Mind the Gap: Autonomous Train Boarding for Mobile Robots

GitHub — Demos

Emma Topolovec
School of Computing
Binghamton University, State University of New York
Binghamton, NY, USA
etopolo1@binghamton.edu

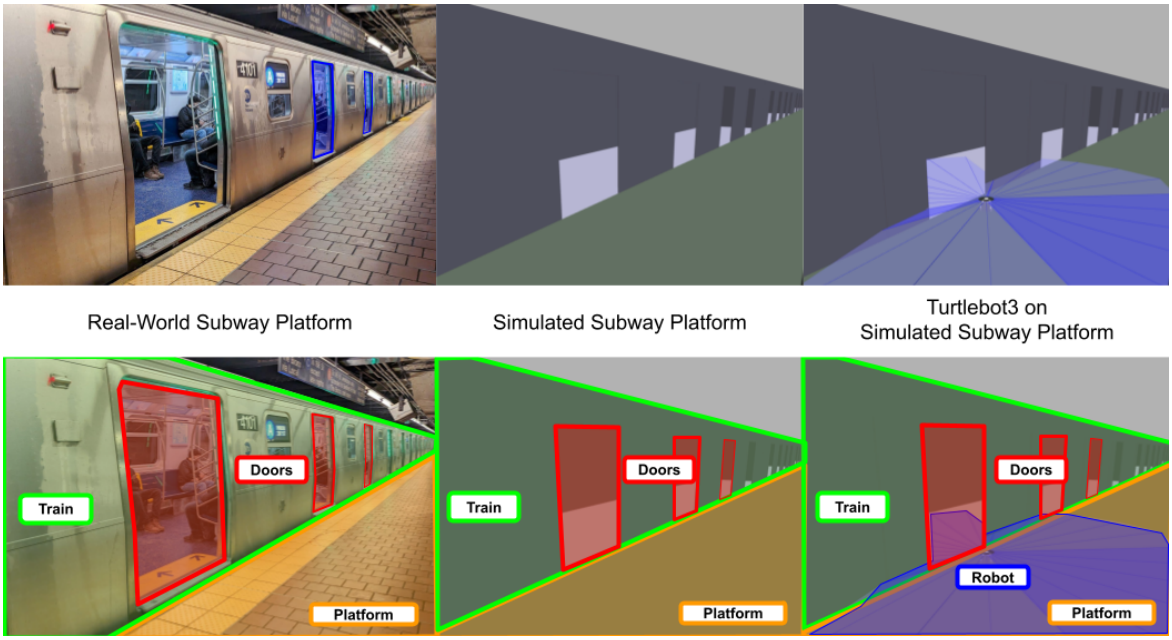

Fig. 1: Real World NYC Subway Platform vs. Simulated NYC Subway Platform

I. Introduction

Robots have the potential to be integrated into the rail industry. Prior research has explored how robots can perform various train-related tasks, including safety checks, basic repairs, and surveillance, but there is a lack of research integrating these tasks with trains. There is no existing method to enable a robot to board a train. Autonomous train boarding is necessary to perform tasks inside train cars or across stations. This project aims to jumpstart the research in autonomous train boarding and deboarding by introducing a novel Gazebo simulation of a train platform, showcased in Figure 1. This dynamic simulation models a real-world New York City Subway station and R211A train. It integrates with ROS and OpenAI Gym to enable a basic reinforcement learning loop. A Turtlebot3 model can move within the simulation to successfully board the train. This is a large initial step in the relatively unexplored applications of robots on trains. This simulation can be utilized in future research to train robots to safely, effectively, and efficiently navigate train platforms to board and deboard trains.

II. Background

A. Uses of Train-Boarding Robots

Robots can perform a variety of tasks. Many of these tasks could be useful onboard a train car. Others could be performed across multiple train platforms, as long as the robot can use an active train to autonomously move between stations.

1) Maintenance: Maintenance is essential to ensure safe and efficient travel. Mobile robots have been applied to com-

plete many maintenance tasks across different environments, from public buildings to power substations, successfully painting walls, performing quality assessments, and completing visual inspections [1]. In the context of trains, there are robots that can travel on rails to repair tracks [2] or gather data [3]. There are also wheeled robots that can inspect the bottom of a stationary train [4]. The quadruped robot ANYmal has been demonstrated to walk inside a train car to perform basic safety checks [5]. There is potential for growth in this research area by applying these existing methods on trains in service. Through boarding and deboarding trains, robots could complete a variety of automated tasks across several stations or train cars.

