**Umeå University**
**Department of Computing Science**
**Ola Ringdahl**
**ringdahl@cs.umu.se**

**10th April 2003**

# Path Tracking and Obstacle Avoidance Algorithms for Autonomous Forest Machines

**Master Thesis**

Tutor: Thomas Hellström

# Contents

# List of Figures

# Abstract

The work presented in this thesis is a part of the IFOR project at Umeå University [Hel02]. It consists of implementing a simulator for an autonomous forest machine (a forwarder). The simulator is able to use *Follow The Carrot* [Bar01] and *Pure Pursuit* [Cou92] to follow a predefined path. To avoid obstacles it can use *VFH* [BK91] or *VFH+* [UB98] methods. The simulator has a Graphical User Interface so the user can make all necessary adjustment in an easy way. The actual simulation is also presented graphically. The simulator is built using Matlab.

# Chapter 1

# Background and introduction

The IFOR project *Autonomous Navigation for Forest Machines* at Umeå University [Hel02], has the goal to develop an unmanned vehicle that transports timber from the area of felling to the main road for further transportation. To do this, algorithms for navigating in a forest environment must be implemented in a forwarder. The idea is that a human driver drives along the chosen path, while the vehicle records the journey. After that, the vehicle automatically travels along the path, avoiding any obstacles on the way.

The purpose with this project is to implement a simulation environment in Matlab for development and evaluation of algorithms for *Path Tracking* and *Obstacle Avoidance*. The vehicle that is to be simulated is an autonomous forest machine (a forwarder). The vehicle shall be able to drive between the forest and a nearby road by following a predefined path and avoid obstacles on the way.

The most important difference between a forwarder and an ordinary vehicle (a car for instance), is the articulated steering. This means that the vehicle is divided in two parts joined in the middle. When steering, the front and rear part of the vehicle moves toward each other, forming a part of a circle. The size of a forwarder differs, but the one seen on the front page is about ten meters in length and three meters wide.

# Chapter 2

# The simulator

In order to simulate the forwarder, some computations must be done. First the turning radius of a given steering angle is determined, and then the position where the vehicle will be after a time $\Delta t$ is calculated. Now it is just a matter of plotting the vehicle at the coordinates $(x, y)$, with an orientation $\theta$ and steering angle $\phi$. Two different coordinate systems exist in the simulation environment, the global and the vehicle coordinate system. The global system is defined in the ordinary coordinate system with xy-axis as usual, while the vehicle system has its y-axis in the direction of the vehicles orientation. The latter coordinate system is used in the *Pure Pursuit* method.

## 2.1 Definitions

In order to calculate the turning radius for a given steering angle we must first define what we mean by steering angle, see Figure 2.1. The steering angle $\phi$ is defined as the angle between the front and rear sections of the vehicle, as in the left part of Figure 2.1. In the simulator, however, we use the angle between the baseline and each section, as in the right part of Figure 2.1, i.e. $\phi/2$.

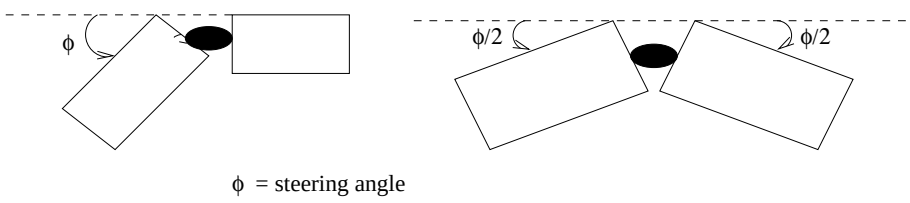


Figure 2.1: Definition of the steering angle $\phi$

The orientation $\theta$ of the vehicle, defined in the global coordinate system as shown in the left part of Figure 2.2, is the direction the vehicle would travel in if the steering angle was zero. It can be seen as the heading of the front part of the vehicle minus half the steering angle, see the right part of Figure 2.2.

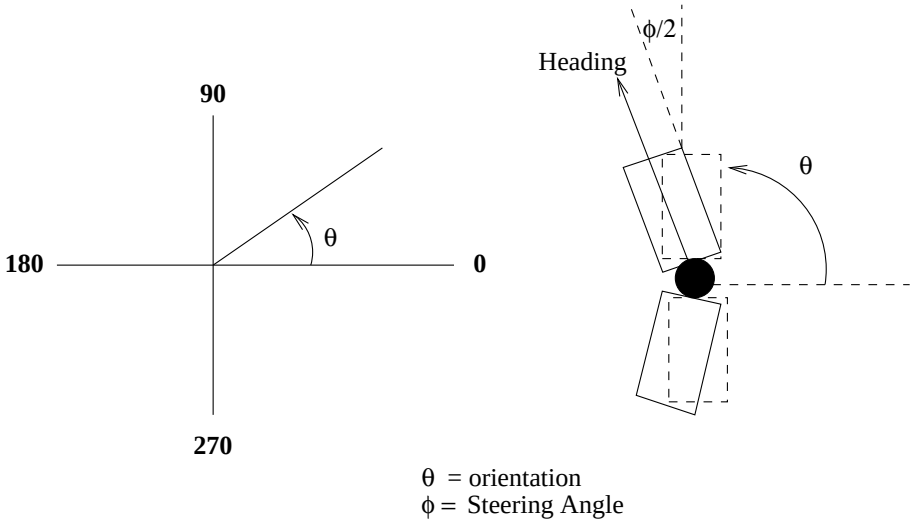


Figure 2.2: Definition of the vehicle's orientation $\theta$.

## 2.2 Calculation of the turning radius

Figure 2.3 shows the geometry involved in calculating the turning radius for a given steering angle. Sub-figure A shows the circle traversed by the vehicle when turning, and sub-figure B shows the angles and distances the calculations are based on. The algorithm for calculation of the turning radius, $r$, is the following:

1. Assume that the vehicle initially is positioned straight forward with zero steering angle (i.e. the front and rear sections are in a straight line).

2. Turn the front and rear end an angle $\phi/2$ respectively from the start position.

3. Let the distance between the vehicles center joint and the wheel axis be *length*. Since the rear (and sometimes even the front) section of the vehicle has two axis, a virtual axle located between the two axis is defined (however, in the simulator it is assumed that the axis are located in the middle of respective section due to limitations in the plotting of the vehicle sections). The radius

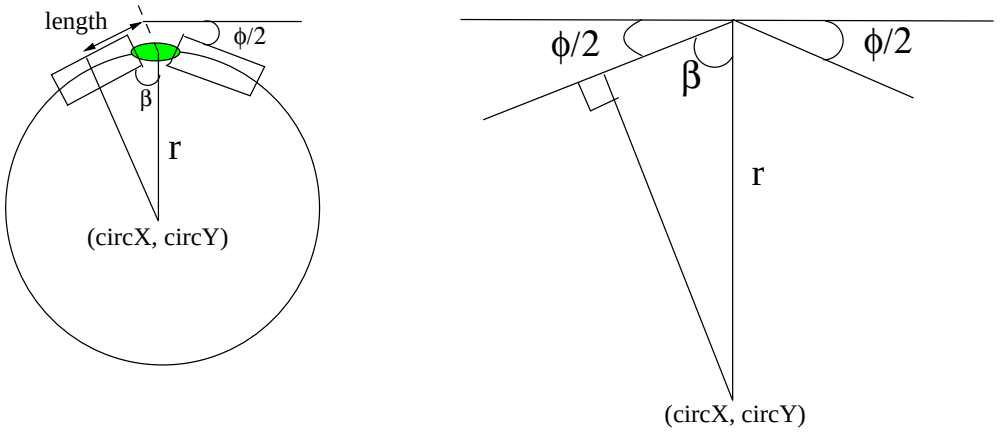


Figure 2.3: Calculation of the turning radius for a certain steering angle.

can now be computed:

$$r = \frac{length}{\cos(\beta)}. \tag{2.2.1}$$

where

$$\beta = \frac{\pi}{2} - \frac{\phi}{2} \tag{2.2.2}$$

## 2.3 Pose after movement

The pose of the vehicle is the position $(x, y)$ and the orientation $\theta$. To decide the pose of the vehicle after a time $\Delta t$, we have to determine the angle moved $\delta$, see Figure 2.4:

$$d = v * \Delta t \tag{2.3.1}$$

$$\delta = \frac{d}{C} * 2\pi \tag{2.3.2}$$

where

| | |
|---|---|
| $d$ | is the distance traveled since last timestep |
| $v$ | is the current velocity of the vehicle |
| $C$ | is the circumference of the turning circle |

Next, the center of the turning circle $(circX, circY)$ is calculated:

$$circX = r * cos(\theta_0 + \frac{\pi}{2}) \tag{2.3.3}$$

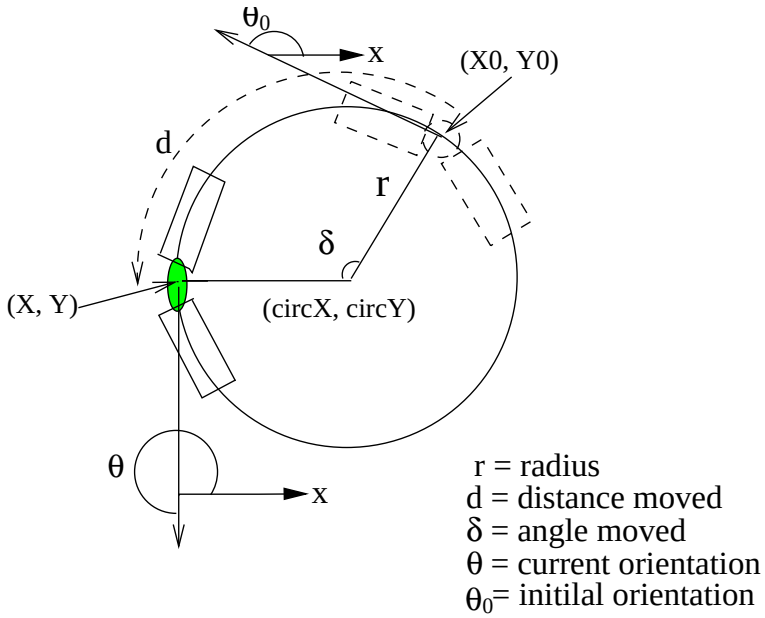


Figure 2.4: Determine where the vehicle will be in the next moment.

$$circY = r * sin(\theta_0 + \frac{\pi}{2}) \tag{2.3.4}$$

where

$\theta_0$ is the initial orientation of the vehicle

$r$ is the radius from Equation (2.2.1).

Now we can calculate the new pose $[(x, y), \theta]$ of the vehicle for a steering angle $\phi$:

1. if $\phi = 0$
   (a) $\theta = \theta_0$
   (b) $x = x_0 + cos(\theta) * (v * \Delta t)$
   (c) $y = y_0 + sin(\theta) * (v * \Delta t)$
2. else
   (a) $\theta = \theta_0 + \delta$
   (b) $x = circX - r * \cos(\theta + \frac{\pi}{2})$
   (c) $y = circY - r * \sin(\theta + \frac{\pi}{2})$

where

| | |
|---|---|
| $x_0, y_0$ | is the initial coordinates for the Vehicle center Point (VCP) |
| $x, y$ | is the new coordinates for the VCP |
| $circX, circY$ | is the center of the turning circle from Equations (2.3.3) & (2.3.4). |
| $\theta_0$ | is the initial orientation of the vehicle |
| $\theta$ | is the new orientation of the vehicle |
| $\phi$ | is the current steering angle |
| $\delta$ | is the angle moved, from Equation (2.3.2). |

# Chapter 3

# Path following

The aim with path following is, as the name indicates, to follow a predefined path, represented in the simulator by a series of $(x, y)$ coordinates (points) joined by line segments. A variety of more or less advanced techniques has been developed to solve this problem, mostly for small two-wheeled robots or car-like vehicles. In more complex techniques, all aspects of a vehicle performance (for example, brake, throttle and steering response, vehicle roll and tire slippage) is controlled. For low speed applications simpler control strategies are often used, where simplifying assumptions of the vehicles dynamics can be made. A forest environment is very complex, and many things can affect how the vehicle will behave in a certain situation (for example, ground conditions, the load of the vehicle and ambient trees). In the simulator however, neither the ground conditions nor the load of the vehicle is considered, but the only aspects of the vehicle being controlled is the velocity and steering angle.

An example of an more advanced technique is the *vector pursuit path tracking* method [Wit00], based on the theory of screws [Bal00]. This very old theory (introduced by Sir Robert Ball in the year 1900), describes the instantaneous motion of a moving rigid body (such as a car-like robot) relative to a given coordinate system. This means screw theory can be used to represent the motion of a vehicle, (i.e. a rigid body), from its current position and orientation to a desired position and orientation that is on a given path (i.e. the carrot point). The point with this method is that it considers not only the position of the carrot point, but also ensures that the vehicle arrives there with the correct orientation and steering angle (with regards to the orientation and curvature of the path at the carrot point). Due to the complexity of this theory and limited time, it has not been implemented.

We have not found any methods developed specifically for, or adapted to, articulated vehicles, so we adapted two different methods: *Follow The Carrot* [Bar01] and *Pure*

*Pursuit* [Ami90]. Both methods make use of a *carrot point* that enables the path follower to make decision based on a point further ahead on the path. Both methods are fairly straight forward and less advanced than for example the vector pursuit method described above.

One important thing that must be considered before developing a path follower for articulated vehicles is to choose the proper navigation point. This is the point of reference the position of the vehicle is determined by, and according to Mäkelä [Mäk01] this choice is critical to controlling an articulated vehicle. Choosing the articulation link as navigation point has, according to Mäkelä, been considered an improper choice by many authors. The reason is that if the navigation point is outside the path, a correction will temporary deviate even further when turning towards the path, which makes it difficult to get a stable controller. Another possible choice is the center of the rear axle, which produce an even more unstable controller according to Mäkelä. In his thesis, Mäkelä concludes that selecting the center of the front axle as navigation point is the best choice when driving forward.
In the current implementation, the navigation point is set to the articulation link. The main purpose of this is to make it easier to combine path following with obstacle avoidance. Both obstacle avoidance methods (VFH and VFH+) use the joint as point of reference when deciding where to go, and thereby it was easier to choose the same navigation point in the two path followers.

## 3.1 Follow The Carrot method

In Follow The Carrot [Bar01], a *carrot point* is selected on a *Look Ahead Distance* ahead on the path and then steer directly toward that point, see Figure 3.1. The analogy is that you hold a carrot in front of the vehicle and try to reach it through steering toward the carrot.

In order to calculate the location of the carrot point, the nearest point on the path, perpendicular to the path, must first be determined, see Figure 3.2. To calculate this, an algorithm from [sof02] is used:

distance( Point $P$, Segment $P_0 : P_1$ )
{
    $\mathbf{v} = P_1 - P_0$
    $\mathbf{w} = P - P_0$
    if $((c_1 = \mathbf{w} \bullet \mathbf{v}) <= 0)$ `//the point is to the left of P0`
        return $d(P, P_0)$

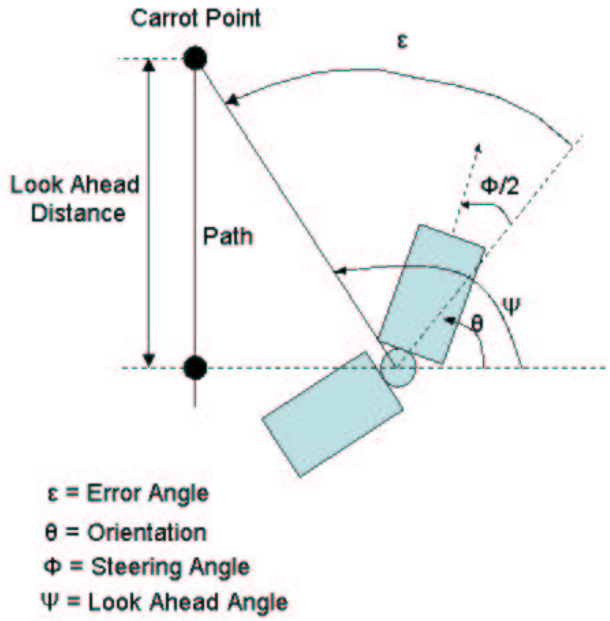


Figure 3.1: Follow The Carrot.

