SPRAWOZDANIE Z PROJEKTU

Tworzenie systemów robotycznych

Temat: Roboty Taxi

Wykonane przez:

Miłosz Stasiak 240471

Filip Grzymski 240410

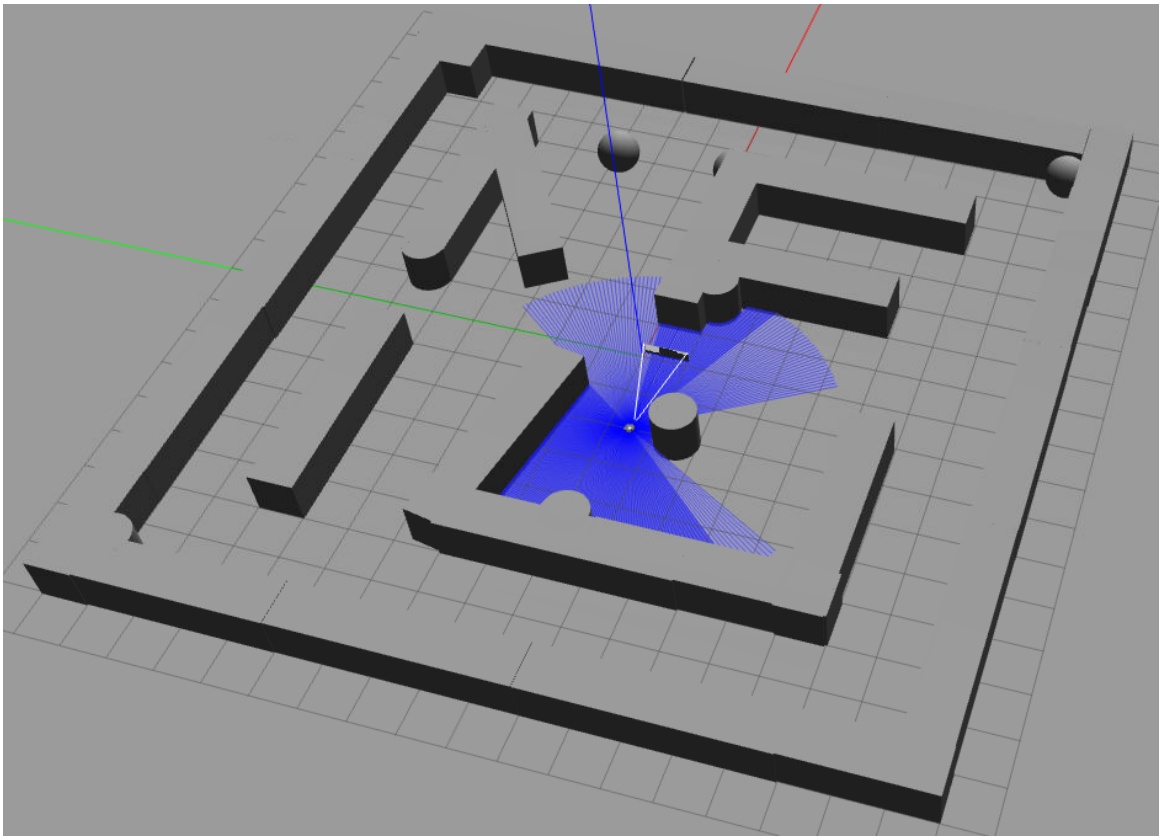
1 Opis

Stworzyliśmy system robotyczny, który zarządza robotyczną taksówką. System wspiera kolejkowanie zadań. Dodatkowo, wykonaliśmy mapę, po której robot porusza się w symulacji. Jej zadaniem jest odwzorowanie miejskiego środowiska, w którym robot mógłby się znaleźć w rzeczywistosci.

