Collision Avoidance of Mobile Robots in Dynamic Environments

Jason Munger
Dept. of Robotics Engineering
Worcester Polytechnic Institute
Worcester, Massachusetts
jpmunger@wpi.edu

Rene Verduzco
Dept. of Robotics Engineering
Worcester Polytechnic Institute
Worcester, Massachusetts
rverduzco@wpi.edu

Clinton Williams
Dept. of Robotics Engineering
Worcester Polytechnic Institute
Worcester, Massachusetts
cbwilliams2@wpi.edu

***Abstract*—As mobile robots become more ubiquitous in our everyday lives, human-robot safety will become paramount when facilitating interactions. The introduction of mobile robots in cluttered environments has increased productivity and opened new opportunities for robot services, but has also introduced the potential for harm to people and damage to the surroundings should the robot collide with them. This project aims to reduce the risk of collisions between mobile robots and obstacles in dynamic environments by implementing a dynamic path planning algorithm that can rapidly react to dynamic obstacles and efficiently re-plan a path avoiding these obstacles to reach the goal. We implemented various velocity-obstacle variants with static and dynamic obstacles and compare the performance to reactive methods implemented in previous work. The planner used RRT* search algorithms to determine an initial feasible path for the robot to move from a start to goal position and attempt to traverse the path as quickly as possible. We demonstrated two methods of re-planning when obstacles/agents are encountered: 1) use the velocity and acceleration-velocity obstacle avoidance strategies to avoid collisions and attempt to move back to the original path and 2) attempt to find a new feasible path from the robots new position after avoiding a collision. Previous work has demonstrated the capability to significantly reduce the chance of collision, but with a modest increase in distance and time to reach the goal position. This current work will demonstrate not only reduced collisions, but also improve execution time without impacting the distance traveled by our robot.**

***Index Terms*—Path Planning, Rapidly-Exploring Random Trees, Velocity Obstacles, Dynamic Environments**

I. Introduction

Mobile robots have a wide variety of use cases in modern society, from autonomous vehicles to automated farming, warehouse transportation, service delivery, and more. The most crucial part of a mobile robot system is the navigation where path planning is needed for the robot to successfully traverse the environment. Though path planning is relatively easy in a well defined indoor static environment, path planning in a cluttered dynamic environment is still a challenging problem within the industry. The added sensing and computational complexity of dynamic environment path planning have driven many of the mobile robot solutions, especially within the warehouse environment, to implement a strict, predefined path planning rules and algorithms. However, these solutions are costly and inefficient in most situations or merely impossible for outdoor applications. One of the most difficult problems for self-driving cars is the navigation in crowded urban environments consisting of other vehicles (autonomous and not), bicycles, people, or various road hazards. Virtual environments for computer graphics or video games can also benefit from dynamic path planning algorithms for the purpose of crowd simulations. This can also be used to design efficient real-world traffic engineering patterns for different scenarios. Thus, efficient algorithms for dynamic path planning are needed for broader use cases of mobile robots in unstructured or complex environments. For the purpose of this project, we want to focus on the topic of automating and optimizing warehouses and distribution centers to ensure fast and efficient supply chains with minimal safety hazards.

The first robots to be introduced into the supply chain were robotic arms that performed dangerous or hazardous jobs in industrial manufacturing. The robots of that time did not have the ability to perceive their surroundings - something which they are capable of now. Today, there are many types of robots able to perceive their surroundings being used in warehouses and distribution centers to help meet business goals. Automated Guided Vehicles (AGVs) and Automated Guided Carts (AGCs) follow a path defined by a set track or set magnetic stripes. Autonomous Mobile Robots (AMRs) on the other hand do not require a laid out path. The introduction of mobile robots has helped companies like Amazon fulfill their ever increasing promise of speedy-deliveries, but also introduced risks to worker safety by adding more variables to the dynamic environment. When a location, such as a distribution center, is filled with an increasing number of workers and robots, the problem of path planning becomes increasingly important. It is necessary for a robot to safely and efficiently navigate from one location to the next while simultaneously avoiding predetermined obstacles and obstacles introduced by the dynamic environment such as workers and other robots. This path planning problem aims to ensure the safety of workers and the environment while meeting the supply chain demand.

Well known motion planning algorithms such as Rapidly-exploring Random Tree (RRT) and Probabilistic Roadmap Method (PRM) work exceptionally well in static environments where the map and robot position are known. However, many if not most real-world application settings have dynamic

environments with moving obstacles where the standard path planning algorithms are insufficient. For example, in a distribution center environment, mobile robots moving around in a warehouse may encounter humans, equipment, or other robots blocking their paths and will need to re-plan their paths accordingly. A dynamic environment typically contains static features such as buildings, walls, trees etc. while also containing dynamically moving objects and unknown objects that must be accounted for during autonomous motion in order for the robot to behave in a manner that is safe and reliable. Also, the robot may not have any way to communicate explicitly with these external obstacles and moving agents making it impossible to build an overall network to organize motion around. Our proposed project addresses the challenges posed in dynamic environments by leveraging robot sensors to perceive obstacles and agents around it to adapt and dynamically re-plan the path as the robot encounters obstacles during its trajectory along the original path. Dynamic path planning remains an open research problem with many opportunities for innovation. We believe that implementing this planner would enhance the human-robot experience in warehouses and in other applications containing multiple fast moving agents and cluttered environments.

II. Related Work

In this section we review existing literature regarding obstacle avoidance and rapidly-exploring random tree algorithms for path planning.

A. Path Planning

Mobile robots have a wide variety of use cases in modern society, from autonomous vehicles to automated farming, warehouse transportation and service delivery. Planning a path for a robot to move from a point in space to another while avoiding obstacles and also considering optimal path cost seems deceptively simple, yet is one of the most challenging problems in computer science and robotics.

Consider a robot attempting to navigate from a point in space to another through an open space while avoiding obstacles. The environment the robot finds itself in is composed of free space and the obstacle space (the complement of free space). The robot can be described as a point in the configuration space, which is a set of parameters that describe it in space. These parameters include robot position (x and y coordinates) and its orientation θ.

Using sensors to perceive and update its environment for use as inputs to path planning algorithms, a robot can plan a set of motions to reach a desired end goal. The added sensing and computational complexity of path planning in dynamic environments has driven many mobile robot solutions in mobile robotics, video games agents, and other applications. Thus, efficient algorithms for dynamic path planning and obstacle avoidance are needed in mobile robots.