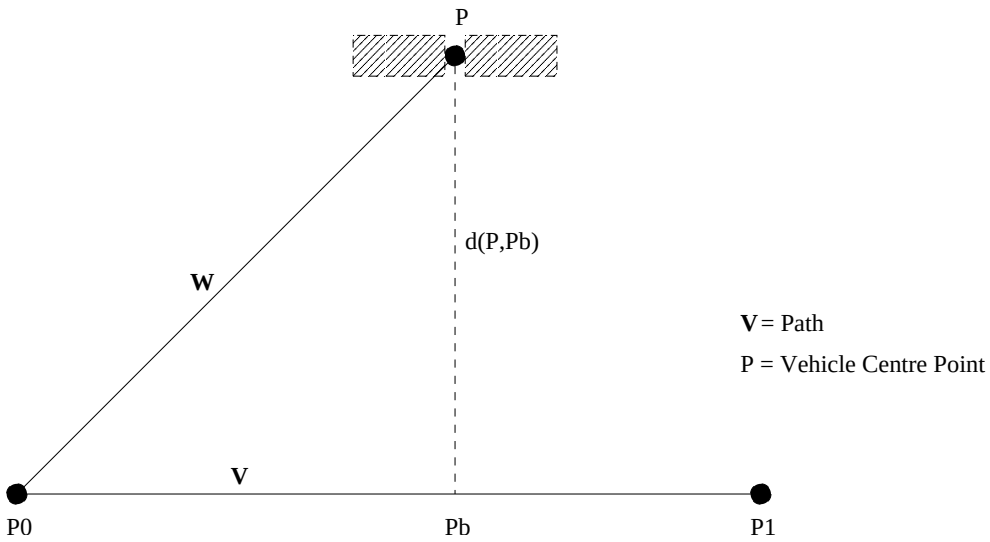


Figure 3.2: Distance between the vehicle and the nearest point on the path.

```
    if ((c2 = v • v) <= c1)  //the point is to the right of P1
        return d(P, P1)
    b = c1/c2
    Pb = P0 + bv
    return d(P, Pb)  //the point is on the segment
}
```

where

| | |
|---|---|
| $p$ | is the center point of the vehicle |
| $p_0, p_1$ | is the endpoints of a path segment |
| $p_b$ | is the point on the path closest to the vehicle |
| $d(p_1, p_2)$ | is the distance between two points |

To decide which segment of the path the vehicle is closest to, loop through all segments and select the segment closest to the vehicle. Because a segment is finite, the vehicle may be outside the perpendicular lines from the endpoints of the segment, as in Figure 3.3. In this case, the closest point is one of the endpoints, and we must determine which. One way to do this, but not a very efficient one, is to compute both distances and use the shortest. We must, however, still determine that the vehicle is actually outside the segment. To combine these two computations, we consider the angles between the segment $P_0\vec{P}_1$ and the vectors $P_0\vec{P}$ and $P_1\vec{P}$ from the segment endpoints to $P$ (the vehicle), see Figure 3.2. If either of these angles is 90°, then the corresponding endpoint is the closest point $P_b$. If the angle is not a right angle, then the base lies to one side or the other of the endpoint according to whether the angle is acute ($< 90°$) or obtuse($> 90°$). These conditions are easily tested by computing the dot product of the vectors involved and testing whether it is positive, negative, or zero. The result determines if the distance should be computed to one of the points P0 or P1, or as the perpendicular distance to the segment itself. In a situation as in Figure 3.3, the algorithm will not find any segment that the vehicle is closest to, and then we have to use the closest endpoint instead.

The carrot point is on a Look Ahead Distance from the point just calculated. If the carrot point lies outside the segment, take the remaining distance and apply it recursively on the next segment. If the carrot point is outside the last segment, let the carrot point be the end point on the last segment. The angle to the carrot point (defined in the global coordinate system) is called *Look Ahead Angle*, and is denoted by $\psi$.

In order to calculate the steering angle, an error angle $\epsilon$ is first determined, see Figure 3.1:

$$\epsilon = \psi - \theta \tag{3.1.1}$$

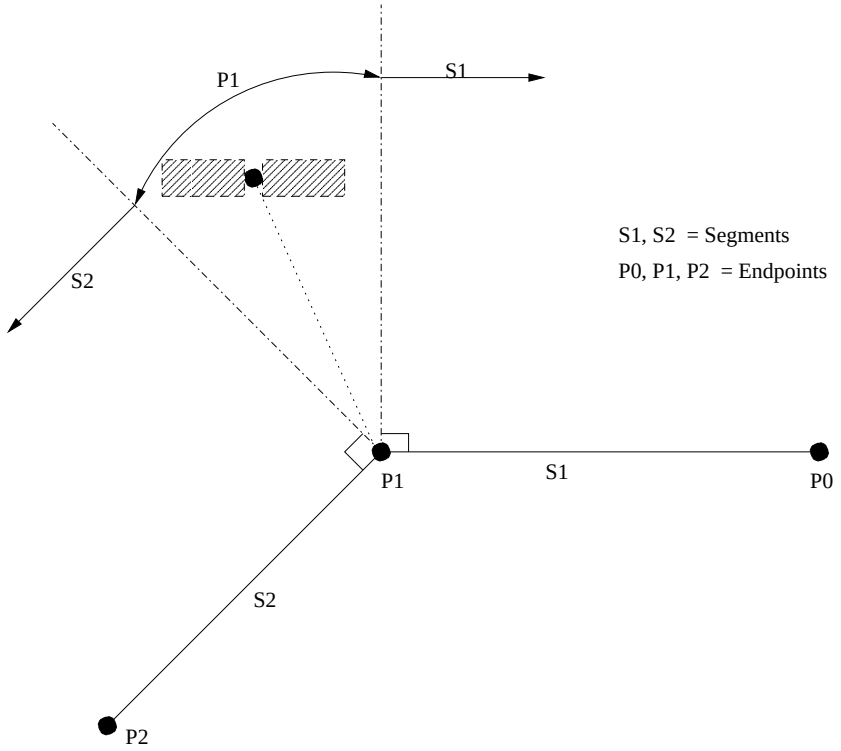


Figure 3.3: Choose an endpoint when the vehicle is between segments.

where
$\psi$ is the Look Ahead Angle
$\theta$ is the current orientation of the vehicle

In order to reduce the tendency of the vehicle taking short cuts through turns, and oscillating around the path, the error angle can be multiplied by a proportional gain $k_p$ and thereby reduce the steering angle somewhat (in the current implementation it is set to 1, and therefore not in use):

$$\epsilon' = \epsilon * k_p \tag{3.1.2}$$

where
$\epsilon$ is the error angle from (3.1.1)
$k_p$ is a proportional gain $(0 < k_p \leq 1)$

Before we calculate the steering angle, we must compensate for error angles larger

than $\pm 180°$:

$$\begin{aligned} &if \quad \epsilon' > \pi \\ &\quad \epsilon_0 = \epsilon' - 2\pi \\ &else\ if \quad \epsilon' < -\pi \\ &\quad \epsilon_0 = \epsilon' + 2\pi \\ &else \\ &\quad \epsilon_0 = \epsilon' \end{aligned} \tag{3.1.3}$$

where

$\epsilon'$ is the error angle from (3.1.2)

$\epsilon_0$ is the error angle compensated for large angles

To compute the steering angle $\phi$, the maximum steering angle of the vehicle (in this case $40°$) must be considered:

$$\phi = max(min(\epsilon_0, 40°), -40°) \tag{3.1.4}$$

where

$\epsilon_0$ is the error angle from (3.1.3)

## 3.2 Pure Pursuit

The Pure Pursuit method is an improvement of the Follow The Carrot method, but the two methods are very similar. Wallace [WST+85] usually gets the credit for having developed the original Pure Pursuit method, and after that Amidi [Ami90] has developed and tested the algorithm further. The main difference between Pure Pursuit and Follow The carrot is that, instead of steering directly towards the carrot point, in pure pursuit a circular arc is fitted between the vehicle and the carrot point, see Figure 3.4. This circle is uniquely defined by adding the condition that the heading of the vehicle should be along the tangent line of the turning circle. The Pure Pursuit method enables a smoother steering control, and improves the vehicles ability to handle curved paths with lesser short cuts than in Follow The Carrot. The reason is that the vehicle normally moves along circular arcs when turning, rather than going along straight lines, hence trying to follow a circular arc is more natural than with a straight line. An analogy is a driver of a car who looks on a point further ahead on the road and then tends to steer smoothly toward that point, and thereby driving in circular arcs on a curved road. The Pure Pursuit Algorithm is as follows:

1. *Determine the current location of the vehicle.*

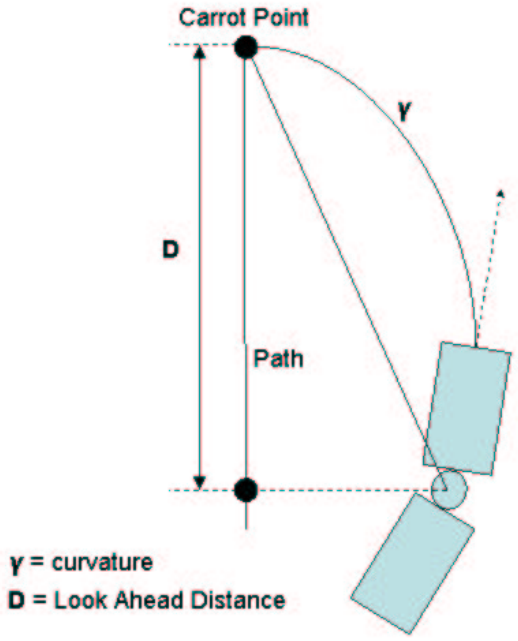


Figure 3.4: Pure Pursuit.

2. *Find the point on the path closest to the vehicle.* This is needed to calculate where the carrot point will be. The algorithm for this is exactly as in Follow The Carrot.

3. *Find the carrot point.* As in Follow The Carrot this point is a constant Look Ahead Distance from the path point. The same function as in Follow The Carrot is used calculate this.

4. *Transform the carrot point and the vehicle location to the vehicles coordinates.* This is the tricky part, the equations to calculate the curvature is only valid in the vehicles coordinates. The Y-axis in vehicle coordinates is the same as the orientation $\theta$, and all angles and coordinates must be with respect to the vehicles current position and orientation, se Figure 3.5

5. *Calculate the curvature, $\gamma$, of the circular arc.* The curvature is defined as the inverse radius $r$ ($\gamma = 1/r$).

6. *Determine the steering angle, $\phi$.* The steering angle is derived from the radius of the circular arc ($1/\gamma$), see below.

The derivation of the curvature $\gamma$ and radius $r$ of the arc connecting the carrot point and the current position of the vehicle (VCP), follows the principles in [Cou92]: Consider Figure 3.5. The point $(x_c, y_c)$ is on a Look Ahead Distance $D$ from the current position of the vehicle. The following two equations hold. The first is from the geometry of the right triangle in Figure 3.5, and the second from the summing of line segments on the x' axis.

$$x_c^2 + y_c^2 = D^2 \tag{3.2.1}$$

$$\Delta x + d = r \tag{3.2.2}$$

- Equation 3.2.1 describes the circle of radius D about the VCP. This is the locus of possible carrot points for the vehicle.
- Equation 3.2.2 describes the relationship between the radius of the arc that joins the VCP and the carrot point $(x_c, y_c)$, and the x offset of the carrot point from the vehicle, $\Delta x$. This equation states that the radius of the arc and the x offset are independent and differs by $d$.

The curvature and radius are derived by:

$$\begin{aligned} d &= r - \Delta x \\ (r - \Delta x)^2 + y_c^2 &= r^2 \\ r^2 - 2r\Delta x + x_c^2 + y_c^2 &= r^2 \\ 2r\Delta x &= D^2 \\ r &= \frac{D^2}{2\Delta x} \\ \gamma &= \frac{2\Delta x}{D^2} \end{aligned} \tag{3.2.3}$$

To calculate the curvature, $\gamma$, the following geometric relation is used, see Figure 3.5:

$$\alpha = \frac{\pi}{2} - \theta + \psi \tag{3.2.4}$$

$$\Delta x = D cos\alpha \tag{3.2.5}$$

$$\gamma = -\frac{2\Delta x}{D^2} \tag{3.2.6}$$

$$r = \frac{1}{\gamma} = -\frac{D^2}{2\Delta x} \tag{3.2.7}$$

where

| | |
|---|---|
| $\Delta x$ | x displacement of the Carrot Point in the vehicle coordinate system |
| $D$ | distance between the Vehicle center Point and the Carrot Point |
| $\theta$ | orientation in the global coordinate system |
| $\psi$ | Look Ahead Angle in the global coordinate system |
| $r$ | radius |

The steering angle $\phi$ is then derived by (see Figure 2.3):

$$\beta = cos^{-1}(\frac{length}{r}) \qquad \text{From (2.2.2)} \qquad (3.2.8)$$

$$\frac{\phi}{2} = \frac{\pi}{2} - \beta \qquad (3.2.9)$$

where

| | |
|---|---|
| $length$ | is the distance from the Vehicle center Point to the wheel axis (defined as in the Follow The Carrot method above). |
| $r$ | is the radius from (3.2.7) |
| $\beta$ | is the angle from Figure 2.3 |

A problem with combining Pure Pursuit and obstacle avoidance is that the VFH-methods give a steering angle directly at the goal point, just like in Follow The Carrot. This means that there would be no difference between the two path following algorithms when combining them with obstacle avoidance. To get around this problem we let the Pure Pursuit algorithm determine the steering angle if there is no obstacles ahead of the vehicle, otherwise the angle from the obstacles avoidance algorithm is used. Se further details in the Obstacle Avoidance section.

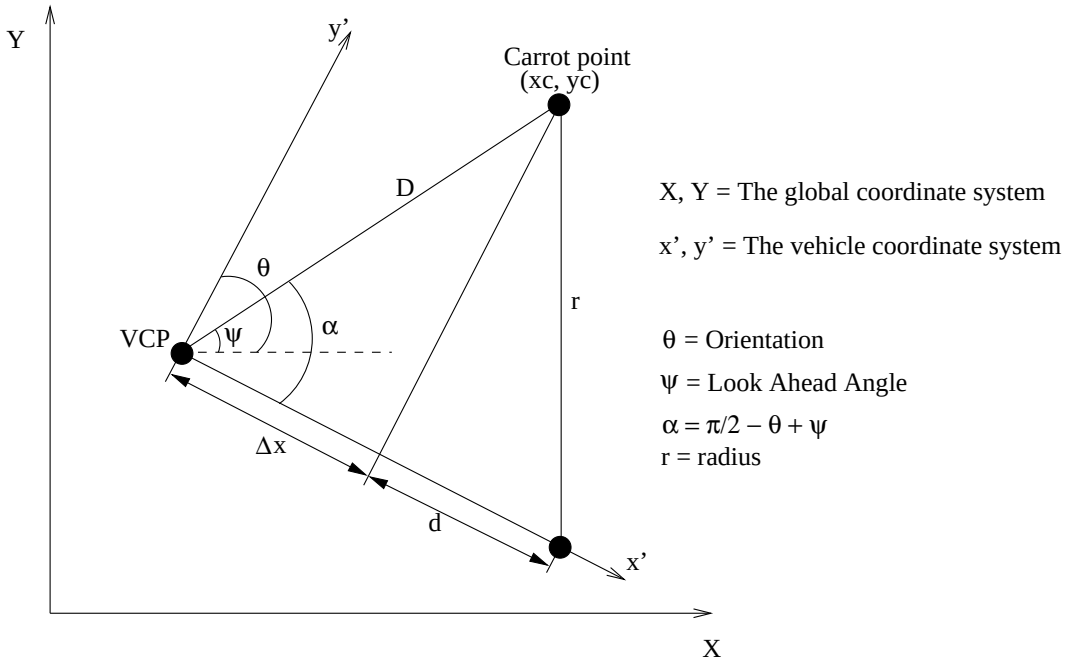


Figure 3.5: Transformation from world coordinates to vehicle coordinates.

# Chapter 4

# Obstacle avoidance

In order to successfully develop an autonomous vehicle that will be able to traverse real terrain, obstacle avoidance is a key issue. The most primitive form of obstacle avoidance stops the vehicle short of an obstacle when detecting it, in order to avoid collision. In applications for avoiding obstacles in indoor environments (typically long corridors) it is common to use the wall-following method. Here the vehicle follows the wall of the corridor at a certain distance. If an obstacle is detected, the robot regards it as just another wall and follows it around at a safe distance until the vehicle resumes its previous course. This is a less demanding technique than other more complex methods, mostly because the exact position of the vehicle does not need to be determined. The drawback with the wall-following method is that it is less versatile and is only suitable for very specific applications.

A vehicle in a forest environment demands a completely different level of complexity though. Here we must be able to detect the size and range of all obstacles, be able to steer around (or between) them and then continue on the right path again. This means that the obstacle avoider must be combined with a path follower in order to know which side of an obstacle is best to go (depending on where the path is), and where to go after the obstacle been safely avoided. Both obstacle avoidance methods described in this thesis (VFH and VFH+) is based on the Virtual Force Field (VFF) method [KB91]. The idea with VFF is that every obstacle within a certain distance of the vehicle applies a repulsive force vector on the vehicle, with a magnitude inversely proportional to the square of the distance between the obstacle and the vehicle, "pushing" the vehicle away from the obstacle. To draw the vehicle towards the goal point, an attractive force is applied to it. To tell where the vehicle should be heading, a resultant vector is produced by taking the vectorial sum of all forces; repulsive from the obstacles and attractive from the goal point. The main

advantage over similar methods that uses force vectors is the use of certainty grid in combination with force vectors. In short, a certainty grid is built much like a radar screen, where each obstacle found by a sensor will count up the certainty value at the corresponding coordinates in the certainty grid. This means that a misreading by a sensor will result in a low certainty value and will be almost completely ignored, while a real obstacle will be detected multiple times by the sensors, and thereby resulting in a higher certainty value, which results in a larger repulsive force from that coordinates.

## 4.1 Vector Field Histogram (VFH)

The VFH method [BK91] is based on the Virtual Force Field (VFF) method [KB91]. VFH tries to overcome some of the shortcomings of VFF with a *two-stage data reduction* technique rather than a single-step technique used in the VFF method. This way, three levels of data representation exist:

1. The highest level holds detailed description of the vehicle's environment. A two-dimensional *histogram grid* that corresponds to the real environment is used to store data about obstacles. Normally this grid would be updated in real-time with data from the on-board sensors, but in the simulator the sensor data is assumed to have been collected already and the vehicle thereby have a complete picture of the closest surroundings.

2. On the next level, a one-dimensional *polar histogram* is constructed around the vehicle. This is done by mapping the histogram grid onto the polar histogram, resulting in a number of sectors with values of the obstacle density in the direction corresponding to each sector.

