Rapid, High Precision Control in Tightly Constrained Environments

Clare Richardson
Computer Science Department
University of Texas at Austin
c.richardson@mail.utexas.edu

1 Problem Description

In environments where a robot has very little side clearance and hence very small margins for error, high precision in controlling the movements of the robot in order to avoid obstacles is necessary. The problem here lies in trying to design a control method that ensures speed, grace, and comfort which will successfully avoid obstacles in such environments.

The basic control loop of a robot is as follows: read in sensor data about the environment, perform some calculation, send velocity commands to the robot motors, and repeat. The calculations in between reading sensor data and sending commands to the motors reflect the behavior of the robot that we want. In each loop, we will calculate the new velocity commands with the aim that these new commands will result in the desired behavior. However, how to translate the sensor data that comes in to the velocity commands that go out is sometimes unclear.

In forming rules for the velocity command calculations, we must consider several desires for the behavior we want. First and foremost, we want the robot to attain some set goal point in the environment. We also want the robot to travel as quickly as possible, without hitting any obstacles along the way, and with fluid motion without jerks. Many approaches to control, such as [1] and [2], attain these goals for a large number of possible environments. Common environments for testing these control methods are labs and corridors, both crowded and not. However, there are also a large number of environments that are often untested and for which these goals are not yet fully achieved: tightly-constrained environments which require high precision control.

Tightly-constrained environments are environments where the side or front clearance of a robot from obstacles surrounding it is very small. Thus, there is little room for error and they require a high precision of movement in navigating around obstacles towards a goal. For the robot used in my experiments, I defined tightly-constrained environments as when the clearance is less than 0.25 m. Defining the clearance is based on the dimensions of the robot, the turning radius, the capabilities of the motors, and the sensor resolution. Environments requiring high precision control are much more common for larger robots since it is much more likely that side or front clearance will be very small than for smaller robots. Even humans have trouble navigating things in some tightly-constrained environments. For example, imagine a person driving a very wide remote control car via a joystick on the remote control. In attempting to move the car through a doorway that is only a couple inches wider than the car, the person may have to back up the car several times and try again before successfully steering the car through the doorway.

Such low clearance requires very careful calculations and planning for the velocity commands to the robot's motors in order to achieve our behavior goals. We must carefully calculate so that the robot does not collide with surrounding obstacles. In the example above, the nearby obstacles are both sides of the door. However, moving too cautiously will result in an unnecessarily low speed, which is contrary to one of our behavior goals (high speed). An extremely high speed, though, may cause a loss of control of the robot and cause it crash into an obstacle. We also want to ensure that the robot moves fluidly through the environment. In the example above, we would not want the car to start and stop every few seconds, each time deciding what the best next move is.

In this paper, I will describe two previously established control methods that are proven to be successful in several environments and discuss their effectiveness in selected tightly-constrained environments.

2 Background

Two different approaches to object collision avoidance were used to demonstrate their control ability in an environment that requires high precision. These control methods were chosen because of their potential success in this kind of environment. Both control methods have been shown to be successful in highly cluttered environments, so the hope was that this success would translate to the environments chosen for our testing purposes. Both the papers describing the selected control methods mention a robot's ability to go through narrow doorways, which is one of our tightly-constrained test environments, when using that particular control method. The dynamic window approach, proposed by Fox, Burgard, and Thrun, focuses on using a scoring function on all possible velocities to determine the best velocity command for the next finite time interval [2]. The vector field histogram approach was proposed by Borenstein and Koren. This approach focuses on reducing the data given about the environment at a certain time to sectors of "safe" robot orientations and selecting the sector most nearly facing the target [1]. Before providing an overview of these two approaches, I will describe the current collision avoidance method used in the UT Intelligence Robotics Lab on the Vulcan robot.

2.1 Hybrid Spatial Semantic Hierarchy

The Hybrid Spatial Semantic Hierarchy (HSSH) software was developed in the UT Intelligence Robotics Lab [4]. HSSH builds local perceptual maps (LPMs) through SLAM methods within small spaces defined by the range limits of the robot's sensors and builds upon them to represent the large-scale space. When run, the software connects to the Player server running on the robot and brings up a graphical display of the local perceptual map. The robot starts at the center of the LPM and the map is built around it. In the figure below of the HSSH software user interface, white signifies known free space, grey signifies that the probability of the space being occupied is unknown, and the red is the endpoints of the laser rangefinders' readings. The user can define a goal for the robot by clicking on any open space in the map, such as the room beyond an open doorway in front of the robot.

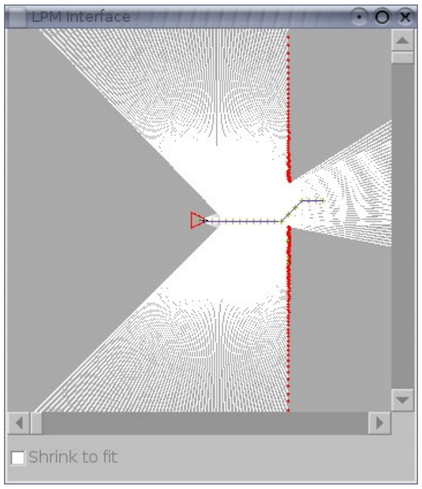

Figure 1. HSSH User Interface

When a goal is chosen, the software determines whether it is possible to attain the goal given the current occupancy grid and forms a plan to reach the goal. The current code for planning the robot's path to the goal within the LPM is very simple since HSSH is still in development. The path behavior consisted only of moving forward in a straight line, stopping, turning, stopping, moving forward and so on until it reached the goal. A sample plan can be seen in the figure above as a gray line. When the robot reaches a kink in the line, it stops and turns to align with the next segment in the line. Because of constant stopping and starting, this rudimentary collision avoidance method requires much more time than necessary and results in a less graceful movement than we would hope for. So, it is obvious that a quicker, more graceful approach is necessary.

2.2 Dynamic Window Approach

According to Fox, Burgard, and Thrun, the most common approach to collision avoidance is to determine the motion direction needed to move towards a target and then generate steering commands to get the robot moving in that direction. However, the commands generated could become extremely large depending on the desired direction, causing the robot to try to accelerate at a rate it is not capable of in order to achieve the commanded speeds. A robot's motors have limited torque, and thus have limitations on acceleration. The dynamic window approach takes into account the acceleration limitations of the robot, and thus, does not send velocity commands to the robot that are physically impossible.

For this approach, it is assumed that velocity can only be changed after constant time intervals (every .25 seconds, for example). So, during every time interval, the translational velocity (v) and rotational velocity (w) of the robot are held constant. Each time interval can be described as having a velocity pair (v,w). Because the velocity pair is constant during the interval, the trajectory of the robot during each time interval is a circular arc. Therefore, the trajectory of the robot is a sequence of circular arcs, depending on what velocities it traveled at during each interval.

This velocity model helps to define the "dynamic window". The dynamic window is first defined as the set of velocity pairs that can be reached within the next time interval at acceleration somewhere between maximum acceleration and maximum deceleration. In order to reduce this set, only "admissible velocities" are kept within the set. A velocity pair is considered "admissible" if the robot is able to stop completely using maximum deceleration before it reaches the closest obstacle on the resulting trajectory. In reducing the total search space of all velocities to the dynamic window search space, our computation becomes much simpler.