B. RRT*

First, lets review the path planning, specifically RRT*. RRT* is a modified version of the RRT algorithm. It tries to smooth the tree branches in each step. It has been mathematically proven that for a node count that reaches ∞, the calculated RRT* path will be the shortest. Two additional features that set it apart from RRT are near neighbor search and rewiring tree operations. The near neighbor calculation determines the best parent node with the least cost for the new node and the rewiring tree operation rebuilds the tree to determine the tree with minimal cost between tree connections. [1] [2]

The distance each new node has traveled relative to its parent node is referred to as the cost. The neighbors of each new vertex are examined a fixed radius away to determine if a cheaper cost is available. If so, then the cheapest node replaces the previous one.

C. Collision Avoidance

Collision avoidance is a key problem in robotics. There has been a significant amount of research and continues to be improvements in this area of robotics. The task can be defined as a robot employing sensing and vision systems [3] [4] [5] to track the robots specific location in the environment versus obstacles in the robot's path and surrounding areas. A current popular method of obstacle avoidance in dynamic environments is model-predictive control (MPC) which is a control feedback scheme that perceives ahead in time for a finite horizon to predict future outputs [6] [7] [8]. Figure 1 is an overview of the current strategies for obstacle avoidance and their respective applications [9].

		Single vehicle	Moving obstacles	Multi-vehicle	Boundary following
MPC	Standard	U*,A,M,F,T	U*,A,M,F,T*		
	Robust	U*,A,M,F,R,T			
	Local planning	U,A,M,F,R	A,M,F		U,A,M,F,T
DMPC	Distributed optimisation			U*,A,M,R	
	Synchronous			U*,A,M,F,R	
Boundary following	Minimal information		U*,A*,M,F,T*		U,A*,M,F,T
	Full information		U*,A*,M,F,T*		U,A*,M,F,T
	Bug algorithms				U,F,T
Velocity obstacle	VO/NLVO	U*,A*,F,T*	U*,A*,M,F,T*		
	DRCA/RCA		U,A*,M*,F,R*	U,A*,M,F,R,T*	
Other	APF	U,A*,M,F,T*	U,A*,M,F	A*,M*,F,T	U,M,F,T
	Tangent following	U,A*,M,F,T	U*,A*,M,F,T		U,A*,M,F,T
	Other reactive	U,A*,M,F,T	U,A*,M,F,T*	U,F	U,M,F,T
	Hybrid logic	A,M,F,T		A,M,F,T	

Fig. 1. Overview of collision avoidance strategies.

Key:	
U	Unknown environment
A	Acceleration bounded
M	Unicycle or bicycle model
F	Fast computation
R	Robust to disturbance
T	Provably convergent
*	Some disadvantages

Based on our proposed application with single robot vehicle and moving obstacles, we will use the concept of velocity obstacles [10] [11] [12] [13] and acceleration-velocity [14] obstacles avoidance strategies to improve upon our baseline work and compare the improvements.

Velocity-based avoidance implies that the robot agent takes into account all of the observed velocities of the object to determine an avoidance free collision. As shown in Figure II-C, robot A will select a new velocity in order to guarantee a

collision-free path with moving obstacle B for a preset amount of time. The new velocity should be maximized to be as close to the preferred velocity but guarantee a collision-free path. This strategy is discussed in detail in [15] [16]. The acceleration component drives the robots to desired velocity using proportional control, i.e. the applied acceleration is continually proportional to the difference between the new velocity and the current velocity.

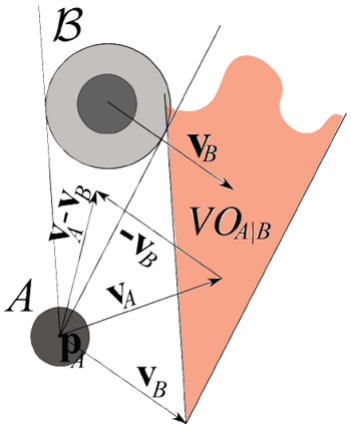

Fig. 2. Velocity avoidance concept.

III. Method Overview

Steps *A* through *C* will be accomplished in Python.

A. Create Environment and Path Planning Algorithm

In this first step, we will *(1)* define the environment and obstacle geometry. We will *(2)* render the environment given the time-step and robot position and then *(3)* implement a function to detect collisions. We will implement RRT* algorithm to plan a feasible path around the static obstacles in the environment.

B. Implement Baseline Reactive Methods

In the second step, we will implement a set of "reactive" distance-based collision avoidance methods. The reactive methods are discussed in the section V.

C. Implement Collision Avoidance Algorithms

In the third step, we will implement velocity obstacle avoidance with our RRT* path planner to compare and improve on the "reactive" methods. This velocity obstacle (VO) and acceleration-velocity obstacle (AVO) avoidance strategies will aid the robot agent in traversing through the environment while avoiding the moving obstacles.

D. Run Simulations and Collect Data

When steps *A*, *B*, *C* are completed, we will begin to *(1)* collect data from simulations. The total *(2)* path distance will be recorded for comparison amongst simulations.

E. Compare Results to Baseline

We will compare the "reactive" collision avoidance methods to the velocity obstacle strategy against the following key metrics discussed: *(1)* completion time, *(2)* quality of path - measure Euclidean distance from start to finish, and *(3)* number of collisions by the robot along the path.

IV. Class Structure

The block diagram in Figure IV represents the class structure of this project.

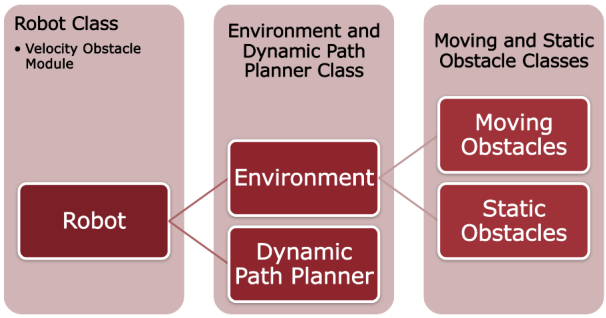

Fig. 3. Project Class Structure

A. Robot Class

The robot class inherits the Environment and Dynamic Path planner classes. Not represented in this diagram is the fact that the path planning class also inherits the environment to plan the initial optimal path the robot traverses.

Using Velocity Obstacle Module, Robot, Environment, and Dynamic Path Planner attributes are used to calculate permissible velocity at each time step.

