Collision avoidance with Vector Field Histogram+ and Nearness Diagram algorithms implemented on a LEGO Mindstorms NXT robot

Per Eriksson       Felix Foborg       Sofia Lindmark
February 9, 2014

Abstract

In this project the two local collision avoidance algorithms Vector Field Histogram+ and Nearness Diagram are implemented and tested on a robot built from a Lego Mindstorms NXT kit. To create a map of the environment the robot has a rotating sensor tower with four ultrasonic sensors. The robot communicates with a computer through a Bluetooth connection which enables the implementation of more computational demanding algorithms. The time lag introduced from this Bluetooth connection is small compared to making all calculations on the NXT brick. In most cases the robot can navigate on a collision-free path to a goal when using a pre-constructed map of the environment. In real environment tests more collisions occur because obstacles are sometimes erased in the map. In most cases the Vector Field Histogram+ performed better than the Nearness Diagram. Using this algorithm resulted in a smoother path to the goal, a faster travel and less collisions compared to the Nearness Diagram.
1 Introduction

A mobile robot that moves without human control in a cluttered environment have to be able to avoid obstacles. It should also be able to react fast on unforeseen changes.

Two main categories of collision avoidance methods exist: global and local methods. Global methods assume that a complete model of the environment is available. In this case an optimal collision-free path from the start point to the goal point can be computed off-line. These methods usually have problems when the model of the world is inaccurate or when the environment is changing over time. There are methods to update the model based on sensor input, but it is computationally heavy to recalculate the optimal path for each new input. Local methods use only the environment closest to the robot to control the robot. This means that no optimal solution can be found and the robot can be trapped in U-shaped areas etc. The main reason to use local methods instead of global is because of their low computational complexity. This means that even though the environment changes fast, the algorithm still have enough time to recalculate the new collision-free direction. In this project two local collision avoidance algorithms will be implemented.

The purpose of this project is to investigate if existing state-of-the-art collision avoidance algorithms can be successfully implemented on a robot built from a Lego Mindstorms kit and work in a real indoor environment. The robot should be able to travel from a start position to a goal position without colliding with any obstacle and the time to reach the goal position should be low.

The group has done most of the work together or in a collaborative fashion but the members have had responsibility of different parts of the implementation. Per has been responsible for the Vector Field Histogram+ method and the odometry, Felix for the Nearness Diagram method and Sofia has been in charge of the mapping implementation.
2 Theory

2.1 Collision avoidance algorithms

Three state-of-the-art collision avoidance algorithms (Vector Field Histogram+, Nearness Diagram and Dynamic Window Approach) are considered for this project. The algorithms are described in the following sections.

2.1.1 Vector Field Histogram+

The Vector Field Histogram+ method is a real-time motion planning method designed for mobile robots which allows the robot to detect and avoid unknown obstacles without stopping. It was developed in 1998 by Johann Borenstein and Iwan Ulrich [4] as an improvement to the original Vector Field Histogram which was made by J Borenstein and Y Koren in 1991 [3].

In order to use the VFH+ method, a histogram grid has to be created first, which is a map over local environment around the robot created from sensory data. The method then uses this map in a four-stage data reduction process in order to compute the new direction of motion.

1st stage - The primary polar histogram

The first data reduction stage is to map the active region $C_a$ of the map grid $C$ onto the primary polar histogram $H^p$. The map grid $C$ covers the entire region in which the robot can move and keeps track of the number of obstacle detections in each cell. The active region $C_a$ is defined as a circular window centred on the robot with a pre-defined diameter. In order to take the width of the robot into account, the cells in the active region (the active cells) are enlarged, see figure 1.

