Detekcja obiektów na drogach dojazdowych do gruntów rolnych oraz polach uprawnych

Dokumentacja

Opiekun projektu:

Dr hab. inż. prof. Krzysztof Cabaj

Właściciel projektu:

Dr inż. Krystian Radlak

Skład zespołu:

Katarzyna Glaza

Natalia Glaza

Olga Krupa

Ewa Roszczyk

Spis treści

1. **Specyfikacja wymagań w konwencji MoSCoW**

1.Wymagania biznesowe

Celem jest opracowanie algorytmu detekcji obiektów pozwalającego wykrywać przeszkody takie jak drzewa, krzaki, czy też słupy elektryczne znajdujące się na polach rolniczych na podstawie informacji pozyskanych z sensorów głębi. Wynikiem projektu będzie algorytm umożliwiający automatyczną detekcję przeszkód w chmurze punktów pozyskanych z lidaru lub kamer stereowizyjnych.

Must have:

- System umożliwiający wykrywanie przeszkód na pozyskanych danych
- Przemyślany i przetestowany algorytm detekcji obiektów
- Rezultat działania detekcji w formie zwizualizowanych i zaznaczonych obiektów

Should have:

- Podstawowy program umożliwiający obserwowanie rezultatów działania systemu (program command line)
- Testy poprawności działania poszczególnych komponentów

Could have:

- Rozbudowana aplikacja do wyświetlania rezultatów działania systemu (wygodna w użytkowaniu, estetyczna)
- Przeniesienie rezultatów detekcji z plików w formacie .pcd do .jpg (aby umożliwić użytkownikowi subiektywną ocenę działania programu)

Won’t have:

- Systemów wspomagania hamowania w pojazdach rolniczych
- Działanie systemu na przygotowywanych na żywo danych z lidaru

2.Wymagania użytkowe

Must have:

- System umożliwiający wykrywanie przeszkód na pozyskanych danych

Should have:

- Rozbudowana aplikacja do wyświetlania rezultatów działania systemu (wygodna w użytkowaniu, estetyczna)

Could have:

- Przeniesienie rezultatów detekcji z plików w formacie .pcd do .jpg (aby umożliwić użytkownikowi subiektywną ocenę działania programu)

Won't have:

- System wspomagania hamowania w pojazdach rolniczych
- Działanie systemu na przygotowywanych na żywo danych z lidaru

Główną potrzebą użytkowników jest stworzenie takiego systemu, który umożliwi wykrywanie przeszkód na drogach rolnych lub dojazdowych. Zależy nam, aby napisać algorytm, który z dużą skutecznością i w możliwie krótkim czasie zlokalizuje przeszkody znajdujące się na drodze.

Efektywność algorytmu pozwoli na to, aby w przyszłości rozszerzyć zakres funkcjonalny - wykorzystać go do budowy systemów wspomagania hamowania w pojazdach rolniczych lub w pracach nad budową autonomicznego traktora.

3.Wymagania systemowe:

- funkcjonalne

Dystrybucja systemu operacyjnego GNU/Linux - Ubuntu

Pamięć RAM – zalecane minimum 11 GB (.bag - 5.2GB, dane .jpg - 1GB, dane .pcd - 3.8GB + aplikacja). Uruchomienie algorytmów na wszystkich danych z pliku .bag (+ 1GB + 3.8GB)

Karta graficzna - niewymagana

- pozafunkcjonalne

Niezawodność/dostępność:

System działa poprawnie, spełnia wszystkie powierzone mu funkcje, każda z nich została sprawdzona. Ponadto, jest on także odporny na wprowadzanie nieprawidłowych danych przez użytkownika (np. przy wprowadzaniu parametrów odcięcia chmur lub przy wyborze opcji z menu)

Wydajność/sprawność/efektywność:

Rozpakowywanie pliku .bag jest dosyć czasochłonnym procesem, zajmuje to kilkadziesiąt sekund. Jednak jest to jednorazowa czynność i należy ją wykonać jedynie za pierwszym razem, przy kolejnym użyciu jest to opcjonalne.