2) Delivery: Another possible use for train-boarding robots is delivery. In 2019, Amazon field-tested Amazon Scout, an autonomous package delivery robot [6]. As of 2025, Uber Eats has deployed mobile food delivery robots in several cities, including Jersey City, USA [7] and Tokyo, Japan [8]. DoorDash has introduced delivery robots to Chicago and Los Angeles [9]. In dense cities with substantial subway or train infrastructure, it may be possible to enable these robots to travel on trains as well as sidewalks. This could increase the range of robots and increase speed of delivery in some cases.

Hotels also have use for delivery robots for luggage. Some hotels have utilized a PFF Kilo robot for transporting guests' bags [10]. The AVRIDE Delivery Robot, which has been used for food delivery, also has the capability to carry luggage [11]. If these robots could board trains, they could improve accessibility or convenience for travelers who cannot independently carry all their bags.

3) Security: The NYPD recently tested a Knightscope K5 Security Robot to surveil subway stations [12]. This robot has also been successfully implemented in the Orange County Convention Center [13]. On actual NYC Subway cars, there were 2,745 violent crimes in 2024 [14]. If these robots could safely board active passenger trains, it may help reduce crime.

B. Gazebo Simulation Basics

From security to maintenance, a strong foundation exists for robot-train integration. In order for a robot to complete tasks on a train or across multiple stations, the tasks should be thoroughly tested in a simulated environment. Training in the real world is not practical, as trains move at dangerously high speeds. A simulated train platform is essential to enable robots to navigate onto trains safely.

Creating a simulation in Gazebo requires a few different components. First, each part of the environment requires a 3D model for its appearance and collision hitbox. C++ plugins can be attached to models to control their position and movement as well as communicate with ROS topics and services. Models must be inserted into a Gazebo world. When a world is launched, Gazebo's built-in physics engine can simulate realistic movements and collisions. There is a simulation clock, which consistently tracks the passage of time within a simulation regardless of runtime speed or the real-world time [15]. Learning these intricacies of Gazebo was essential in creating a high-quality simulation.

Fig. 2: New York City Subway Side Platform. Source: [19]

III. Case Study: New York City Subways

A. Subway Use

This project models a NYC Subway car and platform. This real-world environment was chosen due to a variety of reasons. First, it is one of the largest transit systems in the world with 472 subway stations and 3.6 million daily passengers [16]. There are a finite number of standardized platform layouts [17]. Each station requires complex maintenance tasks, including repainting surfaces, replacing tiles, and fixing benches [18]. There is also opportunity for delivery and security tasks to be completed on subway cars. If a NYC subway platform and car can be simulated, it should be possible to modify it to model many other rail networks.

B. Cars and Platforms

There are a few different NYC subway platform layouts and car models. For simplicity, this project models a side platform, which consists of one track and a waiting area against a wall, pictured in Figure 2. Every platform is typically hundreds of meters long, around 3-4 meters wide, and 1.15 meters above the tracks [20]. There are many car models in the current fleet, but the newest and one of the most common is the R211A [21]. This model was used as a reference for the simulated train. Each car is $18.35m$ long, $3.048m$ wide, $3.658m$ tall, and has 4 sets of $1.5m$ wide doors on each side. In service, trains typically consist of around 5 cars [22].

C. Real-World Physics

In order to minimize the sim-to-real gap, the simulation models the movement of a R211A car as accurately as possible. R211A cars travel at a max speed of $89km/h$, decelerate at $1.3m/s^2$, and accelerate at $1.1m/s^2$ [22]. Velocity and deceleration were properly simulated to model real-world physics.