In order to select the best velocity pair from the dynamic window search space that will result in a quick movement towards the target without colliding with an obstacle, we score each velocity pair using a scoring function and select the pair with the best score as the new velocity command to the robot. This is repeated after each time interval.

The scoring function used in the dynamic window approach is based on three criteria: heading, clearance, and velocity. Heading is the projected alignment of the robot towards the target after the next time interval traveling at (v,w). Heading encourages moving directly towards the target and results in a straight-line trajectory to the target if there are no obstacles. Clearance is the distance to the closest obstacle on the projected trajectory traveling at (v,w). Clearance ensures effective obstacle avoidance, since the robot's desire to move away from an obstacle increases as it approaches it. This criterion results in a smooth arc around obstacles. Velocity is simply a projection on the translational velocity and encourages quick movement of the robot. All three criterion functions return a value within the interval [0,1]. As shown in Equation (1), each criterion is weighted and all the weighted resulting values are added together. Fox, Burgard, and Thrun use the following values: $\alpha = 0.8$, $\beta = 0.1$, and $\lambda = 0.1$. The weight values must satistify Equation 2 so that *G(v,w)* always returns a value within the interval [0,1].

$$G(v,w) = \alpha \cdot heading(v,w) + \beta \cdot dist(v,w) + \lambda \cdot velocity(v,w) \tag{1}$$

$$\alpha + \beta + \lambda = 1 \tag{2}$$

2.3 Vector Field Histogram Approach

The Vector Field Histogram approach manipulates data in three sequential steps. First, it takes raw sensor reading data to create the histogram grid to describe the robot's environment in detail. Second, it uses the histogram grid to create the polar histogram. Finally, it uses the polar histogram to find new velocity commands for the robot.

The histogram grid is a grid of the environment made up of cells. Each cell (i,j) of the grid holds a value representing the confidence of the algorithm of the existence of an obstacle in that cell in the environment. For each sensor reading, the certainty value of the cell where the range of the reading lies is incremented. The certainty value of a cell is never decremented. Therefore, a high cell value implies a high probability of the existence of an object.

After updating the histogram grid with each sensor's range reading, we define a subset of the histogram grid that will be used in the future: the "active region". The active region is a square window of cells centered on the robot. Within this region, each cell that has a non-zero certainty value is then treated as an obstacle vector with direction and magnitude. The direction of the obstacle vector is the direction from the cell to the robot. Equation 3 gives the direction of cell *i,j* using the x and y coordinates of the cell and the robot:

$$\beta_{i,j} = \tan^{-1}\frac{y_{cell} - y_{robot}}{x_{cell} - x_{robot}} \tag{3}$$

The magnitude of the obstacle vector is based on the cell's certainty value and the distance from the cell to the robot. Equation 4 gives the magnitude of cell *i,j*:

$$m_{i,j} = (c^*_{i,j})^2(a - bd_{i,j}) \tag{4}$$

where $c^*_{i,j}$ is the certainty value of the cell, *a* and *b* are positive constants, and $d_{i,j}$ is the distance between the cell and the robot. Borenstein and Koren chose *a* and *b* such that $a - bd_{\max} = 0$ and $d_{\max}$ is the distance from the robot to the edge of the active window.

Once the active region is ready, the polar histogram is created. The polar histogram consists of n angular sectors around the robot's current location of width α degrees each. The resolution α is chosen such that n = 360 / α is an integer. Borenstein and Koren use α = 5° and n = 72. Each sector then covers angles n*α to (n+1)*α. Each sector of the polar histogram holds the polar obstacle density value for that sector. The polar obstacle density (POD) tells us how populated with obstacles that sector is. Specifically, the POD (*h*) of a sector is the sum of the magnitude of the obstacle vectors of the active region cells that fall within that sector, as shown in Equation 5.

$$h_k = \sum\nolimits_{i,j} m_{i,j} \tag{5}$$

The polar histogram is used to find the new direction of the robot. A typical polar histogram has peaks, which are contiguous sectors with high POD values, and valleys, which are contiguous sectors with low POD values. The Vector Field Histogram approach sets a threshold such that valleys that fall below this threshold are deemed free space that is safe for robot travel. These valleys are called candidate valleys. The candidate valley that most closely matches the direction to the target is selected for the new direction of the robot. The exact new steering direction is the center angle of the selected candidate valley.

After the new steering direction is chosen, the rotational velocity, *w*, for the robot will be the difference between the current angular pose of the robot and the new steering direction. After the new rotational velocity is chosen, the new translational velocity for the robot must be chosen. Before the algorithm starts running, a max translational velocity for the robot is set. The robot will maintain this max velocity unless it is forced to slow down in order to turn away from nearby obstacles, so the robot will always move as quickly as possible. The translational velocity is reduced from the set maximum speed in two ways: by using the POD of the sector of the current direction to slow down near obstacles (see Equation 6) and then by anticipating a change in rotational velocity in order to turn away from nearby obstacles (see Equation 7). The robot's translational velocity will never decrease to zero because of the use of $v_{\min}$ in Equation 7.

$v' = v_{\max}(1 - \frac{\min(h_c, h_m)}{h_m})$ where h_c is the polar obstacle density of the robot's current direction and h_m is a constant. (6)

$v = v'(1 - w / w_{\max}) + v_{\min}$ where *w* is the robot's current translational velocity. (7)

3 Theoretical Comparison

3.1 Strengths and Weaknesses

The Dynamic Window approach has several strengths. Because the objective function incorporates the projected heading and velocity of the robot as well as the distance to the nearest obstacle along the projected trajectory, it encourages quick, smooth movement around both static and dynamic obstacles. The approach also takes into account physical limitations of robot. Through incorporating the robot's maximum accelerations, it will never generate a velocity command beyond the robot's motor torque limitations.

The Vector Field Histogram approach also has the strengths of the Dynamic Window approach. By eliminating sectors that have a high obstacle density as possible steering directions and by setting the velocity at the maximum velocity whenever deemed possible, this approach encourages quick, smooth movement around obstacles as the Dynamic Window approach does. By setting incorporating the minimum and maximum translational and rotational velocities of the robot, the approach takes into account its physical limitations. The Vector Field Histogram approach also has several strengths of its own. Because of the ability to identify extremely narrow valleys in the polar histogram and to select a centered path through it, it has the potential to navigate through extremely narrow passages. The Dynamic Window approach will not necessarily test a velocity pair that will result in a centered path through a narrow passage. The approach also guarantees that the robot will move into free space in the environment because it only selects a steering direction towards areas with obstacle density below a set threshold. The Dynamic Window approach does not necessarily guarantee steering only towards free space, because obstacle clearance is only a weighted factor.

The two approaches have some weaknesses in common. Both only involve very short term path planning and do not attempt to find an optimal path. Therefore, they do not determine when a goal is impossible and stop the robot. The robot instead ends up going in circles trying to reach goal. Neither approach takes into account the goal orientation of the robot at the target point. Both approaches are also subject to getting the robot stuck in a dead-end situation. However, some of these weaknesses can be easily overcome. Through integrating into global path planning software such as HSSH, we can determine first whether a path towards the goal from the current pose is possible before turning over control to the collision avoidance approach. In both approaches we can detect whether the robot has traveled into a dead end and simply rotate the robot away. In the Dynamic Window approach, none of the trajectories generated by the velocity pairs allow the robot to move. In the Vector Field Histogram approach, all of the sectors in front of the robot have high obstacle densities while the sectors behind the robot have relatively low obstacle densities.