Ergonomia:

Program commandline'owy, który został stworzony pozwala na, zarówno uruchomienie pełnej funkcjonalności systemu (od początku do końca), jak i obserwowanie poszczególnych etapów działania: rozpakowywania danych, wyświetlanie chmur .pcd przed przetwarzaniem i detekcją oraz po. Użytkownik może wybrać, na jaką opcję się decyduje.

Modyfikowalność/rozszerzalność:

Obecny system pozwala na wykrywanie obiektów z chmur uzyskanych poprzez rozpakowanie danych z bazy Rellis-3D. Projekt można by rozszerzyć o analizę danych pobieranych na żywo tak, aby możliwe było wykorzystanie systemu na otwartej przestrzeni i w czasie rzeczywistym. Możliwe jest także optymalizowanie czasu trwania detekcji. Kolejnym udoskonaleniem, jakie można wprowadzić w przyszłości jest klasyfikowanie obiektów do odpowiednich kategorii (ludzie, drzewa, krzewy, pojazdy, barierki, płoty, budynki itp.). Parametry algorytmów (ransaca oraz dbscan) zostały dostrojone po przeprowadzeniu testów na danych z Rellis-3D. I tak, w zależności od dalszego wykorzystania systemu, można by go uwrażliwić (lub przeciwnie) na obiekty niewielkich rozmiarów, które mogłyby stanowić zagrożenie (wysokie trawy, kamienie, dziury niewielkich rozmiarów).

Przenośność:

W naszym systemie wykorzystano technologię ROS. W przypadku tej biblioteki Ubuntu Linux jest wymieniony jako “obsługiwany”, natomiast Microsoft Windows oraz macOS są oznaczone jako “eksperymentalne”. Z tego względu przeniesienie aplikacji na inny system operacyjny może okazać się problematyczne, jeśli nie niemożliwe.

Testowalność:

Algorytmy zostały przetestowane na przykładowych danych z bazy RELLIS – 3D. Wygenerowano także obrazki w formacie jpg, aby umożliwić subiektywną ocenę wykrytych przeszkód na chmurach, na których detekcja może okazać się problematyczna (w przypadku małych lub niedostrzegalnych obiektów). W taki sposób sprawdzono, czy wytworzony produkt spełnia założone cele i wymagania.

Ponadto zaimplementowano testy oceny poprawności działania algorytmów użytych w systemie. Sprawdzano między innymi poprawność wczytywanych plików, wykonywanych obliczeń, prawidłowe przyporządkowywanie punktów do konkretnych obiektów itd.

2. Definicja architektury

1.Kluczowe elementy struktury i ich interfejsy

Rellis-3D

RELLIS-3D to multimodalny zestaw danych dla robotyki terenowej. Został zebrany w środowisku terenowym zawierającym adnotacje do 13 556 skanów LiDAR i 6 235 obrazów (dane z różnych sensorów takich jak kamery RGB, lidary, obrazy stereowizyjne, IMU, GPS wraz z adnotacjami, dzięki czemu możliwa będzie automatyczna ewaluacja skuteczności opracowanego algorytmu detekcji przeszkód).

Robot Operating System

ROS - Robot Operating System (ROS) to zestaw bibliotek oprogramowania i narzędzi, które pomagają tworzyć aplikacje robotów.

Point Cloud Library

PCL - biblioteka algorytmów typu open source do zadań przetwarzania chmury punktów i przetwarzania geometrii 3D, na przykład w trójwymiarowej wizji komputerowej.

Algorytm RANSAC

RANSAC - iteracyjna metoda stosowana w celu estymacji parametrów poszukiwanego modelu matematycznego obiektu w zbiorze danych zawierającym znaczną ilość punktów nienależących do modelowanej powierzchni. RANSAC jest szczególnie interesujący dla celów przetwarzania chmur punktów obarczonych szumem i błędnymi pomiarami. Działanie algorytmu obejmuje zasadniczo dwie fazy: inicjalizację i test, obydwie powtarzane iteracyjnie.