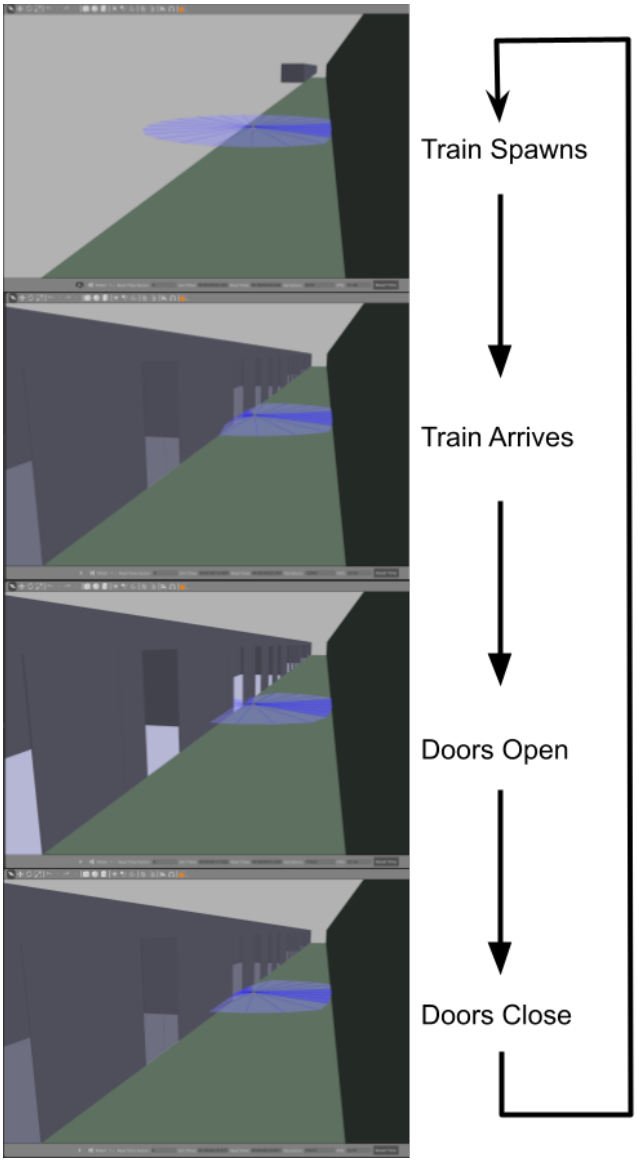

Fig. 3: Simulation Stages in Gazebo

IV. Approach

All of the code for this project can be found at https://github.com/EmmaTopolovec/MindTheGap. Some of the ROS topic names have been modified in this report to be easier to understand.

A. Simulation Timeline

The simulation consists of several phases. Initially, the train is far away from the platform. Next, the train starts with an initial velocity and decelerates until it reaches the platform. After reaching the platform, 3 seconds pass before the doors begin to open. The doors remain open for 15 seconds before closing. When the doors finish closing, the simulation concludes. This timeline enables training a robot to board the train. Figure 3 depicts the simulation stages in Gazebo.

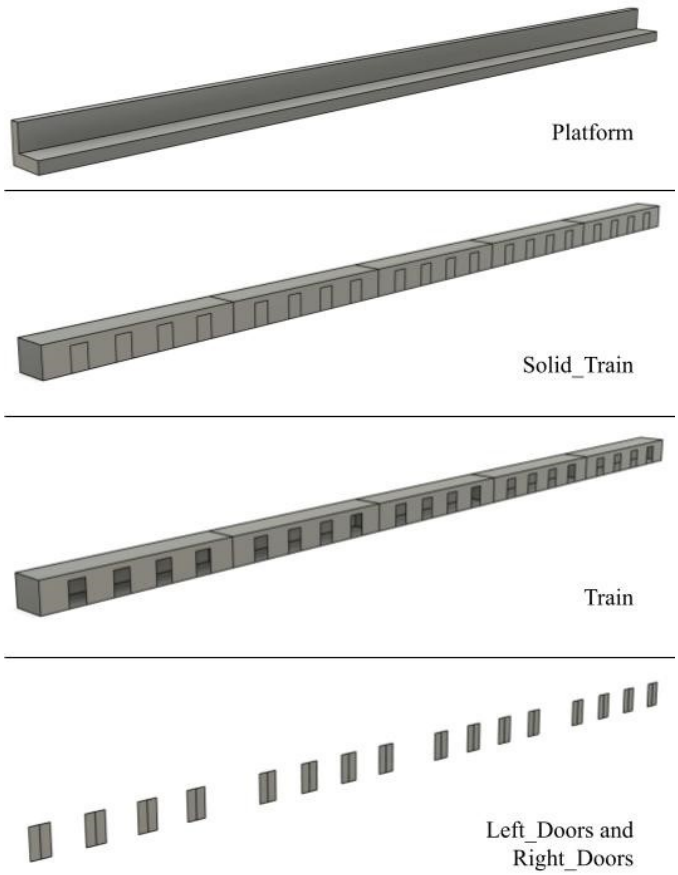

Fig. 4: Autodesk Fusion 3D Models

B. 3D Modeling

There are existing realistic 3D models of NYC R211A subway cars [23]; however, they are not free. As this is a course project, there is no budget to purchase existing models. Instead, I utilized Autodesk Fusion, a CAD software, to model the train and platform. In total, five models were created: $Platform$, $Solid\_Train$, $Train$, $Left\_Doors$, and $Right\_Doors$, pictured in Figure 4.