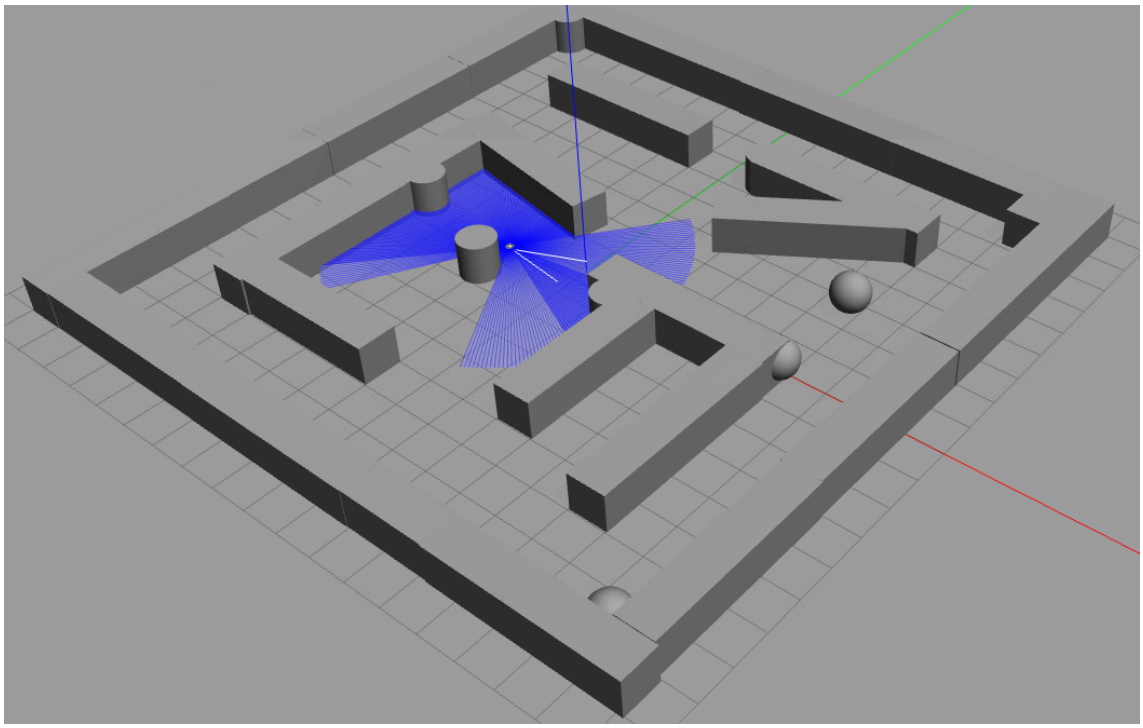
2 Mapa

Zbudowaliśmy mapę w Gazebo, a następnie uruchomiliśmy program `cartographer`, który wygenerował model mapy do Rvizz. Niestety ten model miał drobne błędy i nieregularności, więć użyliśmy programu do edycji zdjęć i wyczyściliśmy model do Rvizz. Warto zaznaczyć, że na początku zbudowaliśmy bardzo powtarzalną mapę z samych prostopadłościanów, ale robot na niej się gubił. Bez charakterystycznych elementów robot napotykając róg mapy mylił go z rogiem wgłebienia, przez oprogramowanie odpowiedzielne za skalowanie obiektów myliło się i wyolbrzymiało wgłębienia (niezależnie od tego czy widziało dwie ściany czy nie. Program uznawał, że wgłębiene jest na całą mapę). Długość poszczególnych bloków też ma znaczenie. Gdy robot jeździł wzdłuż ściany bez dodatkówych bodźców to znowu zaczynał nadinterpretowywać długość korytarza. Rozwiązaliśmy ten problem dodając dużo więcej różnych kształtów (punktów orientacyjnych). Możliwe, że taki probelm występuje tylko i wyłącznie w symulacji i przy wprowadzeniu robota do świata rzeczywistego, nie występowałboby takie zjawisko. W świecie rzeczywistym jednak występują takie "urozmaicenia"krajobrazu jak ławki, latarnie, różne kształy i odległości bloków.

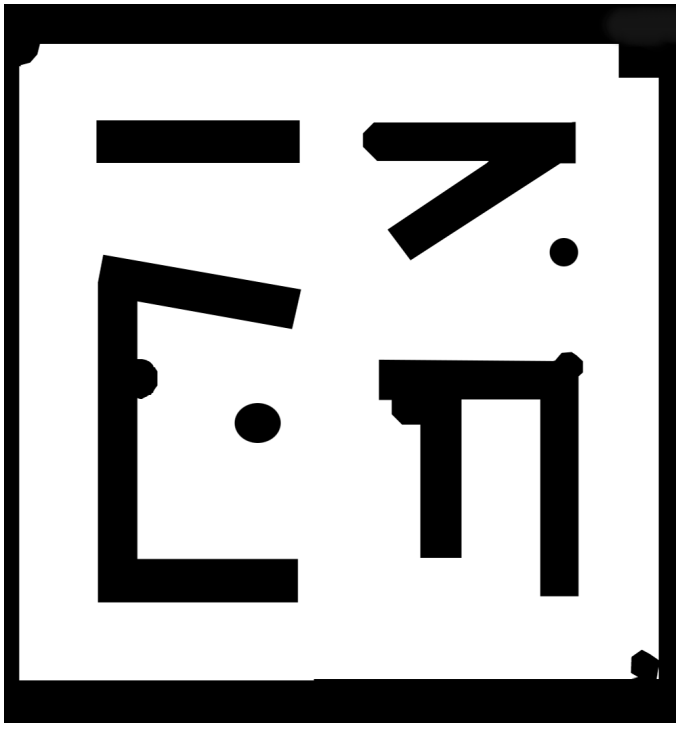
Ilustracja 1: Mapa prototyp



Ilustracja 2: Widok mapy w rzucie izometrycznym



Ilustracja 3: Inny rzut izometryczny



Ilustracja 4: Model mapy do nawigacji w Rvizz

3 Kod

3.1 Zasada działania

Nasz system składa się z dwóch programów, które można zaklasyfikować jako back-end i front-end. Pierwszy program komunikuje się bezpośrednio w robotem, a drugi program komunikuje się z użytkownikiem końcowym. Oba programy wymieniają się informacjami za pomocą wspólnej bazy danych w technologii "sqlite3".