3. The lowest level of data is the actual output of the VFH-algorithm; the angle of the preferred steering direction.

### 4.1.1 Creation of the Polar Histogram

A window moves with the vehicle, covering a circular region of cells in the histogram grid, see Figure 4.1. We now split the active window in *n sectors* (72 sectors of 5 degrees in this case). The contents of each cell is used to create an obstacle vector, with direction $\beta$ from the cell to the *Vehicle center Point* (VCP), and with magnitude

$m$. Direction and magnitude is given by:

$$\beta = tan^{-1}(\frac{y_j - y_0}{x_i - x_0}) \tag{4.1.1}$$

$$m_{i,j} = (C^*_{i,j})^2(a - bd_{i,j}) \tag{4.1.2}$$

where

| | |
|---|---|
| a, b | positive constants |
| $C^*_{i,j}$ | Certainty value of active cell (i,j) |
| $d_{i,j}$ | Distance between active cell (i,j) and the VCP |
| $m_{i,j}$ | Magnitude of the obstacle vector at cell (i,j) |
| $x_0, y_0$ | Current coordinates of the VCP |
| $x_j, y_j$ | Coordinates of active cell (i,j) |
| $\beta$ | Direction from active cell (i,j) to the VCP |

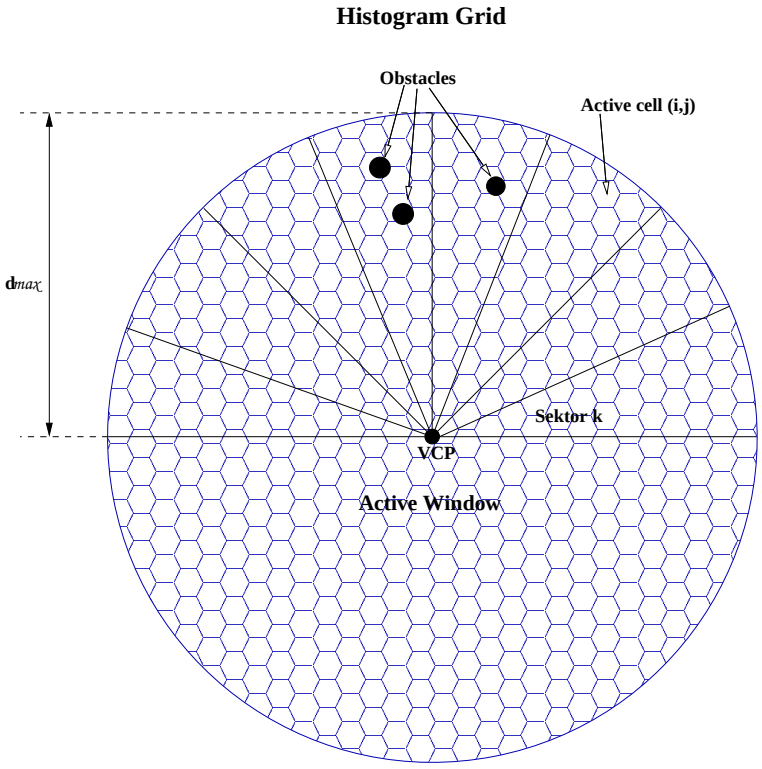


Figure 4.1: Creation of the polar histogram.

The certainty value is used to ensure that it is a real obstacle and not just noise from the sensors. In a real situation this would be done by updating this value each time the sensors sense an obstacle. In the simulator however, a constant certainty value is used because noise in the sensors is not simulated.

The magnitude of an obstacle vector is proportional to $-d$, which ensures that the magnitude is larger when closer to an obstacle. The constants a and b is chosen

such that $a - bd_{max} = 0$, where $d_{max}$ is the distance to the farthest active cell (in the current implementation, to the carrot point). This means that $m_{i,j} = 0$ for the farthest active cell and increases closer to the vehicle.
To decide which sector an obstacle belongs to, we need an angular resolution $\alpha$ such as $n = 360/\alpha$ is an integer (in this case $\alpha = 5$ and $n = 72$). To calculate the sector $k$ where an obstacle vector with angle $\beta$ belongs we use:

$$k = INT(\frac{\beta_{i,j}}{\alpha}) \tag{4.1.3}$$

For each sector $k$ $(0 \leq k \leq n)$, the *Polar Obstacle Density (POD)* $h_k$ is calculated as:

$$h_k = max(m_{i,j}) \qquad\qquad i, j \in k \tag{4.1.4}$$

In the original VFH method, the sum of all magnitudes in every sector is used. However, we found the result to be better when only the nearest obstacle in each sector was considered. This way the magnitude of a sector does not depend of the number of obstacles in a sector, but just the distance to the nearest obstacle in that sector.
To ensure a smoother steering control, a *smoothing function* is applied to $h_k$:

$$h_k^{'} = \frac{h_{k-l} + 2h_{k-l+1} + ... + lh_k + ... + 2h_{k+l-1} + h_{k+l}}{2l + 1} \tag{4.1.5}$$

$h_k^{'}$ is called a *Smoothed Polar Obstacle Density*. In the current implementation $l = 4$ is used.

Figure 4.2 shows a Polar Histogram for two obstacles. The x-axis shows which angle the sectors lies in, and the y-axis the magnitude of the sectors. Note that the sector to the right has a lower magnitude, depending on the obstacle in that sector being further away than in the other sector. Figure 4.3 shows the corresponding *Smoothed Polar Histogram*. Note that each of the two original sectors is now surrounded by several sectors, as a result of applying the smoothing function.

### 4.1.2 Selecting the steering angle

In the Smoothed Polar Obstacle Density, free sectors are those with $h_k^{'}$ below a certain threshold $\tau$. If more than $s_{max}$ consecutive sectors fall below the threshold we have a *wide candidate valley*, otherwise it is a *narrow candidate valley* (in the current implementation a narrow valley must be $> \frac{s_{max}}{5}$). Each candidate valley represents a free space where the vehicle may pass. In the original VFH algorithm the goal valley is selected simply by taking the valley closest to the look ahead

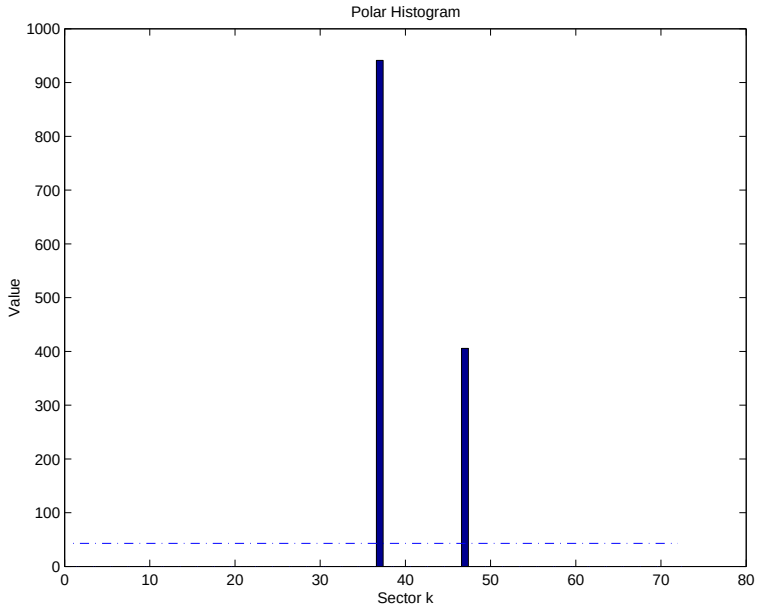


Figure 4.2: A Polar Histogram.

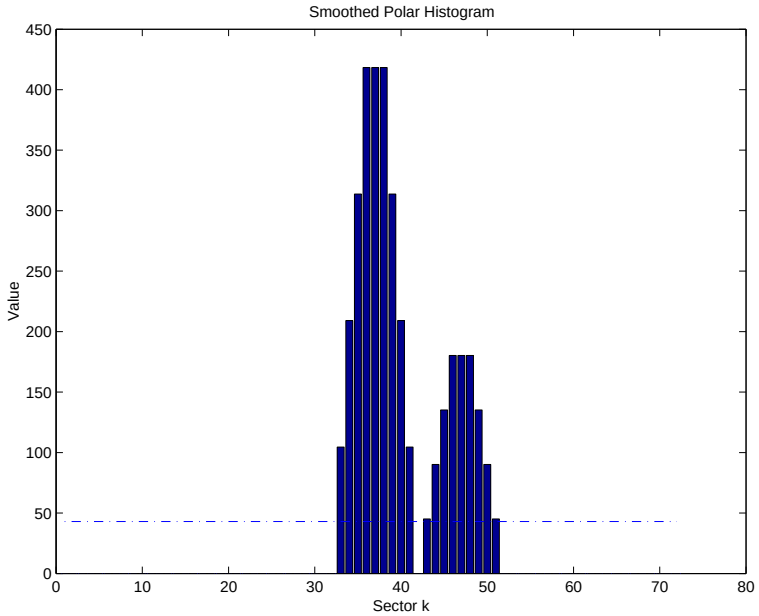


Figure 4.3: A Smoothed Polar Histogram.

sector $k_t$ (the sector which contains the carrot point). But because of very poor performance in our implementation (the vehicle bumped into about every obstacle in sight), the cost function from the VFH+ method is used instead. This function takes into account not only the distance between the candidate valley and the Look Ahead Angle, but also the current orientation of the vehicle and the previous selected goal angle and thereby has a better performance. To choose the best direction of travel, all candidate valleys are first detected. Since a valley can be wide, we could choose different paths through it, and therefore a set of candidate directions, $c$, must be determined:

- If $\frac{s_{max}}{5} < |k_l - k_r| < s_{max}$, it is considered to be a narrow valley and a centered direction of travel is chosen:
$$c_n = \frac{k_r + k_l}{2} \tag{4.1.6}$$
- For a wide opening we can choose to move along the right or left side of the valley $(c_r, c_l)$. If the target (look ahead) sector $k_t$ is within the valley, it also is considered a candidate direction $(c_t)$:
$$c_r = k_r + \frac{s_{max}}{2} \qquad \text{right side of valley} \tag{4.1.7}$$
$$c_l = k_l - \frac{s_{max}}{2} \qquad \text{left side of valley} \tag{4.1.8}$$
$$c_t = k_t \qquad \text{if } k_t \in [c_r, c_l] \tag{4.1.9}$$

In this way we have between one and three candidate direction for each candidate valley in the smoothed polar histogram. To choose the appropriate direction of travel at timestep $i$, the cost function $g$ from the VFH+ method is applied to a candidate direction $c$:

$$g(c) = \mu_1 * \Delta(c, k_t) + \mu_2 * \Delta(c, \frac{\theta_i}{\alpha}) + \mu_3 * \Delta(c, k_{i-1}) \tag{4.1.10}$$

where

| | |
|---|---|
| $\mu_1, \mu_2, \mu_3$: | cost factors |
| $k_t$: | target (look ahead) sector |
| $\theta_i$: | the current orientation |
| $\alpha$: | angular resolution |
| $k_{i-1}$: | previous selected sector |

$\Delta(c_1, c_2)$ is a function returning the number of sectors between two sectors $c_1$ and $c_2$ such that the result is $\leq n/2$, where n is the total number of sectors:

$$\Delta(c_1, c_2) = min\{|c_1 - c_2|, |c_1 - c_2 - n|, |c_1 - c_2 + n|\} \tag{4.1.11}$$

For more details about the cost function, se Section 4.2.4.
In order to choose the best direction to travel, the cost function $g$ is applied to all

candidate directions $c$. The candidate with the lowest $g$ is selected, and is denoted $c_{sel}$. The corresponding angle $\psi$ is given by:

$$\psi = c_{sel} * \alpha \tag{4.1.12}$$

The computed angle $\psi$ is used to decide whether or not to use the angle calculated in Pure Pursuit. If $\psi = LookAheadAngle$, VFH report back that there are no obstacles in front of the vehicle, the angle from Pure Pursuit is used and the Pure Pursuit algorithm is allowed to work as usual. Otherwise the angle is sent to the path follower, where it is treated as the new Look Ahead Angle, and the Pure Pursuit algorithm works as the Follow The Carrot algorithm.

### 4.1.3 Summary

A brief summary of the VFH method:

- First we build a two-dimensional *histogram grid* for obstacle representation. Each cell $i, j$ in the histogram grid holds a *certainty value* that represents the confidence that there actually is an obstacle in that cell, and not just a misreading. In the simulator, a constant certainty value is used.
- Next, we map the histogram grid onto a *polar histogram*, containing $k$ sectors holding a value $h_k$ that represents the *polar obstacle density* in the direction corresponding to sector $k$, see Equation (4.1.4). In order to minimize errors in the steering control, a smoothing function is applied to $h_k$, called *smoothed polar obstacle density* $h_k^{'}$, see Equation (4.1.5).
- The last step is to compute the required steering direction. This is done by checking the number of sectors whose value $h_k^{'}$ is below a certain threshold $\tau$, representing a narrow or wide *candidate valley*. If the valley is wide, we can choose to go in more than one direction within the valley, see Equations (4.1.7), (4.1.8) and (4.1.9). If it is a narrow valley, we choose a centered direction of travel, se Equation (4.1.6). To select the best direction of travel from these candidate directions, we use a cost function, see Equation (4.1.10).

## 4.2 VFH+

VFH+ [UB98] is a enhanced version of VFH. This method was developed for a special type of robot, called the *GuideCane*. The GuideCane is a modern version of an

ordinary guide cane for visually handicapped people, automatically steering around obstacles when detected. Because of the similarities between a conventional mobile robot and the GuideCane, the VFH+ algorithm should be applicable to a vehicle such as the forwarder as well.
The main difference between VFH and VFH+ is that the latter has a four-stage data reduction process to select the new direction of travel. In the first three stages, the two-dimensional *histogram grid* (the same as in VFH) is reduced to one-dimensional *polar histograms*. In the fourth stage a cost function is applied on the polar histograms to select the best steering direction.

### 4.2.1 The Primary Polar Histogram

The first stage is very similar to the VFH approach. The obstacle vectors are calculated as above, see Figure 4.1, apart from the magnitude which uses the squared distance ($d^2$) to an obstacle:

$$\beta = tan^{-1}(\frac{y_j - y_0}{x_i - x_0}) \tag{4.2.1}$$

$$m_{i,j} = (C^*_{i,j})^2(a - bd^2_{i,j}) \tag{4.2.2}$$

where

| | |
|---|---|
| a, b | positive constants |
| $C^*_{i,j}$ | Certainty value of active cell (i,j) |
| $d_{i,j}$ | Distance between active cell (i,j) and the VCP |
| $m_{i,j}$ | Magnitude of the obstacle vector at cell (i,j) |
| $x_0, y_0$ | Current coordinates of the VCP |
| $x_j, y_j$ | Coordinates of active cell (i,j) |
| $\beta$ | Direction from active cell (i,j) to the VCP |

The parameters $a$ and $b$ is chosen such that

$$a - bd_{max} = 1 \tag{4.2.3}$$

where

| | |
|---|---|
| $d_{max}$ | is the distance to the farthest active cell (in the current implementation, to the carrot point). |

A drawback with the original VFH method is that it does not take into account the width of the vehicle. To compensate for this, the VFH+ method enlarge every obstacle cell with the robot radius $r_r$ (the distance from the VCP to the furthest point on the vehicle). For further safety, the obstacle cells are enlarged by

$$r_{r+s} = r_r + d_s \tag{4.2.4}$$

where

$d_s$ is the minimum distance between an obstacle and the vehicle.

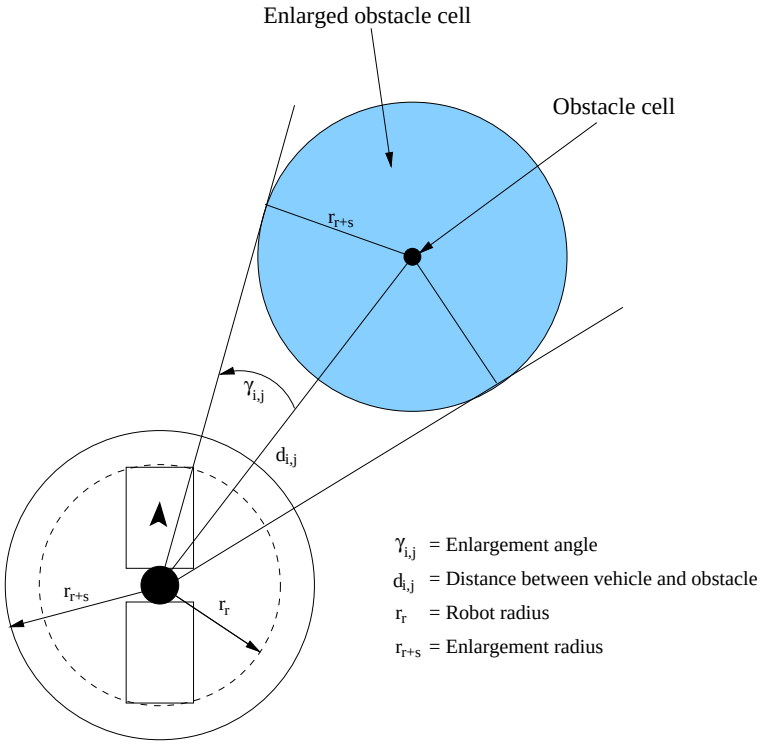


Figure 4.4: Enlargement angle.

For each obstacle cell an enlargement angle $\gamma_{i,j}$ is defined by

$$\gamma_{i,j} = arcsin(\frac{r_{r+s}}{d_{i,j}}) \tag{4.2.5}$$

This is the angle between the obstacle vector and the vector between the VCP and the enlarged obstacle cell, see Figure 4.4. For each sector $k$, the polar obstacle density $H_k^p$ is then calculated by

$$H_k^p = (max(m_{i,j}))h'_{i,j} \qquad i,j \in k \tag{4.2.6}$$

with

$h'_{i,j} = 1$ if $k\alpha \in [\beta_{i,j} - \gamma_{i,j},\ \beta_{i,j} + \gamma_{i,j}]$

$h'_{i,j} = 0$ otherwise

This creates a polar histogram that takes into account the width of the vehicle, see Figure 4.5. The $h'$ function replaces the smoothing function in VFH and thereby eliminates the difficulties fine tuning the low-pass filter involved in calculating the smoothing function in the original VFH algorithm.