1) Platform: The platform model is L-shaped to represent a NYC Subway side platform. The wall of the platform is $3.658m$ tall, which is the height of an R211A. The wall could be raised to add a ceiling, but as this implementation only uses a 2D LiDAR sensor, a ceiling was not necessary. The floor of the platform is $1.15m$ tall, which is the height of NYC subway platforms above the tracks. The floor is $3m$ wide, which fits within the average NYC subway width of 3-4m. The platform is $91.75m$ long, which is the total length of a 5-car train.

2) Solid_Train: The train is split into two groups of models. The first is $Solid\_Train$. This model is a solid rectangular prism that does not have doors. The model is used when simulating the train arrival. As the doors will always remain closed while the train is moving, reducing the complexity of the model for this portion of the simulation increases performance and simplifies implementation.

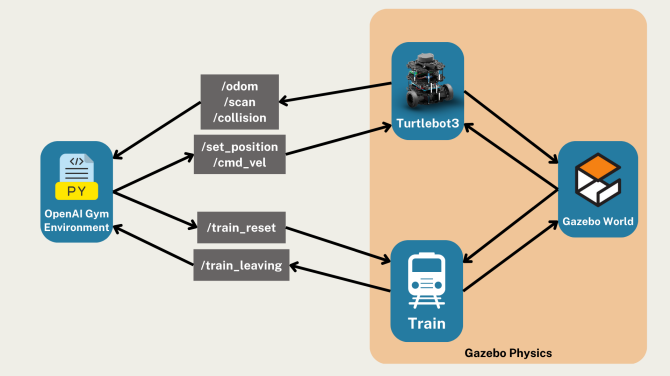

Fig. 5: Communication Diagram

I used the car dimensions I researched to create a rectangular prism representing each car. I duplicated the car to create 5 total cars. The rectangular prism is $91.75m$ long, $3.048m$ wide, and the top of it reaches a height of $3.658m$. The bottom $1.15m$ of the prism was removed, which is where the train wheels would be. Only the portion of the train that is at or above the platform floor is necessary for the simulation. Removing the bottom of the train helped reduce unwanted physics collisions with the platform.

3) Train, Left_Doors, and Right_Doors: The next group of train models consists of $Train$, $Left\_Doors$, and $Right\_Doors$. This group is used when the train is stationary and the doors are opening and closing. $Train$ is the same dimensions as $Solid\_Train$, but with cutouts for the doors. Each car has 4 cutouts on the platform-side, producing a total of 20 cutouts. Each cutout is $1.5m$ wide and $2m$ tall. $Left\_Doors$ and $Right\_Doors$ each consist of 20 doors, where each door is $0.75m$ wide and $2m$ tall. The doors are placed up against the train and slide open. The $Solid\_Train$, $Train$, $Left\_Doors$, and $Right\_Doors$ 3D models are combined into one Gazebo model file, which is simply called $Train$.

The placements of the models inside the Gazebo simulation can be seen in Figure 3. The first image depicts the $Solid\_Train$ approaching the $Platform$. The other images all depict various stages of the $Train$, $Left\_Doors$, and $Right\_Doors$ at the $Platform$.

C. Gazebo World Setup

To create the Gazebo world, I forked the $turtlebot3\_gazebo$ package [24]. I created a new world file that contained the platform and train models. I created a new launch script that also spawns a Turtlebot3 and has the ability to run with or without the Gazebo GUI. I coded custom Gazebo plugins in C++ for the train and Turtlebot3. The plugins subscribe and publish to various ROS topics, seen in Figure 5.

D. Turtlebot3

In the forked $turtlebot3\_gazebo$ package, I modified the Turtlebot3 Burger model slightly. I reduced the LiDAR sensor to sample 24 points as opposed to the default 360 in order to improve performance. I added a custom collision plugin that detected collisions and published them to the ROS topic $/turtlebot3/collision$. I added another plugin that subscribes to the ROS topic $/turtlebot3/set\_bot\_position$ so external scripts can reset the position of the robot.

The Turtlebot3 has several built-in plugins that the simulation takes advantage of. The robot is subscribed to the $/turtlebot3/cmd\_vel$ topic, which enables an external script to set the velocity of the robot wheels. The robot publishes to $/turtlebot3/odom$ and $/turtlebot3/scan$, which each contain the position and LiDAR data of the robot, respectively.