Kolejne etapy postępowania można przedstawić jako:

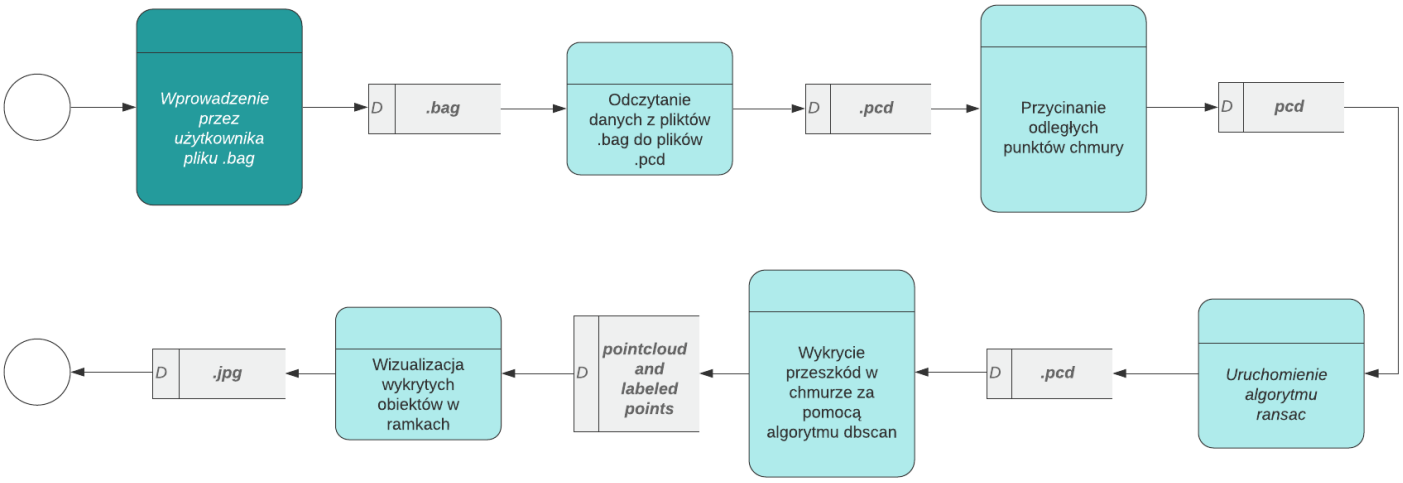
1. Wybór w sposób losowy minimalnego zbioru punktów, który pozwoli na wyznaczenie parametrów estymowanego modelu geometrycznego. Ponieważ w naszym problemie badamy przestrzeń trójwymiarową,
2. Porównanie pozostałych punktów z wejściowej chmury z utworzonym modelem (przyjęcie progu dopuszczalnego do uznania punktu za pasujący - ang. treshold).
3. Dodanie pasującego punktu do zestawu konsensusów (ang. inliers).

Powtarzanie procesu do momentu, gdy liczba punktów znajdujących się w chmurze pozwala sensownie przeszukiwać (dostrojenie ilości punktów, która spełnia ten warunek nastąpi podczas testowania i oceny rezultatów algorytmu).

Algorytm DB Scan

DB Scan - algorytm grupowania oparty na gęstości: biorąc pod uwagę zbiór punktów w jakiejś przestrzeni, grupuje razem punkty, które są blisko siebie upakowane (punkty z wieloma pobliskimi sąsiadami), oznaczając jako odstające punkty, które leżą samotnie w regionach o małej gęstości (których najbliżsi sąsiedzi są zbyt daleko). DB Scan to jeden z najpopularniejszych algorytmów klastrowania.