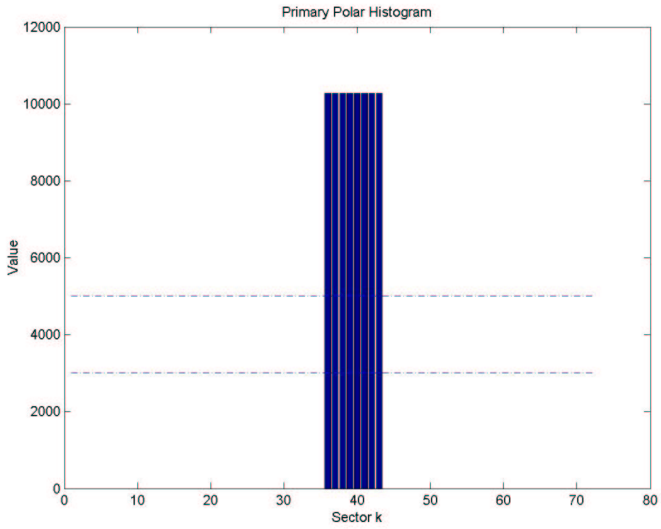


Figure 4.5: A Primary Polar Histogram.

### 4.2.2 The Binary Polar Histogram

The second stage is to make a binary polar histogram $H^b$ from the primary histogram, see Figure 4.6. Instead of having density values, the sectors of $H^b$ are either *free* (0) or *blocked* (1). Unlike the fixed threshold from VFH, VFH+ uses two thresholds: $\tau_{low}$ and $\tau_{high}$. Bar $k$ in the binary polar histogram has height $H_k^b$ defined by

$$\begin{aligned} H_k^b &= 1 && \text{if } H_k^p > \tau_{high} \\ H_k^b &= 0 && \text{if } H_k^p < \tau_{low} \\ H_k^b &= H_{k-1}^b && \text{otherwise} \end{aligned} \tag{4.2.7}$$

### 4.2.3 The Masked Polar Histogram

The original VFH method does not take into account the limits of the vehicle's turning radius, but simply assumes the vehicle can travel in any direction at any given time, as in the left part of Figure 4.7. The VFH+ method creates a masked polar histogram that considers the maximum turning radius of the vehicle, as in the right part of Figure 4.7, and determines which sectors are blocked by obstacles. As we see in Figure 4.8, all angles to the right of the right obstacle will be blocked due to the maximum turning radius.

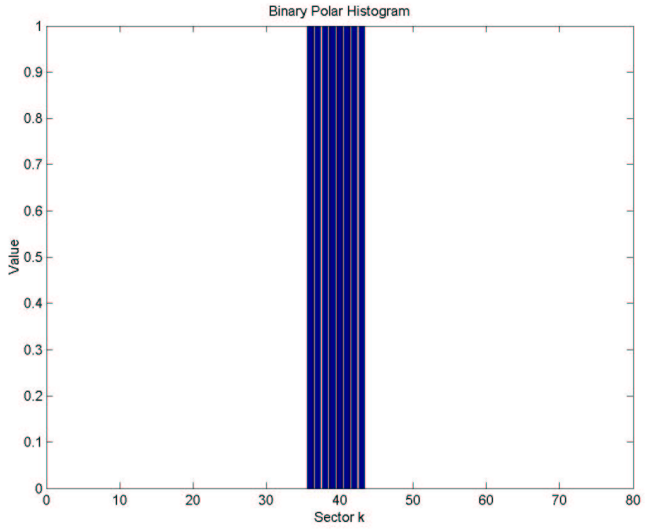


Figure 4.6: A Binary Polar Histogram.

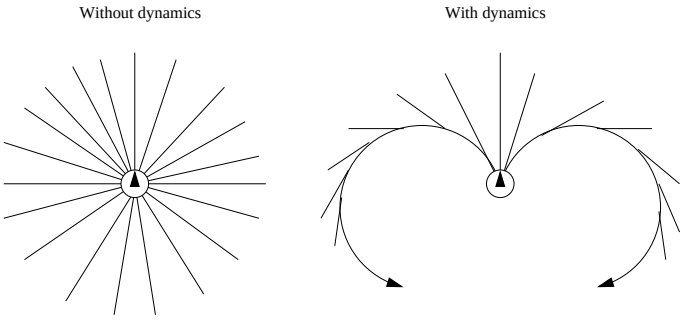


Figure 4.7: Trajectories with and without dynamics of the vehicle.

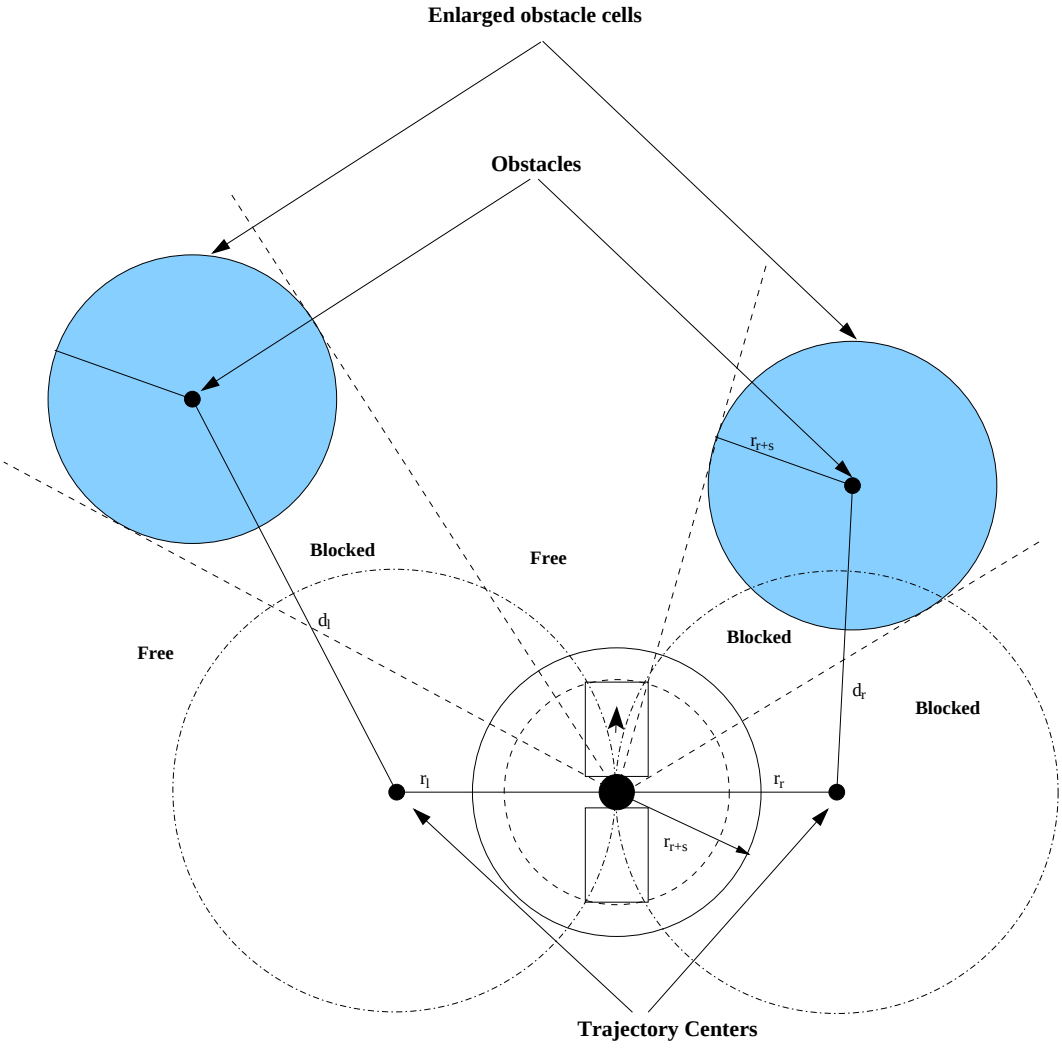


Figure 4.8: Directions blocked by obstacles.

To start with we define two circles on each side of the vehicle, see Figure 4.8, representing the left and right trajectories at maximum steering angle. The positions of the centers of these circles are defined by

$$\Delta x_r = r_r * sin\theta \qquad \Delta y_r = r_r * cos\theta \tag{4.2.8}$$

$$\Delta x_l = -r_l * sin\theta \qquad \Delta y_l = -r_l * cos\theta \tag{4.2.9}$$

where

| | |
|---|---|
| $r_r$ and $r_l$: | radius to the right and left trajectory centers respectively. |
| $\theta$: | the current orientation of the vehicle |
| $\Delta x$ and $\Delta y$: | the distance from the VCP to the trajectory centers. |

The distances between an obstacle in cell $C_{i,j}$ and the two trajectory centers are given by:

$$d_r^2 = (\Delta x_r - \Delta x(i))^2 + (\Delta y_r - \Delta y(j))^2 \tag{4.2.10}$$

$$d_l^2 = (\Delta x_l - \Delta x(i))^2 + (\Delta y_l - \Delta y(j))^2 \tag{4.2.11}$$

where

| | |
|---|---|
| $\Delta x(i)$ and $\Delta y(j)$ | is the distance between obstacle cell $i, j$ and the VCP. |

The direction to the right of an obstacle is blocked if

$$d_r^2 < (r_r + r_{r+s}) \qquad \text{[condition 1]} \tag{4.2.12}$$

And the direction to the left of an obstacle is blocked if

$$d_l^2 < (r_l + r_{r+s}) \qquad \text{[condition 2]} \tag{4.2.13}$$

where

| | |
|---|---|
| $r_{r+s}$ | is the radius of an enlarged obstacle cell from (4.2.4) |
| $r_r$ and $r_l$: | radius to the right and left trajectory centers respectively. |

By computing these two conditions for every obstacle cell, we get two limit angles $\varphi_l$ and $\varphi_r$ which defines the maximum steering angle to the left and right respectively of an obstacle. The direction directly backwards from the current direction is defined as $\varphi_b = \theta + \pi$. The algorithm for calculating the limit angles is as follows:

1. Determine $\varphi_b$. Set $\varphi_l$ and $\varphi_r = \varphi_b$
2. For every obstacle cell $C_{i,j}$:
   (a) If $\beta_{i,j}$ in (4.2.1) is to the right of $\theta$ and to the left of $\varphi_r$, check condition 1 in (4.2.12). If satisfied, set $\varphi_r = \beta_{i,j}$.

(b) If $\beta_{i,j}$ is to the left of $\theta$ and to the right of $\varphi_l$, check condition 2 in (4.2.13). If satisfied, set $\varphi_l = \beta_{i,j}$.

where
$\theta$: the current orientation of the vehicle
$\beta_{i,j}$: the angle between the vehicle and obstacle cell $i, j$

To determine if an angle $a_1$ is to the left or right to an angle $a_2$, the following algorithm is used:

```
compAngles(a1, a2)
{
    a3 = (a2 + π) mod 2π        // directly backwards from a2 (180°)
    if (a1 = a2)
        return 'forward'
    else if (a2 ≤ π)
    {
        if (a1 > a2  &  a1 < a3)
            return 'left'
        else
            return 'right'
    }
    else
    {
        if (a1 < a2  &  a1 > a3)
            return 'right'
        else
            return 'left'
    }
}
```

Now we are ready to build the Masked Polar Histogram $H^m$ with the help of the Binary Polar Histogram, $\varphi_l$ and $\varphi_r$, see Figure 4.9:

$$\begin{aligned} H_k^m &= 0 \qquad \text{if } H_k^b = 0 \text{ and } (k * \alpha) \in \{[\varphi_r, \theta], [\theta, \varphi_l]\} \\ H_k^m &= 1 \qquad \text{otherwise} \end{aligned} \tag{4.2.14}$$

If all sectors are blocked, there is no possible direction of travel and the vehicle has to halt. Otherwise we just have to choose from all candidate valleys in the Masked

Polar Histogram to get the new direction of travel, which will be the fourth and last stage in the VFH method.

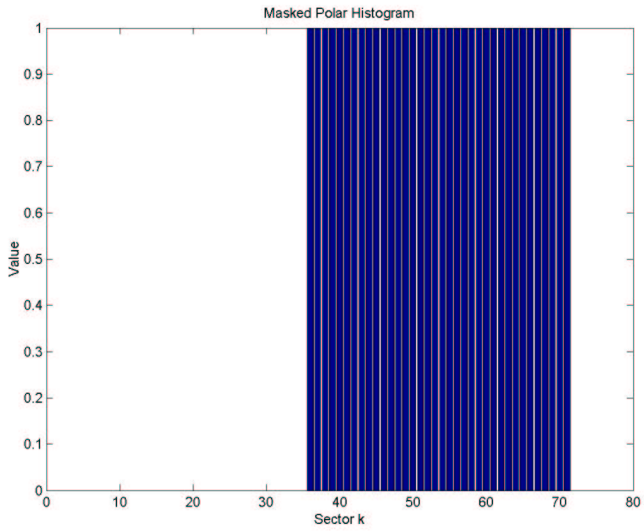


Figure 4.9: A Masked Polar Histogram.

### 4.2.4 Selection of the steering angle

This last stage is done by a cost function that determines the cost of going in a certain direction. The original VFH method is very goal oriented, because it chooses the direction closest to the look ahead direction. The VFH+ method finds all openings in the Masked Polar Histogram and then determines a set of candidate directions $c$. First we determine if a candidate valley is wide or narrow, depending on how many subsequent sectors are free for travel. If the difference between the left and right borders $(k_l, k_r)$ is less than $s_{max}$ it is considered to be a narrow valley and a centered direction of travel is chosen:

$$c_n = \frac{k_r + k_l}{2} \tag{4.2.15}$$

For a wide opening we can choose to go at the right or left side of the valley $(c_r, c_l)$. If the target sector $k_t$ (the sector with the carrot point) is within the valley, it also is considered a candidate direction $(c_t)$:

$$\begin{aligned} c_r &= k_r + \frac{s_{max}}{2} && \text{right side of valley} \\ c_l &= k_l - \frac{s_{max}}{2} && \text{left side of valley} \\ c_t &= k_t && \text{if } k_t \in [c_r, c_l] \end{aligned} \tag{4.2.16}$$

So now we have between one and three candidate direction for each valley in the masked polar histogram. To choose the appropriate direction of travel at timestep $i$, a cost function $g$ is applied to each candidate direction $c$:

$$g(c) = \mu_1 * \Delta(c, k_t) + \mu_2 * \Delta(c, \frac{\theta_i}{\alpha}) + \mu_3 * \Delta(c, k_{i-1}) \tag{4.2.17}$$

where

| | |
|---|---|
| $\mu_1, \mu_2, \mu_3$: | cost factors |
| $k_t$: | target (look ahead) sector |
| $\theta_i$: | the current orientation |
| $\alpha$: | angular resolution |
| $k_{i-1}$: | previous selected sector |

$\Delta(c_1, c_2)$ is a function returning the number of sectors between two sectors $c_1$ and $c_2$, such that the result is $\leq n/2$, where n is the total number of sectors:

$$\Delta(c_1, c_2) = min\{|c_1 - c_2|, |c_1 - c_2 - n|, |c_1 - c_2 + n|\} \tag{4.2.18}$$

The cost function is divided in three parts:

1. The first part represents the cost for going in the candidate direction, given the target direction. The larger the difference between the candidate direction and the target direction, the larger the cost. This term represent the goal-oriented behavior of the vehicle.

2. Next is the cost for going at the candidate direction, given the current orientation of the vehicle. The larger the difference, the more we are required to change direction, hence larger cost.

3. The last term is the cost for changing direction from the previous selected direction of travel. The larger the difference between the candidate and the previous direction, the larger the cost.

The cost factors, $\mu_1, \mu_2, \mu_3$, controls the importance of the respective parts in the cost function. To get a more goal-oriented behavior, $\mu_1$ must be larger than the other terms together ($\mu_1 > \mu_2 + \mu_3$). In order to select the best direction of travel, the cost function $g$ is applied to all candidate directions $c$. The candidate with the lowest $g$ is selected, and is denoted $c_{sel}$. The corresponding angle $\psi$ is given by:

$$\psi = c_{sel} * \alpha \tag{4.2.19}$$

The computed angle $\psi$ is used to decide whether or not to use the angle calculated in Pure Pursuit. If $\psi = LookAheadAngle$, VFH+ report back that there are no obstacles in front of the vehicle, the angle from Pure Pursuit is used and the Pure Pursuit algorithm is allowed to work as usual. Otherwise the angle is sent to the path follower, where it is treated as the new Look Ahead Angle, and the Pure Pursuit algorithm works as the Follow The Carrot algorithm.

### 4.2.5 Summary

A brief summary of the VFH+ method:

- First we build a two-dimensional *histogram grid* for obstacle representation. Each cell $i, j$ in the histogram grid holds a *certainty value* that represents the confidence that there actually is an obstacle in that cell, and not just a misreading. In the simulator, a constant certainty value is used.
- Next, we map the histogram grid onto a *primary polar histogram*, containing $k$ sectors holding a value $h_k^p$ that represents the *polar obstacle density* in the direction corresponding to sector $k$, see Equation (4.2.6). In this process we also take into account the width of the vehicle. The $h'$ function in Equation (4.2.6) serves as a low-pass filter, and thereby eliminating the need for the smoothing function in the VFH method.
- The third step is to build a *binary polar histogram* $h_k^b$, where we use two thresholds (instead of the one in the VFH method) to reduce indecisive steering behavior when choosing between two or more candidate valleys. The sectors of $h^b$ are either *free* (0) or *blocked* (1), see Equation (4.2.7).
- In the fourth stage we build a *masked polar histogram*. In this stage we consider the dynamics of the vehicle, i.e the maximum turning radius. To decide if the directions to the left respectively right of the vehicle is blocked by obstacles, we use Equations (4.2.12) and (4.2.13). With this information we get two limit angles, $\varphi_l$ and $\varphi_r$, which defines the maximum steering angle to the left and right respectively of an obstacle. With the help of these two angles, the current orientation $\theta$ of the vehicle and the binary polar histogram $h_k^b$, we can build the masked polar histogram $h_k^m$, see Equation 4.2.14.
- The last step is to compute the required steering direction. This is done by checking the number of subsequent free sectors in the masked polar histogram $h_k^m$. If less than $s_{max}$ subsequent sectors are free, it is considered a *narrow candidate valley*, and we choose a centered direction of travel, see Equation (4.2.15). Otherwise, it is called a *wide candidate valley*, and here we can choose to go in more than one direction within the valley, see Equation (4.2.16). To select the best direction of travel from these candidate directions, we use the cost function in Equation (4.2.17).

