# AROB Practical Work

Ignacio Herrera Seara, Juan García-Lechuz Sierra

***Abstract*— Practical work abstract. 200 words approx.**

## I. Introduction

The ultimate goal of research on mobile robots is that they have sufficient autonomy to be able to complete tasks in different scenarios despite the complexity of the environment. This complexity includes scenarios in which there are moving objects or unforeseen changes to which the robot must adapt by means of the information obtained by its sensors. As we have studied during the course, different reactive navigation strategies have been developed to achieve this goal. In this work, specifically, the technique developed by the Robotics Group of our university is studied, as it represents an improvement over its predecessors, both in the way of interpreting the environment and in the calculation of the actions to move through it. This technique is based on the identification of a series of concrete situations and the application of their corresponding actions, which allows it to reduce complex situations into simple actions reachable by the robot. The geometric implementation of the Nearness Diagram [1] [2] technique is based on some basic ideas of the Vector Field Histogram, to identify the proximity of obstacles to the robot and the areas through which it can move.

## II. Algorithm description

As mentioned in the introduction, this algorithm consists of two main parts in its execution. The first is the interpretation of the environment and the second is the development of a navigation strategy based on the interpretation.

### *A. ENVIRONMENTAL INFORMATION EXTRACTION AND INTERPRETATION*

The environmental information extraction is carried out in three steps:

1) Two nearness diagrams are constructed (the PND and the RND)(Fig. 1).
2) The **PND** (the nearness of the obstacles to the central point) is analyzed to identify regions and to select one of them.
3) The **RND** (the nearness of the obstacles to the robot bounds) is analyzed to evaluate the robot safety situation.

If we consider **b** $\in$ A the point of the robot which is used as the center of sectors (center of the robot), **n** the number of sectors (n=144 in this work, 2.5º being the angle of each sector), $\varphi$ the robot orientation in the global reference, and $\delta$ a function that, when given a robot configuration **q** and a point **b**, calculates a vector such that $\delta_i(q;b)$ is the smallest distance to an obstacle in the sector i. Then the nearness diagrams can be defined as:

**Nearness Diagram from the central Point(PND)**

$$PND : C_{free} \times A \rightarrow (\mathbb{R}^+ \cup \{0\})^n$$
$$(q,b) \quad \rightarrow \quad (D_1, ..., D_n)$$

$$if\ \delta_i(q,b) > 0, \quad D_i = PND_i(q,b) = d_{max} + l - \delta_i(q,b)$$
$$otherwise, \quad D_i = PND_i(q,b) = 0$$

Where:

- $d_{max}$: maximum value of $\delta$, representing the maximum range of the sensor used.
- l: maximum distance between two points of the robot (being the diameter for a circular robot)

**Nearness Diagram from the Robot (RND)**

$$RND : C_{free} \times A \rightarrow (\mathbb{R}^+ \cup \{0\})^n$$
$$(q,b) \quad \rightarrow \quad (D_1, ..., D_n)$$

$$if\ \delta_i(q,b) > 0, \quad D_i = RND_i(q,b) = d_{max} + E_i(b) - \delta_i(q,b)$$
$$otherwise, \quad D_i = RND_i(q,b) = 0$$

Where:

- $E$: is an enlarging function that depends on the robot geometry. The value of this function in each sector $E_i(b)$ is the robot radius for a circular robot.

