Autonomous Intersection Management Systems (AIMS)

NSF Self-Drive REU 2022

Grant No. 2150292

Introduction

The aim is to build an Automated Intersection Management System. Our proposal for the key contributions of this work are as follows:

1. A model to plan the trajectory of at least four AVs across an intersection using occupancy grid mapping and a modified A* algorithm using purely Vehicle to Vehicle (V2X) communication. We use a custom heuristic function with the A* algorithm to smooth the path and restrict the direction of motion of the vehicles to forward, left and right.

2. Vehicles use a priority order system that takes into account accident prone driving like left turns and to simulate real-time conditions of emergency.

3. Implementation of the model in a simulated environment using ROS and Gazebo.

Related Work

The authors use the pure pursuit algorithm for vehicle steering. However, the authors rely on sensor local information and existing topology maps, instead of V2V communication as in our work.

V2V communication has proven to be effective in intersections.

The author's account for both fully autonomous and driver assistance (parallel autonomy) operating modes in a physical implementation which is a future research direction for our proposed work.

https://scripts.mit.edu/~ddv/publications/ICA_Conf.pdf

Choice of Simulator - CARLA

Pros:

- Multiple vehicle support
- ROS support
- Visually appealing

Cons:

- Requires lots of processing power
- Only works with certain versions of Ubuntu and ROS

“Running Simulator Standalone.” https://carla.readthedocs.io/en/stable/running_simulator_standalone/

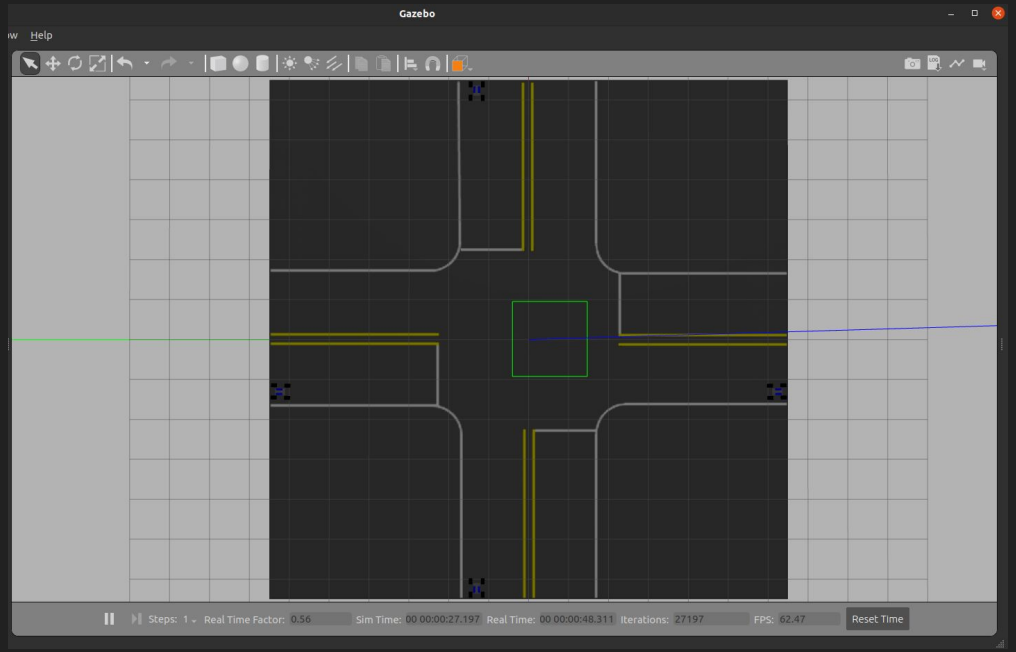
Choice of Simulator - Gazebo

Pros:

- Can run on any computer
- Has multiple vehicle support

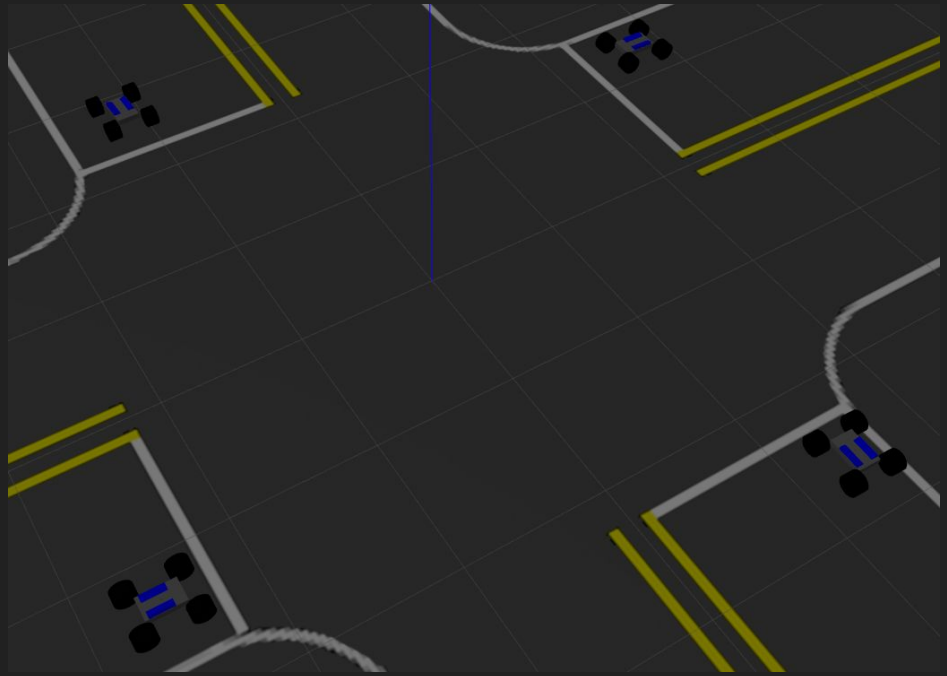
Cons:

- Multiple vehicle support is poorly documented
- Cannot run well with a large number of vehicles



Gazebo Simulator

- Multiple vehicles are possible
- Works with ROS
- Twist messages
- Vehicles can be spawned and despawn sequentially
- Dynamic reconfigure integration
- Modularity



Gazebo Simulator

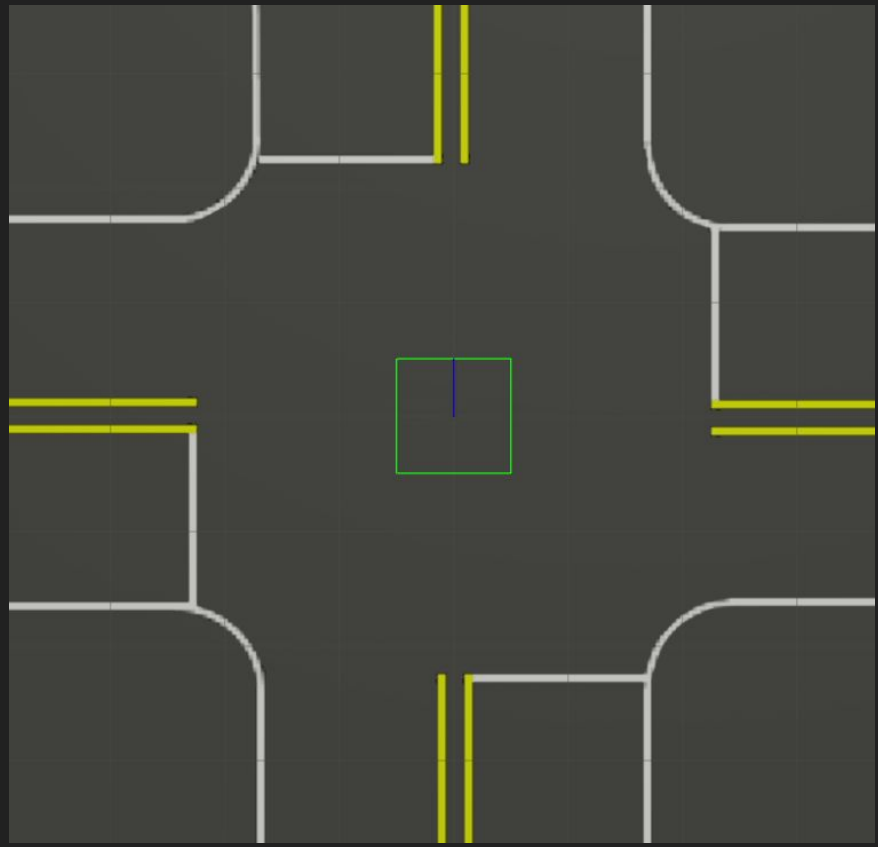
Caveat:

- Time synchronization issues
- Requires computing power to run smooth

Good news:

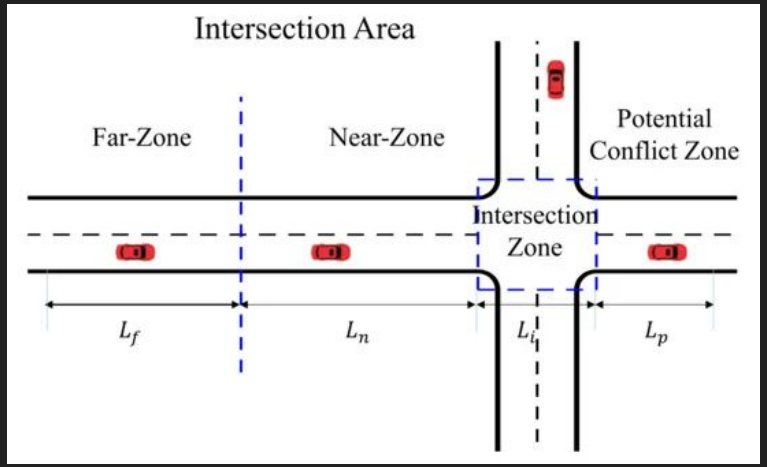
- AIMS using Gazebo and ROS should be doable!

ROS Package of the Simulator:
https://github.com/irisfield/aims_sim_gazebo



Priority System

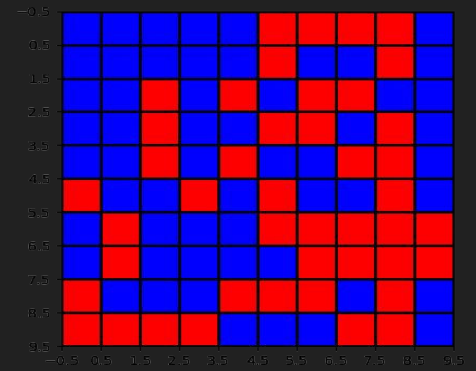
- A first-come-first-serve (FCFS) policy
  - Split into 2 zones for service, a near-zone and far-zone
- Exceptions for emergency vehicles
  - Like in real-life, for life-threatening situations, an ambulance or police car would have to be able to override the FCFS policy and make its way quickly and safely through the intersection
  - To make sure the emergency vehicle isn't blocked, autonomous vehicles in the near-zone will still travel first so the way is cleared for emergency vehicles
- Further research into energy efficient priority systems

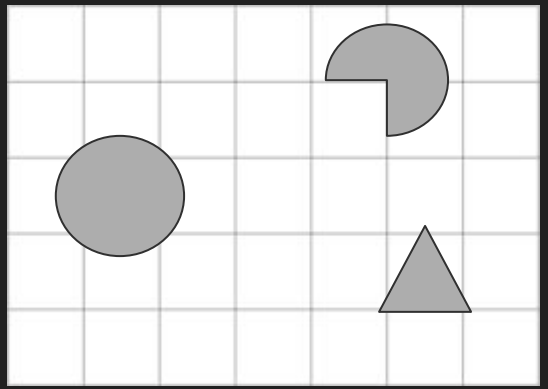


[1]