E. Train Plugin

I created a separate plugin for the Train model that does most of the work to control the simulation timeline. The plugin subscribes to ROS topic $/train/reset$, which resets train movement, moves the $Solid\_Train$ to its initial position, and hides the $Train$, $Left\_Doors$, and $Right\_Doors$. It also subscribes to topic $/train/start$, which begins the simulation by setting the initial velocity of the $Solid\_Train$. The plugin then updates every simulation tick and reduces the velocity by $1.3m/s$. When the $Solid\_Train$ reaches a velocity of 0, it is swapped out for the $Train$, $Left\_Doors$, and $Right\_Doors$, which then open slowly, wait, then close slowly. Once the doors close, the plugin publishes to topic $/train/leaving$, signifying that the simulation has concluded.

F. OpenAI Gym Environment

The OpenAI Gym Python library was used to implement reinforcement learning in the simulation [25]. A custom Gym environment was created that subscribes or publishes to the ROS topics used by the Gazebo models. This method is extremely basic and consists of the following components:

1) Environment
   - Gazebo world with train and platform models
2) Agent
   - Turtlebot3 Burger
3) Observation Space
   - 24 LiDAR Readings
   - Odometry (x, y, z, and angle)
4) Action Space
   - Move Forward
   - Turn Left
   - Turn Right
   - Stop
5) Reward Function

G. Reward Function

A basic reward function was implemented. Every step, the reward is increased if the robot is facing the train tracks, moves closer to the train, or boards the train. The reward is decreased if the robot is not facing the tracks, moves further from the train, or the LiDAR sensors detect an object within 0.3m. The reward also decreases if the robot collides with anything, falls

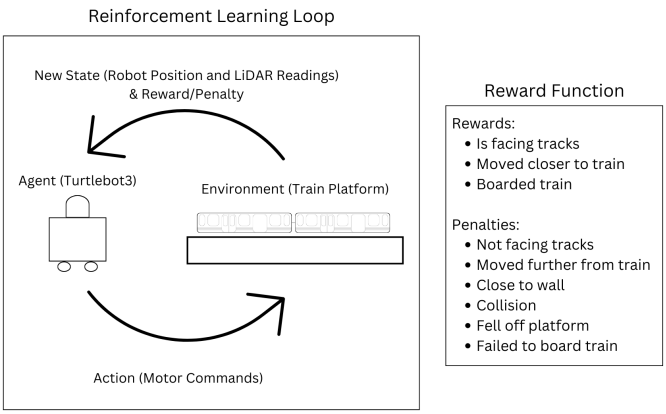

Fig. 6: Reinforcement Learning Loop and Reward Function

off the platform, or fails to board the train before the doors close.

A reset function is provided that publishes to the various ROS topics to reset the train and Turtlebot3. A reset is triggered whenever the robot detects a collision, falls off the platform, or when the train doors close. With this function, a basic reinforcement loop is implemented using Proximal Policy Optimization from the stable-baselines3 Python library [26]. The entire OpenAI Gym environment is basic and a proof of concept. It should be expanded upon in future work.

Figure 6 showcases the training loop of the Gym environment. A video of the training in action can be found at this link (titled "RL Training.mov").

Experiment	Successes	Failures	Success %
Train Arrival	10	0	100%
Turtlebot3 Boarding via TeleOp	10	0	100%
Turtlebot3 Boarding via Reinforcement Learning	0	10	0%

TABLE I: Simulation Experiment Results

V. Evaluation

To evaluate the reliability of the simulation, three experiments were performed. The first tested the train physics and simulation loop without the Turtlebot3. The second introduced the Turtlebot3 and used teleoperated commands to board the train. The last experiment tested the reinforcement loop. Videos of all experiments can be found at this link (titled "Train Arrival Tests.mov", "TeleOp Tests.mov", and "RL Tests.mov", respectively). Each video shows the 10 trials performed for each experiment. Table I showcases the results of all experiments.

A. Train Arrival

To evaluate the reliability of the simulation, the train arrival was tested. The Turtlebot3 robot was excluded from the Gazebo simulation for these tests. The only models present were the platform and train. The simulation was run and reset 10 times. The simulation was marked as a success if the train model was able to arrive at $x = 0$ and the doors were able to smoothly open and close. The simulation was marked as a failure if any unexpected behavior was visually observed.

The simulation worked as intended for all 10 trials. In each trial, the simulation appeared to run identically. The train physics are expected to run flawlessly 100% of the time.

B. Turtlebot3 Boarding via TeleOp

In order to ensure that collisions with the Turtlebot3 were working properly, 10 trials were performed using teleoperation. The Turtlebot3 was added to the simulation with a static starting position in front of a train door. Each trial reset the simulation and waited for the train to arrive and the doors to open. The Turtlebot3's forward velocity was set for one second, then reset to zero. A trial was marked as a success if the robot was on the train when the doors closed. A trial was marked as a failure if the robot fell off the platform, had a collision, or was not on the train when the doors closed.

