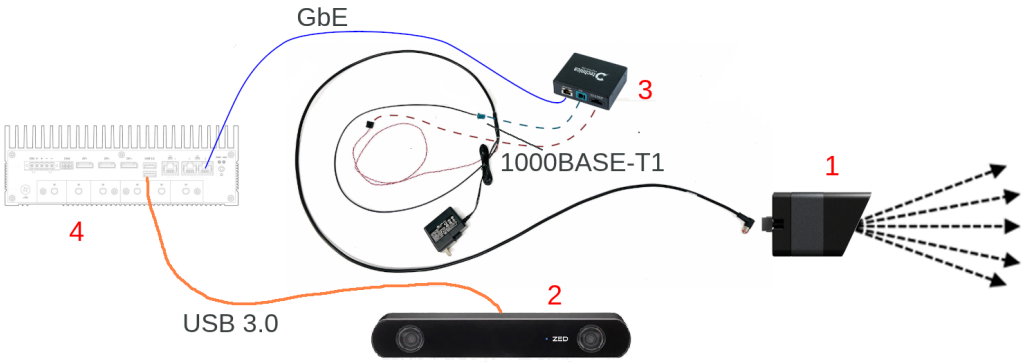
3.4.3 Kamera

Do percepcji otoczenia bolidu RT14e wykorzystano kamerę Stereolabs Zed 2i [21]. Jest to kamera stereoskopowa o polu widzenia 110°, która posiada wbudowane sensory IMU, żyroskop, barometr i magnetometr. Kamera jest podłączona do modułu obliczeniowego za pomocą interfejsu USB 3.0 (zobacz rysunek 3.6). Kamera jest obsługiwana za pomocą ZED SDK 5.0.0 [22]. Kamera na potrzeby tej pracy jest używana tylko do akwizycji obrazu z jednego obiektywu[9]. Konfiguracja kamery jest następująca:

- rozdzielczość: 1280x720,
- maksymalna częstotliwość: 60Hz,
- tryb głębi: brak[1].

[7] ang. Laser Return Mode
[8] ang. Single Return Mode: Strongest
[9] w przeszłości był wykorzystywany również obraz z drugiego obiektywu do obliczenia głębi obrazu
[1] ang. Depth Mode: None

Rysunek 3.5 Kamera zamontowana na bolidzie RT14e



Rysunek 3.6 Poglądowy schemat połączeń lidaru (1), kamery (2), konwertera mediów (3) i modułu obliczeniowego (4). Na podstawie [20, 21]

Rozdział 4

Implementacja

W rozdziale opisano implementację fuzji sygnałów z sensorów percepcji. Przedstawiono funkcjonalności wchodzących w skład systemu węzłów, które dokonują synchronizacji i fuzji danych z lidaru i kamery. Zostały również opisane procesy kalibracji potrzebne do wyznaczenia macierzy kamery K oraz macierzy transformacji z układu współrzędnych lidaru do układu współrzędnych kamery.

4.1 Kalibracja kamery

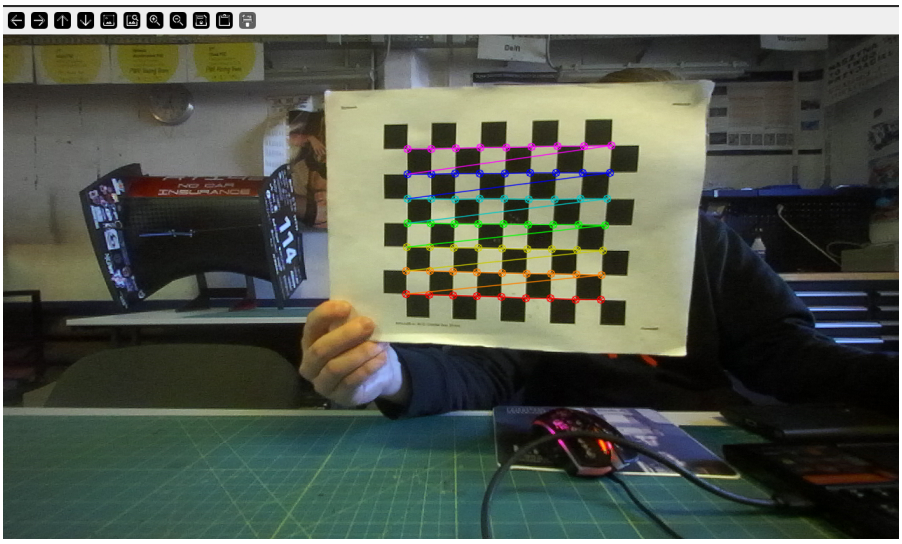
Kalibracja kamery jest procesem, w którym określane zostają parametry kamery: długość ogniskowej i współrzędne punktu głównego w postaci macierzy kamery K. Kalibracja została wykonana za pomocą skryptu w języku Python, który korzysta z biblioteki OpenCV [23]. Do kalibracji jest wykorzystywana szachownica o znanej wielkości kwadratów. Na rysunku 4.1 przedstawiono przykładowy obraz z kalibracji zawierający szachownicę i naniesione punkty kalibracyjne.

4.2 Kalibracja lidar-kamera

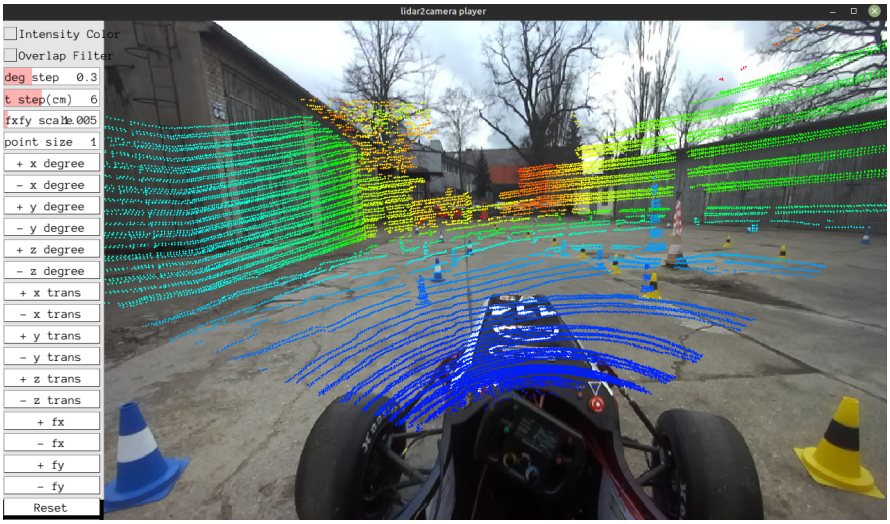
Kalibracja lidaru i kamery polega na określeniu macierzy transformacji z układu współrzędnych lidaru do układu współrzędnych kamery 2.3.1. W pracy zastosowano kalibrację dwuetapową z pierwszym etapem kalibracji manualnej i drugim automatycznej.