Occupancy Grid

- We need to know where obstacles are.
  - Squares assigned 0 for non-occupied areas
  - The more squares, the more detail, the more resources needed
- NumPy Array
  - Simple, Easy to visualize, doesn't require a lot of resources
  - Not as accurate as other methods





https://stackoverflow.com/questions/52566969/python-mapping-a-2d-array-to-a-grid-with-pyplot

Occupancy Grid

- Octomap
  - Built in ROS library which allows for the implementation of either a 2D or 3D Occupancy Grid within Gazebo. Utilization of any kind of occupancy grid is important as it allows for the implementation of path planning which requires the positions of all objects, obstacles, and vehicles on the field at any given point in time.
  - Octomap is flexible and dynamically expanding, this means that the extent of the map does not need to be known beforehand. Octomap is also compact in memory, this means that large environments can be mapped and a vehicle can keep the model within its main memory. This is very useful in the context of AIMS as information can be transmitted efficiently between multiple robots.

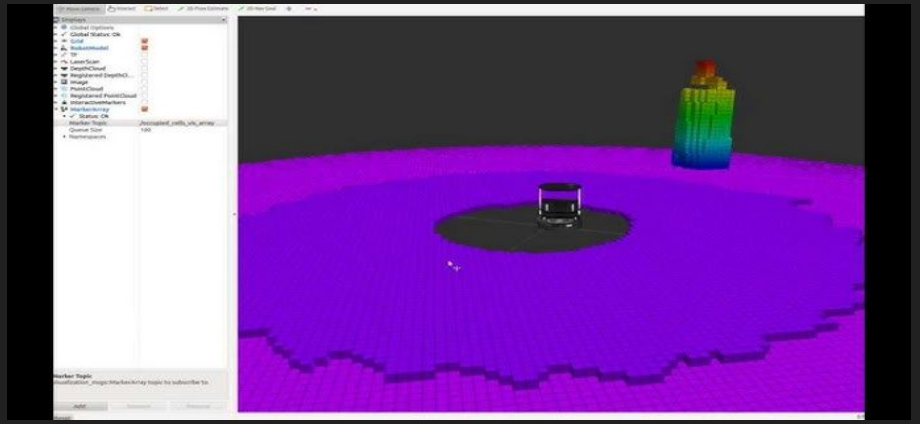


Fig 1. Octomap allows for vehicles to keep track of the location of all other objects within range

http://octomap.github.io/

https://wiki.ros.org/octomap

Path Planning

Proposed algorithm-

A* path planning algorithm combined with pure pursuit tracking algorithm with a heuristic function using cubic splines instead of the conventional Euclidean distance function for linear path smoothing.

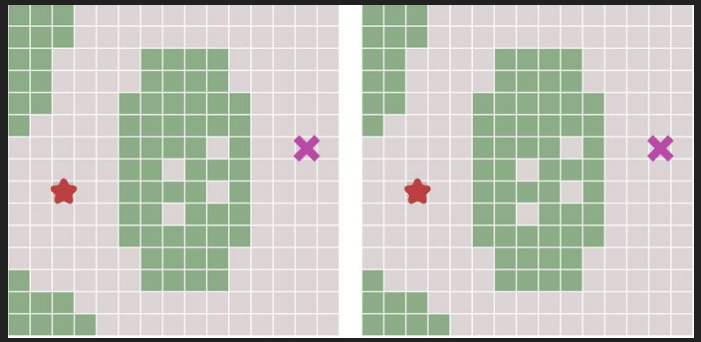
A* algorithm would work the best for our model due to its low computation time and near optimal solution.

It has also been used before in previous intersection management research. https://www.sciencedirect.com/science/article/pii/S2405896320322072

Path Planning A*

A* is a local path planning algorithm that adds a heuristic function to Dijkstra's algorithm. In the classical Dijkstra's algorithm, the computer searches for a route between two positions by examining in each iteration the neighbors of a parent node.