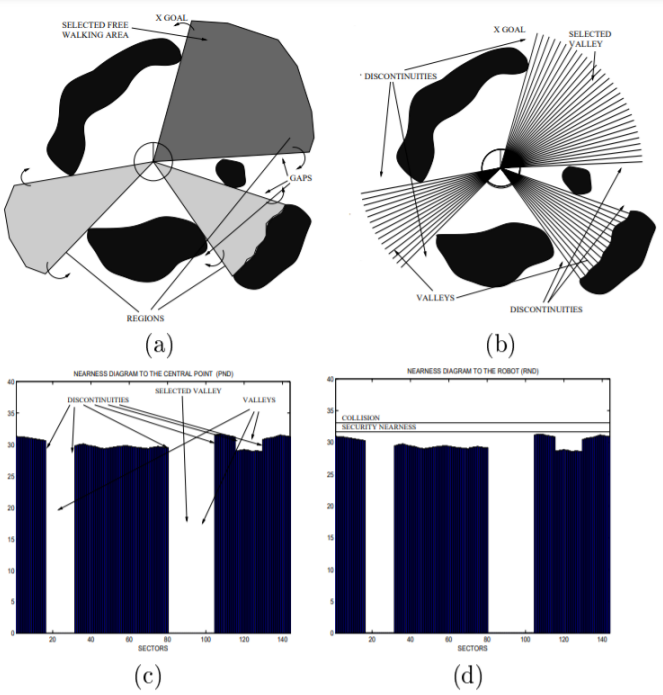


Fig. 1. a) Gaps, regions and the selected free walking area. b) Discontinuities, valleys and the selected one mapped on the environment. c) PND generated. d)RND generated. [2].

The following steps are taken to analyze the PND:

1) Search for gaps in the environment.
2) Obtain regions of the free space (valleys).
3) Select the valley with minimal sectorial distance, to the goal sector, and which is navigable. This is the free walking area.

On the other hand, by analyzing the RND we can distinguish between two safety situations:

- **Low Safety situation (LS)**: When, in the RND, at least one sector exceeds the security nearness.
- **High Safety (HS)**: Otherwise.

From them, five general situations can be defined, which cover all possibilities among the robot location, goal location and obstacle configuration, if checked in order. Decision tree is shown in figure 2.

There are two LS situations:

1) **Low Safety 1 (LS1)**: When there is at least one sector that exceeds the security nearness in the RND, only on one side of the rising discontinuity, which is closer to the goal sector, of the selected valley.
2) **Low Safety 2 (LS2)**: When there is at least one sector that exceeds the security nearness in the RND, on both sides of the rising discontinuity, which is closer to the goal sector, of the selected valley.

There are three HS situations:

1) **High Safety Goal in Valley (HSGV)**: When the goal sector belongs to the selected valley.
2) **High Safety Wide Valley (HSWV)**: When the selected valley is wide (this is greather than $s_{max}$, where $s_{max} = n/4$) and the goal is not inside.
3) **High Safety Narrow Valley (HSNV)**: When the selected valley is narrow and the goal is not inside.

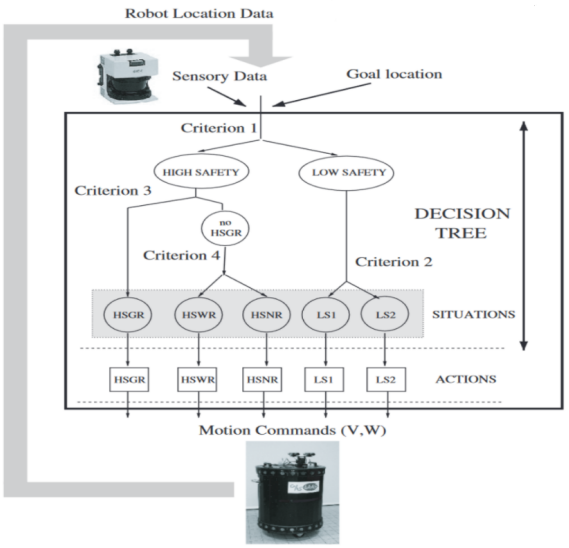


Fig. 2. ND closed loop and decision tree [3].

### B. NAVIGATION STRATEGY

As described in one of the developer's publications [2], the navigation strategy is based on two assumptions: the first, that it is very difficult to solve the local navigation problem with a single motion heuristic, due to the complexity of the environment. The second, the direct use of the goal location in a motion heuristic can have a counterproductive effect. Due to this last assumption, the goal location is directly used only in one of the five situations obtained in the analysis and interpretation of the environment.