# Chapter 5

# Combining path following and obstacle avoidance

As mentioned earlier, there is a slight complication when combining Pure Pursuit with the VFH algorithms. In order to combine path following with the obstacle avoidance algorithms, we have to take some steps to ensure that the vehicle gets the correct information depending on the situation. Figure 5.1 shows a flowchart of the crucial steps involved:

1. First we have to determine a Carrot Point, on a Look Ahead Distance ahead on the path.
2. Then we compute the corresponding direction to the Carrot Point.
3. If there is no obstacles nearby, calculate the steering angle based on the current goal direction. Otherwise, send the goal direction to the obstacle avoider (VFH or VFH+).
4. Let the obstacle avoidance algorithm determine a new goal direction.
5. If the new goal direction is not the same as the one computed in step 2, a steering angle is calculated directly towards the goal direction (as in Follow The Carrot), whether the Pure Pursuit algorithm is used or not. This is done to ensure that the vehicle steers directly towards the open space between obstacles, instead of plotting an indirect course to the goal direction as is usually done in the Pure Pursuit algorithm.
   If the new goal direction is the same as the previous one, it is sent back to the path follower which will compute the correct steering response from the vehicle.

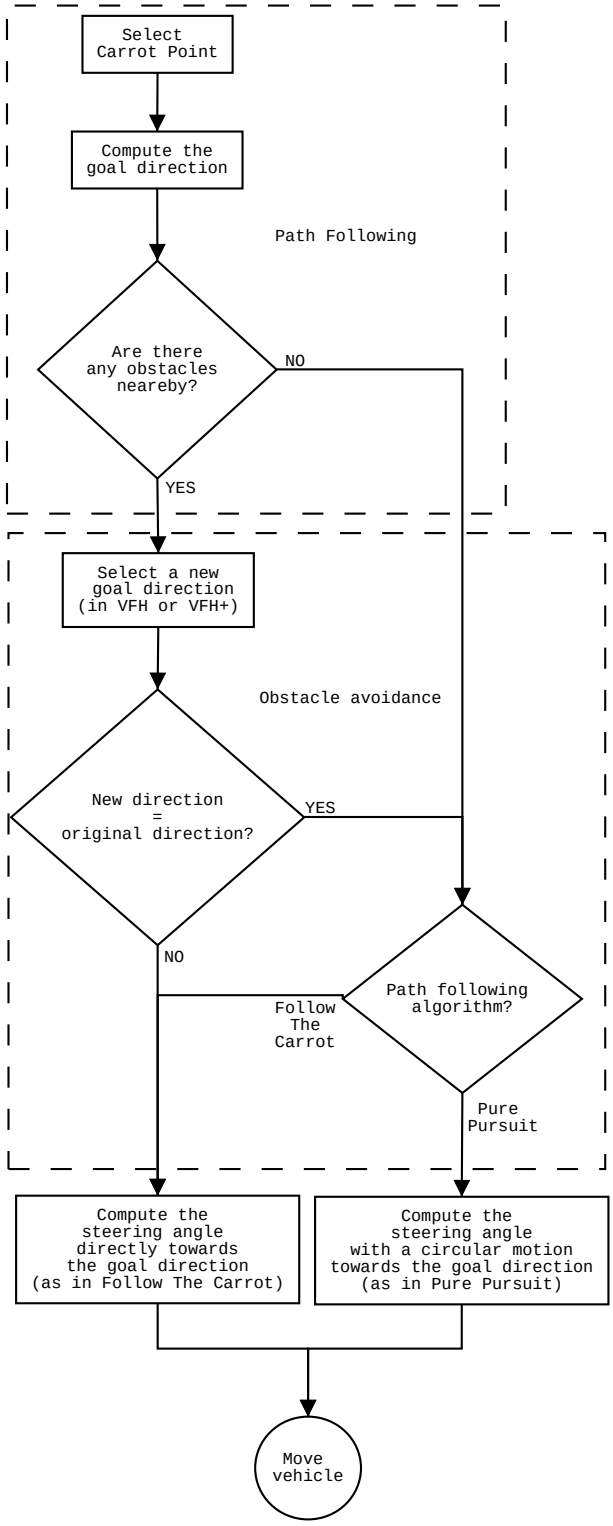


Figure 5.1: Flowchart over the process of combining path following and obstacle avoidance.

# Chapter 6

# System implementation and performance analyses

## 6.1 The simulation environment

In order to make the simulation easier to control, a Graphical User Interface (GUI) with all the crucial settings for the environment was added to the simulator. When first started, a standard setting will be displayed, se Figure 6.1. The GUI consist of three different areas:
The right area is where the actual simulation takes place. The path is surrounded by obstacles (the dots). The vehicle is represented by two rectangles connected by the joint in form of a filled circle, see Figure 6.2.

The top left area is where all settings for the simulation is controlled:

- Velocity - the velocity of the vehicle in m/s. The standard value is 0.5, and the normal range is about 0 to 6 m/s. If the input is not an integer, an error message will appear.
- Delta Time - the time between each timestep in seconds. If the input is not numeric, an error message will appear.
- Path - select between seven different paths. The path will be shown in the simulation window with corresponding obstacles. If a simulation is currently in progress, it will be aborted and the simulation window will be redrawn with the new settings.
- Path Tracking Algorithm - select *Follow The Carrot* or *Pure Pursuit*.

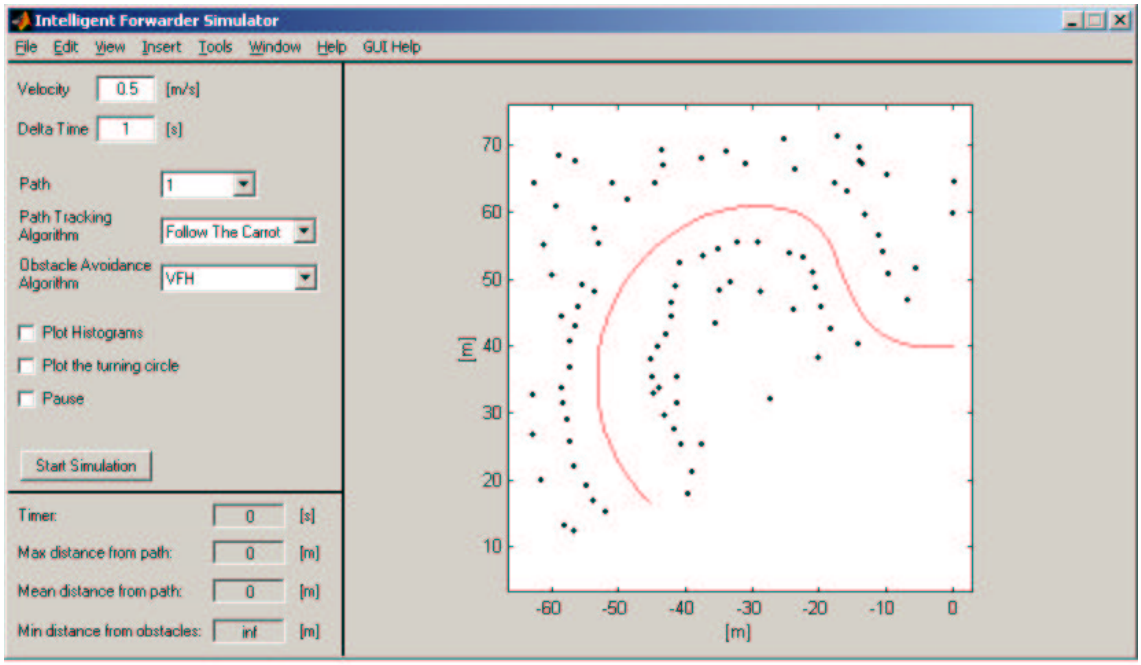


Figure 6.1: The Graphical User Interface for the simulator.

- Obstacle Avoidance Algorithm - select *VFH*, *VFH+* or *No Obstacles*. If the latter is selected, no obstacles will be plotted and the vehicle can follow the path without interruption.
- Plot Histograms - check this to display the histograms corresponding to each situation in the simulator.
- Plot the turning circle - Displays a circle showing the current turning radius of the vehicle.
- Pause - check this to pause the simulation. A new button *"Next Timestep »"* will appear that enables the user to step through the simulation one timestep at a time. Any button pressed while the simulator window is focused will accomplish the same thing.
- Start Simulation - starts a new simulation with the current settings from above. A new button *"Abort Simulation"*, will appear that aborts the simulation and redraws the simulation window with the current settings.

The bottom left area displays some information regarding distances between the vehicle, the obstacles and the path. The distance between the vehicle and an obstacle is measured from the eight corners of the vehicle, and may therefore not be entirely

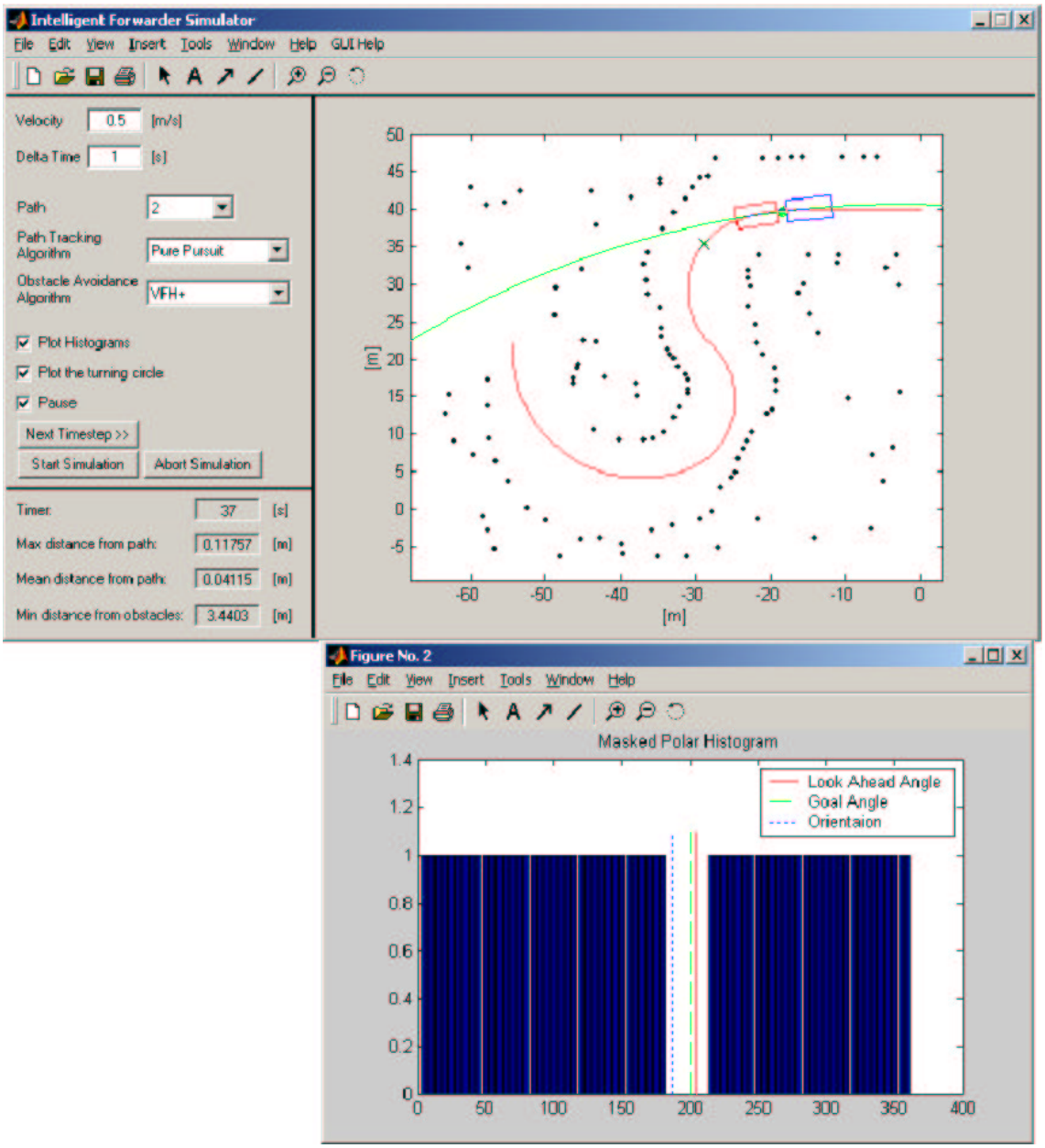


Figure 6.2: The vehicle on a path, surrounded by obstacles. The lower part shows a histogram for the current situation.

accurate (because some other point of the vehicle may be closer to the obstacle then the corners). The distance between the vehicle and the path is measured from the VCP (the joint). The *Timer*-field shows the time elapsed in simulated time. This means that for each timestep the timer will count up with Delta Time seconds. This way, the actual time it takes to simulate will have no effect on the timer.

A Help feature is also added in the program. At the menu bar, press *GUI Help*, then *Help*, and a help window will appear with information on how to handle the simulator. Under *GUI Help* is a menu called *About*, which will bring up some information about the program and the author.

## 6.2 Path Following

The difference in how Pure Pursuit and Follow The Carrot steer the vehicle can easily be seen in the simulator. Follow the carrot steers directly at the carrot point, and therefore the turning circle hits the path a bit to the side of the point, se Figure 6.3. The carrot point is marked by 'x' on the path in front of the vehicle. Pure Pursuit on the other hand steers so that the vehicle will hit the carrot point if it continues with the same steering angle, and therefore the turning circle in Figure 6.4 crosses the middle of the carrot point.

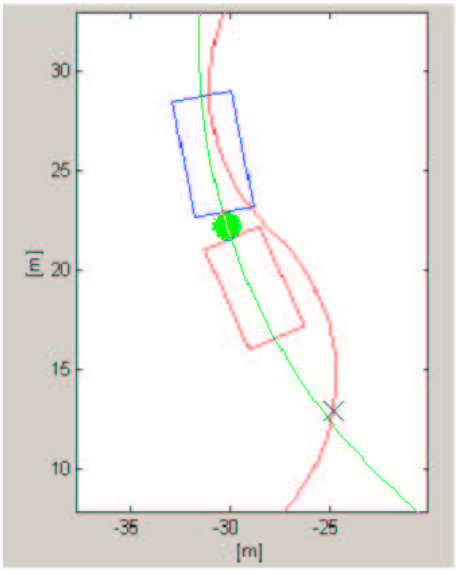


Figure 6.3: The steering circle for Follow The Carrot.

According to [Bar01], the vehicle tends to oscillate around the path with the Follow

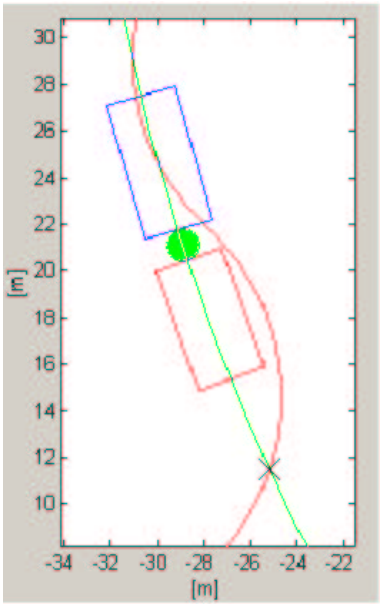


Figure 6.4: The steering circle for Pure Pursuit.

The Carrot method, especially for short Look Ahead Distances. This behavior can be noticed in the simulation as well, but the oscillation is not very large, especially not for the Look Ahead Distance used. The second drawback pointed out by [Bar01] is the tendency to cut corners. Because of the nature of the algorithm, it can not be avoided, but can be minimized by a shorter Look Ahead Distance.

The Pure Pursuit approach has similar drawbacks as Follow The Carrot. According to [Bar01] the oscillation problem is no big issue in Pure pursuit. However, [Cou92] points out that the smaller the Look Ahead Distance, the larger the oscillation around the path. However, no larger oscillations have been noticed in the simulation. The problem with cutting corners is less severe than in Follow The Carrot, but can still be quite large in certain situations. The problem dependence of the size of Look Ahead Distance can be studied by altering this distance in the simulator. To show this, tests with two different Look Ahead Distances have been performed. In Figure 6.5, Follow The Carrot is used, and the Look Ahead Distance is approximately twelve meters ahead of the vehicle. In Figure 6.6 we can se the behavior of the Pure Pursuit method with the same Look Ahead Distance, which is more than double the normal distance used in the simulator. Notice the limited ability to follow the curved path. Another interesting thing is that in this particular situation, Follow The Carrot is slightly better than Pure Pursuit, which would suggest that one algorithm is not always better than the other.