In contrast, the A* algorithm adds a priority queue, which encourages the examination of new nodes closer to the goal and/or with a lower cost. In this algorithm, the cost is defined by the vehicle and/or environment restrictions and the desirable movement. Therefore, higher costs can be assigned to nodes next to obstacles or some types of movements can be forbidden. This is especially useful for intersection management as we need to restrict the movement of the vehicles to forward, left, and right.



Path Tracking Pure Pursuit Algorithm

The pure pursuit algorithm is a geometric path tracking method. We focus on the overall velocity and heading changes of vehicles, so the kinematics model based on the vehicle motion is selected to update the position. Pure pursuit algorithm produces the steering angle required to bring the vehicle back to the reference path.

It calculates and defines the look-ahead point on the reference path by the look-ahead distance and the current vehicle’s actual position. The pure pursuit algorithm outputs the steering angle of the vehicle to follow the input look-ahead point. Finally, it outputs the vehicle’s actual position and then continues to calculate the look-ahead point.

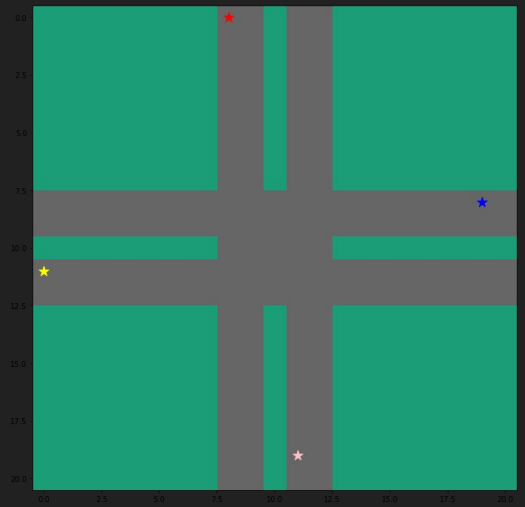
Cubic Splines

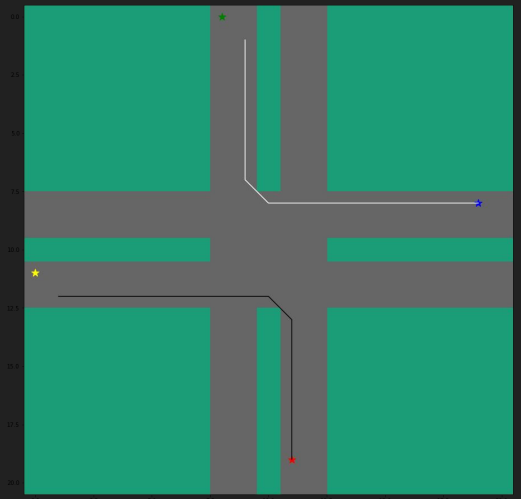
- Continuity of connected line segments
- Continuous on first and second derivative
- Predetermined clamped endpoints of free endpoints

$$S(x) = \begin{cases} s_1(x) & \text{if} \quad x_1 \le x < x_2 \\ s_2(x) & \text{if} \quad x_2 \le x < x_3 \\ \vdots & \\ s_{n-1}(x) & \text{if} \quad x_{n-1} \le x < x_n \end{cases}$$

1. The piecewise function $S(x)$ will interpolate all data points.
2. $S(x)$ will be continuous on the interval $[x_1, x_n]$
3. $S'(x)$ will be continuous on the interval $[x_1, x_n]$
4. $S''(x)$ will be continuous on the interval $[x_1, x_n]$