The data reduction begins by mapping all these enlarged cells in the active window to a corresponding angular sector $k$. The mapping is done by using equation 1. For an angular sector $k$, we sum the value $m_{i,j}$ multiplied by $h'_{i,j}$ for all active cells. Here, $m_{i,j}$ is proportional to the square of the distance between the cell and the robot. The value $h'_{i,j}$ is 1 if the enlarged cells border is touching the current angular sector, and 0 if it is not.

$$H^p_k = \sum_{i,j \in C_a} m_{i,j} h'_{i,j}$$  \hspace{1cm} (1)

2nd stage - The binary polar histogram

In this data reduction step, the previous polar obstacle value is reduced to one of two values, free (0) or blocked (1) based on the value in the previous timestep $n-1$ and two thresholds, $\tau_{low}$ and $\tau_{high}$. This is summarized in equation 2. The purpose of having two thresholds is to prevent the heading of the robot to quickly alternate between for example two narrow openings.
Figure 1: Example of blocked directions. The circles around C1 and C2 are enlarged obstacle cells.

\[
\begin{align*}
H_{k,n}^b &= 1 & \text{if } H_{k,n}^p > \tau_{\text{high}} \\
H_{k,n}^b &= 0 & \text{if } H_{k,n}^p < \tau_{\text{low}} \\
H_{k,n}^b &= H_{k,n}^{b-1} & \text{otherwise}
\end{align*}
\]

\(2\)

3rd stage - The masked polar histogram

The third stage of the data reduction process is where the actual dynamics of the robot is taken into account. First, the trajectory of the robot is approximated as segments of circular arcs and straight lines. Based on the current speed, a maximum turn rate can be found, which corresponds to a circular arc with a minimum steering radius. With the information about the minimum steering radius and the map grid, the masked polar histogram shows which direction of motion are possible at the current speed. This is calculated by first enlarging the obstacle cells just as in the first stage, and then placing two circle to the right and to the left of the robot, as can be seen in figure 1.

The boundary lines of the two circles next to the robot define where the robot can go if it constantly turns at the maximum turn rate. If any of these circles touch an enlarged obstacle, turning in that direction before you have passed the obstacle would therefore result in a collision.
4th stage - Selection of the steering direction

Upon reaching this step, the masked polar histogram will contain all the remaining free angular sectors and from these a set of candidate sectors are determined. If more than a set value $S_{max}$ of sectors are free in a row, it is considered a wide opening. Otherwise, it’s considered to be a narrow opening. A candidate sector $c$ is set as the middle sector in all narrow openings, and for wide openings one candidate sector is set for the left edge, middle and right edge of the opening. If the goal direction is free, the corresponding sector is also set as a candidate sector. A cost function $g(c)$ is then applied to these candidate sectors and the one with the lowest score is chosen as the next direction. The cost function can be seen in equation 3

$$g(c) = \mu_1 \Delta(c, k_t) + \mu_2 \Delta(c, \frac{\theta_i}{\alpha}) + \mu_3 \Delta(c, k_{n,i-1})$$

(3)

where $c$ is a candidate sector, $k_t$ is the sector of the goal direction, $\theta_i$ is the current heading sector, and $k_{n,i-1}$ is the steering sector chosen in the previous time step. The multiplier $\mu_1$ makes the steering goal-oriented and the multipliers $\mu_2$ and $\mu_3$ tries to make the steering smooth and steady. The function $\Delta(c,c_x)$ returns the number of sectors between $c$ and $c_x$. To ensure a goal-oriented selection, equation 4 should be fulfilled.

$$\mu_1 > \mu_2 + \mu_3$$

(4)

2.1.2 Nearness Diagram

The nearness diagram is a reactive collision avoidance method that is designed “for vehicles that move in very dense, cluttered and complex scenario” [1]. It was published in 2004 by Javier Minguez and Luis Montano. The method assumes that the sensory information already is available as a point map of obstacles in an area around the robot. It uses a “divide and conquer” approach where it classifies the current information about the surroundings and the goal location into five different situations, see figure 2. For each of these situations, there is a corresponding action associated with it. The set of situations is complete and exclusive which means that there will be no ambiguity in the action selection. Complete in the sense that the set covers all possible situations that can be detected and exclusive meaning that none of the situations overlap.