W pierwszej kolejności wykorzystano narzędzie do kalibracji manualnej LiDAR2Camera z biblioteki OpenCalib [24], które wizualizuje projekcję chmury punktów lidaru na obraz kamery oraz umożliwia użytkownikowi dynamiczną zmianę macierzy transformacji za pomocą przycisków interfejsu graficznego. Jako dane wejściowe jest wykorzystywana zsynchronizowana para chmury punktów w formacie pliku Point Cloud Data (PCD) [25] oraz zdjęcia w formacie PNG. Na rysunku 4.2 przedstawiono początkowy obraz interfejsu graficznego narzędzia do kalibracji manualnej, natomiast na rysunku 4.3 przedstawiono wizualizację projekcji chmury punktów lidaru na obraz kamery po manualnej kalibracji. Na każdy obraz jest nałożona projekcja chmury punktów z lidaru, kolorem niebieskim są oznaczone punkty bliskie, a czerwonym punkty dalekie. Kształty tworzone przez punkty zostały manualnie dopasowane do odpowiadających im obiektów na obrazie.

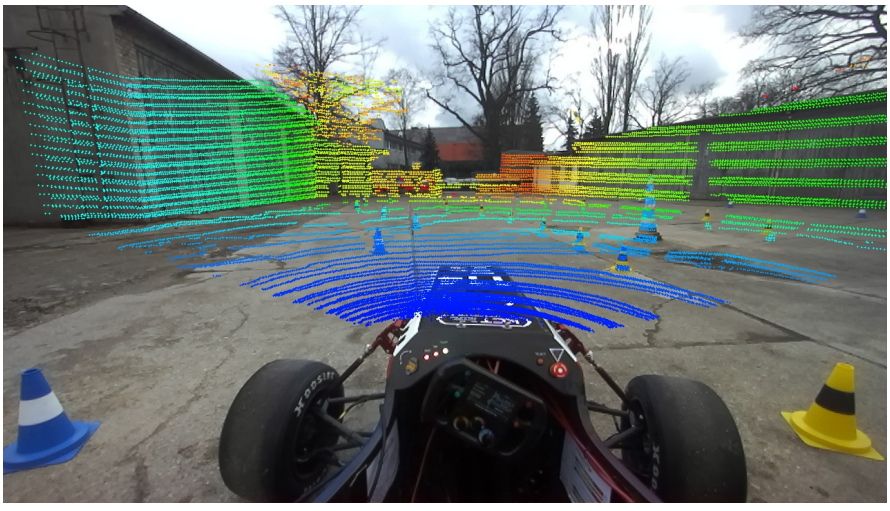
Uzyskaną w ten sposób wstępną macierz transformacji wykorzystano do automatycznej kalibracji za pomocą biblioteki CalibAnything [26]. Na tym etapie również wykorzystuje



Rysunek 4.1 Przykładowy obraz z kalibracji kamery



Rysunek 4.2 Przykładowy obraz z manualnej kalibracji lidaru i kamery

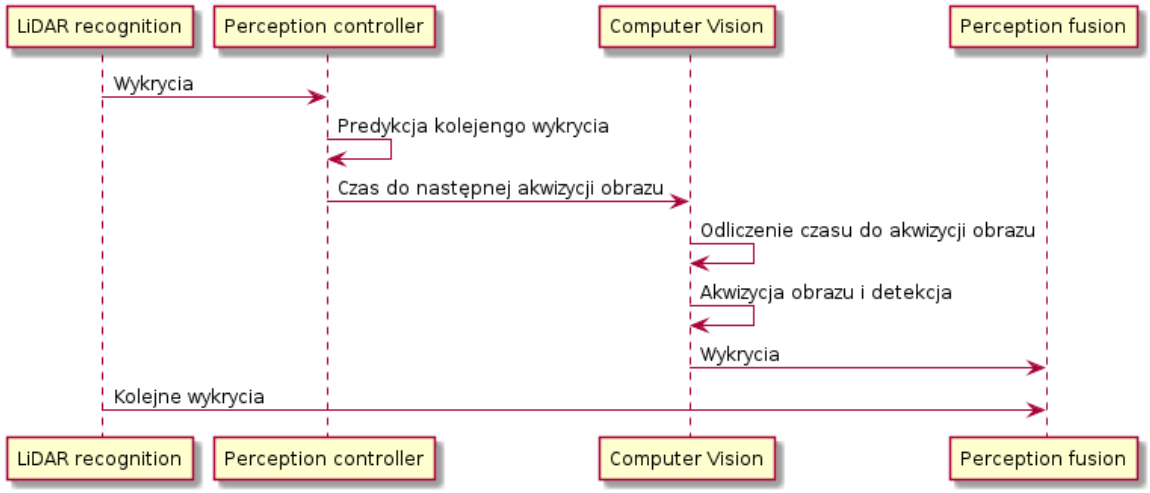


Rysunek 4.3 Projekcja chmury punktów lidaru na obraz kamery po manualnej kalibracji

Rysunek 4.4 Projekcja chmury punktów lidaru na obraz kamery z segmentacją przed automatyczną kalibracją

Rysunek 4.5 Projekcja chmury punktów lidaru na obraz kamery z segmentacją po automatycznej kalibracji. Czerwonymi elipsami zaznaczono obszary, gdzie różnica przed i po kalibracji jest najbardziej widoczna

się parę złożoną z chmury punktów w formacie PCD i zdjęcia PNG oraz dodatkowo maskę uzyskaną przez segmentację obrazu za pomocą biblioteki SegmentAnything [27]. Na rysunku 4.4 przedstawiono projekcję chmury punktów wraz z segmentacją na obraz kamery przed automatyczną kalibracją, natomiast na rysunku 4.5 po automatycznej kalibracji. Na obrazach widać projekcję chmury punktów dopasowanej do segmentacji obiektów. Tym samym kolorem oznaczone są punkty odpowiadające poszczególnym segmentacjom obiektów. Dodatkowo na rysunku 4.5 czerwonymi elipsami zaznaczono obszary, gdzie różnica przed i po kalibracji jest najbardziej widoczna.



Rysunek 4.6 Diagram sekwencji synchronizacji danych w kontrolerze percepcji

4.3 Kontroler percepcji

Głównym zadaniem węzła kontroler percepcji jest synchronizacja danych z lidaru i kamery, służy również jako watchdog, który nadzoruje działanie pozostałych węzłów percepcji oraz zarządza aktualnym stanem percepcji.