Path Planning Initial Implementation





The coordinates for both of the paths

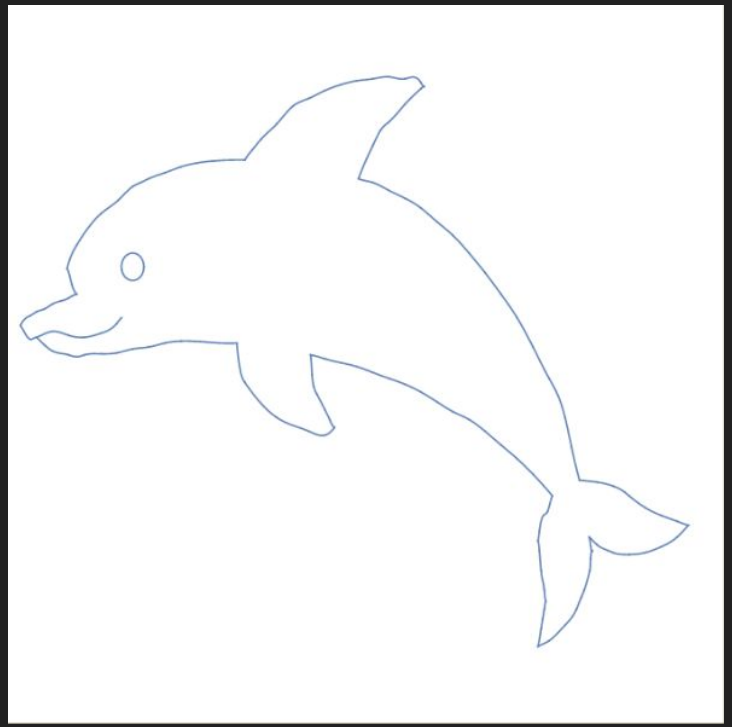
are as follows:

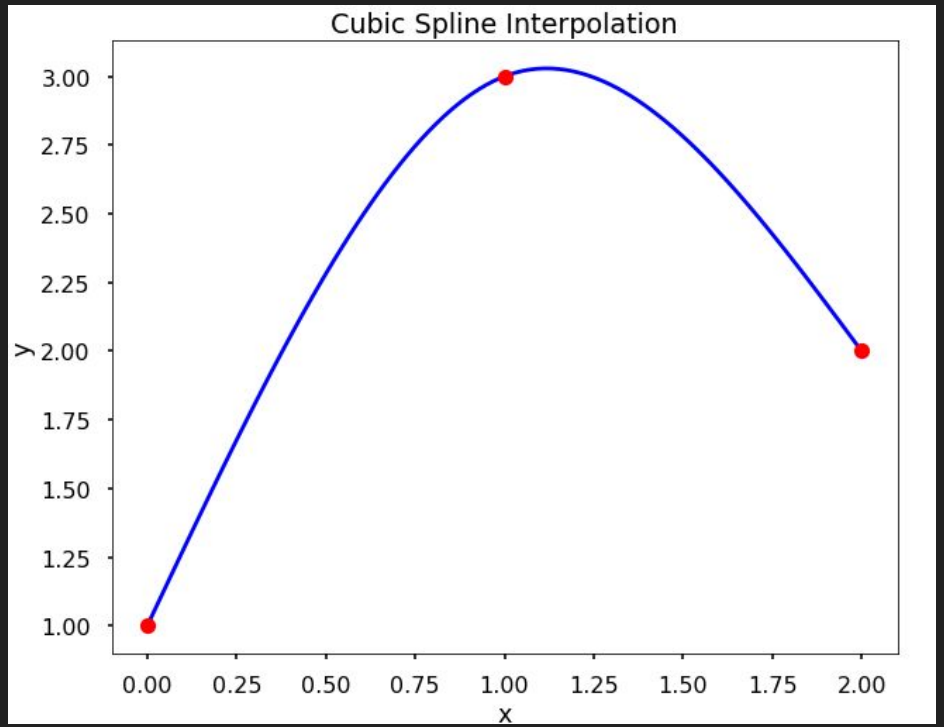
[[(19, 11), (18, 11), (17, 11), (16, 11), (15, 11), (14, 11), (13, 11), (12, 10), (12, 9), (12, 8), (12, 7), (12, 6), (12, 5), (12, 4), (12, 3), (12, 2), (12, 1)], [(8, 19), (8, 18), (8, 17), (8, 16), (8, 15), (8, 14), (8, 13), (8, 12), (8, 11), (8, 10), (7, 9), (6, 9), (5, 9), (4, 9), (3, 9), (2, 9), (1, 9)]]

