Chapter 1

Introduction

1.1. Motivation

The ability of a robotic vehicle to safely and autonomously navigate among stationary and moving obstacles in an unknown setting is essential to the vehicle's operation in most environments. Until autonomous vehicles can be assured of not colliding with moving and stationary obstacles as they navigate towards their destination, they cannot be widely and generally employed. Therefore, a system is needed which can control the movement of an autonomous robot and allow it to avoid moving obstacles as it reaches a target. This is especially important in the presence of humans (e.g., pedestrians); not only in terms of avoiding collisions, but also that humans will feel comfortable in the presence of such autonomous vehicles.

The advent of autonomous vehicles in military and civilian sectors of society is quickly approaching, and in some cases has already occurred, as demonstrated by the Congressional and DoD mandate that one-third of all military land vehicles be unmanned by 2015 (between 2004 and 2008, the number of robots operating in Iraq increased from around 150 to approximately 120,000) and by the development of the Google Driverless Car [1] [2]. However, while the issue of obstacle avoidance by an autonomous vehicle has previously been addressed and (to some degree) solved in such forums as the DARPA Grand and Urban Challenges and the Google Driverless Car, most of the solutions that have been developed require numerous, prohibitively-expensive sensors and vast amounts of processing power. While these resources may be available for well-funded research projects and military ventures, in order to make more ubiquitous autonomy possible, safe obstacle avoidance must be made possible while using a limited set of lower-cost sensors that require a more modest amount of processing power.

The initial motivation behind this research was to improve pedestrian safety around moving vehicles. However, throughout the course of research, the focus has shifted from exclusively pedestrian safety to developing a method by which all types of moving (and stationary) obstacles can be safely avoided by a lower-cost autonomous vehicle. Building on the foundation presented in this research, pedestrian safety will be revisited as a primary focus in future research [3].

The initial concession made towards achieving lower-cost autonomy was the recognition that there would be only limited sensor data available with which to allow the vehicle to avoid obstacles and that this data would contain a higher degree of error than was desirable. In order to account for the limited and error-prone data, an autonomous obstacle avoidance and navigation method called Velocity Occupancy Space (VOS) was developed (see Chapter 3 and [4] [5]). However, the VOS method was initially designed for an ideal, holonomic vehicle; so in preparation for experimental analysis, the VOS method was extended so that it could be applied to the more common experimental vehicle configuration of a differential drive (see Chapter 4, [6]and [7]). Later, while transitioning VOS from the simulation to the experimental stage, the shortcomings of many potential vehicle platforms became apparent. The VOS extension for a differential-drive vehicle assumes that the vehicle will be able to quickly and repeatably obey velocity or acceleration commands; however, it was found that many vehicles, including some commonly used experimental platforms, do not possess this capability. The actuation systems of these platforms appear to be designed with the assumption that there will be some sort of feedback system—probably a human tele-operating the vehicle - who will be able to account for the error. But, as this was not the case with the platform used for this research, the VOS method was again extended, this time to apply to vehicles with varying degrees of actuation error (see Chapter 5 and [8]).

1.2. Review of Related Literature

The field of robot navigation (particularly the problems of obstacle avoidance and navigation or path planning) has been well researched over the years. Various types of stationary and moving obstacle avoidance algorithms have been developed and successfully implemented for a range of applications. Sensor error is not always taken into consideration with these methods, but it is much more likely to be considered with stationary obstacle avoidance methods than with methods that allow for the avoidance of moving obstacles. Navigation methods also range from high level global planners to simple reactive obstacle avoidance systems that use only local, sensor information. Global path planners, such as those reviewed by Siegwart and Nourbakhsh, allow a robot to navigate along a predefined path in a known environment and assume that the robot's environment is either stationary or that the planner has complete knowledge about the movement of all obstacles [9]. As neither type of environment is very common in the real world, local obstacle avoidance algorithms are often integrated into global planners to allow for some degree of reactive behavior.

1.2.1. Stationary Obstacle Avoidance and Path Planning

There are two main types of configuration space path planers, topological and metric. Topological path planning uses previously known landmarks to direct a vehicle. As the environment assumed for this project is not well enough known for the vehicle to navigate via landmarks, this review will focus on metric path planning, which can easily be broken down into sub-goals for short term path planning.

There are numerous types of metric path planners, the most common of these, that are applicable for obstacle avoidance with limited knowledge of the environment, are roadmap, cell decomposition and artificial potential field methods. Roadmap and cell decomposition path planners generally consist of two stages, representing the environment in configuration space and then an algorithm to determine the best path or roadmap through that space.

1.2.1.1. Roadmap Methods

There are several frequently utilized roadmap representation methods. Using the generalized Voronoi diagram, points are found that are equidistant from nearby obstacles [9]. A path along these points, directed towards the goal location, is an extremely conservative means of obstacle avoidance. Ó'Dúnlaing and Yap described the method of retracting free space into a Voronoi diagram where the vehicle is a disc [10]. Latombe gives a good general description of the Voronoi diagram algorithm and possible variations [11]. Choset and Burdick define the hierarchical generalized Voronoi diagram and show how it can be used for exploration of an unknown environment [12].

Overholt *et al* expand on the concept of the Voronoi diagram and produce Voronoi classifiers and regions [13]. The Voronoi classifiers play the same role in Overholt *et al*'s method that obstacles play in the more traditional Voronoi method (i.e. the Voronoi diagram is produced so that all points forming the path are equidistant from the classifiers). However, instead of using the Voronoi diagram as a path, the authors use it to separate configuration space into Voronoi regions. Within these regions a robot's trajectory is determined by the classifier. The regions and classifiers are iteratively determined to ensure that a robot starting at any location will end up at the goal.

Unlike the Voronoi diagram, which maximizes the path's distance from obstacles, a Visibility graph (or shortest-path roadmap) creates a path that hugs the vertices of obstacles in order to find the shortest path from the starting location to the goal. As the name suggests, a Visibility graph creates path segments from a vertex of an obstacle to all other obstacle vertices that are visible from the initial vertex. The path planning algorithm will then select the shortest set of segments to reach the goal. Nilsson first introduced this idea in 1969 [14]. Both Edelsbrunner and Latombe both provide a thorough description [11, 15]. Oommen *et al* use visibility graphs for robot navigation in an unexplored environment [16]. Through repeated exploration, the robot builds a visibility map of the environment using its sensors after which, it can navigate through the environment without sensors, assuming that all obstacles are permanent and stationary.

After the roadmap is created using one of these methods, or the countless other methods mentioned in literature, a variety of algorithms exist with which to find the best route along the roadmap to the goal. The choice of these algorithms depends greatly on the amount and reliability of the information available to the path planner. Dijkstra's algorithm works by finding the lowest cost path from the initial state to a signal state, assuming that all path costs are non negative. This algorithm does not require a heuristic function to predict the cost from future states to the goal state. Urdiales *et al* use Dijkstra's algorithm to find the shortest path within a multi-level path planning algorithm [17]. Qin *et al* use a particle swarm optimization algorithm after Dijkstra's algorithm to find an optimal path [18].

A*, which is an extension of Dijkstra's algorithm, is often considered one of the best general planners. However to use A*, the path planner must have an admissible heuristic function to estimate (without overestimating) the cost to move from every state to the goal state. As the heuristic function is often not available for unexplored territory, this algorithm, while powerful, is limited in its application and can often only be used for path planning in the local region of an unknown environment. Alexopoulos and Griffin use a variation of A* to find the shortest-time collision free path among stationary obstacles and collision free path among moving obstacles, assuming that the obstacles display only linear, constant velocity movement [19]. Oriolo *et al* use iterative applications of A* to generate local paths in an unknown environment. This method requires the robot to stop periodically to collect information about its surroundings and update its world map [20].

Gilbert and Johnson reformat the path planning problem into an optimal control problem which takes into account both the orientation and the velocity of the robot [21]. They apply their method to the case of a robot manipulator, but the same technique could be used for finding or selecting a path in configuration space that balances the need to avoid obstacles with the ability to quickly reach the goal location.

1.2.1.2. Cell Decomposition

The environment can also be broken down, via cell decomposition, into a grid of either regular or irregular elements. Depending on the technique used to create the grid, this method can be referred to as fixed, adaptive, approximate variable-cell or Quadtree cell decomposition, certainty grids or occupancy grids. For basic fixed cell decomposition, a Cartesian grid is superimposed on the environment. Grid elements are considered occupied if an obstacle resides in any part of the grid element; the path planning algorithm finds a path through unoccupied cells. Lozano-Perez uses cell decomposition for automatic planning of manipulator movement [22]. Moravec and Elfes use wide angle sonar to map an area and classify regions as empty, occupied or unknown [23]. They use multiple, overlapping sensor readings to create a higher resolution map from lower resolution sonar measurements. Thrun uses data from multiple robots to create two- or three-dimensional maps using occupancy grids [24]. Jigong *et al* use cell decomposition along with LGODAM to plan a path while avoiding stationary obstacles and obstacle traps [25].

For irregular grids, if an obstacle falls in part of a grid element, that element is divided up into smaller and smaller segments until each segment is either fully occupied by an obstacle or completely free. Zhu and Latombe use constraint reformulation with hierarchical approximate cell decomposition to reduce the amount of area that contains a mix of occupied and unoccupied space [26]. This method decreases the necessary computational time by allowing large grid elements to be used (in general), without necessitating the use of ‘mixed’ (partially occupied) elements– as individual gird elements can be appropriately subdivided.

1.2.1.3. Artificial Potential Field Methods

Potential Field planners create a field or a gradient throughout the environment based on an attractive force exerted by the goal location and a localized repulsive force created by obstacles that should be avoided. The robot is then treated as a point under the influence of this field and is smoothly guided to the goal. O. Khatib first introduces the artificial potential field concept and uses it to control a manipulator in a complex environment [27]. Borenstein and Koren produce a virtual force field by combining certainty grids with an artificial potential field [28]. However, as Koren and Borenstein [29] showed, potential field-based methods inherently cause steering oscillations when driving between densely-spaced obstacles. To overcome this problem, they developed the Vector Field Histogram Method [29, 30]. Hwang and Ahuja determine how to reach a goal by searching a global graph for the shortest path, a local planner then uses the potential field to avoid obstacles and optimize the path in real time [31]. If the local path proves to be un-navigable, then the global planner determines the shortest detour. The use of a global path reduces the usefulness of this technique for navigation in unknown environments. Montano and Asensio create an artificial potential field using a 3D laser range rotating sensor and show its usefulness on basic tasks such as avoiding obstacles or following walls [32]. Their paper provides a good discussion of the dynamic robot model used in their algorithm. Batavia and Nourbakhsh use a grid-based global potential field to perform the planning and navigation of a personal robot [33]. To create the global potential field, they take terrain type, whether or not the area has been explored and obstacle proximity into account. The ability to consider the risk of navigating through unexplored terrain in order to shorten a path makes this algorithm useful for partially known terrain.

M. Khatib *et al* introduce the rotation and task potential fields, which they refer to as the extended potential field [34]. The rotation potential field takes the vehicle's orientation into account when calculating an obstacle's repulsive field. In this way, a vehicle traveling parallel to an obstacle would not suffer from the same repulsion as a vehicle directly approaching an obstacle. The task potential field allows the vehicle to ignore the repulsive fields of an obstacle that it will not be approaching while completing its tasks.

Borenstein and Koren introduce a variation on the artificial potential field method and the occupancy grid method; the Vector Field Histogram. The Vector Field Histogram uses polar coordinates to prevent the vehicle from assuming trajectories that will approach obstacles while directing it to the goal [30]. This method allows vehicles to navigate smoothly down narrow corridors or between close obstacles, when this is the shortest path, a path that frequently causing steering oscillation with traditional potential field methods. Later, Ulrich and Borenstein introduce VHF+, to improve reliability and smooth the robot trajectories, and VHF*, a combination of VHF+ and A*, to deal with traps that arise from typical short term planning methods [35, 36].

1.2.2. Avoidance of Moving Obstacles

The ability to avoid moving obstacles is necessary for robots that must perform tasks in environments that contain vehicles, people or other non-stationary objects. Dynamic obstacle avoidance, while navigating to a goal, is a rapidly growing field due to the increasing number of situations where mobile robots or other autonomous systems are present. Numerous papers and a few books have been published on this subject; however no single method appears to be universally preferred, perhaps due in part to the wide variety of environments and applications for which autonomous or semi-autonomous robots are being used.

1.2.2.1. Adapted Static Obstacle Avoidance

Velocity Control

One of the simplest forms of motion planning in a dynamic environment involves generating a path among any static obstacles using traditional path planning algorithms (see the previous section on static path planning) and then modifying the robot's velocity along that path, in real-time, to avoid dynamic obstacles. While this method is often successful at reaching the goal without encountering an obstacle, it cannot be guaranteed to find a time optimal path and in certain situations proves unable to avoid the dynamic obstacles. It does, however, reduce the computational time needed to determine a path for the robot as dynamic obstacles that do not intersect the pre-planned path can be ignored and obstacles that do intersect the path need only be located a short time in advance and considered only in terms of speed adjustment.

Kant and Zucker present this method and use modified velocity profiles along an original static path which was generated via a visibility graph approach [37]. Lee and Lee, and Fujimura and Samet also use a combination of velocity control along a visibility graph generated path [38, 39].

Re-planning

Another method that utilizes global planning methods to avoid dynamic obstacles involves planning a path, using a very fast algorithm, around static obstacles and the present location of dynamic obstacle (treating the dynamic obstacle as temporarily static). As time progresses new paths are planned to take into account any change in the environment. While these paths are not always optimal, due to the continuous re-planning, they are well suited for environments where dynamic obstacles move infrequently or only small distances, such as an office where a chair may be shifted or a drawer opened. Oriolo *et al* generate local paths within an explored area while building a global map [20]. Konolige uses a gradient field to locally evaluate paths and determine if they are obstacle free [40]. This technique can easily be scaled to include more and different sensors. Fujimori *et al* adapts a direct navigation algorithm to also allow a robot to also detect and avoid obstacles in real time while respecting the dynamic limitations of the robot [41]. However, as noted in the paper, restrictive and unrealistic conditions must be placed on the robot and obstacles to achieve navigation and collision avoidance. A more generalized version of this algorithm without such restrictions would have to be developed in order for it to be of practical use.

Obstacle Circumvention

Zhuang *et al* use a path planner with a Visibility graph-like obstacle avoidance scheme to follow a very direct path to a goal [42]. A robot, under this algorithm, follows a straight path to a goal (without any static obstacle pre-planning). When the algorithm detects an object within the current planning window the algorithm determines if it is a static or dynamic obstacle. If the obstacle is static, the algorithm plots sub-goal(s) to allow the robot to efficiently circumnavigate the obstacle and return to its original path. If the obstacle is dynamic then the algorithm uses auto-regression to predict the obstacles future position and the robot circumnavigates that position, updating its path in real time.

While this method does not always find a time or distance optimal path, no global knowledge is required which drastically reduces computation time. The short-term planning limit also makes the path planner more flexible for unknown and poorly known obstacle dynamics. However, modifications would have to be made to adapt the planner for uncertainty in the location of immediate obstacles.

Elastic Band Concept

Elastic bands, as proposed by Quinlan and Khatib, are intended to close the gap between path pre-planning and obstacle avoidance [43]. As such, they can convert a path that is planned around stationary obstacles and which contains discontinuities and other actions that are kinematically or dynamically impossible for the robot to accomplish into a smooth path that the robot can navigate. This is done by considering the path as an elastic band acted on by two forces, a contraction force that removes slack from the path and a repellant force that moves the path away from obstacles. To reduce computation, the elastic band can be considered as a series of overlapping ‘bubbles’ of free space centered on the path. In this way only the open area around the path needs to be considered when re-planning instead of all of configuration space. The elastic bands also allows the robot to avoid moving obstacles in real time, as the repellant force from the obstacles will push the path away collision causing trajectories.

The elastic band concept is similar to the artificial potential field method in that it considers obstacles as producing a repellant force that is used to direct the robot away from the obstacles. However, the global pre-planning limits the use of elastic bands in unknown environments. Elastic bands applied in conjunction with a simple “bug” obstacle avoidance algorithm (where the robot moves as directly as possible towards the goal while following the external contours of any obstacles that blocks its path) would produce a smoother, safer path without the need for pre-planning.

1.2.2.2. Dynamic Obstacle Avoidance

Path Planning with a Time Dimension, State-Time Space

Fraichard presents a way to plan trajectories in a dynamic workspace which he entitles the ‘state-time space’ approach [44]. If the current position of all static and dynamic obstacles and the velocity and acceleration of all dynamic obstacles is fully known before navigation, three dimensional path planners can plot an obstacle free path through ‘state-time space,’ in which the environment at each time step is treated as a two dimensional plane with time as a third dimension. Depending on the path planning algorithm used with this method, an optimal path to the goal (if one exists) can be guaranteed to be found.

However, the limitations on this method make it difficult to apply to real-world scenarios. While, as previously mentioned, methods exist to predict future (specifically) human movement none of these methods has nearly the certainty of performance that would be needed in order to use the ‘state-time space’ approach to its fullest advantage. In addition, the computational cost is prohibitive to completely up-date the path as changes in the velocity of obstacles are detected in the three-dimensional world.

Kindel *et al* apply kinematic and dynamic constraints to the robot space-time planning and apply their results to an experimental robot with an overhead vision system [45]. Their method is effective only for static obstacles and moving obstacles with a constant, linear velocity.

Yu and Su use a variation of ‘state-time space’ planning but limit the region of planning by focusing on “observation space,” the area that the robot can sense, and “work space” the obstacles that are close to the robot [46]. They also make extensive use of path repair algorithms to deal with dynamic obstacles and the inability to completely predict their future movement.

Genetic and Evolutionary Algorithms

Wang *et al* use genetic algorithms to generate a path around static obstacles and the predicted collision points of dynamic obstacles based on a polygon representation of the obstacles [47]. They reduce the calculation time needed for off-line planning and path re-calculation by considering only the vertices of obstacles using vertex++.

Xiao *et al* develop and revise the Evolutionary Planner/ Navigator (EP/N) [48, 49]. This planner/navigator can utilize specific environment knowledge to enhance its path planning performance. The planner/navigator's ability to self tune for a given environment is valuable; however the need to repeatedly navigate through the environment in order to accomplish the tuning reduces the usefulness for unknown environments.

Han *et al* use genetic search algorithms to generate a goal directed dynamic path [50]. Their use of a cost function instead of global optimization decreases computational time and allow for efficient, real-time navigation. Sugihara and Smith use a genetic algorithm for path and trajectory planning [51]. Their method is suitable for pre-planning as well as real-time motion planning.

Gradient Methods

The potential field method of obstacle avoidance, described by Borenstein and Koren in the previous section on static obstacle avoidance, can be adapted for dynamic obstacle avoidance [28]. O. Khatib in his initial artificial potential field paper surmises that a combination of high level (global) path planning with low level (local goal) planning could allow a manipulator to avoid moving obstacles [27].

Malik, on the other hand, develops the concept of the Extrapolated Potential Field, which predicts an obstacle's path and uses a time and distance weighting scheme to generate a path, for the robot to the goal, which avoids all obstacles [52]. Similar to the static potential field planners, this method is usually quite fast at generating a path but will often miss potentially shorter routes between obstacles that are close together. As Borenstein and Koren mentioned for the static case, this type of planner is also subject to oscillation and can run into problems with local minima when confronted with a combination of static and dynamic obstacles.

The Linear Programming Navigation gradient method (LPN) was originally developed for static obstacles by Konolige, but Farinelli and Iocchi modify this method for environments with dynamic obstacles [40, 53]. The LPN method uses numerical artificial potential fields that take both intrinsic (situational) and adjacency (movement) costs into account to compute a gradient using a generalization of the *wavefront* algorithm. The dynamic variation (LPN-DE) computes the projected motion of the obstacle and increases the weight of the region where the obstacle is predicted to travel to account for future movement.

Inevitable Collision States, ICS

Fraichard and Asama propose and explore the concept of *inevitable collision states (ICS),* the use of which allows a vehicle to plan safe motion around obstacles [54]. These states take into account the robot's future positions as well as its kinematic and dynamic properties in addition to the positions and velocities of all detectable obstacles. In this way, the robot is assured of maintaining safety as it never reaches a position where it cannot avoid a collision (either by changing or maintaining its current state).

Parthasarathi and Fraichard limit the set of trajectories that are considered with ICS so that only a conservative subset of future vehicle trajectories (which are modeled on observed behaviors from other objects in the environment) are considered [55]. They also adapt ICS for a vehicle with car-like dynamics.

Martinez-Gomez and Fraichard develop ICS- AVOID which prevents a vehicle from moving to ICS[56]. They also compare ICS to velocity obstacles and the dynamic window obstacle avoidance approach and determine that the ICS approach is superior due to the way in which it reasons about the future and is able to select safe controls.

Velocity Obstacles

Fiorini and Shiller develop and Shiller *et al* expand upon the notion of the velocity obstacle [57, 58]. Velocity obstacles are a first-order method of motion planning that use robot and obstacle velocities directly to avoid collisions in time-varying environments. This method computes a collision cone of robot velocities that will lead to probable collisions with an obstacle, based on the obstacle's current (and in later works) projected velocity. The velocity obstacle method takes the dynamic constraints of the robot into consideration to narrow down the field of potential robot velocities. However, the shape and dynamics of the obstacle must be well known in order for this method to be effective.

In later papers, Large *et al* adapt the velocity obstacle concept to account for risk and long obstacles (such as hallway walls) and non-linear velocities [59].

Yamamoto *et al* apply the velocity obstacle concept to situations more likely to be encountered in the real world including obstacles that change velocity during sensor cycles and they also introduce the idea of a collision distance index to prioritize the avoidance of obstacles that are closer (and thus pose a more imminent threat) to the robot [60].

Probabilistic Velocity Obstacles

Kluge and Prassler develop *probabilistic velocity obstacles* (PVOs) where a probabilistic collision cone is developed for each obstacle and these are combined to form composite probabilistic velocity obstacles [61].

Fulgenzi *et al* combine Probabilistic Velocity Obstacles (PVOs) with a Bayesian Occupancy Filter (BOF) [62, 63]. These authors use the BOFs to represent the obstacles and estimate their velocities in an unknown and uncertain environment and then employ the PVOs to find safe robot velocities. More details on this method are provided in Section 1.2.4

Dynamic Window

The dynamic window approach, which can also be used as a simplifying adaptation on other algorithms, reduces the complexity of path planning by only considering velocities that the robot can reach safely within a short time interval. Using this method, all of the safe and reachable velocities of the robot make up the dynamic window, which is represented in velocity space. On its own, the Dynamic Window Method is best suited for static environments or environments that have few, slowly moving dynamic obstacles. However, it can be a very powerful tool when combined with other algorithms.

Fox *et al* use the dynamic window approach to account for the robot's dynamic constraints and applied the algorithm to their robot RHINO [64]. Brock and O. Khatib propose the global dynamic window approach to combine path planning with real-time obstacle avoidance in order to safely navigate in a dynamic environment while approaching a goal [65].

Seder and Petrovic combine the dynamic window method with the D* algorithm to enable long term path planning with obstacle avoidance [66]. Later, they also allow for the avoidance of moving obstacles by adapting dynamic windows to avoid *moving cells* with known trajectories by performing obstacle/robot collision checking at fixed time intervals [67].More details on this method are provided in Section 1.2.4

Probabilistic Methods

Probability based data association methods have also been used to track multiple moving obstacles. Schulz et al and later Almeida and Araujo use a Sample-based Joint Probabilistic Data Association Filter (SJPDAFs)in order to accurately track the state of a moving object and propose that this knowledge could be used for autonomous navigation [68, 69]. While this method is fairly accurate, the computational load is very high (necessitating a low sensor sampling rate) and grows exponentially with each additional object that is tracked. Benenson *et al*, a Bayesian estimation form of SLAMMOT is used to detect and track obstacles while a Partial Motion Planner combined with the Inevitable Collision State formulation are used to direct the vehicle [70].

Rapidly-exploring Random Trees, RRTs

Rapidly-exploring Random Trees (RRTs) can be used to perform navigation while avoiding obstacles and accounting for vehicle constraints by utilizing a high dimensional state space [71, 72]. Kuwata et al adapt the RRT to perform on-line planning for an actual vehicle in a dynamic and uncertain environment through the use of lazy check, a risk penalty tree (as well as other extensions) [73].

Fulgenzi et al combine RRTs with Gaussian Processes to allow for the avoidance of moving obstacles when path planning [74]. The future motion of an obstacle is modeled as a Gaussian Process and the RRT planner avoids paths that have a high probability of leading to a collision.

1.2.3. Detailed Literature Comparison

VOS (as developed in Chapter 2) will be compared in detail to the previously mentioned BOF/PVO method [62, 63] and the Dynamic Window method [67]. All three of these algorithms are velocity space based, reactive obstacle avoidance and navigation methods. Because VOS has been designed to fulfill the same purpose as these algorithms its performance and characteristics will be analyzed in comparison in order to validate the usefulness of this contribution.

1.2.3.1. Summary of Comparison Methods

BOF/PVO

The first comparison method is a combination of Bayesian Occupancy Filters (BOF) and Probabilistic Velocity Obstacles (PVO) [62] and is conceptually similar to the VOS method developed earlier in this chapter.

A BOF is used to determine the probability of occupancy of each cell in occupancy space and create a probabilistic distribution function (pfd) of the velocities of the obstacles that occupy these cells. The pdf is translated into a three dimensional grid, where each slice represents a specific obstacle velocity value and then this grid is used to estimate the obstacle’s next location so that filled cells can be tracked and clustered into obstacles. Using this information, probabilistic velocity obstacles are created, and used to calculate the probability of a collision for each velocity and velocities that are deemed sufficiently safe are retained. This subset of safe velocities is then evaluated to find the safe velocity with the lowest difference between the velocity direction and the direction of the goal and this velocity is used as the next robot command[63].

DW

The original dynamic window (DW) [64] method is a velocity space based stationary obstacle avoidance method. This algorithm operates by translating the configuration space locations of obstacles into occupied locations in velocity space (v,ω). A subset of admissible velocities - velocities which fall within the robot's kinematic and dynamic bounds and allow the robot to break before colliding with an obstacle – is then created and from this subset a robot velocity for the next time step can be chosen. In [66] dynamic windows were combined with the FD* (focused D*) algorithm to allow for optimal path planning (within the bounds of sensor information). Later, in [67] the dynamic window was again extended to allow for moving obstacle avoidance by calculating collision points between the robot's future locations (based on admissible robot velocities) and future obstacle positions.

1.2.3.2. Obstacle Avoidance

BOF/PVO

Both the VOS and BOF/PVO methods only require data from a scanning range finder in order to avoid obstacles. In addition, both methods take the uncertainty inherent in the range finding sensor into account when building this grid. However, the BOF/PVO method uses a probabilistic representation of occupancy space based on both the sensor characteristics and the velocity distributions for previously observed obstacles.

The occupancy grid for the BOF/PVO method is calculated at every time step using Bayes rule:

$$P_c(Occ|z(t)) = \frac{P_c(z(t)|Occ) \cdot \sum_n (P_{c_{a(n)}}(Occ) \cdot P_{c_{a(n)}}(v_n))}{P(z(t))} \tag{1.1}$$

where $P_c(Occ|z(t))$ is the probability of the cell c being occupied given the sensor observation $z(t)$. The summation term updates the previously constructed grid using the prior occupancy of the antecedent cells, $P_{c_a(n)}(Occ)$, and the cell's velocity distributions, $P_{c_a(n)}(v_n)$.

The BOF/PVO method then creates a pdf for the velocity of cell c based on the probability that the contents of an occupied cell from the previous time step, $P_{ca(n)}$, moved to occupy this cell. The velocities of each cell that might move to occupy the new cell in question are normalized using the equation:

$$\forall n. P_c(v_n) = \frac{P_{ca(n)}(v_n)}{\sum_{n'} P_{ca(n')}(v_{n'})} \tag{1.2}$$

where v_n is a velocity distribution and n' encompasses all velocity probabilities of all possible antecedent cells. This produces a velocity distribution associated with the occupants of each cell. The cells are independently clustered for each time step (there is no obstacle continuity between time steps) based on physical proximity and a similarity in their velocity distribution and a velocity profile for the cluster is developed based on the distributions of each member cell.

The velocity distribution for the obstacles is discretized (usually to integer values) and limited to a specific range. In essence, this algorithm considers the probability of every occupied cell (from the previous time step) in the occupancy space grid moving to every reachable cell in the current time step in order to calculate the probabilities of occupancy and the velocity distributions for the current time step. The larger the number of obstacle velocities that must be considered, and therefore the larger the number of potential new cells that the contents of each previously occupied cell might move to, the more complex the probabilities in the grid become to calculate.

The velocity obstacle can then be formed by determining the probability of a collision, for each robot velocity, $P_{coll}(v)$. This is found by (in short) multiplying the probability of a cell being occupied, $P_c(Occ|z(t))$, by the probability of the contents of that cell having the specific velocity, $P_c(v_n)$, that would cause it to be part of the velocity obstacle and then summing over all obstacles and the number of time steps into the future that are being considered.

DW

The DW method also uses laser range finder data to build an occupancy grid for the positions of stationary and moving obstacles, however this algorithm requires independent knowledge of the of the velocity vector (v,ω) and motion heading of all of the occupied cells that comprise each moving obstacle. Presumably, the DW method could use either of the velocity estimation techniques developed for the VOS or BOF/PVO methods.

Using the obstacle position and velocity information, the DW algorithm computes all future locations of the robot (based on each kinodynamically feasible robot velocity) and the obstacles at specified time intervals. A total of N_t of these potential mutual collisions points are computed and a collision check is performed between the robot and all of the obstacles at each time interval. The earliest collision (if there is one) is used to locally re-plan the FD* path, by considering all of the obstacles as stationary at the position which they will occupy at the time of that first collision.

Comparison to VOS

The VOS and BOF/PVO methods both estimate the velocity of moving obstacles and account for both the uncertainty in the sensor information and in the velocity estimation. The BOF/PVO method creates a discretized velocity distribution for each obstacle which decreases the precision of the obstacle velocity estimates but, proportionally, decreases the relative computational complexity. The VOS method calculates a single velocity value for each moving obstacle as well as a velocity uncertainty, V_u, term for each estimated velocity (see Section 2.3). The VOS method is significantly less computationally complex, but does not capture as much information as the BOF/PVO method, where a broad range of possible obstacle velocities are considered. However, the velocity estimation method used in VOS has been shown to have relatively low amounts of error (<6%) with low speed testing (Appendix A), therefore the more complete information captured by the BOF/PVO method is not necessary around these types of obstacles. In addition, given the success of VOS in simulations with higher speed obstacles (velocities up to $2\frac{m}{s}$), the lower complexity, center of certainty velocity estimation method compares very favorably with the BOF method.

The DW method depends on external information for the velocity of the obstacles and, while it could be modified to partially incorporate the velocity estimation techniques developed by either the VOS or BOF methods, the use of the mutual collision points as the obstacle avoidance method in DW means that incorporating uncertainty in obstacle position or velocity would increase the amount of collision checking required and therefore the computational complexity.

Another difference between the algorithms is the time limited aspect of the BOF/PVOs and DW methods versus the perpetual nature of the velocity obstacles used in VOS. Both the BOF/PVOs and DW methods will only avoid future collisions which occur within a given number of time steps, while the VOS method takes into account all future collisions, regardless of how far in the future they are predicted to occur. This time limitation will not lead to a collision in the near future, but may cause the robot to end up in a trap situation or degrade the quality of the chosen path (as long term avoidance is not considered). The difference between the perpetual and time-limited obstacle avoidance becomes more pronounced for obstacles with constant velocities (or stationary obstacles), but the difference diminishes for more erratic and unpredictable obstacles.

1.2.3.3. Goal Finding

BOF/PVO

The BOF/PVO method uses a goal finding/navigation technique that is similar to that used for VOS. Possible robot velocities are evaluated based on a weighted combination of goal finding and collision avoidance properties. Possible robot velocities are ranked using the equation:

$$K(v) = \alpha \cdot |v| + \beta \cdot h(v, goal) \tag{1.3}$$

where $h(v, goal)$ is the difference between the direction of the goal and the velocity direction and α and β are empirically chosen constants.

Robot velocities that produce a probability of collision (over the specified time period, $(t_0, t_0 + \kappa\tau]$) that is lower than a safety threshold ($P(t_{coll}(v) \in (t_0, t_0 + \kappa\tau] < P_{safe}$) are evaluated as possibly robot velocities using the equation:

$$K^*(v) = K(v) * ((1.0 - P(t_{coll}(v) \in (t_0, t_0 + \kappa\tau])) \cdot \frac{\kappa\tau}{max_v(T_{safe}(v))}. \tag{1.4}$$

The last term of this equation, $\frac{\kappa\tau}{max_v(T_{safe}(v))}$, allows for velocities that will be safe over a longer period of time to be preferred. The most favorably rated velocity is then chosen as the robot command.

Similar to VOS, this method does not perform long term planning and is not suitable for any type of maze situation.

DW

The DW method has a two stage path planning strategy. In the first stage, the FD* algorithm is used to find an optimal path to the goal in a stationary environment – if one is visible. The environment is assumed to be stationary for this planning, with moving obstacles located at future collision positions. However, this variation of the algorithm also uses a 'safety cost map' that, in effect, enlarges the size of obstacles in configuration space in order to encourage the FD* algorithm to plan a path that keeps the robot far away from any obstacles. This improves the safety of the path but can lead to the robot following very inefficient paths around groups of obstacles.

The second stage of the path selection is similar to that used by BOF/PVO and VOS. The DW evaluates potential velocities based on a weighted sum of safety and goal finding priorities using the equation:

$$\Gamma(v, \omega) = \lambda\vartheta_{clear} + (1 + \lambda)\vartheta_{path} \tag{1.5}$$

where λ is the weighting factor, ϑ_{clear} is a clearance measure (which is based on the time needed for the robot to break) and ϑ_{path} is a measure of the velocity alignment with the FD* path. The path planning with this version of the DW algorithm has the ability to plan an optimal path, however it is limited (as VOS and BOF/PVO are) by the extent to which it can perceive the environment (its planning cannot account for occluded areas) and by the need to re-plan with the FD* due to moving obstacles and uncertain sensor information.

Comparison to VOS

All three methods employ a tradeoff between obstacle avoidance and efficient goal seeking when selecting a velocity. The BOF/PVO method is the only method that takes probability explicitly into account; however the use of the P_{safe} threshold also creates the potential for no robot velocity being considering sufficiently safe and the robot performing emergency breaking. This choice may lead to additional collisions with aggressive obstacles and prevents the robot from being able to choose the least harmful potential velocity (this type of threshold led to simulation failures with the original velocity obstacle method in Section 2.5.4.2.). The relative nature of the velocity occupancy space gird in VOS means that the robot will always choose the best (or least harmful) velocity based on the provided information, even if that velocity is just one that leads to the lowest speed collision.

The first stage of the DW method allows for longer term planning, which the VOS and BOF/PVO methods lack. However, both of these methods could be adapted to include this sort of long term planning by incorporating the FD* plan as a series of local goals.

1.2.3.4. Computational Complexity

BOF/PVO

The computational complexity of the BOF/PVO method is dependent on:

1) the size and resolution of the spatial occupancy space grid

2) the resolution of velocity space (i.e. the number of different possible robot velocities)

3) the number of discrete velocities that are in the pdf for each obstacle

4) the number of time steps for which a collision is being avoided

In order to improve the operational speed of the BOF/PVO method, the number of discrete velocities that are considered for the obstacles are usually reduced so that the obstacles are assumed to be moving at one of only a few different speeds in each direction. This assumption improves the processing time of the algorithm[62], but makes it less precise at avoiding moving obstacles.