4.3.1 Synchronizacja danych

Synchronizacja danych z lidaru i kamery jest realizowana poprzez predykcję punktu w czasie, w którym dane z kamery mają zostać zebrane, co przedstawia diagram sekwencji na rysunku 4.6. W tym celu kontroler percepcji odbiera wykrycia pochodzące z węzła lidaru i uwzględniając fakt, że lidar działa z częstotliwością 20Hz, przewiduje czas, w którym zostanie zebrana następna chmura punktów z lidaru, a następnie określony czas jest wysłany do węzła kamery, gdzie za pomocą zegara jest odliczany czas do zebrania danych z kamery.

4.3.2 Zarządzanie stanem percepcji

Na potrzeby wstecznej kompatybilności z poprzednimi wersjami percepcji systemu autonomicznego bolidu RT14e, optymalizacji zużycia zasobów oraz dodatkowej redundancji, percepcja może działać w czterech trybach:

1. ”Fusion” – tryb fuzji danych z kamery i lidaru. W tym trybie kontroler percepcji synchronizuje dane z kamery i lidaru. Kamera wykrywa pachołki tylko z obrazu.

2. ”DepthCamera” – tryb tylko kamery. W tym trybie kamera działa w trybie stereoskopowym, dzięki czemu jest uzyskana mapa głębi, na podstawie której jest określana odległości pachołków od kamery. Węzły lidaru i fuzji są nieaktywne.

3. ”Lidar” – tryb tylko lidaru. W tym trybie zwracane są tylko wykrycia z lidaru. Węzły kamery i fuzji są nieaktywne.

4. ”Inactive” – tryb nieaktywny. W tym trybie żaden z węzłów percepcji nie jest aktywny.

Tryb działania percepcji jest określany przy użyciu programowego watchdoga bazującego na profilach QoS standardu DDS zaimplementowanego na podstawie biblioteki software_watchdog [28]. W przypadku, gdy węzeł percepcji nie wyśle wiadomości kontrolnej w określonym czasie, jest on uznawany za nieaktywny i kontroler percepcji biorąc pod uwagę aktywne węzły percepcji przełącza tryb percepcji do najlepszego możliwego trybu. Gdy nieaktywny węzeł percepcji stanie się ponownie aktywny również następuje przełączenie do aktualnie najlepszego możliwego trybu. Dodatkowo za pomocą parametrów można określić docelowy stan percepcji, który ma być osiągnięty. Pozwala to na działanie percepcji w trybie pojedynczych sensorów, nawet jeśli wszystkie węzły percepcji są aktywne. Przykładowo, gdy docelowym trybem będzie ”Lidar” i wszystkie węzły percepcji są aktywne, to kontroler wybierze tryb ”Lidar”, natomiast gdy węzeł lidaru stanie się nieaktywny, stan percepcji zostanie przełączony na tryb ”Inactive”.

4.4 Fuzja percepcji

Zadaniem węzła fuzji percepcji jest docelowa fuzja danych z kamery i lidaru. W zależności od konfiguracji może on wykorzystywać algorytm 3D->2D (zobacz rysunek 2.4) lub algorytm 2D->3D (zobacz rysunek 2.5).

4.4.1 Bazowa implementacja węzła

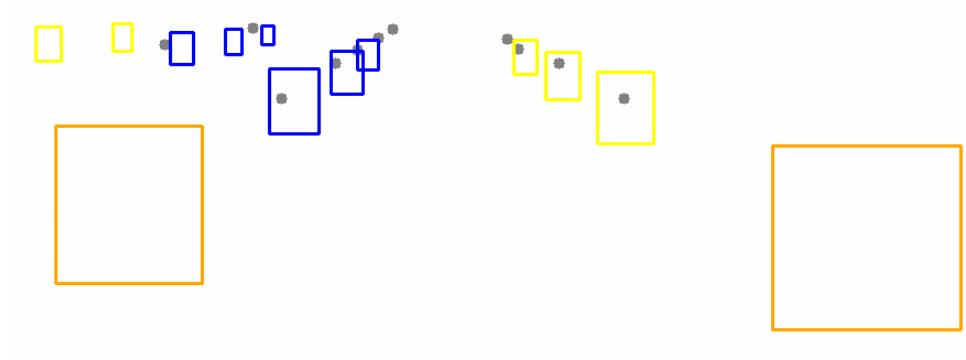
Węzeł fuzji percepcji nasłuchuje par tematów, które zawierają w zależności od używanego algorytmu fuzji danych, wykrycia pachołków pochodzące z węzłów kamery i lidaru lub wykrycia pachołków na podstawie obrazu kamery oraz nieprzetworzonej chmury punktów lidaru. Aby dane były odbierane jednocześnie użyto synchronizator, wykorzystujący politykę przybliżonego czasu[1] z biblioteki message_filters [29], który łączy zsynchronizowane dane i wywołuję wskazaną przez użytkownika funkcję, która jako argumenty przyjmuje dane z obu tematów.

Macierz kamery K oraz macierz transformacji z układu współrzędnych lidaru do układu współrzędnych kamery są przekazywane do węzła jako parametry. Do wczytywania macierzy kamery jest wykorzystywana biblioteka camera_calibration_parsers [30], która pozwala na wczytanie danych konfiguracyjnych kamery, w tym macierz kamery K, z pliku YAML. Macierz transformacji jest bezpośrednio przekazywana do węzła jako dwa parametry: macierz rotacji i wektor translacji. Implementacje obu algorytmów fuzji danych do obliczeń na macierzach i operacji związanych z przetwarzaniem punktów obrazu wykorzystują bibliotekę OpenCV [23].

4.4.2 Implementacja algorytmu z projekcją 3D na 2D

Implementacja algorytmu 3D->2D (zobacz rysunek 2.4) do wszystkich operacji, w skład których wchodzą projekcja danych, wizualizacja fuzji i docelowa fuzja, zostały zaimplementowane za pomocą blioteki OpenCV.

[1] ang. ApproximateTime Policy



Rysunek 4.7 Wizualizacja projekcji wykryć pachołków z lidaru na obraz kamery

Aby ułatwić sprawdzenie poprawności działania i ewentualne wykrycie błędów zaimplementowano wizualizację, której przykład przedstawiono na rysunku 4.7. Obraz zawiera jedynie białe piksele, na które naniesiono ramki ograniczające reprezentujące wykrycia z węzła kamery. Kolor każdej ramki odpowiada wykrytemu pachołkowi. Dodatkowo zaznaczono szare punkty przedstawiające projekcje detekcji z węzła lidaru.

Zaimplementowano również skalowanie ramek ograniczających, za pomocą konfigurowalnego parametru. Pozwala to dostosować algorytm do niedokładności wynikających z kalibracji lub synchronizacji, powodujących przesunięcia między detekcjami z węzłów lidaru i kamery.