3.2 Complexity Differences

The computational complexity of a collision avoidance method is important. If the robot's view of and pose in the environment changes too drastically while the new robot velocity is being computed, the new velocity may not be effective for the robot's new pose in guiding the robot towards the goal while avoiding obstacles. Hence it is essential for the computational time required by the approach to generate a new velocity for the robot to be small enough in order to keep up with the robot's travel through the environment. In comparing two approaches which may be equally effective in collision avoidance assuming zero computation time, we must choose the approach which in reality requires less computation time than the other. To compare the computational complexity of the two approaches considered in this paper, we first compare the computational complexity of their access to the occupancy grid of the environment.

The Dynamic Window approach accesses the occupancy grid to determine where the closest obstacle lies along its projected trajectory for a particular velocity pair. Assuming that cells in the grid are checked for occupancy at least 1 m along the trajectory and we check cells up to (*robotwidth*/2) m on either side of the trajectory line to accommodate for the width of the robot, the area of the grid that is accessed is at least about *robotwidth* m^2. The robot used by Fox, Burgard, and Thrun is an RWI B21, which has a diameter of 0.53 m in diameter. Therefore, the portion of the grid accessed each time is at least 0.53 m^2. This approach must then access the grid for each velocity pair. Each velocity pair checks a different portion of the grid since each pair results in a different trajectory. For example, if 100 velocity pairs are considered, the grid is accessed 100 times. The range of velocities we consider and the time interval used determines the number of velocity pairs we generate. The range is the maximum acceleration multiplied by the time interval above and below the current velocities. In other words, the range is 2 *x* max acceleration *x* time interval centered about the current velocities. Fox, Burgard, and Thrun use maximum accelerations of 20 cm/sec^2 and 30 deg/sec^2 (or $\pi/6 \approx 0.52$ rad/sec^2) and a time interval of 0.25 sec. Hence, the range of velocities considered is 0.1 m/sec and approximately 0.26 rad/sec centered about the current velocities. Assuming we use intervals of 0.01 m/sec and 0.02 rad/sec in generating velocity pairs, we are considering (0.1 m/sec) / (0.01 m/sec) = 10 translational velocity value possibilities and (0.26 rad/sec) / (0.02 rad/sec) = 13 rotational velocity value possibilities. Thus, we generate 130 velocity pairs at any current velocity.

The Vector Field Histogram approach accesses the portion of the grid that lies within the active window, turns each non-zero cell in that portion into an obstacle vector, and uses those obstacle vectors to compute the polar histogram. Borenstein and Koren use an active window of 3.3 m x 3.3 m, so 10.89 m^2 of the grid is accessed. The worst case is that every cell within the active window is non-zero, which means that the portion of the grid within the active window is accessed twice.

In comparing the two approaches, we see that the Dynamic Window approach checks 68.9 m^2 in total for obstacle occupancy, while the Vector Field Histogram approach checks only 21.78 m^2. Even though the Dynamic Window approach checks a much smaller area of the grid than the Vector Field Histogram approach, it checks a different small area each time for each different trajectory. Assuming that the two approaches use the same cell size for the occupancy grid, the Dynamic Window approach is more than three times as computationally expensive thus far as the Vector Field Histogram approach.

Beyond checking a reduced portion of the grid, the Dynamic Window approach must determine the possible velocity pairs, compute the objective function for each velocity pair, and pick the velocity pair with the best objective function score as the new translational and rotational velocity. The Vector Field Histogram must determine which sectors in the polar histogram are valleys, pick the best valley, and compute the new translational and rotational velocity. While classifying polar histogram sectors as valleys or peaks and determining which valley is the best is more expensive than simply computing the objective function, the objective function must be computed for each of the 130 velocity pairs. Since finding the valleys in the polar histogram and picking the best is not 130 times more expensive than computing the objective function, the Dynamic Window approach is again more computationally expensive.

3.3 Comparison Conclusions

Given the extreme difference in computation time between the two approaches, the Vector Field Histogram approach is far more effective than the Dynamic Window approach. Moving beyond just computational complexity, we have seen that not only does the Vector Field Histogram approach match some of the strengths of the Dynamic Window approach, it has strengths especially needed in tightly constrained environments unmatched by the Dynamic Window approach. While the Dynamic Window approach is shown by Fox, Burgard, and Thrun to allow the robot to obtain a higher speed than by using the Vector Field Histogram in cases where the robot's trajectory spans an entire corridor, the Vector Field Histogram is the better approach for tightly constrained environments where the trajectory is relatively short since rapid calculations and updates of the robot's velocity is extremely important. Hence, I tested only the Vector Field Histogram approach in the conducted experiments.

4 Experiment Details

4.1 Vulcan

The Intelligent Robotics Lab's intelligent wheelchair, Vulcan, was used for experimentation with the implemented control methods. Vulcan is 67 cm in width and 92 cm in length. The robot is equipped with two laser range finders that each return ranges for 180° in 1° increments. The sensors are each angled 45° away from the front of the robot, such that range information is available for 270° around Vulcan. These sensors can detect objects up to 50 m away, but they can detect objects with a reflectance of 20 – 30 % only up to 15 m away. The sensors have a resolution of 70 mm up to 4 meters. The maximum measuring error is 94 mm at less than 2 m and is 131 mm at greater than 2 m away.

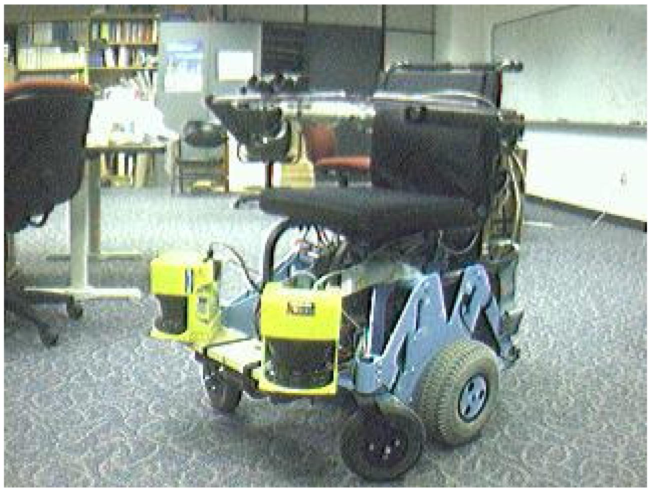
Figure 2. “Vulcan” Robot

Vulcan has an on-board computer running Debian Linux. Most computation is done off-board on another computer connected to Vulcan through wireless ethernet. Vulcan’s on-board computer sends sensory information to and receives resulting motor commands from the off-board computer through the Player server software while a client runs on the off-board computer [3]. The on-board computer acts as the low-level controller for Vulcan and sends the received commands to its motors.

4.2 Hybrid Spatial Semantic Hierarchy Software

Both control methods were integrated into the HSSH software. I replaced the path planning method for paths within the LPM with an implementation of the Vector Field Histogram Approach with the hope that the robot would follow a much smoother and more graceful path to its goal. I took advantage of the occupancy grid and localization functionality already implemented in the LPM methods to provide information about the location of obstacles with respect to the robot’s location to the control methods. The LPM built when HSSH is run is 10 m x 10 m with an occupancy grid cell size of 3 x 3 cm, giving a grid of 333 x 333 cells. Once the control approach returns a translational and rotational velocity command, the HSSH software takes care of sending the command to the robot.