2.Interakcje pomiędzy elementami



3.Wyjaśnienie istoty przyjętych rozwiązań, odniesienie do wymagań

Przed wykonaniem detekcji na chmurach .pcd, dane odpowiednio przygotowano. Ponieważ algorytm może zostać w przyszłości wykorzystany do budowy systemów wspomagania hamowania w pojazdach rolniczych, należy go uwrażliwić na przeszkody znajdujące się w najbliższym otoczeniu. Również ze względu na dużą ilość punktów w chmurze wejściowej, zdecydowałyśmy się ją odpowiednio przyciąć - usunąć z niej pewien procent punktów najbardziej odległych od punktu środkowego - parametr ten jest opcjonalny. Chmurę poddano algorytmowi ransac - iteracyjnej metodzie szacowania parametrów modelu matematycznego ze zbioru obserwowanych danych, który zawiera wartości odstające. Można go również interpretować jako metodę wykrywania wartości odstających. Wykorzystanie ransaca było koniecznością - usunięcie z chmury części punktów znacząco wpłynęło na szybkość detekcji obiektów. Pominięcie tego kroku powoduje, że okres oczekiwania na rezultaty może potrwać do kilkudziesięciu minut.

4.Określenie podstawowych mechanizmów technicznych

- Sprzęt - wymagany komputer PC z dystrybucją systemu operacyjnego GNU/Linux - Ubuntu.
- Zainstalowany system ROS

Użyte biblioteki do implementacji:

Algorytm RANSAC: Pyransac3d

Algorytm DB Scan, wizualizacja chmur punktów, zapis do pliku, odczyt: Open3D

Pozostałe biblioteki: cv2, cv_bridge, glob, math, multiprocessing, numpy, os, time, random, re, sensor_msgs_msg, shutil, subprocess, sympy, sys, tkinter, typing, unittest

4. Specyfikacja projektowa

1.Określenie metod realizacji

- Języki programowania

Cały system został napisany w Pythonie.

- Frameworki

Robot Operating System (ROS), platforma typu open source

- Środowisko programowania, uruchamiania, testowania

Program commandline'owy uruchamiany w środowisku programistycznym IntelliJ IDEA/Visual Studio Code. Środowisko testowania – Visual Studio Code.

5. Podręcznik użytkownika

Instrukcja przygotowania systemu

Do odczytu danych wykorzystujemy system ROS. Umożliwia on za pomocą udostępnianych bibliotek prosty oraz szybki sposób odczytywania danych z pliku .bag

Wykorzystanie systemu ROS łączy się jednak z ograniczeniem działania programu do systemu Linux z zainstalowanym systemem ROS.

Aby móc korzystać z programu użytkownik musi najpierw posiadać zainstalowany system ROS – ros-noetic. Jest to dystrybucja systemu ROS dedykowana na system operacyjny Linux – Ubuntu 20.04. Wspiera on wersję Pythona – Python3.

Można tego łatwo dokonać za pomocą instrukcji udostępnionej na stronie systemu ROS - http://wiki.ros.org/Installation/Ubuntu

Dodatkowo przed uruchomieniem należy upewnić się, że wszystkie wymagane paczki z projektu są zainstalowane, w tym z ROS - pcl_ros oraz cv_bridge.

Wypisanie wszystkich paczek ROS dokonujemy za pomocą komendy: rospack list-names

Przy instalacji systemu ros-noetic pobierane są podstawowe paczki. W przypadku błędu braku jakiejś paczki możemy zainstalować ją własnoręcznie:

sudo apt install ros-noetic-nazwa-paczki

Po przygotowaniu oraz zainstalowaniu systemu ROS należy pobrać na komputer lokalny plik .bag z którego będą odczytywane chmury punktów 'sensor_msgs/PointCloud' oraz obrazy z kamery 'sensor_msgs/Image'.

W trakcie projektu korzystałyśmy z pliku .bag udostępnianego przez bazę RELLIS-3D https://github.com/unmannedlab/RELLIS-3D

W projekcie wykorzystałyśmy przykładowy plik .bag udostępniający 3 podstawowe informacje

„Synced data (60 seconds example 2 GB): includes synced /os1_cloud_node/points, /pylon_camera_node/camera_info and /pylon_camera_node/image_raw „

Plik zawiera 600 wiadomości z lidaru i oraz 600 wiadomości z kamer, co powoduje, że jest dużej wielkości - 5.2 GB.