After 10 trials, the robot boarded the train 10 times. This shows that robotic train boarding is possible. The trials appeared to be identical, leading to the conclusion that a teleoperated robot should be able to reliably board the train. In the video of this experiment, the robot is seen wobbling after stopping. This is due to the high forward velocity used in this test. This behavior would not be concerning on its own; however, the wobbling continued after each reset. This shows that the simulation logic that resets the Turtlebot3 was not properly resetting the velocity of the model. I looked into this issue and was able to solve it before the reinforcement learning tests.

This experiment also showed that a wheeled robot can "mind the gap". In the Gazebo simulation, there is a small gap between the platform and the train. The Turtlebot3 may have been successful because its wheels were significantly wider than the gap. It may be difficult for smaller robots to bridge this gap.

C. Turtlebot3 Boarding via Reinforcement Learning

To evaluate my reinforcement learning loop, I trained a model using the Proximal Policy Optimization algorithm for 100,000 timesteps. The robot ran through approximately 100 simulation loops. The trained model was tested 10 times to see if the robot could board the train before it departed.

Unfortunately, the robot failed to board the train every time. During training, the robot was able to occasionally board the train successfully, but it did not properly learn that behavior. Instead, the robot learned to stay completely still to reduce the chance of hitting the train or falling off the platform.

To increase the success rate, more time could be spent tuning the reward function or increasing the time spent training; however, I believe the main issue is a bug in the Python script that causes inefficiencies while training. Due to issues with synchronization, the Gym environment takes many steps after the robot falls off the platform or collides with the train. These steps do not give the proper penalty, preventing the model from learning not to fall or hit the train.

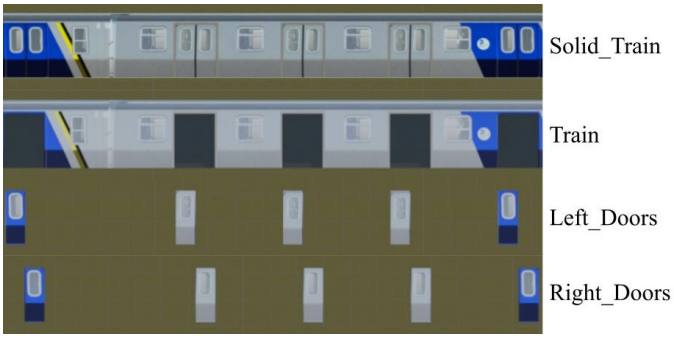

Fig. 7: Textured 3D Models

VI. Challenges

The reinforcement learning failure was one of a few challenges I faced this semester. I want to document a few of the most time-consuming issues I encountered.

A. Concurrency

When using a Python script for reinforcement learning, there is a disconnect between the Python code and the Gazebo simulation due to multithreading. The Python script is not perfectly synchronized with Gazebo, mainly causing issues with collisions and simulation resets. There is a bit of a delay before the Python script realizes the Gazebo simulation published to the collision ROS topic. This leads to issues with training efficiency, as the RL environment sometimes takes steps after a collision has happened and fails to add a penalty to the model. As I had no prior experience with ROS before this semester, I did not design the ROS nodes and topics with synchronization in mind. More work would need to be done to overhaul all of the nodes to properly synchronize.

Luckily, this bug is within the Python script. The Gazebo simulation by itself works as intended. It should be possible to fix this issue by just modifying the reinforcement learning code.

B. Simulation Visuals

Another challenge was creating a realistic looking simulation. My simulation is extremely basic looking. There are no textures and the train detail is minimal. When 3D modeling the train and platform, I was able to add textures, shown in Figure 7. To accomplish this, I found a side-view photo of a R211A train. I cropped it and wrapped it around the models using Blender, a 3D modeling software.

While I successfully added the textures within Blender, I ran into issues importing the textured models into Gazebo. The texture details were modified as the models moved through three different softwares. The models were exported from Autodesk Fusion to Blender, then from Blender to Gazebo. In this process, the texture wrapping was lost, causing the textures to render improperly in Gazebo. I was not able to resolve this issue and had to leave the final simulation with basic colors instead of detailed textures.

Additionally, the train model is extremely basic. This is due to the lack of existing free train models, as well as my limited 3D modeling experience. I believe that a more detailed model is necessary to limit the sim-to-real gap, but unfortunately I was unable to implement this for my project.