B. Environment Class

The first step in our path planning problem was to create a simulated environment that our path planning algorithm could be built on top of. The path planning algorithm functions as a consumer of the environment class by planning paths dynamically based on the evolving environment state. An Environment class, a Moving Obstacle class, and a Static Obstacle Class in the env.py file were created to render our dynamic environment, which consist of the following core responsibilities:

- Updating the environment at each time step.
- Tracking the positions and velocities of the dynamic obstacles.
- Detecting collisions
- Rendering the environment as a two-dimensional image.

The environment class consists of the following methods:

- step(): updates the environment for one time step.
- check collision(): checks for collisions between the robot and obstacles present in the environment using the two methods listed below.
  - check static collision(): checks if the circle collides with any static obstacle.
  - check dynamic collision(): checks if the circle collides with a dynamic obstacle.
- draw(): renders the environment as a 2D image for display by calling the methods listed below:

- draw static obstacles(): draws static obstacles
- draw moving obstacles(): draws moving obstacles

Figure 4 represents the dynamic environment rendered as a 2-dimensional image with RRT* planned path. The green circle is our 2D robot. The blue circle represents our goal,which remains static across different simulations. In each individual simulation, the black rectangles remain static whereas the red dots move freely about the map. However, there is some variability in the size and position of the static obstacles across different simulations.

V. Reactive Methods

The ACT() method that performs the different strategies on how the mobile robot deals with obstacles it encounters as it moves along its planned trajectory. We have defined three strategy methods in this class that dictate how the robot will react when it detects moving obstacles in its vicinity. The three strategies are:

- LOOK AHEAD() : main strategy compared against. Computes n future positions of both robot and detected moving obstacles.
- WAIT() : keeps robot in the same position when an obstacle is detected.
- CONTINUE() : continues to move the robot along the path regardless of collisions with moving obstacles.

The LOOK AHEAD strategy is our main reactive algorithm, and the other two strategies are baselines to compare against. The LOOK AHEAD strategy computes n future positions of both the robot and the detected moving obstacles. Each future position of the robot is checked for collisions with the future obstacle positions. If any future collisions are predicted, re-planning is performed and the future obstacle positions are taken into account so that re-planning avoids them. That is, LOOK AHEAD anticipates the robot's trajectory to determine if a collision will happen by examining the robot's path a number of steps ahead of its current position, and re-planning accordingly.

The WAIT strategy does not modify the initial path when a moving obstacle is detected, it instead keeps the robot in the same position when an obstacle is detected and only continues to follow its planned path when the obstacle has moved away from it.

The CONTINUE strategy continues to move the robot along the path regardless of it colliding with moving obstacles.

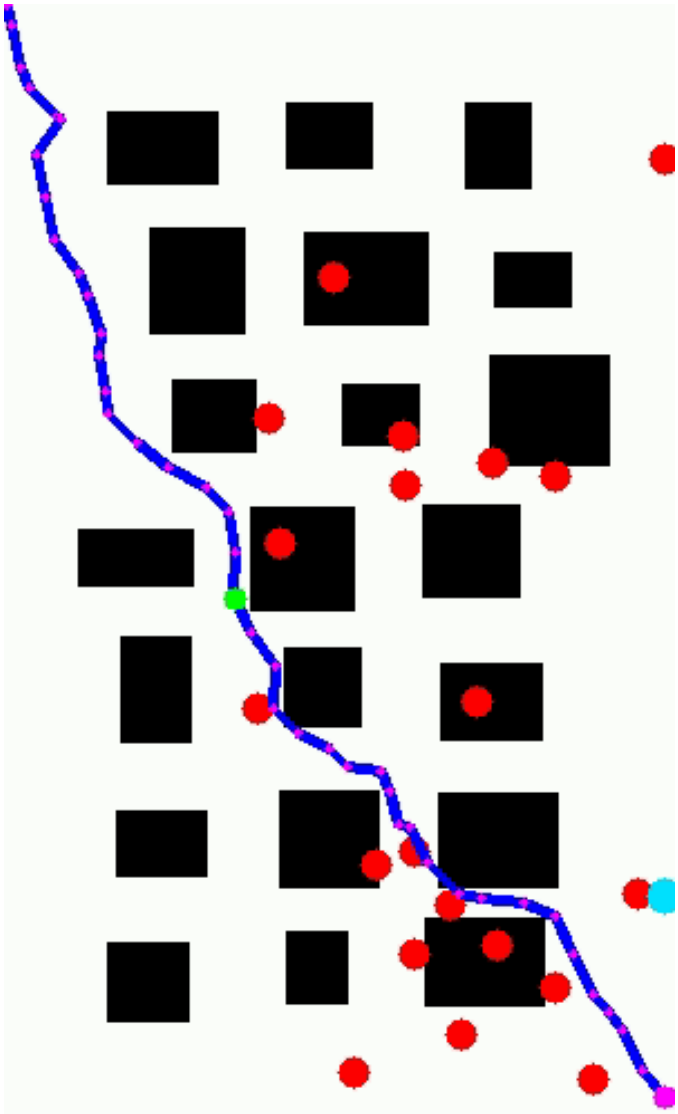

Fig. 4. RRT* traversed path in initial environment

Figure 4 represents the initial robot environment and a path traversed using the RRT* sample-based path planning algorithm. The RRT* path planner finds an initial feasible path from start to goal considering static obstacles.

The robot then traverses the initial found path and applies one of the three 'reactive', distance-based collision avoidance strategies when dynamic obstacles are encountered.

VI. Velocity Obstacle Avoidance Methods

A. Velocity Obstacles

The first obstacle avoidance method used to compare against the baseline reactive methods is the generalized velocity obstacle (VO). As described previously, the goal of the VO algorithm is to identify safe, collision-free velocities the robot can apply to ensure obstacles are avoided as the robot moves toward a goal position. This is done under the assumption the robot is equipped to measure or know obstacle positions, velocities, and shape (and therefore radius assuming disk shapes) to use for the VO computations. The algorithm