Because the trajectory of the robot in a tightly constrained environment is short, the action of moving through a doorway, into an elevator, or down a wheelchair ramp takes place within the local perceptual map. The Vector Field Histogram is not yet integrated into the global path planning capabilities of the HSSH software. As a result, the robot cannot currently reach a goal point such as the other side of a door that is beyond the LPM.

4.3 Vector Field Histogram Approach

In my implementation of the Vector Field Histogram approach, I used an active window of 4 m x 4 m centered on the robot. Since HSSH uses a cell size of 3 x 3 cm for the occupancy grid of the environment, the active window contains 133 x 133 cells. Borenstein and Koren’s implementation used a window size of 33 x 33 cells with cells of 10 x 10 cm, which gives an active window covering 3.3 m x 3.3 m. A much smaller cell size than 10 x 10 cm is needed for tightly constrained environments, since high precision is needed when calculating where obstacles are.

Borenstein and Koren also smooth each polar obstacle density in the polar histogram by incorporating the polar obstacle densities around it. However, I found that the approach can work successfully and consistently without smoothing. This decreases the computation required by a very small amount, but reduces significantly when implementing the approach for a particular robot the amount of testing needed to find a threshold for determining whether a sector in the polar histogram is free space. Without smoothing, the threshold is simply zero (no obstacles are seen in that sector).

5 Experiment Environments

Three different locations were chosen in the Taylor building at the University of Texas at Austin to be considered as environments that require high precision movement. These environments served as testing for each control approach above. A specific task for the robot to complete was associated with each environment in order to demonstrate the approach's success rate of high-precision control. In each environment, the robot also has a high probability of encountering dynamic obstacles in the form of people, which it must avoid hitting.

4.1 Doorway

Figure 1 shows the doorway to the UT Intelligent Robotics lab at UT Austin. The doorway is 84 cm wide when the door is fully open. When centered in the doorway, Vulcan has a clearance of 8.5 cm on either side. The space on either side of the doorway is clear of static obstacles.

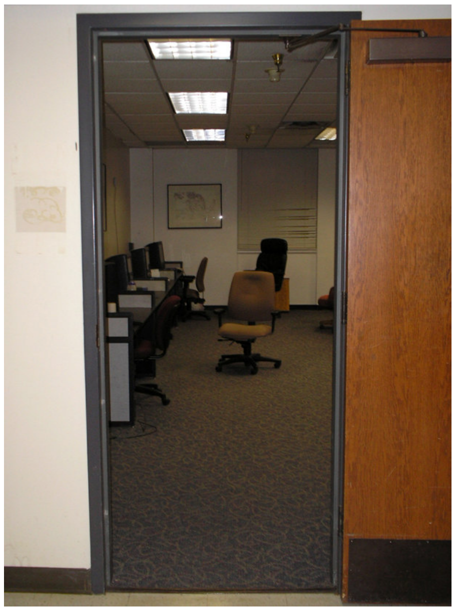

Figure 3. Lab doorway

In this environment, the robot must move from the inside of the lab to the hallway through the doorway. The robot must clear its back wheels through the doorway in order to have successfully completed the task. The high-precision challenge for the robot is avoid the doorway frame on either side of it.

4.2 Elevator

Figure 3 shows the Taylor building elevator on the second floor. The doorway of the elevator is 1.04 m wide and the elevator is 2.3 m deep. Going straight through the elevator doors, Vulcan has a clearance of 18.5 cm on either side. The elevator and the hallway around it are free from static obstacles. When the elevator doors open, they will remain fully open for 5 seconds if they do not detect any movement between the doors.

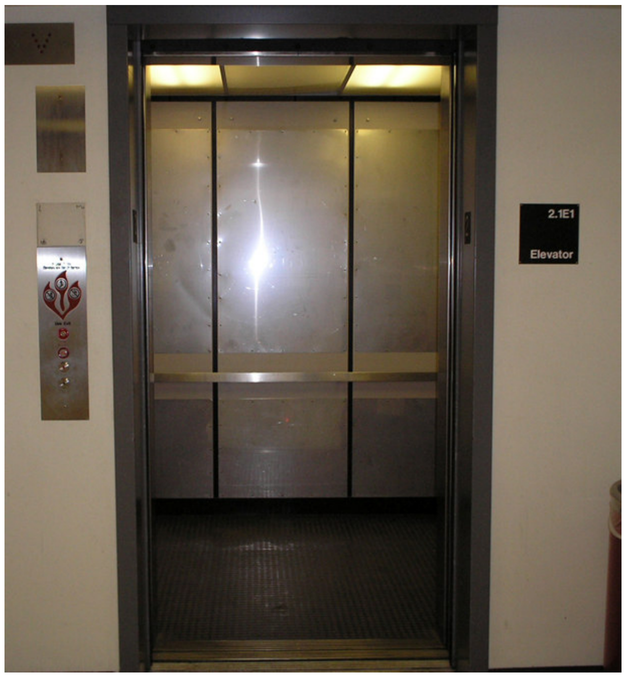

Figure 4. Taylor building elevator

In this environment, the robot must wait in the hallway for the elevator doors to open, move into the elevator, and turn around to face the doors of the elevator. The robot must move completely into the elevator such that the elevator doors are able to close when the robot has turned around in order to have successfully completed the task. The high-precision challenge for the robot is to avoid colliding with either side of the elevator doors and the back wall of the inside of the elevator.

4.3 Wheelchair Ramp

Figure 2 shows the wheelchair ramp from the Taylor building to the ACES building lobby at UT Austin. The ramp is 1 m wide, 3.6 m long, and has a 4.78 degree incline. While following the center of the ramp, Vulcan has a clearance of 16.5 cm on either side. The beginning and end of the ramp are free from static obstacles.