Once one of the situations has been recognized, ND calculates the translational and rotational velocity in each sample time as motion commands for the mobile robot as follows:

1) **Translational velocity direction ($\theta$)**: Obtained as the bisector of $s_\theta$. Being $s_\theta \in \Re$ the solution sector which computation will be studied in the implementation.
2) **Translational velocity module (v)**: Calculated depending on the safety situation of the robot.
   - High Safety:

$$\mathbf{v} = v_{max} * Abs(1 - \theta/\frac{\pi}{2}) \tag{1}$$

   - Low Safety:

$$\mathbf{v} = v_{max} * \frac{d_{obs}}{d_s} * Abs(1 - \theta/\frac{\pi}{2}) \tag{2}$$

   Being $d_{obs}$ the distance of the closer obstacle to the robot bounds and $d_s$ the security distance.
3) **Rotational velocity (w)**: Calculated from the direction of the translational velocity:

$$\mathbf{w} = w_{max} * \theta/\frac{\pi}{2} \tag{3}$$

Being $w_{max}$ the maximum rotational velocity.

## III. IMPLEMENTATION DETAILS

The ND has been implemented as a local planner using the Navigation Stack of the ROS Melodic version. The planner receives as input the path given by the global planner, the odometry of the robot and the local costmap, and calculates the linear and angular velocity of the robot as a result. All calculations have been made with reference to the robot and all operations on sectors are performed by applying the modulus function to give continuity to the diagrams.

The Algorithm 1 shows the proposed pseudocode implementation. Where $x_r$ is the robot position, $x_g$ is the currrent goal position and $L$ is the obstacle position list.

**Algorithm 1** Nearness Diagram

**Require:** $x_r$, $x_g$, $L$
$L_d$ = {}, $L_{th}$ = {} # distance, angle of each obstacle
$d_s$ = {1..n, $d_{max}$} # min distance to each sector
**while** $obs$ in $L$ **do**
  $d$ = distance($obs$, $x_r$); Add $d$ to $L_d$;
  $th$ = angle($obs$, $x_r$); Add $th$ to $L_{th}$;
  $s$ = angle2Sector($th$)
  **if** $d < d_s[s]$ **then**
    $d_s[s] = d$
  **end if**
**end while**
$PND$, $RND$ = computeDiagrams($d_s$)
$G$ = identifyGaps($PND$)
$s_g$ = angle2Sector(angle($x_g$, $x_r$))
$V$ = getValleys($G$, $s_g$)
$V$ = sortValleys($V$, $s_{gl}$)
**while** $u$ in $V$ **do**
  $u_g$ = computeGoal($x_g$, $u$)
  **if** isValleyNavigable($x_r, u_g, L, L_d, L_{th}$) **then**
    break
  **end if**
**end while**
$s_{ml}$, $s_{mr}$ = searchClosestObs($RND$, $u.s_{rd}$)
$v$, $w$ = computeVel($u$, $s_{ml}$, $s_{mr}$) # choose situation and compute v,w
**return** $v$, $w$

The Algorithm 1 first calculates the distance and the angle of each obstacle to the robot, at the same time it calculates the minimum distance to an obstacle in each sector, if there is no obstacle in a sector it takes the value $d_{max}$ (the maximum distance of the sensor). Then the diagrams, PND and RND, are computed and from them the discontinuities, gaps, are obtained. We then identify the valleys from the gaps and choose the best one. Finally, the velocities are calculated once the situation has been chosen.

A class called *Valley* has been created that stores the information related to a valley. The function *sortValleys* orders the valleys obtained by smallest sectorial distance to the goal, using the $s_{rd}$ of each one.

The function *isValleyNavigable* implements the proposed algorithm in [1] to check when a valley is navigable. The algorithm tells us whether the robot can reach the target in a straight line, taking into account the obstacles and the dimensions of the robot. This algorithm requires a target input ($u_g$), if the goal is within the region of $u$ then $s_g$ is used directly, otherwise the midpoint of the gap of $s_{rd}$ is used. The Algorithm 2 shows the proposed pseudocode implementation. This implementation returns $False$ if there is an obstacle whose distance to the target is smaller than the robot radius or if there are two obstacles in front of the robot(one on the left and one on the right) whose distance between them is smaller than the robot diameter.