The situations are divided by checking if there are obstacles within a security zone around the robot and by using a concept called the “free walking area”. The environment is divided in angular regions by looking for gaps in the obstacle distribution. The free walking area is then defined as the “navigable” region that is closest to the goal. A region is navigable if it seems possible for the robot to get to the goal in that direction. The situation is then classified according to the following criteria, also shown as a decision tree in figure 3:

**Criterion 1:** Are there obstacles within the security zone (low safety) or not (high safety)? The situations in low safety are obtained by criterion 2 and the situations in high safety are classified by criteria 3 and 4.
Criterion 2: Are the obstacles in the security zone only on one side of the sector closest to goal in the free walking area (Low Safety 1 (LS1), figure 2a) or both (Low Safety 2 (LS2), figure 2b)?

Criterion 3: Is the goal inside the free walking area (High Safety Goal in Region (HSGR), figure 2c)?

Criterion 4: Is the free walking area wide (High Safety Wide Region (HSWR), figure 2d) or narrow (High Safety Narrow Region (HSNR), figure 2e) compared to an angular limit?

The navigation actions associated with each situation are geometry-based and consider a circular robot that can move instantly in any direction over a flat surface. The chosen motion command consists of speed $v$ and direction angle $\theta$. The different actions are exemplified in figure 2 and can be summarized as follows:

1. **LS1**: Move away from the obstacle and toward the sector of the free walking area that is closest to the goal.

2. **LS2**: Move in the center of the closest obstacle to the left and to the right of the sector of the free walking area closest to the goal.
Figure 3: The current situation classified using a decision tree with criteria 1-4.

3. **HSGR**: Drive toward the goal.

4. **HSWR**: Move alongside the obstacle closest to goal.

5. **HSNR**: Move toward the center of the free walking area.

### 2.1.3 Dynamic Window Approach

The dynamic window approach is designed to deal with constraints enforced by limited velocities and accelerations of the robot. It was published in 1997 by Dieter Fox, Sebastian Thrun and Wolfram Burgard [5]. The method assumes that information of the position of obstacles in the area around the robot is available. First the algorithm reduces the search space of possible translational and rotational velocities \( (v, \omega) \) to only include admissible velocities within a dynamic window (those velocities that can be reached within the next time
interval). Then the algorithm optimizes an objective function. A more detailed
description of the algorithm follows.

**Search space:** The search space of possible translational and rotational ve-
locities for the robot is reduced in the following three steps:

1. **Circular trajectories:** The dynamic window approach assumes piece-
wise constant velocities, which approximates the trajectories of the robot
by a sequence of circular trajectories (curvatures). These curvatures are
uniquely determined by pairs of forward and rotational velocity \((v, \omega)\).
This results in a two-dimensional search space.

2. **Admissible velocities:** The search space is restricted to admissible ve-
locities which guarantees that only safe curvatures are considered. If the
robot is able to stop before it collides with an obstacle on a curvature, the
对应的 pair \((v, \omega)\) is considered admissible.

3. **Dynamic window:** Given the limited accelerations of the robot a further
restriction, a dynamic window, is imposed on the admissible velocities.
With the dynamic window the robot only considers the velocities that can
be reached within the next time interval.

**Optimization:** The objective function

\[
G(v, \omega) = \sigma(\alpha \cdot \text{heading}(v, \omega) + \beta \cdot \text{dist}(v, \omega) + \gamma \cdot \text{vel}(v, \omega))
\]  

(5)
is maximized. \(\alpha, \beta\) and \(\gamma\) are user parameters and \(\sigma\) is a function that smooth
the weighted sum. The objective function makes a tradeoff between the following
three aspects:

1. **Target heading** (heading): The measure of progress towards the goal.
   It is given by \(180 - \theta\), where \(\theta\) is the angle of the target point relative to
   the robot’s heading direction.

2. **Clearance** (dist): The distance to the closest obstacle on the trajectory.
   This value is set to a large constant if no obstacle is on the curvature.

3. **Velocity** (vel): The forward velocity of the robot.

The function \(\sigma\) results in more side-clearance from obstacles.