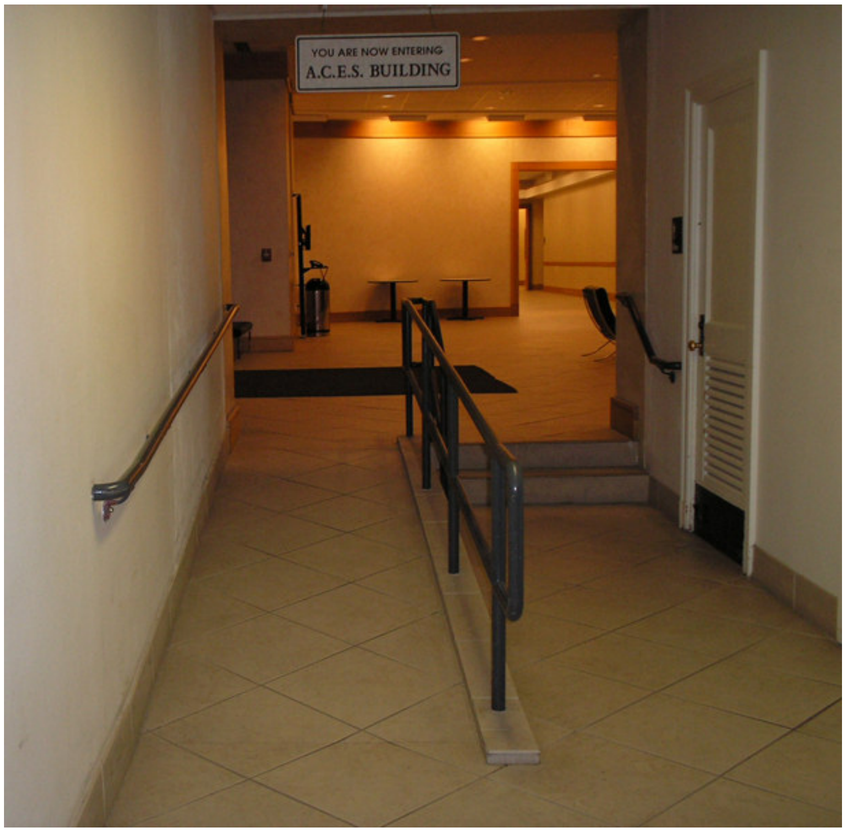

Figure 5. Taylor/ACES wheelchair ramp

In this environment, the robot must move from a location at the bottom of the ramp, onto the ramp, up the ramp, and off of the top of the ramp into the ACES building lobby. The robot must clear its back wheels from the last pole of the handrail at the top of the ramp in order to have successfully completed the task. The high-precision challenge for the robot is to avoid colliding with the wall on the left side and the handrail poles on the right side.

6 Experiment Results

The following sections describe the robot's behavior in the three environments described above. The velocity statistics are taken from the actual odometry changes of the robot, not what the Vector Field Histogram approach returns and what is sent to the robot as a command.

6.1 Doorway

The Vector Field Histogram approach consistently moves the robot through the doorway without colliding with the door frame with an average velocity of 34 cm/sec. During the action, the robot reaches a maximum velocity of 88 cm/sec. The total time required by the Vector Field Histogram approach to compute a new velocity command for the robot is approximately, including updating the LPM, is approximately 0.34 seconds. However, this time is inflated a bit by the test UI that is currently updated each time a new velocity command is computed. The total time to travel from one side of the doorway to the other is typically between 8 and 9 seconds. The path is about 2.7 m long.

The following figure shows the trajectory for Vulcan while moving into the lab. The blue dots are waypoints along Vulcan's path, and the two rectangles show Vulcan's initial and final poses. The other black figures come from the LPM's occupancy grid and give a good idea about what the environment looks like. The square centered on the final pose of the robot is the active window at that point. Through the trajectory, we can see the robot's success in moving along a smooth, centered path through the doorway.

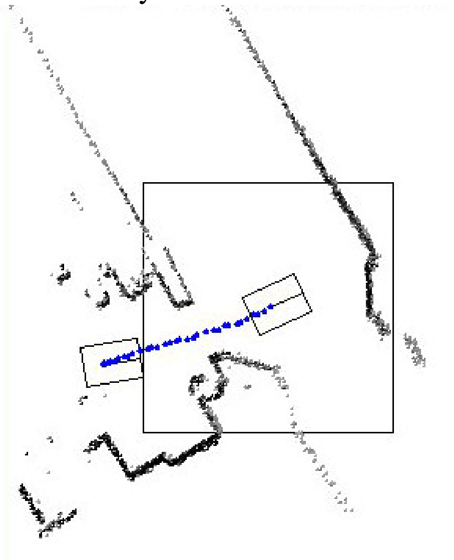

Figure 6. Trajectory through doorway

The following figures show both the commanded and observed translational velocity over time.

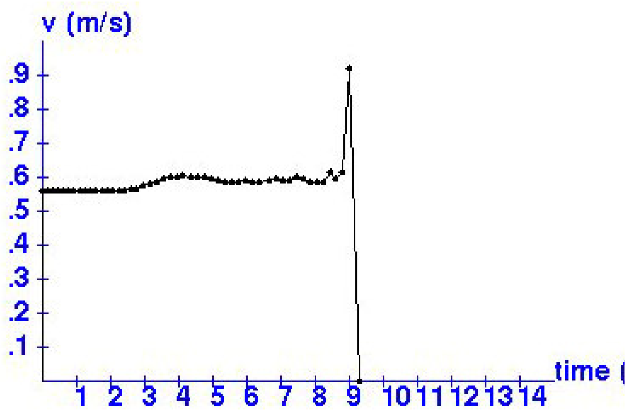

Figure 7. Commanded translational velocity for doorway experiments

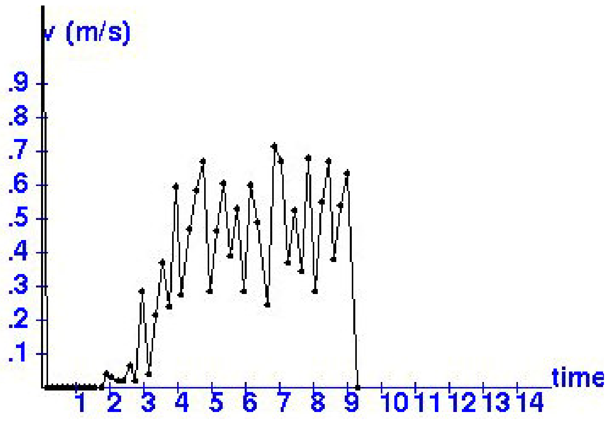

Figure 8. Observed translational velocity for doorway experiments

The graphs show that while the velocity increased smoothly at the beginning, it changed dramatically throughout the rest of the trajectory. The most graceful approach to the doorway would involve a gradual ramp-up of the velocity in the beginning, a fairly constant velocity while traveling through the doorway, and a gradual slowing down at the end. While the Vector Field

Histogram approach sends velocity commands that are fairly constant and close to the maximum velocity while traveling through the doorway, the robot responds to changes in velocity severely. One possible source for this behavior would be error in the recorded position and odometry, since the observed velocity is derived from the changes in position. A small error in the position would be magnified once velocity is derived from it. However, we can look at an example where translational velocity is set at 0.25 m/s and rotational velocity is set to zero, such that the robot travels in a slow, straight line. The following results show that the position does not stray from the expected straight line and the deviation in the observed velocity is still large. The changes in velocity are actually fairly visible when watching Vulcan traveling in a straight line at such a slow speed. So, error in position probably does not account for a large portion of the acceleration we see during Vulcan's movement.