4.4.3 Implementacja algorytmu z projekcją 2D na 3D

Implementacja algorytmu 2D->3D (rysunek 2.5) została podzielona na dwie części: pierwsza z nich to estymacja odległości pachołków od kamery na podstawie wielkości ramki ograniczającej a druga to docelowa implementacja algorytmu w węźle fuzji percepcji.

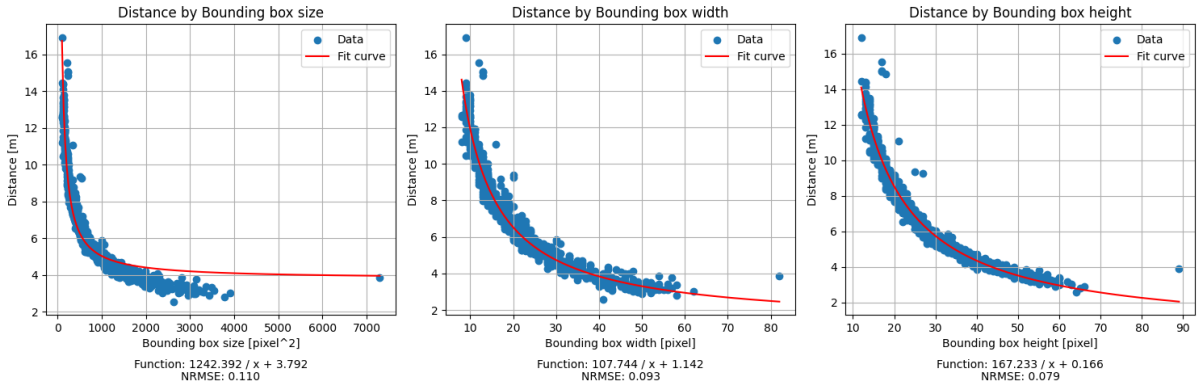
Estymacja odległości pachołków

Na potrzeby estymacji odległości pachołków zostały zaimplementowane dwa skrypty w języku Python. Pierwszy odbiera wiadomości z tematu węzła fuzji percepcji, który używa algorytmu projekcji 3D na 2D, a następnie dla każdego wykrycia pachołka zapisuje dane o odległości pachołka od kamery i wielkości ramki ograniczającej w postaci pliku CSV. Drugi skrypt wczytuje plik CSV i korzystając z biblioteki Matplotlib [31] wykreśla trzy wykresy odległości pachołków względem kamery odpowiednio w funkcji pola powierzchni, szerokości i wysokości ramki ograniczającej. Dla każdego wykresu jest wyznaczana krzywa najlepszego dopasowania za pomocą funkcji curve_fit z biblioteki SciPy [32] i zostaje wyliczony błąd dopasowania jako znormalizowany pierwiastek z błędu średniokwadratowego[2]. Ze względu na kształt funkcji, który przypomina hiperbolę. Zdecydowano, że funkcja najlepszego dopasowania jest opisana równaniem:

$$f(x) = \frac{a}{x} + b,$$

gdzie a i b są parametrami. Rysunek 4.8 przedstawia wykresy wraz z dopasowaną krzywą i wyliczonymi parametrami najlepiej dopasowanej funkcji oraz błąd dopasowania. Ze

[2]ang. NRMSE – Normalized Root Mean Square Error



Rysunek 4.8 Wykresy odległości pachołków względem kamery w funckji odpowiednio pola powierzchni, szerokości i wysokości ramki ograniczającej wraz z krzywą najlepszego dopasowania i wyliczonym znormalizowanym pierwiastkiem z błędu średniokwadratowego

względu na najlepsze dopasowanie i najmniejszy błąd dopasowania zdecydowano, że odległość pachołka od kamery będzie szacowana na podstawie wysokości ramki ograniczającej, funkcją najlepszego dopasowania z parametrami $a = 167.233$ i $b = 0.166$.

Implementacja algorytmu w węźle fuzji percepcji

Projekcję 3D na 2D zrealizowano za pomocą biblioteki OpenCV, natomiast do wszystkich operacji na chmurze punktów została wykorzystana biblioteka PCL [33]. Dalszy opis implementacji przedstawia szczegóły poszczególnych operacji algorytmu.

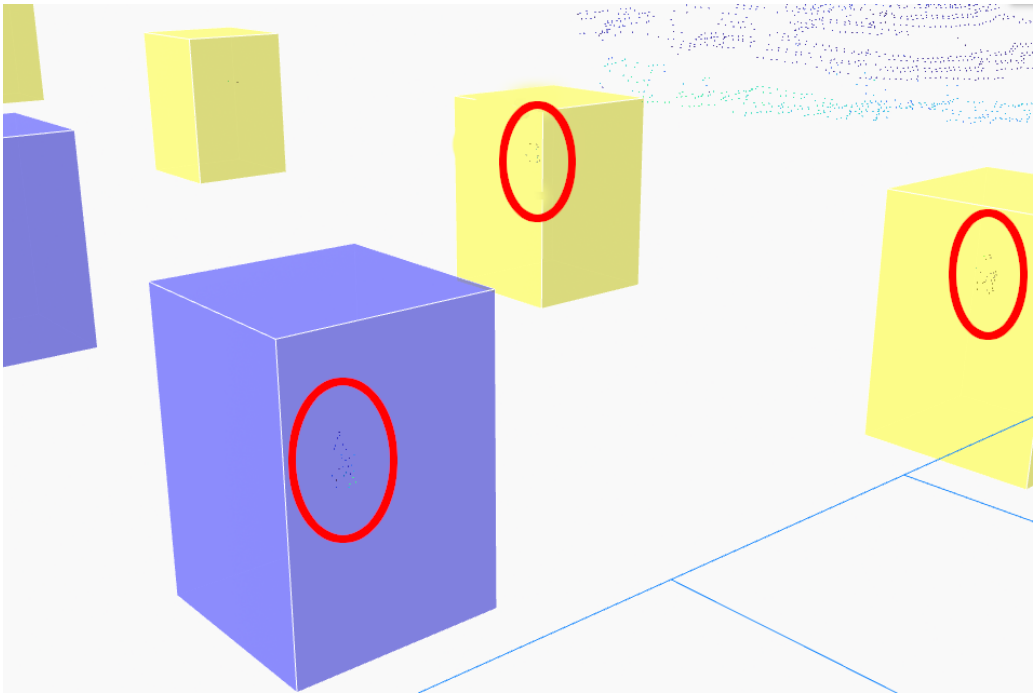
Filtracja chmury punktów składa się z dwóch operacji. Pierwszą jest ograniczenie przestrzenne za pomocą filtra prostopadłościennego[3], który usuwa punkty znajdujące się zbyt daleko lub będące poza polem widzenia kamery. Drugą operacją jest filtracja gruntu za pomocą algorytmu RANSAC [34].