implemented for this approach can be seen in Algorithm 1 from [16]. This approach can be characterized as a sample-based method. A number of controls are selected from an available set. In this case, the control would be robot velocities. A sampling time limit is chosen that discretizes the time from 0 to some maximum limit known as the time horizon. This can be thought of as how far into the future the robot looks at possible collisions with obstacles. The implicit assumption here is that all obstacle and agent behavior is consistent enough to make reliable future predictions. This may be considered a limitation for obstacles that may not have as predictable motions such as humans or animals crossing in front of or near the robot. A variable called free is used to track whether a given control (i.e. velocity) is outside the velocity obstacle or not. True indicates a valid velocity the robot can take and avoid collision while False indicates an invalid velocity that will result in collision at some time over the time horizon and is thus discarded. In this project implementation, a grid of free values is initialized that corresponds to the number of controls, *num controls*, and the number of discrete velocity magnitudes the algorithm checks for between 0 and the maximum robot velocity. The distance between the robot and all moving obstacles is computed for each velocity over the range of time. The minimum distance is found between the robot and each obstacle indicating the closest approach the two will have. The minimum time required to reach this minimum distance is found and then checked to see if the velocity at this time results in a collision by computing the distance between the robot and obstacle at the minimum time. If the distance at t_{min} is less than the combined radius of the robot and the moving obstacle, this indicates the velocity will result in a collision and the corresponding index in the free grid is switched to False. When all sampled controls have been evaluated, the free grid can then be used to identify the set of all valid and invalid velocities for the robot at the current state of the environment over a specified time horizon. The set of valid velocities is checked to see which is closest to the preferred velocity. In this case, the preferred velocity is the maximum speed of the robot possible in the direction between the current position and goal position. The valid velocity whose magnitude it nearest the magnitude of the preferred velocity is chosen to be the updated robot velocity.

Algorithm 1 Find best feasible control

for $i = 0$ to n **do**
  $u \leftarrow$ sample controls from the set of all controls U
  $t_{\text{lim}} \leftarrow$ sample time limit $\in (0, max]$
  $free \leftarrow$ true
  $min \leftarrow \infty$
  for all Moving Obstacles B **do**
    let $D(t) =$ the distance between $A(t, u)$ and $B(t)$
    $t_{\text{min}} \leftarrow$ solve $\min(D(t))$ for $t \in [0, t_{\text{lim}}]$.
    $d \leftarrow D(t_{\text{min}})$.
    if $d < r_A + r_B$ **then**
      $free \leftarrow$ false
    end if
  end for
  if $\|u - u^*\| < min$ **then**
    $min \leftarrow \|u - u^*\|$
    $argmin \leftarrow u$
  end if
end for
return $argmin$

Fig. 5. Sampling Algorithm used to determine best feasible robot velocity for VO and AVO concepts.

Figure 6 shows the VO algorithm implemented for a robot and single moving obstacle. In the left chart, a, the robot is represented as a green dot. The dynamic obstacle is a blue disk of radius r_b. The cyan region represents the robot radius r_a. In this algorithm the it can be assumed the robot is represented as a point and the moving obstacle is represented by a radius equal to the Minkowski sum of the robot and obstacle radii. The additional yellow zone around the cyan represents an uncertainty factor to account for control sampling density in the algorithm and in the real-world would represent measurement or knowledge uncertainty of the obstacle shape. It was found that the algorithm computation time increases as the number of sampled controls increases. This has the benefit of more precise velocity cones, but slower performance. To overcome this, the sampling number can be reduced and uncertainty factor increased as this has the effect of artificially increasing the combined radius and thus increasing the size of the VO cone. This ensures that updates to the robot can be made quickly while still ensuring safe, collision-free traversal. The arrow on the blue obstacle indicates the velocity it is traveling while the arrow on the robot indicates the chosen velocity to avoid collision.

The middle chart, b, represents the theoretical velocity obstacles and valid velocities the robot can take at some time instance over a specified time horizon. Red represents invalid velocities or velocities that will result in collision over the time horizon while green are collision-free velocities the robot can take over the time horizon. The VO cone can clearly be seen in this image. The right figure, c, represents the results of the algorithm implementation to calculate the velocity obstacles for the robot at each time step and shows

which velocity the robot should choose that is as close the preferred velocity as possible while avoiding collision with the obstacle. The middle and right figures showing similar results indicate successful algorithm implementation. It can also be seen on the right how the algorithm samples and test the controls to a density corresponding to the number of controls selected. The lower the number of sampled controls, the less dense the red and green regions would be and potentially lead to unsafe velocities as the algorithm would not be able to accurately calculate the VO cone. Figure 7 shows the results of the same environment with a low control sampling density. It can be seen the shape of the VO is not as full and precise as the higher sampling version.

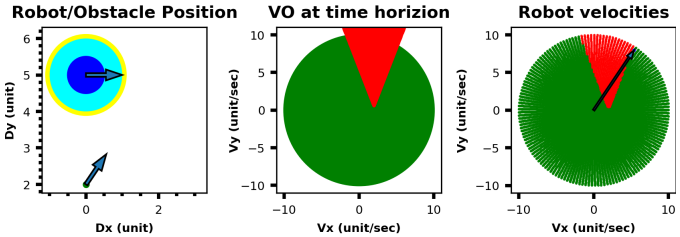

Fig. 6. Velocity avoidance concept: a) Robot and dyamaic obstacle in an environment, b) Theoretically calculated velocity obstacle, c) Velocity obstacle calculated from control sampling algorithm

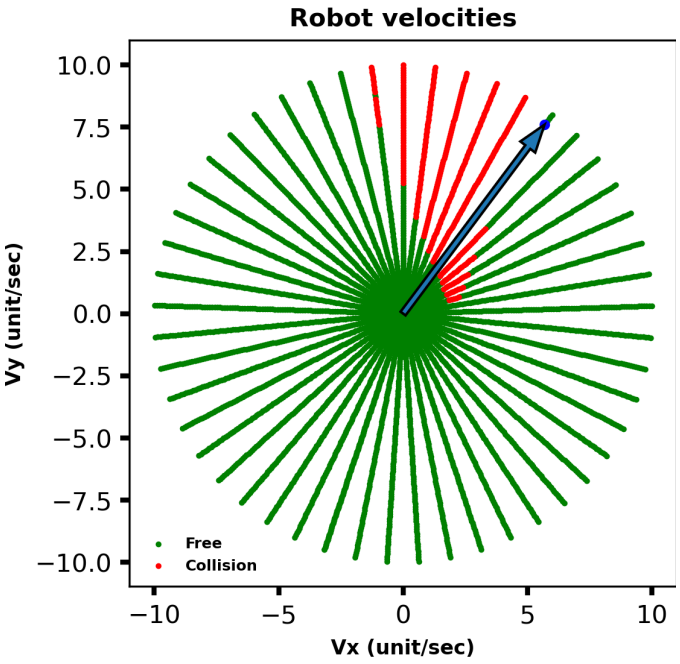

Fig. 7. Velocity Obstacle generated with low number of sample controls.