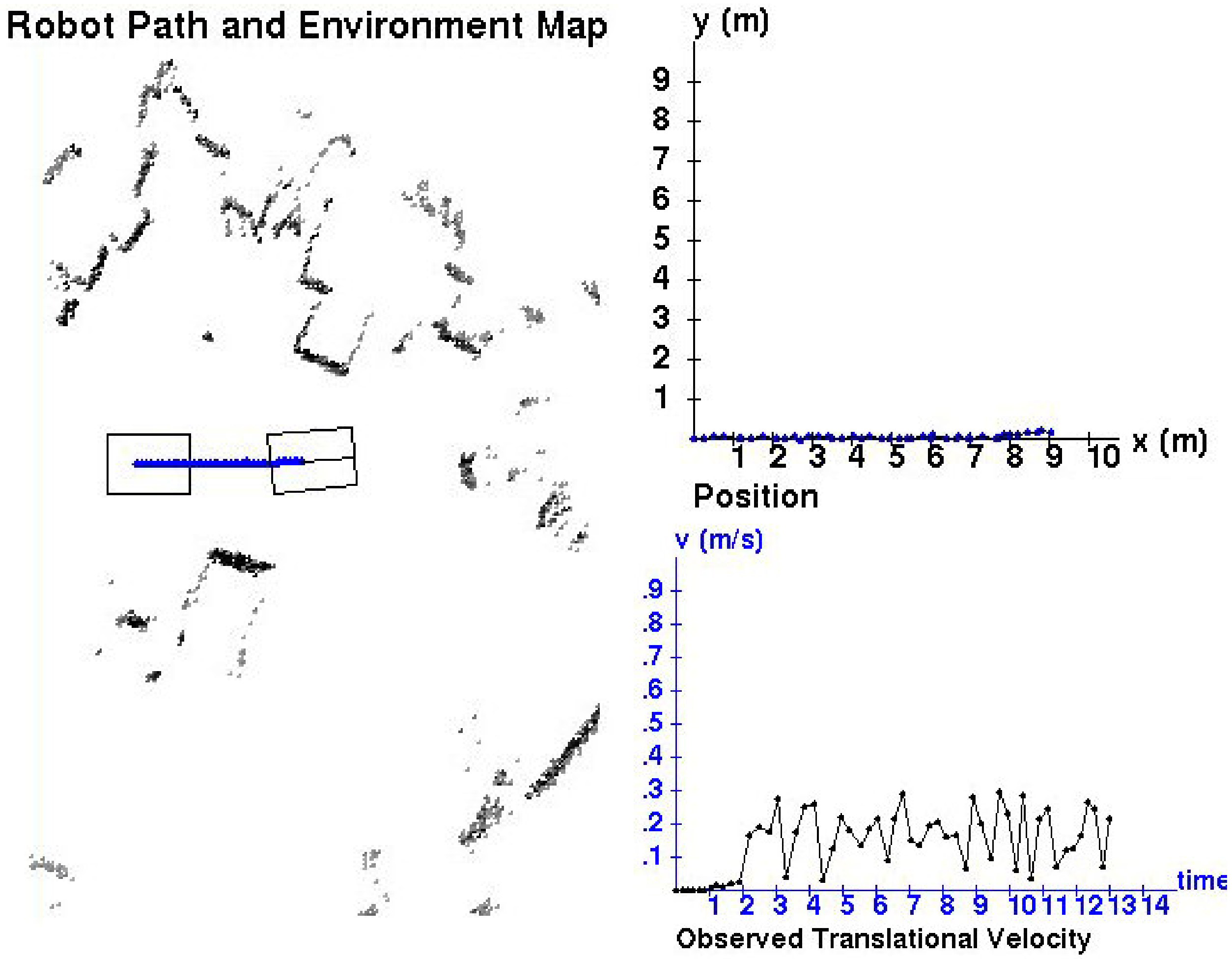

Figure 9. Results from moving in a straight line

The following figures show both the commanded and observed rotational velocity over time.

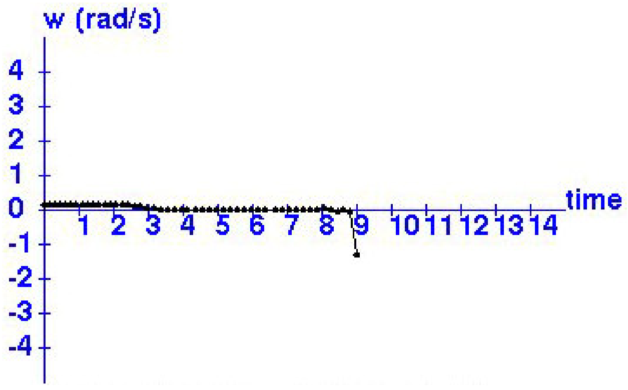

Figure 10. Commanded rotational velocity for doorway experiments

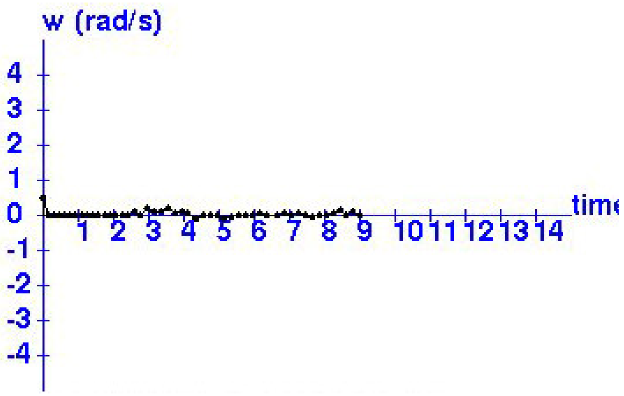

Figure 11. Observed rotational velocity for doorway experiments

In these figures we can see that the robot made smooth, graceful turns when moving through the doorway since there are no large jumps in the rotational velocity graphs until the end when the robot has cleared the door. The large jump at the end of the commanded rotational velocity graph occurs when the robot is very close to its goalpoint and is trying to move onto it exactly. This corresponds to the jump at the end of the commanded translational velocity graph. This seems to suggest that the Vector Field Histogram approach is not necessarily the best approach for moving to a specific point in an environment gracefully, but is somewhat graceful in moving through a tightly constrained environment like a doorway.

6.2 Elevator

Using a clean implementation of the Vector Field Histogram, the robot moves into an elevator with an average velocity of 39 cm/sec. However, it does not always clear the back wheels and move the robot completely into the elevator. Hence, it does not turn completely around and allow the doors to close, as success in this environment was defined in Section 4.2. Once the robot enters the elevator, the back wall enters the active window so the robot will not continue to move towards the back wall. The robot instead starts turning around towards the only sectors in the polar histogram with no obstacles within the active window: the entry. Figure 12 shows this behavior.

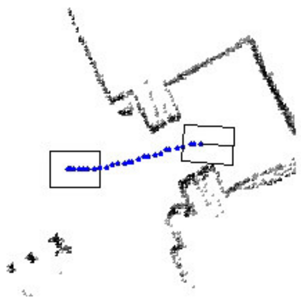

Figure 12. Trajectory into elevator before improvements

In order to successfully move into the elevator, I modified the VFH approach: if the goalpoint is closer to the robot than any of the obstacles in the active window, the polar histogram is ignored and the desired direction becomes the direction of the goalpoint (not the direction of the best candidate valley in the polar histogram). If there is an object that the robot must navigate around, the behavior is the same as the clean implementation, since there is obstacle closer to the robot than the goalpoint.

The following figure shows the trajectory for Vulcan while moving into the elevator using this improvement. We can see the robot's success in moving along a smooth, centered path through the elevator doors. Since the robot moves completely into the elevator, the elevator doors can close. During the task, the robot reaches a maximum velocity of 55 cm/sec and an average velocity of about 35 cm/sec. The total time required to compute a new velocity command for the robot is approximately, including updating the LPM and the testing UI, is approximately 0.22 seconds. The total time to travel into the elevator is typically between 10 and 11 seconds.