**Algorithm 2** isValleyNavigable

**Require:** $x_r$, $v_g$, $L$, $L_d$, $L_{th}$
$th_g$ = angle($u_g$, $x_r$)
$d_g$ = distance($u_g$, $x_r$)
$FL$ = {}, $FR$ = {} # obstacles in front(left,right) of the robot
**while** $i$ in $L.size$ **do**
  **if** $distance(L[i], u_g) < robot\_radius$) **then**
    **return** $False$
  **end if**
  **if** $L_d[i] > d_g$ **then**
    continue
  **end if**
  $diff = diffAngles(L_{th}[i], th_g)$
  **if** $|diff| > \pi/2$ **then**
    continue
  **end if**
  $P = lineFrom(x_r, u_g)$
  **if** $distance(L[i], P) > 2 * robot\_radius$ **then**
    continue
  **end if**
  **if** $diff > 0$ **then**
    Add $i$ to $FR$
  **else**
    Add $i$ to $FL$
  **end if**
**end while**
**while** $(x_{FL}, x_{FR})$ in $allPairs(FL, FR)$ **do**
  **if** $distance(x_{FL}, x_{FR}) <= 2 * robot\_radius$ **then**
    **return** $False$
  **end if**
**end while**
**return** $True$

On the other hand, the $computeVel$ function is in charge of calculating the linear and angular velocity according to the situation in which the robot finds itself, as discussed in Section II-A.

The translational velocity direction has also been limited in the range $[-\pi/2, \pi/2]$, to prohibit instantaneous backwards motion. The following is how $s_\theta$ is calculated according to the situation:

1) **LS1**: Moves the robot away from the closest obstacle, and toward the gap (closest to the goal). The $s_\rho$ proposed in the paper increased as the sectorial distance from the nearest obstacle ($s_{ml}$) and the sector of the valley closest to the goal ($s_{rd}$) became larger. This has been changed so that the sector distance to move is greater the closer $s_{ml}$ is. This is achieved by knowing that the maximum distance between two sectors is $n/2$.

$$\begin{aligned} \rho &= 1 - \frac{|s_{rd} - s_{ml}|}{n/2} \\ s_\rho &= \frac{s_{max}}{4} * \rho + \frac{s_{max}}{2} \\ s_\theta &= s_{rd} + sign(s_{rd} - s_{ml}) * s_\rho \end{aligned} \tag{4}$$

2) **LS2**: Centers the robot between the two closest obstacles at both sides of the gap. The direction solution is computed as the bisector of the direction of the two closest obstacles plus a correction value $c$. This value depends on the distance of both obstacles to the robot and a constant value that is passed as a parameter ($D$). If the distance of the obstacles is similar, the correction value will be smaller.

$$c = D * (1 - \frac{min(d_l, d_r)}{d_l + d_r}) \qquad s_\theta = \frac{s_{ml} + s_{mr}}{2} \pm c \tag{5}$$

3) **HSGV**: The direction of motion is toward the goal location.

$$s_\theta = s_g \tag{6}$$

4) **HSWV**: The direction of motion is the addition of $s_{rd}$ and a dependent value of $s_{max}$.

$$s_\theta = s_{rd} + \frac{s_{max}}{2} \tag{7}$$

5) **HSNV**: The direction of motion is toward the central zone of the free walking area.

$$s_\theta = \frac{s_l + s_r}{2} \tag{8}$$

## IV. Experimental Results

The experiments have been carried out with a holonomic circular robot of radius $0.225m$. We set the maximum translational velocity to $0.5m/s$ and the maximum rotational velocity to $1.57rad/s$. The robot has a sensor with a FOV of 360º and a maximum range of $3m$. The safety distance has been set to $0.38m$. In addition, the number of sectors has been set to 144 and a valley is considered wide if its size is larger than 90º (36 sectors). Finally, the parameter used in LS2 has been set to 3 sectors. These parameters, with the exception of robot and sensor dimensions, can be varied to suit different scenarios. In addition, AMCL has been used as the robot's localization system.