2.2 **LEGO Mindstorms**

The used mobile robot is built from a standard Lego Mindstorms NXT kit. The
kit includes a programmable computer (NXT), a set of sensors and motors, and
Lego parts. It is possible to connect up to three motors and four sensors to one
NXT brick.
2.2.1 Ultrasonic sensors

An Ultrasonic sensor measures the distance from the sensor to the closest object in front of it within the lobe. The range of the ultrasonic sensor in the standard Lego Mindstorms kit is 4 to 255 cm. For objects further away or when no object is found it returns the value 255 cm. The precision of the sensor is +/- 3 cm and the lobe is about 30 degrees [8].

2.2.2 Infrared sensors

An infrared (IR) sensor measures the distance from the sensor to the closest object that it is facing. Available IR sensors for this project have a range of 7 to 30 cm or 10 to 80 cm and the precision is +/- 1 mm. The lobe of the sensor is approximately 5 degrees. When no object exists the sensor can easily be disturbed by background noise [8].

3 Implementation

3.1 Robot Construction

We use a robot with differential steering, which means it has a motor connected to each of two wheels and a third wheel for balance. The reason for this design is that it is easy to build and use and implies no constraints on the steering radius which means that the robot can rotate in place. The robot has a rotating sensor tower on top with four ultrasonic distance sensors 90 degrees apart. We chose this instead of fixed sensors because our collision avoidance algorithms require a 360 degree view of the environment and we are limited to four sensors by the NXT brick. The ultrasonic sensors have the disadvantage of a much wider lobe than the infrared sensors but were chosen because they give much more reliable measurements. The infrared sensors often give random distance measurements when there is no object in front of them and are sensitive to varying conditions like sunlight. The robot was constructed using a standard Lego Mindstorms NXT kit and is shown in figure 4.

3.2 Collision Avoidance System

The layout of the collision avoidance system is shown in figure 5 and can be explained in three steps. First the NXT brick program combines readings from the ultrasonic sensors with the angular position of the sensor tower motor to get polar coordinates of the detected objects. At the same time, the brick uses an odometry algorithm based on the tachometer count of the two driving wheels to calculate its position. When this is completed, the updated position and the processed sensor data are sent via Bluetooth to the PC. On average 80 readings per second in total is collected from the ultrasonic sensors.

Secondly, the PC uses the new data to update the world map and displays it on the monitor. The Nearness Diagram algorithm or the Vector Field Hist-
The histogram+ method is then applied to the updated map to calculate the next driving instructions which are sent back to the brick.

In the third and last step the motor control thread on the brick changes the driving angle and speed in accordance with the new instructions.
3.3 LeJos and NXC
At the beginning of the project, NXC was chosen as the programming language and the code was implemented with the BricxCC program. Later on a decision to switch to LeJos was made due to computational benefits. With LeJos it is possible to have a program on the NXT brick and another program on a PC running simultaneously, with the ability to move data back and forth using a Bluetooth connection. With the additional computational power from the PC, advanced algorithms such as the Nearness Diagram and Vector Field Histogram+ methods could be implemented without significant delays except the Bluetooth send and receive delays.

3.4 Odometry
A part of the program running on the NXT brick is a odometry algorithm that estimates the current position and heading of the robot relative to the starting position. It keeps track of the total rotation count for each of the two driving engines, and uses this to calculate the distance the wheels have driven. The algorithm then estimates the distance the robot has moved since the last update according to

\[
\begin{align*}
R &= \frac{d_w}{2} + d_w \text{Min}\{L_a, L_b\}/(L_a - L_b) \\
\phi &= \frac{(L_a - L_b)}{d_w} \\
dX &= R(1 - \cos(\phi)) \\
dY &= R\sin(\phi)
\end{align*}
\]

(6)

where \(d_w\) is the distance between the wheels, \(L_a\) is the distance the left wheel has driven, and \(L_b\) is the distance the right wheel has driven. \(\text{Min}\{L_a, L_b\}\) returns the smaller value of \(L_a\) or \(L_b\).