Although the velocity obstacle method is designed to navigate moving obstacles, it has the ability to take static obstacles into account as well. A static obstacle can be thought of as a dynamic obstacle with a velocity of 0. Therefore, as long as the robot has the same capability to measure or know the position and shape of the obstacle, a VO can be computed using the same sampling strategy as detailed for dynamic obstacles. Figure 8 shows the same details of the VO method for a single static obstacle in front of the robot. Here the static obstacle is black, the robot radius added to the obstacle is represented in gray, and the uncertainty is again yellow. The algorithm is still able to compute the VO in the case and have the robot choose a velocity in a direction that prevents it from colliding with it.

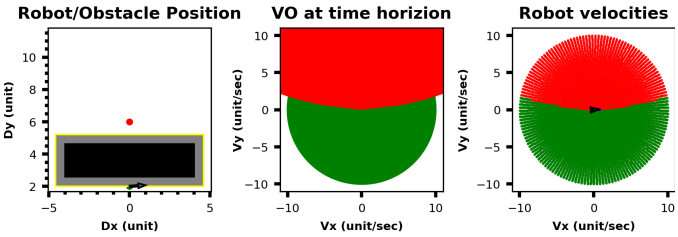

Fig. 8. Velocity obstacle generated from a single static obstacle in front of the robot. Used to verify algorithm compatibility with static obstacle.

Figure 9 shows the robot situated between two static obstacles on its way to the goal in red. The theoretical VO and sampled VO can be seen in the middle and right images showing they are in agreement.

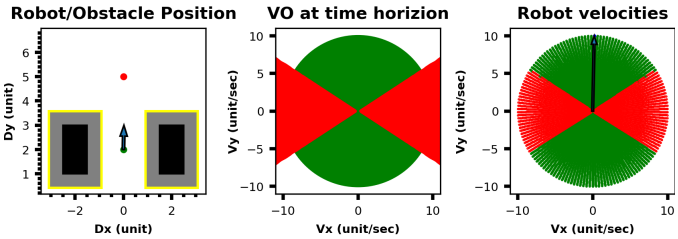

Fig. 9. Velocity obstacles generated for robot passing in between two static obstacles. Used to verify algorithm compatibility with more than one static obstacle in environment.

Finally, the entire scene with both static and dynamic obstacles can be generated using the sampling method. Figure 10 shows how the VOs begin to overlap each other as more obstacles are present reducing the amount of collision-free velocities available for the robot to take. It can be seen that the robot is taking advantage of the fact the dynamic closest to the goal in red is moving directly left creating an opening for the robot to maneuver behind it as it moves toward the goal. This algorithm will be run at each time step during the simulation as robot moves from the start position and terminate once the goal position has been reached.

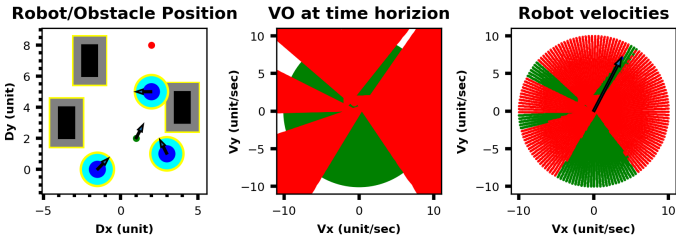

Fig. 10. Velocity obstacles generated for a full scene including both static and dynamic obstacles. Demonstrates successful implementation of the algorithm to handle complex environment.

B. Acceleration-Velocity Obstacles

The second obstacle avoidance method implemented is a variant and extension of the velocity obstacle concept to include an acceleration constraint. It was discussed in [15] that one drawback to the general VO method is that is assumes the robot is able to instantaneously change its velocity to whichever new safe velocity the algorithm finds. In a real-world application this is not the case as the robot is constrained by how quickly it can accelerate to a velocity and how quickly it can change direction. The generalized VO method also assumes the dynamic obstacles move with a constant velocity. This may not be realistic depending on the application and therefore the authors of [15] address this with the concept of acceleration-velocity obstacles (AVO). The same algorithm scheme is used for this implementation with the addition of the robot being able to measure the acceleration of the moving obstacles in addition to the position, velocity, and shape. Incorporating the acceleration into the robot kinematics and integrating to obtain the velocity and position equations results in a non-linear acceleration-velocity obstacle cone. This is illustrated in Figure 11. The same environment setup with a robot and single dynamic obstacle can be seen in in the left figure. The dynamic obstacle has a velocity to the right with acceleration to the left (deceleration). The resulting AVO has the robot select a velocity on the other side of the velocity cone than was chosen in the original VO method. The theoretical AVO is again shown in the middle figure and the AVO computed from the sampling algorithm can be seen on the right. There is one additional difference to note: the yellow region that overlaps the valid velocities. This is a subset of the total valid velocities that takes into account the robot's ability to accelerate over a given time. The robot has a max acceleration magnitude, a_{max}, it is able to achieve that is dependent on the the robot design. A control parameter, δ, is used that represents a time required to achieve the new velocity from the current velocity. The resulting robot valid velocities can be computed by checking all VO valid velocities calculated from the algorithm against the equation $a_{max}\delta$. The robot will then choose a velocity within this subset that is closest to the preferred velocity. This can be seen in right most image in Figure 11. At each time step, the AVO algorithm will sample a number of velocity controls to obtain valid velocities, a subset of valid velocities the robot can actually attain, and the chosen velocity closest to the preferred velocity. From here, an acceleration can be computed to apply to robot to update its current velocity and position.

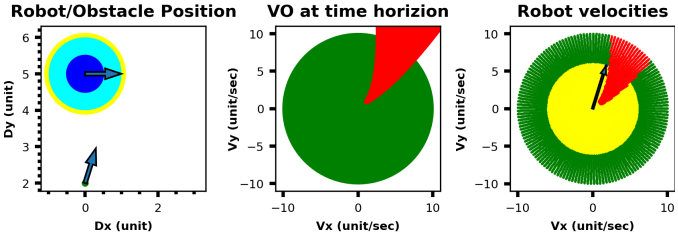

Fig. 11. Acceleration-Velocity avoidance concept: a) Robot and dyamaic obstacle in an environment, b) Theoretically calculated acceleration-velocity obstacle, c) Acceleration-velocity obstacle calculated from control sampling algorithm

Static obstacles again could be taken into account with the assumption they are moving obstacles with zero velocity and zero acceleration. As long as the robot has the capability to measure or know the shape of the static obstacle, it can utilize the same sampling algorithm process to obtain the computed AVOs. Figure 12 shows the same single static obstacle environment with the robot and the calculated AVO. It can be seen it chooses a different velocity than the VO method as it must select a velocity the robot can realistically move to in a given amount of time. Therefore, the chosen velocity is selected in the yellow region point in the opposite direction of the obstacle rather than moving along the edge as was done in the VO method.