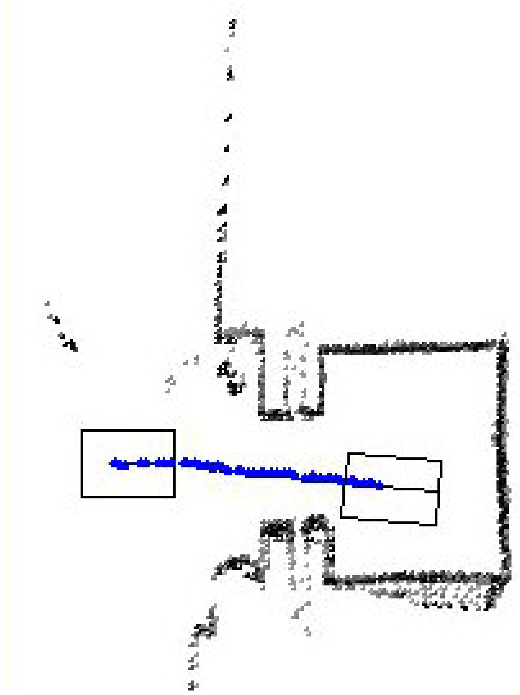

Figure 13. Trajectory into elevator after improvements

The following figures show the translational and rotational velocity and acceleration over time for moving into the elevator.

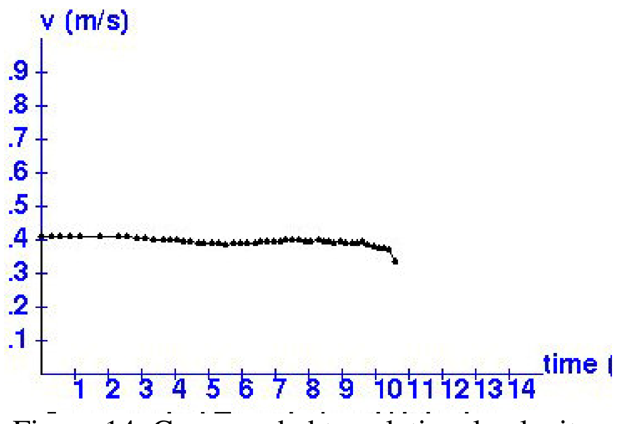

Figure 14. Commanded translational velocity

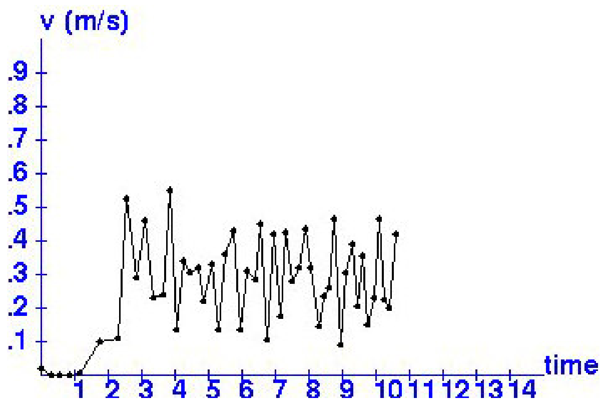

Figure 15. Observed translational velocity for

for elevator experiments

elevator experiments

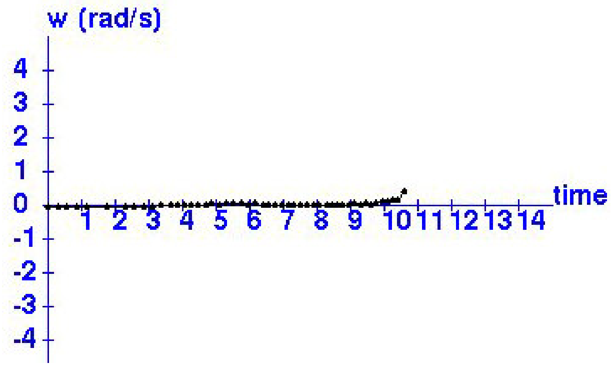

Figure 16. Commanded rotational velocity for elevator experiments

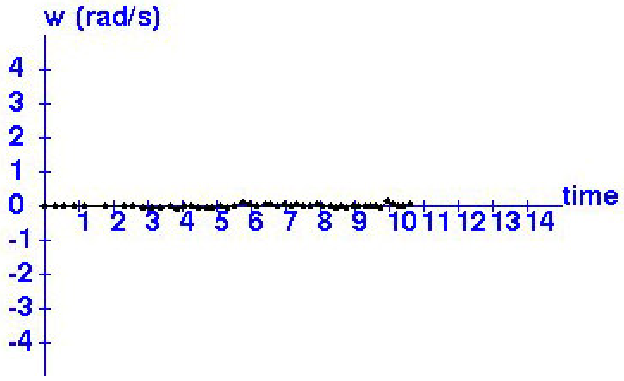

Figure 17. Observed rotational velocity for elevator experiments

The graphs reveal similar results to that of the doorway experiments. The change in translational velocity throughout the trajectory is dramatic, implying that velocity changes in the Vector Field Histogram approach may need to be smoothed. However, the robot made graceful turns, since while the robot was entering the elevator in the middle of the graph, the rotational velocity does not change dramatically.

Successfully moving into the elevator does not complete the task as defined in Section 4.2. The robot must still turn towards the elevator doors. Since the Vector Field Histogram approach does not take into account goal orientation, I implemented a “turning” mode in order to turn towards the doors. In the turning mode, the maximum translational velocity is zero, such that the robot can only turn. In the HSSH UI, the user selects the desired orientation by selecting a point in the graph that is in the desired direction. For example, if we want the robot to face the doorway, we would select a point in between the edges of the doorway. VFH then uses this goalpoint to calculate the appropriate rotational velocity command. The robot stops when it is aligned with the selected goalpoint. In my implementation, VFH alternates between the normal mode and the turning mode when a goalpoint is selected. So, to enter and exit an elevator via Vulcan, the user clicks on a point inside the elevator to enter it, clicks again in the direction of the doorway to turn around Vulcan, and clicks a third time on a point outside the elevator to exit it. The following picture shows what the UI looks like when Vulcan turns around in turning mode.

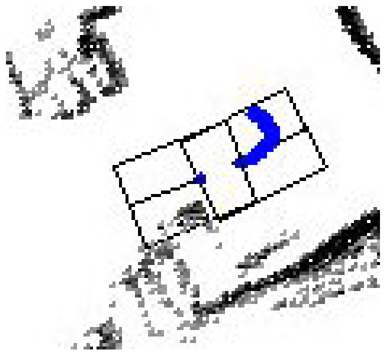

Figure 18. Vulcan turning around

In the elevator experiments, the user must currently take the time to select the point within the LPM in the UI after the elevator doors have opened and the LPM has been updated, so the elevator doors must be manually held open for longer than five seconds. Even if the LPM has a prior map what is on the other side of the elevator doors, HSSH will not accept a point as the goalpoint on the other side of the doors while they are closed so the user must still select the point after the doors have been opened. Since HSSH is still in development, there is no other way to start a control method than to select a point on the UI that HSSH believes can be reached.