With \(L_a\) and \(L_b\) a circle is calculated and centred at the point R along the \(\beta\)-axis. Since the last time-step the robot has moved \(\phi\) degrees along the arc of this circle, see figure 6. \(dX\) and \(dY\) are then the shifts in position in this reference frame since the last time-steps.

These shifts has to be translated to the coordinate system for the map, by use of equation 7

\[
\begin{align*}
Y &= dY\sin(H) + dX\cos(H) \\
X &= dY\cos(H) - dX\sin(H) \\
H &= \frac{L_b,\text{tot} - L_a,\text{tot}}{d_w} + H_{\text{start}}
\end{align*}
\]

(7)

where \(Y\) and \(X\) are the coordinates for the current position of the robot in the map and \(H\) is the current heading in the map-coordinate system. \(H_{\text{start}}\) is the starting heading which is predefined as \(\pi/2\).

3.5 Mapping
To use a collision avoidance algorithm a local map of the environment around the robot is necessary. We chose to update a larger world map with the sensor
readings and then use the part of that map closest to the robot as the local map. The reason for this is that it takes too long time to even get one full sensor scan of the environment (90 degrees rotation of the tower) and that we preferably want to use more than one reading in the same direction to make a better estimation. The world map is created from distance measures from the four rotating ultrasonic sensors and built up by 5x5 cm cells. The position of the robot in the map is updated by odometry. The map is then updated with a method described in [7] that combines measurements to take care of erroneous readings. When a sensor finds an obstacle it is assumed to be positioned at the measured distance on the acoustic line of the sensor. That cell value is increased by 3 and all cells on the line between the sensor and the found obstacle are decreased by 1, see figure 7, since there cannot be any object closer. When no obstacle is found all cells on the line up to the maximal range of the sensor is decreased by 1. The minimal value of a cell is set to 0 and the maximal value is set to 16. Since the lobe of the ultrasonic sensors is very wide and therefore enlarges obstacles, an approach with two extra erasing rays on an angle of 9 degrees from the acoustic line were used. These two lines decreases every cell value by 1 up to the measured distance, see figure 8.

3.6 Collision Avoidance Algorithms

3.6.1 Vector Field Histogram+

The design process mostly followed the process outlined in the orginal paper, and some additional information was found in a paper by Babinic, Andrej, et al.
Figure 7: Updating the map. Each cell is 5x5 cm. The obstacle is assumed to be positioned at the measured distance on the acoustic line of the sensor. Its cell value is increased by 3 and all cells on the line between the sensor and the found obstacle are decreased by 1.

Figure 8: Updating the map. Two extra rays on an angle of 9° from the acoustic line are used for each new measurement to decrease more values in the map where it probably is free space. These two lines decrease every cell value by 1 up to the measured distance.

"Navigation of Robot Using VFH+ Algorithm."[6]. Babinec et al. used a static value for the threshold $\tau_{low}$, while we used a more dynamic approach which makes the threshold depend on the sum of all collected m values for a time-step. The reason for this is that the robot stands still for about 2 seconds at the start
of each run, to have some time to create a map before it starts to move. This makes the initial values of the active region very high compared to later on, since the four sensors which are collecting data are quite slow compared to the maximum speed of the robot.

The parameters in the Vector Field Histogram+ method were set as following: The number of angular sectors was set to 60, and an opening was considered narrow if it consists of less than 12 sectors. The size of the active window was set to the smaller of 1400x1400 mm or \((\text{distance to goal} \times 2) \times (\text{distance to goal} \times 2)\), as we wanted this size to be smaller than the sensor range but big enough to avoid obstacles on time. The reason for the non-static window size is that if the robot is close to the goal, we are not really interested in how the environment looks behind the goal. Lets say the goal is very close to a wall. In that case, if the size of the active window was still big, then the algorithm would consider the road to the goal to be blocked.