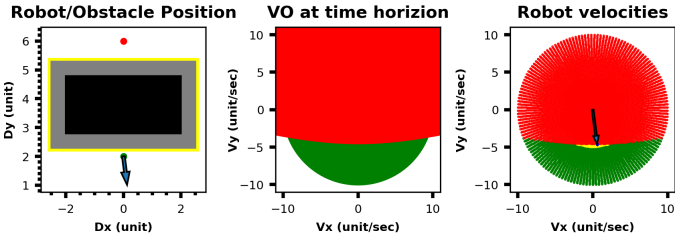

Fig. 12. Acceleration-velocity obstacle generated from a single static obstacle in front of the robot. Used to verify algorithm compatibility with static obstacle.

Figure 13 shows the 2 static obstacle environment with with the robot passing between them again. The AVO cones can be seen to have a non-linear shape compared to the VO method almost as if they are being stretched in plane. The robot is only able to choose a velocity it can realistically accelerate to in a given time period and is therefore forced to choose a smaller velocity than VO but in the same direction.

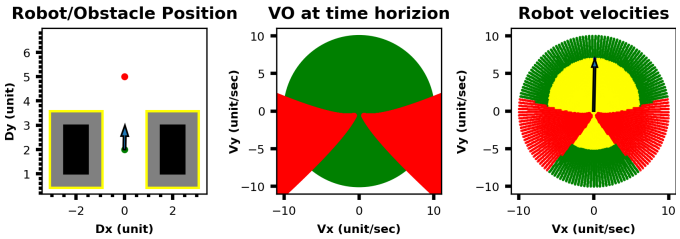

Fig. 13. Acceleration-velocity obstacles generated for robot passing in between two static obstacles. Used to verify algorithm compatibility with more than one static obstacle in environment.

Finally, the full scene with static and dynamic obstacles is shown again with the AVO method in Figure 14. The positions and velocities of all the obstacles are the same, however, different accelerations are being applied to the moving obstacles resulting in a different set of valid velocities as shown in the right figure. Although there appears to be more open velocities for the robot to choose from compared to the VO method in this scenario, the robot is still constrained to a subset of realizable velocities the robot can take as shown in yellow. Either way, both the VO and AVO methods have shown the ability to handle static and dynamic obstacles provided all the necessary information is provided.

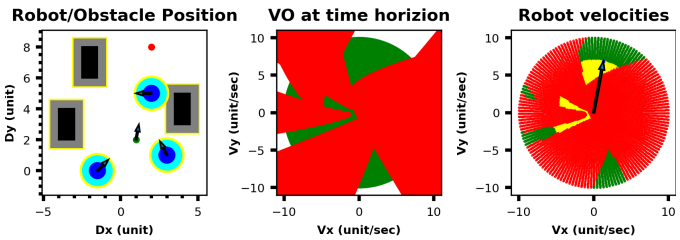

Fig. 14. Acceleration-velocity obstacles generated for a full scene including both static and dynamic obstacles. Demonstrates successful implementation of the algorithm to handle complex environment.

VII. Results

Before describing the results of the methods in detail we felt it important to mention a change to the dynamic environment that was made during the testing phase. During the testing phases of the VO/AVO integration to the environment, it was found the rectangular obstacles were not behaving correctly with respect to the VO algorithm. It was determined this was a limitation in how the static obstacle radii are calculated at each step and the method used may yield incorrect results causing what appeared to be collision in the simulation when there should not be. Circular dynamic obstacles posed no issues during testing and therefore all static obstacles were converted to circular disks as well for all methods to ensure a direct comparison. The new environment with circular static and dynamic obstacles can be seen in Figure 15.

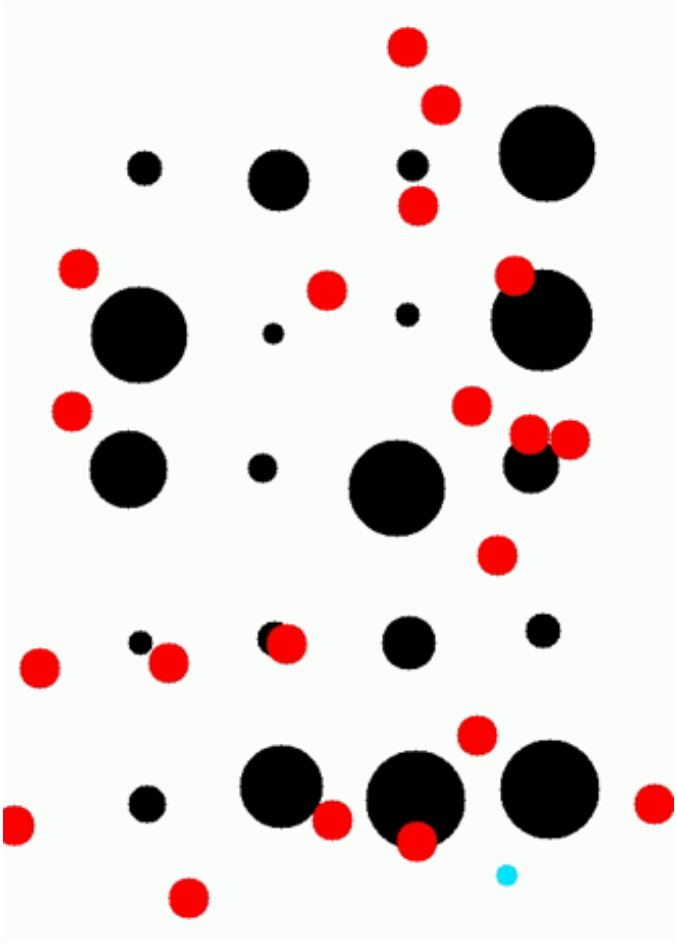

Fig. 15. New dynamic environment with both circular dynamic and static obstacles.

Under the assumption of constant velocity, a fixed map, and 20 moving obstacles, the robot class was used to run 10 simulations using the two velocity obstacle methods and three reactive strategies look ahead, wait, and continue strategies using the RRT* algorithm as a lower-dimensional trajectory. Data was collected on the number of collisions, execution time, and distance traveled to make comparisons between the three strategies. Figures 16, 17, and 18 illustrate the number of collisions by strategy, average distance traveled by strategy, and average time steps by strategy respectively.