C. Physics and Simulation Reset

Gazebo's built-in physics engine was a challenge to learn about and work around. Even when marking models as "kinematic", meaning they are not affected by the physics of the world, the models still moved when colliding with each other. For example, the two door models would be launched into the air if spawned directly touching the body of the train. To avoid this, I had to spawn the doors so they were not touching the body of the train at all. Additionally, I had to add many safeguards and redundancies in the code to ensure the positions and velocities of all of the models were correct.

Resetting the Turtlebot3 also posed a challenge. I had difficulty properly resetting the velocities of all of the robot's components. This issue can be seen in the TeleOp tests, where the robot wobbled after each reset. This affected the reinforcement learning loop. If the robot got hit by the train, triggering a collision, the Turtlebot3 was respawned but continued to wobble as if it was just hit by the train. Luckily, I was eventually able to fix this. Each reset, my code pauses the Gazebo physics engine, waits a second, resets the Turtlebot3's velocities, waits another second, then unpauses physics. This fix took me many hours to find and still is not perfect, as it lengthens the time of each reset significantly.

D. Getting Started

As an individual with no prior experience with ROS, Gazebo, Reinforcement Learning, or robotics, this project was extremely challenging. Simply getting an Ubuntu virtual machine with ROS and Gazebo running took many hours. Once ROS and Gazebo were running, I had to learn how to use ROS topics, create a Gazebo world from scratch, import custom models, configure physics and collisions, code custom Gazebo plugins, and create a reinforcement learning environment. This learning slowed the initial progress of my project.

E. Computing Resources

As this project was developed inside a virtual machine, it lacked in computation power. Due to this, Gazebo ran around 50% slower than real-time. This hindered development speed and slowed reinforcement learning training. Running Gazebo without the GUI helped speed up the training process, but it still took multiple hours to train a basic RL model.

VII. Changes from Proposal

A. Curriculum Learning

Curriculum learning was explored as a possibility if a basic reinforcement learning approach was unsatisfactory in the train platform environment. While the current RL approach was not successful, curriculum learning was not explored due to a lack of time in the semester.

B. Variance

The project proposal also stated that variance would be added to the simulation to allow the reinforcement method to become more generalized. As I wasn't able to get the reinforcement learning method working accurately, I did not have time to add variance. The train and platform dimensions are static. There are no static or dynamic obstacles. The train arrival time is always the same.

C. Evaluation Metrics

In the proposal, average boarding time was proposed to be a evaluation metric. Again, as the reinforcement learning is not successful, this metric was not used. The model failed to board the train every time, so the average boarding time would have been N/A for all trials.

The creation of the Gazebo simulation took more time than estimated in the proposal research plan. This is the main reason why the reinforcement learning component was not finished, leading to the lack of curriculum learning, variance, and the average boarding time metric. I believe the creation of a novel train platform Gazebo simulation is substantial enough on its own for a solo semester project.

VIII. Conclusion and Future Work

Despite these challenges, I was able to create a working Gazebo simulation with a dynamic train. This simulation mirrors a real-world New York City Subway platform and R211A train. The platform and train models were created in Autodesk Fusion to be the exact dimensions of their real-world counterparts. Through custom Gazebo plugins, the train is able to properly decelerate to the platform. The train doors are able to open and close smoothly. The entire simulation has proper physics collisions. A Turtlebot3 robot can move around the platform and board the train using teleoperated controls. A basic reinforcement learning can control the simulation through various ROS topics, but currently fails to learn how to get a robot to board a train.

Future development is needed to improve the simulation. First, the reinforcement learning needs to be fixed. There are possible synchronization bugs to be corrected. Additionally, more detailed train and platform models could be introduced to help close the sim-to-real gap. Static obstacles, such as trash cans and benches, could also be added. Dynamic obstacles representing train passengers could also be added, enabling a robot to learn how to board a train among humans. Deboarding the train can also be explored. Other rail networks besides NYC Subways could be simulated.

Once a robot can board a train as realistically as possible, there are many opportunities to combine tasks from other research fields. A robot may be able to perform maintenance at one train station, then ride a train to the next station to complete maintenance there. Automated surveillance bots could survey both train platforms and inside train cars. Hotels may be able to offer a robotic service to their guests that carries their luggage for them from or to the airport.

This simulation lays the foundation for all future research on autonomous robot train boarding. I hope this project can be utilized to increase robot-train integration.

References

[1] I-Ming Chen, Ehsan Asadi, Jiancheng Nie, Rui-Jun Yan, Wei Chuan Law, Erdal Kayacan, Song Huat Yeo, Kin Huat Low, Gerald Seet, and Robert Tiong. Innovations in infrastructure service robots. In Vincenzo Parenti-Castelli and Werner Schiehlen, editors, *ROMANSY 21 - Robot Design, Dynamics and Control*, pages 3–16, Cham, 2016. Springer International Publishing.