Link do pliku .bag - https://drive.google.com/file/d/13EHwiJtU0aAWBQn-ZJhTJwC1Yx2zDVUv/view

Po zainstalowaniu i pobraniu pliku .bag możemy przystąpić do odczytu i działania na plikach.

Instrukcja użytkowania

Głównym celem projektu było przygotowanie algorytmu do detekcji. Aby wydobyć dane o chmurach punktów należy najpierw wydobyć je z pliku bag. Aby program poprawnie zadziałał i mógł wyeksportować dane z pliku .bag **konieczne jest uruchomienie systemu ROS**.

Docelowo system działa w środowisku Linux z uruchomionym systemem ROS. Aby możliwe było przedstawienie działania algorytmu została zbudowana aplikacja - program command line, która umożliwia uruchomienie całego algorytmu lub poszczególnych jego fragmentów. Aby móc korzystać z systemu ROS należy najpierw go uruchomić. Jeśli aktualnie system ROS nie jest uruchomiony, **należy otworzyć nowy terminal i wpisać komendę 'roscore'.**

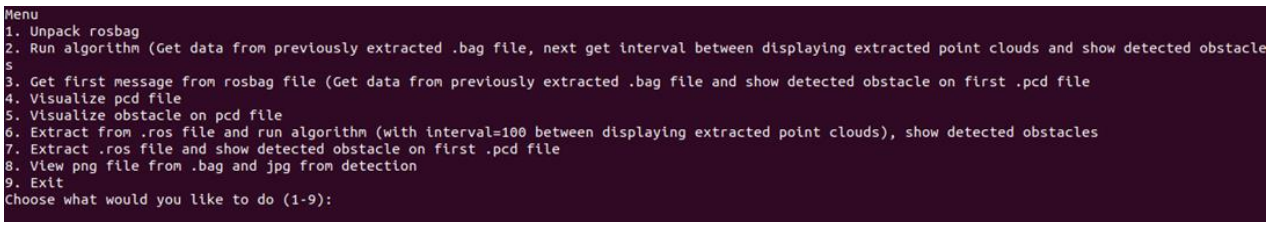
Następnie pozostawiając działający system ROS, możemy przejść do uruchomienia programu command line.

Następnie uruchamiamy terminal, przechodzimy do folderu z głównym plikiem *menu.py*, i za pomocą komendy **python menu.py** uruchamiamy program.

Program command line

Po uruchomieniu zostajemy zapytanie o docelową ścieżkę, w której będą zapisywane pliki. Domyślnie jest to aktualna ścieżka, w której uruchomiliśmy program, lecz można ją zmienić. Ustawiona ścieżka zostaje na stałe przepisana do programu, wszelkie pliki będą do niej zapisywane.

Następnie uruchamia się główny program, który dostarcza 8 funkcji:



Funkcje:

1.Unpack rosbag – pobranie od użytkownika lokalizacji pliku .bag, wypisanie danych znajdujących się w pliku oraz wypakowanie danych:

- chmury punktów zostają zapisane do folderu ./pointclouds

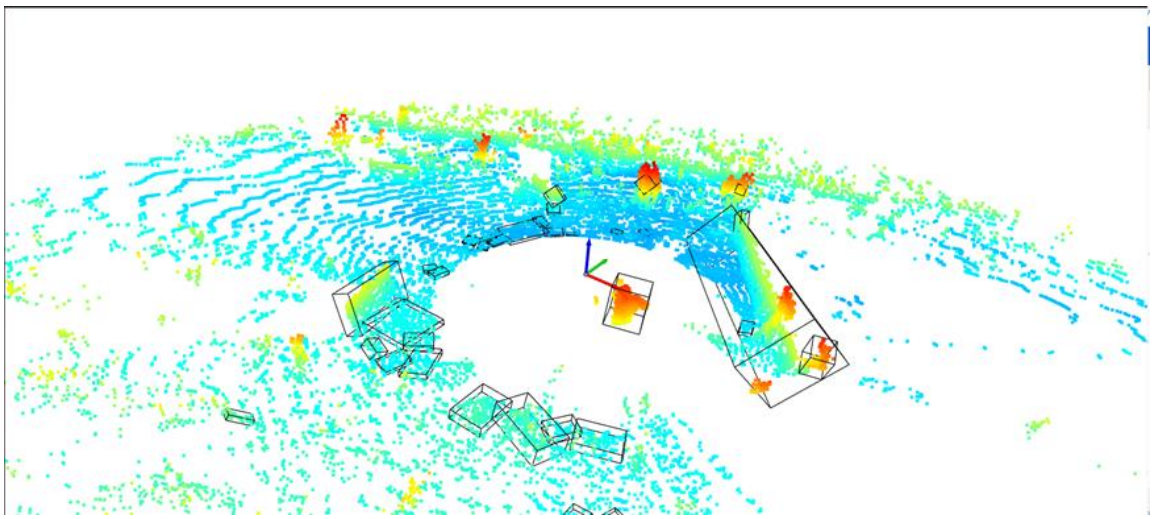
- obrazy z kamer zostają zapisane do pliku ./images

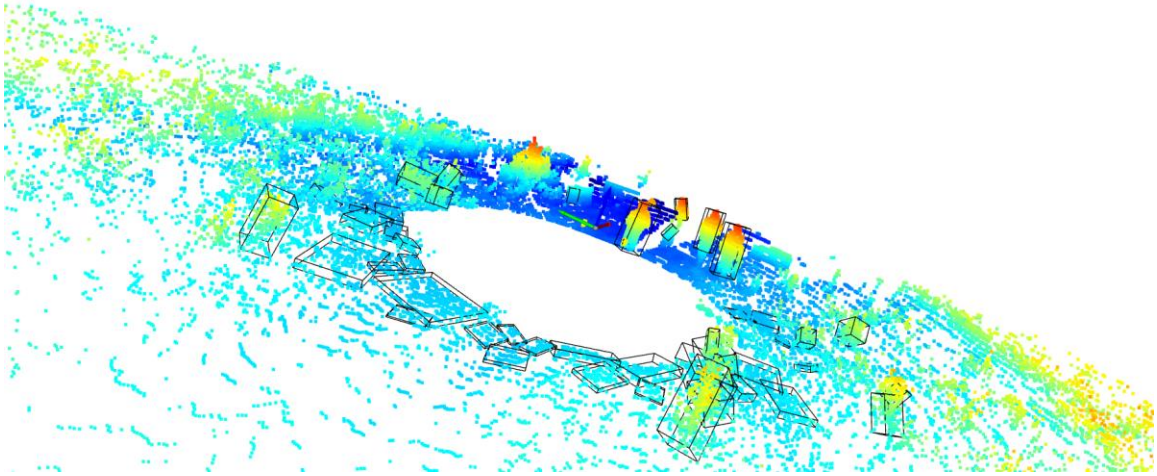
Jest to proces czasochłonny, ponieważ zostaje wypakowane ponad 4 GB informacji.

2. Run algorithm (Get data from previously extracted .bag file, next get interval between displaying extracted point clouds and show detected obstacles – jest to uruchomienie głównego algorytmu. Funkcja ta dokonuje detekcji punktów o zadanym interwale, tzn. analizowana jest co ‘x’ klatka/chmura punktów a następnie wyznaczane są przeszkody na drodze oraz zaznaczane na chmurze punktów. Funkcja dokonuje wizualizacji przeszkód.

Wewnątrz funkcji dokonujemy przycięcia chmury punktów(opcjonalnie), następnie zostaje ona poddana działaniu algorytmu Ransac. Pliki po algorytmie zapisywane są do folderu ransac_files. Kolejnym krokiem jest klasyfikacja punktów w chmurze do obiektów, a następnie na ich podstawie dokonujemy wyznaczenia przeszkód oraz ich wizualizacji.