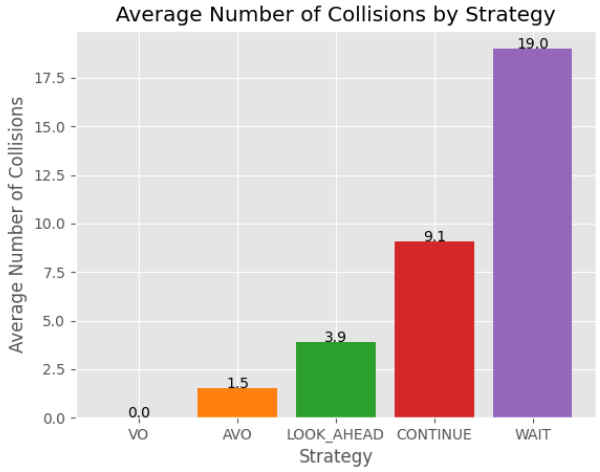

Fig. 16. Average number of collisions over 10 simulations per strategy

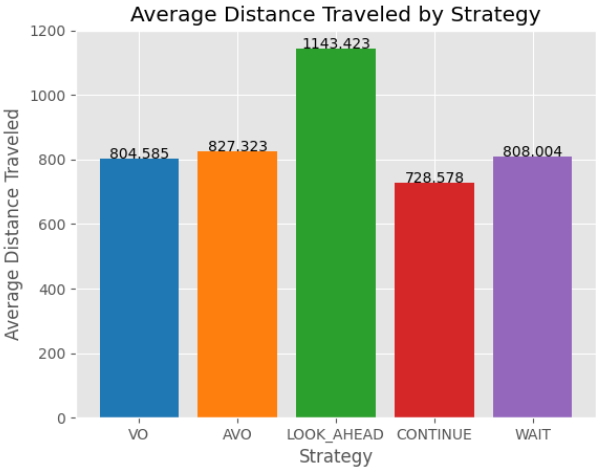

Fig. 17. Average distance traveled over 10 simulations per strategy

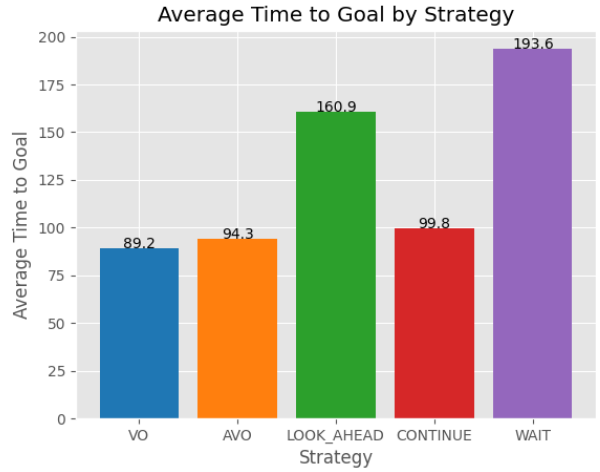

Fig. 18. Average time steps over 10 simulations per strategy

Focusing first on the three reactive strategies, the results show that the LOOK AHEAD method dramatically reduces the number of collisions without significantly increasing the time and distance needed for the robot to reach the goal. The drawback to this method compared to the other reactive methods is the higher computational energy needed to recalculate the path at every instance the robot must avoid a moving obstacle. This may be reduced increasing the number of tree nodes for the RRT* planner and storing for this space to reduce the need to recalculate a new path.

Now comparing the three reactive strategies to the newly implemented velocity obstacle methods, we see from Fig. 16 that the velocity obstacle method has the least amount of average number of collisions as well as the least average time to goal. The average distance is also far superior in the sense that the CONTINUE strategy disregards all moving obstacles as it traverses the optimal path. Compared to the LOOK AHEAD strategy, which had the lowest number of collisions of the three reactive strategies, it is apparent that the velocity obstacle methods are superior in this regard.

From Fig. 18, it is apparent that the time to goal for both velocity obstacle methods are superior to the three reactive methods. Again, the reactive CONTINUE strategy is not a valid comparison since this strategy disregards all moving obstacles regardless of collision as it traverses the path.

Referring to Fig. 17, the majority of the strategies' average distance traveled are comparable to one another. LOOK AHEAD strategy is the highest, as expected, because of its re-planning process.

VIII. Limitations Future Work

Both the velocity obstacle and acceleration-velocity obstacle methods have proven to be an effective approach for robot navigation of dynamic and static obstacles. However, this particular implementation contains certain drawbacks that need to be addressed in order to use in real-world applications. There are also opportunities for future work in this area that may help overcome some of these limitations and allow for implementation of these methods in more complex environments.

The first concession of these methods is that they are implemented entirely in a 2D environment. This may be a good approximation for large open areas where robots/agents move in the same plane, however, the real-world involves 3-dimensions and by not having the velocity obstacle for agents or obstacles that could be above or below the robot limits its application. The implemented methods also assume holonomic robots whereby they can adjust to any velocity heading with no constraints. In reality, robots would have imposed kinematic constraints that will effect performance particularly in more densely populated environments. However, if the robot being used is equipped with omni-directional wheels, then these methods are valid.

It was shown that during the testing process of integrating

the VO methods into the environment, the rectangular static obstacles were problematic particularly when the robot attempted to round the corners. Given the success of the methods with circular obstacles, we believe further investigation is warranted to determine the best way to incorporate polygonal obstacles, both from an algorithm processing perspective, but also as an overall robot sensing perspective. This implementation demonstrates the algorithm can handle static obstacles just as well as dynamic obstacles provided the right information is collected. In a real-world environment, work would need to be done to determine how best to identify static obstacles especially if the environment can only be known locally as opposed to globally.

As mentioned previously, the velocity obstacle method performs well in the dynamic environment as it has no kinematic or dynamic constraints imposed other than the robots maximum velocity capability. This is unrealistic in practice as robots additional constraints and inertia that needs to be accounted for. The acceleration-velocity obstacles attempts to address this somewhat by including a maximum acceleration constraint, however, it still does not have kinematic constraints imposed or inertia represented to provide the most realistic performance. The AVO method already demonstrates that even due to the modest constraints added with acceleration, the robot would need to operate in a less dense environment to allow for enough reaction time to adjust speed and change directions and avoid collisions.