In addition, the PVO method computes the cumulative probability (over multiple time steps) of a collision occurring based on each specific robot velocity selection. The more time steps in the future that are considered, the higher the complexity in calculating this value, therefore, the PVO is usually limited to only avoiding collisions that will occur a few time steps in the future. This might not unduly affect the algorithms performance if the robot is surrounded by obstacles that frequently change velocities, however it does decrease the robot’s safety for situations where the obstacle velocities are more constant and could cause the robot to end up in a trap situation when around stationary obstacles [62, 63].

DW

The computational complexity of the DW method is dependent on:

1) the size and resolution of the configuration occupancy space grid

2) the number of different possible robot velocities

3) the number of moving obstacles

4) the number of mutual collision points along each potential robot and obstacle path (Nt, this is also a factor of the number of time steps in the future that are being considered)

5) required path re-planning (new FD* plan) based on moving obstacles or discrepancies in sensor information (the algorithm attempts to re-plan locally, but sometimes needs to create an entirely new plan)

The original DW method had very low computational complexity, but the extensions for FD* planning and moving obstacles require significantly more computation. DW can operate more quickly if few mutual collision points (Nt) are considered between the robot [67] and all of the moving cells that make up each obstacle, however if too few points are considered the algorithm may miss a potential collision and choose an unsafe velocity.

VOS

The computational complexity of VOS was designed to be low in order to allow for quicker updates and faster responses to changes in the robot's environment. The computational complexity of VOS is dependent on:

1) the size and resolution of the configuration occupancy space grid (specifically, the number of visible obstacle elements)

2) the resolution of velocity occupancy space (i.e. the number of different potential robot velocities).

VOS is significantly less computational complex than the two comparison methods. Like the other two methods, its complexity does scale up with the size and resolution of configuration space and the number of potential robot velocities. However, unlike these methods VOS (without increasing its computational complexity) looks ahead an infinite number of time steps (see Section 2.3.1.1), considers all potential mutual collision points (assuming constant velocity values) and does not limit obstacles to having one of a specific set of velocities.

In addition, the construction of velocity occupancy space is highly parallelizable due to the independent nature of each robot velocity and, in [3], VOS was programmed using a graphics card and the velocity occupancy space grid was able to be constructed and fully populated in less than 10ms. Therefore, this algorithm can operate extremely quickly and is usually only limited by the speed at which it receives sensor data and the speed at which the robot can receive new velocity commands.

1.2.4. Differential Drive Vehicle Obstacle Avoidance

Most of the previously mentioned obstacle avoidance and navigation methods operate, at the most basic level, by selecting location or velocities for the vehicle in question to assume. Even though some of these methods compensate for sensor error, they select desired vehicle velocities under the assumption that the vehicle is holonomic and can instantaneously accelerate to the selected velocity. While these assumptions are acceptable in simulations, they are not realistic for experimental platforms.

Adding constraints to velocity obstacles, or to other adaptations of velocity space based obstacle-avoidance methods, has been considered by a few different authors. Owen and Montano solve for the time at which a robot (moving at a certain velocity) and a moving obstacle will arrive at the same location in order to differentiate between safe and collision causing robot angular velocities [75]. They model a differential drive robot's path as a circle (given different, constant velocities for each wheel) and the obstacle's path as a straight line and then solve for the locations at which the line and circle intersect. When selecting between safe angular velocities, they choose a velocity command that will allow the robot to reach the desired angular velocity as soon as possible. In a later paper, Owen and Montano use the selected angular velocity as the seed for an optimization process in which they converge on a desirable robot trajectory [76]. Owen and Montano's work differs from the work presented in Chapter 4, in that they select angular velocities and assume instantaneous velocity change (though bounded by acceleration limits), while VOS selects linear velocities and assume only instantaneous acceleration change.

Wilkie *et al* develop generalized velocity obstacles in order to take the constraints of a car-like robot into account [77]. Similar to Owen and Montano, they find the time at which the robot and an obstacle will be at their closest point, given that the robot follows a specific control command (based on its kinematic model). If, at this time, there is not a collision between the robot and the obstacle then the control can be considered collision free. While this method takes the kinematics of the vehicle into account, it does not consider the vehicle dynamics.

Instead of developing a specific algorithm that accounts for vehicle constraints, Minguez and Montano create an abstraction layer that can be applied to almost any collision avoidance algorithm in order to allows the algorithm to innately take any vehicle's shape, kinematics and dynamics into account (even if the algorithm is designed for a holonomic robot) [78]. However, while their method takes acceleration limits into account, it does not appear to account for the time required for acceleration (a necessary consideration for high speed navigation using the velocity obstacle method), instead it relies on commands that are reachable within a short time period.

1.2.5. Vehicles with Actuation Error

Morales and Con Son considered heading actuation error via the interval method that they use to control their robot *Diablo* [79]. They compensate for this error by periodically adjusting their robot's orientation so that it follows a desired path.

Widyotriatmo and Hong integrate sensor and actuation uncertainty into a Partially Observable Markov Decision Process (POMDP) in order to obtain an optimal action policy for a robot at each time step [80]. While using a probabilistic framework to account for actuation error is appropriate for path planning, it is a hazardous choice for performing obstacle avoidance. Even if there is a low probability of a large (and collision causing) actuation error occurring, it is still necessary for the obstacle avoidance system to assume a worst case scenario in order to assure the robot's safety, instead of only compensating for most probable scenario.

1.2.6. Summary of Key Obstacle Avoidance Methods

In Table 1.1, the properties of some of the more pertinent obstacle avoidance algorithms from Section 1.2 that either account for sensor error or allow for the avoidance of moving obstacles (or both) are summarized. The second column in the table referrers to if the methods has been validated through experimental trials, if it has, a relevant source is listed. The general computational complexity of each algorithm is indicated by the rate at which new velocities were produced for the robot (V rate) and the speed of the processor producing these velocities for the experimental trials (or for simulations, if experimental results were not available). The susceptibility of each algorithm to visible local minima is listed as well as the degree to which the method exhibits goal oriented navigation and incorporates sensor uncertainty. If theses later capabilities are fundamentally part of the method, then the property is listed as 'inherent'. However, if goal seeking or sensor uncertainty compensation is independent of the obstacle avoidance method (e.g. a specific navigation method is not inherent to the Inevitable Collision States (ICS) method - ICS could be combined with many different types of navigation algorithms which would alter the amount of computation needed to navigate with this algorithm) then the ability is simply listed as 'yes' and, if appropriate, a relevant source from Section 1.2 is listed. The algorithm's ability to avoid moving obstacles is also summarized. If the algorithm requires knowledge of the positions or velocities of moving obstacles from a source other than a range finder (the velocities are not calculated based on laser range finder data and the algorithm does not account for error in the velocities), then this knowledge is listed as 'required'. Finally, if the algorithm assumes that the obstacles will move in a specific way (i.e. at a specific velocity) then this restriction is also noted.

Table 1.1 Summary of Obstacle Avoidance Algorithms from Section 1.2.
(Please note that this table is not exhaustive)

Method	Experimental Results	Computational Complexity		Visible Local Minima	Goal Oriented Navigation	Sensor Uncertainty	Moving Obstacles	
		V Rate	Processor Speed				Independent Obstacle Knowledge	Restricted Obstacle Velocities
Potential Fields [27]	Yes [27]	100Hz	1986^{x}	Yes	Inherent	Yes [28]	Only Stationary Obstacles	
Vector Field Histogram [28-30]	Yes [28]	337Hz	20MHz	Yes	Inherent	Inherent	Only Stationary Obstacles	
State-Time Space[44-46]	Yes [46]	4-10Hz	333MHz	NMD$^{\pm}$	Yes	No	Required	No
Dynamic Gradient Methods [40, 52, 53]	Yes [53]	10Hz	266MHz	Yes	Inherent	No	Required	No
ICS [54-56]	No	10Hz^{+}	1.6GHz	NMD$^{\pm}$	Yes [54]	No	Required	No
Velocity Obstacles [57, 58]	Yes [81]	3.3Hz	166MHz	Yes	Yes[82]	No	Usually Required	No [59, 83]
BOF/PVO [62, 63]	No	See Section 1.2.3		Yes	Inherent	Inherent	Not required	Yes
Dynamic Window [64]	No	See Section 1.2.3		NMD$^{\pm}$, No for [67]	Yes [65]	No	Required [67]	No
SJPDAFs [68, 69]	Yes [69]*	4Hz	2008^{p}	NMD$^{\pm}$	Yes [84]	Inherent	Not required	No
Probabilistic RRTs [71, 72]	Yes [73]	10Hz	2.23GHz	No	Inherent	Yes [74]	Required [74]	Yes [85]

*Experimental results with a stationary robot, tracking obstacles only
+ Rate with an A* planner
x Processor speed not provided, research was performed in 1986
p Processor speed not provided, research was performed in 2008
± Navigation Method Dependent; susceptibility to visible local minima is dependent on the navigation method in use

The various available algorithms have a range of strengths and weaknesses; however, no single algorithm is generally accepted and utilized. A desirable obstacle avoidance method should have the properties listed in Table 1.2.

Table 1.2 Desired obstacle avoidance algorithm properties

Experimental Results	Computational Complexity		Visible Local Minima	Goal Oriented Navigation	Sensor Uncertainty	Moving Obstacles	
	V Rate	Processor Speed				Independent Obstacle Knowledge	Restricted Obstacle Velocities
Yes	>10Hz	1-3GHz	No	Inherent	Inherent	Not required	No

Unfortunately, none of the review obstacle avoidance methods possess all of these characteristics. Most of the algorithms with low computational complexity and that account for sensor uncertainty are only capable of avoiding stationary obstacles. Of the algorithms that can avoid moving obstacles, those that are faster than 10Hz require independent knowledge of the position and velocity of surrounding obstacles. The few methods that are able to avoid moving obstacles and that account for sensor uncertainty are either too computational complex to operate in real-time or must make very restrictive assumptions about potential obstacles in order to accelerate the processing.

1.3. Original Contributions

Based on the need for an obstacle avoidance algorithm indicated by Table 1.1, the initial goal of this thesis is to develop an algorithm that is successful according to all of the categories listed in Table 1.2 – an algorithm with low computational complexity (the algorithm should produce new robot velocities at a rate of at least 10Hz on a modern laptop computer) that is not susceptible to local minima, which can perform goal oriented navigation and is able to avoid moving obstacles using only uncertain sensor data without independent knowledge of the obstacles or making restrictive assumptions about their velocities. In addition, while most obstacle avoidance algorithms are designed for ideal, holonomic/omni-directional vehicles the algorithm developed in this thesis should be applicable to realistic (non-holonomic) kinodynamic robot configurations and this capability should be demonstrated experimentally.

The primary original contribution of this thesis is the development of VOS which combines the sensor noise and uncertainty representation of configuration occupancy space [23] with the long term avoidance of moving obstacles provided by the velocity obstacle concept [57, 82]. In order to facilitate obstacle avoidance, the ability to estimate the velocity of moving obstacles from configuration space has been developed. Also contributed is the relative weighting scheme between velocities that lead to various obstacles or the goal. This allows the robot to safely avoid obstacles while ultimately navigating towards the goal. This work is described in Chapter 2 and in [4, 5].

The two extensions to VOS, presented in Chapters 3 and 4, are also original contributions. Velocity based navigation has been used by other researchers, but it is almost always assumed that the vehicle being controlled is capable of just assuming another velocity without taking time to accelerate or decelerate. The acceleration based method developed here does take for granted that the robot can instantaneously change accelerations, but this is a more realistic (and less error generating) assumption than instantaneous velocity change. In addition, the acceleration based method adapts velocity based navigation for an acceleration controlled vehicle both for the specific VOS algorithm and for any other type of velocity based navigation. This work is detailed in Chapter 3 and in [6, 7].

Finally, the adaption of VOS for actuation error is an original contribution that allows for velocity based navigation to be used when there is uncertainty about how a vehicle will respond to actuation commands. While the methodology created to compensate for this error can only be used with VOS or other velocity obstacle based obstacle avoidance methods, it does apply to any type of actuation error for which an upper bound on velocity error can be produced. This contribution is detailed in Chapter 4 and in [8].

1.4. Purpose and Scope

The purpose of the research presented in this thesis is to provide a new method of safe autonomous vehicle navigation in an unknown environment in the presence of moving obstacles using uncertain sensor data. This method, termed *velocity occupancy space* (VOS), combines the sensor error and uncertainty representation of certainty grid occupancy space with the velocity obstacle representation of moving obstacles. In addition, VOS allows for active velocity selection which will enable the robot to navigate efficiently and autonomously, as well as perform obstacle avoidance, while moving toward the desired destination. The VOS algorithm is developed and described in detail in Chapter 3 and [4, 5].

VOS has also been extended in order to allow for autonomous vehicle navigation under specific circumstances. First, while the original VOS was designed for a holonomic vehicle, it has been adapted to control a differential drive vehicle with acceleration based actuation. The purpose of this extension is to compensate for the non-holonomic and non-instantaneous acceleration properties of a more realistic experimental vehicle without compromising the safe and effective means of obstacles avoidance and navigation inherent in the original VOS algorithm (see Chapter 4 and [6, 7]).

The second extension of VOS is to allow a linear and rotational velocity controlled vehicle that suffers from a significant amount of actuation error to be effectively operated using VOS. This error may be caused by a delayed motor response, an ill tuned motor feedback system or uncertain terrain – anything which makes the velocity and position of the vehicle difficult to predict and control. As long as the bounds on this error are known, VOS can still be used to provide safe navigation (see Chapter 5 and [8]).

The goal of this research has been to design a system that is inexpensive yet highly versatile; suitable for both military and civilian applications in structured and unstructured environments. As such, no assumptions are made about the types of obstacles that are likely to be encounter and it is assumed that the algorithm has almost no prior knowledge of the environment (no map is provided, nor is a permanent map built). The two assumptions are made is that the local environment is relatively flat so that it can be approximated as two dimensional and that the maximum velocity of all of the obstacles is equal to or less than the maximum velocity of the vehicle – otherwise, the vehicle cannot be assured of avoiding a collision.

However, the scope of the research presented herein is limited to the specific elements of autonomous navigation that are addressed by VOS and its extensions. Over the past few decades, there has been an enormous amount of research devoted to many different aspects of autonomous navigation and obstacle avoidance including sensor development and characterization, vehicle localization, motor feedback control loop design, etc. It is beyond the scope of this thesis to address most of these topics and while some of these other methods have been utilized in order to allow for the simulation or experimental testing of VOS, they do not represent an original contribution nor have they been developed significantly beyond what has been referenced from the work of other researchers.

In addition, VOS has also been designed as a relatively low-level (almost reactive) obstacle avoidance algorithm, so it would not be an appropriate choice for any sort of complex navigation or maze-type scenario. However, VOS has the potential to be integrated with other (higher level) navigation functions that may be available on a UGV in order to compensate for this shortcoming. For example, VOS could be combined with a map/GPS interface – a device which has become common in many commercial vehicles. The map and GPS would provide long term path planning or higher level navigation, but could also provide the VOS algorithm with short-term or moving goals which the algorithm could follow while avoiding local obstacles – a function which is currently performed by a human operator.

In Chapter 2, background will be presented on VOS and on some of the pertinent research from other authors that has been used in order to develop VOS and the original VOS algorithm for a holonomic vehicle that suffers from sensor error will be developed. In Chapter 3, the first extension of VOS for a differentially driven vehicle will be given. The second extension, for a vehicle with actuator error will be presented in Chapter 4 as well as experimental results from testing VOS on an actual vehicle. Finally, in Chapter 5 the conclusions and plans for future work will be given.

Chapter 2

Velocity Occupancy Space (VOS)

The primary contribution of this thesis is the development of a VOS-based obstacle avoidance algorithm, this development was inspired by the sensor noise and uncertainty representation of configuration occupancy space [23] and the long term avoidance of moving obstacles provided by the velocity obstacle concept [57, 82]. The combination of these two concepts led to velocity occupancy space where, similar to a configuration occupancy space grid, individual gird elements are given values based on the likelihood of a collision occurring if the robot adopts the state represented by that grid element. However, instead of each grid element representing a location, as in configuration occupancy space, the elements represent velocities. The collision causing properties of these velocities are determined using the velocity obstacle concept, where all potential collision causing velocities for a robot can be found based on the relative location and speed of surrounding obstacles.

In order to form VOS, the locations of all detected obstacles and the robot's goal are first represented in velocity space based on their respective locations and velocities. Next, velocity occupancy space is populated with repulsive and attractive weights based on the likelihood and speed with which each specific velocity will lead the vehicle towards a collision or towards the goal. Finally, the most advantageous velocity is selected as the vehicle's next velocity. Extensive optimization was performed in order to determine the most desirable way to weight the velocities based on the objectives of safety, rapid and efficient goal finding and smooth operation.

2.1. Background on VOS

In this chapter details are provided on related previous research, including a discussion of how this research will be utilized in this thesis as well as discussion of the implementation and extensions of these methods.

2.1.1. Background on Configuration Space and Timing

Cell decomposition and certainty grids have been used by many researchers in order to allow a robot to navigate and avoid stationary obstacles using uncertain sensor data. Using a distance-finding sensor, such as a laser range finder, the robot can determine the approximate angle, θ, and distance, r, that the obstacle is from the robot (see Figure 2.1). The accuracy and precision of the data collected is dependent on the quality of the sensor in use. As such, when using a low cost sensor a high error rate is unavoidable and a certainty grid can be employed in order to account for data errors. Moravec and Elfes and later Borenstein and Koren use a certainty grid, which gives each cell a certainty value that indicates the confidence that the cell is occupied, in order to represent the uncertainty and error inherent in the sensor measurements [23, 28].

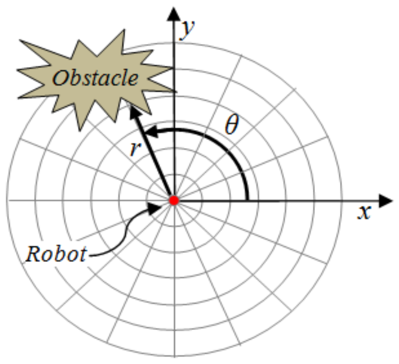


Figure 2.1 The robot detecting an obstacle

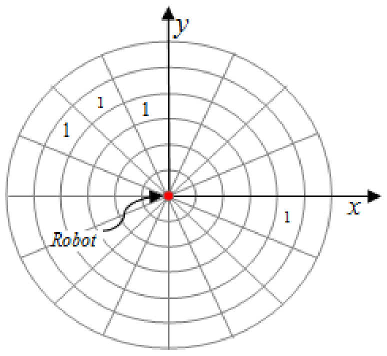


Figure 2.2 Polar space occupancy grid of Figure 2.1

A polar, configuration space grid, Figure 2.2, has been used in order to determine the momentary occupancy of the robot's environment. The occupancy value of each element in the polar space grid is found based on the equation

$$O_p\big(r_i(t_s), \theta_i(t_s)\big) = \begin{cases} 1 & occupied \\ 0 & empty \end{cases} \tag{2.1}$$

where $O_p\big(r_i(t_s), \theta_i(t_s)\big)$ is the binary occupancy value of a region at sensor time step t_s at a radius of r_i and an angle of θ_i from the robot. For accuracy and ease of manipulation, the location of the obstacle in local, polar configuration space is converted into global, Cartesian configuration space using the standard conversion,

$$O_c\big(x_i(t_s), y_i(t_s)\big) = O_p\big(r_i(t_s)\cdot cos\theta_i(t_s) + x_r(t_s), r_i(t_s)\cdot sin\theta_i(t_s) + y_r(t_s)\big) \qquad (2.2)$$

where $x_r(t_s)$ and $y_r(t_s)$ are the coordinates of the robot's position at time t_s.

Three separate time steps are used in this derivation, a motion time step, Δt_m, a sensor time step, Δt_s, and an acceleration time step, Δt_a (which will be defined in Section 4.1). The first two time steps are related according to

$$\Delta t_m = k \cdot \Delta t_s \qquad (2.3)$$

where k is an integer greater than one. The time steps are related in this way as it is assumed that a large number of sensor measurements will be read for every motion command that is produced by the algorithm. Figure 2.3 shows a graphical representation of the time steps.

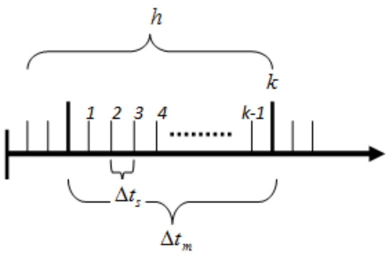


Figure 2.3 Relative time steps

The Cartesian grids are summed for h sensor time steps (see Figure 2.4) in order to compute the weighted Cartesian occupancy space grid (see Figure 2.5), used for the subsequent velocity calculations. The grids are summed using the equation

$$O_c\big(x_i(t_s), y_i(t_s)\big) = \frac{1}{h} \cdot \sum_{\tau = t_s - h\cdot\Delta t_s}^{t_s} \left(\frac{1}{\beta \cdot (t_s - \tau) \cdot \|\vec{v}_r\| + 1}\right) O_c\big(x_i(\tau), y_i(\tau)\big) \quad (2.4)$$

where the $\frac{1}{h}$ term is used to normalize the occupancy grid values so that a change in the number of sensor time steps that are summed to form the occupancy grid does not affect the overall weighting (as detailed later), $\|\vec{v}_r\|$ is the magnitude of the robot's velocity, and β is a user defined variable ($\beta \in [0, \infty)$) that regulates how much influence the time-lag and robot velocity should have on the sensor measurement from each previous time step. By using the later two terms, the most recent sensor measurement is given its full weight while previous measurements have reduced weights based on the time elapsed and the velocity of the robot. These terms help both to reduce the error in the position estimate of moving obstacles as well as compensate for error in the robot's movements.

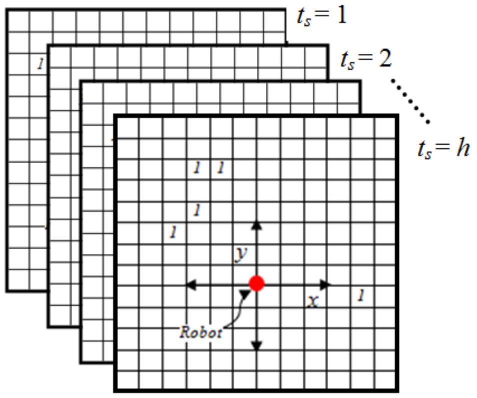


Figure 2.4 Cartesian grid for several time steps

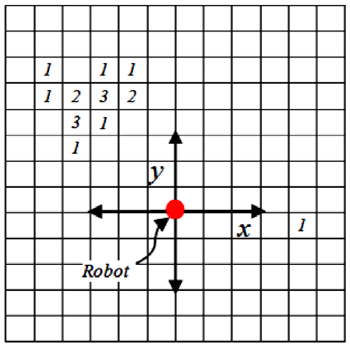


Figure 2.5 Summed Cartesian grid indicating range detection over a past horizon (values are shown before normalization)

2.1.2. Background on Velocity Obstacles

The concept of a velocity obstacle, as first introduced by Fiorini and Shiller and later expanded by Shiller *et al* and Large *et al*, has been used in the development of VOS [58, 59, 82]. Under the velocity obstacle concept, all robot velocities that will lead to a collision between the robot and an obstacle, *Obstacle A* in Figure 2.6 moving with a velocity of $\vec{v}_a$, are considered to be part of the velocity obstacle, VO_A, of that obstacle. In other words, all robot velocities that fall within the velocity obstacle (i.e. the cone shaped area labeled VO_A) will lead the robot to a collision with *Obstacle A.* As long as the tip of the robot's velocity vector, $\vec{v}_r$, remains outside of VO_A and the obstacle's velocity, $\vec{v}_A$, remains constant, then the robot will avoid a collision with *Obstacle A*. The velocity obstacles' of multiple obstacles, *Obstacles A* and *B,* can be combined along with the set of dynamically possible robot velocities (i.e. *Reachable Velocities* in the rhombus-shaped area) in order to find a safe and dynamically feasible velocity for the robot.

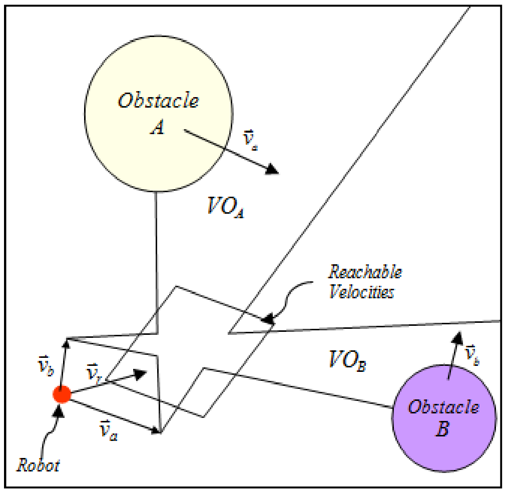


Figure 2.6 Robot and velocity obstacles of Obstacles A and B. Adapted from (Fiorini and Shiller 1998)

2.2. Representing Obstacles and the Goal in VOS

In order to produce the velocity occupancy space for a robot based on uncertain sensor data, the approximate location and velocity of each obstacle and the goal in configuration space must be determined. Using this information, the velocity obstacles, a set of velocities which will lead to a collision between the robot and an obstacle, can be found.

2.2.1. Center of Certainty

Sensor data from a laser range finder is collected in the form of robot-based, polar coordinates of obstacles. It is then converted into global, Cartesian coordinates based on the robot's perceived location and used to build an occupancy grid. The specifications of the laser range finder (LRF) that was mounted on the robot (±30mm accuracy and 0.25 ° resolution) are used when converting the scan data into the occupancy grid in order to account for sensor errors. Occupied obstacle elements are clustered together and the approximate location of each obstacle is found using a variation of the center of mass equation, termed the Center of Certainty, C_j,:

$$C_j\big(x(t_s), y(t_s)\big) = \frac{\sum_{i=0}^{n_j}(x_i, y_i) O_c\big(x_i(t_s), y_i(t_s)\big)}{\sum_{i=0}^{n_j} O_c\big(x_i(t_s), y_i(t_s)\big)} \tag{2.5}$$

where (x_i, y_i) is the location of the element i which has the occupancy value, $O_c\big(x_i(t_s), y_i(t_s)\big)$, at time t_s (see Figure 3.1). It should be noted that the obstacles are numbered as $j = 1,2, \dots, N_O$, where N_O is the number of obstacles that the robot detects throughout the simulation. In addition, each obstacle, j, consists of n_j elements, numbered as $i = 1,2, \dots, n_j$. The center of certainty equation uses the number of times that an obstacle is detected in an element of occupancy space as the certainty of that element being occupied. This data is used to create a weighted average and locate the approximate center of the obstacle.

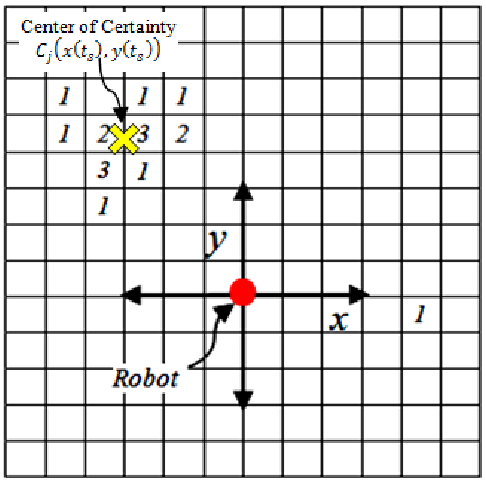


Figure 2.7 Center of Certainty

The velocity of the center of certainty of obstacle j, $\dot{C}_j$, can be estimated by calculating how far the center of certainty of the obstacle moves between sensor time steps using simple differencing techniques. Because the configuration space is discrete, rounding errors are produced when finding the estimated velocities, especially when a lower resolution configuration space grid is used. To compensate for this, the velocities are smoothed by averaging the obstacle's velocities over a number of sensor time steps, h, to find the obstacle velocity at motor time step, t_m,

$$\dot{C}_j(\dot{x}, \dot{y}, t_m) = \frac{\sum_{\tau=t_m-h\cdot\Delta t_s}^{t_m} \dot{C}_j(\dot{x}, \dot{y}, \tau)}{h} \tag{2.6}$$

This method finds the center of the side(s) of the obstacle presented to the robot, not the physical center of the obstacle. For obstacles with a large aspect ratio, this will produce some velocity error when the obstacle turns or the robot circles the obstacle and a new side is presented to the robot. However, using the obstacle's center of certainty (instead of the center of the obstacle's observed physical dimensions) to estimate its velocity and averaging the estimated velocities over multiple sensor time steps decreases the error in the estimated obstacle velocity used for the formulation of VOS. Averaging the estimated velocities does create a delay in the obstacle velocity calculation, as historical position data is used to calculate the current velocity, and this drawback should be considered when selecting a value for h. Low-speed experimental tests showed that the error produced from obstacles with large aspect ratios was usually less than 6% of the actual obstacle velocity for obstacles moving at more than $0.2\frac{m}{s}$ (see Appendix A for details). This method is similar to an approach explored by Fuerstenberg *et al*, but here the ability to operate in any environment by not making any assumptions about the properties of the obstacles is retained [86].

For ease of notation, the velocity of obstacle element i will be referred to as $(\dot{x}_i(t_m), \dot{y}_i(t_m))$. For this notation, the velocity $\left(\dot{x}_i(t_m), \dot{y}_i(t_m)\right)$ is equal to the obstacle velocity $\dot{C}_j\left(\dot{x}(t_m), \dot{y}(t_m)\right)$, if the obstacle element *i* is a part of obstacle *j*.

The method introduced here differs significantly from the more commonly used probability based data association methods, such as those used by Schulz *et al* and Almeida and Araujo, that are used to populate occupancy grids and track obstacles [68, 69]. While these methods will almost always produce more accurate results in terms of locating and tracking obstacles (especially occluded obstacles), most data association techniques are very time consuming and computationally expensive, especially as the number of obstacles that they are tracking increases.

The crude, yet fast, obstacle tracking and velocity estimation method used in the VOS algorithm takes only around 8ms to update the occupancy grid and estimate the obstacle locations and velocities (running in parallel with the rest of the algorithm, described in section 2.3, on a 2.53GHz laptop). As this is faster than the scan rate of our LRF (Hokuyo UTM-30LX, 40Hz) the algorithm is able to make use of all available sensor data. In addition, the early loss of accuracy is compensated for by giving the velocity weighting algorithm the ability to compensate for error and obstacle unpredictability (see section 2.3.1). By shifting most of the computational load from the map building stage to the velocity selection stage much more of the sensor data is utilized, less odometry error is accumulated between sensor readings and the robot is able to recognize and respond more quickly to unanticipated events.

2.2.2. Obstacles and the Goal in Velocity Space

Using the location and approximate velocity that was previously calculated for the obstacle, the obstacle's location in velocity occupancy space can be determined, and from this location the velocity obstacle (i.e. the set of robot velocities that will lead to a collision between the robot and obstacle) can be found.

Figure 2.8 shows the position and velocity, $\vec{v}_r$, of the robot and the position of element i, of obstacle j, and the vector, $\vec{\lambda}_{i,r}$, between the robot and the obstacle element. The center of certainty velocity of the obstacle that element i belongs to is $(\dot{x}_i, \dot{y}_i, t_m)$. While we have clustered the occupied elements into obstacles in order to determine their velocity, we will still use the individual elements from configuration space to populate velocity space so that their occupancy certainty values can be directly utilized.

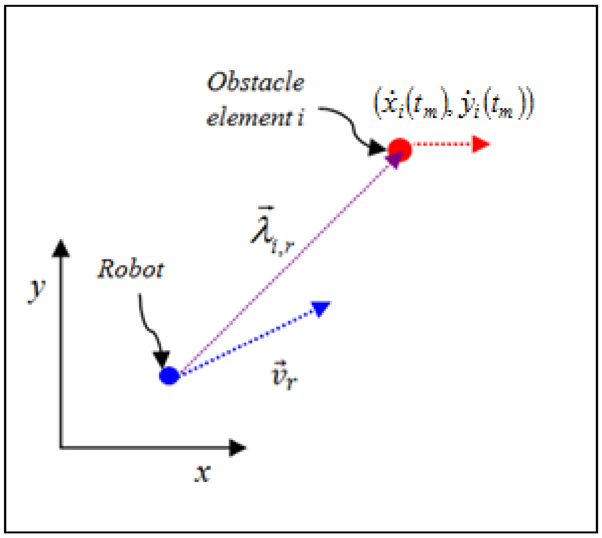


Figure 2.8 Configuration space representation of the robot and an obstacle

In Figure 2.9, the robot and one obstacle are shown in the velocity space. The robot is located at its velocity, $\vec{v}_r$, and the obstacle is located at the sum of the obstacle's center of certainty velocity, $(\dot{x}_i(t_m), \dot{y}_i(t_m))$, and the vector between the robot and the obstacle in configuration space, $\vec{\lambda}_{i,r}(\dot{x}(t_m), \dot{y}(t_m))$. In other words, the obstacle is located at the velocity that the robot would need to assume in order to collide with the obstacle in one motor time step, which takes into account both the obstacle's distance from the robot as well as the obstacle's own velocity. The vector originating at $(\dot{x}_i(t_m), \dot{y}_i(t_m))$ and intersecting, $\vec{\lambda}_{i,r}(\dot{x}(t_m), \dot{y}(t_m)) + (\dot{x}_i(t_m), \dot{y}_i(t_m))$ in velocity space, is the set of collision causing velocities that makes up the velocity obstacle, $\overrightarrow{VO}_i$. Any of these velocities will cause the robot to collide (at some point in time) with the obstacle, assuming constant obstacle velocity.

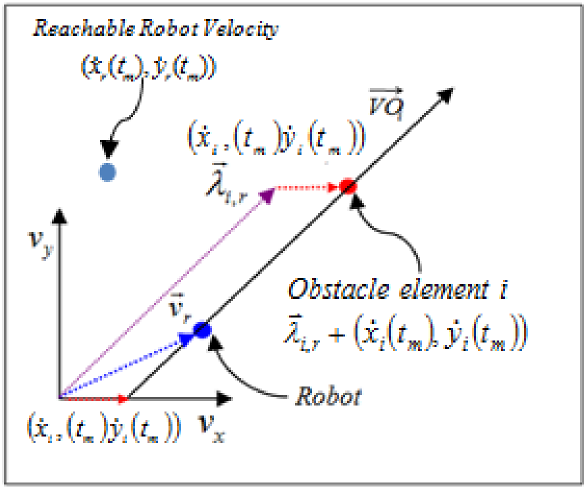

Reachable Robot Velocity
$(\dot{x}_r(t_m), \dot{y}_r(t_m))$
$\overrightarrow{VO}$
$(\dot{x}_i, (t_m) \dot{y}_i (t_m))$
$\vec{\lambda}_{i,r}$
v_y
$\vec{v}_r$
Obstacle element i
$\vec{\lambda}_{i,r} + (\dot{x}_i(t_m), \dot{y}_i(t_m))$
Robot
$(\dot{x}_i, (t_m) \dot{y}_i (t_m))$
v_x

Figure 2.9 Velocity space representation of the robot velocity,$\vec{v}_r$, and the velocity obstacle, $\overrightarrow{VO}_i$

In this research, it is assumed that the relative position and velocity of the goal are always known. As such, locating and tracking the center of certainty is not needed to find the *velocity goal,* $\overrightarrow{VG}$, or the set of robot velocities which will lead the robot to the goal, in the same manner that the velocity obstacle was found. However, if a sensor was employed that could distinguish the goal from surrounding obstacles, then the same technique used for the obstacles could be used to locate the goal, track it and determine its location in velocity space.