The tests were carried out on the Stage simulator. The local planner has been tested in different scenarios, from an empty map at the beginning of the development to the final test on the map shown in figure 3. This map has different regions to test the different situations of the algorithm. The colours of each region correspond to:

- LS1: red.
- LS2: green.
- HSGR: yellow.
- HSNR: blue.

This scenario has both small and large spaces. It has regions with irregular structures, as can be seen in the first red region. This part is the most difficult for the robot, but it finally manages to pass it without problems. The map also has U-shaped structures, as can be seen for example on the left side. In this area the robot does not enter or get trapped in these structures because they are perfectly visible to the robot. This demonstrates the fact that, by having a high-level structure (regions) to reason and calculate motion commands, it is possible to avoid trap situations with the information available at the time.

In this map the average computation time of the algorithm is $8.15ms$, while the robot takes on average approximately $3min\ 24s$ to complete the entire path.

During this experiment it's been observed how other local planners, such as the DWA, had difficulty moving in this same scenario, being unable to reach the goal.

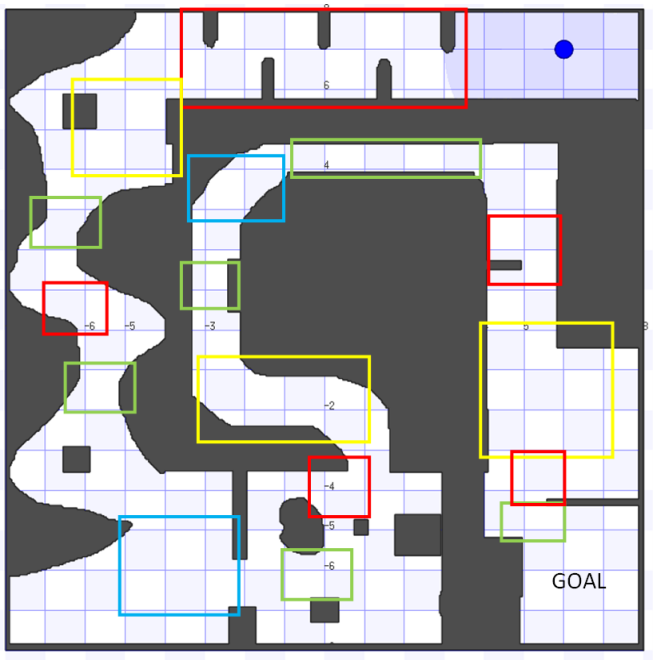


Fig. 3. Scenario used in the final test. Different situations marked in colors.

During the tests carried out throughout the development, we found that the $s_{th}$ calculation in LS1 proposed in the paper did not behave correctly, in some cases getting too close to the obstacles or not far enough away from them. Therefore, as discussed in Section III, it was decided to change the calculation.

## V. Conclusions

The ND local planner allows the robot to navigate unknown, highly complex and dynamic environments where other approaches have difficulty navigating safely and fluidly. To make this possible it reduces the complexity of navigation to a series of motion commands, extracted from a specific analysis of the environment.

One of the advantages of this method is that it allows to improve the development of navigation by adding more situations to the generation of movement commands, as occurs in the ND+, where a new low-security situation is added.

In this work ND has been developed for the case of a holonomic circular robot, but the planner can also be extended to work with robots of different shapes and constraints.

## REFERENCES


[1] J. Minguez and L. Montano, "Nearness diagram (nd) navigation: collision avoidance in troublesome scenarios," *IEEE Transactions on Robotics and Automation*, vol. 20, no. 1, pp. 45–59, 2004.

[2] ——, "Nearness diagram navigation (nd): a new real time collision avoidance approach," in *Proceedings. 2000 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2000) (Cat. No.00CH37113)*, vol. 3, 2000, pp. 2094–2100 vol.3.

[3] E. Montijano and L. Montano, "Reactive navigation, basic techniques: Nearness diagram (nd)," Course Autonomous Robots, Master in Robotics, Graphics and Computer Vision., 2021.