Both the VO and AVO methods were implemented to follow a trajectory generated by an RRT* path for direct comparison with previous methods and was allowed to traverse the enthronement without a path to follow. While the RRT* path allowed for an optimal feasible path to be obtained, the VO and AVO algorithm on its own cannot guarantee the same performance depending on the environment. This means that while the VO and AVO methods are sufficient for collision avoidance, depending on how they are used, they could sacrifice other performance parameters they may be important for an optimal and efficient implementation.

We recommend for future tasks in this space to implement these methods in a 3D environment and incorporate with kinematic constraints such as that of a car or bicycle model. We also would suggest implementing reciprocal velocity obstacle methods as described in [15] to account for the multiple robots in an environment that make decisions independently of the others and avoid the "dance" maneuver that is possible. And finally, we would suggest additional work be done to handle robot sensor data with uncertainties and process it for determining the location of static obstacles and use in the VO and AVO algorithms.

IX. Conclusion

This paper presented and compared several methods that aim to reduce the risk of collisions between mobile robots and obstacles in dynamic environments by implementing a dynamic path planning algorithm that can rapidly react to dynamic obstacles and efficiently re-plan a path avoiding these obstacles to reach the goal. We assumed the robots move in 2D workspace and capable of omni-directional velocity and acceleration.

We implemented various velocity-obstacle variants with moving obstacles and static obstacles and compare the performance to the baseline reactive methods. It was shown that the VO and AVO methods out performed the reactive methods in terms of collision avoidance and average time to goal. However, the average distance traveled was on par with other reactive methods. This is likely due to the pre-planned RRT* path generated that each method followed. This represented a feasible and likely optimal path for the robot to take to avoid static obstacles in the workspace. Though the VO and AVO methods are the most likely to avoid collision during robot traversal, without the pre-planned RRT* path present, these methods may not always be the most efficient in terms of time and distance traveled though they are capable of navigating the environment without the need of a global planner provided the robot is able to sense or know enough information locally to implement the obstacle avoidance algorithms.

X. Roles and Responsibilities

A. Program Manager - Jason Munger

The Program Manager was responsible for reviewing the timeline with team members to ensure key milestones are met on time. This person facilitated action items between team members. The program manager also worked with Technical Expert I on the development of the velocity obstacle avoidance algorithm.

B. Technical Expert - Rene Verduzco

Technical Expert I worked on the development of the environment simulator, velocity obstacle avoidance algorithms, integration of the different strategies into environment, and ran simulations to collect data.

C. Technical Writing Fellow - Clinton Williams

Technical Writing Fellow was in charge of reviewing the simulated data from Program Manager and Technical Expert and filling out the report and presentation.

References

[1] Z. H. Iram Noreen1, Amna Khan, "A comparison of rrt, rrt* and rrt*-smart path planning algorithms," COMSATS Institute of Information Technology, Tech. Rep.

[2] D. Connell and H. M. La, "Extended rapidly exploring random tree-based dynamic path planning and replanning for mobile robots," *International Journal of Advanced Robotic Systems*, pp. 1–15, 2018.

[3] D. Fox, W. Burgard, and S. Thrun, "The dynamic window approach to collision avoidance," *IEEE Robotics Automation Magazine*, vol. 4, no. 1, pp. 23–33, 1997.

[4] P. Ogren and N. Leonard, "A tractable convergent dynamic window approach to obstacle avoidance," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, vol. 1, 2002, pp. 595–600 vol.1.

[5] C.-x. Shi and Y. Wang, "A local obstacle avoidance method for mobile robots in partially known environment," *Robotics and Autonomous Systems*, vol. 58, pp. 425–434, 05 2010.

[6] J. H. A. Richards, "Robust variable horizon model predictive control for vehicle maneuvering," *Int. J. Robust Nonlinear Control*, 2006.

[7] M. Rubagotti, D. M. Raimondo, A. Ferrara, and L. Magni, "Robust model predictive control with integral sliding mode in continuous-time sampled-data nonlinear systems," *IEEE Transactions on Automatic Control*, vol. 56, no. 3, pp. 556–570, 2011.

[8] J. Kabzan, L. Hewing, A. Liniger, and M. N. Zeilinger, "Learning-based model predictive control for autonomous racing," *IEEE Robotics and Automation Letters*, vol. 4, no. 4, pp. 3363–3370, 2019.
[9] A. V. S. Michael Hoy, Alexey Matveev, "Algorithms for collision-free navigation of mobile robots in complex cluttered environments: a survey," *Robotica 33,no. 3*, pp. 463–497, 2015.
[10] "Motion planning in dynamic environments using velocity obstacles," *Int. J. of Robotics Research*, vol. 17, pp. 760–772, 1998.
[11] F. Large, C. Laugier, and Z. Shiller, "Navigation among moving obstacles using the nlvo: Principles and applications to intelligent vehicles," *Autonomous Robots*, vol. 19, pp. 159–171, 09 2005.
[12] F. Fahimi, C. Nataraj, and H. Ashrafiuon, "Real-time obstacle avoidance for multiple mobile robots," *Robotica*, vol. 27, pp. 189–198, 03 2009.
[13] J. Snape, J. v. d. Berg, S. J. Guy, and D. Manocha, "The hybrid reciprocal velocity obstacle," *IEEE Transactions on Robotics*, vol. 27, no. 4, pp. 696–706, 2011.
[14] J. van den Berg, J. Snape, S. J. Guy, and D. Manocha, "Reciprocal collision avoidance with acceleration-velocity obstacles," in *2011 IEEE International Conference on Robotics and Automation*, 2011, pp. 3475–3482.
[15] e. a. van Den Berg, Jur, "Reciprocal collision avoidance with acceleration-velocity obstacles," *IEEE*, 2011.
[16] J. V. D. B. Wilkie, David and D. Manocha, "Generalized with acceleration-velocity obstacles," *IEEE*, pp. 5573–5578, 2009.
[17] V. Vlasov, J. E. M. Mosig, and A. Nichol, "Dialogue transformers," 2020.
[18] D. Connell and H. M. La, "Dynamic path planning and replanning for mobile robots using rrt," in *2017 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, 2017, pp. 1429–1434.
[19] T. Chin, "Robotic path planning: Rrt and rrt*," Medium Software Engineering, Tech. Rep.
[20] P. S. Foundation, "Classes," 2020, last accessed 3 May 2020. [Online]. Available: https://docs.python.org/3/tutorial/classes.html
[21] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," in *International Journal of Robotics Research*, 2011.