In Figures 6.7 and 6.8 the difference when having a normal Look Ahead Distance

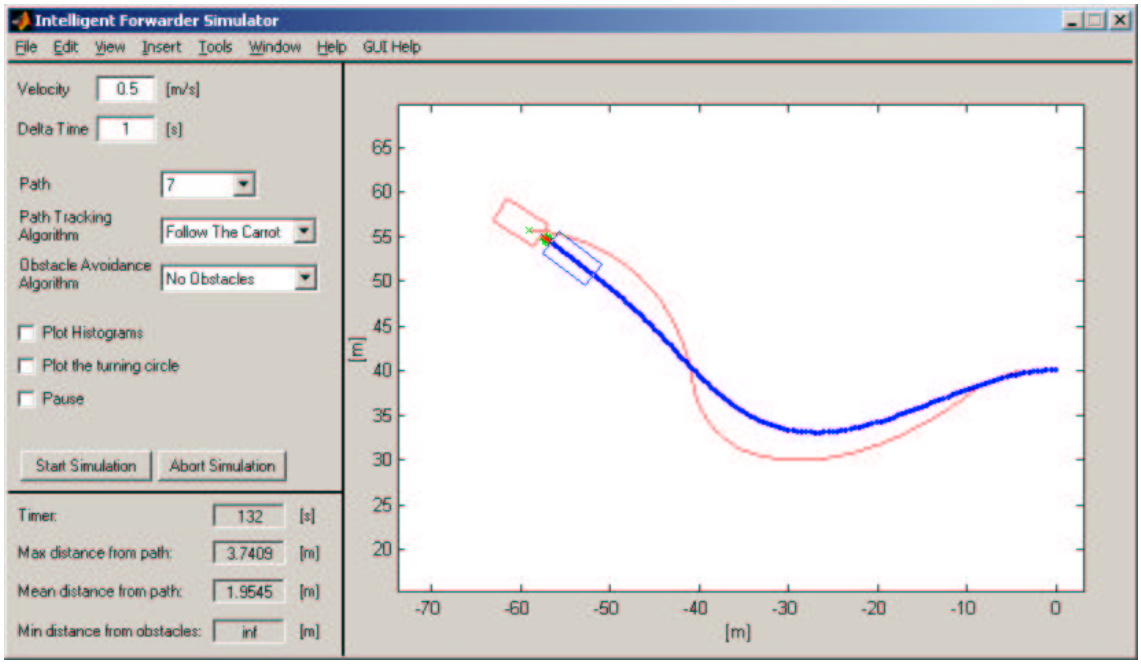


Figure 6.5: Follow The Carrot with approximately twelve meters look ahead.

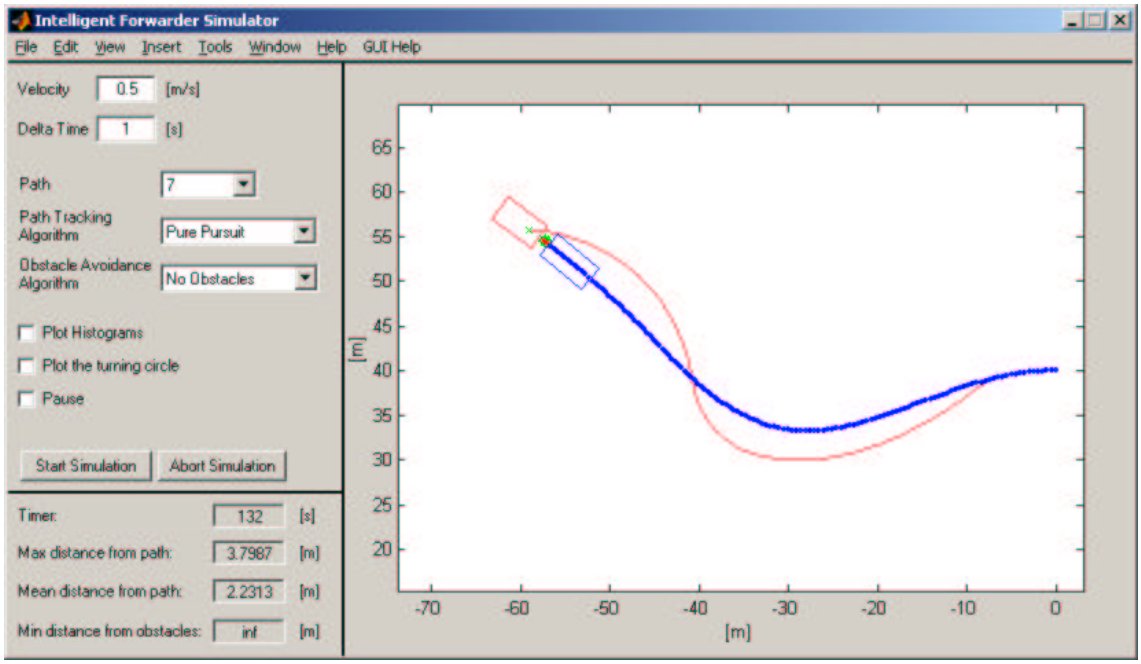


Figure 6.6: Pure Pursuit with approximately twelve meters look ahead.

(approximately five meters in front of the vehicle) can be seen. The ability to follow the path is greatly increased. The vehicle still takes short cuts, but significantly less than with larger look ahead. We can also observe that Pure pursuit has a slightly better performance than Follow The Carrot.

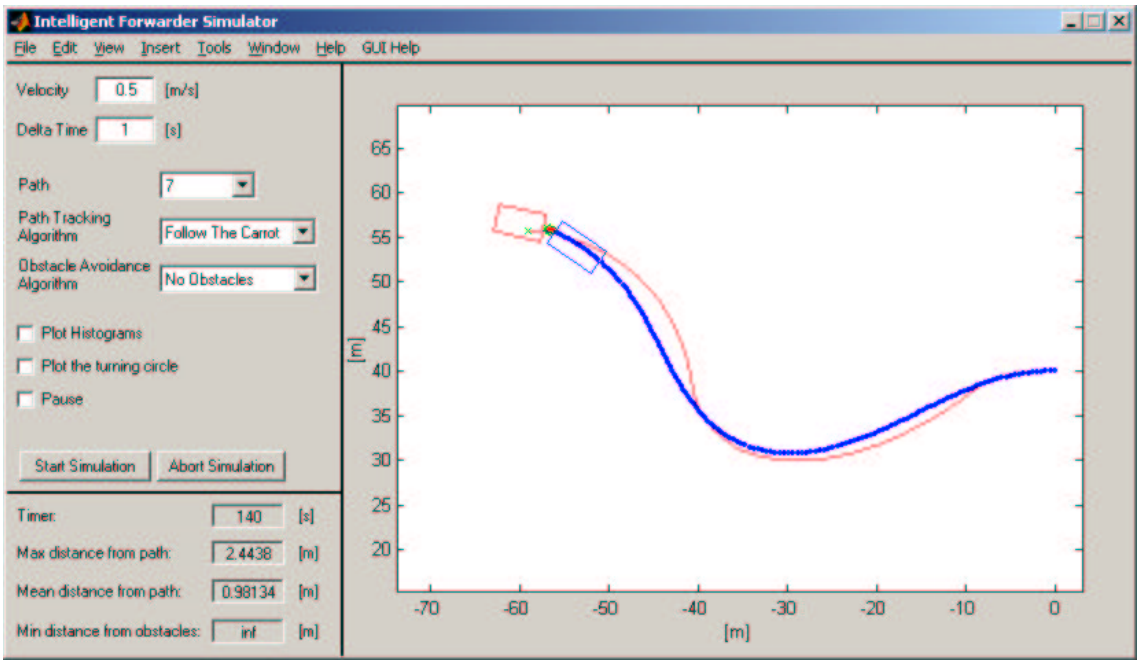


Figure 6.7: Follow The Carrot with approximately five meters look ahead.

## 6.3 Obstacle Avoidance

The ability to avoid obstacles depends greatly on the situation (i.e. the position and the number of obstacles). This makes it hard to verify that the algorithms actually work for all situations. They do work in most situations, but have problems in other situations as described below. One situation that should be fairly common in the forest is a path with trees on both sides distanced so that the vehicle has room to maneuver. Both VFH and VFH+ handle such a situation, see Figures 6.9 and 6.10. Notice that VFH is further from the path and closer to the obstacles than VFH+. In the following two sections, some things that differ between the two algorithms will be pointed out.

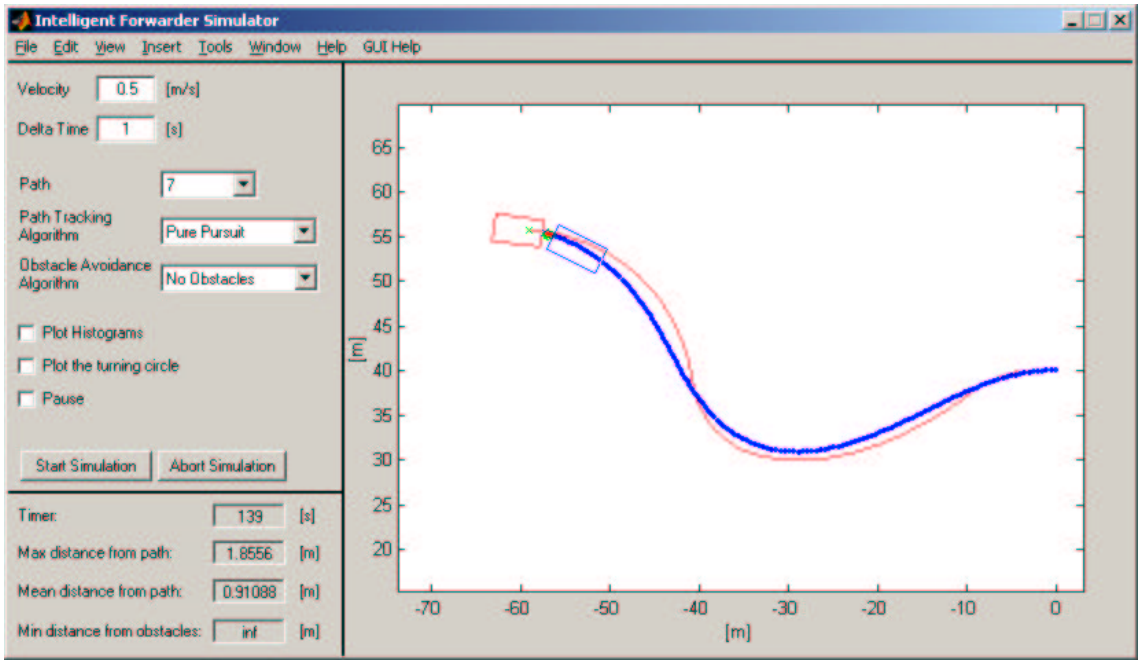


Figure 6.8: Pure Pursuit with approximately five meters look ahead.

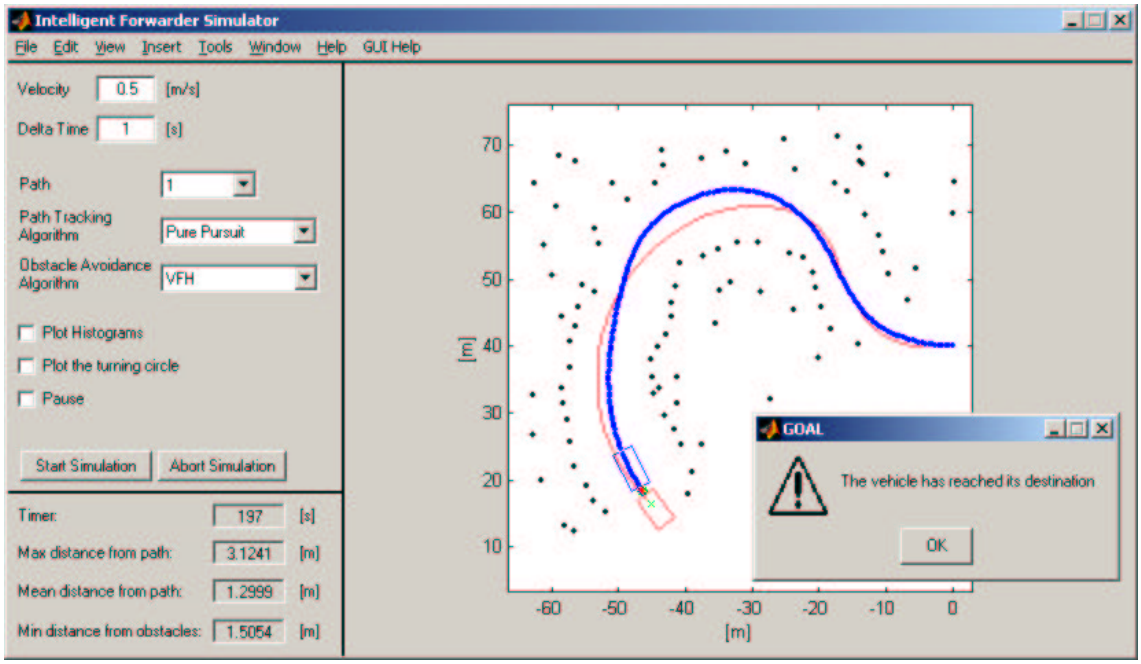


Figure 6.9: VFH handles a standard forest road without problem.

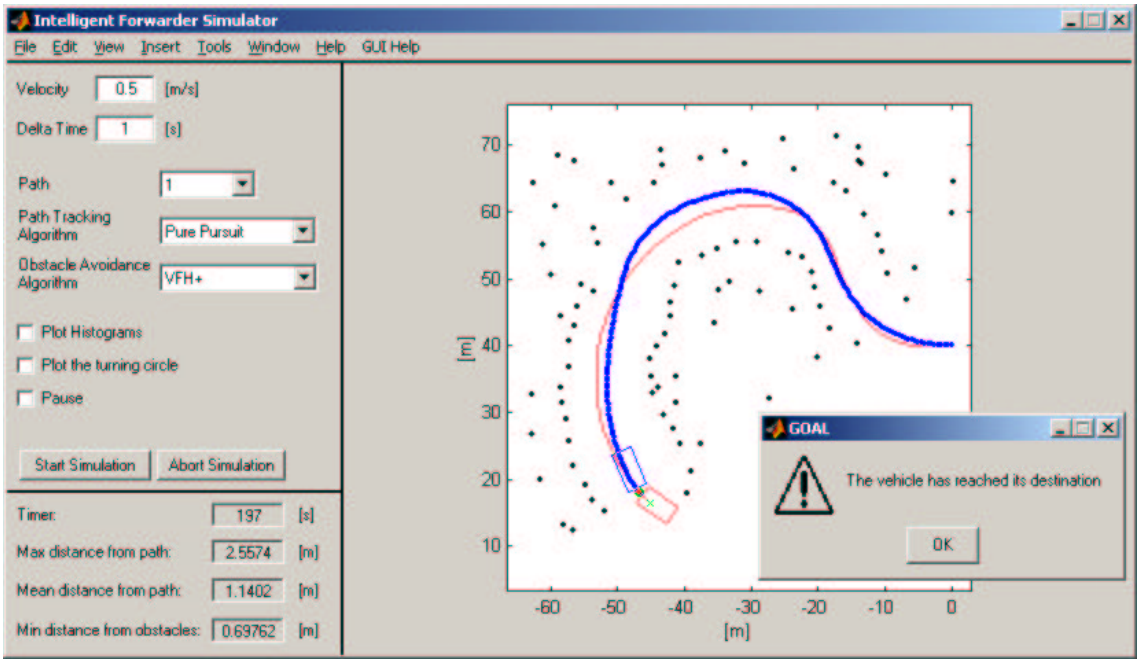


Figure 6.10: VFH+ handles a standard forest road without problem.

### 6.3.1 VFH

One of the advantages with the VFH method over the VFH+ method, is that it can guide the vehicle between obstacles which are very close together. This can also be seen as the method's greatest weakness. When the vehicle succeeds in going safely between tight spaces it has an advantage compared to the VFH+ method, but this also leaves possibilities for errors. The obstacles are very close to the vehicle and therefore it has very little space to move sideways. An articulated vehicle is a bit special in the sense that, even if the velocity is zero, the vehicle can move sideways just by turning. In Figure 6.11, there is an example of the vehicle turning too much when being close to an obstacle, and thereby crashing into it with the rear end. Figure 6.12 is an example of the advantage of being able to move through tight spaces (compare to Figure 6.16). The reason for the vehicle doing an extra lap around the goal point is that, even with maximum turning angle, the vehicle is not able to get close enough to the goal point and therefore have to try going around the obstacles one more time.

The most significant drawback with VFH in the performance tests is that it does not take into consideration the dynamics of the vehicle. This means that VFH presumes the vehicle can go in any direction at any given time. This is true for small robots with differential steering, but a large forest machine needs space to

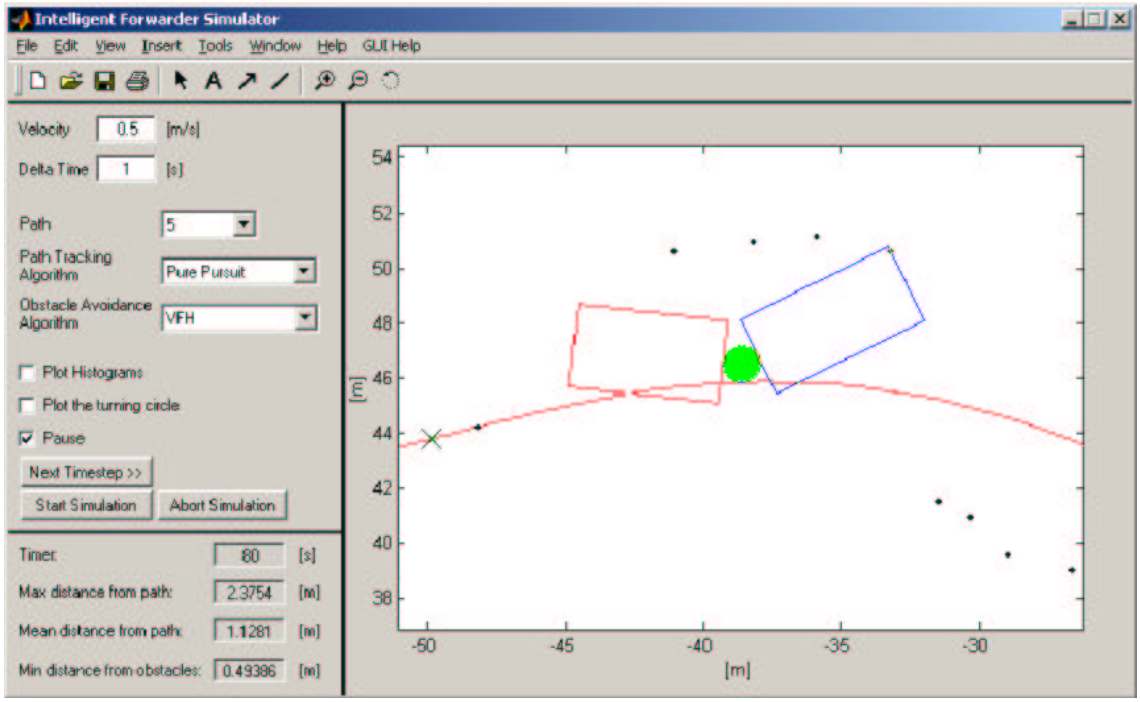


Figure 6.11: The rear end hits an obstacle when turning.