[(8, 19), (8, 18), (8, 17), (8, 16), (8, 15), (8, 14), (8, 13), (8, 12), (8, 11), (8, 10), (7, 9), (6, 9), (5, 9), (4, 9), (3, 9), (2, 9), (1, 9)]

https://colab.research.google.com/drive/1Pyg4dipUIkFz9N0IGcLS7NdQXDmBuq_U#scrollTo=zP_CAFr09Xy0

Cubic Splines Initial Implementation





Path Planning using 1 premade vehicle and course

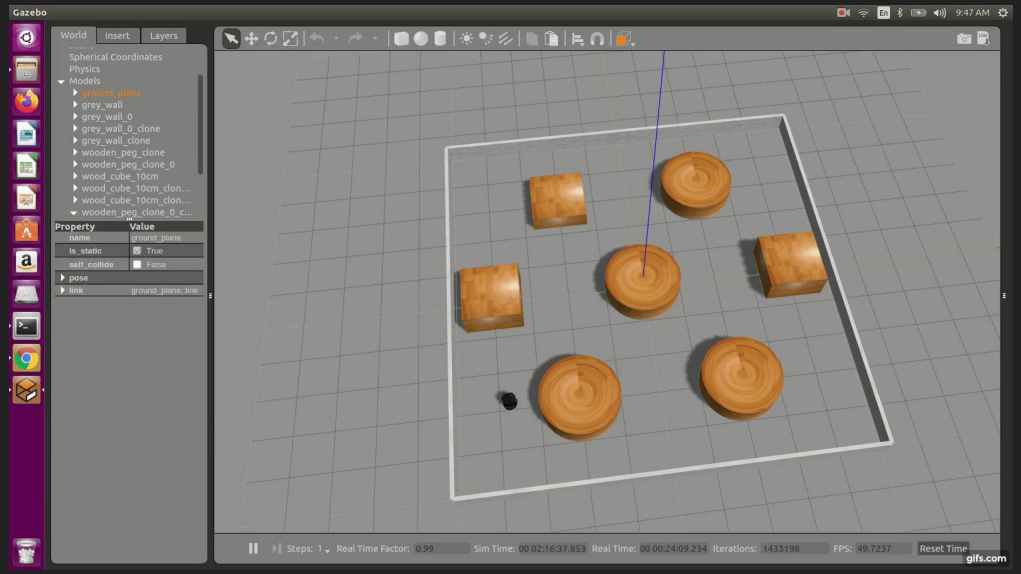
https://github.com/arp95/turtlebot_astar