The enlargement size of the cells described in figure 1 was set to 150 mm in simulations and 200 mm in the real environment. The threshold \(\tau_{\text{high}}\) was set to \(\sum_{i,j \in C_{\text{past}}} m_{i,j} / 150\) and \(\tau_{\text{low}} = 0.95 \tau_{\text{high}}\). The third data reduction step; the masked polar histogram, was skipped by setting the maximum turn angle to \(\pi\), since the robot has a minimum turn radius of approximately zero. The multipliers for finding a new direction was set to \(\mu_1 = 4, \mu_2 = 2\) and \(\mu_3 = 1\).

After deciding the next steering direction, a velocity must somehow also be chosen. This is not a part of the VFH+ method, so we made our own implementation. Details about this implementation can be found in the Motor Control section.

3.6.2 Nearness Diagram

The Nearness Diagram method was implemented as described in the article with some small modifications. Since our robot has a 360 degrees view of the environment and does not have a specific forward direction we do not limit the angle of motion. Even though we use our sensor readings to update a large world map we only use a smaller local map for the navigation algorithm. The reason for this is that we want to avoid using map values outside of sensor range that are not updated and keep the computational time down. The parameter \(p\) that is used in low safety mode to adjust how much the robot should turn away from the closest obstacle is not used since we didn’t understand it completely and it gave a strange behaviour.

The parameters in the Nearness Diagram method were set as following: the number of angular sectors was set to 91 and the radius of the local map to 1 m. A map cell with a value of at least 5 was counted as an obstacle. The radius of the robot was set to 12 cm and the radius of the security zone to 25 cm. The free walking area was considered wide if it covered more than 23 sectors (90 degrees) and narrow elsewhere.
3.7 Motor Control

For this project we choose to use the method TravelArc in the DifferentialPilot class in leJOS for robot steering. The method TravelArc moves the robot along circles which limits the necessary input to only two parameters, velocity and turn radius.

Both the collision avoidance algorithms assume that the robot is holonomic, which means that it can move in any direction instantly, and the algorithms give a new angle of direction $\theta$ as output (compared to the straight forward axis). $\theta$ is positive in the anti-clockwise direction and negative in the clockwise. From this angle the turn radius must be approximated and the velocity chosen. The velocity $v$ is set as

$$v = V_{\text{max}}K\frac{\pi/2 - |\theta|}{\pi/2}$$

where the velocity parameter $K$ is 1 for VFH+, and for Nearness Diagram it is a parameter that limits the speed based on which situation the algorithm is currently in [1]. $V_{\text{max}}$ is the maximally allowed speed defined in each method. The third factor is used because slower and sharper turns are preferable when the change of direction is large.

The turn radius $R$ is given by first calculating the position where the robot would be in $dt = 0.5$ seconds given a movement with direction $\theta$ and velocity $v$. $|R|$ is then the radius of the circle that connects the robot with this calculated position and has its origin along the axis perpendicular to the straight forward axis. $R$ is positive for right turns and negative for left turns. This leads to the expression

$$R = \begin{cases} 
\alpha = 0.5|v|\cos(\theta) \\
\beta = 0.5|v|\sin(\theta) \\
\frac{\alpha^2 + \beta^2}{2\beta} & \text{if } \alpha \neq 0 \\
0 & \text{otherwise}
\end{cases}$$

where $\alpha$ is the coordinate for the straight forward axis and $\beta$ is the coordinate for the perpendicular axis with positive values to the right, see figure 6. If $v \approx 0$ the velocity is set to $V_{\text{max}}/4$ to turn slowly on the spot.

The time $dt$ was chosen as a trade off between smooth behaviour (larger $dt$ = larger radius) and not deviating to much from the preferred angle.

4 Experiments and Results

Experiments were performed to measure the performance of the mapping algorithm, the collision avoidance algorithms on pre-constructed maps and the complete collision avoidance system in a real environment. The reason to test the collision avoidance algorithms on a constructed map is to separate the problems with the algorithms from errors in the map. In the figures for both the simulated tests and real environment tests, the robots driven path can be spotted by the line of circles leading to the goal.
4.1 Mapping

4.1.1 Experiment

The robot was positioned inside a square of walls as in figure 9 without the obstacle. The robot started to scan the environment and after 3 s an obstacle was placed next to the robot and the scanning continued for 3 more seconds. Two different mapping approaches were tested. The first test used the mapping algorithm without the two extra erasing rays while the second used the mapping algorithm with the extra rays.