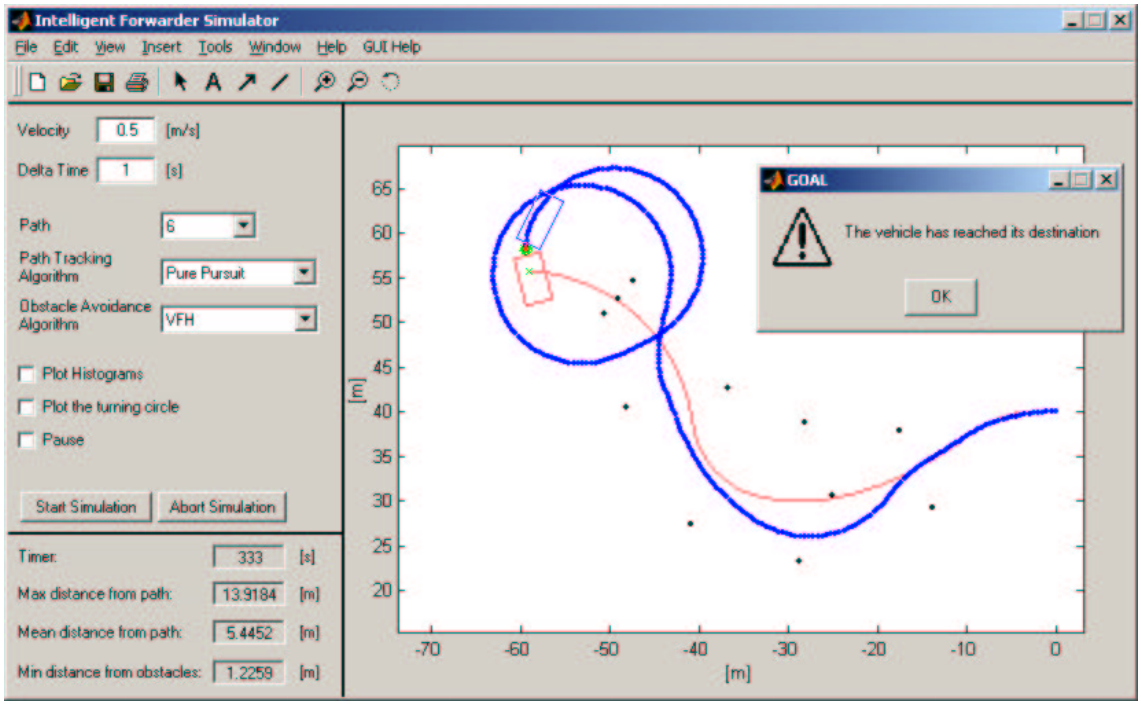


Figure 6.12: The VFH method is good at moving through tight spaces.

turn around and continue in a different direction. This becomes a problem when the VFH method sense that all sectors in front of the vehicle are blocked by obstacles and simply propose driving back the way it came (or another direction the vehicle can not reach without turning around). When this happens, VFH does not consider the obstacles the vehicle must drive through to get to the new direction, because the proposed heading is still free and VFH keeps proposing that heading. Figure 6.13 is an example of this. The vehicle is traveling with a heading of 150° (from the lower right to the upper left in the figure) and VFH propose a new heading of 300°, resulting in the vehicle crashing into the obstacles to the right. Something worth noting is that in this situation, the VFH+ method is better at going through tight spaces (compare to Figure 6.17).

As mentioned earlier, there is a slight problem in combining the VFH-methods and Pure Pursuit (see the section about Pure Pursuit for more details). Normally this is not a problem, but in certain situations it does constitute a serious problem. This has been noticed with VFH, but not with VFH+. This is probably because VFH+ is safer method than VFH. The problem is that even if VFH reports that the carrot point is free from obstacles, Pure Pursuit does not steer directly toward that point by definition. This could mean that there still are obstacles in the direction that Pure Pursuit wants to go. The lack of considering the vehicle dynamics could also be one possible explanation. Figures 6.14 and 6.15 shows the difference between using Follow The Carrot and Pure Pursuit in a situation like this.

### 6.3.2 VFH+

As mentioned in the previous section, the VFH+ method takes a safer path than VFH. Sometimes this makes the vehicle take a longer way around instead of going in between obstacles. Figure 6.16 shows an example of the vehicle going around obstacles where VFH manage to go between them (compare to Figure 6.12).

Though VFH+ does take the long way around sometimes, it still has the ability to go through narrow openings between obstacles, see Figure 6.17. The figure also shows the three histograms created in VFH+ for this specific situation.

Because VFH+ takes into account the dynamics and size of the vehicle, it sometime happens that the algorithm signals failure when in fact the vehicle could very well proceed without any problems. You could say that the algorithm becomes too safe sometimes. This shows in the Figures 6.18 and 6.19, where the vehicle manages to safely traverse the path with VFH, but VFH+ signals that the vehicle is trapped. Notice that the Masked Polar Histogram in Figure 6.18 has no free sectors. Thereby

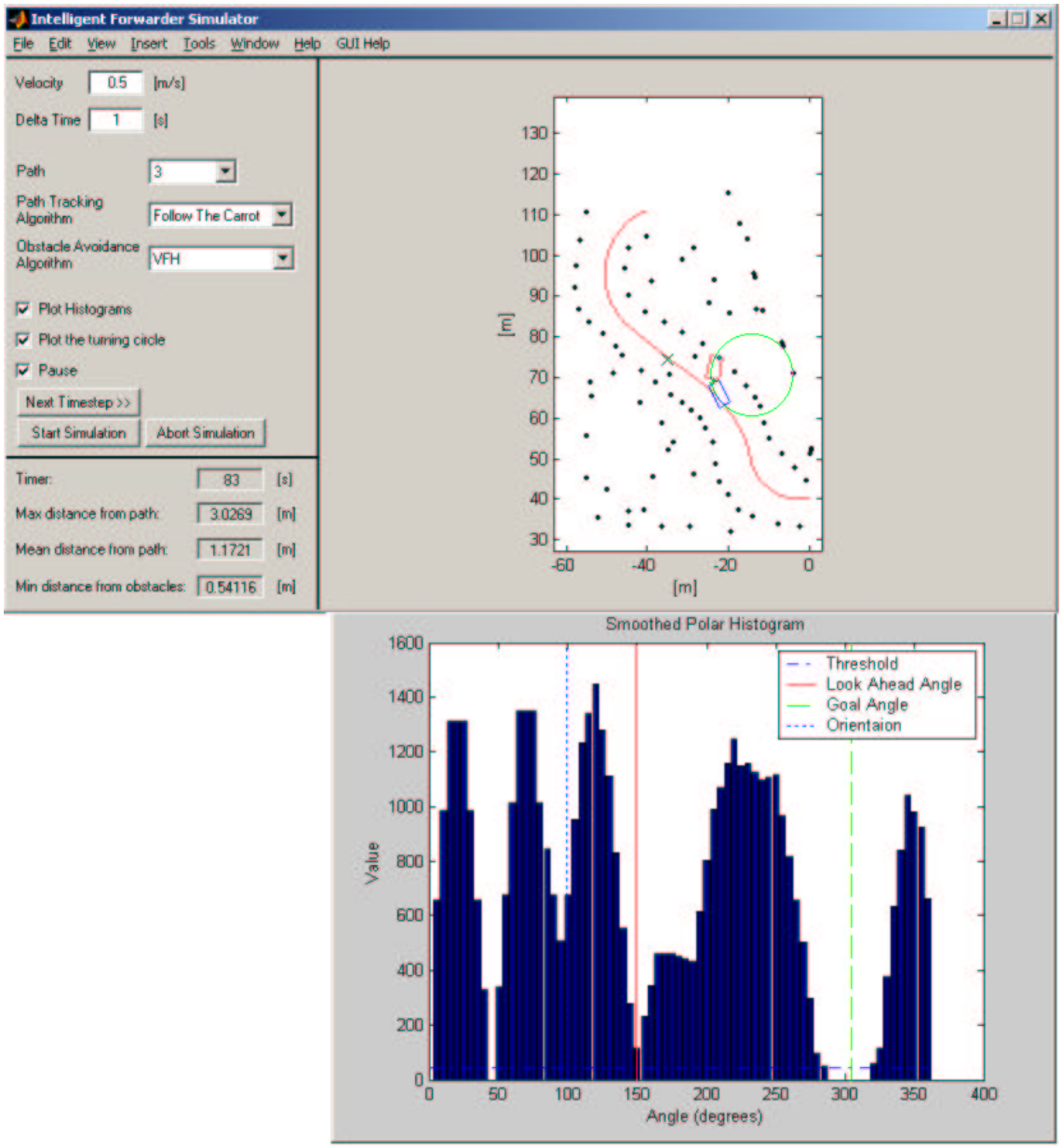


Figure 6.13: The vehicles dynamics are ignored in VFH.

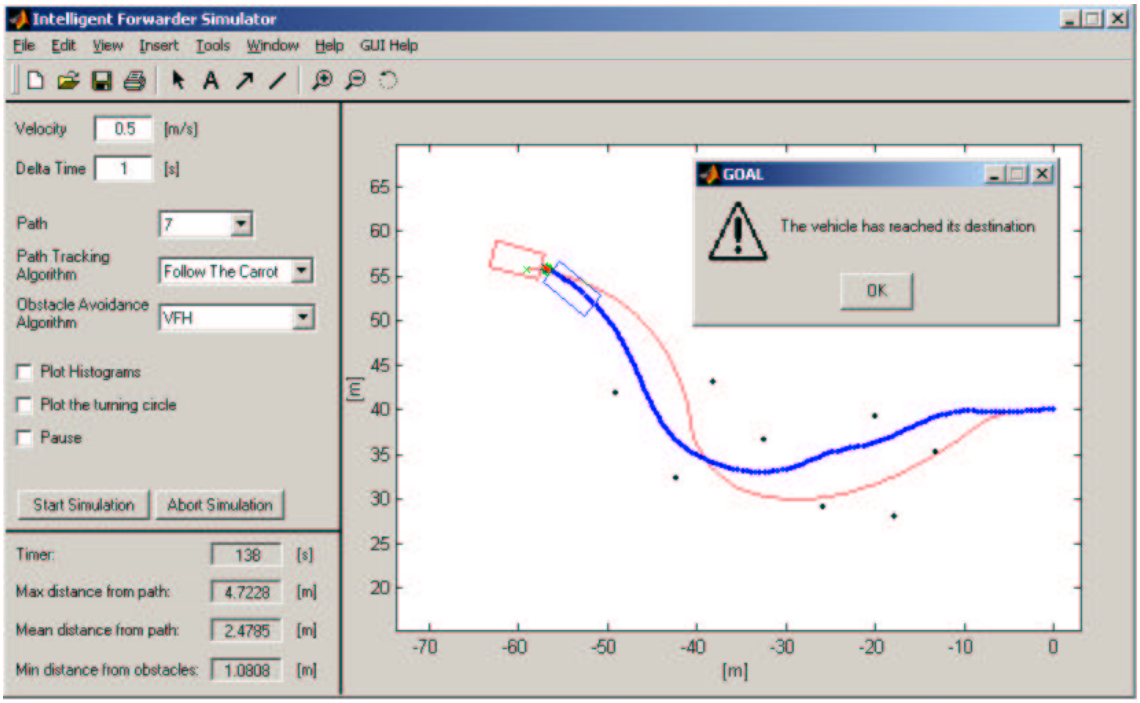


Figure 6.14: VFH with Follow The Carrot.

VFH+ can signal error and halt the vehicle.

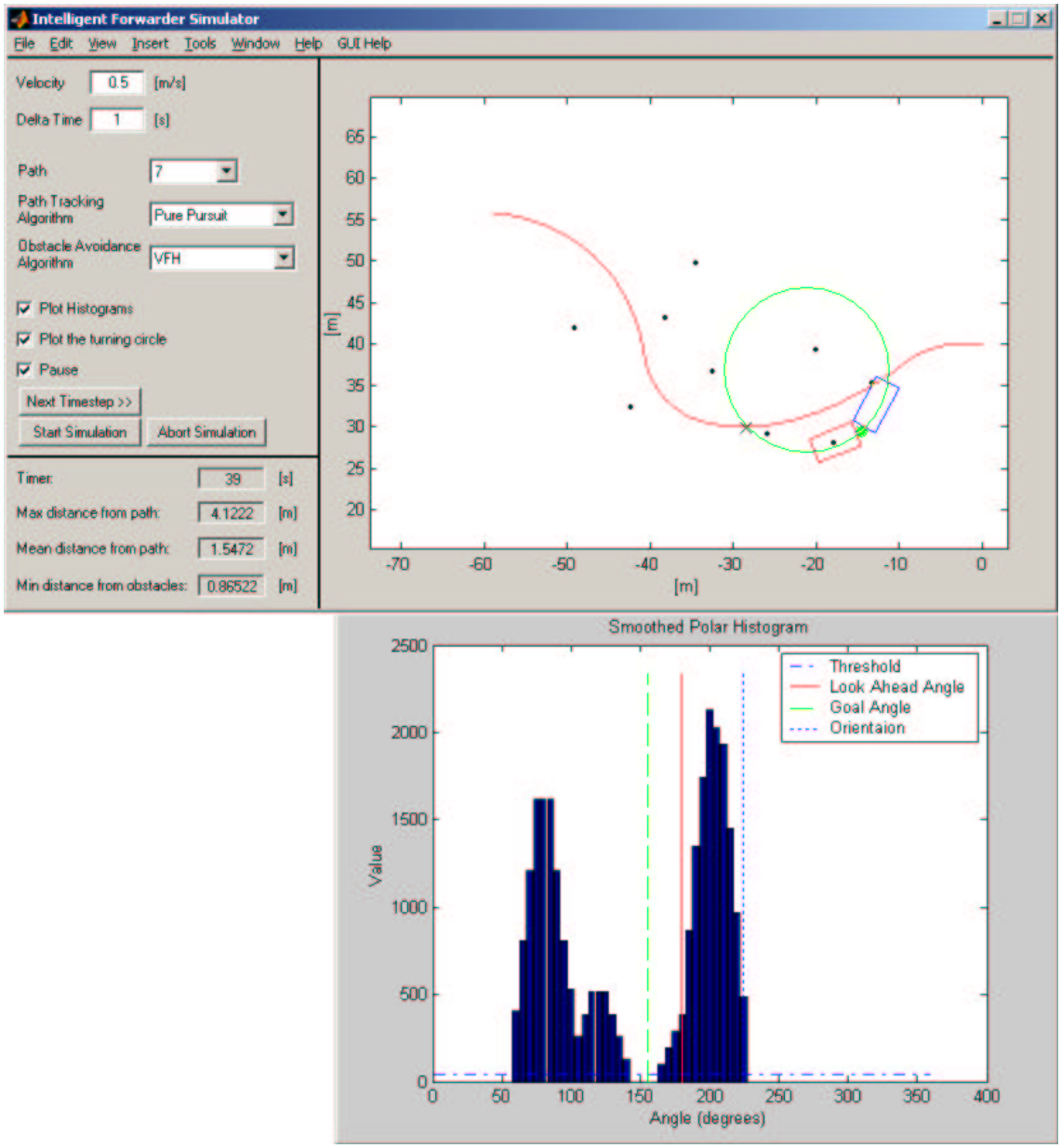


Figure 6.15: VFH with Pure Pursuit does not always work.

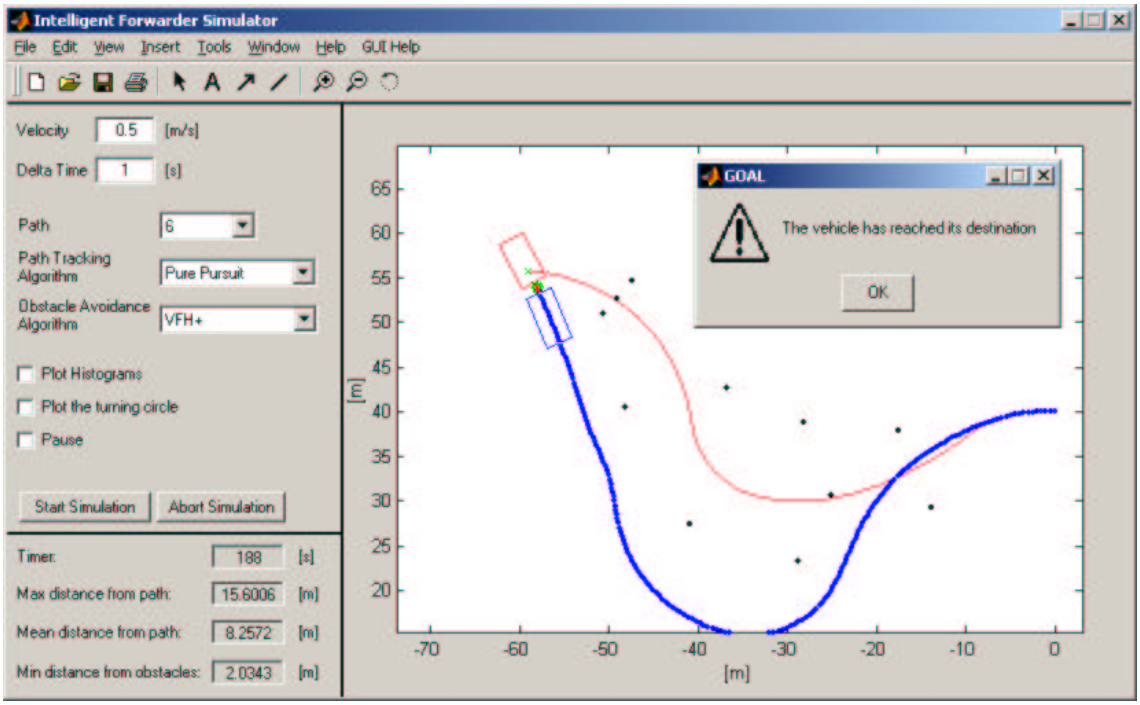


Figure 6.16: VFH+ playing it safe.