This was used as a source to test

the implementation of

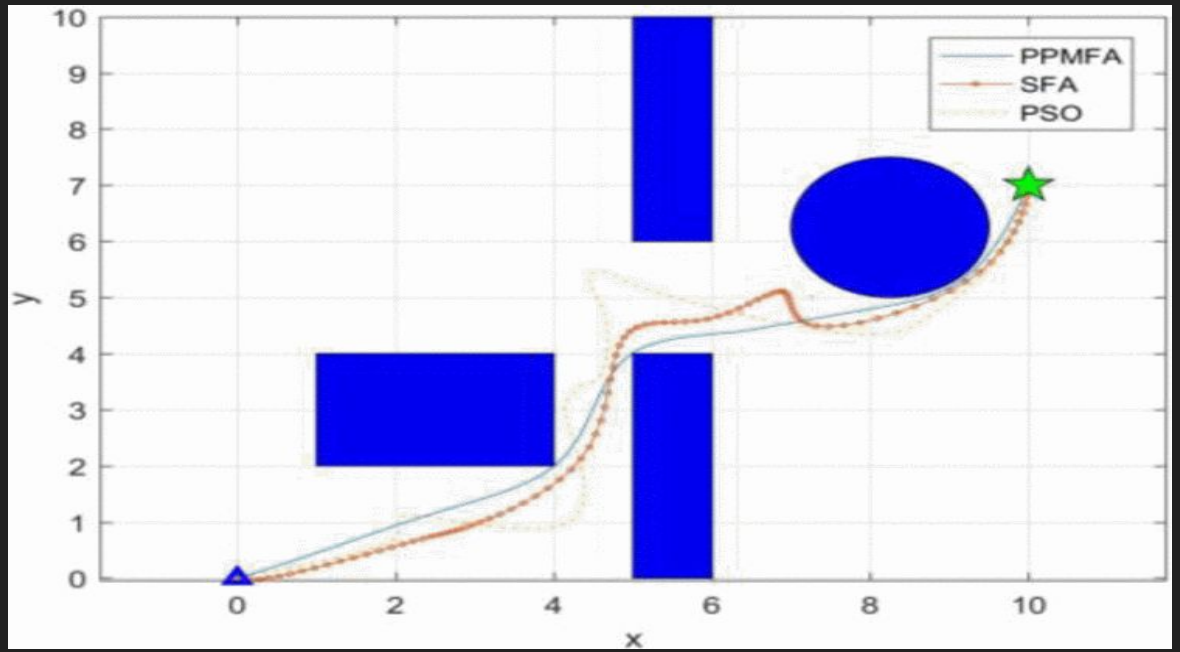
A * for 1 vehicle in ROS and

Gazebo in a premade environment.



Path Planning Comparison for an AIMS

- Particle Swarm Optimization (PSO)
- Path Planning Modified Firefly Algorithm (PPMFA)



[3]

How it all comes together?

We have separate files for the independent implementation of cubic splines and path planning using A* with multiple vehicles, a cubic splines implementation, and path planning for 1 vehicle using A* in ROS and Gazebo. We are still trying to integrate all 3 together.

Other options for the future

1. A motion planning algorithm sampling-based, PRM, solves the problem of determining a path between a starting configuration and a goal configuration while avoiding collisions. We didn't adopt this as we are using occupancy grid mapping and that is proven to work well with A* which is a grid based search.

2. The RRT is an algorithm designed to search non-convex, high-dimensional spaces by randomly building a space-filling tree. This algorithm is combined with Artificial Potential Field (APF), Dynamic Window Approach (DWA) and Timed Elastic Bands (TEB).

The APF treats a robot's configuration as a point in a potential field that combines attraction to the goal, and repulsion from obstacles to track the safe path. One velocity-based local planner, DWA, calculates the optimal collision-free velocity for a robot required to reach its goal, which translates a Cartesian goal (x,y) into a velocity.

These are some other options which have been worked upon in previous literature-
A*+DWA, A*+TEB and TD3 but not for an AIMS.

https://mdpi-res.com/d_attachment/sensors/sensors-20-05493/article_deploy/sensors-20-05493.pdf?version=1601021069

In the above paper, the PRM path planning approach with TD3 to be a PRM+TD3 planner (DRL algorithm Twin Delayed Deep Deterministic policy gradients (TD3) with the traditional global path planning algorithm Probabilistic Roadmap (PRM) as a novel path planner (PRM+TD3)) was proposed for an indoor mobile robot but not for an AIMS.

References

1) Zhang, H.; Zhang, R.; Chen, C.; Duan, D.; Cheng, X.; Yang, L. A Priority-Based Autonomous Intersection Management (AIM) Scheme for Connected Automated Vehicles (CAVs). Vehicles 2021, 3, 533-544. https://doi.org/10.3390/vehicles3030032
2) Jai BheemanJai Bheeman 4111 gold badge22 silver badges44 bronze badges, Willem Van OnsemWillem Van Onsem 403k2929 gold badges374374 silver badges491491 bronze badges, & Space ImpactSpace Impact 12.4k1818 silver badges4444 bronze badges. (2018, September 29). Python - mapping a 2D array to a grid with pyplot? Stack Overflow. Retrieved July 13, 2022, from https://stackoverflow.com/questions/52566969/python-mapping-a-2d-array-to-a-grid-with-pyplot
3) X. Chen, M. Zhou, J. Huang and Z. Luo, "Global path planning using modified firefly algorithm," *2017 International Symposium on Micro-NanoMechatronics and Human Science (MHS)*, 2017, pp. 1-7, doi: 10.1109/MHS.2017.8305195.