[2] Haochen Liu, Miftahur Rahman, Masoumeh Rahimi, Andrew Starr, Isidro Durazo-Cardenas, Cristobal Ruiz-Carcel, Agusmian Ompusunggu, Amanda Hall, and Robert Anderson. An autonomous railroad amphibious robotic system for railway maintenance using sensor fusion and mobile manipulator. *Computers and Electrical Engineering*, 110:108874, 2023.

[3] RailPod Inc. RailPod Inc - Our Platform. https://www.rail-pod.com/our-platform/. Accessed: 2025-05-01.

[4] SHENHAO. TVIS 1000 Train Bottom Inspection Robot. https://www.shenhaorobotics.com/train-bottom-inspection-robot/. Accessed: 2025-05-01.

[5] ANYbotics. ANYbotics and Stadler Service AG Explore the Future of Train Maintenance. https://www.anybotics.com/news/robotic-inspection-in-train-maintenance/, 2021.

[6] Sean Scott. Meet Scout. https://www.aboutamazon.com/news/transportation/meet-scout, 2019.

[7] Nick Caloway. Uber Eats introduces delivery robots in Jersey City, New Jersey. https://www.cbsnews.com/newyork/news/uber-eats-robot-deliveries-jersey-city-new-jersey/, 2025.

[8] Amy Harrison. Bringing the future to today: making robot delivery a reality. https://www.uber.com/blog/so-funakoshi/, 2024.

[9] DoorDash. DoorDash and Coco Expand Global Partnership with U.S. Sidewalk Robot Delivery Launch. https://about.doordash.com/en-us/news/doordash-and-coco-expand-global-partnership, 2025.

[10] Piaggio Fast Forward. PFF Kilo. https://piaggiofastforward.com/business/kilo. Accessed: 2025-05-01.

[11] AVRIDE. AVRIDE Delivery Robot. https://www.avride.ai/robot. Accessed: 2025-05-01.

[12] Andrew Siff. NYC retires 420-pound NYPD subway surveilling robot after pilot ends. https://www.nbcnewyork.com/news/local/nyc-retires-420-pound-nypd-subway-surveilling-robot-after-pilot-ends/5099973/, 2024.

[13] Knightscope. Knightscope. https://knightscope.com/. Accessed: 2025-05-01.

[14] Aaron Chaflin, John Hall, and Paul Reeping. What To Do (and Not To Do) About Subway Safety. https://www.vitalcitynyc.org/articles/what-to-do-about-subway-safety-nyc-policy-recommendations, 2025.

[15] Gazebo. Gazebo Tutorials. https://classic.gazebosim.org/tutorials. Accessed: 2025-05-01.

[16] Metropolitan Transportation Authority. Subway and bus ridership for 2023. https://www.mta.info/agency/new-york-city-transit/subway-bus-ridership-2023. Accessed: 2025-05-01.

[17] Metropolitan Transportation Authority. Subway and bus facts 2019. https://www.mta.info/agency/new-york-city-transit/subway-bus-facts-2019. Updated Aug 7, 2024.

[18] Metropolitan Transportation Authority. We're giving stations Re-NEW-vations. https://www.mta.info/agency/new-york-city-transit/renewvation. Accessed: 2025-05-01.

[19] David Meyer and Stephen Yang. Manhattan officials ask MTA to 'quickly' test subway platform barriers after fatal shove. Accessed: 2025-05-01.

[20] Candy Chan. Project Subway NYC 3D Models. https://www.projectsubwaynyc.com/3d-models. Accessed: 2025-05-01.

[21] Metropolitan Transportation Authority. 2025-2029 CAPITAL PLAN. https://www.mta.info/document/151266, 2024.

[22] Wikipedia. R211 (New York City Subway car). https://en.wikipedia.org/wiki/R211_(New_York_City_Subway_car). Accessed: 2025-05-01.

[23] NoneCG. New York City R211 Subway Trains 3D Model. https://www.renderhub.com/nonecg/new-york-city-r211-subway-trains, 2024.

[24] Pyo, Darby Lim, and Gilbert. turtlebot3_gazebo. https://wiki.ros.org/turtlebot3_gazebo. Accessed: 2025-05-01.
[25] OpenAI. Gym Documentation. https://www.gymlibrary.dev/. Accessed: 2025-05-01.
[26] Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. Stable-Baselines3: Reliable Reinforcement Learning Implementations. *Journal of Machine Learning Research*, 22(268):1–8, 2021.