2.3. Populating Velocity Occupancy Space

After the velocity obstacles and goal have been found in the velocity space, it is necessary to populate the velocity occupancy space with values in order to select the best robot velocity. The velocity occupancy space consists of weighted elements that correspond to possible robot velocities. The weight of each element is based on two sets of factors. The first set forms a repulsive weight, based on the possibility that this velocity might lead the robot to a collision with an obstacle. The second set is based on how quickly and directly a velocity will lead the robot to its goal.

2.3.1. Repulsive Weights

The repulsive weighting value of each element of a velocity obstacle is influenced by a number of variables that determine how much of a threat an obstacle is and to what degree it should be avoided over other obstacles. The repulsive value, R, of each element is defined by the equation

$$R = W_R\left[D_R \cdot A_R(W_{AR}) \cdot \left(\frac{W_{TTC}}{TTC} + \frac{1}{CD}\right) \cdot E_{Oc}\right] \tag{2.7}$$

where the weights (W_{R}, W_{TTC}, and W_{AR}) are defined and optimized based on the robot's environment, and W_{R} is the overall repulsive weight; which is used to prioritize obstacle avoidance over reaching a goal. The other terms in Equation (2.7) are detailed as follows.

The term E_{Oc} is the occupancy value, $O_C(x_i(t_m), y_i(t_m))$, for the obstacle element with which each robot velocity will lead to a collision. The other terms are variables related to the robot's state and environment and include D_R, which is the repulsive direction term, A_R, which is the repulsive angular term, *TTC*, which is a measure of the time to collision and *CD* which is the Cartesian distance between the robot and the obstacle. It should be noted that, unlike the other weights, A_R, is a factor of W_{R}, rather than being multiplied by it.

2.3.1.1. Angle and Direction Equations

A VOS velocity obstacle is formed for each filled element from Cartesian occupancy space using the observed obstacle location and estimated velocity, which are measured as previously described. By using the information from the individual elements, the certainty that each element is occupied can be preserved and used to find the likelihood that a specific robot velocity will lead to a collision with the occupant of that element. Using the VOS method, an element (robot velocity) in velocity space is assumed to be part of the velocity obstacle if it fulfills the following criteria. First, the velocity represented by the element must cause the robot to move with a negative speed relative to the obstacle, as defined by:

$$D_R = \begin{cases} 1\ if \left(\dfrac{\vec{\lambda}_{i,r}(\dot{y}(t_m))}{\dot{y}_r(t_m) - \dot{y}_i(t_m) \cdot (1 \pm V_U)} \wedge \dfrac{\vec{\lambda}_{i,r}(\dot{x}(t_m))}{\dot{x}_r(t_m) - \dot{x}_i(t_m) \cdot (1 \pm V_U)} \right) \geq 0 \\ 0 \quad otherwise \end{cases} \tag{2.8}$$

where $\vec{\lambda}_{i,r}(\dot{y}(t_m))$ and $\vec{\lambda}_{i,r}(\dot{x}(t_m))$ are the relative displacement vectors between the robot and the obstacle element i (see Figure 2.9), $\dot{y}_r(t_m)$ and $\dot{x}_r(t_m)$ are the *x*- and *y*-velocities that the velocity space element represents, and $\dot{x}_i(t_m)$ and $\dot{y}_i(t_m)$ are the obstacle element velocity. In other words, the direction criteria for a velocity element is fulfilled if this velocity will cause the robot and the obstacle to approach each other. If a value of zero is found in Eq. (2.8), then the velocity is not part of a velocity obstacle and will therefore have no repulsive weight. The velocity uncertainty, V_U, represents the uncertainty of the estimate of the obstacle's velocity and is found from the equation

$$V_U = min\left(|(\dot{x}_i(t_m), \dot{y}_i(t_m)) - (\dot{x}_i(t_{m-1}), \dot{y}_i(t_{m-1}))|, max\left|(\dot{x}_r(t_m), \dot{y}_r(t_m))\right|\right) \tag{2.9}$$

This uncertainty factor, V_U is used to increase the range of obstacle velocities that are avoided. If the currently measured velocity is the same as what was measured on the previous time step, then only the estimated obstacle velocity is avoided. However, as the prediction and the observation differ, the range of obstacle velocities that are assumed to be hazardous also proportionally increases. The upper bound on this term is the robot's maximum velocity, as the robot cannot be guaranteed of avoiding an obstacle moving at a higher velocity than that which the robot is capable.

The $\pm$ symbol is used throughout this thesis to represent 'within range'. For example, in Eq. 2.8, if there exists any values between $(1 - V_U)$ and $(1 + V_U)$ that will cause $\frac{\vec{\lambda}_{i,r}(\dot{y}(t_m))}{\dot{y}_r(t_m) - \dot{y}_i(t_m) \cdot (1 \pm V_U)}$ and $\frac{\vec{\lambda}_{i,r}(\dot{x}(t_m))}{\dot{x}_r(t_m) - \dot{x}_i(t_m) \cdot (1 \pm V_U)}$ to both be greater or equal to zero then

D_R will equal one. The values of $(1 \pm V_U)$ in the two components do not need to be the same.

The second criterion is that the element's velocity must move the robot into a collision course with the obstacle as defined by the relative angles between the robot's and obstacle's positions and velocities. The equation

$$A_R = \begin{cases} 1 & tan^{-1}\left(\frac{\vec{\lambda}_{i,r}(\dot{y}_i(t_m))}{\vec{\lambda}_{i,r}(\dot{x}_i(t_m))}\right) = tan^{-1}\left(\frac{|\dot{y}_r(t_m) - \dot{y}_i(t_m) \cdot (1 \pm V_U)| \pm P_A}{|\dot{x}_r(t_m) - \dot{x}_i(t_m) \cdot (1 \pm V_U)| \pm P_A}\right) \cdot (1 \pm (W_{AR} - 1)) \\ 0 & otherwise \end{cases} \quad (2.10)$$

defines the angle of collision where V_U has the same role as before, only now it increases the range of angles instead of the radial direction, based on the predictability of the obstacle's velocity. In other words, if the angle of the relative velocity vector between the robot and the obstacle, $tan^{-1}\left(\frac{|\dot{y}_r(t_m) - \dot{y}_i(t_m))|}{|\dot{x}_r(t_m) - \dot{x}_i(t_m))|}\right)$, is equivalent to the angle of the relative position between the robot and the obstacle, $tan^{-1}\left(\frac{\vec{\lambda}_{i,r}(\dot{y}_i(t_m))}{\vec{\lambda}_{i,r}(\dot{x}_i(t_m))}\right)$, then the element's velocity fulfills the repulsive angle criteria.

The weighted angular term, W_{AR}, allows a more or less conservative range of velocity angles that are assumed to lead to a collision to be defined based on the situation (a W_{AR} value of one will not affect the range). Finally, the angular proximity, P_A, is defined by

$$P_A = min\left[\left(\frac{sr(t_m) - \|(x_i(t_m), y_i(t_m)) - (x_r(t_m), y_r(t_m))\|}{sr(t_m)}\right)^2, 1\right] \quad (2.11)$$

where the term $sr(t_m)$ is the robot's sensor range at the current time step. Using this equation in conjunction with Eq. (2.10) a larger range of angles are considered occupied when an obstacle is close to the robot than when the obstacle is some distance away. This helps the robot to account for additional obstacles that might be hidden behind a closer, occluding obstacle. If one of these undetected obstacles were to unexpectedly move out from behind an occluding obstacle that was very close to the robot a collision would very likely result. However, if the occluding obstacle was farther away from the robot, then the robot would have time to detect the newly revealed obstacle and respond appropriately, so avoiding larger range of angles is not necessary.

2.3.1.2. Time To Collision and Cartesian Distance

The terms time to collision, *TTC*, and the Cartesian distance, *CD,* are variables which measure the physical relationship between the robot and an obstacle. If a velocity does not meet the angle and direction requirements, as described above, then the *TTC* and *CD* are not calculated, as the overall repulsive weight, *R*, is already set to zero, Eq.(2.7). Therefore, every velocity value for which *TTC* and *CD* are computed is assumed to lead to a collision. This limitation on the set of robot velocities examined greatly reduces the complexity of these equations. The weighting term, W_{TTC}, is used as a ratio between the two variables. It represents how important it is to avoid velocities that will lead to a collision as opposed to velocities that lead to an obstacle that is close to the robot. These priorities can be radically different for an obstacle that is close to the robot, but moving in the opposite direction.

As commonly defined, the time to collision term, TTC measures the amount of time that it will take the robot to collide with an obstacle for each element's velocity value, $\left(\dot{x}_r(t_m), \dot{y}_r(t_m)\right)$, assuming that both the robot and the obstacle maintain a constant velocity. For this derivation, the equation for the *TTC* is

$$TTC = \begin{cases} \dfrac{\left\|\vec{\lambda}_{i,r}\left(\dot{x}(t_m), \dot{y}(t_m)\right)\right\|}{\left\|\left(\dot{x}_r(t_m), \dot{y}_r(t_m)\right) - \left(\dot{x}_i(t_m), \dot{y}_i(t_m)\right)\right\|} & \left\|\left(\dot{x}_r(t_m), \dot{y}_r(t_m)\right) - \left(\dot{x}_i(t_m), \dot{y}_i(t_m)\right)\right\| < \left\|\vec{\lambda}_{i,r}\right\| \\ \dfrac{\left\|\vec{\lambda}_{i,r}\left(\dot{x}(t_m), \dot{y}(t_m)\right)\right\|}{sr(t_m)} & \left\|\left(\dot{x}_r(t_m), \dot{y}_r(t_m)\right) - \left(\dot{x}_i(t_m), \dot{y}_i(t_m)\right)\right\| \geq \left\|\vec{\lambda}_{i,r}\right\| \end{cases} \quad (2.12)$$

If the magnitude of the difference between the robot velocity and the obstacle's velocity is less than $\vec{\lambda}_{i,r}\left(\dot{x}(t_m), \dot{y}(t_m)\right)$ then the *TTC* value is calculated. In other words, if the difference between the robot and obstacle's velocities is small enough that there will not be a collision within the next time step then the *TTC* value is used. If the magnitude of the difference between the robot's velocity and the obstacle's velocity is greater than $\vec{\lambda}_{i,r}\left(\dot{x}(t_m), \dot{y}(t_m)\right)$, then that robot velocity will cause a collision in less than one time step (i.e. before the robot has a chance to respond) so these velocities are given the smallest possible value (which makes Eq. (2.7) highly repulsive). In this case the sensors range ($sr(t_m)$) is used as the denominator as this is the greatest possible distance between the robot and an obstacle.

The Cartesian distance,

$$CD = \left(\left(x_i(t_m), y_i(t_m)\right) - \left(x_r(t_m) + \frac{\dot{x}_r(t_m)}{\Delta t_m}, y_r(t_m) + \frac{\dot{y}_r(t_m)}{\Delta t_m}\right)\right)^2 \quad (2.13)$$

is a measure of how far away the obstacle will be from the robot at the end of the time step. Obstacles will be close by, and therefore present a more imminent threat, are given higher repulsive weightings than obstacles which will be farther away.

2.3.2. Attractive Weights

As previously mentioned, only velocity space elements that are part of a velocity obstacle--and will therefore lead to a collision between the robot and the obstacle--are given any repulsive weighting. All other elements are assumed to represent safe robot velocities. However, all elements in velocity occupancy space are given distinct attractive weightings in order to prevent large portions of VOS from being equally weighted when the elements do not represent equally advantageous velocities. The attractive value for each VOS element is found from the equation

$$A = [W_{VD} \cdot VD + VC + W_A \cdot A_A] \tag{2.14}$$

where the weights, W_{VD} and W_A, are defined based on the robot's objectives. The velocity difference term, *VD,* is found from the equation

$$VD = -\left[\frac{\left\|\left(\dot{x}_r(t_m), \dot{y}_r(t_m)\right) - \beta\right\|}{2 \cdot \left\|\max\left(\dot{x}_r(t_m), \dot{y}_r(t_m)\right) - \min\left(\dot{x}_r(t_m), \dot{y}_r(t_m)\right)\right\|} - 1\right]^2 \tag{2.15}$$

where the β is defined as

$$\beta = \begin{cases} \max\left(\dot{x}_r(t_m), \dot{y}_r(t_m)\right) & if \quad \max\left(\dot{x}_r(t_m), \dot{y}_r(t_m)\right) < \left(\vec{\lambda}_{G,r} + \left(\dot{x}_G(t_m), \dot{y}_G(t_m)\right)\right) \\ \vec{\lambda}_{G,r} + \left(\dot{x}_G(t_m), \dot{y}_G(t_m)\right) & if \quad \max\left(\dot{x}_r(t_m), \dot{y}_r(t_m)\right) \geq \left(\vec{\lambda}_{G,r} + (\dot{x}_G, \dot{y}_G)\right) \geq \min\left(\dot{x}_r(t_m), \dot{y}_r(t_m)\right) \\ \min\left(\dot{x}_r(t_m), \dot{y}_r(t_m)\right) & if \quad \left(\vec{\lambda}_{G,r} + \left(\dot{x}_G(t_m), \dot{y}_G(t_m)\right)\right) < \min\left(\dot{x}_r(t_m), \dot{y}_r(t_m)\right) \end{cases} \tag{2.16}$$

where $\left(\vec{\lambda}_{G,r} + \dot{x}_G(t_m), \dot{y}_G(t_m)\right)$ is the location of the goal in velocity space, and $\max\left(\dot{x}_r(t_m), \dot{y}_r(t_m)\right)$ and $\min\left(\dot{x}_r(t_m), \dot{y}_r(t_m)\right)$ are the maximum and minimum velocities that the robot can reach during this time step. The β term is used in the velocity difference equation so that a consistent weighting can be maintained no matter how far the goal is from the robot. In other words, it is desirable that the velocity which will lead most quickly to the goal will always have the same attractive weighting no matter how far the goal is from the robot.

The next term, velocity change, *VC,* is given by the equation

$$VC = \frac{\left\| \left(\dot{x}_r(t_{m-1}), \dot{y}_r(t_{m-1})\right) - \left(\dot{x}_r(t_m), \dot{y}_r(t_m)\right) \right\|}{\max\left(\dot{x}_r(t_m), \dot{y}_r(t_m)\right) - \min\left(\dot{x}_r(t_m), \dot{y}_r(t_m)\right)} - 1 \qquad (2.17)$$

The purpose of this term is to discourage frequent accelerations and decelerations so it gives velocities closer to the robot's current velocity a more attractive weight than velocities which require more acceleration to reach.

Finally the cosine of the angle between the goal's location in velocity space and the velocity element in question, is found from

$$\alpha = \tan^{-1}\left(\frac{\vec{\lambda}_{G,r}\left(\dot{y}(t_m)\right) + \dot{y}_G(t_m)}{\vec{\lambda}_{G,r}\left(\dot{x}(t_m)\right) + \dot{x}_G(t_m)}\right) - \tan^{-1}\left(\frac{\dot{y}_r(t_m)}{\dot{x}_r(t_m)}\right) \qquad (2.18)$$

and the attractive angle term, A_A, is set as the negative of that angle or zero, if the angle is large enough that the velocity would no longer be leading in the direction of the goal

$$A_A = \begin{cases} -cos(\alpha) & if\ |\alpha| \leq \dfrac{\pi}{2} \\ 0 & if\ |\alpha| > \dfrac{\pi}{2} \end{cases} \tag{2.19}$$

Similar to the β term in Eq. (2.15), the contents of the first inverse tangent term in Eq. (2.18) are replaced with the maximum or minimum reachable robot velocity ($\max\left(\dot{x}_r(t_m), \dot{y}_r(t_m)\right)$ or $\min\left(\dot{x}_r(t_m), \dot{y}_r(t_m)\right)$) if the goal's location in velocity space is outside of these bounds. This prevents the preferred angle from decreasing too much to encourage circumnavigation of an obstacle if the robot is far from the goal.

The equations shown in this section form the basis of velocity occupancy space. Other logic was included to avoid numerical contingencies, such as division by zero, in the actual program, but is omitted here for brevity.

2.3.3. Velocity Selection and Navigation

In simulations, negative values are used to represent how attractive an element of VOS is, *A* from Eq. (2.14), and positive values to represent how repulsive the element is, *R* from Eq. (2.7). This allows the attractive and repulsive values of a single element to be summed so that the final value of a single element in velocity occupancy space can be influenced by multiple factors. The value of each element indicates the desirability of that robot velocity and the element with the lowest value (i.e. the most desirable velocity) can be found by minimization with a simple gradient search.

The weights in Eqs. (2.7 and 2.14) can be adjusted both to influence how the various terms should rank relative to each other, as well as to govern the interplay between the attractive and repulsive values. These weights are pre-calculated by the process outlined in Section 2.4.

After the robot velocity has been chosen for a motor time step, the process described in the previous two sections is repeated so as to allow the robot to continuously adjust its velocity to deal with non-constant obstacle velocities and newly detected obstacles. The processes described in Sections 2.2 (building the occupancy space grid and estimating obstacle locations and velocities) and 2.3 (populating velocity occupancy space and selecting the next velocity) are performed in parallel.

2.4. Optimization of Weights

The value of each element in the velocity occupancy space is defined by the sum of Eqs. (2.7 and 2.14):

$$\begin{aligned} VOS\big(\dot{x}_r(t_m), \dot{y}_r(t_m)\big) &= R + A = \\ W_R\left[D_R \cdot A_R(W_{AR}) \cdot \left(\frac{W_{TCC}}{TTC} + \frac{1}{CD}\right) \cdot E_{Oc}\right] &+ [W_{VD} \cdot VD + VC + W_A \cdot A_A] \end{aligned} \tag{2.20}$$

Along with defining the value of the individual terms based on the physical properties of the system and environment, the individual weights (W terms) must be determined in order to effectively prioritize the various aspects of obstacle avoidance and goal finding. Initially, these weights were hand-tuned based on empirical knowledge and observation of the system [4]. In this section a combination of an exhaustive search and an optimization process, which produced significantly better weights, is described.

2.4.1. Evaluation Criteria

Four evaluation metrics were used in order to judge the quality of the path that the robot followed given each set of weights. During each time step the position of the robot, the robot's velocity and the relative position with respect to the robot of each obstacle to the robot was recorded. The magnitude of the robot's change in position (the distance that it traveled during the simulation), change in velocity and the square of the inverse of the closest obstacle's proximity to the robot were each summed for every time step and used for the first three evaluation metrics: distance traveled, acceleration and obstacle proximity. In addition, the number of time steps required for the robot to reach the goal and two binary values that indicated if a collision occurred during the scenario and if the robot was successful at reaching the goal were also used as evaluation metrics.

The number of collisions and the number of times that the robot successfully reaches that goal are the most important measures of the algorithm's performance, however, the other evaluation metrics, shown in Table 2.1, were also recorded in order to compare the quality of the paths that the robot chooses using each set of weights.

Table. 2.1 Evaluation Metrics

N = number of iterations, T_n = number of time steps in iteration n, J = number of obstacles, C_r = robot position, C_o = obstacle position, V_r = robot velocity

Evaluation Metrics	Equations
Obstacle Proximity $\left(\frac{1}{m^2}\right)$	$\frac{1}{N}\cdot\sum_{n=1}^{N}\sum_{\tau=1}^{T_n}\frac{1}{\sum_j^J\left\lvert C_r(\tau)-C_{Oj}(\tau)\right\rvert^2}$
Change in Velocity $\left(\frac{m}{s}\right)$	$\frac{1}{N}\cdot\sum_{n=1}^{N}\sum_{\tau=1}^{T_n}\lvert V_r(\tau)-V_r(\tau-1)\rvert$
Distance Traveled (m)	$\frac{1}{N}\cdot\sum_{n=1}^{N}\sum_{\tau=1}^{T_n}\lvert C_r(\tau)-C_r(\tau-1)\rvert$
Time (s)	$\frac{1}{N}\cdot\sum_{n=1}^{N}T_n$

A random scenario generator was used in order to produce a broad range of situations in which the algorithm's ability to successfully guide a robot could be tested. The scenario generator produced a number of obstacles (between one and eight) with a range of velocities and starting positions, as well as different positions for the goal. Impossible scenarios (e.g. if the scenario started with a collision) were removed. An example of the initial conditions of one of these scenarios is shown in Figure 2.10. While Figure 2.10 (and later figures) shows an overhead view of the robot and obstacles, for all of the simulations the robot only had access to the simulated noisy LRF data that would have been produced from the environment. The LRF data was simulated to have a 20%

chance of producing a radial error of ±0.1m. Using the simulated LRF the robot was only able to 'see' the nearest edges of obstacles that were within the range of the LRF (set at 20m for the simulations).

It should be noted that while some error was considered when simulating the LRF data there are other difficulties inherent to laser range finders that were not accounted for in this simulation, such as specular reflections and obstacles with variable cross-sections (such as the legs verse torso on a human). However, these difficulties have been addressed by other researchers, such as [87] and [88].

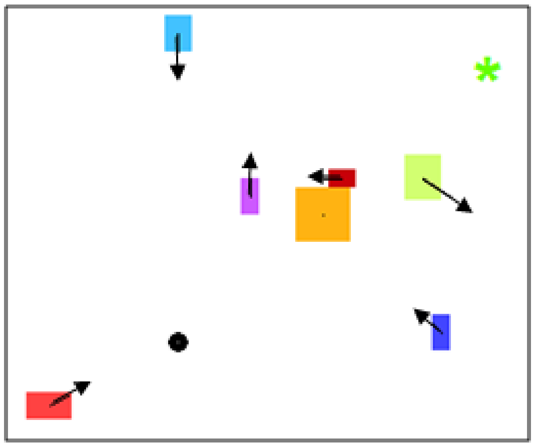


Figure 2.10 Initial conditions of a sample scenario. Figure contains the robot (circle), obstacles (rectangles with velocity vectors) and the goal (asterisk)

The performance measures for ten different scenarios were used for the optimization process so that the results of the optimization would be appropriate for a more general environment, instead of being overly specific for a single scenario. In addition, the same set of ten scenarios was used throughout the optimization processes so that the results could be accurately compared.

2.4.2. Optimization

A coarse exhaustive search was performed in order to find a selection of better initial design variables that could avoid some of the less desirable local minima. The coarse exhaustive search showed that there were a significant number of non-optimal local minima and also some discontinuities within the design variable search space. However, using some of the better sets of variables from the exhaustive search as the initial set of design variables, the optimization process produced improved results. For validation, the results of the optimization process were tested on a set of one thousand scenarios, from the random obstacle scenario generator, in order to verify that the VOS algorithm would operate acceptably for almost any scenario.

Optimization was performed using MATLAB's `fgoalattain` function. This function uses sequential quadratic programming to reduce a set of nonlinear functions to below a given goal level. It was used for this research in the following manner:

$$\underbrace{minimax}_{x,\gamma}\ \gamma\ \ such\ that \begin{cases} \boldsymbol{F}(\boldsymbol{x}) - weight \cdot \gamma \leq \boldsymbol{F_g}(\boldsymbol{x}) \\ \boldsymbol{h}(\boldsymbol{x}) = 0 \end{cases} \tag{2.21}$$

where the design variables, $\boldsymbol{x}$, were the weights: $\{W_R, W_{TTC}, W_{AR}, W_{VD}, W_A\}$. The equality constraints, $\boldsymbol{h}(\boldsymbol{x})$, were set so that the number of collisions and the number of times that the robot was unsuccessful at reaching the goal had to equal zero. The hand tuned weights and the optimized weights are compared in Table 2.2.

Table 2.2. Coefficients / design variable values used for velocity element weighting

		Hand-tuned Weights	Optimized Weights
Repulsive Weights	Repulsive Weight, W_R	1.0	0.4
	Time to Collision, W_{TTC}	3.5	7.0
	Angular Range, W_{AR}	1.0	1.0
Attractive Weights	Velocity Distance, W_{VD}	2.7	3.2
	Angle, W_A	0.3	2.2

For the optimization, each of the design variables was constrained to be positive and less than ten (ten was chosen as a reasonable upper bound based on previous experience with hand-tuning). The evaluation metrics of obstacle proximity and time were given twice the weight of the other as they are more indicative of a successfully completed simulation than acceleration and distance traveled. The optimization was run for either six hundred iterations or until the function value ($\boldsymbol{F}(\boldsymbol{x})$ in Eq. (2.18)) varied by less than 10^{-20}.

The optimization process did not neatly converge to a global minimum that was the optimal set of weights for any situation. The inability of the optimization to find a global optimum is probably the result of two aspects of the system. First, this is a five-dimensional design problem; the function is non-convex and has some discontinuities. As such, finding the global minimum, even for a set of ten scenarios, is an extremely challenging and time consuming optimization process. Second, some of the solutions obtained using optimization appear to have been overly designed for and narrowly focused on for the ten design scenarios that were used for the optimization process, as they failed to cause the VOS algorithm to work acceptably for a broader range of scenarios. Ideally, a much larger set of scenarios should be used for both the initial exhaustive search and the later optimization, however, even using just ten scenarios made for an extremely time consuming processes (weeks of dedicated CPU time on a laptop computer), so optimizing with a more complete set of scenarios would not be practical without access to parallel computing resources.

If such resources were available, then the optimization process could, in theory, be applied to not just the weight factors but to the exponential relationship between the various terms in Eq. (2.20). For instance, through optimization it might be revealed that a cubing the time to collision term leads to better robot performance than just increasing its weight.

2.5. Results

2.5.1. Initial VOS Results

An example of the VOS algorithm run with a fairly simple scenario is shown in Figures 2.11 – 2.14. In Figure 2.11, the robot is avoiding two moving and two stationary obstacles. The simulation covers the first six motor time steps. The robot is initially stationary so that it can locate surrounding obstacles and make an initial estimate of their velocities before selecting its first velocity.

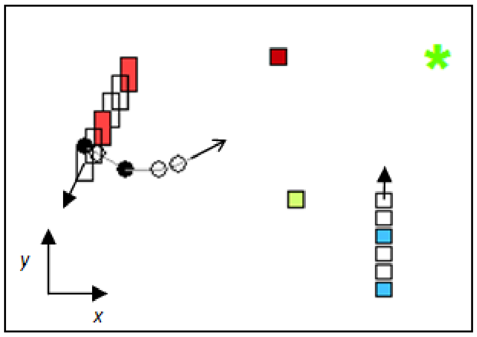


Figure 2.11 First six simulation steps of an example scenario

The velocity occupancy space representation of the obstacles in Figure 2.11 is shown in Figure 2.12 where the velocity values (on the x- and y-axes) that will lead to a collision can be seen as the cones of repulsive values (positive values on the z-axis) in velocity occupancy space.

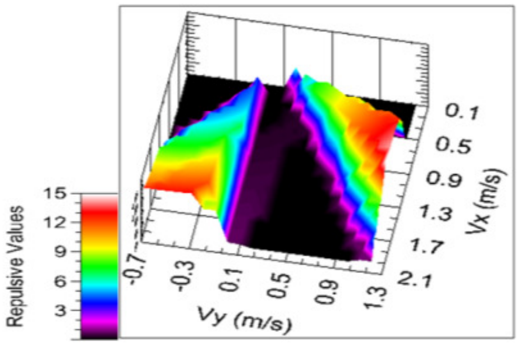


Figure 2.12. VOS populated with repulsive values

All of the velocity occupancy space elements are given attractive values (negative values on the z-axis) in proportion to how effectively each velocity will lead the robot to the goal. In Figure 2.13, velocity occupancy space is shown populated with attractive values based on the scenario in Figure 2.11.

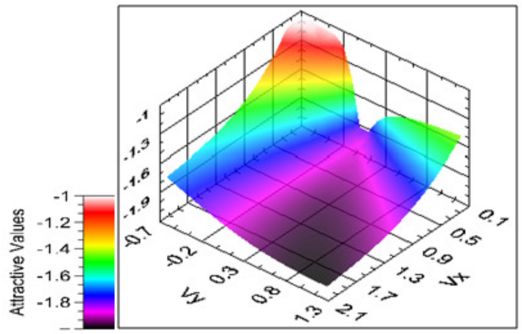


Figure 2.13. VOS populated with attractive values

After fifteen time steps the robot has successfully reached the goal while avoiding all obstacles, as shown in Figure 2.14.

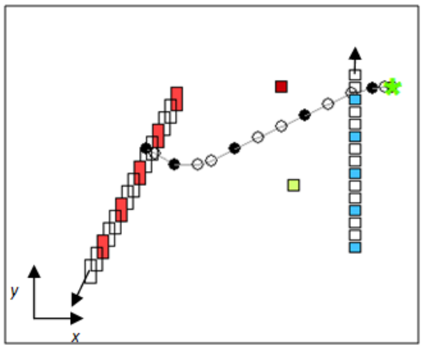


Figure 2.14. Simulation results after fifteen time steps

2.5.2. Performance with Optimization

The set of weights that resulted from the optimization process are shown in Table 2.2, and results from the algorithm run with these weights are shown in Figures 2.15 and 2.16.

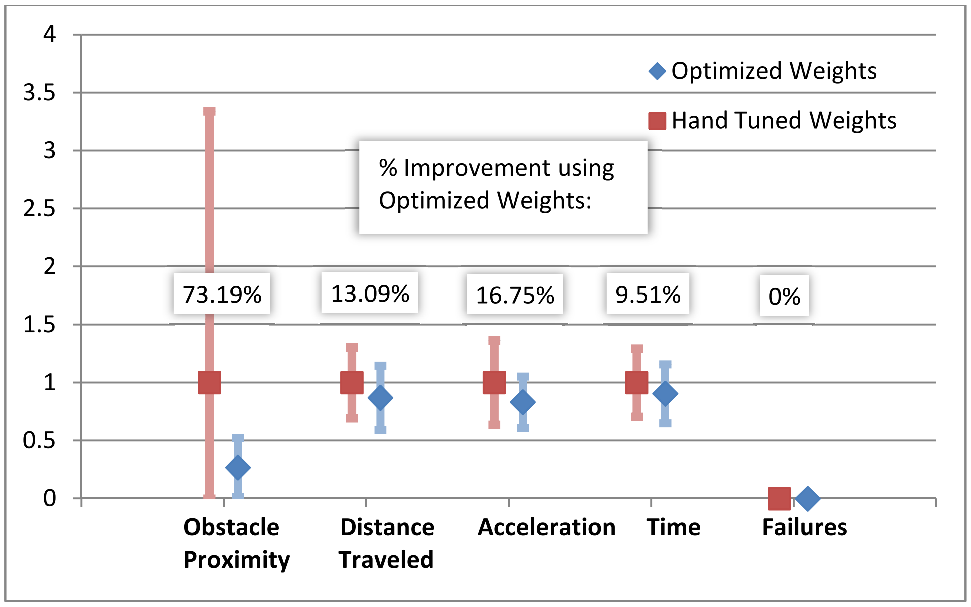


Figure 2.15. Comparison of Normalized Evaluation Metrics between Hand Tuned and Optimized Weights for 10 Design Scenarios (one sigma error bars)

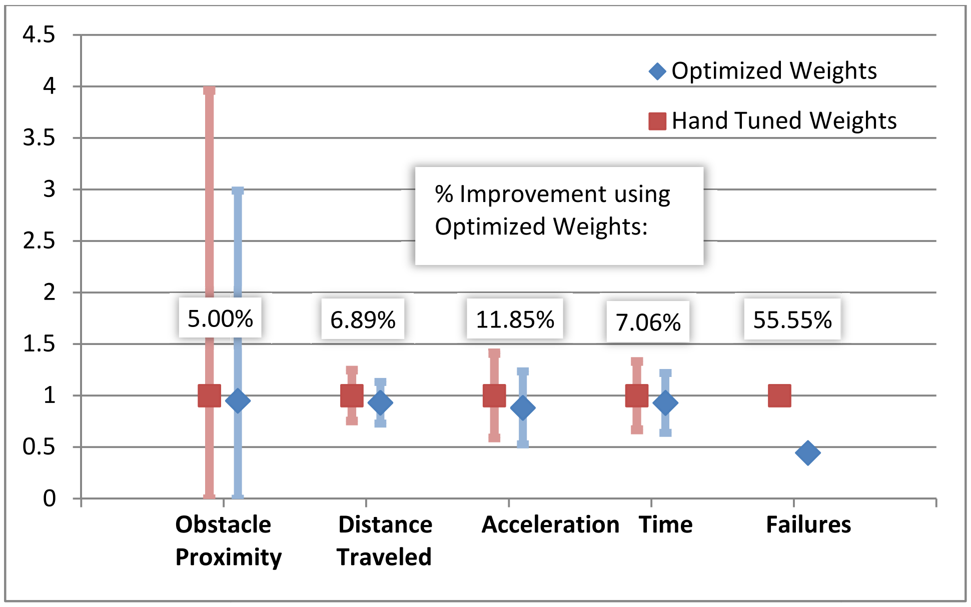


Figure 2.16. Comparison of Normalized Evaluation Metrics between Hand Tuned and Optimized Weights for 1000 Scenarios (one sigma error bars)

In Figure 2.15, the performance of the algorithm using the original hand-tuned weights versus the performance with the optimized weights is shown for the set of ten design scenarios which were used to perform the optimization. In Figure 2.16, the performance of the algorithm with the different sets of weights is again compared, but this time for the one thousand randomly generated scenarios, which were used to validate the results of the optimization. The values of the evaluation metrics were normalized independently for the two sets of scenarios against the values found using the hand tuned weights; the lower the value of the evaluation metrics, the better the performance of the algorithm. Neither set of weights caused a failure (either due to a robot collision or the inability of the robot to reach the goal within a specified amount of time) of the simulation for the set of ten design scenarios. For the one thousand validation scenarios, there were nine failures (0.9%) for the hand-tuned weights and four failures (0.4%) for the optimized weights. The simulations in which failures did occur where usually situations that even a human driver would have had difficulty successfully navigating. For example, one of the failures using the optimized weights occurred when a couple of obstacles converged almost immediately on the robot, see Figure 2.17. An omniscient agent would have been able to find a successful path; however the algorithm had very little time to collect velocity data on the surrounding obstacles and there were a very limited number of velocity choices that would have allowed the robot to successfully avoid all of the obstacles.

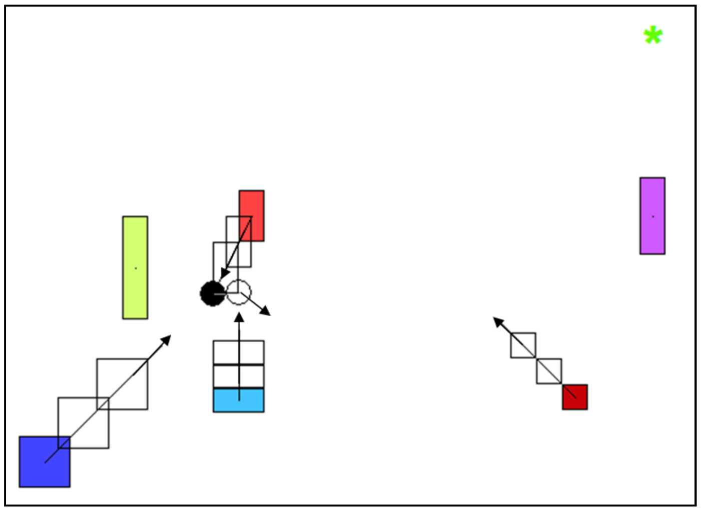


Figure 2.17 Example of a failed scenario. Collision occurs after three motion time steps.