Projekcje wykryć z węzła kamery są traktowane jako środki pachołków, na podstawie których jest tworzony prostopadłościan wielkości pachołka. Kolejno każda uzyskana bryła jest skalowana na podstawie konfigurowalnego parametru, co pozwala na dostosowanie algorytmu do niedokładności związanych z projekcją lub synchronizacją. Uzyskane prostopadłościany są następnie używane jako filtry prostopadłościenne na przefiltrowanej chmurze punktów. W ten sposób uzyskiwane są klastry pachołków.

Detekcja pachołków na podstawie klastrów sprowadza się do jednego testu, który sprawdza, czy dany klaster zawiera wystarczającą liczbę punktów. Ilość wymaganych punktów w klastrze jest konfigurowana za pomocą parametru. Jeśli detekcja się powiedzie, to wyliczana jest średnia pozycja punktów w klastrze, która jest traktowana jako współrzędne pachołka.

Oprócz głównej części algorytmu zaimplementowano również wizualizację jego działania. Przefiltrowana chmura punktów oraz uzyskane z projekcji wykryć prostopadłościany są wysyłane w osobnych tematach. Następnie w interfejsie graficznym Foxglove [35] ustawiając w widoku 3D obydwa tematy, można zobaczyć wizualizację działania algorytmu. Na rysunku 4.9 przedstawiono jej przykład. Czerwonymi elipsami zaznaczone zostały klastry punktów odpowiadające pachołkom.

[3] ang. Box Filter



Rysunek 4.9 Wizualizacja działania algorytmu projekcji 2D na 3D, czerwonymi elipsami zaznaczone zostały klastry punktów odpowiadające pachołkom

Rozdział 5

Testy

Dla porównania algorytmów zostały przeprowadzone następujące testy: określenie relatywnej dokładności wykryć na podstawie oceny wizualnej, pomiar całkowitej długość przetwarzania potokowego[1], pomiar wykorzystania zasobów podczas przetwarzania. Zostały one szczegółowo przedstawione w dalszej części rozdziału. Wszystkie testy przeprowadzono na module obliczeniowym opisanym w podrozdziale 3.4.1, wykorzystując nagranie z przejazdu testowego bolidu RT14e w formie rosbaga [36]. Na rysunku 5.1 przedstawiono wizualizację toru testowego bolidu RT14e, początek toru znajduję się z prawej strony.

5.1 Dokładność wykryć

W celu porównania dokładności wykryć zostały wykonane testy wizualne. Na potrzeby tego testu w obydwu algorytmach zaimplementowano publikację wykryć w postaci znaczników [37]. Wykrycia algorytmu 2D->3D są reprezentowane jako prostopadłościany, natomiast dla algorytmu 3D->2D zastosowano cylindry. Następnie zostały one wyświetlone w programie Foxglove. Na rysunkach 5.2, 5.3 i 5.4 zostały przedstawione kolejno wykrycia algorytmu 3D->2D, algorytmu 2D->3D oraz połączone wykrycia obu algorytmów.

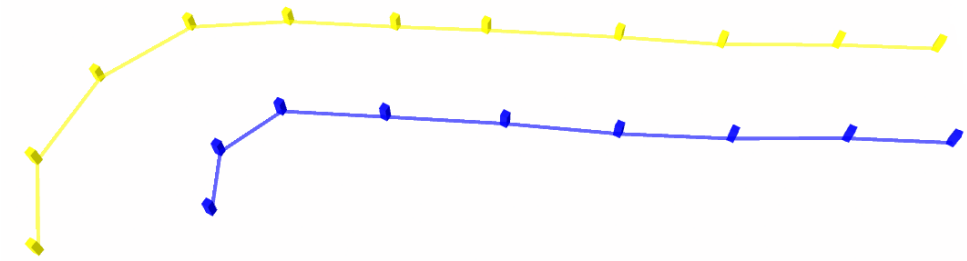
Wykrycia algorytmu 3D->2D w porównaniu do wykryć algorytmu 2D->3D są dokładniejsze, gdyż jest ich relatywnie mniej, jak również wykrycia są bardziej skupione wokół rzeczywistego pachołka, szczególnie jest to widoczne na zakręcie, gdzie pachołki są blisko siebie.

5.2 Długość przetwarzania potokowego

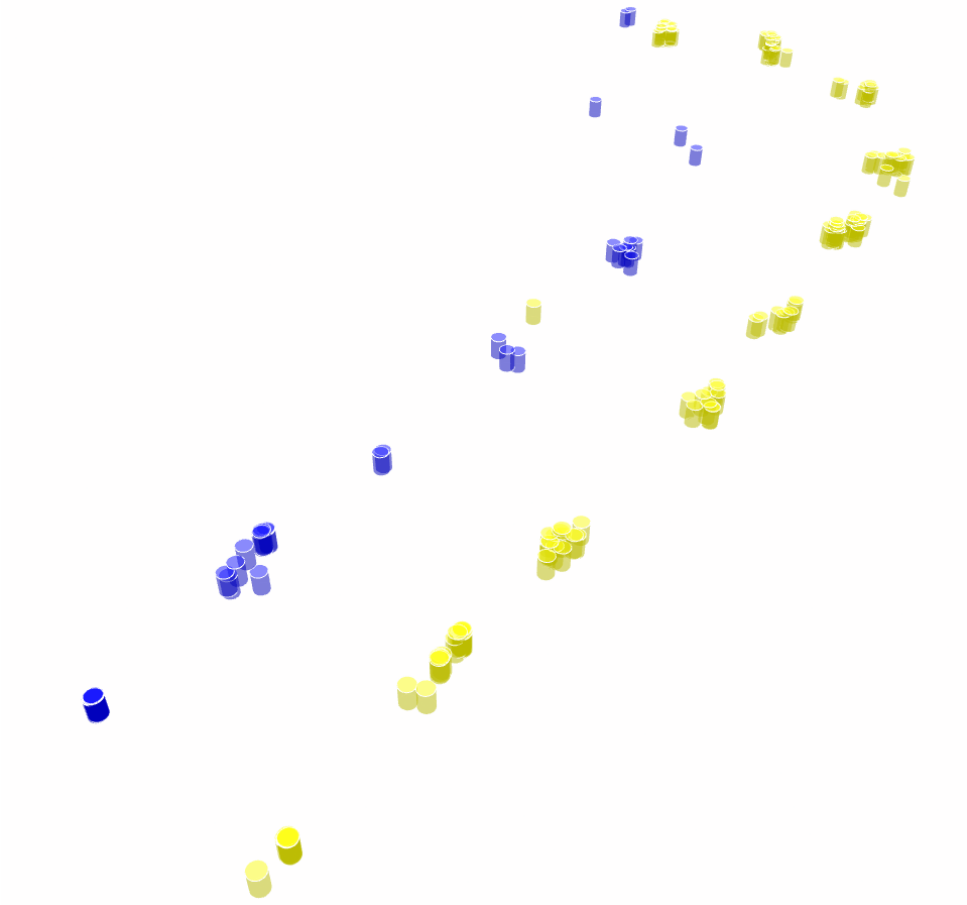
Długość przetwarzania potokowego jest mierzona jako czas od momentu otrzymania danych z węzłów kamery i lidaru lub sterownika lidaru do momentu publikacji przetworzonych danych w odpowiednim temacie. Na potrzeby tego testu zostały dodane dodatkowe rejestry zdarzeń[2] i niezbędne pomiary czasu w poszczególnych momentach potoku.