Figure 9: Testing the mapping algorithm. The robot and an obstacle are placed within a square of walls.

4.1.2 Results

The results from testing the mapping algorithm can be seen in figure 10. To the left only one ray was used in the mapping algorithm and to the right also two extra erasing rays were used. The figure shows that when using the extra erasing rays the found obstacle is closer to it’s real size and there are less errors in the map.

4.2 Simulation

4.2.1 Experiment

To test the performance of the collision avoidance algorithms separately from the mapping performance, two different world maps were pre-constructed and pre-loaded on the PC instead of using sensor data to build the map. The first map had narrow corridors, tight turns and only one possible way to travel to the goal while the second map was larger and had two possible ways to go.
4.2.2 Result

The results for the small map can be seen in figure 11. To the left is a run using the Vector Field Histogram+ (VFH+) method and to the right is a run using the Nearness Diagram (ND) method. The average time to reach the goal over five runs, was 25.1 seconds for the VFH+ method and 32.1 seconds for the ND method (maximum speed was set to 250 mm/s).

The results from the larger map, can be seen in figure 12. The average time to reach the goal was 71.5 seconds for the VFH+ method. The ND method did not manage to navigate to the goal because it got stuck in the large open area of the map.

4.3 Real environment tests

4.3.1 Experiment

Two different tracks were used when testing the algorithms in a real environment, one easy and one difficult. The tracks can be seen in figure 13 and the goals are marked with white crosses. 10 trials were performed for each track and algorithm and the success percentage was measured. For the successful trials an average runtime was calculated. The maximum velocity of the robot was set to 250 mm/s for all trials. To see how well the system performed when the environment changed over time an extra robot driving back and forth along the blue line in figure 13 was added to the difficult track in a third test.
4.3.2 Results

Videos for the different algorithms and tracks can be found here: http://www.youtube.com/playlist?list=PLozfQr0eKjY2vousxDoJATko8IhPUCQ!5

The success percentage and average runtime for each track and algorithm can be seen in table 1 and 2 respectively. The reasons for colliding was in all cases that an obstacle or part of an obstacle was erased by the mapping algorithm. This happened especially with corners of obstacles that was erased by the extra erasing rays or walls at large incident angles where the ultrasound wave never got reflected back. The results show that the real significant difference is that ND has a lot worse success performance on the difficult track. On the difficult track errors in the positioning estimate showed and the goal was believed to be reached up to one meter away from the true goal. These errors were almost always larger when using the Nearness Diagram algorithm than the Vector Field Histogram+.
Figure 12: Result from simulating the collision avoidance methods on the large map. Results from the VFH+ method is shown to the left and from the ND method to the right. The blue, red and yellow lines are only visual features of the ND algorithm.

Table 1: The success percentage for the different tracks and algorithms.

<table>
<thead>
<tr>
<th>Track</th>
<th>VFH+ (%)</th>
<th>ND (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Easy track</td>
<td>80</td>
<td>90</td>
</tr>
<tr>
<td>Difficult track</td>
<td>80</td>
<td>40</td>
</tr>
</tbody>
</table>

An image of the map after one successful run from each algorithm using the difficult track can be seen in figure 14. Due to the mapping algorithm some of the detected values that existed earlier during the run have been erased before reaching the goal. For example in the run done by the VFH+ algorithm (to the left in the figure) the last wall next to the goal is still present while it has been completely erased in the run done by the ND algorithm (to the right in the figure). Judging from the outline of the ND run we can see that it was present earlier or else the line of green circles would have followed the yellow solid line.
Figure 13: Left - the difficult track with the moving robot shown. Right - the easy track.

Table 2: The average time to reach the goal for the different tracks and algorithms.