The optimization process significantly improved the algorithm's performance for the design scenarios that were used in the optimization. While the overall improvement for the validation scenarios was not as large, it was still statistically significantly for three of the evaluation metrics: the distance traveled, acceleration, time ($p < 0.005$, on a two-tailed, paired *t*-test) and for the number of simulation failures ($p < 0.025$).

While the obstacle proximity evaluation metric did not see a significant improvement using the optimized weights, the true improvement may be covered up by the improvement in the collision failure rate. The values of the evaluation metrics for a simulation in which a failure occurred where not included in the statistics. Therefore, improvements in the performance of the algorithm with the optimized weights which allowed the robot to avoid a collision (presumably by decreasing the robot's proximity to the obstacles) while the hand tuned weights led to a collision would not influence the final value of the obstacle proximity for either set of weights. In other words, the simulations where the obstacle proximity metric would have been the worst (due to a collision) for the hand tuned weights, were removed from the analysis, which may have improved the value of the obstacle proximity for the hand tuned weights when compared to the optimized weights. To make the comparison more fair, if there are a significantly different number of failures (collisions and time outs) than this should be considered of more importance than the difference in the evaluation metrics. The evaluation metrics become more meaningful, between two tests, as the number of failures between the two tests becomes closer.

2.5.3. Results for Obstacles with Variable Velocities

The VOS algorithm works most effectively when the obstacles maintain a constant velocity and the optimization was performed using constant velocity obstacles. However, the algorithm was also tested using obstacles with variable velocities. In this situation, every sensor time step the obstacles that had a 20% chance of altering their x- or y- velocity by a value (randomly generated) in the range of $\left[-0.5\frac{m}{s}, 0.5\frac{m}{s}\right]$. The obstacles' velocities were still bound to be within the robot's velocity range of $\left[-2.0\frac{m}{s}, 2.0\frac{m}{s}\right]$.

Using the optimized weights and the one thousand validation scenarios (with random obstacle velocity changes), eight simulation failures where recorded (0.8% failure rate). There was also no statistically significant difference between three of the four evaluation metrics (obstacle proximity, distance and time) between the scenarios where the obstacles all had constant velocities and the scenarios where the velocities randomly changed.

Figure 2.18 shows a representative scenario where the algorithm allows a robot to navigate around stationary and moving obstacles and reach a goal. Figures 2.19 through 2.21 show sequential segments of the robot navigation in this scenario. In Figure 2.19, the robot starts to accelerate in the positive x- and y-directions to avoid *Obstacles A* and *B*. In Figure 2.20, the robot speeds up to circle around *Obstacle C* and starts moving in the positive y-direction to avoid *Obstacle D*. Finally, Figure 2.21, the robot outpaces *Obstacle A* and then continues on its course to avoid *Obstacle E* and reach the goal.

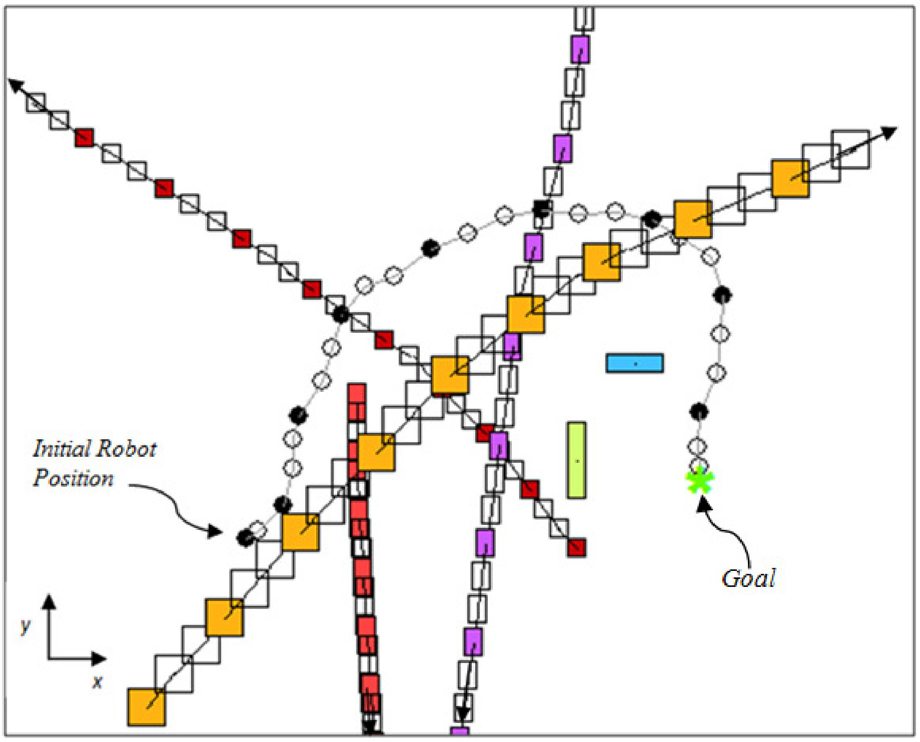
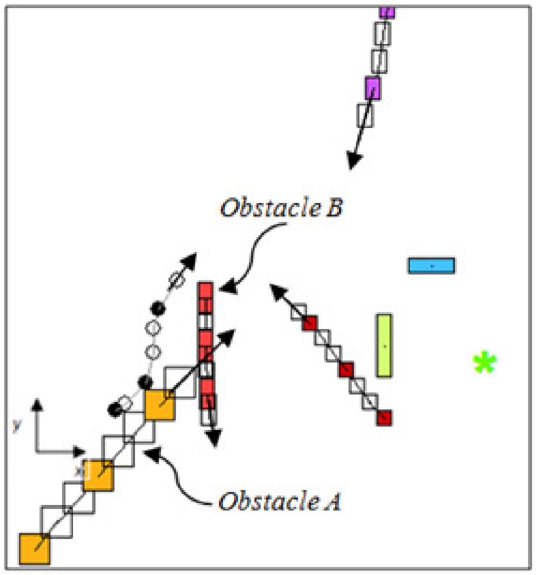


Figure 2.18 Simulation results with four moving and two stationary obstacles for twenty-seven time steps



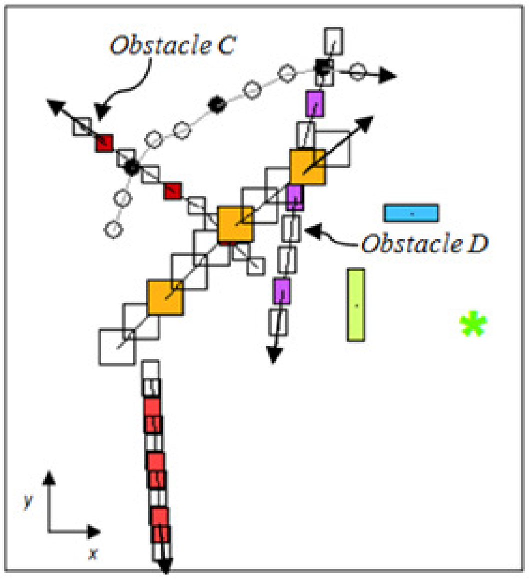


Figure 2.19 First eight motor time steps **Figure 2.20** Motor time steps eight to seventeen

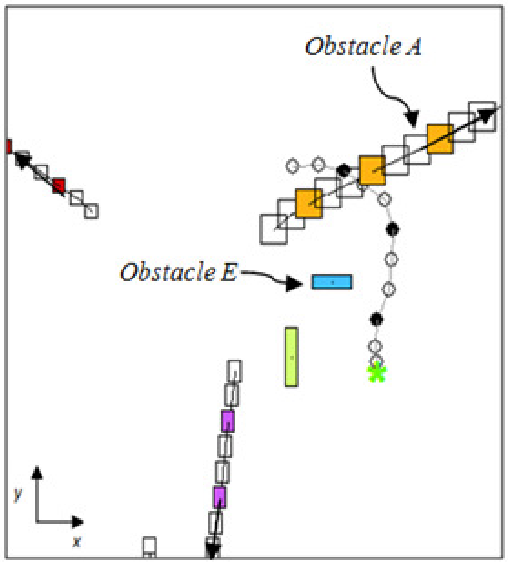


Figure 2.21 Motor time steps seventeen to twenty-seven

2.5.4. Comparison of VOS to other Obstacle Avoidance Algorithms

In order to validate the performance of VOS, it will be compared via simulation to the VFH method [35] and the original Velocity Obstacle (VO) concept [57, 58] by testing all three algorithms against a range of obstacle/goal scenarios. These two methods were chosen for comparison as they were the initial inspiration for VOS. VOS uses the same method to build configuration occupancy space that is fundamental to VFH and utilizes the concept of velocity obstacles to avoid moving obstacles.

VOS is not being compared in detail to the other algorithms on the table as most of these algorithms either require additional or different environmental information from VOS (i.e. Probabilistic RRTs require training samples, State-Time Space requires complete environmental knowledge and) or produce different types of paths (i.e. State-Time Space can find an optimal path and Probabilistic RRTs plan a significantly longer path than the reactive VOS algorithm).

2.5.4.1. Comparison of VOS and VFH

Background on the VF

The VFH [30] method of obstacle avoidance is a well tested method of static obstacle avoidance. For the comparison between VOS and VFH, the VFH+ method [35], an extension of the basic Vector Field Histogram will be used. This extension decreases the need to hand tune specific parameters of the algorithm and also, according to the authors, produces more reliable results.

The VFH converts ranging sensor data into a Cartesian gird using the same method as VOS (detailed in Section 2.1), and then uses this grid to build a histogram of all of the angles of navigation around the robot. The equation,

$$\beta_{i,j} = arctan\left(\frac{y_j - y_o}{x_i - x_o}\right) \tag{2.23}$$

is used to find the angle that each filled element in the Cartesian grid is from the robot, where $(x_{i,}\ y_j)$ are the coordinates of the filled element and $(x_{o,}\ y_o)$ are the coordinates of the robot. The magnitude of each element is based both on the of the certainty of element being occupied, $c_{i,j}$, as well as the distance between the element and the robot, $d_{i,j}$ and is found from the equation,

$$m_{i,j} = {c_{i,j}}^2 \cdot (1 + sr - d_{i,j})^2, \tag{2.24}$$

where *sr* is the robot's sensor range. The equation was formulated to fulfill the requirements given in [35] and tuned in order to allow the VFH algorithm to operate as effectively as possible for the comparison simulations.

The robot is treated as a point so the angle that each obstacle fills must be increased by the radius of the robot, *r,* as well as the minimum distance that the robot should maintain between itself and obstacles, $\mathrm{d_{min}}$. Therefore each angle is increased by

$$\gamma_{i,j} = arcsin\left(\frac{r + \mathrm{d_{min}}}{d_{i,j}}\right). \tag{2.25}$$

A histogram can then be computed for each robot navigation angle using the equation

$$H_\theta = \sum_{i,j\epsilon C} m_{i,j} \cdot h_{i,j} \tag{2.26}$$

where

$$h_{i,j} = \begin{cases} 1 & if\ \theta \in [\beta_{i,j} - \gamma_{i,j}, \beta_{i,j} + \gamma_{i,j}] \\ 0 & if\ \theta \notin [\beta_{i,j} - \gamma_{i,j}, \beta_{i,j} + \gamma_{i,j}] \end{cases}. \tag{2.27}$$

The histogram can be converted into binary form by apply a two stage threshold, where angles are given a value of one, or filled, if they have an H_θ value above a certain threshold, τ_{high}, and angles are given a value of zero, or open, if they have an $H_\theta$below τ_{low}. If the H_θ value is between the two thresholds, then the angle remains at its value from the previous time step. This double threshold prevents angles from frequently oscillating between filled and empty.

Each open navigation angle is given a cost based on its difference from the goal angle, θ_g, as well as the difference from the robot's current heading, θ_R, using the equation,

$$g(\theta) = \mu_1 \cdot (|\theta - \theta_g|) + \mu_2 \cdot (|\theta - \theta_R|) \tag{2.28}$$

Based on the guidelines detailed in [35], the terms of the cost function were weighted as $\mu_1 = 5$ and $\mu_2 = 4$; this produced a goal seeking algorithm that tends to avoid oscillation and follow a steady path to the goal.

Simulation Comparison of VOS and VFH

The performance of VOS was compared to that VFH over 1000 randomly generated scenarios (identical sets of scenarios were used to test each algorithm). The first 500 scenarios each included ten stationary obstacles and the second scenarios 500 had a mix of ten moving and stationary obstacles. The moving obstacles and the robot had a top speed of $2\frac{m}{s}$. The scenarios were designed to be challenging in order to better differentiate between the two algorithms. Both VOS and VFH build a similar configuration occupancy space map; however the VFH method then selects safe headings that lead towards the goal based on the occupancy space data, while the VOS method builds a velocity occupancy space map and selects safe velocities that will lead to the goal.

In Figures 2.22 and 2.23, the VOS and VFH algorithms are shown navigating around the same set of ten stationary obstacles. They take different paths, but both manage to reach the goal after 14 time steps. However, the VFH algorithm follows a slightly shorter path.

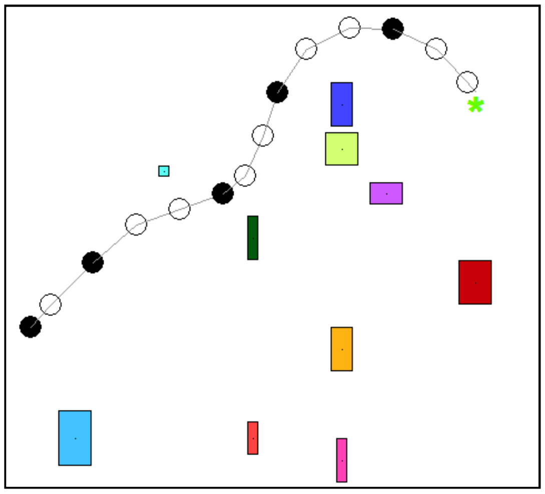


Figure 2.22 Robot navigating using VOS

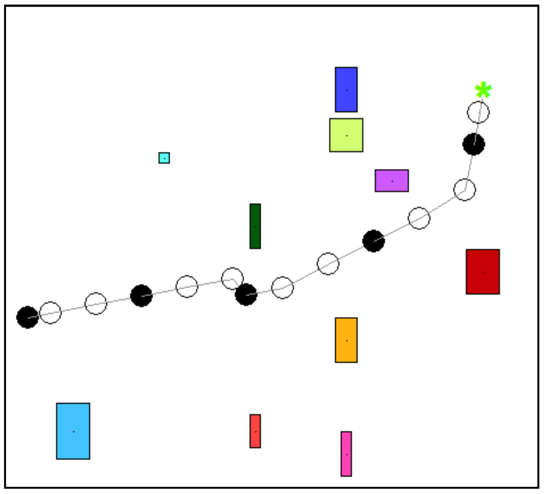


Figure 2.23 Robot navigating using the VFH

As shown in Table 2.3, for the 500 scenarios with only stationary obstacles, the VOS and VFH methods performed very similarly; both experienced a single collision and the VOS method experienced one timeout while the VFH method had none. The timeout occurred when the goal was very close to an obstacle. The VFH method was successful at reaching the goal, but the VOS method, due to the increased safety margin that it keeps around obstacles due to the chance that they might start to move, was not able to reach the goal.

Table 2.3 Comparison of the performance of VOS and VFH on 500 Scenarios with Stationary Obstacles

	Collisions	**Timeouts**	**Obstacle Proximity** $\left(\frac{1}{m^2}\right)$	**Distance Traveled** (m)	**Change in Velocity** $\left(\frac{m}{s}\right)$	**Time** (s)
VOS	1	1	3.22	25.20	5.76	15.53
VFH	1	0	5.59	24.64	3.35	13.94

Simulations where either method experienced a failure (either due to a collision or timeout) were removed from the data set for the calculation of the of the four evaluation metrics (this was done for all algorithm comparisons in this chapter). The VOS method maintained a greater distance from obstacles, but the VFH method was able to (on average) find slightly shorter routes and reach the goal more quickly with fewer changes in velocity. The difference in time and change in velocity is due to the VFH method always attempting to operate at the robot’s highest velocity while the VOS method frequently chooses slower, more cautious velocities.

For the 500 scenarios with both moving and stationary obstacles, the VOS both experienced seven collisions while the VFH method experienced 73 collisions, as shown on Table 2.4. Because the VFH method selects a new heading each time step based on current sensor data, it was able to avoid moving obstacles in the majority of scenarios, however the VFH's inability to detect or respond to moving obstacles still caused it to experience a statistically significant higher number of collisions ($p < 0.001$) than the VOS method.

Table 2.4 Comparison of the performance of VOS and VFH on 500 Scenarios with Stationary and Moving Obstacles

	Collisions	**Timeouts**	**Obstacle Proximity** $\left(\frac{1}{m^2}\right)$	**Distance Traveled** (m)	**Change in Velocity** $\left(\frac{m}{s}\right)$	**Time** (s)
VOS	7	0	3.23	25.18	5.02	15.33
VFH	73	0	4.27	24.69	3.81	13.97

Again, the VOS method maintained a greater distance from obstacles, but the VFH method was able to find slightly shorter routes and reach the goal more quickly with fewer changes in velocity when it did not encounter a collision. However, the large difference in the number of collisions between the two methods outweighs the performance of the algorithms in successful simulations (as it would be irresponsible to choose a navigation method that had an order of magnitude greater chance of colliding with an obstacle in order to decrease the time it would take to reach a goal by a few seconds). Therefore, this data demonstrates that VOS performs comparably to the VFH method in environments with only stationary obstacles and performs superiorly in environments where there are both stationary and moving obstacles.

2.5.4.2. Comparison of VOS and VO

The performance of VOS was also compared to that of the original velocity obstacle (VO) method (described in Section 2.1.2) using 500 randomly generated scenarios with both stationary and moving obstacles. The T_h was tuned, for the comparison simulations, to a value of nine seconds, as this produced the best simulation results for the VO algorithm in the given scenarios. Initially (Table 2.5), both algorithms had access to perfect environmental data (both obstacle position and velocity data) about the 500 scenarios, then both algorithms were tested again with the same 500 scenarios but only given laser range finder position data and obstacle velocities calculated using the center of certainty (COC) method detailed in Section 2.2.1 (Table 2.6). Again, the scenarios were designed to be very challenging in order to better differentiate between the two algorithms.

Table 2.5 Comparison of the performance of VOS and VO on 500 Scenarios with Complete Obstacle Knowledge

	Collisions	**Timeouts**	**Obstacle Proximity** $\left(\frac{1}{m^2}\right)$	**Distance Traveled** (m)	**Change in Velocity** $\left(\frac{m}{s}\right)$	**Time** (s)
VOS	2	0	2.35	25.22	4.77	15.18
VO	4	0	3.86	26.97	6.62	16.82

For the 500 scenarios where the algorithms had complete knowledge of the obstacles, the VOS method experienced two collisions while the VO method experienced four (this is not a statistically significant difference) and neither algorithm experienced a timeout. The VOS method performed somewhat better than the VO method according to the four evaluation metrics; this is probably due to the weights of the VOS method being turned according to these metrics.

Table 2.6 Comparison of the performance of VOS and VO on 500 Scenarios with LRF based Position and COC Velocity Data

	Collisions	Timeouts	Obstacle Proximity $\left(\frac{1}{m^2}\right)$	Distance Traveled (m)	Change in Velocity $\left(\frac{m}{s}\right)$	Time (s)
VOS	4	0	3.08	25.26	4.97	15.26
VO	20	3	3.93	26.19	7.23	15.74

For the 500 scenarios where the algorithms used position data from a simulated laser range finder and obstacle velocity data calculated from the laser range finder data using the COC method, the VOS method experienced four collisions while the VO method experienced twenty collisions (this *was* a statistically significant difference, $p >$ 0.001).

The timeouts using the VO method were due to the Boolean nature of this method. Velocities are labeled as safe (admissible) or unsafe based on a specific threshold (time to collision) and a goal directed velocity is chosen from the set of safe velocities. If no safe velocity exists, then the robot performs emergency breaking in order to avoid a collision. The three timeouts and a number of the collisions occurred when the robot reached a position where it considered none of the velocities to be sufficiently safe and the robot would remain stationary until the simulation ended or a moving obstacle collided with the robot. This shortcoming is also mentioned in [56], where the author notes the difficulty in setting a threshold that maintains the safety of the robot without leading to too many situations where there are no admissible velocities. With the VOS algorithm, even if all possibly velocities are somewhat dangerous, the algorithm is still able to select the safest velocity and therefore has a better chance at avoiding aggressive obstacles and reaching the goal.

Based on the comparable performance of VOS and VO algorithms in simulations with complete obstacle knowledge and the superior performance of VOS in simulations without complete obstacle knowledge, VOS is again shown to be a useful addition to the literature.

2.6. Conclusions about VOS

In this chapter *velocity occupancy space* (VOS), a navigation algorithm which allows a robot to operate using only a range finding sensor with uncertainty in an unknown environment and successfully avoid stationary and moving obstacles while navigating towards a goal, has been developed and presented. This method uses the uncertain obstacle representation of occupancy space to estimate the location of each obstacle and finds the *center of certainty*, a variation of the center of mass, for each obstacle. The center of certainty of each obstacle is tracked over multiple time steps and the movement of the center of certainty is used to estimate the obstacle's velocity. This basic obstacle information is then converted into velocity obstacle form and used to calculate variables that describe the benefits or detriments of each possible robot velocity. The relative weights that each of these variables should have, in comparison to the other variables, were then optimized and used to form VOS. From this space, the robot can find a velocity that is both safe and that will lead it towards the goal, if such a velocity exists. While the choice of velocity using the optimized weights may not always be ideal, it has been shown that, in general, the weights will allow the robot to avoid a collision and reach its destination in the vast majority of situations. The obstacle avoidance and goal finding abilities of VOS were also evaluated against two other obstacle avoidance algorithms and VOS was shown to have at least comparable capabilities as these algorithms.

Chapter 3

Velocity Occupancy Space for Differential Drive Vehicles

The Velocity Occupancy Space (VOS) algorithm, which was introduced and described in the previous chapters, selects robot velocities under the assumption that the robot is holonomic and is able to instantaneously accelerate to the desired velocity. Unfortunately, this assumption is not valid for real-world robots, and therefore the algorithm cannot ensure robot safety under realistic experimental conditions, especially at high speeds.

Therefore, this chapter will focus on a method by which VOS (and the use of velocity obstacles, in general) can be extended to accommodate non-holonomic robots with acceleration constraints. Specifically, this chapter focuses on how a sequence of accelerations can be used to approximate a desired velocity and how the velocity selection in VOS can be restricted in order to accommodate the constraints of a differential drive robot.

3.1. Differential Drive Formulation

In this section it is shown how a series of accelerations can be generated in order to approximate a desired instantaneously change in velocity for a differential drive vehicle. Intuitively, when a path is planned for a differentially driven robot, the path would consist of a series of arcs of various radii (when the two wheels have different velocities) and straight lines (when the wheels have the same velocity). However, the holonomic robot assumption in VOS implies a series of constant, discontinuous, linear velocities. This series of constant linear velocities can be approximated with a kinodynamically feasible set of arcs and lines by a) considering only the robot's initial and final position and velocity (at the beginning and end of each motion time step), and b) selecting a series of piecewise-continuous accelerations that will produce a differentiable and continuous velocity profile. This series of accelerations will allow the robot to reach the same approximate velocity and (for the three-step approximation) position at the end of each motion time step as the velocity chosen from the VOS search space for a holonomic vehicle.

The piecewise constant acceleration approximation of a holonomic velocity is necessary because, in VOS, the velocity obstacles cannot be simply altered to take the robot dynamics into account. Currently, a single velocity obstacle is created for each physical obstacle and is used to determine the relative safety of each robot velocity. In order to take the robot's dynamics into account, a separate velocity obstacle would have to be made for every potential robot velocity. Depending on the velocity space resolution, this could increase the computational complexity of populating VOS by over a thousand fold at each motion time step.

When calculating the piecewise accelerations it is acceptable to consider only the robot's position and velocity at the beginning and end of each motion time step, with length Δt_m, as long as each time step is short enough that significant movement of the robot, or of surrounding obstacles, would not be expected. In VOS, the robot is always required to maintain a safe distance from any surrounding obstacles due to the error present in the system which makes exact localization and velocity prediction for the obstacles impossible. If Δt_m is short enough, the robot's position will not deviate significantly from the position that it is attempting to reach so the risk of a collision is minimal.

One of the most common kinematic configurations for robots used as experimental platforms is a basic differential drive vehicle (usually with a third caster wheel for stability). In Figure 3.1, a simple differential drive vehicle is shown at two consecutive time steps. The vehicle has a wheel radius *d* and the distance between the two wheels is *L*. When calculating the movement of the vehicle between two times steps, the vehicle's initial position is considered to be oriented along the x-axis (in a vehicle based coordinate frame) and the vehicle's heading in its final position, ψ, is given relative to this axis.

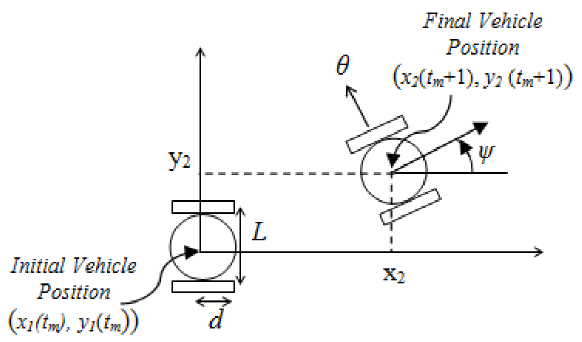


Figure 3.1 Differential drive vehicle at two, consecutive motion time steps

ISO orientation and SAE coordinates are used to define the position, rate of change and acceleration of the vehicle's heading and wheels angles. These coordinates are shown in Table 3.1.

Table 3.1 Coordinates used in derivation

Angle	Rate	Acceleration
θ	$q = \frac{(q_l + q_r)}{2}$	$Q = \frac{(Q_l + Q_r)}{2}$
$\psi = \tan^{-1}\left(\frac{v_{y2}}{v_{x2}}\right)$	$r = \frac{d}{L}(q_r - q_l)$	$R = \frac{d}{L}(Q_r - Q_l)$

The subscripts l and r are used to differentiate between the left and right wheels. The vehicle is defined to be initially moving in only the x-direction, $v_y = 0$, and the velocity at the beginning and end of each motion time step is assumed to be constant (i.e $q_l = q_r$ and $Q_l = Q_r = 0$). These assumptions allow the vehicle's movement to more closely mimic that of a vehicle that is able to instantaneously change velocities.

The third time step, the acceleration time step, t_a, is used when calculating accelerations for a differential drive vehicle. The acceleration time step has a lower bound of the frequency with which the vehicle's actuators can respond to acceleration commands, while the motion time step is the time allotted for the vehicle to execute the series of accelerations and attain the desired velocity. The acceleration time step and the sensor time step are only related to each other through their relationships to the motion time step; they are not directly dependent on each other.

For this derivation, it is assumed that the vehicle cannot instantaneously change velocity, but that it can instantaneously change acceleration: jerk limits are not considered. The length of the acceleration time step is either one half (for the two-step method) or one third (for the three-step method) of the length of the motion time step, Eq.(3.1). It is assumed that the vehicle can accelerate quickly enough that the vehicle acceleration can be approximated as constant over the acceleration time step.

$$\Delta t_a = \begin{cases} \frac{\Delta t_m}{2} & Two - Step \\ \frac{\Delta t_m}{3} & Three - Step \end{cases} \tag{3.1}$$

3.1.1. Two-step Velocity Approximation Method

The first method considered is a naïve approximation of the selected velocity using a two-step series of accelerations for each of the two wheels (see Figure 3.2). That is the motion time step, Δt_m, is divided into two equal segments, and constant wheel accelerations (for each wheel) are specified during each segment. These wheel accelerations allow the robot to change its heading and speed in order to match the desired command velocity by the end of the motion time step. However, the robot's final position does not necessarily match that of a robot that instantaneously started moving at the desired velocity.

As shown in Figure 3.2, the initial and final angular wheel velocities must be equal (i.e. $q_{l1} = q_{r1} = q_{t_m}$ and $q_{l2} = q_{r2} = q_{t_m+1}$) in order for the robot change from its previous command (constant, linear) velocity to the new command (constant, linear) velocity. As the acceleration time step length is fixed ($\Delta t_a = \frac{\Delta t_m}{2}$) this constrains the angular wheel accelerations to maintain the relationships:

$$Q_{l1} = Q_{r2} \tag{3.2}$$

and

$$Q_{l2} = Q_{r1} \tag{3.3}$$

where Q_{r1} is the angular acceleration of the right wheel during the first acceleration time step (the first half of the motion time step). The other angular accelerations are similarly distinguished.

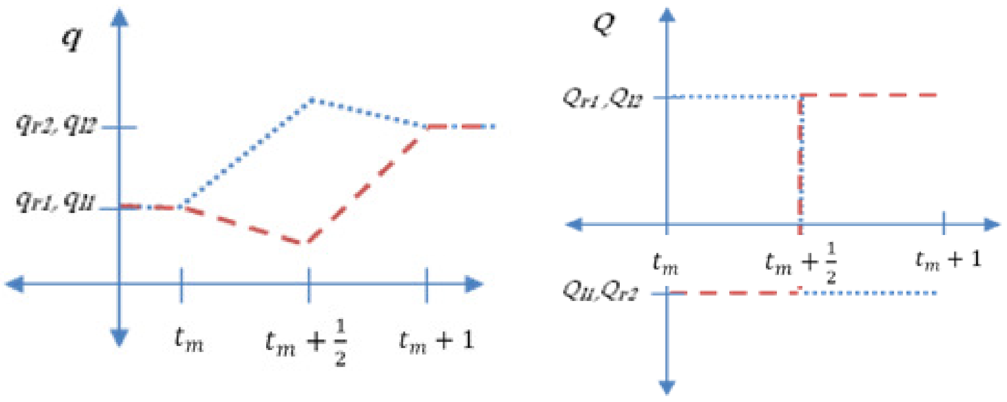


Figure 3.2 Angular wheel velocity and acceleration for two step method. (Note, figures are not to scale)

In addition, basic equations of motion and the equations in Table 3.1 produce the relationship:

$$q_{t_m+1} - q_{t_m} = \left[\frac{(Q_{l1} + Q_{l2})}{2} + \frac{(Q_{r1} + Q_{r2})}{2}\right]\Delta t_m \tag{3.4}$$

between the initial and final angular wheel velocities.

The change in the robot's heading, ψ, also puts further constraints on the angular wheel accelerations which (again using basic equations of motion and the equations in Table 3.1), generates the relationship:

$$\psi_{t_m+1} = \frac{1}{2}\left[\frac{(Q_{r1} - Q_{l1})}{2} + \frac{(Q_{r2} - Q_{l2})}{2}\right]\Delta t_m{}^2. \tag{3.5}$$

Solving Eqs. (3.2-3.5) simultaneously for the angular accelerations of each wheel produces the following equations:

$$Q_{r1} = Q_{l2} = \frac{q_{t_m+1} - q_{t_m}}{\Delta t_m} + \frac{2\psi L}{d(\Delta t_m)^2} \tag{3.6}$$

$$Q_{r2} = Q_{l1} = \frac{q_{t_m+1} - q_{t_m}}{\Delta t_m} - \frac{2\psi L}{d(\Delta t_m)^2}. \tag{3.7}$$

The offset error in the robot's position at the end of the motion time step arises due to the difference in magnitude and direction between the initial and final velocities. As shown in Figure 3.3, the desired final robot position is equal to the (holonomic) velocity command (v_{x2}, v_{y2}) multiplied by the length of the motion time step (Δt_m). Using the two-step acceleration method, the robot will reach the desired velocity (v_{x2}, v_{y2}) and therefore will also be oriented along the desired heading, ψ, but it will not (in general) reach the desired position unless the new command velocity is the same as the robot's current velocity.

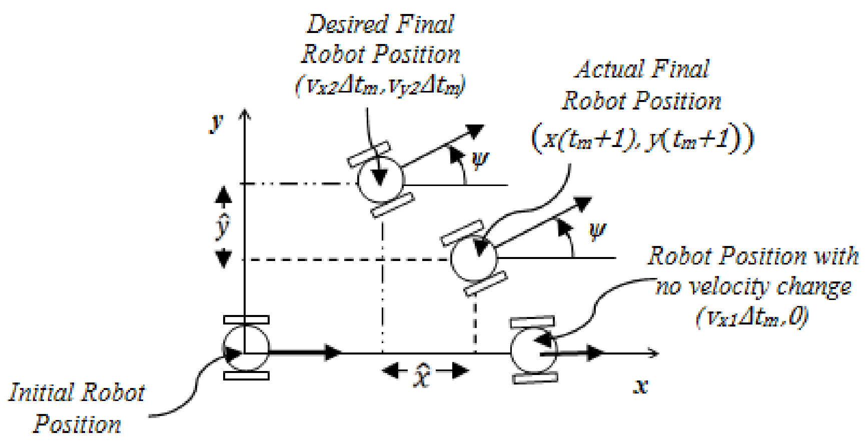

Desired Final Robot Position
$(v_{x2}\Delta t_m, v_{y2}\Delta t_m)$
Actual Final Robot Position
$(x(t_m+1), y(t_m+1))$
y
ψ
ψ
$\hat{y}$
Robot Position with no velocity change
$(v_{x1}\Delta t_m, 0)$
Initial Robot Position
$\hat{x}$
x

Figure 3.3 Possible robot positions at the end of the motion time step

The robot's position at the end of the time step can be calculated using:

$$x(t_m+1) = x(t_m) + \int_0^{\Delta t_a} [q_{t_m} d + Q_1 \tau d] \cos\left[\tfrac{1}{2} R_1 \tau^2\right] d\tau$$
$$+ \int_{\Delta t_a}^{\Delta t_m} [q_{t_m+\Delta t_a} d + Q_2 \tau d] \cos\left[\psi_{t_m+\Delta t_a} + r_{t_m+\Delta t_a}\tau + \tfrac{1}{2} R_2 \tau^2\right] d\tau \qquad (3.8)$$

and

$$y(t_m+1) = y(t_m) + \int_0^{\Delta t_a} [q_{t_m} d + Q_1 \tau d] \sin\left[\tfrac{1}{2} R_1 \tau^2\right] d\tau$$
$$+ \int_{\Delta t_a}^{\Delta t_m} [q_{t_m+\Delta t_a} d + Q_2 \tau d] \sin\left[\psi_{t_m+\Delta t_a} + r_{t_m+\Delta t_a}\tau + \tfrac{1}{2} R_2 \tau^2\right] d\tau \qquad (3.9)$$

where the variables are defined in Table 3.1 and the subscripts refer to the first and second acceleration time steps. The first integral in each equation calculates the distance that the robot travels (in the respective direction) during the first acceleration time step $(0\ to\ \Delta t_a)$ and the second integral is the distance that the robot travels during the second acceleration time step$(\Delta t_a\ to\ \Delta t_m)$.

The error in the robot's position is therefore:

$$\hat{x} = v_{x2}\Delta t_m + x(t_m) - x(t_m + 1) \tag{3.10}$$

$$\hat{y} = v_{y2}\Delta t_m + y(t_m) - y(t_m + 1) \tag{3.11}$$

The robot's position at the end of the motion time step (see Figure 3.3) will be somewhere between the position that it would have reached had it continued at its previous velocity $(v_{x1}\Delta t_m, v_{y1}\Delta t_m)$ and the desired position based on the new command velocity, $(v_{x2}\Delta t_m, v_{y2}\Delta t_m)$, therefore the greater the change between the initial and final velocity (magnitude and heading), the more offset error, $(\hat{x}, \hat{y})$, there will be in the robot's final position.