Dla algorytmu 3D->2D czas przetwarzania potoku został zmierzony dla węzłów lidaru i fuzji percepcji. W tym przypadku algorytm fuzji danych wymaga wykryć pochodzących z węzła lidaru. Wyniki pomiarów zostały przedstawione w tabeli 5.1. Jak widać średni czas przetwarzania potoku wynosi 34.77 ms. Dla algorytmu 2D->3D czas przetwarzania potoku

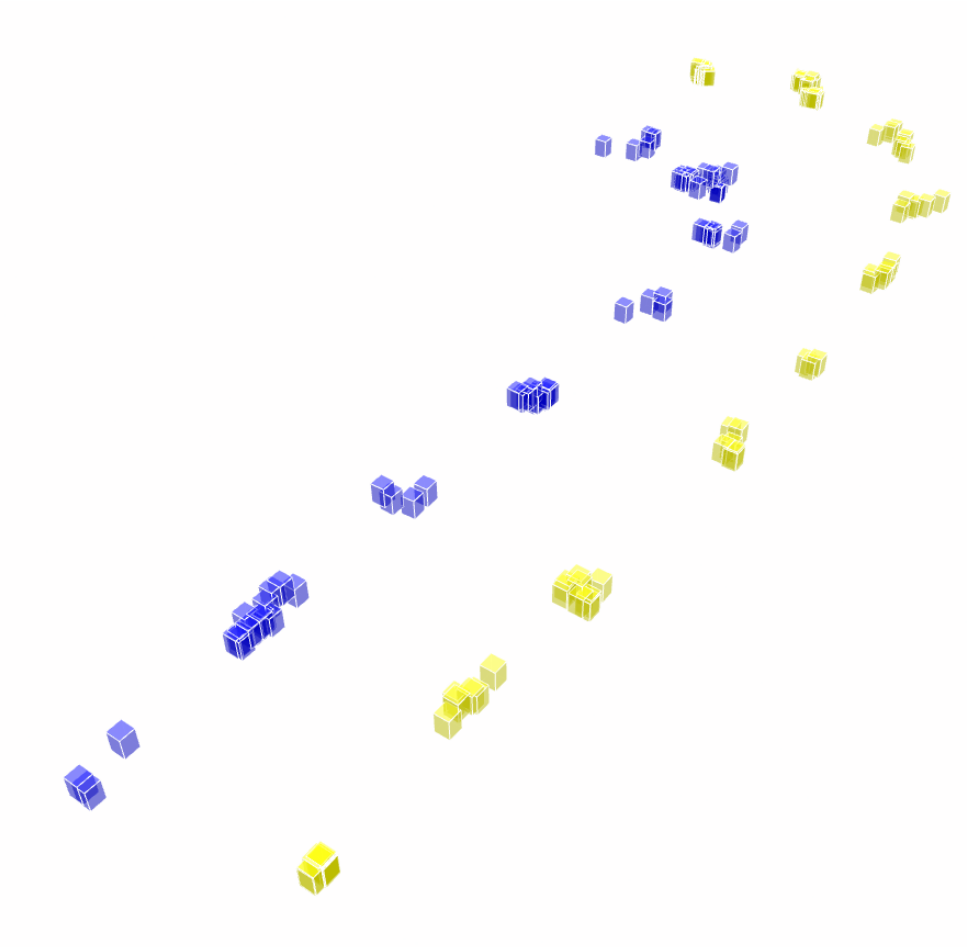
[1] ang. pipeline
[2] ang. logs



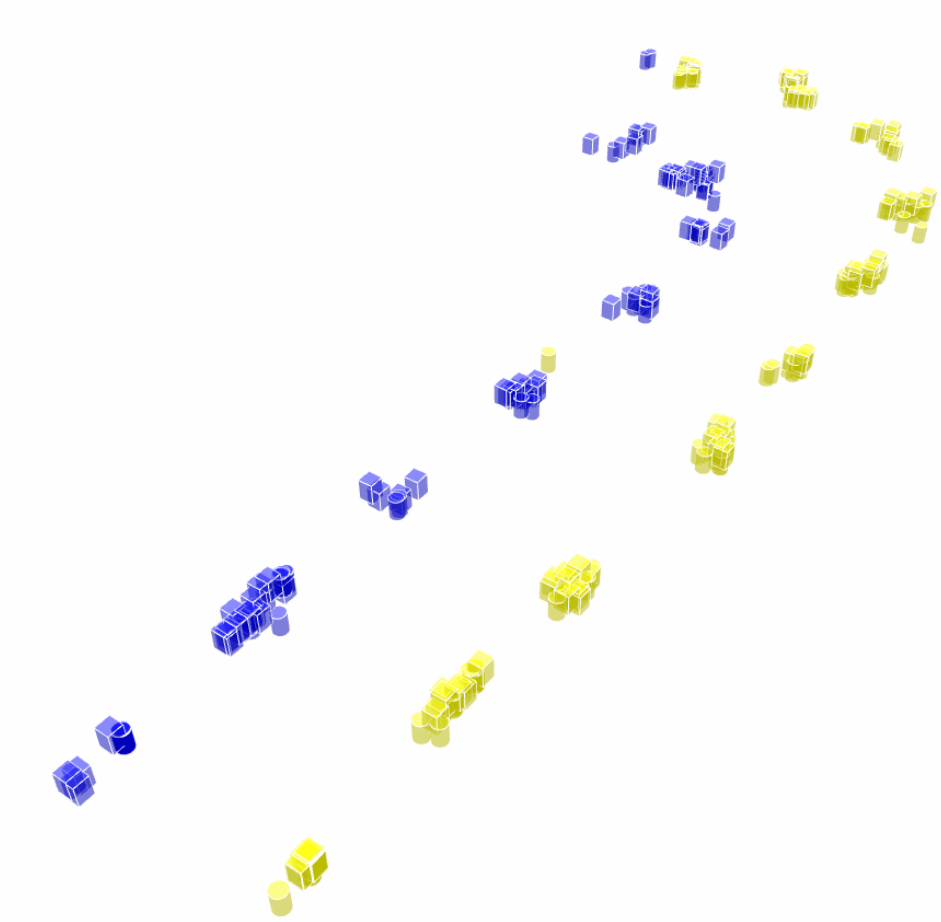
Rysunek 5.1 Tor testowy bolidu RT14e



Rysunek 5.2 Wykrycia algorytmu 3D->2D w programie Foxglove



Rysunek 5.3 Wykrycia algorytmu 2D->3D w programie Foxglove



Rysunek 5.4 Połączone wykrycia obu algorytmów w programie Foxglove

Czas [ms]	Min	Max	Średni
Węzeł lidaru	15.72	47.22	34.59
Węzeł fuzji percepcji	0.07	0.95	0.18
Suma	15.79	48.17	34.77