Przykład wizualizacji:





3.Get first message from rosbag file (Get data from previously extracted .bag file and show detected obstacle on first .pcd file" – działa na tej samej zasadzie co wcześniejsza funkcja, dokonuje obliczeń tylko dla 1 chmury.

4. Visualize pcd file – opcja umożliwiająca wizualizację dowolnego pliku .pcd

5. Visualize obstacle on pcd file – detekcja obiektów oraz wizualizacja wprowadzonego pliku .pcd

6. Extract from .ros file and run algorithm (with interval=100 between displaying extracted point clouds), show detected obstacles" – opcja ma takie samo działanie jak opcja 2 – run algorithm. Różni się wcześniejszym ekstracji informacji z pliku .bag podanego przez użytkownika

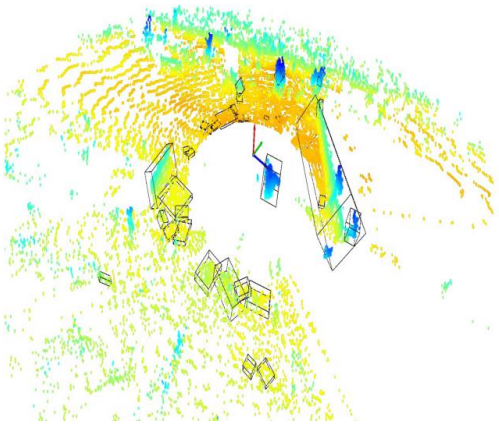
7. Extract .ros file and show detected obstacle on first .pcd file – takie samo działanie jak w punkcie 6 dla pierwszego pliku .pcd z pliku .bag.

8. View png file from .bag and jpg from detection – jest to funkcja umożliwiająca porównanie działania algorytmu detekcji z aktualnym zdjęciem z kamery. W trakcie wykonywania opcji nr 2,3,6 lub 7 zostaje przechwycony zrzut z ekranu z aktualnej wizualizacji przeszkód. Następnie możemy dokonać porównania wizualizacji przeszkód oraz zdjęcia w danej chwili z kamery.

Przykład porównania wizualizacji oraz aktualnego obrazu

Png file from .bag

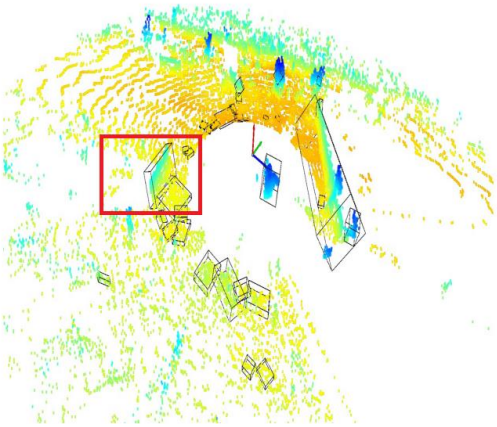
Screenshot from detecting obstacles



Dla ułatwienia lokalizacji murka ze zdjęcia został on ręcznie zaznaczony czerwoną ramką.

Png file from .bag

Screenshot from detecting obstacles



9. Exit – zamknięcie programu

6. Literatura, materiały pomocnicze

Projekt został zrealizowany w oparciu o poniższe materiały:

- Praca inżynierska Jakuba Dawida Barejki “Rozpoznawanie obiektów 3D”

Praca dyplomowa w dużym stopniu wpłynęła na wybór wariantu algorytmu ransac. Dokładna analiza pokrewnych wariantów oraz testy przyczyniły się do podjęcia finalnej decyzji odnośnie algorytmu.

- Ransac: https://en.wikipedia.org/wiki/Random_sample_consensus
- DB Scan: https://en.wikipedia.org/wiki/DBSCAN
- Artykuł o bazie RELLIS-3D: https://arxiv.org/pdf/2011.12954v2.pdf

Materiały pomocnicze przy implementacji

Dokumentacje bibliotek:

- Open3D: http://www.open3d.org/docs/0.12.0/index.html
- Pyransac3d: https://pypi.org/project/pyransac3d/