The calculated velocity obstacle (as constructed in Figure 2.9) is based on both the distance between the robot and the obstacle, $\vec{\lambda}_{i,r}$, as well as the obstacle's velocity. Therefore, regardless of the robot's ability to reach the desired velocity within one motion time step, if the robot's position at the end of that time step is different from what it would have been had it instantaneously changed velocities, then there is an offset in velocity space. This offset invalidates the assumption of continuous obstacle avoidance for velocities outside of the velocity obstacle.

In Figure 3.4 the construction of a velocity obstacle, $\overrightarrow{VO_i}$, is shown both as it would be calculated assuming that the robot could instantaneously change velocity (shown in solid lines), as well as what the actual velocity obstacle would be if acceleration constraints were taken into account (shown in bold and dashed lines). It should be remembered that the velocity obstacle consists of all of the robot velocities that will lead to a collision between the robot and the obstacle at some point in time. Under the instantaneous acceleration assumption, almost all of the velocities along the actual velocity obstacle are considered safe. This assumption may cause real-world robots to inadvertently collide with an obstacle.

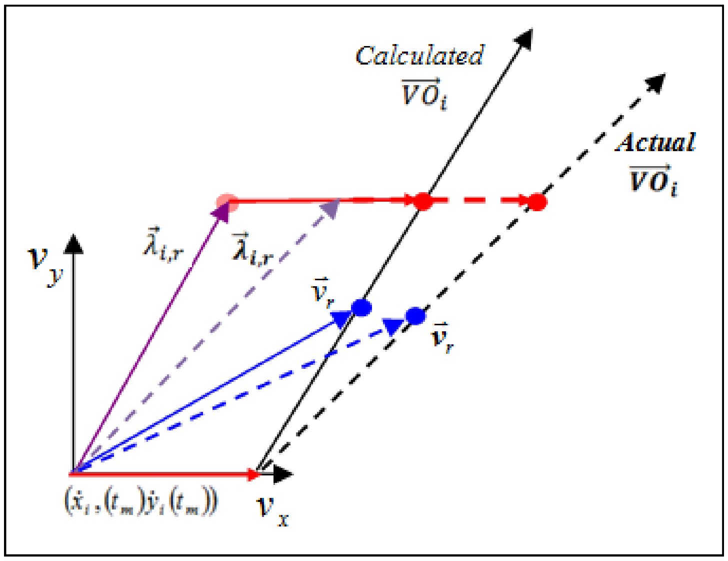

Calculated $\overrightarrow{VO_i}$
Actual $\overrightarrow{VO_i}$
v_y
$\vec{\lambda}_{i,r}$
$\vec{\lambda}_{i,r}$
$\vec{v}_r$
$\vec{v}_r$
$(\dot{x}_i(t_m)\dot{y}_i(t_m))$
v_x

Figure 3.4 Velocity space representation of the robot, with velocity $\vec{v}_R$, an obstacle, with velocity $(\dot{x}_i(t_m), \dot{y}_i(t_m))$, and the velocity obstacle, $\overrightarrow{VO_i}$, where $\vec{\lambda}_{i,r}$ is the vector between the robot and obstacle. The velocity obstacles and components are calculated both with instantaneous acceleration assumptions (solid lines) and with actual robot dynamics (bold, dashed lines).

The effects of this offset may range from negligible to catastrophic depending on many factors. For instance, if the obstacle position and velocity estimation error is significantly greater than the offset error, then it is very likely that the offset error will not cause a collision. In this situation, because the error inherent to the system is already forcing the VOS algorithm to choose overly-cautious velocities that avoid obstacles by a large margin, the offset will probably not be enough to cause a collision. However, in a system with less error or with more quickly moving obstacles, this offset could cause problems. The motion time step, t_m, also plays a role in the offset error. As t_m decreases, the offset error and overall distance traveled also proportionally decrease. However, a shorter t_m also means that the robot requires much greater accelerations to reach the same desired velocity.

3.1.2. Three-step Approximation Method

Due to the offset error produced by the two-step velocity approximation method, a three-step velocity approximation method, which will both accelerate the robot to the desired velocity and move it to the position that it would have reached had it been moving at the desired velocity for the entire time step, will be considered. In this case, the motion time step, Δt_m, is divided into three equal segments, and the left and right wheel accelerations are determined for each segment.

The three-step method requires the angular wheel accelerations to fulfill the constraints used in the two-step method (final velocity and turn rate, and change in heading) but also constrains the final position of the robot in the x- and y-directions. To simplify this approximation, the required change of heading is accomplished by the first two sets of accelerations and the final set of accelerations, $Q_3,$ is the same for both wheels. The accelerations can be found from the following equation:

$$\begin{bmatrix} 1 & 1 & 1 & 1 & 2 \\ 1 & -1 & -1 & 1 & 0 \\ 1 & -1 & 1 & -1 & 0 \\ a+2b & a+2b & b & b & -2c \\ e+2f & e+2f & f & f & -2g \end{bmatrix} \begin{bmatrix} Q_{r1} \\ Q_{l1} \\ Q_{r2} \\ Q_{l2} \\ Q_3 \end{bmatrix} = \begin{bmatrix} \left(q_{t_m+1} - q_{t_m}\right)\left(\frac{6}{\Delta t_m}\right) \\ 2\psi\left(\frac{3}{\Delta t_m}\right)^2\left(\frac{L}{d}\right) \\ 0 \\ \left(\frac{6v_{x2}}{d} - 2q_{t_m}(a+b) - 2cq_{t_m+1}\right)\left(\frac{6}{\Delta t_m}\right) \\ \left(\frac{6v_{y2}}{d} - 2q_{t_m}(e+f) - 2gq_{t_m+1}\right)\left(\frac{6}{\Delta t_m}\right) \end{bmatrix} \tag{3.12}$$

where the variables *a* through *f* are defined as follows:

$$\begin{aligned} a &= cos(0.163\psi) \\ b &= cos(0.837\psi) \\ c &= cos(\psi) \\ d &= sin(0.163\psi) \\ e &= sin(0.837\psi) \\ f &= sin(\psi) \end{aligned} \tag{3.13}$$

In Eq. (3.12), the first row of the matrix forces the final robot speed to be equal to the command speed. The equation that comprises this row is very similar to Eq. (3.4), only modified so that the command speed is reached after three time steps, instead of two. The second row of Eq. (3.12) constrains the final robot heading to equal the angle ψ. The equation in this row has the same purpose and is analogous to Eq. (3.5), except here it is modified to account for the three sets of accelerations. Q_3 is multiplied by zero in this row because all of the heading change occurs during the first two accelerations. The third row of Eq. (3.12) is used to make the turn-rate of the robot equal to zero (i.e $q_l = q_r$ and $Q_l = Q_r = 0$) at the end of the time step. This line is produced by combining Eqs. (3.2 and 3.3).

The last two rows of Eq. (3.12) constrain the change in the robot's position, during the motion time step, to be approximately equal to the distance that a holonomic, instantly accelerating robot would travel during that time step $\left(i.e. \Delta x \approx v_{x2}\Delta t_m \; and \; \Delta y \approx v_{y2}\Delta t_m\right)$, see Figure 3.5. However, as integration would be required in order to calculate the exact distance that the robot travels while it is accelerating the distance is estimated with Euler approximations and by using the average robot heading and speed for each acceleration time step. This is done because it would not be possible to explicitly solve the matrix for the wheel accelerations if they were contained within an integral.

The robot's position during each acceleration time step can be calculated using the average angular wheel velocity and heading from that time step. In Table 3.2 the robot's position (in the x-dimension) and angular wheel velocities are given over the course of the acceleration time step shown in Figure (3.5).

Table 3.2 Robot positions and wheel velocities during three-step acceleration

Interval	*x*-Position (at end of time step)	Average heading (during time step)	Angular wheel velocity (at end of time step)
0 to Δt_a	$x_1 = \left[\frac{dq_0+dq_1}{2}\right]\frac{\Delta t_a}{3}cos\bar{\psi}_1$	$\bar{\psi}_1 = 0.162\psi$	$q_1 = q_{t_m} + \frac{(Q_{r1}+Q_{l1})}{2}\frac{\Delta t_a}{3}$
Δt_a to $2\Delta t_a$	$x_2 = x_1 + d\left[\frac{q_1+q_2}{2}\right]\frac{\Delta t_a}{3}cos\bar{\psi}_2$	$\bar{\psi}_2 = 0.837\psi$	$q_2 = q_1 + \frac{(Q_{r2}+Q_{l2})}{2}\frac{\Delta t_a}{3}$
$2\Delta t_a$ to $3\Delta t_a$	$x_3 = x_2 + d\left[\frac{q_2+q_3}{2}\right]\frac{\Delta t_a}{3}cos\psi_3$	$\psi_3 = \psi$	$q_3 = q_2 + Q_3\frac{\Delta t_a}{3}$

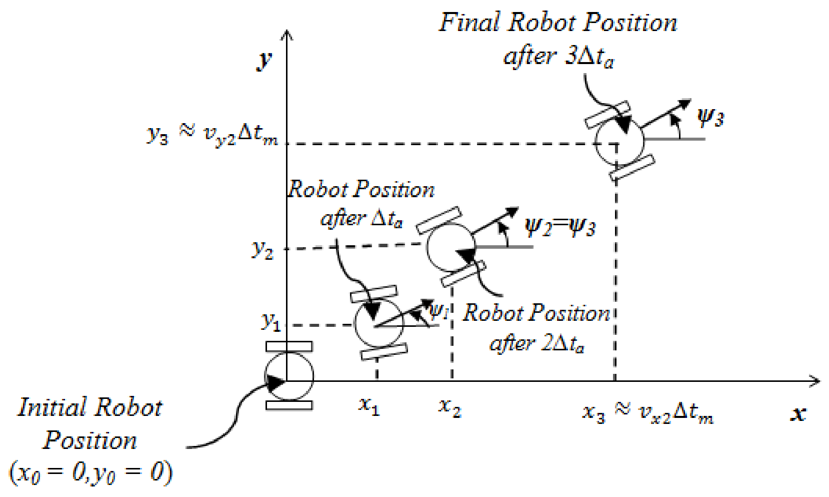


Figure 3.5 Robot positions during three-step acceleration
Coordinates are based on initial, local robot frame for simplicity

The robot position at the end of the motion time step is x_3 and the relationship

$$v_{x2}\Delta t_m = x_3 \tag{3.14}$$

can be rearranged to form the fourth row of Eq. (3.12) (after the appropriate angular wheel accelerations have been substituted into the equation). The same process can be used to form the fifth row of Eq. (3.12), if all of the cosine functions are replaced with sine functions and *x* replaced with *y*.

The variables *a* through *f*, Eq. (3.13), utilize the average robot heading during the first third of (*a* and *d*), the second third (*b* and *e*) and the final third (*c* and *f*) of the motion time step. These values were determined empirically, and found to be constant through all possible velocity changes that could occur between velocities of $\left[-4\frac{m}{s}, 4\frac{m}{s}\right]$ in both directions.

This distance approximation still includes some error, which increases with greater changes of the robot heading, ψ. This error can be reduced by breaking each actuator time step down into smaller components (thus more closely approximating the integral) and solving for the distance traveled in each direction during each of these smaller components. However, it was found in simulation that the error with the three-part approximation was usually a few orders of magnitude smaller than the actual distance traveled during the motion time step, which made the risk of a collision (due to positional error arising from the lack of integration) negligible. The change in the length of Δt_m, again, has an effect on both the acceleration and position error. Smaller values of Δt_m necessitate higher accelerations to reach the same velocity. However smaller values of Δt_m also produce proportionally smaller overall position changes as well as error in the final position. The opposite is true for larger values of Δt_m.

The set of linear equations, Eq. (3.12), can be solved explicitly for the five accelerations (i.e., Q_{r1}, Q_{l1}, Q_{r2}, Q_{l2}, and Q_3).

In the case where $\psi = 0$ (i.e. there is no change in heading, $v_{y2} = 0$) the matrix in Eq. (3.12) becomes singular and it is necessary to use another method to find the required accelerations. Conveniently, if no change in heading is required, there are only two constraints on the required accelerations (final velocity and distance traveled in the x-direction) so only two sets of accelerations are needed in order to approximate the change in velocity and position. The accelerations are the same for both wheels and there is no error in the final position (because the distance traveled can be found without approximate integration). The accelerations then are:

$$Q_{r1} = Q_{l1} = 3 \cdot \left(\frac{q_{t_m+1} - q_{t_m}}{\Delta t_m}\right) \tag{3.15}$$

$$Q_{r2} = Q_{l2} = \left(\frac{q_{t_m} - q_{t_m+1}}{\Delta t_m}\right) \tag{3.16}$$

3.2. Comparison of the Two- and Three-Step Methods

3.2.1. Modified Velocity Occupancy Space

The kinodynamic velocity approximations were included in the VOS algorithm in two places. First, the simulation was augmented so that the robot's movement was restricted by differential drive constraints and the robot's position and velocity were found by integrating over each of the acceleration time steps. Second, the kinodynamic velocity approximations where used to restrict the velocity search space so that the algorithm could only choose velocities for which the robot had sufficient acceleration capability. In other words, accelerations necessary to mimic each velocity in the velocity search space of VOS were computed. If the accelerations were outside the bounds of the robot capabilities, then that velocity was deemed unreachable and would not be considered for the robot at that time step. Initially, this greatly increased the computational cost of populating VOS (especially when using the three step acceleration method), however, the net effect on the computational load was almost insignificant as it was no longer necessary to compute the attractive and repulsive weight for each velocity which was considered unreachable and this greatly reduced the computational cost of populating VOS.

3.2.2. Results and Comparison of Methods

Figures 3.6 and 3.7 show simulations of a robot avoiding moving and stationary obstacles as it navigates to a goal, all while operating under the constraints of an acceleration limited, differential drive robot. The same simulation parameters (i.e. obstacle locations and velocities, robot acceleration constraints, etc.) were used for the simulations shown in both figures. Using both methods, the robot is able to successfully find a collision free path to the goal. In both figures, the grey line originating from the robot is the robot's velocity vector for each motion time step (see Figure 3.6). The vector also points to the location that the algorithm expects the robot to move to over the course of the next time step. Using the two-step approximation, the robot frequently does not end up exactly where the algorithm expects. However, as the desired velocity is recomputed after every time step, the robot is still able to quickly find the goal.

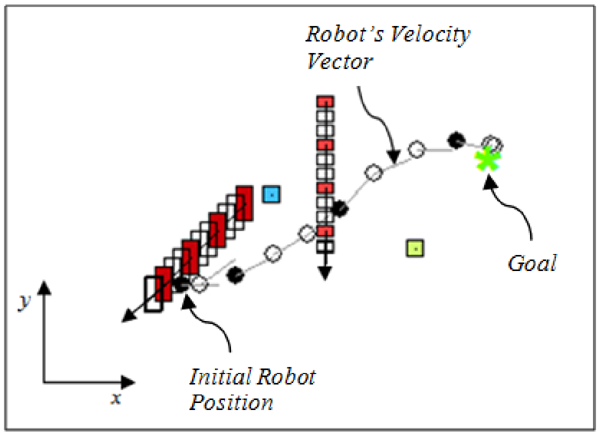


Figure 3.6 Simulation, using two-step velocity approximation, of the robot (circle), obstacles (rectangles) and the goal (asterisk). The robot takes 12 motion time steps to reach the goal.

Using the three-step approximation, Figure 3.7, the robot's final position at the end of each time step is almost indistinguishable from the expected position. However, it takes the robot almost twice as long to reach the goal. This extra time is due to the more restricted velocity search space. Using the three-step approximation, when the robot needs to turn, it is usually required to slow down substantially in order to change its heading (and still end up at the desired position velocity) while not exceeding the acceleration constraints. However, the three-step method reduces the error in the robot's position at the end of each motion time step by (on average) 84.33% over the robot position found using the two-step method. If the robot is close to an obstacle, this improvement could make the difference between safety and a collision. However, in order to improve the position, the three step method requires (on average) 44.57% higher acceleration to reach the same velocity, and must therefore frequently use lower velocities than the two-step method due to the robot's acceleration constraints.

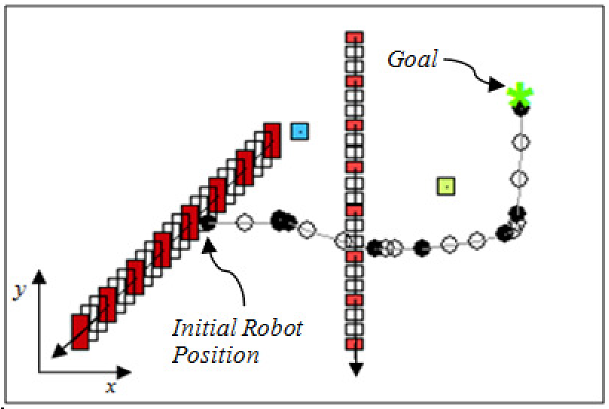


Figure 3.7 The same simulation as in Figure 3.6, but with the three-step velocity approximation. The robot takes 22 motion time steps to reach the goal.

In Figures 3.9 and 3.10, the velocity search space of the very simple scenario in Figure 3.8 is shown for both the two- and three- step approximations. In these figures, positive weights indicate repulsive regions in velocity space (i.e., velocities which are dangerous as they may lead to a collision) and negative weights indicate attractive regions (i.e., velocities which should safely avoid a collision). Kinodynamically unreachable velocities were given a repulsive weight slightly higher than the most repulsive (but reachable) velocity so that they would never be selected by the algorithm (in Figure 3.10, the majority of the velocities, shown in white, are unreachable). In Figures 3.5 and 3.6, the velocity obstacles of these two obstacles shown in Figure 3.8 can be seen as the cones of repulsively-weighted velocities (weighted a little under thirty) in Figure 3.9. It should be noted that x- and y-axes in Figure 3.10 are an order of magnitude smaller than those in Figure 3.9. This scaling choice was made to increase the scale of Figure 3.10 so that the reachable velocities could be more easily seen.

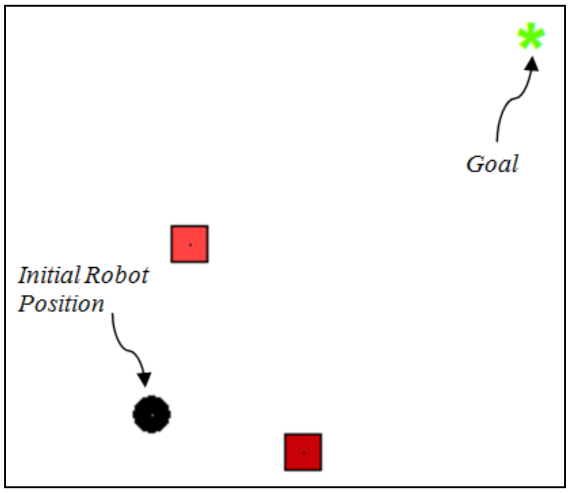


Figure 3.8 Simple goal/obstacle scenario

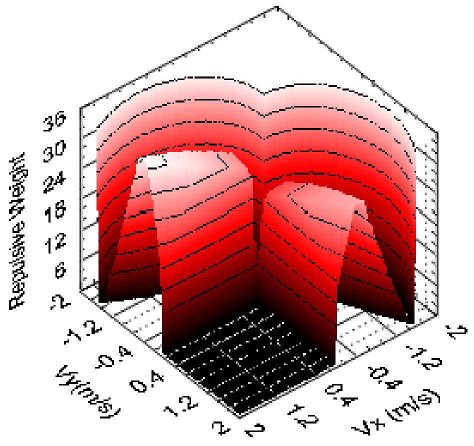


Figure 3.9 Velocity search space of scenario in Figure 3.8 using the two-step velocity approximation

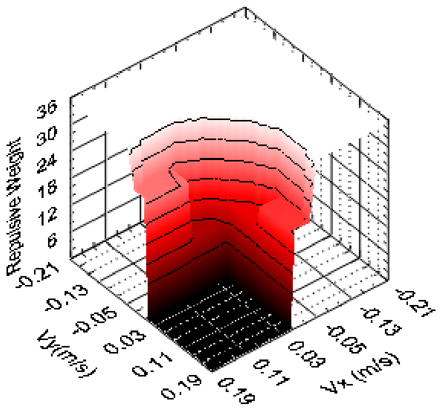


Figure 3.10 Velocity search space of scenario in Figure 3.8 using the three-step velocity approximation (the axes are an order of magnitude smaller than those in Figure 3.9)

The two-step approximation provides a much richer velocity search space for the algorithm (as shown in Figure 3.9) and the robot is generally able to move more quickly to the goal. The constraint of having to reach a specific position as well as velocity for the three-step approximation method means that much greater accelerations are required using this method, than are required using the two-step method, in order to approximate the same desired velocity. However, the loss of accuracy in the final position, using the two-step method, can have highly undesirable consequences.

In Figure 3.11, a slightly different simulation than that shown in Figures 3.6 and 3.7 is presented using the two-step velocity approximation method. By the fifth motion time step, the algorithm has detected *Obstacle A* and attempts to speed up the robot in order to cross in front of the obstacle (in the negative y-direction). However, the robot does not reach the desired position by the end of the time step, so, in the next time step, the algorithm chooses a velocity that will allow the robot to turn sharply and move in the positive y-direction and avoid the obstacle. Unfortunately, again the robot does not end up in the desired position and this leads to a collision. Although not shown here, the three-step method was able to safely reach the goal under the same simulation conditions.

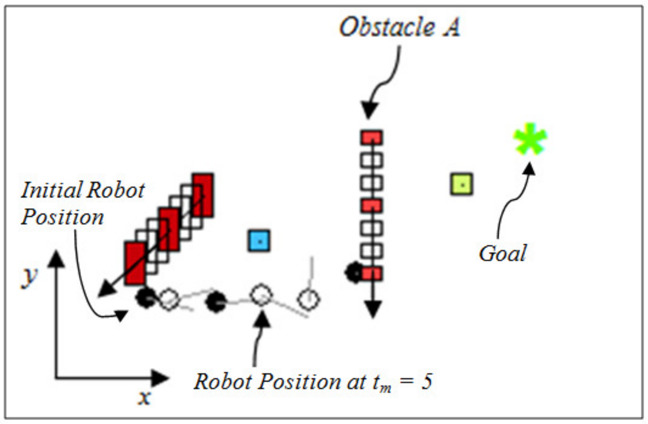


Figure 3.11 Failed simulation using two-step velocity approximation method

3.3. Obstacle Proximity Dependent Method

The two- and three-step acceleration methods have complementary strengths; the two-step method enables the robot to move efficiently to a goal, but with a greater possibility of a collision, while the three-step method is safer, but much less efficient at reaching the goal. Therefore, a combination of the two methods that utilizes the strengths of each has been developed. This method allows the algorithm to alter is velocity selection method based on the proximity of surrounding obstacles.

3.3.1. Proximity Dependent Method

The proximity dependent method was initially designed so that the algorithm selected the method (two- or three-step) solely based on the distance to the nearest obstacle. However, it was found that this method frequently led to over-conservative (unnecessarily switching to the three-step method) or reckless (unsafely switching to the two-step method) switching. The amount of positional error possible at each time step using the two-step method varies greatly - based on the robot's current velocity, the maximum velocity of which it is capable and its acceleration capabilities - so a switch in methods may be overly-conservative or reckless during one part of a simulation, but appropriate during another, even if an obstacle is the same distance away. Therefore, in order to more precisely tune the selection of the method to the specific scenario, the expected positional error for a possible velocity change was used.

Ideally, the (two-step) positional error would be found for every possible velocity change, while building VOS, and if that error for any velocity change was greater than the distance between the robot and the nearest obstacles then that velocity change would be considered unreachable and the three-step method would be used instead to ensure safety. However, as it would be extremely computationally expensive to determine all the potential positional errors while building velocity occupancy space at every time step, the maximum positional errors were pre-computed off-line and built into look-up tables. In this manner, the maximum positional error could be quickly found and used to determine the appropriate acceleration method based on the current proximity to the closest obstacle. This is a less precise approximation than considering the position error for every possible velocity change, but it greatly decreases the required computational time while still allowing the robot's current state (velocity and acceleration) to be considered.

Tables were built for a range of possible maximum robot velocities and accelerations. For each table, the current robot x- and y-velocities were used along with the maximum robot velocities and wheel accelerations to create an array of every possible subsequent robot velocity. Using this array, and Eq (3.10 and 3.11), the positional error for every velocity change was found and the maximum positional error was added to the table.

The maximum positional error was found so that the acceleration method could be selected before constructing VOS for each time step. Because the next velocity to be selected could not be known at this point, it was necessary to use the maximum position error.

The maximum position error ranged from 0.2m for low speed, low acceleration cases (maximum velocity = $0.5\frac{m}{s}$, maximum acceleration =$|1|\frac{m}{s^2}$) to 3.47m for high speed, high acceleration cases (maximum velocity = $2\frac{m}{s}$, maximum acceleration =$|6|\frac{m}{s^2}$).

The proximity dependent method was run with the two-step method as the default acceleration method, unless the following inequality was satisfied:

$$\left\|\left(x_i(t_m), y_i(t_m)\right)_c - \left(x_r(t_m), y_r(t_m)\right)\right\| - \left\|\max\left(\hat{x}(t_m), \hat{y}(t_m)\right)\right\| < \left(\max\left(\dot{x}_r(t_{m+1}), \dot{y}_r(t_{m+1})\right)\right) \cdot \Delta t_m \tag{3.8}$$

where $\left(x_i(t_m), y_i(t_m)\right)_c$ is the position of the closest obstacle, $\left(x_r(t_m), y_r(t_m)\right)$ is the robot's current position, $max\left(\hat{x}(t_m), \hat{y}(t_m)\right)$ is the maximum position error that is possible at the next time step (obtained from the pre-computed look-up table) and $max\left(\dot{x}_r(t_{m+1}), \dot{y}_r(t_{m+1})\right)$ is the maximum possible velocity of which the robot is capable during the next time step.

Therefore, if during each motion time step, the magnitude of the distance between the robot and the closest obstacle minus the maximum positional error was less than the distance that the robot could travel during that time step, then the three-step method was used.

3.3.2. Simulation Results

Two sample simulations using the proximity dependent method are shown below. In both sets of figures, the algorithm uses the two-step method when the robot is shown in green and the three-step method when the robot is shown in blue.

In the first set of figures (Figures 3.12 – 3.16), due to the cluttered environment in this simulation, the algorithm is mostly dependent upon the three-step method. The two-step method is used for the first five time steps (Figure 3.12). Until, at time step five, *Obstacle A* approaches closely enough that the algorithm switches to the three-step method. The change in methods is due both to the obstacles proximity as well as to the relatively high speed at which the robot was moving. In Figure 3.13, the algorithm briefly uses the two-step method (time step 13), but then relies on the three-step method to avoid *Obstacle B*.

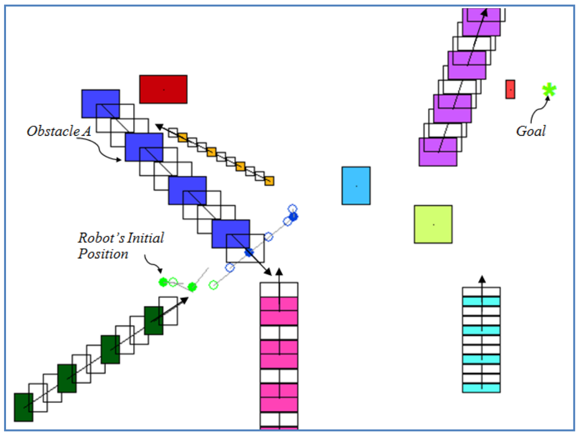


Figure. 3.12 Simulation using the proximity dependent method. Time steps 1 – 11

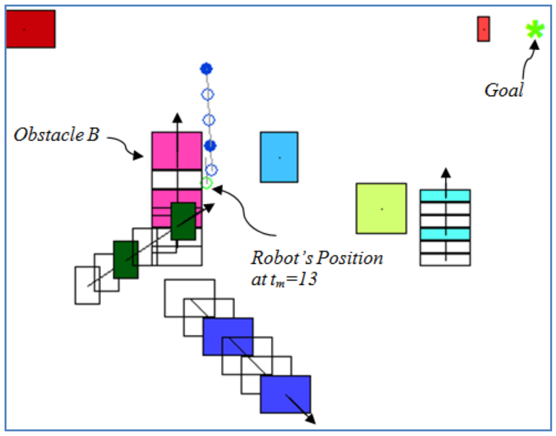


Figure 3.13 Time steps 13-18

Between time steps 18 and 32, Figure 3.14, the algorithm occasionally uses the two-step method, but is mostly dependent on the three-step as the robot is moving almost parallel to *Obstacle C*. In Figure 3.15, the robot is carefully avoiding both *Obstacles C* and *D,* and therefore exclusively using the three-step method. Finally, in Figure 3.16, the robot is able to reach the goal fairly efficiently – though the proximity of *Obstacle E* at the end of the simulation forces the algorithm to again mostly use the three-step method.

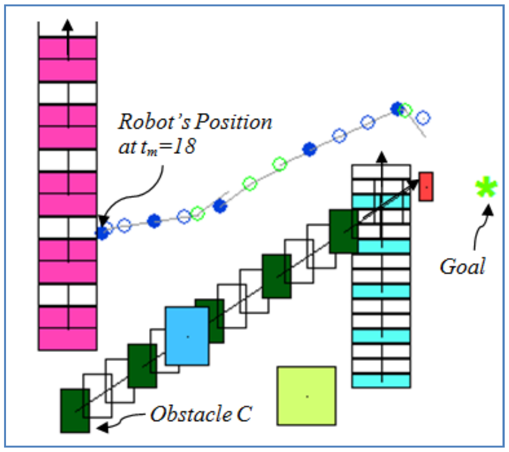


Figure. 3.14 Time steps 18–32

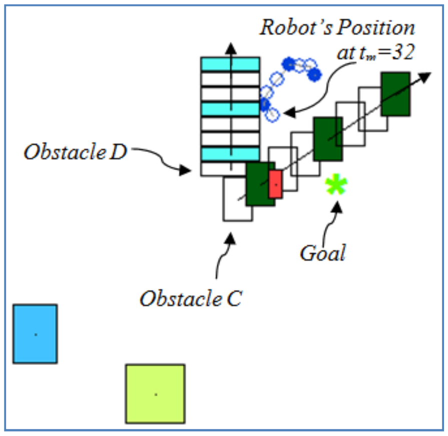


Figure 3.15 Time steps 32-39

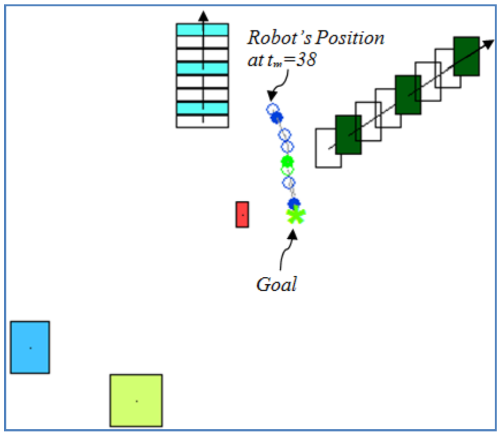


Figure 3.16 Time steps 38-45

For the next simulation, shown in Figures 3.17 though 3.19, which is less cluttered, the algorithm depends more heavily on the two-step method in order to navigate to the goal. In Figure 3.17, the algorithm almost exclusively uses the two-step method, except briefly when the robot is close to *Obstacles A* and *B.*, Next, in Figure 3.18, the three-step method is used to avoid a collision while the robot is navigating between *Obstacles A* and *C*. Finally, after the robot clears *Obstacle C*, it is able to reach the goal using the two-step method, Figure 3.19.

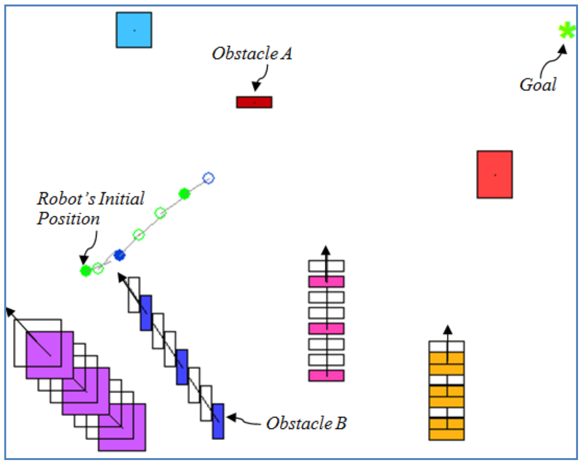
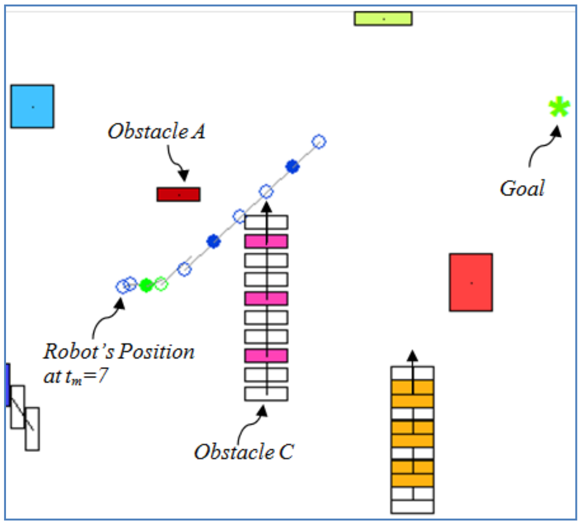


Figure. 3.17 Simulation using the proximity dependent method.
Time steps 1-7



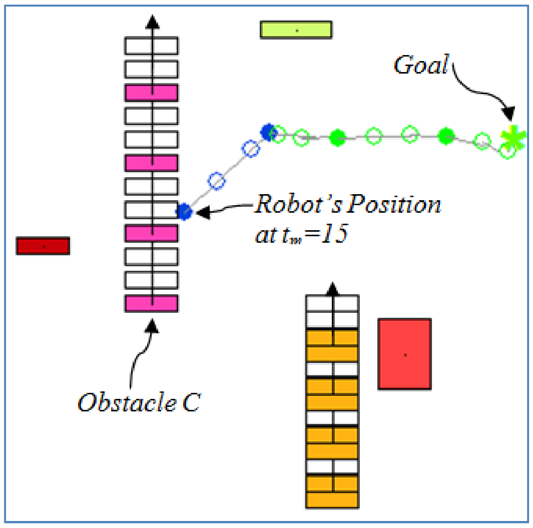


Figure 3.18 Time steps 7-16

Figure 3.19 Time steps 15-26

3.3.3. Comparison of the Three Methods

Using a random obstacle generator, similar to that described in Section 2.5, the three different methods (two-step, three-step and proximity dependent) were tested against the same set of 500 randomly generated simulations. The results were analyzed using the evaluation metric equations in Table 2.1 and are displayed in Table 3.3.

Table 3.3 Evaluation Metrics for Three Methods

	Obstacle Proximity $\left(\frac{1}{m^2}\right)$	Distance Traveled (m)	Velocity Change $\left(\frac{m}{s}\right)$	Time (s)	Collisions	Time-outs
Two-Step	18.29	25.08	4.89	30.69	7 (1.4%)	0
Three-Step	5.73	25.47	5.80	37.19	2 (0.4%)	2
Proximity Dependent	5.74	25.35	6.23	33.71	5 (1.0%)	0

In general, the performance of the proximity dependent method was between that of the two- and three-step method. The proximity dependent method suffers from fewer collisions and maintains a larger distance between the robot and the nearest obstacle than the two-step method (note: the four evaluation metrics do not include data from scenarios that failed due to a collision or a time-out). However, the proximity dependent method does take a longer time to reach the goal with more acceleration (on average) than the two-step method. In contrast, the proximity dependent method is able to reach the goal more quickly and with fewer time-outs than the three-step method while maintaining a similar distance between the robot and obstacles, though the proximity dependent method does suffer from more collisions.