Tabela 5.1 Czas przetwarzania potoku dla algorytmu 3D->2D

Czas [ms]	Min	Max	Średni
Węzeł fuzji percepcji	3.49	11.65	5.19

Tabela 5.2 Czas przetwarzania potoku dla algorytmu 2D->3D

Średnie wykorzystanie	CPU [%]	RAM [MB]	GPU [%]	GPU MEM [MB]
Węzeł lidaru	37.22	584.52	3.05	88.75
Węzeł fuzji percepcji	0.71	443.33	-	-
Suma	37.93	1027.85	3.05	88.75

Tabela 5.3 Średnie wykorzystanie poszczególnych zasobów przez algorytm 3D->2D

Średnie wykorzystanie	CPU [%]	RAM [MB]	GPU [%]	GPU MEM [MB]
Węzeł fuzji percepcji	4.08	402.50	-	-

Tabela 5.4 Średnie wykorzystanie poszczególnych zasobów przez algorytm 2D->3D

został zmierzony tylko dla węzła fuzji percepcji, ponieważ w tym przypadku algorytm fuzji danych wymaga nieprzetworzonej chmury punktów z lidaru. Wyniki pomiarów zostały przedstawione w tabeli 5.2, gdzie średni czas przetwarzania potoku wynosi 5.19 ms. Co daje wynik około 6.7 razy szybszy niż w przypadku algorytmu 3D->2D.

5.3 Wykorzystanie zasobów

Wykorzystanie zasobów podczas przetwarzania potoku zostało zmierzone za pomocą dwóch narzędzi. Skryptu w języku Python, który używa biblioteki psutil [38] do pomiaru zużycia zasobów procesora (CPU) i pamięci RAM oraz programu jtop [39], który jest interfejsem do monitorowania zasobów komputera NVIDIA Jetson użytym do pomiaru zużycia zasobów karty graficznej (GPU). Pomiar GPU został wykonany tylko dla węzła lidaru, gdyż tylko on używa karty graficznej do obliczeń.

W przypadku algorytmu 3D->2D wykorzystanie zasobów zostało zmierzone dla węzłów lidaru i fuzji percepcji. W tabeli 5.3 zostały przedstawione wyniki pomiarów. W przypadku algorytmu 2D->3D wykorzystanie zasobów zostało zmierzone tylko dla węzła fuzji percepcji. W tabeli 5.4 zostały przedstawione wyniki pomiarów. Dla algorytmu 2D->3D wykorzystanie CPU jest około 9.3 razy mniejsze, a wykorzystanie RAM około 2.5 razy mniejsze niż w przypadku algorytmu 3D->2D. Dodatkowo wykorzystanie GPU jest zredukowane do zera.

Rozdział 6

Podsumowanie

Celem pracy jest zaprojektowanie i implementacja systemu percepcji otoczenia bolidu autonomicznego, który wykorzystuje kamerę wizyjną i lidar oraz porównanie efektywności zaimplementowanego systemu do metod i systemów powszechnie stosowanych w literaturze. Analiza porównawcza efektywności obejmowała zbadanie dokładności, szybkości działania oraz wykorzystania zasobów obliczeniowych systemu. Zaprezentowany w pracy nowatorski algorytm fuzji danych bazujący na projekcji 2D na 3D, został porównany z algorytmem używającym powszechnie stosowaną projekcję 3D na 2D. Obydwa algorytmy zostały zaimplementowane i przetestowane na bolidzie RT14e. Cel ten został zrealizowany.

W ramach pracy opisano percepcję otoczenia bolidu, niezbędny aparat matematyczny do implementacji algorytmów projekcji i proces fuzji danych sensorycznych pochodzących z lidaru i kamery wizyjnej. Przedstawiono system autonomiczny bolidu RT14e, jego architekturę oprogramowania, wykorzystywane środowisko programistyczne i sprzęt, z dodatkowymi szczegółami omówiono moduł percepcji. Ukazano procesy kalibracji kamery wizyjnej i kalibracji kamera–lidar oraz proces estymacji odległości pachołków na podstawie rozmiarów ramek ograniczających. Opisano implementację węzłów niezbędnych do poprawnego działania fuzji danych sensorycznych z kamery wizyjnej i lidaru, wykorzystujących różne metody projekcji. Przeprowadzono eksperymenty porównawcze, które wykazały, że algorytm wykorzystujący projekcję 2D na 3D jest bardziej efektywny pod kątem zużycia zasobów obliczeniowych oraz szybkości działania. Algorytm wykorzystujący projekcję 3D na 2D jest dokładniejszy.

W przyszłości warto rozważyć dalsze prace nad optymalizacją algorytmu wykorzystującego projekcję 2D na 3D, aby poprawić jego dokładność. Można również rozważyć zastosowanie segmentacji obrazu w celu detekcji pachołków na obrazie, co mogłoby poprawić dokładność szacowania odległości pachołków od kamery, jak również ogólną dokładność wykryć w projekcji 2D na 3D.

Literatura

[1] Formula Student Germany. Formula Student Germany – Official Website. https://www.formulastudent.de/fsg, 2025.

[2] Formula Student Germany. Formula Student Rules 2025, Version 1.1. https://www.formulastudent.de/fileadmin/user_upload/all/2025/rules/FS-Rules_2025_v1.1.pdf, 2025.

[3] Kieran Strobel, Sibo Zhu, Raphael Chang, Skanda Koppula, Accurate, Low-Latency Visual Perception for Autonomous Racing: Challenges, Mechanisms, and Practical Solutions. W: *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, strony 1969–1975, 2020.

[4] Víctor Moreno Borràs, *Enhancing cone detection in the perception system of a Formula Student team*. Praca dyplomowa licencjacka, Escola Tècnica Superior d'Enginyeria de Telecomunicació de Barcelona, Universitat Politècnica de Catalunya, 2024.

[5] Leiv Andresen, Adrian Brandemuehl, Alex Honger, Benson Kuan, Niclas Vödisch, Hermann Blum, Victor Reijgwart, Lukas Bernreiter, Lukas Schaupp, Jen Jen Chung, Mathias Burki, Martin R. Oswald, Roland Siegwart, Abel Gawel, Accurate Mapping and Planning for Autonomous Racing. W: *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, strony 4743–4749. IEEE, 2020.