<table>
<thead>
<tr>
<th></th>
<th>VFH+ (s)</th>
<th>ND (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Easy track</td>
<td>15.6</td>
<td>12.4</td>
</tr>
<tr>
<td>Difficult track</td>
<td>30.9</td>
<td>31.2</td>
</tr>
</tbody>
</table>

Trials with a moving robot as obstacle in the difficult track showed that both algorithms handled a dynamic environment pretty well and avoided the robot. On the other hand the collision avoidance system didn’t account for the movement of the object at all. It didn’t care about the direction of the motion and the small obstacle robot could for example make our robot turn around because the entire path to goal seemed blocked.
5 Discussion

With our design the robot could almost always navigate on a collision-free path in the simulated maps. With the larger map the ND algorithm had problems reaching the goal, but it still avoided the walls. In the real environments the collisions occurred because the obstacle did not exist in the map so the mapping capability showed to be our biggest limitation.

5.1 Mapping

Objects found by an ultrasonic sensor can be positioned anywhere on the measured distance within the lobe of the sensor. Since ultrasonic sensors have a wide lobe this results in a problem of enlarged objects. To handle this we used three erasing rays in the mapping algorithm instead of just one. For a stationary robot this gave a map looking closer to the true environment, see figure 9, but when this design was used together with a moving robot we realised some
problems. When the incident angle between the ultrasound wave and a plane wall is large, only a small amount of the energy in the sound signal sent from the ultrasonic sensor comes back and the wall will probably be undetectable. When no object is detected, the mapping algorithm starts to erase objects up to the range of the sensor. With three rays this erasing goes faster. We also have problems with erased corners of obstacles for similar reasons. When the incident angle on a rectangular object is pretty large the sensor will sometimes find a part of the wall that is further away and therefore erase the closer corner with it’s side rays. Because of these reasons it might have been better to use only one ray, but three rays still have the advantage that objects are not as enlarged as they would be with only one ray so narrow openings can be detected by the robot. In environments with a lot of obstacles, an improvement of the mapping algorithm might be to ignore the measurements when no object is found since this probably happens because of an undetected obstacle.

Another possible design that might improve the mapping performance would be to have a tower with two pairs of an IR sensor and an US sensor looking in the same direction. The readings can then be combined to compensate for their respective weaknesses (erroneous readings in empty sectors versus wide lobe). Since we get so many sensor readings per second we could speed up the sensor tower and therefore not be effected too much by the decreased sensor view.

5.2 Nearness Diagram

From the results on the large pre-constructed map we noticed that the Nearness Diagram algorithm could get stuck in an open area that was larger than our local map. The reason is that it takes a new motion decision each time independent of the previous decision and could therefore go back and forth between the true opening to the goal and a blocked path where the obstacles get in and out of range. This was never a problem in our real environment tests since our tracks were more narrow than the chosen local map size.

Especially from trials on the small pre-constructed map we noticed a few times that the robot drove straight into the wall in low safety 2 mode. The reason is that the chosen motion is in the center of the two closest obstacles in the direction closest to goal. This direction could very well be toward another obstacle. A solution to this would be to consider more than two of the closest obstacles.

5.3 Comparison VFH+ and ND

The real environment test showed that the VFH+ method performed a lot better than ND on a more difficult track. It was especially more robust against incorrectly erased obstacles in the map, perhaps mainly since it uses the distance information of all found obstacles in a sector and not only the closest one as ND. That VFH+ favors the current direction also make it less sensitive to suddenly erased obstacles. In addition VFH+ handles open spaces larger than the local map better than ND as seen in the large pre-constructed map. This
is also because it opts to keep it’s current direction and therefore continues on it’s path until it’s considered completely blocked. We could also notice that VFH+ gave a smaller error in the odometry because it changes directions more smoothly and therefore does not slip as much. ND was a little faster and better at the easy track since it does not react on obstacles if it safely can drive in between. Another aspect that could be important is that ND was a lot more complicated and time consuming to implement.

Overall VFH+ outperformed the ND method and was easier to implement. ND might be able to navigate better in more complicated environments thanks to it’s divide and conquer approach but that did not show in our tests and with our mapping limitations.
6 References

References