The one evaluation metric that the proximity dependent method performed consistently poorly on was the amount of acceleration. The increase in acceleration is due to the switching of methods since, when the method is switched, the feasibility of many potential robot velocities changes and often velocities that require only a small acceleration or deceleration are no longer available. However, given the general improvement over both component methods, the proximity dependent method is a good choice when both safety and efficiency need to be considered for robot navigation.

3.4. Conclusions about VOS for Differential Drive Robots

In this chapter, VOS has been augmented so that it can be used by a differential drive robot that is not capable of instantaneous change in velocity. Two basic methods have been derived and simulated. The first is a two-step velocity approximation method that provides the algorithm with a wide range of velocities to select from. But, while the robot does end up moving at the correct velocity, the robot does not move to the correct position by the end of the time step. The second is a three-step velocity approximation method which is more computationally complex and greatly reduces the selectable set of robot velocities, but causes the robot to end up at both the correct position and velocity.

The combination of the two- and three-step methods into a proximity dependent method allows the complementary strengths of both methods to be utilized: the two-step method allows for faster navigation (when the robot is not avoiding dangerously close obstacles) and the three-step method allows for slower, but predictable and safe, obstacle avoidance. The proximity dependent method has been shown to be almost as safe as the three-step method and almost as fast as the two-step.

Chapter 4

Velocity Occupancy Space for Vehicles with Actuation Error

The derivation of VOS for differential drive vehicles (Chapter 3) was developed in order to allow an experimental vehicle to be controlled using the VOS algorithm. One of the main assumptions made in that derivation was that the experimental vehicle in question would exhibit relatively repeatable acceleration responses when given the same acceleration command. However, the two vehicles that were available for experimentation were an iRobot PackBot and a SuperDroid ATR – both of which are controlled via linear and angular velocity commands. Therefore substantial testing was performed in order to determine what accelerations would be produced for a specific velocity (or changes in velocity) command.

Both available vehicles exhibited a significant amount of variation in the acceleration response for a given velocity command (probably due, at least in part, to the motor controllers). This variation is primarily due to errors in the vehicles' internal control system in attempting to follow a velocity command. Since unmanned ground vehicles (UGVs) are typically tele-operated, rather than operated autonomously, the operator provides another level of feedback (e.g. using a joystick). When operated autonomously, significant errors were observed in the UGVs response to motion commands. However, the two vehicles in question are both commonly used platforms and, thus, such actuation errors are typical of UGVs. This realization led to an additional research extension to VOS, where such actuation (in addition to sensing) uncertainties are accounted for.

As mentioned in the previous chapter, even though VOS is a velocity based navigation method, it is not possible to control a velocity commanded vehicle with VOS unless the vehicle is capable of instantaneous velocity changes (a capability which is decidedly rare among vehicles having mass). To compensate for this shortcoming, VOS was extended to apply to velocity-commanded vehicles that suffer from actuation error.

While the actuation error that this extension is primarily designed for is delayed velocity change, the method described in this chapter is capable of compensating for multiple types of actuation error, including slip (either due to the kinematics of the vehicle or to the terrain), poor motor control or a (well-known, e.g. one motor has partially lost power) malfunction of the vehicle: this method can be used as long as the effects of the actuation error can be bounded.

4.1. Summary of Error Causes and Effects

4.1.1. PackBot Specific Error Profiles

The first vehicle that was tested was the iRobot PackBot. The PackBot is a skid-steered, tracked vehicle that is typically tele-operated by a human. However, for this research, it was operated autonomously and controlled via linear and angular velocity commands produced by an onboard computer running the VOS algorithm. The PackBot is able to receive commands and return packets of information (including encoder positions, encoder-based velocity measurements, etc.) at a rate of 10Hz.

In order to convert desired acceleration responses into acceptable velocity commands, the PackBot was initially given a range of different velocity commands, and the resulting acceleration profiles were recorded. The purpose of this calibration was to produce a look-up table that would allow the algorithm to translate desired vehicle accelerations into actual vehicle velocity commands. To use the VOS algorithm, the vehicle must be able to accept and follow a new VOS-generated velocity every motion time step - which should typically be no longer than one second (though, much shorter response times would lead to better performance). Because the acceleration based VOS method requires the vehicle to change its acceleration up to three times during each motion time step, in order to replicate each VOS generated velocity, the vehicle accelerations were averaged over the first one-third of a second after the vehicle started to respond to the vehicle velocity command.

To clarify, two different velocity commands are being discussed. The first is the VOS generated velocity command (generated by the process described in Chapter 2) that will allow the vehicle to avoid obstacles and navigate towards a goal. This VOS generated velocity command can be broken into two or three acceleration commands (as described in Chapter 3) that allow an acceleration controlled vehicle to approximate an instantaneous velocity change. The second type of velocity commands are individual vehicle velocity reference commands to the UGV controller which are being analyzed to determine what accelerations they will generate in a given vehicle.

However, it was found that the Packbot did not repeatably generate the same accelerations for a given velocity reference command to the UGV. Specifically, for linear and rotational velocity commands of $1\frac{m}{s}$ and $1\frac{rad}{s}$, there was up to a 56% difference in the encoder-measured track acceleration averaged over the first one-third of a second after the vehicle had started to respond to the command. In addition, the tracks (due to the skid-steered configuration) were also subject to a good deal of slip which made the difference in average accelerations as measured by an IMU (accelerometers and gyroscopes mounted on the PackBot) as much as 186%.

An additional difficulty that was uncovered with the PackBot was a time-delay in the robot's response to a reference command. Due to the non-real-time PackBot operating system, the robot would unpredictably take between 0.1s and 0.4s to start to respond to a command (after it had been confirmed that the command had been received), see Figure 4.1. Given that it was necessary to update the velocity command every third of a second, this variable delay prevented the PackBot from being an acceptable platform for use with the acceleration based method.

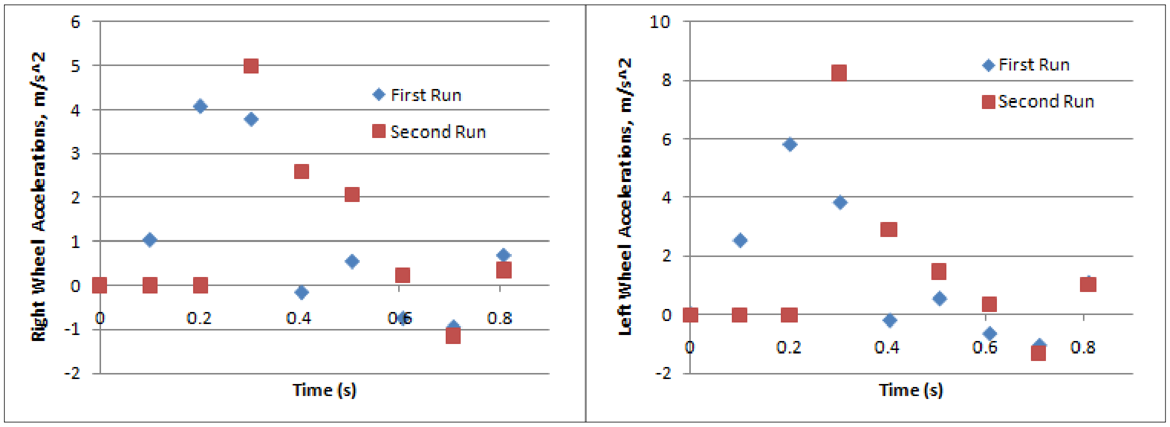


Figure 4.1. Two representative PackBot wheel acceleration responses for linear and angular velocity commands of $1\frac{m}{s}$ and $1\frac{rad}{s}$

4.1.2. SuperDroid Specific Error Profiles

The second vehicle that was considered for experimental testing was a SuperDroid with two driven, heavily treaded front wheels and one omni-directional back wheel (see Figure 4.2). Similar to the PackBot, the SuperDroid receives linear and rotational velocity commands that it translates into individual motor commands at a rate of 10Hz. However, the SuperDroid was able to return encoder and gyro information at a rate of 50Hz, significantly faster than the Packbot. Unfortunately, the SuperDroid was not able to be equipped with an independent accelerometer (vibration issues caused too much error in the accelerometer's reading for the data to be of any use) so independent acceleration information was not available.

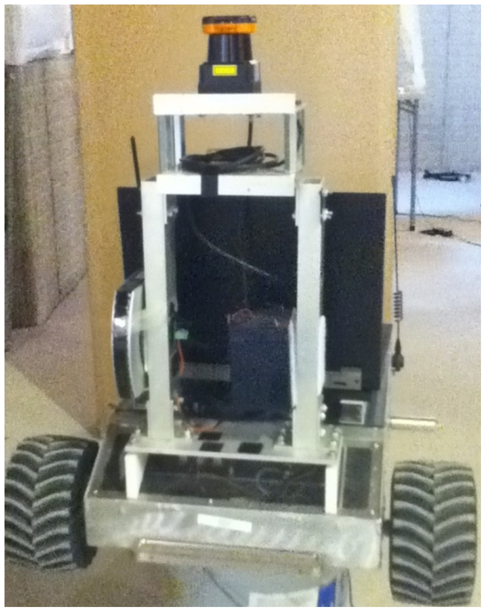


Figure 4.2 SuperDroid Robot

The SuperDroid exhibited less variation in the acceleration response to a velocity command (33% difference for linear and rotational velocity commands of $1\frac{m}{s}$ and $1\frac{rad}{s}$) probably due to the dual wheel configuration instead of the track configuration. While this variation was significantly better than the PackBot, it was still not sufficient for use directly with the acceleration based VOS method, except at low speeds.

4.2. Effect of Error on Velocity Obstacles

As neither of the available robotic platforms was suitable for use directly with the acceleration based VOS method, a new extension of VOS was developed in order to control a linear and rotational velocity commanded vehicle with a degree of actuation error (both from non-instantaneous velocity changes and vehicle slip) similar to that seen with the PackBot and the SuperDroid. With regards to accurately building velocity occupancy space the actuation error led to several difficulties in ensuring a safe and effective robot velocity selection.

4.2.1. Positional Error

The first type of error was in the robot's position. The VOS algorithm operates in two parallel loops. The first loop acquires sensor data, builds configuration space, tracks the obstacles and estimates their velocities. The second loop uses the obstacle and robot information in order to build velocity occupancy space. By necessity, the second loop must project the locations of the obstacles and robot ahead by one motion time step as it is building velocity occupancy space while the robot is in the process of carrying out the last velocity command. Because the actuation error causes the robot's actual position at the end of that time step to be uncertain, the $\vec{\lambda}_{i,r}\left(\dot{x}(t_m), \dot{y}(t_m)\right)$ term used to compute the velocity obstacles may be incorrect and therefore the velocity obstacle cannot be accurately constructed.

For example, in Figure 4.3, the algorithm will assume, based on the calculated lambda, that $\vec{v}_r$ will lead to a collision (and therefore that velocity will be avoided). However, if ${\vec{\lambda}_{i,r}}^{A}$ is the actual distance then ${\vec{v}_r}^{A}$, and not $\vec{v}_r$, will lead to a collision. In this scenario, the algorithm may mistakenly choose an unsafe robot velocity due to the error in the robot's position.

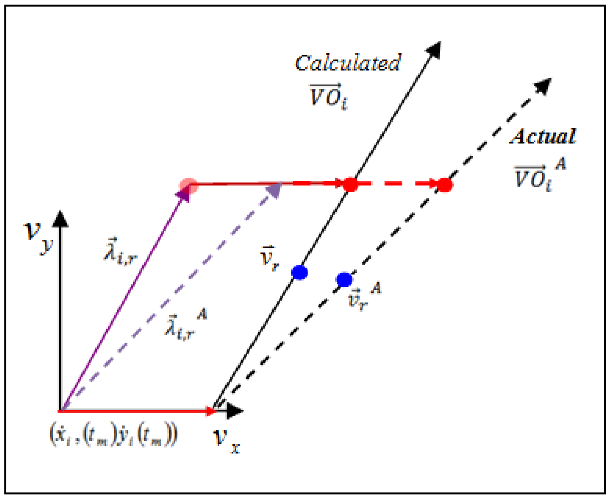


Figure 4.3. Velocity obstacles based on the calculated robot position and the actual (error influenced) robot position

To compensate for the positional error, the maximum error in the average robot linear and angular velocities were found over the course of one motion time step. The error for the SuperDroid was due primarily to the non-instantaneous velocity change and therefore the pervious and current velocity commands where used as the basis for calculating the positional error. The dependence of the actual, measured velocity on the current and previously commanded velocities was found from the SuperDroid test data using the equation

$$\varepsilon_v(t_{m-1}) = \frac{\bar{v}_{m-1} - v_{m-2}}{v_{m-1} - v_{m-2}} \tag{4.1}$$

where $v_{m-1}$is the current linear velocity *command* and $\bar{v}_{m-1}$is the average of the *measured* velocities over one motion time step. It should be remembered that the VOS algorithm is computing v_m while executing v_{m-1}, so v_{m-2} is the previously commanded linear velocity. The maximum linear velocity error for a specific motion time step can therefore be calculated using the equation

$$e_v(t_{m-1}) = \left(1 - \varepsilon_v(t_{m-1})\right) \cdot v_{m-2} + \varepsilon_v(t_{m-1}) \cdot v_{m-1} \tag{4.2}$$

The same two equations can also be used to find, ε_ω and e_ω, the maximum error for the angular velocity, by substituting ω for v .

Based on the test data from the SuperDroid it was found that $\varepsilon_v = 0.51$ and $\varepsilon_\omega = 0.44$. Using these values, $e_v(t_{m-1})$ and $e_\omega(t_{m-1})$ will bound the maximum error for 98% of velocity changes (based on the experimental test data). For most of the other 2% of velocity changes, the actual change was typically so slight, that the error was due more to the steady state variation in the average velocity than to error from acceleration and including these more extreme values would make ε_v and ε_ω unnecessarily conservative.

After the maximum velocity errors have been found they can be used to determine the maximum positional errors in the x- and y-directions using

$$e_x(t_m) = \int_0^{\Delta t_m} e_v(t_{m-1}) \cdot cos(e_\omega(t_{m-1}) \cdot \tau)\, d\tau - \int_0^{\Delta t_m} v(t_{m-1}) \cdot cos(\omega(t_{m-1}) \cdot \tau)\, d\tau \tag{4.3}$$

and

$$e_y(t_m) = \int_0^{\Delta t_m} e_v(t_{m-1}) \cdot sin(e_\omega(t_{m-1}) \cdot \tau)\, d\tau - \int_0^{\Delta t_m} v(t_{m-1}) \cdot sin(\omega(t_{m-1}) \cdot \tau) d\tau \tag{4.4}$$

where Δt_m is the length of one motion time step.

The maximum positional errors were then applied to the $\vec{\lambda}_{i,r}$ term in the direction and angle equations in order to ensure that the actual robot position at the end of the current time step would be considered. The directional equation (see Eq. (2.8)) was augmented using the calculated $e_y(t_m)$ and $e_x(t_m)$ in the following manner

$$D_R = \begin{cases} 1\ if \left(\dfrac{\vec{\lambda}_{i,r}\big(\dot{y}(t_m)\big) \pm \dfrac{e_y(t_m)}{\Delta t_m}}{\dot{y}_r(t_m) - \dot{y}_i(t_m) \cdot (1 \pm V_U)} \wedge \dfrac{\vec{\lambda}_{i,r}\big(\dot{x}(t_m)\big) \pm \dfrac{e_x(t_m)}{\Delta t_m}}{\dot{x}_r(t_m) - \dot{x}_i(t_m) \cdot (1 \pm V_U)} \right) \geq 0 \\ 0 \quad otherwise \end{cases}. \qquad (4.5)$$

The $\vec{\lambda}_{i,r}$ term in the angular equation (see Eq. (2.10)) was likewise increased,

$$A_R = \begin{cases} 1\ \ if\ tan^{-1}\left(\dfrac{\vec{\lambda}_{i,r}\big(\dot{y}_i(t_m)\big) \pm \frac{e_y(t_m)}{\Delta t_m}}{\vec{\lambda}_{i,r}\big(\dot{x}_i(t_m)\big) \pm \frac{e_x(t_m)}{\Delta t_m}} \right) = tan^{-1}\left(\dfrac{|\dot{y}_r(t_m) - \dot{y}_i(t_m) \cdot (1 \pm V_U)| \pm P_A}{|\dot{x}_r(t_m) - \dot{x}_i(t_m) \cdot (1 \pm V_U)| \pm P_A} \right) \cdot \big(1 \pm (W_{AR} - 1)\big) \\ 0 \qquad otherwise \end{cases} \qquad (4.6)$$

Please see Chapter 2 for definitions and a detailed description of the other variables in these equations.

Using the modified angle and direction equations, an expanded velocity obstacle was formed that took into account possible error in the robot's position at the end of the time step. As shown in Figure 4.4, the velocity obstacle has been expanded to include additional velocities that may lead to a collision, based on the uncertainty in the robot's position.

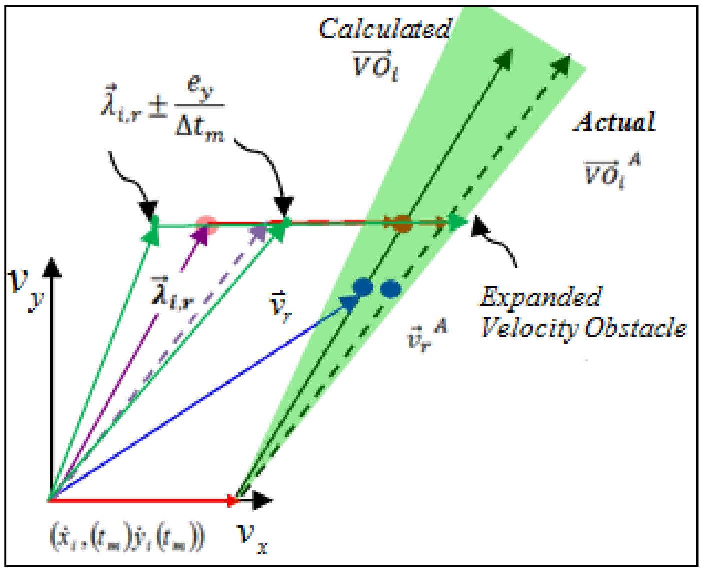

Calculated $\overrightarrow{VO_i}$
$\vec{\lambda}_{i,r} \pm \frac{e_y}{\Delta t_m}$
Actual $\overrightarrow{VO_i}^A$
v_y
$\vec{\lambda}_{i,r}$
$\vec{v}_r$
$\vec{v}_r^A$
Expanded Velocity Obstacle
$(\dot{x}_i(t_m), \dot{y}_i(t_m))$
v_x

Figure 4.4 Expanded velocity obstacle using robot's positional error bounds

This method may seem overly conservative as it expands the velocity obstacle in both directions – instead of only in the direction corresponding to the change in the robot's velocity. However, it was found that the experimental robot would sporadically over- or undershoot the command velocity by accelerating or decelerating too rapidly. By the end of the time step, the robot's velocity had settled to the command velocity, but the positional error was now the opposite of what was expected (based on Eq. (4.1-4.4)). This error occurred frequently enough that it was considered prudent to allow the velocity obstacle to be expanded in both directions as a conservative velocity choice around obstacles was considered preferable to a possible collision.

4.2.2. Velocity Error

The second type of error was in the robot's selected velocity. Each velocity obstacle is composed of all of the robot velocities that will lead to a collision between the robot and a specific obstacle – velocities which must be avoided. However, given the actuation error, the robot's actual velocity frequently differed from the command velocity selected by the algorithm. Therefore the robot might inadvertently move at a dangerous velocity. For example, in Figure 4.5, the algorithm has selected $\vec{v}_r$ as a safe velocity at which the robot may operate, however, the actuation error causes the robot to actually move at $\vec{v}_r{}^A$, which is along a velocity obstacle and places the robot at risk of a collision. In this scenario, the algorithm correctly selects a safe velocity, but the robot is unable to obey the commanded velocity.

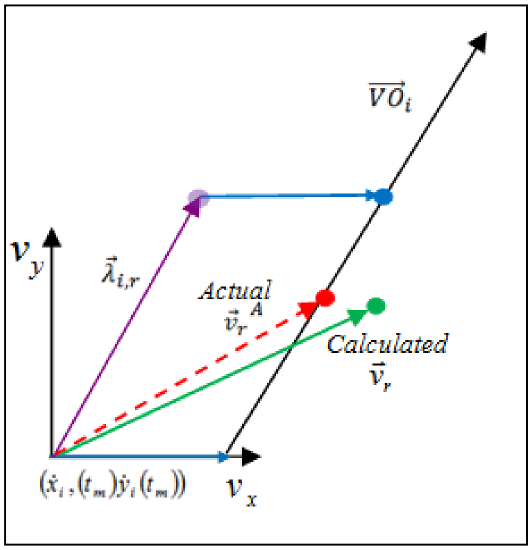


Figure 4.5. Scenario where a possible collision may occur due to the robot's current velocity error. Please note, for simplicity, this figure does not show the positional error from Section 4.2.1.

To compensate for the error in the robot's current velocity, the maximum variations in the robot's linear and angular velocity, Eqs. (4.2 and 4.3), were again used. However, this time, the error terms were found using the currently commanded linear and angular velocities (v_{m-1} and ω_{m-1}) and the linear and angular velocity commands that the algorithm was currently selecting using VOS (v_m and ω_m). Therefore the error dependency equation is slightly altered from Eq. (4.1) to

$$\varepsilon_v(t_m) = \frac{\bar{v}_m - v_{m-1}}{v_m - v_{m-1}} \tag{4.7}$$

and the maximum linear velocity error is likewise slightly altered from Eq. (4.2) to be

$$e_v(t_m) = \big(1 - \varepsilon_v(t_m)\big) \cdot v_{m-1} + \varepsilon_v(t_m) \cdot v_m \tag{4.8}$$

Again, the angular velocity error terms, ε_ω and e_ω, can also be found by substituting ω for v.

The upper bounds on the directional velocity errors are

$$e_{vx}(t_m) = e_v(t_m) \cdot cos(e_\omega(t_m) \cdot \Delta t_m) \tag{4.9}$$

and

$$e_{vy}(t_m) = e_v(t_m) \cdot sin(e_\omega(t_m) \cdot \Delta t_m). \tag{4.10}$$

These terms were also used to augment the direction and angle equations (see Eqs. 3.5 and 3.7), but this time they increase the range of the robot's selected velocity. The direction and angle terms, with both the position error terms from Section 4.2.1 and the velocity error terms from this section are

$$D_R = \begin{cases} 1\, if \left(\dfrac{\vec{\lambda}_{i,r}(\dot{y}) \pm \dfrac{e_y}{\Delta t_m}}{\left(\dot{y}_r \pm e_{vy}\right) - \dot{y}_i \cdot (1 \pm V_U)} \wedge \dfrac{\vec{\lambda}_{i,r}(\dot{x}) \pm \dfrac{e_x}{\Delta t_m}}{\left(\dot{x}_r \pm e_{vx}\right) - \dot{x}_i \cdot (1 \pm V_U)} \right) \geq 0 \\ 0 \quad othewise \end{cases} \tag{4.11}$$

and

$$A_R = \begin{cases} 1 \quad if\ tan^{-1}\left(\dfrac{\vec{\lambda}_{i,r}(\dot{y}_i) \pm \dfrac{e_y}{\Delta t_m}}{\vec{\lambda}_{i,r}(\dot{x}_i) \pm \dfrac{e_x}{\Delta t_m}}\right) = tan^{-1}\left(\dfrac{\left|(\dot{y}_r \pm e_{vy}) - \dot{y}_i \cdot (1 \pm V_U)\right| \pm P_A}{\left|(\dot{x}_r \pm e_{vx}) - \dot{x}_i \cdot (1 \pm V_U)\right| \pm P_A}\right) \cdot \left(1 \pm (W_{AR} - 1)\right) \\ 0 \qquad otherwise \end{cases} \quad (4.12)$$

where all time dependent variables are from time t_m (t_m has been removed for brevity).

Using the expanded angle and direction terms, the velocity obstacles will now take into account both the error in the robot's position as well as the potential for error in the velocity that is selected for each motion time step.

An example of an actual extended velocity obstacle can be seen in the following figures. Figures 4.7 and 4.8 show the velocity obstacle for the simple scenario in Figure 4.6. The velocity obstacle is notably broader, especially at velocities that are further away from the robot's current velocity (which thus may produce more error). It should be noted that the magnitude of the repulsive values assigned to velocities in the obstacles do not increase, just the number of velocities that are considered repulsive.

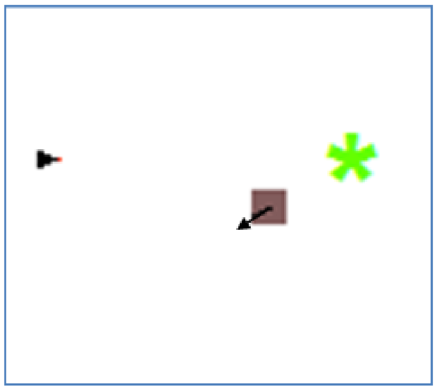


Figure 4.6 Simple scenario with the robot (triangle), goal (asterisk) and one moving obstacle (square)

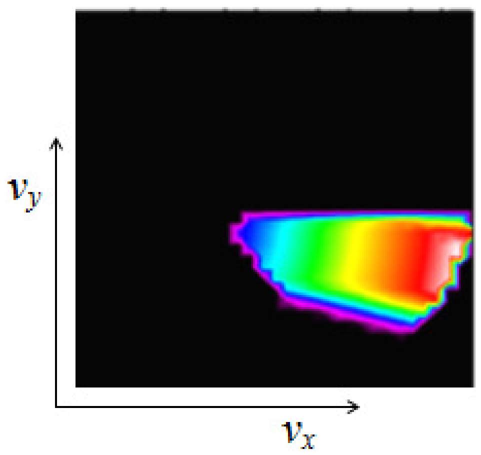


Figure 4.7. Velocity obstacle without considering actuation error

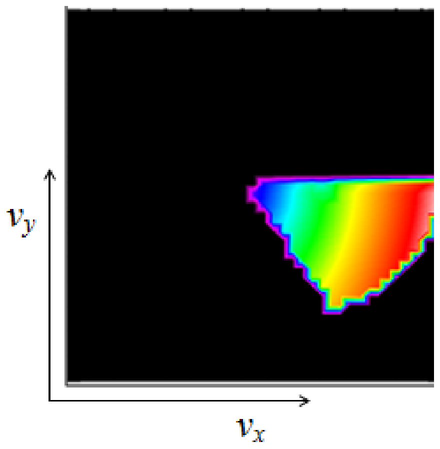


Figure 4.8. Velocity obstacle with actuation error extensions to VOS

4.3. Simulation Results

4.3.1. Actuation Error Simulation Results

An environment of approximately the same size and shape as the experimental testing area was simulated in order to exhaustively test the algorithm and analyze its performance with and without the actuation error extension. In order to cause the simulated robot to perform comparably to the experimental robot, error was added to the simulated robot's position, following the same trend as was observed with the experimental robot, using the equations

$$x(t_m) = x(t_{m-1}) +$$
$$\left(v(t_m) + e_v(t_m) \cdot RN(0-1.1)\right) \cdot cos\left(\left(\omega(t_m) + e_\omega(t_m) \cdot RN(0-1.1)\right) \cdot \Delta t_m\right) \cdot \Delta t_m \quad (4.13)$$

and

$$y(t_m) = y(t_{m-1}) +$$

$$\left(v(t_m) + e_v(t_m) \cdot RN(0 - 1.1)\right) \cdot sin\left(\left(\omega(t_m) + e_\omega(t_m) \cdot RN(0 - 1.1)\right) \cdot \Delta t_m\right) \cdot \Delta t_m. \quad (4.14)$$

where the maximum linear and angular velocity errors, $e_v(t_m)$ and $e_\omega(t_m)$, were calculated according to Eq.(4.2) (using the current and previous velocity commands) and $RN(0 - 1.1)$ is a random number between 0 and 1.1. This random number was obtained using a uniform distribution and was used for two reasons. First, the observed velocity error was relatively evenly distributed between zero and the maximum velocity error bound so the random number reproduced the actual robot behavior fairly accurately. Second, an upper bound of 1.1 was used instead of 1.0 both to represent additional sources of error that had not been quantified (such as inexact timing of robot commands) and error from wheel slip that was not captured by the gyroscope and encoder data used to calculate the experimental robot's velocity error.

A total of six sets of 100 simulations each were performed, with different velocity limits on the robot and the obstacles. Each simulation consisted of six randomly generated obstacles (in addition to the walls of the testing environment), with between one and four obstacles - moving at constant velocities randomly generated between the bounds shown in Table 4.1. The robot's maximum linear velocity and accelerations for the simulations are also shown in this table (the maximum angular velocity was always $1\frac{rad}{s}$).

Table 4.1 Simulation Specifications

Simulation Set #	Actuation Error Extension	Robot Maximums		Obstacles
		Linear Velocity	Linear Acceleration	Velocity Range
1	Yes	$0.5\frac{m}{s}$	$0.5\frac{m}{s^2}$	$-0.3\ to\ 0.3\frac{m}{s}$
2	No	$0.5\frac{m}{s}$	$0.5\frac{m}{s^2}$	$-0.3\ to\ 0.3\frac{m}{s}$
3	Yes	$0.7\frac{m}{s}$	$0.7\frac{m}{s^2}$	$-0.4\ to\ 0.4\frac{m}{s}$
4	No	$0.7\frac{m}{s}$	$0.7\frac{m}{s^2}$	$-0.4\ to\ 0.4\frac{m}{s}$
5	Yes	$1.0\frac{m}{s}$	$1.0\frac{m}{s^2}$	$-0.5\ to\ 0.5\frac{m}{s}$
6	No	$1.0\frac{m}{s}$	$1.0\frac{m}{s^2}$	$-0.5\ to\ 0.5\frac{m}{s}$

Figures 4.9-4.11 show the algorithms response with and without the actuation error extension to the same simulation.

In Figure 4.9, the algorithm has not increased the velocity obstacles based on the actuation error derivation, but the robot's velocity response still suffers from the actuation error. The algorithm starts out moving the robot quickly towards the goal. However, at time step 7, the algorithm unsuccessfully attempts to evade an obstacle by selecting a velocity that is up and to the left (relative to the image's orientation). Actuation error causes the robot not to respond in the desired manner and the robot collides with the obstacle at time step 8.

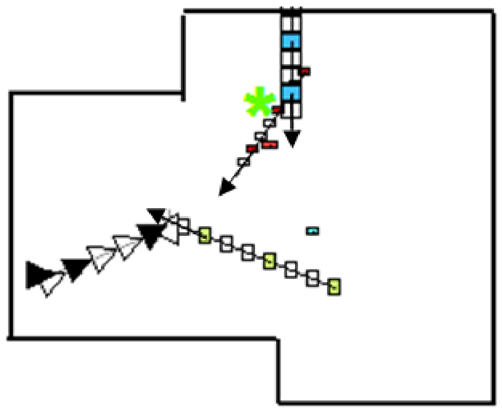


Figure 4.9. Algorithm's response to a basic simulation without actuation error extension

In Figures 4.10 and 4.11 the actuation error extension was applied to the velocity obstacles. In Figure 4.10, the robot does not initially move as quickly towards the goal as in the previous simulation, as the extended velocity obstacles react to the presence of approaching obstacles and cause more of the faster robot velocities to have a repulsive value. However, as an obstacle gets closer the algorithm selects more conservative velocities and waits for the obstacle to pass, instead of attempting to pass in front of the obstacle. In Figure 4.11, once the obstacle is no longer threatening the robot, the algorithm selects faster velocities as it successfully directs the robot to the goal.

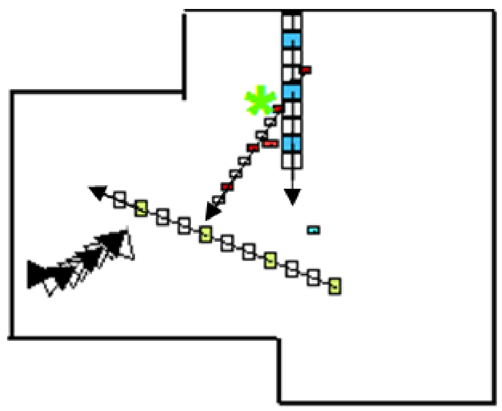


Figure 4.10. Algorithm's response with actuation error extension, time steps 1-11

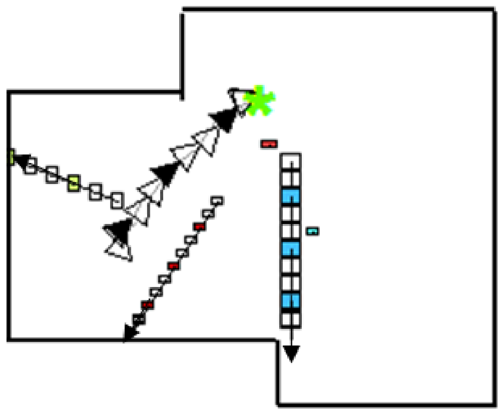


Figure 4.11. Algorithm's response with actuation error extension, time steps 11-20

In Table 4.2, the values of the evaluation metrics for the simulations are tabulated. The evaluation metrics were calculated using the equations from Table 2.1 of Chapter 2. However, the obstacle proximity and the acceleration were divided by the number of motion time steps in each simulation so that the results could be more accurately compared between simulation and experimental scenarios with different goal locations.

Table 4.2 Simulation Evaluation Metric Values for 100 Trials

Simulation Set #	Evaluation Metrics					
	Obstacle Proximity $\left(\frac{1}{m^2}\right)$	Distance Traveled (m)	Velocity Change $\left(\frac{m}{s}\right)$	Time (s)	# of Collisions/ % of Collisions	# of Timeouts/ % of Timeouts
1	3.15	8.26	0.082	29.52	3 / 3%	3 / 3%
2	2.50	8.28	0.080	28.81	5 / 5%	2 / 2%
3	1.62	8.79	0.285	31.95	4 / 4%	0
4	1.80	8.93	0.345	27.46	7 / 7%	0
5	1.41	8.92	0.173	26.09	3 / 3%	0
6	2.13	9.13	0.292	15.31	11 / 11%	1 / 1%

As summarized in Table 4.2, the difference between the algorithm's performance with and without the actuation error extension diminished as the robot's and obstacle's maximum velocity decreased. For simulations sets 5 and 6 (where the obstacles and robot had the highest velocities), the actuation error extension made a statistically significant improvement in the number of collisions ($p < 0.011$, on a two-tailed, paired *t*-test) and in the robot's acceleration ($p < 0.005$) between the two sets. However, the actuation error extension also significantly increased the average amount of time that it took the robot to reach the goal ($p < 0.005$), probably due to the more conservative velocity selection.

For simulation sets 4 and 5, there was a smaller difference in the number of collisions ($p < 0.181$) and the statistical difference between the sets for time and acceleration were the same as for sets 5 and 6. Finally, for simulation sets 1 and 2, there was not a statistically significant difference in the number of collisions. However, the simulations with the actuation error extension experienced more timeouts (being unable to reach the goal within 100 motion time steps) than the simulations without the extension. Two of these timeouts occurred in the simulation where the extension-less simulation experienced a collision. While, from a safety perspective, timing out is preferable to a collision, it still means that the robot failed to reach the goal.