[6] HaiLong Gong, YunJi Feng, TaiRan Chen, ZuoOu Li, YunWei Li, Fast and Accurate: The Perception System of a Formula Student Driverless Car. W: *2022 6th International Conference on Robotics, Control and Automation (ICRCA)*, strony 45–49. IEEE, 2022.

[7] Gustaf Broström, David Carpenfelt, *Robust Perception for Formula Student Driverless Racing*. Praca dyplomowa magisterska, Department of Automatic Control, Lund University, 2021.

[8] Formula Student Germany. Formula Student Germany 2025 Competition Handbook, Version 1.1. https://www.formulastudent.de/fileadmin/user_upload/all/2025/important_docs/FSG25_Competition_Handbook_v1.1.pdf, 2025.

[9] Satya Mallick. Geometry of image formation, Listopad 2024. https://learnopencv.com/geometry-of-image-formation/.

[10] Richard Szeliski, *Geometric primitives and transformations*, strony 36–60. Springer, wydanie drugie, 2022.

[11] Wojciech Rymer i Marcin Zawada, *Zastosowanie technologii CUDA w przyspieszaniu procesu klasteryzacji obiektów w przestrzeni euklidesowej*. Praca dyplomowa inżynierska, Politechnika Wrocławska, 2023.

[12] Sergio Garrido-Jurado, Rafael Muñoz-Salinas, Francisco J. Madrid-Cuevas, Manuel Jesús Marín-Jiménez, Automatic generation and detection of highly reliable fiducial markers under occlusion. W: *Pattern Recognition (ICPR), 2014 22nd International Conference on*, strony 4491–4496. IEEE, 2014.

[13] Steven Macenski, Tully Foote, Brian Gerkey, Chris Lalancette, William Woodall, Robot operating system 2: Design, architecture, and uses in the wild. *Science Robotics*, 7(66), 2022. https://www.science.org/doi/abs/10.1126/scirobotics.abm6074.

[14] Steve Macenski, Alberto Soragna, Michael Carroll, Zhenpeng Ge, Impact of ROS 2 Node Composition in Robotic Systems. *IEEE Robotics and Automation Letters*, 8(7):3996–4003, 2023.

[15] Tully Foote, tf: The transform library. W: *2013 IEEE Conference on Technologies for Practical Robot Applications (TePRA)*, strony 1–6, 2013.

[16] Rejin Varghese, Sambath M., YOLOv8: A Novel Object Detection Algorithm with Enhanced Performance and Robustness. W: *2024 International Conference on Advances in Data Engineering and Intelligent Computing Systems (ADICS)*, strony 1–6, 2024.

[17] Rahima Khanam, Muhammad Hussain. YOLOv11: An Overview of the Key Architectural Enhancements, 2024. https://arxiv.org/abs/2410.17725.

[18] NVIDIA Corporation. NVIDIA Jetson Orin. https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-orin/.

[19] NVIDIA Corporation. JetPack SDK. https://developer.nvidia.com/embedded/jetpack.

[20] Velodyne Lidar Inc., *Velarray User Manual*, wydanie rev. 2, October 2022.

[21] Stereolabs Inc. Technical Specification ZED 2i Stereo Camera, 2025. https://www.stereolabs.com/en-pl/store/products/zed-2i.

[22] Stereolabs Inc. Repozytorium Github zed-sdk, 2025. https://github.com/stereolabs/zed-sdk/tree/5.0.1_RC.

[23] G. Bradski, The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.

[24] Guohang Yan, Zhuochun Liu, Chengjie Wang, Chunlei Shi, Pengjin Wei, Xinyu Cai, Tao Ma, Zhizheng Liu, Zebin Zhong, Yuqian Liu, Ming Zhao, Zheng Ma, Yikang Li, Opencalib: A multi-sensor calibration toolbox for autonomous driving. 2022. https://arxiv.org/abs/2205.14087.

[25] Radu Bogdan Rusu, Steve Cousins, 3D is here: Point Cloud Library (PCL) - Point Cloud Data (PCD) Format. W: *IEEE International Conference on Robotics and Automation (ICRA)*, Shanghai, China, May 9-13 2011. IEEE. https://pointclouds.org/documentation/tutorials/pcd_file_format.html.

[26] Zhaotong Luo, Guohang Yan, Yikang Li. Calib-anything: Zero-training lidar-camera extrinsic calibration method using segment anything, 2023. https://arxiv.org/abs/2306.02656.

[27] Alexander Kirillov, Eric Mintun, Nikhila Ravi, Hanzi Mao, Chloe Rolland, Laura Gustafson, Tete Xiao, Spencer Whitehead, Alexander C. Berg, Wan-Yen Lo, Piotr Dollár, Ross Girshick, Segment anything. *arXiv:2304.02643*, 2023.

[28] Organizacja ROS Safety. Bilioteka software_watchdogs, 2025. https://github.com/ros-safety/software_watchdogs.

[29] Josh Faust, Vijay Pradeep, Dirk Thomas Jacob Perron. Biblioteka message_filters, 2025. https://wiki.ros.org/message_filters.

[30] Patrick Mihelich. Biblioteka camera_calibration_parsers, 2025. https://wiki.ros.org/camera_calibration_parsers.

[31] J. D. Hunter, Matplotlib: A 2D graphics environment. *Computing in Science & Engineering*, 9(3):90–95, 2007. https://doi.org/10.1109/MCSE.2007.55.

[32] Pauli Virtanen, i inni, SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020. https://doi.org/10.1038/s41592-019-0686-2.

[33] Radu Bogdan Rusu, Steve Cousins, 3D is here: Point Cloud Library (PCL). W: *IEEE International Conference on Robotics and Automation (ICRA)*, Shanghai, China, May 9-13 2011. IEEE. https://doi.org/10.1109/ICRA.2011.5980567.

[34] Martin A. Fischler, Robert C. Bolles, Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, 1981. https://doi.org/10.1145/358669.358692.

[35] Foxglove Inc. Foxglove studio. https://foxglove.dev, 2025.

[36] Open Source Robotics Foundation. rosbag2. https://github.com/ros2/rosbag2, 2025.

[37] Open Source Robotics Foundation. visualization_msgs/Marker.msg – ROS 2 message definition. https://github.com/ros2/common_interfaces/blob/humble/visualization_msgs/msg/Marker.msg, 2023.

[38] Giampaolo Rodola. psutil: Process and system utilities. https://pypi.org/project/psutil/, 2025.

[39] Raffaello Bonghi. jtop – Monitor and control Jetson devices. https://github.com/rbonghi/jetson_stats, n.d.