Baza danych

Baza danych zawiera jedną tabele, której definicja w naszym narzędziu ORM jest poniżej. Każdy rekord w tabeli `schedules` odpowiada jednemu rozkazowi, który ma zostać wydany robotowi.

```
self.tables = {
    "schedules":
        BaseTable(name="schedules", fields=[
            IntegerFieldSQLite(name="id", primary_key=True),
            BoolFieldSQLite(name="is_completed", null=False, default=0),
            BoolFieldSQLite(name="in_progress", null=False, default=0),
            BoolFieldSQLite(name="timestamp", null=False),
            IntegerFieldSQLite(name="x", null=False),
            IntegerFieldSQLite(name="z", null=False),
            IntegerFieldSQLite(name="y", null=False),
        ]),
}
```

Odczytywanie następnego polecenia odpowiada się za pomocą kwerendy

```
SELECT * FROM schedules WHERE is_completed=0 AND in_progress=0 ORDER BY timestamp
```

Back-end

Wydawanie rozkazów typu `navigate_to_pose` zaimplementowaliśmy za pomocą timera, który co jedną sekundę wykonuje zapytanie do bazy danych. Baza danych odpowiada listą zadań, które robot ma wykonać, a program decyduje, które zadanie mu przydzielić. Po wykonaniu zadania robot zgłasza koniec akcji, co w programie powoduje wykonanie kodu, który dokonuje odpowiednich zmian w bazie danych.

```
class Taxi(Node):
def __init__(self):
    super().__init__('taxi')

    self.is_robot_busy = False

    self.publisher_ = self.create_publisher(Twist,
                                            'cmd_vel',
                                            10)



    self.timer = self.create_timer(
        1.0,
        self.read_db)

[...]

def read_db(self):
    navigate_action = ActionClient(
        taxi,
        NavigateToPose,
        '/navigate_to_pose')


    BASE_DIR = Path("/home/ubuntu/turtlebot3_ws/src/jupyter_notebooks")
    storage_obj = BaseStorage(BASE_DIR)
    with storage_obj as s:
        cursor = s.conn.cursor()
        rows = cursor.execute("SELECT * FROM schedules WHERE is_completed=0 AND
            in_progress=0 ORDER BY timestamp").fetchall()


    if len(rows) > 0 and not self.is_robot_busy:
        navigate_pose_goal = NavigateToPose.Goal()
        goal_pose = PoseStamped()
        goal_pose.header.stamp = self.get_clock().now().to_msg()
        goal_pose.pose.position.x = float(rows[0]["x"])
        goal_pose.pose.position.y = float(rows[0]["y"])
```

```
        goal_pose.pose.orientation.z = 0.0
        goal_pose.pose.orientation.w = 0.0
        navigate_pose_goal.pose = goal_pose

        self.goal_id = rows[0]["id"]

        goal_future_pose = navigate_action.send_goal_async(navigate_pose_goal,
            feedback_callback=self.print_feedback_goal)

        goal_future_pose.add_done_callback(self.received_task)


[...]

    def history_success(self, future):
    BASE_DIR = Path("/home/ubuntu/turtlebot3_ws/src/jupyter_notebooks")

    storage_obj = BaseStorage(BASE_DIR)
    with storage_obj as s:
        cursor = s.conn.cursor()
        rows = cursor.execute(f"UPDATE schedules SET is_completed=1, in_progress=0 WHERE
            id={self.goal_id}")
        s.conn.commit()

    self.is_robot_busy = False
```

Front-end

Aplikacja front-end pozwala użytkownikowi komunikować się z robotem. Aplikacja opiera się na interaktywnych komórkach Jupyter Notebook, w których użytkownik musi wpisać miejsce do, którego robot ma się udać. W rzeczywistosci mogłoby to wyglądać w następujący sposób. Użytkownik w aplikacji mobilnej wpisuje, gdzie się aktualnie znajduje, robot do niego przyjeżdża. Następnie po wejściu do taksówki użytkownik wpisuje w aplikacji dokąd chcę jechać, a robot taxi go tam zabiera.

```
def send_robot(x="0", y="0"):
    BASE_DIR = Path("/home/ubuntu/turtlebot3_ws/src/jupyter_notebooks")

    timestamp = int(time.time()*1000)

    storage_obj = BaseStorage(BASE_DIR)
    with storage_obj as s:
        timestamp = int(time.time()*1000)
        s.insert_row({"is_completed":False, "in_progress":False, "x":float(x),
            "y":float(y), "z":0, "timestamp": timestamp}, "schedules")
```

Ilustracja 5: Interfejs użytkownika