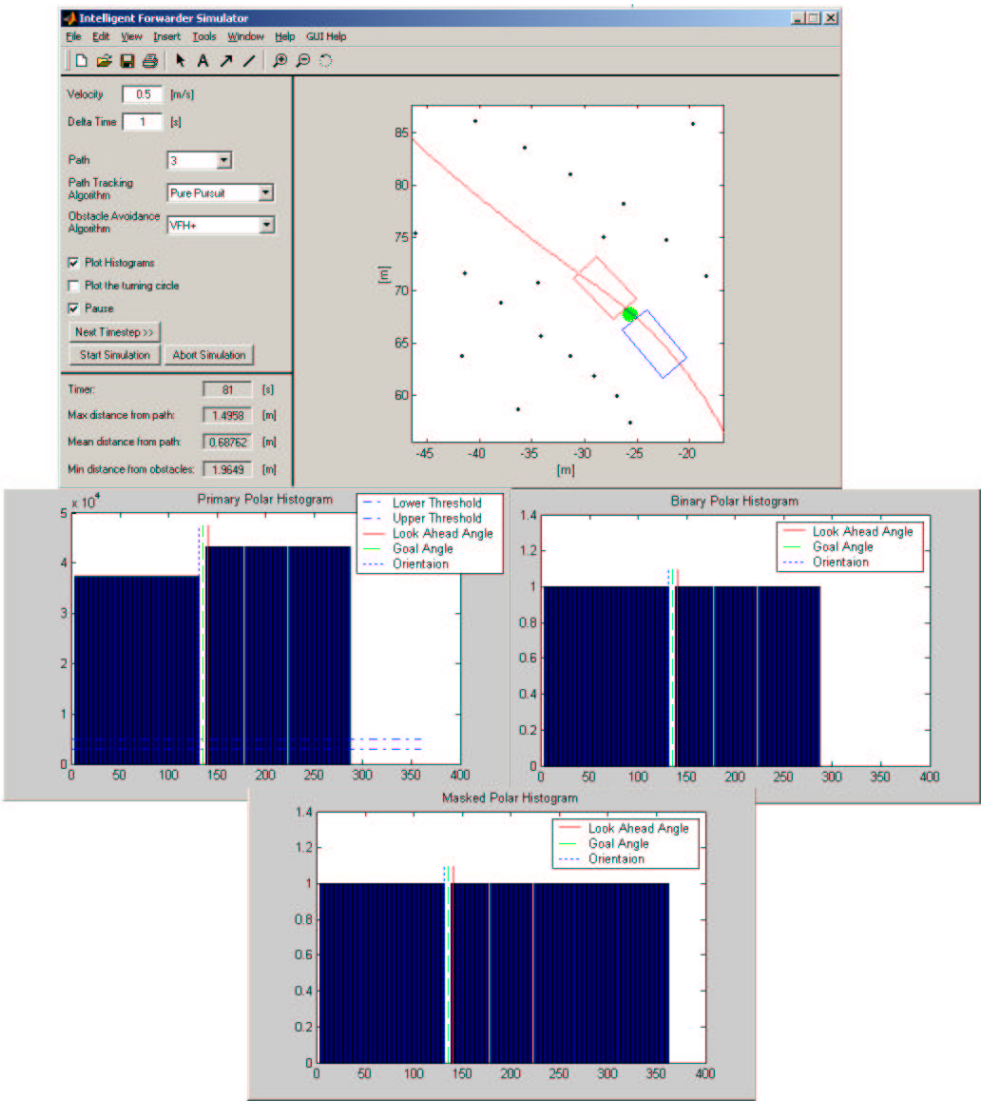


Figure 6.17: VFH+ going through a very narrow opening.

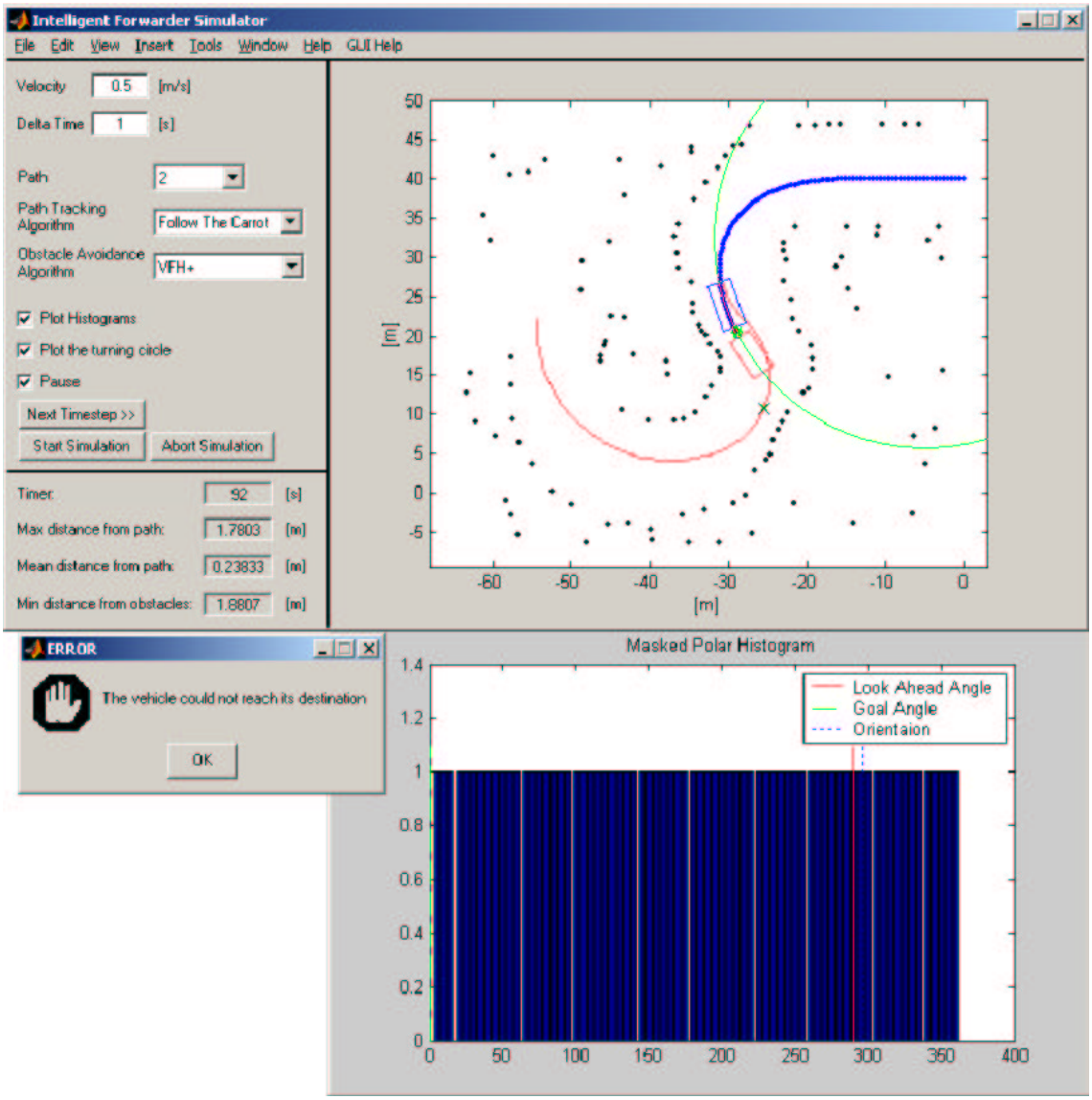


Figure 6.18: VFH+ signals error and halts the vehicle.

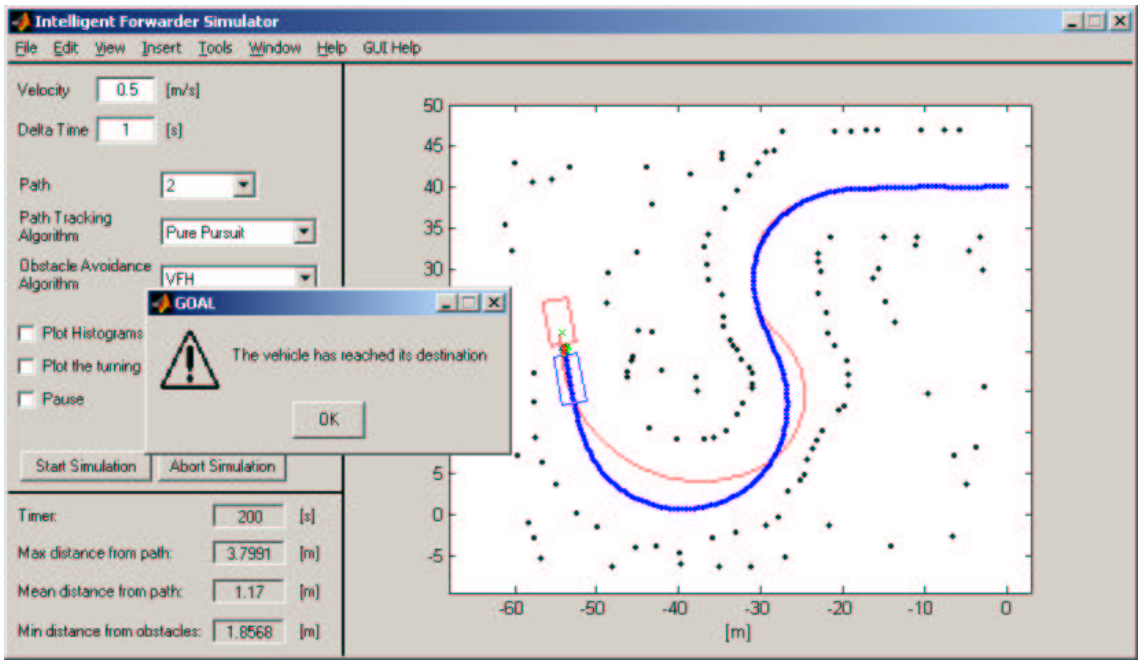


Figure 6.19: VFH safely takes the vehicle through the course.

# Chapter 7

# Conclusions

When following paths without any obstacles present, the Pure Pursuit algorithm performs better than Follow The Carrot with regards to the distance to the path, and the tendency to cut corners. The problem comes when combining Pure Pursuit with obstacle avoidance. The VFH-methods give a steering angle directly at the goal point, but Pure Pursuit lays a circular arc at the goal point and thereby wants to steer at a slightly different direction. This can not be tolerated when there are obstacles in the vicinity. Because of this, the vehicle will go in the direction proposed by Pure Pursuit only if the VFH methods propose the same direction. If the direction proposed by the obstacle avoidance method is another, the vehicle will steer directly in the goal angle. In this situation there is no difference between the two path following algorithms, and therefore no reason using Pure Pursuit. No other solution has been found to this problem. Therefore, in the current implementation, Pure Pursuit is only useful while there are no obstacles in the way; otherwise it works just like Follow The Carrot. Of course there is no harm in having an algorithm that gets better when no obstacles block the path, but if it could work better all the time it would be even better.

Regarding obstacle avoidance, VFH+ is the safer alternative compared to VFH. The lack of considering vehicle dynamics in the VFH method is unacceptable as shown in the examples above. Instead of signaling error when all sectors in front of the vehicle is blocked by obstacles, the VFH method proposes a heading in a completely new direction that may not be reachable without hitting any obstacles, as in Figure 6.11. The drawback of VFH+ is that it sometimes takes the long way around obstacles, despite the fact the vehicle could fit between them. This could probably be solved by tuning the algorithm further, together with other changes. A compensation that is proposed in [UB98] is that the obstacles is enlarged according to the dimensions and

the momentary orientation of the robot. This way the vehicle could fit in between to obstacles, while still not hitting them when driving straight at them.
It was very difficult to get both algorithms to work, but VFH was even harder because of the many parameters that must be tuned. Neither of the algorithms is very stable in the sense that if implemented on a different vehicle it must be re-tuned. But when successfully tuned, the VFH+ method performed better than VFH.

# Chapter 8

# Limitations of the simulator

One of the most serious limitations in the implementation is that the vehicle can turn from $+40°$ to $-40°$ in one timestep. In reality the turning rate depends on the forward velocity, and of course the length of one timestep. This probably has an impact on the obstacle avoidance, since the vehicle may not be able to turn away from an obstacle quick enough. This is something that must be considered when tuning the obstacle avoidance algorithm, and is no big problem to overcome.

Another limitation is that the position of the joint remains fixed, while front and rear end moves when turning the vehicle. On a real articulated forwarder, the joint moves as well as the rest of the vehicle. But which part moves most depends on how much load the forwarder has, and on the ground conditions. So to simplify the calculations involved, the joint is assumed to stay at the same position.

A feature that has not been implemented is noise in the vehicles position. If the vehicle gets its position from GPS, the accuracy is about a half meter (depending heavily on the price tag of the equipment). This means that the position the vehicle reads may not be the real position, and the path following algorithm will be fed with false data. The impact on path following and obstacle avoidance would be interesting to study. For obstacle avoidance the vehicle depends on the on-board sensors, and is therefore less sensitive to position noise. But a decision to steer left or right depends on information on where the carrot point, and thereby the path, is. If the path follower gives false information about where the path is, the obstacle avoider could make the wrong decision.

# References


[Ami90] Omead Amidi. *Integrated Mobile Robot Control.* Technical Report CMU-RI-TR-90-17, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, May 1990.

[Bal00] Sir Robert Stawell Ball. *A Treatise on the Theory of Screws. Cambridge University Press, Cambridge, United Kingdom*, 1900.

[Bar01] Matthew J. Barton. *Controller Development and Implementation for Path Planning and Following in an Autonomous Urban Vehicle.* Undergraduate thesis, University of Sydney, November 2001.

[BK91] J. Borenstein and Y. Koren. *The Vector Field Histogram - Fast Obstacle Avoidance for Mobile Robots. IEEE Transactions on Robotics and Automation*, 7(3):278–288, 1991.

[Cou92] R. Craig Coulter. *Implementation of the Pure Pursuit Path Tracking Algorithm.* Technical Report CMU-RI-TR-92-01, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, January 1992.

[Hel02] Thomas Hellström. *Autonomous Navigation for Forest Machines.* Pre-study, University of Umeå, aug 2002.

[KB91] Y. Koren and J. Borenstein. *Potential field methods and their inherent limitations for mobile robot navigation. Proceedings of the IEEE Int. Conf.on Robotics and Automation*, 1991.

[Mäk01] Hannu Mäkele. *Outdoor navigation of mobile robots.* Technical report, Heksinki University of Technology, November 2001.

[sof02] softSurfer. *About Lines and Distance to a Line*, 27 September 2002, `http://geometryalgorithms.com/Archive/algorithm_0102/`.

[UB98] I. Ulrich and J. Borenstein. *VFH+: Reliable Obstacle Avoidance for Fast Mobile Robots. IEEE Int. Conf. on Robotics and Automation*, pages 1572–1577, May 1998.

[Wit00] Jeffery S. Wit. *Vector Pursuit path Tracking for Autonomous Ground Vehicles.* PhD thesis, University of Florida, 2000.

[WST+85] R. Wallace, Anthony (Tony) Stentz, Charles Thorpe, Hans Moravec, William Red L. Whittaker, and Takeo Kanade. *First Results in Robot Road-Following.* In *Proceedings of the International Joint Conference on Artificial Intelligence*, 1985.

# Appendix A

# Variables used

$\phi$ Steering Angle
$\theta$ Orientation of the vehicle

## A.1 Simulator

| | |
|---|---|
| $\delta$ | The angle moved each timestep |
| $d$ | The distance traveled each timestep |
| $v$ | The current velocity of the vehicle |
| $\Delta t$ | One timestep |
| $C$ | The circumference of the turning circle |
| $r$ | The radius of the turning circle |
| $x_0, y_0$ | Coordinates for the Vehicle center Point (VCP). |
| $circX, circY$ | The center of the turning circle |

## A.2 Path Following

$\psi$ The Look Ahead Angle
$\epsilon$ The error angle in Follow The Carrot
$\gamma$ The curvature in Pure Pursuit

## A.3 Obstacle Avoidance

| | |
|---|---|
| $C^*_{i,j}$ | Certainty value of active cell (i,j) |
| $d_{i,j}$ | Distance between active cell (i,j) and the VCP |
| $m_{i,j}$ | Magnitude of the obstacle vector at cell (i,j) |
| $x_0, y_0$ | Present coordinates of the VCP |
| $x_j, y_j$ | Coordinates of active cell (i,j) |
| $\beta$ | Direction from active cell (i,j) to the VCP |
| $\alpha$ | Angular resolution |
| $n$ | Number of sectors |
| $r_r$ | The distance from the VCP to the furthest point on the vehicle |
| $r_{r+s}$ | Radius of an enlarged obstacle |
| $\gamma_{i,j}$ | Enlargement Angle |
| $\tau$ | Threshold for a valley |
| $s_{max}$ | Limit for wide valley |
| $\varphi_r$ | Maximum steering angle to the left respectively right of an obstacle |
| $r_r$ and $r_l$: | Radius to the right and left trajectory centers respectively |
| $\Delta x, \Delta y$: | The distance from the VCP to the trajectory centers |
| $\Delta x(i), \Delta y(j)$ | The distance between obstacle cell $i, j$ and the VCP |
| $c_n$ | A set of candidate directions |
| $\mu_1, \mu_2, \mu_3$: | cost terms |
| $k_t$: | target (look ahead) sector |
| $k_{n,i-1}$: | previous selected sector |
| $\psi$ | selected target angle |