The actuation error extensions significantly improved the robot's ability to avoid collisions at higher speeds, but had more negligible effect at lower speeds. At higher speeds the robot accumulated more positional error within one time step and the possible difference in velocities was also higher (leading to additional error). However, while the robot was more careful in its avoidance of obstacles it also selected more conservative velocities which resulted in the robot to taking a longer time to reach the goal.

For lower speeds, there was less possible position and velocity error and while Eq. (4.1) takes this into account, the normal precautions that the traditional VOS algorithm takes to compensate for sensor error appeared to be sufficient to keep the robot safe even with additional actuation error. The more conservative velocity selection that was produced using the actuation error extension was not necessary and served only to increase the time needed for the robot to reach the goal.

4.3.2. Evaluation of VOS in Scenarios with Narrow Passageways

An important capability of any navigation algorithm is the ability to safely direct a robot through a narrow opening, such as a doorway or between two closely spaced obstacles, if this is the most efficient route to the goal. The VOS algorithm was tested in simulation to determine the narrowest gap (relative to the robot's size, *n*) through which it would reliably pass. The original VOS method (developed in Chapter 2), as well as the proximity dependent, acceleration based VOS method for differentially driven vehicles (developed in Chapter 3) and VOS for vehicles with actuation error (developed in this chapter) were all tested in the narrow passageway scenarios.

Two specific scenarios were evaluated; the first (Case 1), occurred when the robot was required to pass between two relatively small obstacles in order to reach the goal as quickly as possible (Figure 4.12). The second scenario (Case 2) occurred when the robot needed to pass between two significantly larger obstacles that completely blocked the robot from the goal; if the robot failed to pass between these obstacles it would not reach the goal (Figure 4.13).

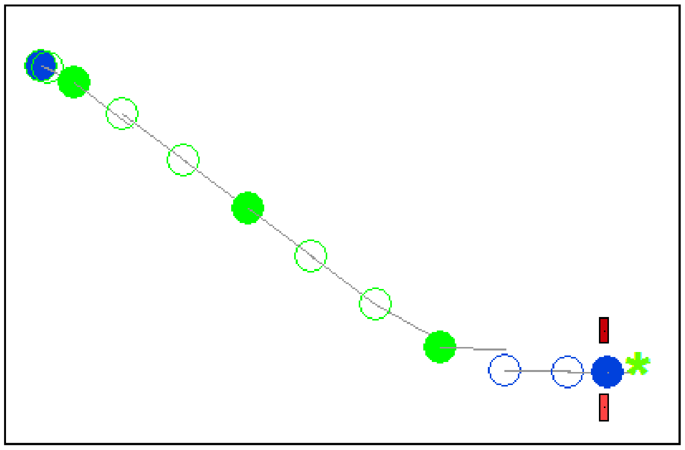


Figure 4.12. Robot, with differential drive constraints, using the proximity dependent VOS extension (Case 1). Distance between obstacles is 1.55*n*, where *n* is the diameter of the robot. When robot is green, the algorithm is using the two-step method and when it is blue, the three-step.

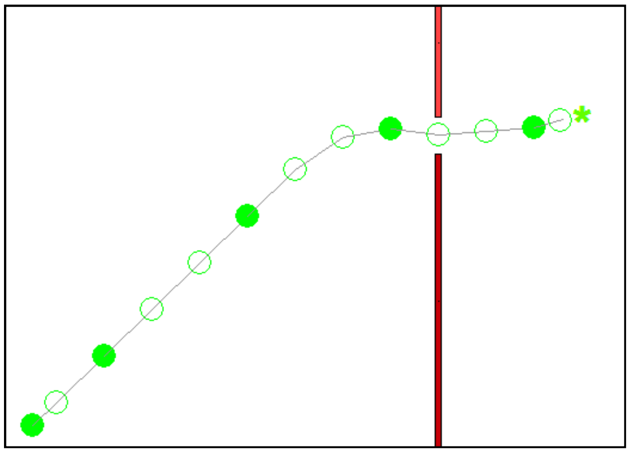


Figure 4.13. Holonomic robot navigating between two obstacles (Case 2). Distance between obstacles is 1.5*n*.

In Table 4.3, the minimum opening through which the robot was able to regularly pass (at least 95% of the time) is listed for both cases for the original VOS algorithm

(Holonomic) and the two extensions. The holonomic robot was the most successful at passing through narrow openings and, when necessitated by the Case 2 configuration, could regularly pass through openings of only 1.4*n.* Using the proximity dependent method the robot was also successful at passing between two obstacles spaced 1.6*n* apart when there was no other way to reach the goal, Case 2. However, for the Case 1 configuration, the robot would frequently take a less efficient route around the obstacles unless they were spaced 2*n* apart. The larger required spacing for Case 1 was mostly due to the use of the two-step method. If the robot did not approach the obstacles/goal from a favorable angle, it would need to back away from the opening in order to reorient itself. During this process the robot would sometimes overshoot its desired position (due to the inexact positioning of the two-step method) and end up in a location far enough from the opening that it would circle around the obstacles instead of passing between them.

Finally, when the robot was operating with the actuation error extension, for Case 1 it was able to pass reliably between the obstacles when they were spaced 1.6*n* apart and for Case 2 between obstacles spaced 1.55*n* apart. The unexpected variations in velocity caused by the (simulated) robot actuation error sometimes led to a collision between the robot and the blocking obstacles – which is why the robot with actuation error required a larger opening than the ideal, holonomic robot.

Table 4.3. Necessary spacing to allow for robot to pass between obstacles (at least 95% of the time)

	Holonomic	Proximity Dependent	Actuation Error
Case 1	1.5*n*	2*n*	1.6*n*
Case 2	1.4*n*	1.6*n*	1.55*n*

Figure 4.14 shows the minimum opening through which the robot was able to successfully pass 95% of the time for Case 2 scenarios. The holonomic robot was the most successful at passing through narrow openings, but the performance of all three of the algorithms dropped off quickly when the opening was less than 1.35*n*.

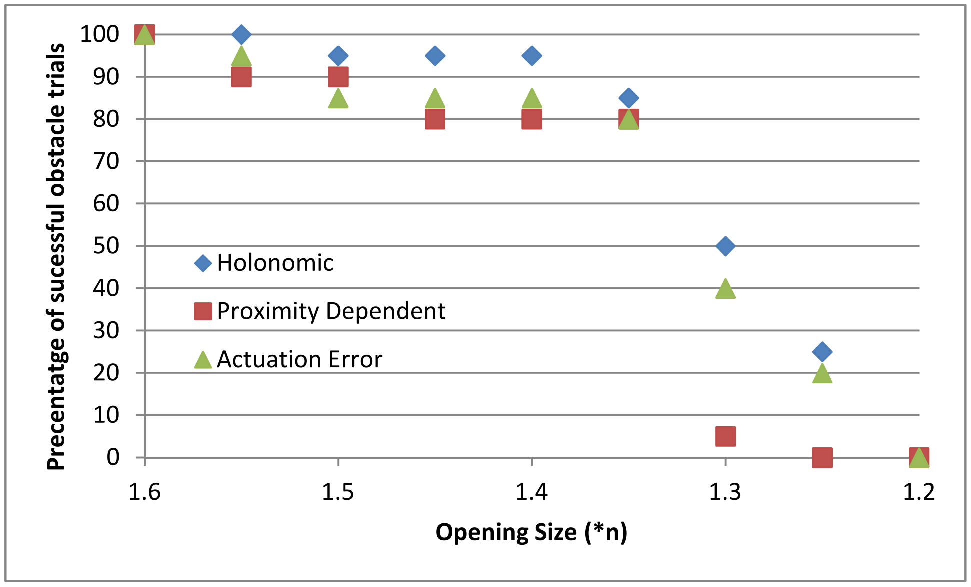


Figure. 4.14 Minimum distance between obstacles required to allow the robot to pass between for Case 2

The results for Case 1 were very similar to those shown in Figure 4.14, with the exception of the proximity dependent method. The proximity dependent method required an opening of 2*n* in order to reliably pass between two obstacles for Case 1 and was only able to pass through an opening of 1.6*n* (as it did reliably for Case 2) 60% of the time – the remainder of the time the robot tended to circumnavigate the obstacles to reach the goal.

4.4. Experimental Results with SuperDroid

The SuperDroid, shown labeled in Figure 4.15, was used for the experimental testing of VOS. As previously mentioned, it had a motor command update rate of 10Hz and reported gyroscope and encoder data at 50Hz. It was equipped with a Hokyuo UTM 30-LX laser range finder (see Appendix B for additional hardware specifications for the robot, computer, the Create robots and laser range finder). The robot received linear and angular velocity commands and ran a simple control loop (using the encoders) to produce motor commands. This loop had a settling time of about one second, which dictated the rate at which VOS could send velocity commands to the SuperDroid. The SuperDroid was also equipped with a Dell Latitude E6400 laptop running VOS in NI's LabVIEW.

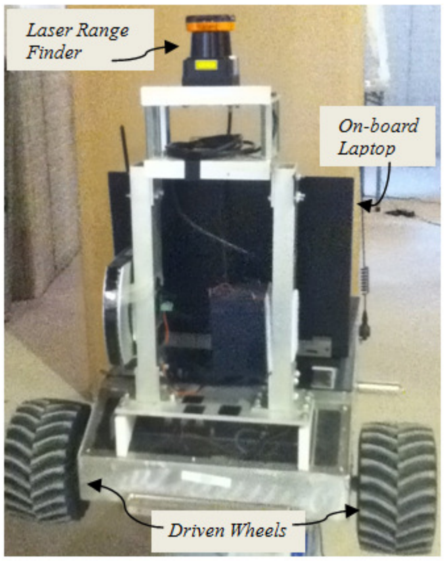


Figure 4.15. Labeled SuperDroid Robot

Three iRobot Create robots were used as the moving obstacles. The Create robots are capable of linear speeds of up to $0.5\,\frac{m}{s}$. Cardboard tubes were used to increase the height of the Create robots so that they would be detectible by the laser range finder.

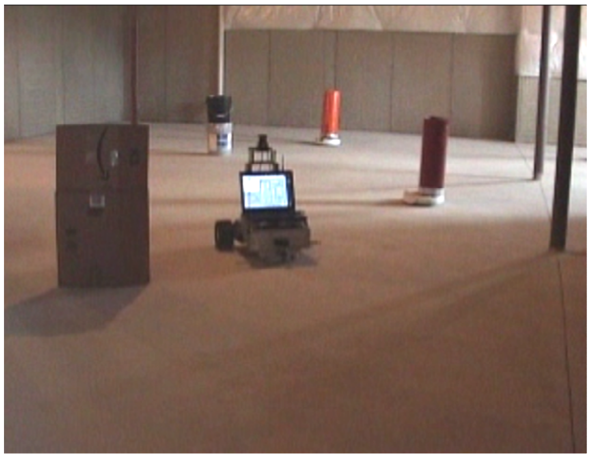


Figure 4.16. SuperDroid and Create Obstacles in experimental testing environment

Over one hundred different experimental scenarios were tested using the SuperDroid and Create robots. It proved to be extremely difficult to exactly recreate specific scenarios in order to produce comparable or statistically significant results so that the effect of the actuation error extension could be clearly demonstrated. This was due to a number of factors. First, the Create robots did not generally adhere to a constant velocity and also had variable lag time in responding to a command. Second, it was difficult to position the robot at exactly the same location and orientation for each test, and even a slight variation was enough to affect the respective locations of observed obstacles and therefore alter the robot's performance. Finally, the robot navigated to the goal position based on dead reckoning with the encoders and gyroscope. The amount of error in the robot's perceived final goal position ranged from a few centimeters to over two meters (in one case), with typical errors in the final robot position of around half a meter.

In Figure 4.17, the robot is shown with three moving obstacles in one of the experimental scenarios. *Obstacles C, D* and *E* are moving at $0.3\frac{m}{s}$ towards the center of the testing area and the robot is capable of linear and angular speeds of up to $1.0\frac{m}{s}$ and $1.0\ \frac{rad}{s}$, respectively.

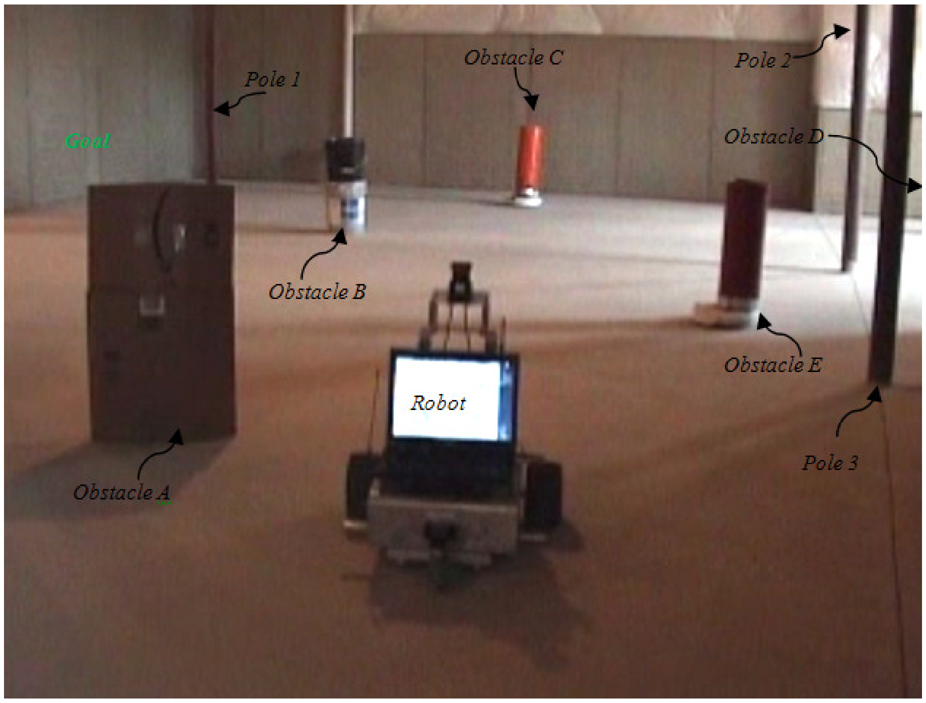


Figure 4.17. Robot and obstacles in testing environment
(Obstacle D is not visible in the image)

In Figure 4.18, the VOS display for Figure 4.17 is shown. The lower, right hand corner shows the laser range finder output (this is the entirety of the external sensor data with which the robot is provided). The upper, left hand corner shows the configuration space occupancy grid that is built from this data – obstacles are labeled to correspond with Figure 4.17. The unlabeled obstacles are either walls, or stationary obstacles that are not visible in Figure 4.17. The lower left and middle images are the obstacle-based (repulsive) and goal-based (attractive) velocity spaces that are built using the obstacle's locations and estimated velocities. Finally, the upper, right hand corner shows the combined velocity occupancy space from which the next robot velocity is selected. The combined occupancy space grid is shown in white color for velocities that are not dynamically feasible for the robot (and therefore, cannot be selected).

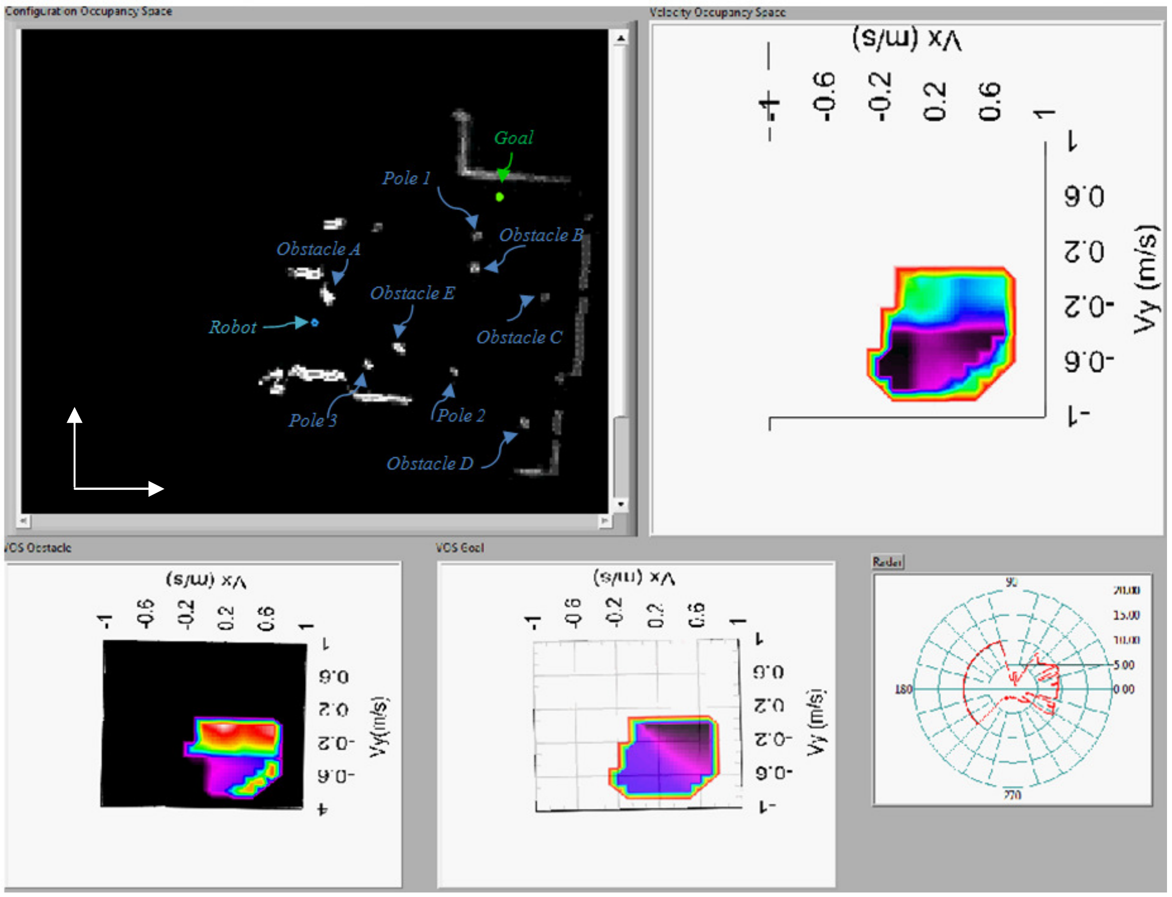


Figure 4.18. VOS Display of the scenario in Figure 4.17

In the obstacle-based velocity space, the most repulsive velocities are shown in red. Velocities that will move the robot towards *Obstacle A* (the closest obstacle) are shown to be highly repulsive, and velocities that will move the robot towards the more distant moving *Obstacle E, Pole 3* or the wall also have relatively high repulsive values. In the goal-based velocity space, the most attractive values (in black) can be seen to move the robot directly towards the goal (as would be expected).

The robot was able to successfully avoid the moving and stationary obstacles and reach the goal location. The video results for this test and others can be seen in Appendix C.

In Table 4.4, the specifications for the experimental tests sets are shown. As with the simulations, the robot's maximum angular velocity was always $1.0\frac{rad}{s}$.

Table 4.4 Experimental Specifications

Experimental Set #	**Actuation Error Extension**	**Robot Maximums** Linear Velocity	**Robot Maximums** Linear Acceleration	**Obstacles** Velocity Range
1	Yes	$0.5\frac{m}{s}$	$0.5\frac{m}{s^2}$	$-0.35\ to\ 0.35\frac{m}{s}$
2	No	$0.5\frac{m}{s}$	$0.5\frac{m}{s^2}$	$-0.35\ to\ 0.35\frac{m}{s}$
3	Yes	$0.7\frac{m}{s}$	$0.7\frac{m}{s^2}$	$-0.40\ to\ 0.40\frac{m}{s}$
4	No	$0.7\frac{m}{s}$	$0.7\frac{m}{s^2}$	$-0.40\ to\ 0.40\frac{m}{s}$
5	Yes	$1.0\frac{m}{s}$	$1.0\frac{m}{s^2}$	$-0.50\ to\ 0.50\frac{m}{s}$

In Table 4.5, the results from the experiments are listed. The results from the experimental tests with the fastest robot and obstacle velocities, but without the actuation error extension (Set #6), proved to suffer from too many collisions to produce useful data, so the experimental results are not included here. Specific results for the various tests can be seen in Appendix D.

Table 4.5 Simulation Evaluation Metric Values

Experimental Set #	**Evaluation Metrics**			
	Obstacle Proximity $\left(\frac{1}{m^2}\right)$	Distance Traveled (m)	Velocity Change $\left(\frac{m}{s}\right)$	Time (s)
1	1.02	12.51	0.114	51.38
2	0.93	9.18	0.103	42.13
3	1.75	12.61	0.116	34.18
4	0.92	13.52	0.131	33.44
5	0.86	12.28	0.125	27.46

It is difficult to draw definitive conclusions from the experimental data, as the testing conditions unavoidably varied (sometimes substantially) between different tests. However, all of the obstacle proximity experimental result metrics fell within one-half of one standard deviation of the simulation results and acceleration experimental result metrics were within 1.5 standard deviations of the simulation results (the variation in the goal location makes the distance traveled and the time not meaningfully comparable). Therefore, as the performance of the VOS algorithm is statistically similar under experimental and simulation conditions, the statistical conclusions that were drawn from the (much larger) set of simulation results can be applied to the experimental results.

It was also not possible to produce an accurate failure rate for the experimental tests because a significant number of failures occurred due to conditions outside of the algorithms control. First, the SuperDroid was equipped with a remote emergency stop button which was used whenever it appeared that a collision might occur. It ended up being used much more liberally then was actually required in order to safeguard the robots. This was usually determined after the robot had been stopped and the data reviewed, as it was frequently the case the robot was aware of the obstacle and responding appropriately – and what was thought to be a threatened collision was due more to the angle and distance from which the robot had been viewed than to any actual danger.

Second, as can be seen in Figures 4.16 and 4.17, the cardboard tubes on the Create robots are significantly smaller then the Creates. As the algorithm is only able to sense the width of the tube (via the laser range finder) it would occasionally nick (or be stopped before it could hit) the end of a Create. Some thought was given to virtually increasing the perceived size of all of the obstacles in the configuration space by the difference in size between the Creates and the cardboard tubes. However, the testing environment frequently required the robot to navigate through somewhat close fitting areas, which this increased size would have severely limited the robot's ability to do. It was determined that the decrease in overall performance was not worth fixing a few specific cases.

Finally, the dead reckoning error that was accumulated throughout the test frequently moved the robot's intended goal to an unreachable location (such as within a wall). This made a number of tests impossible for the robot to successfully complete.

Out of the over one hundreds tests that were performed, if all of the cases where the there was a failure due to one of the scenarios previously mentioned (or, an all too common, hardware failure) were removed then there would only be a very small subset of tests available for analysis. In order to produce a larger set of test data so that more representative results could be derived, the data from as many (even partially successful) tests as was possible was included in the experimental test evaluation and the calculation of the evaluation metrics.

4.5. Conclusions about VOS for Vehicles with Actuation Error

The actuation extension for VOS has been shown to significantly improve the algorithm's performance at higher speeds $\left(0.7\frac{m}{s^2} \text{ to } 1\ \frac{m}{s^2}\right)$ over the original VOS algorithm. However, this improvement comes at the cost of increasing the time that it will take the robot to reach its goal. As has been the theme throughout the development of VOS, there are always tradeoffs between performance and safety for the algorithm and as safety must be the priority if the robot is to interact with other vehicles, it has been necessary to accept slightly degraded performance in order to improve the algorithm's ability to avoid collisions.

Conversely, at lower speeds there is not a significant advantage (or disadvantage) to using the actuation error extension with VOS. This is due to the smaller amount of accumulated positional error that can occur within one time step as well as the smaller possible change in the robot's velocity. The original VOS algorithm is sufficient to compensate for the lesser amount of actuation errors that may occur under these circumstances. However, this change implies that at velocities higher than those tested for this research, the actuation error extension will be even more critical to ensuring the safety of a robot operating with VOS.

The work in this chapter points clearly to the need to improve the motion control of UGVs for autonomous operation. Whereas current UGVs are adequately designed for their intended use as tele-operated vehicles, for reliable autonomous operation of UGVs, more stringent motion control specifications are necessary.

Chapter 5

Summary, Conclusions and Future Work

5.1. Summary and Conclusions

In this thesis *velocity occupancy space* (VOS), an algorithm that allows a robot to operate in an unknown environment and, with the use of only a range finding sensor with uncertainty, successfully avoid stationary and moving obstacles while navigating towards a goal, has been developed. In addition, extensions to VOS that allow the algorithm to safely operate on an acceleration controlled, differential drive vehicle or on a velocity controlled vehicle with actuation error are also presented.

In Chapter 1, literature related to obstacle detection and avoidance was presented and this literature demonstrated the need for an obstacle avoidance method with the key properties (and original contributions) of VOS. VOS was also compared to two algorithms that were also designed for the avoidance of moving obstacles, specifically the PVO/BOF method and the Dynamic Window method. VOS was shown to have comparable obstacle avoidance and goal reaching capabilities as these two algorithms with significantly less computational complexity.

In Chapter 2, background information was given on occupancy space and velocity obstacles (two of the main elements of the original VOS algorithm) as well as on some of the fundamental concepts utilized by the VOS derivation. Then the basic VOS algorithm was derived and presented for a holonomic robot with the ability to instantaneously change its velocity. The VOS algorithm utilizes occupancy space to estimate the location of each obstacle and to find each obstacle's center of certainty. The centers of certainty are then tracked in order to approximate each obstacle's velocity over multiple time steps. Next, the obstacle information is converted into velocity obstacle form and used to calculate a set of variables that describe the advantage or disadvantage of each possible robot velocity. The relative weights of these variables are determined either by hand-tuning or through an optimization process (using a set of obstacle/goal scenarios) and the results are used to form VOS. From this space, the algorithm can find a velocity that is both safe and that will lead it towards the goal. While the choice of velocity using the optimized weights may not always be ideal, it has been shown that, in most simulations, those weights will allow the robot to avoid a collision and reach its destination.

VOS was also compared to two reference obstacle avoidance algorithms from literature. The comparisons involved the two reference algorithms, the VFH and the velocity obstacle concept, being tested on the same scenarios as VOS. Based on these comparisons, it was found that VOS performs comparably to these algorithms in the scenarios that the reference algorithms were designed for (stationary obstacles or complete environmental knowledge) and VOS performs superiorly for the scenarios that it was designed for (moving obstacles with uncertain sensor data).

In short, VOS has the characteristics from Table 1.2 that were determined to be needed in an obstacle avoidance algorithm based on the literature review summarized in Table 1.1. Specifically, VOS has comparatively low computational complexity as demonstrated by the fact that the velocity occupancy space gird can be built and fully populated in less than 10ms (Section 1.2.3.). The algorithm is not susceptible to visible local minima due to the perpetual nature of the relative velocity obstacles that are used (Section 1.2.4.). VOS also possess inherent goal navigation (Section 2.3.2.) and automatically incorporates sensor uncertainty through the use of occupancy space (Section 2.2.1.). The center of certainty technique allows VOS to estimate obstacle velocities from laser range finder data (as derived in Section 2.2.1. and tested experimentally in Appendix A) and VOS accounts for velocity uncertainty and possible occlusions using the V_U and P_A terms (Section 2.3.1.) so the algorithm does not require any independent obstacle knowledge. Finally, while VOS assumes that obstacles will have constant velocities (at least in the short term), the fast update rate and the accommodations for obstacle position and velocity uncertainty built into the algorithm free VOS from being dependent on restricted obstacle velocities, as shown by its 0.8% failure rate around obstacles with variable velocities (Section 2.5.3.).

In Chapter 3, the basic VOS algorithm was augmented so that it could be used by a non-holonomic, differential drive vehicle without the ability to instantaneously change its velocity. Two different methods that allow a differential drive vehicle to ‘mimic’ a holonomic vehicle were derived and simulated. The first was a two-step velocity approximation method that provides the VOS algorithm with a wide range of velocities from which to select. However, when using this method, the vehicle does not move to the correct position by the end of the motion time step – though it does end up moving at the correct velocity. The second method was a three-step velocity approximation method which is more computationally complex and greatly reduces the selectable set of vehicle velocities, but causes the vehicle to end up at both the correct position and velocity. These two methods were combined to form a proximity dependent method that allows the algorithm to select the most appropriate (two- or three-step) method based on the robot's current state as well as the locations of surrounding obstacles. The material presented in this chapter show that the VOS algorithm is suitable for a non-holonomic robot with a realistic kinodynamic configuration.

In Chapter 4, VOS was again extended, this time to compensate for the actuation error in experimental vehicles. The velocity error of a SuperDroid robot was characterized, and this data was used to augment each velocity obstacle so that the robot's actuation error would not cause a collision. Using the actuation error extension, the simulated robot suffered from fewer collisions and, in general, maintained more distance between the robot and nearby obstacles. However, the robot generally took longer to reach the goal due to the more conservative velocity selection. This extension of VOS was tested extensively in simulation and also shown to be effective in an experimental setting via numerous trials with the SuperDroid robot. Videos of some of these experiments can be viewed on-line at:

https://sites.google.com/site/rachaelbis/thesis-appendix-c

VOS, along with its extensions, has been shown to be an effective algorithm to control autonomous robot navigation in an unknown environment with moving obstacles where there is error both from the sensor data that the algorithm receives as well as from the robot's actuators' response to the algorithms commands. The performance of VOS has been validated statistically through numerous simulations and it has been shown to be successful under real-world conditions through experimental trails.

VOS is not meant as a complete and independent robot navigation system. It is designed to perform low level robot navigation and obstacle avoidance. While it can operate independently in many types of environments, it would not be suitable to plan long, complex paths. As such, VOS would ideally be integrated with other algorithms to allow for higher level planning and navigation. For example, integration with an algorithm such as SLAMMOT [47] would allow the robot to respond quickly to avoid moving and stationary obstacles (while still choosing desirable, goal approaching velocities) using VOS, but also build a map of and recognize an environment so that it could make more complex navigation decisions. A higher level algorithm could break down a long, but desirable, route to a destination and provide VOS with more direct, intermediate goals.

5.2. Future Work

5.2.1. Modifications of the original VOS

In the future, as previously mentioned, VOS could be adapted to be directly integrated with different types of high level planners. Different sets of variable weights could be optimized for various types of specific environments, such as structured roads or uncluttered areas. During navigation, the algorithm could identify what type of environment it was in (structured, cluttered, etc.) and then select the weights that had been optimized for such a situation.

In addition, the linear combination of terms and weights in Eq. (2.20) has not been shown to be better than other possible arrangements. Through more extensive optimization, the exponential values of these terms could be evaluated to see if a different relationship between the powers of these terms would lead to superior performance. Additional terms and relationship could also be experimented with to see how they affect the algorithm.

The computational complexity of VOS is dependent on the resolution of the configuration and velocity occupancy space grids. Fairly high resolutions were used for the simulation and experimental work in this thesis; the configuration resolution was set at $0.1m$ and the velocity resolution was between $0.05\frac{m}{s}$ and $0.10\frac{m}{s}$. A lower resolution would decrease the precision with which the algorithm could avoid obstacles but would also decrease the processing time. The resolutions should be selected based on the environment (e.g., obstacle size and spacing) and the robot's capabilities (e.g., how precisely the robot can follow a specific velocity) and could be determined through more extensive testing or possibly through an optimization process.

Finally, VOS accounts for sensor error based on the error characteristics of the Hokuyo laser range finder which was used for the experimental trials. Additional research with lower cost (and more error prone) laser range finders would indicate the extent to which VOS can accommodate sensor uncertainty.

5.2.2. Additional Acceleration Method Selection Criteria

The proximity dependent combination of the two- and three-step acceleration methods could also be modified to possibly improve the algorithm. While proximity dependence has proven to be an adequate way to combine the two acceleration methods, they could also be combined using any number of factors. For example, the results may be improved by determining if the previously selected robot velocity was a part of a velocity obstacle and, if so, what the time to collision was for that velocity. If the robot velocity was not part of a velocity obstacle (or if there was a very high time to collision) then this would indicate that the current velocity is relatively safe and the robot is not headed towards any obstacles and this information may alter the proximity at which the three-step method should be selected over the two-step method. An optimization processes could be employed to find the ideal combination.

In addition, if a higher level planner were being used, the proximity dependent method could consider information from this planner when selecting the acceleration method. For example, if the higher level planner is aware that the robot will soon be entering a more cluttered region then the three-step method could be employed earlier to improve the initial alignment of the robot's path. Conversely, if the robot was about to exit a cluttered region, the higher level planner could indicate that the two-step method was a safe option, even if there were obstacles close behind the robot.

5.2.3. Additional Sensor Data

The VOS framework also lends itself well to the inclusion of data from additional sensors, such as infrared or visible light cameras, which could be used to identify different types of obstacles. If vulnerable obstacles, such as pedestrians, were identified then the velocities that led to a collision with these vulnerable obstacles could be assigned more repulsive weights so that the avoidance of these obstacles would be prioritized. Other researchers are currently exploring the possibility of adapting VOS so that it will respond in a safe and non-threatening manner to any pedestrians that it may encounter [3].

5.2.4. Modifications for Different Terrain and Vehicle Types

One of the first assumptions in the formation of VOS was that the terrain was flat and could therefore be assumed to be 2D. This is not the case for many experimental environments, so it would be useful to also explore how VOS would operate in a more complex terrain. Other researchers have considered terrain when developing robot navigation algorithms, and some of their results may be able to be adapted for VOS [89]. It would also be interesting to expand the VOS algorithm so that it could be used for 3D navigation on an UAV. While the 3D VOS would have substantially higher computational complexity, the velocity obstacle concept has been applied to a 3D workspace [90] with complete obstacle knowledge, so VOS may be a logical extension when sensor error is present.

In addition, while VOS has been extended from its original holonomic derivation, to operate with acceleration and velocity commanded vehicles, it could also be extended to operate with additional types of vehicles with other kinodynamic configurations, such as steered-cars.

Finally, the actuation error extension can compensate for error induced from both the environment as well as from the vehicle – as long as the extent of the error can be bounded. In the future, the error caused by various types of unstable terrain (e.g. sandy or muddy soil) and possible vehicle malfunctions (such as partial loss of power) could be characterized and the compensation for these errors could be integrated into the VOS actuation error framework.

Appendix A

Results from Low Speed Obstacle Velocity Estimation

Three obstacles with different aspect ratios were mounted on top of an iRobot Create robot and driven across a room at eight different command velocities. The obstacles were tracked over a distance of approximately 10m with a stationary laser range finder. Laser range finder data from the middle portion of the test, when the obstacle was at a mostly steady state velocity, was analyzed to determine the obstacles' velocities.

The obstacles' velocities were calculated from this data in two ways. First, they were calculated by hand from the raw laser range finder data. The obstacle was located in the laser range finder data and its position was measured and recorded at regular intervals in order to calculate the obstacle's velocity. This was a tedious but accurate way to measure the velocity, and the results from these measurements can be seen in Table A1 in the column *Measured Velocity.* Second, the obstacle velocities were calculated using the center of certainty method described in Section 3.1.1 and the results from this calculation are shown in Table A1 in the column *COC Calculated Velocity.*

As can be seen from the table, there was a significant amount of error in the COC velocity calculation at lower velocities, but (with one exception) the velocity error was less than 6% for all of the tests where the obstacle was moving at over $0.2\frac{m}{s}$.