6.3 Wheelchair Ramp

Using a clean implementation of VFH, the robot does not avoid obstacles in moving towards the target because it is unable to see the poles on the right side of the ramp. Only a few, sparse lasers hit each pole. Since the cell size in the occupancy grid is so small, each of the lasers hitting a pole can fall in different cells. However, more lasers can pass through the rest of each of the cells making it probable that that cell is free space. Lasers may pass through a cell where another laser hit the pole if the laser hit the pole's edge and the rest of the cell really is free space. They may also be due to small errors in localization. As a result, the poles are washed out from the occupancy grid and the space appears to be free of obstacles. The space is then included in the candidate valleys in the polar histogram when selecting the next desired direction. In trying to attain a goalpoint at the top of the ramp, the robot often moves too far to the right towards the poles and runs into them. In order to overcome this problem, I modified the calculation of the polar obstacle densities. After converting each non-zero occupancy grid cell in the active window into an obstacle vector, I convert the current endpoint of each laser into an obstacle vector. This way, there is at least one obstacle vector for each pole in sight, since at least one laser hits the pole. This modification counts twice non-zero occupancy cells where a laser endpoint falls. This has the effect of reinforcing again the certainty that an obstacle is there. After the modification, the VFH approach effectively keeps the robot from veering towards the poles and keeps it going towards the target. The following figures are examples of Vulcan traveling up the ramp.

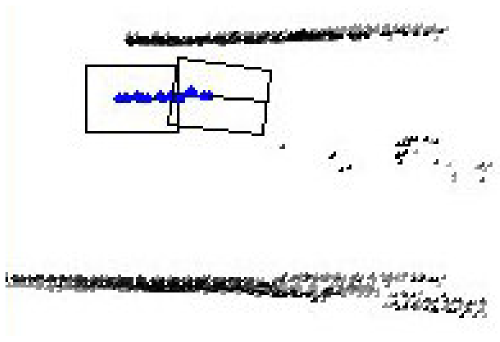
Figure 19. Starting up the ramp

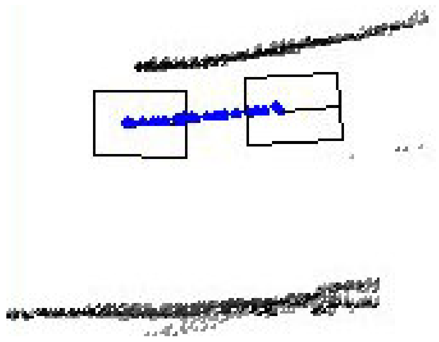
Figure 20. Longer path up the ramp

However, in experiments with the ramp case, I experienced a problem with the robot's localization. When the robot moves onto or off of the ramp, its orientation in three dimensions changes. If the robot approaches the ramp completely head on and the wheels are in parallel with the wall, only its pitch will change. However, if the robot does not approach the ramp with its wheels parallel to the wall, its roll with also change. When the roll changes, the angle of the wall relative to the robot changes. So, a laser will now hit the wall farther away or closer, depending on how the roll has changed. Since we do not take into account the roll or pitch of the robot when mapping laser ranges onto our environment map, the wall will now appear farther away or closer in the map because the laser range has become larger or smaller.

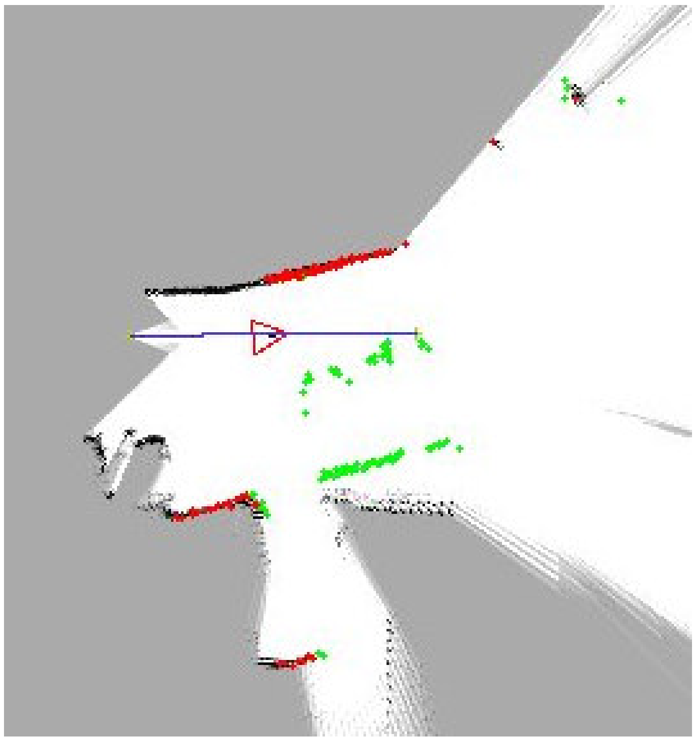
Figure 21. HSSH UI after Vulcan's roll changes

Figure 21 shows the re-localization that occurs after the wall's angle appears to change when the robot's roll changes. Red dots are laser endpoints and green dots are laser endpoints that fall in free space in the occupancy grid, which signifies they are falling on dynamic obstacles. The blue line connects the initial pose of the robot to the goalpoint. In this case, the robot is moving off of the top of the ramp at an angle to the wall. The walls' location in the occupancy grid then changes because of the differing laser ranges that are now being returned. However, the position of the robot and the position of the goalpoint are not re-localized, so the goalpoint is now on top of a pole on the right side, which is obviously impossible to attain.

It is nearly impossible to guarantee that the robot will be entering or exiting the ramp parallel to the wall. Therefore, in order to solve this problem, we need to know how the robot is oriented in 3D, most specifically what its roll orientation is. Given Vulcan's current sensors, this is not possible. We must also determine how to map the laser ranges onto our 2D environment map given the robot's 3D orientation.

7 Future Work

The Vector Field Histogram approach promises to be an extremely useful method for control in tightly constrained environments. However, several steps must be taken before it is truly useful.

First, it must be integrated into HSSH's global path planning capabilities so that it is not reliant upon the user to pick an appropriate goalpoint for the task and so that it is not restrained to goalpoints with the LPM. Another practical improvement would be to automatically select an appropriate goal point based on a given task. For example, if we know that the robot needs to go through a doorway, we can identify where the nearby doorway is and select a goalpoint just beyond it. For the elevator case, we can select the goalpoint as the center of the elevator.

To improve upon the grace of the movement caused by the Vector Field Histogram approach, the approach should be modified in order to gradually increase the speed at the beginning of a task and gradually decrease the speed when within the close vicinity of the goalpoint. The translational velocity commands should also be smoothed, so that the robot does not react so dramatically to changes in velocity.

References

[1] Borenstein, J. and Koren, Y. The vector field histogram – fast obstacle avoidance for mobile robots. *IEEE Transactions on Robotics and Automation,* 7(3):278-288, 1991.

[2] Fox, D.; Burgard, W.; and Thrun, S. The Dynamic Window Approach to Collision Avoidance. *IEEE Transactions on Robotics and Automation,* 4(1), 1997.

[3] Gerkey, B.; Vaughan, R.; and Howard A. The Player/Stage Project: Tools for Multi-Robot and Distributed Sensor Systems. *Proceedings of the 11th International Conference on Advanced Robotics*, 317-323, 2003.

[4] Kuipers, B.; Modayil, J.; Beeson, P.; MacMahon, M.; and Savelli, F. Local metrical and global topological maps in the hybrid spatial semantic hierarchy. *IEEE International Conference on Robotics & Automation*, 2004.