Table A1. Low Speed Obstacle Velocity Estimation

Command Velocity (m/s)	Test	Measured Velocity (m/s)	% Error from Command Velocity	Standard Deviation	COC Calculated Velocity (m/s)	% Error from Measured Velocity	Standard Deviation
0.10	1	0.095	4.507	0.134	0.129	35.330	0.081
	2	0.094	5.669	0.158	0.108	14.023	0.049
	3	0.094	6.401	0.172	0.107	14.035	0.047
	AVG	**0.094**	**5.526**	**0.155**	**0.115**	**21.130**	**0.059**
0.15	1	0.144	4.225	0.151	0.154	7.455	0.043
	2	0.144	3.704	0.180	0.153	6.117	0.056
	3	0.144	3.937	0.205	0.160	10.913	0.066
	AVG	**0.144**	**3.955**	**0.179**	**0.156**	**8.162**	**0.055**
0.20	1	0.195	2.386	0.227	0.205	5.087	0.061
	2	0.195	2.486	0.247	0.206	5.399	0.067
	3	0.196	1.882	0.219	0.207	5.443	0.066
	AVG	**0.195**	**2.251**	**0.231**	**0.206**	**5.310**	**0.065**
0.25	1	0.244	2.564	0.250	0.252	3.604	0.072
	2	0.246	1.657	0.287	0.253	2.951	0.071
	3	0.243	2.692	0.256	0.254	4.365	0.080
	AVG	**0.244**	**2.304**	**0.264**	**0.253**	**3.640**	**0.075**
0.30	1	0.292	2.828	0.267	0.308	5.550	0.076
	2	0.295	1.514	0.262	0.329	11.428	0.126
	3	0.295	1.731	0.275	0.302	2.408	0.071
	AVG	**0.294**	**2.024**	**0.268**	**0.313**	**6.462**	**0.091**
0.35	1	0.346	1.066	0.292	0.356	1.664	0.085
	2	0.349	0.216	0.278	0.359	2.554	0.098
	AVG	**0.348**	**0.641**	**0.285**	**0.355**	**2.109**	**0.091**
0.40	1	0.397	0.657	0.301	0.408	2.693	0.088
	2	0.395	1.158	0.268	0.406	2.657	0.078
	3	0.391	2.254	0.258	0.407	3.983	0.086
	AVG	**0.395**	**1.357**	**0.276**	**0.407**	**3.111**	**0.084**
0.45	1	0.450	0.062	0.257	0.463	2.914	0.091
	2	0.458	1.683	0.262	0.464	1.466	0.077
	3	0.459	1.989	0.294	0.464	1.109	0.084
	AVG	**0.455**	**1.245**	**0.271**	**0.464**	**1.830**	**0.080**

Appendix B

Hardware Specifications

2.7. B.1 Specifications for robot used for the experimental testing: SuperDroid

- Chassis: SuperDroid ATR Enclosed Heavy Duty 4WD Chassis with Acrylic Covers
    - Modified to front 2WD with third omi-directional wheel
- Motor Controller: Pololu 18V25 High-Power Motor Driver
- Battery: 26V 9.9Ah LiFePO4
- Motors: SuperDroid IG42 24VDC 252 RPM Gear Motor

Table B1. SuperDroid Motor Specifications

Rated voltage	24 V
Gear reduction ratio	1:24
Rated torque	10 kgf-cm
Rated speed	252 rpm
Rated current	< 2300 mA
No load speed	290 rpm
No load current	< 650 mA

B.2 Specifications for Gyroscope: MicroInfinity Cruizcore XA3300

Table B2. Gyroscope Specifications

Performance	Input Range		± 100 ˚/sec (Continuous)
			± 300 ˚/sec (Instantaneous)
	Roll, Pitch Accuracy	Static Error	< 0.5 ˚
		Dynamic Error	< 2 ˚
	Heading Accuracy	Static Error	< 1 ˚
	Resolution		0.05 ˚
	Bandwidth		20 Hz
	Update Rate		> 100 Hz (USB, RS-232)
Physical	Weight		20 g (Including case)
	Size (L, W, H)		53.9 mm X 35.9 mm X 17 mm
Electrical	Power Consumption		< 400 m W
	Input voltage		4.75 ~ 5.25 V
Environmental	Operating temperature		-40 ~ 70 °C
	Shock		200 gRMS

B.3 Specifications for Laser Range Finder: Hokuyo UTM-30LX

Table B3. Laser Range Finder Specifications

Voltage	12.0 V ±10%
Current	0.7 A (Rush current 1.0 A)
Detection range	0.1 m to approximately 60 m (<30 m guaranteed)
Laser wavelength	870 nm, Class 1
Scan angle	270°
Scan time	25 ms/scan (40.0 Hz)
Angular resolution	0.25°
Interface	USB 2.0
Weight	8.2 oz (233 g)

B.4 Specifications for moving obstacles: iRobot Create

Table B4. iRobot Create Relevant Specifications

Driven wheels	two; left and right
Caster wheels	two; front and back
Wheel velocity range	$-0.5\frac{m}{s}\ to\ 0.5\frac{m}{s}$
Communication	Bluetooth®

B.5 Specifications for on-board laptop computer: Dell Latitude E6400

Table B5. Computer Specifications

Operating System	Windows Vista
Processor	Intel® Core™2 Duo CPU 2.54 GHz
Memory (RAM)	4.00GB
System type	32-bit Operating System

Appendix C

Video Results from SuperDroid Testing

Video results from many of the experimental trials are available at www.youtube.com at the URLs listed in Tables C1-C3. A list of all of the videos (with links) is available at:

https://sites.google.com/site/rachaelbis/thesis-appendix-c

There are two videos available for each trial. The first is video recorded of the robot and environment during the trial (Figure 4.17 is a single frame taken from one of these videos) and the second is the VOS display (Figure 4.18 is a single frame taken from one of these videos).

In the VOS display, the lower, right hand corner shows the laser range finder output (this is the entirety of the external sensor data with which the robot is provided). The upper, left hand corner shows the configuration space occupancy grid that is built from this data. The lower left and middle images are the obstacle-based (repulsive) and goal-based (attractive) velocity spaces that are built using the obstacle's locations and estimated velocities. Finally, the upper, right hand corner shows the combined velocity occupancy space from which the next robot velocity is selected.

Table C1. Experimental Results from fast SuperDroid test with Actuation Error Extension

$$max\ v\ =\ 1.0\frac{m}{s},max\ w\ =1.0\frac{rad}{s}$$

Test #	Video URL	VOS URL
F1	http://www.youtube.com/watch?v=dolZdmnaE68	http://www.youtube.com/watch?v=TJLT4JR5-rw
F2	http://www.youtube.com/watch?v=SPTGMaTsVOs	http://www.youtube.com/watch?v=WWyCbWFP_mo
F3	http://www.youtube.com/watch?v=jHZxBF9KCTw	http://www.youtube.com/watch?v=rj2XEmP9Dlo
F4	http://www.youtube.com/watch?v=HMey7DfsoOE	http://www.youtube.com/watch?v=yNz2cyyQUg4
F5	http://www.youtube.com/watch?v=wkRstzR3P9M	http://www.youtube.com/watch?v=GAkDL9utM8Y
F6	http://www.youtube.com/watch?v=BvyW_Iyv0Xw	http://www.youtube.com/watch?v=BWEBw0ImPM
F7	http://www.youtube.com/watch?v=W2gZxQnk4dE	http://www.youtube.com/watch?v=et0hqHCxDU0
F9	http://www.youtube.com/watch?v=9zNHCA_0gXE	http://www.youtube.com/watch?v=2yYLsRsCSFw
F10	http://www.youtube.com/watch?v=JW6CfqyTa4E	http://www.youtube.com/watch?v=MdFKcCskMFs
F11	http://www.youtube.com/watch?v=jYmzmSAZfNs	http://www.youtube.com/watch?v=0LiRFp5CrbI
F12	http://www.youtube.com/watch?v=8Iq4OZzjDjk	http://www.youtube.com/watch?v=iwrIsJJVlsM
F13	http://www.youtube.com/watch?v=CQFBgPlTZ4M	http://www.youtube.com/watch?v=72T3cBbdzsY
F14	http://www.youtube.com/watch?v=3UI0BV4HlpM	http://www.youtube.com/watch?v=47q5DHkPYr0

Table C2. Experimental Results from medium SuperDroid test with Actuation Error Extension

$$max\ v\ =\ 0.7\frac{m}{s}, max\ w\ =1.0\frac{rad}{s}$$

Test #	Video URL	VOS URL
M1	http://www.youtube.com/watch?v=b4eHIh2m5tY	http://www.youtube.com/watch?v=K3UaqJRVrTk
M2	http://www.youtube.com/watch?v=9aj_mc5sRbk	http://www.youtube.com/watch?v=cCivH32N6kE
M4	http://www.youtube.com/watch?v=bU8T4t9pP0k	http://www.youtube.com/watch?v=nUna2wcfU20
M5	http://www.youtube.com/watch?v=qS2rVpEvI54	http://www.youtube.com/watch?v=q1Uru_doe5w

Table C3. Experimental Results from slow SuperDroid test with Actuation Error Extension

$$max\ v\ =\ 0.5\frac{m}{s}, max\ w\ =1.0\frac{rad}{s}$$

Test #	Video URL	VOS URL
S1	http://www.youtube.com/watch?v=Q8dy0mEIaDg	http://www.youtube.com/watch?v=aQnvhgKtAKA
S2	http://www.youtube.com/watch?v=-fVS2-JXsjQ	http://www.youtube.com/watch?v=RFBE791XDXo
S3	http://www.youtube.com/watch?v=crZBgRRkK18	http://www.youtube.com/watch?v=3STjAtL1L-g
S4	http://www.youtube.com/watch?v=mjTLjKI84Rk	http://www.youtube.com/watch?v=0TgrSNzkdlo
S5	http://www.youtube.com/watch?v=-kFAcEo3fzw	http://www.youtube.com/watch?v=cpX2eSW5G0A
S6	http://www.youtube.com/watch?v=WgL3ayrOJ1g	http://www.youtube.com/watch?v=R-OI8La-RZU
S7	http://www.youtube.com/watch?v=ifFQH1h2quQ	http://www.youtube.com/watch?v=BoRP0QEYgKc
S8	http://www.youtube.com/watch?v=-RoORS_70ww	http://www.youtube.com/watch?v=864FbhepHFw
S9	http://www.youtube.com/watch?v=x0_3xRTLVVM	http://www.youtube.com/watch?v=AJ5kXmx97io
S10	http://www.youtube.com/watch?v=Pb5o2xKnpOk	http://www.youtube.com/watch?v=vljXPYEmW6s
S11	http://www.youtube.com/watch?v=h7zaC_uEz04	http://www.youtube.com/watch?v=Mpn7oF4QaVg
S12	http://www.youtube.com/watch?v=Y-Wm6n2Ane0	http://www.youtube.com/watch?v=1d3h-F_Xlec
S13	http://www.youtube.com/watch?v=435IySAvek8	http://www.youtube.com/watch?v=-l50z_YfjZg
S14	http://www.youtube.com/watch?v=WyClf8zHCBA	http://www.youtube.com/watch?v=paXgV5lJr3M

Appendix D

Experimental Results with SuperDroid

Table D1. Experimental Results from fast SuperDroid test with Actuation Error Extension

$$max\ v\ =\ 1.0\frac{m}{s}, max\ w\ =\ 1.0\frac{rad}{s}$$

Test #	Evaluation Metrics					
	Obstacle Proximity $\left(\frac{1}{m^2}\right)$	Distance Traveled (m)	Acceleration $\left(\frac{m}{s^2}\right)$	Time (s)	# of Moving Obstacles	Obstacle Velocity $\left(\frac{m}{s^2}\right)$
F1	0.842	10.69	0.132	26.50	3	±0.4
F2	0.615	9.25	0.137	22.47	1	±0.5
F3	0.739	13.18	0.133	22.58	3	±0.4
F4	1.080	11.48	0.141	20.23	3	±0.3
F5	1.283	18.44	0.127	35.39	3	±0.3
F6	1.250	10.78	0.125	18.50	3	±0.5
F7	0.931	10.68	0.121	17.14	3	±0.5
F8	0.747	11.69	0.109	55.85	1	±0.3
F9	0.702	12.41	0.135	19.94	0	N/A
F10	0.610	12.56	0.113	20.40	0	N/A
F11	0.890	11.30	0.130	21.69	1	±0.5
F12	0.735	16.71	0.091	55.04	0	N/A
F13	0.796	10.47	0.134	21.29	1	±0.5
F14	1.476	7.80	0.107	40.35	3	±0.5
Average	0.907	11.96	0.124	28.38		

Table D2. Experimental Results from medium SuperDroid test with Actuation Error Extension

$$max\ v\ =\ 0.7\frac{m}{s}, max\ w\ =1.0\frac{rad}{s}$$

Test #	Evaluation Metrics					
	Obstacle Proximity $\left(\frac{1}{m^2}\right)$	Distance Traveled (m)	Acceleration $\left(\frac{m}{s^2}\right)$	Time (s)	# of Moving Obstacles	Obstacle Velocity $\left(\frac{m}{s^2}\right)$
M1	0.617	10.47	0.102	23.33	1	±0.5
M2	0.747	10.96	0.133	26.58	3	±0.4
M3	3.451	7.98	0.091	37.52	1	±0.3
M4	2.747	16.27	0.128	40.09		
M5	1.180	17.35	0.124	43.40	1	±0.3
Average	1.749	12.61	0.116	34.18		

Table D3. Experimental Results from slow SuperDroid test with Actuation Error Extension

$$max\ v\ =\ 0.5\frac{m}{s}, max\ w\ =1.0\frac{rad}{s}$$

Test #	Evaluation Metrics					
	Obstacle Proximity $\left(\frac{1}{m^2}\right)$	Distance Traveled (m)	Acceleration $\left(\frac{m}{s^2}\right)$	Time (s)	# of Moving Obstacles	Obstacle Velocity $\left(\frac{m}{s^2}\right)$
S1	1.167	10.16	0.109	44.63	0	0
S2	0.856	9.65	0.119	33.27	0	0
S3	0.984	10.95	0.099	52.96	0	0
S4	1.646	14.76	0.117	72.47	0	0
S5	1.267	8.92	0.106	44.38	0	0
S6	0.839	9.18	0.128	24.15	1	±0.3
S7	0.631	16.55	0.123	64.98	0	0
S8	0.656	23.42	0.128	97.69	0	0
S9	1.399	13.10	0.114	46.68	3	±0.3
S10	0.829	11.47	0.113	38.78	0	0
S11	1.179	10.56	0.111	41.46	2	±0.35
S12	0.865	13.05	0.117	54.59	3	±0.45
S13	0.703	11.46	0.128	44.08	3	±0.45
S14	1.459	18.30	0.120	72.30	2	±0.35
S15	0.792	6.24	0.077	38.24	1	±0.25
Average	1.018	12.52	0.144	51.38		

Table D4. Experimental Results from fast SuperDroid test without Actuation Error Extension

$$max\ v = 1.0\frac{m}{s}, max\ w = 1.0\frac{rad}{s}$$

Test #	Evaluation Metrics					
	Obstacle Proximity $\left(\frac{1}{m^2}\right)$	Distance Traveled (m)	Acceleration $\left(\frac{m}{s^2}\right)$	Time (s)	# of Moving Obstacles	Obstacle Velocity $\left(\frac{m}{s^2}\right)$
F1, NAE	9.200	2.67	0.020	32.69	3	±0.5
F2, NAE	2.527	11.95	0.079	50.88	3	±0.5
F3, NAE	2.256	6.98	0.045	44.02	3	±0.5
F4, NAE	2.069	5.09	0.054	26.88	3	±0.5
F5, NAE	2.353	11.16	0.137	17.93	3	±0.5
Average	2.301	8.79	0.079	34.93		

Table D5. Experimental Results from medium SuperDroid test without Actuation Error Extension

$$max\ v = 0.7\frac{m}{s}, max\ w = 1.0\frac{rad}{s}$$

Test #	Evaluation Metrics					
	Obstacle Proximity $\left(\frac{1}{m^2}\right)$	Distance Traveled (m)	Acceleration $\left(\frac{m}{s^2}\right)$	Time (s)	# of Moving Obstacles	Obstacle Velocity $\left(\frac{m}{s^2}\right)$
M1, NAE	0.918	13.52	0.131	33.44	3	±0.4
M2, NAE	1.416	12.06	0.130	28.71	3	±0.4
M3, NAE	6.381	14.45	0.100	44.29	3	±0.4
M4, NAE	0.912	7.80	0.064	40.35	3	±0.4
Average	2.407	11.96	0.106	36.70		

Table D6. Experimental Results from slow SuperDroid test without Actuation Error Extension

$$max\ v = 0.5\frac{m}{s}, max\ w = 1.0\frac{rad}{s}$$

Test #	**Evaluation Metrics**					
	Obstacle Proximity $\left(\frac{1}{m^2}\right)$	Distance Traveled (m)	Acceleration $\left(\frac{m}{s^2}\right)$	Time (s)	# of Moving Obstacles	Obstacle Velocity $\left(\frac{m}{s^2}\right)$
S1, NAE	0.727	10.85	0.110	50.09	3	±0.3
S2, NAE	0.836	11.07	0.119	43.15	3	±0.3
S3, NAE	1.235	5.62	0.079	33.15	3	±0.3
Average	0.933	9.18	0.103	42.13		

Bibliography

[1] P. W. Singer, "Military Robots and the Laws of War," *The New Atlantis,* pp. 27-47, 2009.

[2] S. Thrun, "Google's Driverless Car," Ted Talk, Ed., March 2011.

[3] W. Westrick, "Improving Pedestrian Safety and Comfort around UGVs through the Enhancement of Velocity Occupancy Space," Masters, Mechanical Engineering, University of Michigan, Ann Arbor, 2011.

[4] R. Bis, H. Peng, and G. Ulsoy, "Velocity Occupancy Space: Robot Navigation and Moving Obstacle Avoidance with Sensor Uncertainty," presented at the Dynamic Systems and Controls Conference, Hollywood, CA, 2009.

[5] R. Bis, H. Peng, and A. G. Ulsoy, "Velocity Occupancy Space: Autonomous Navigation in an Uncertain, Dynamic Environment," *International Journal of Vehicle Autonomous Systems,* 2012.

[6] R. Bis, H. Peng, and A. G. Ulsoy, "Velocity Occupancy Space for Differential Drive Vehicles," presented at the *Dynamic Systems and Controls Conference*, Cambridge, MA, 2010.

[7] R. Bis, H. Peng, and A. G. Ulsoy, "Velocity Occupancy Space for Acceleration Controlled, Differentially Driven Vehicles," *International Journal of Vehicle Autonomous Systems,* In preparation, 2012.

[8] R. Bis, H. Peng, and A. G. Ulsoy, "Velocity Occupancy Space Based Navigation for Vehicles with Actuation Error," *Autonomous Robotics,* In preparation, 2012.

[9] R. Siegwart and I. Nourbakhsh, *Introduction to Autonomous Mobile Robots*. Cambridge: The MIT Press, 2004.

[10] C. O'Dunlaing and C. K. Yap, "Retraction method for planning the motion of a disc," *Journal of Algorithms,* vol. 6, pp. 104–111, 1982.

[11] J.-C. Latombe, *Robot Motion Planning*. Boston: Kluwer, 1991.

[12] H. Choset and J. Burdick, "Sensor-based exploration: The hierarchical generalized Voronoi graph," *International Journal of Robotics Research,* vol. 19, pp. 96-125, 2000.

[13] J. Overholt, G. Hudas, G. Fiorani, M. Skalny, and A. Tucker, "Dynamic waypoint navigation using voronoi classifier methods," U. S. A. R.-T. R. M. Laboratory, Ed., ed. Warren, 2004.

[14] N. Nilsson, "A mobile automaton: an application of artificial intelligence techniques," presented at the Proceedings IJCAI-I, Washington, DC, 1969.

[15] H. Edelsbrunner, *Algorithms in Combinatorial Geometry*. Berlin,: Springer-Verlag, 1988.

[16] B. Oommen, S. Iyengar, N. Rao, and R. Kashyap, "Robot navigation in unknown terrains using learned visibility graphs. Part I: The disjoint convex obstacle case," *IEEE Journal of Robotics and Automation,* vol. 3, pp. 672-681, 1987.

[17] C. Urdiales, A. Bandera, F. Arrebola, and F. Sandoval. (1998, Multi-level path planning algorithm for autonomous robots. *Electronics Letters 34(2),* 223-224.

[18] Y. Qin, D. Sun, N. Li, and CenY., "Path planning for mobile robot using the particle swarm optimization with mutation operator," presented at the Proceedings of 2004 International Conference on Machine Learning and Cybernetics, 2004.

[19] C. Alexopoulos and P. Griffin, "Path planning for a mobile robot," *IEEE Transactions on Systems, Man and Cybernetics,* vol. 22, pp. 318-322, 1992.

[20] G. Oriolo, G. Ulivi, and M. Vendittelli, "Real-time map building and navigation for autonomous robots in unknown environments," *IEEE Transactions on Systems, Man and Cybernetics,* vol. 28, pp. 316-333, 1998.

[21] E. Gilbert and D. Johnson, "Distance functions and their application to robot path planning in the presence of obstacles," *IEEE Journal of Robotics and Automation,* vol. 1, pp. 21-30, 1985.

[22] T. Lozano-Perez, "Automatic planning of manipulator transfer movements," *IEEE Transactions on Systems, Man and Cybernetics,* vol. 11, pp. 681-698, 1981.

[23] H. Moravec and A. Elfes, "High resolution maps from wide angle sonar," in *Robotics and Automation. Proceedings. 1985 IEEE International Conference on*, St. Louis, Missouri, 1985, pp. 116-121.

[24] S. Thrun, "A probabilistic online mapping algorithm for teams of mobile robots," *International Journal of Robotics Research,* vol. 20, pp. 335-363, 2001.

[25] Li Jigong, F. Yiwei, and Z. Chaoqun, "A Novel Path Planning Method Based on Certainty Grids Map For Mobile Robot," presented at the Chinese Control Conference, 2007.

[26] D. J. Zhu and J. Latombe, "New heuristic algorithms for efficient hierarchical path planning," *IEEE Transactions on Robotics and Automation,* vol. 7, pp. 9-20, 1991.

[27] O. Khatib, "Real-time obstacle avoidance for manipulators and mobile robots," in *Robotics and Automation. Proceedings. 1985 IEEE International Conference on*, 1985, pp. 500-505.

[28] J. Borenstein and Y. Koren, "Real-Time Obstacle Avoidance For Fast Mobile Robots," *Ieee Transactions on Systems Man and Cybernetics,* vol. 19, pp. 1179-1187, 1989.

[29] Y. Koren and J. Borenstein, "Potential field methods and their inherent limitations for mobile robot navigation," in *Robotics and Automation, 1991. Proceedings., 1991 IEEE International Conference on*, 1991, pp. 1398-1404 vol.2.

[30] J. Borenstein and Y. Koren, "The vector field histogram-fast obstacle avoidance for mobile robots," *Robotics and Automation, IEEE Transactions on,* vol. 7, pp. 278-288, 1991.

[31] Y. K. Hwang and N. Ahuja, "A potential field approach to path planning," *IEEE Transactions on Robotics and Automation,* vol. 8, pp. 23-32, 1992.

[32] L. Montano and J. R. Asensio, "Real-time robot navigation in unstructured environments using a 3D laser rangefinder," in *Proceedings of the 1997 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 1997, pp. 526-532.

[33] P. Batavia and I. Nourbakhsh, "Path Planning for the Cye Robot," in *Proceedings of IROS*, 2000, pp. 15 - 20.

[34] M. Khatib, H. Jaouni, R. Chatila, and J. P. Laumod, "Dynamic Path Modification for Car-Like Nonholonomic Mobile Robots," presented at the IEEE International Conference on Robotics and Automation, Albuquerque, USA, 1997.

[35] I. Ulrich and J. Borenstein, "VFH+: reliable obstacle avoidance for fast mobile robots," in *Robotics and Automation, 1998. Proceedings. 1998 IEEE International Conference on*, 1998, pp. 1572-1577 vol.2.

[36] I. Ulrich and J. Borenstein, "VFH*: local obstacle avoidance with look-ahead verification," in *IEEE International Conference on Robotics and Automation*, 2000, pp. 2505-2511.

[37] K. Kant and S. Zucker, "Toward efficient trajectory planning: The path-velocity decomposition," *International Journal of Robotics Research,* vol. 5, pp. 72-89, 1986.

[38] B. Lee and C. Lee, "Collision-free motion planning of two robots," *IEEE Transactions on Systems, Man and Cybernetics,* vol. 17, pp. 21-32, 1987.

[39] K. Fujimura and H. Samet, "Planning a time-minimal motion among moving obstacles," *Algorithmica,* vol. 10, pp. 41-63, 1993.

[40] K. Konolige, "A gradient method for real time robot control," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, Takamatsu, Japan, 2000, pp. 639-646.

[41] A. Fujimori, P. N. Nikiforuk, and M. M. Gupta, "Adaptive navigation of mobile robots with obstacle avoidance," *IEEE Transactions on Robotics and Automation,* vol. 13, pp. 596-601, 1997.

[42] H. Zhuang, H. Li, and D. S., "Real-time Path Planning of Mobile Robots in Dynamic Uncertain Environment," presented at the The Sixth World Congress on Intelligent Control and Automation, 2006.

[43] S. Quinlan and O. Khatib, "Elastic bands: connecting path planning and control," in *Robotics and Automation, 1993. Proceedings., 1993 IEEE International Conference on*, 1993, pp. 802-807 vol.2.

[44] T. Fraichard, "Trajectory Planning Amidst Moving Obstacles: Path-Velocity Decomposition Revisited," *Journal of the Brazilian Computer Society,* vol. 4, 1998.

[45] R. Kindel, D. Hsu, J.-C. Latombe, and S. Rock, "Kinodynamic motion planning amidst moving obstacles," in *IEEE International Conference on Robotics and Automation*, 2000, pp. 537-543.

[46] H. Yu and T. Su, "A destination driven navigator with dynamic obstacle motion prediction," in *IEEE International Conference on Robotics and Automation*, 2001, pp. 2692-2697.

[47] Y. Wang, I. P. W. Sillitoe, and D. J. Mulvaney, "Mobile Robot Path Planning in Dynamic Environments," presented at the IEEE International Conference on Robotics and Automation, 2007.

[48] J. Xiao, Z. Michalewicz, and L. Zhang, "Evolutionary planner/navigator: Operator performance and self-tuning," in *Proceedings 3rd IEEE International Conference Evolutionary Computation*, Nagoya, Japan, 1996, pp. 366-371.

[49] J. Xiao, *Evolutionary planner/navigator in a mobile robot environment*. New York: Oxford Univ. Press and Institute of Physics, 1997.

[50] W. Han, S. Baek, and T. Kuc, "Genetic algorithm based path planning and dynamic obstacle avoidance of mobile robots," presented at the IEEE International Conference on Systems, Man, and Cybernetics, 1997.

[51] K. Sugihara and J. Smith, "Genetic Algorithms for Adaptive Planning of Path and Trajectory of a Mobile Robot in 2D terrains," *IEICE Transactions Information and Systems,* vol. E82-D, pp. 309--316, 1999.

[52] W. Malik, "Motion planning of mobile robot in dynamic environment using potential field and roadmap based planner," Texas A&M University, 2003.

[53] A. Farinelli and L. Iocchi, "Planning trajectories in dynamic environments using a gradient method," in *Proceedings of RoboCup Symposium*, Padua, Italy, 2003.

[54] T. Fraichard and H. Asama, "Inevitable collision states. A step towards safer robots?," in *Intelligent Robots and Systems, 2003. (IROS 2003). Proceedings. 2003 IEEE/RSJ International Conference on*, 2003, pp. 388-393 vol.1.

[55] R. Parthasarathi and T. Fraichard, "An Inevitable Collision State-Checker for a Car-Like Vehicle," in *Robotics and Automation, 2007 IEEE International Conference on*, 2007, pp. 3068-3073.

[56] L. Martinez-Gomez and T. Fraichard, "Collision avoidance in dynamic environments: An ICS-based solution and its comparative evaluation," in *Robotics and Automation, 2009. ICRA '09. IEEE International Conference on*, 2009, pp. 100-105.

[57] P. Fiorini and Z. Shiller, "Motion planning in dynamic environments using the relative velocity paradigm," in *IEEE International Conference on Robotics and Automation*, 1993, pp. 560-565.

[58] Z. Shiller, F. Large, and S. Sekhavat, "Motion planning in dynamic environments: Obstacles moving along arbitrary trajectories," in *2001 IEEE International Conference on Robotics and Automation, Vols I-IV, Proceedings*, Seoul, Korea, 2001, pp. 3716-3721.

[59] 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, 2005.

[60] M. Yamamoto, M. Shimada, and A. Mohri, "Online navigation of mobile robot under the existence of dynamically moving multiple obstacles," in *Assembly and Task Planning, 2001, Proceedings of the IEEE International Symposium on*, 2001, pp. 13-18.

[61] B. Kluge and E. Prassler, "Reflective navigation: individual behaviors and group behaviors," in *Robotics and Automation, 2004. Proceedings. ICRA '04. 2004 IEEE International Conference on*, 2004, pp. 4172-4177 Vol.4.

[62] C. Fulgenzi, A. Spalanzani, and C. Laugier, "Dynamic Obstacle Avoidance in uncertain environment combining PVOs and Occupancy Grid," in *Robotics and Automation, 2007 IEEE International Conference on*, Roma, Italy, 2007, pp. 1610-1616.

[63] C. Fulgenzi, "Autonomous navigation in dynamic uncertain environment using probabilistic models of perception and collision risk prediction," Ph.D.,

Engineering, Institut National Polytechnique de Grenoble, Rhne-Alpes, France, 2009.

[64] D. Fox, W. Burgard, and S. Thrun, "The dynamic window approach to collision avoidance," *Ieee Robotics & Automation Magazine,* vol. 4, pp. 23-33, 1997.

[65] O. Brock and O. Khatib, "High-speed navigation using the global dynamic window approach," in *IEEE International Conference on Robotics and Automation*, 1999, pp. 341-346.

[66] M. Seder, K. Macek, and I. Petrovic, "An integrated approach to real-time mobile robot control in partially known indoor environments," in *Industrial Electronics Society, 2005. IECON 2005. 31st Annual Conference of IEEE*, 2005, p. 6 pp.

[67] M. Seder and I. Petrovic, "Dynamic window based approach to mobile robot motion control in the presence of moving obstacles," in *Robotics and Automation, 2007 IEEE International Conference on*, 2007, pp. 1986-1991.

[68] D. Schulz, W. Burgard, D. Fox, and A. B. Cremers, "People Tracking with Mobile Robots Using Sample-Based Joint Probabilistic Data Association Filters," *The International Journal of Robotics Research,* vol. 22, pp. 99-116, February 1, 2003 2003.

[69] J. Almeida and R. Araujo, "Tracking multiple moving objects in a dynamic environment for autonomous navigation," in *Advanced Motion Control, 2008. AMC '08. 10th IEEE International Workshop on*, 2008, pp. 21-26.

[70] R. Benenson, S. Petti, T. Fraichard, and M. Parent, "Towards urban driverless vehicles," *International Journal of Vehicle Autonomous Systems,* vol. 6, pp. 4-23, 2008.

[71] S. M. LaValle and J. J. Kuffner, Jr., "Randomized kinodynamic planning," in *Robotics and Automation, 1999. Proceedings. 1999 IEEE International Conference on*, 1999, pp. 473-479 vol.1.

[72] S. M. LaValle and J. J. Kuffner, "Randomized Kinodynamic Planning," *The International Journal of Robotics Research,* vol. 20, pp. 378-400, 2001.

[73] Y. Kuwata, G. A. Fiore, J. Teo, E. Frazzoli, and J. P. How, "Motion planning for urban driving using RRT," in *Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on*, 2008, pp. 1681-1686.

[74] C. Fulgenzi, C. Tay, A. Spalanzani, and C. Laugier, "Probabilistic navigation in dynamic environment using Rapidly-exploring Random Trees and Gaussian processes," in *Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on*, 2008, pp. 1056-1062.

[75] E. Owen and L. Montano, "Motion planning in dynamic environments using the velocity space," in *Intelligent Robots and Systems, 2005. (IROS 2005). 2005 IEEE/RSJ International Conference on*, 2005, pp. 2833-2838.

[76] E. Owen and L. Montano, "A Robocentric Motion Planner for Dynamic Environments Using the Velocity Space," in *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on*, 2006, pp. 4368-4374.

[77] D. Wilkie, J. van den Berg, and D. Manocha, "Generalized velocity obstacles," in *Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on*, 2009, pp. 5573-5578.

[78] J. Minguez and L. Montano, "Extending Collision Avoidance Methods to Consider the Vehicle Shape, Kinematics, and Dynamics of a Mobile Robot," *Robotics, IEEE Transactions on,* vol. 25, pp. 367-381, 2009.

[79] D. Morales and T. C. Son, "Interval Methods in Robot Navigation," *Reliable Computing,* vol. 4, pp. 55-61, 1998.

[80] A. Widyotriatmo and H. Keum-Shik, "Decision making framework for autonomous vehicle navigation," in *SICE Annual Conference, 2008*, 2008, pp. 1002-1007.

[81] E. Prassler, J. Scholz, and P. Fiorini, "Navigating a Robotic Wheelchair in a Railway Station during Rush Hour," *The International Journal of Robotics Research,* vol. 18, pp. 711-727, July 1, 1999 1999.

[82] P. Fiorini and Z. Shiller, "Motion planning in dynamic environments using velocity obstacles," *International Journal of Robotics Research,* vol. 17, pp. 760-772, 1998.

[83] F. Large, S. Sekhavat, Z. Shiller, and C. Laugier, "Towards real-time global motion planning in a dynamic environment using the NLVO concept," *2002 IEEE/RSJ International Conference on Intelligent Robots and Systems, Vols 1-3, Proceedings,* pp. 607-612, 2002.

[84] R. Benenson, S. Petti, T. Fraichard, and M. Parent, *Towards urban driverless vehicles*. Olney, SUISSE: Inderscience Enterprises, 2008.

[85] C. Fulgenzi, A. Spalanzani, and C. Laugier, "Probabilistic motion planning among moving obstacles following typical motion patterns," in *Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on*, 2009, pp. 4027-4033.

[86] K. C. Fuerstenberg, D. T. Linzmeier, and K. C. J. Dietmayer, "Pedestrian recognition and tracking of vehicles using a vehicle based multilayer laserscanner," in *ITS 2003, 10th World Congress on Intelligent Transport Systems*, Madrid, Spain, 2003, pp. 31- 35.

[87] A. M. Flynn, "Combining Sonar and Infrared Sensors for Mobile Robot Navigation," *The International Journal of Robotics Research,* vol. 7, pp. 5-14, 1988.

[88] S. A. Niyogi and E. Adelson, "Analyzing and recognizing walking figures in XYT," in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 1994, pp. 469-474.

[89] C. Urmson, C. Ragusa, D. Ray, J. Anhalt, D. Bartz, T. Galatali, A. Gutierrez, J. Johnston, S. Harbaugh, H. “Yu” Kato, W. Messner, N. Miller, K. Peterson, B. Smith, J. Snider, S. Spiker, J. Ziglar, W. “Red” Whittaker, M. Clark, P. Koon, A. Mosher, and J. Struble, "A robust approach to high-speed navigation for unrehearsed desert terrain," *Journal of Field Robotics,* vol. 23, pp. 467-508, 2006.

[90] J. Snape and D. Manocha, "Navigating multiple simple-airplanes in 3D workspace," in *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, 2010, pp. 3974-3980.