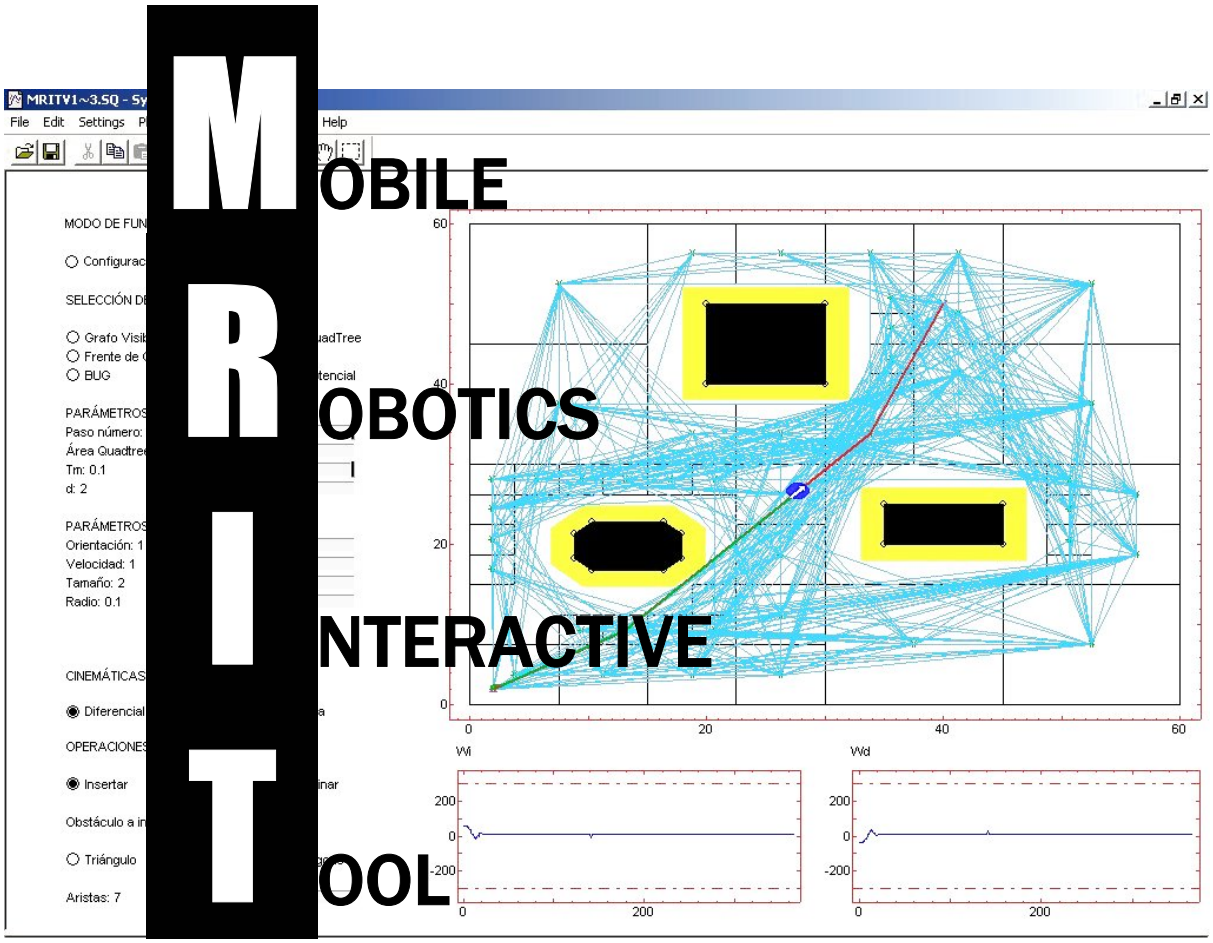


MOBILE ROBOTICS INTERACTIVE TOOL

V 1.4

HELP MANUAL

José Luis Guzmán
Oliver López
Manuel Berenguel
Francisco Rodríguez
Sebastián Dormido

Internal Report reference:

MRIT – Mobile Robotics Interactive Tool.

* Universidad de Almería, Dpto. de Lenguajes y Computación, Área de Ingeniería de Sistemas y Automática. Ctra. Sacramento s/n, 04120, Almería, España.
Tel. +34 950 015683, Fax. +34 950 015129,
E-mail: [joguzman, olopez, beren, frrodrig]@ual.es.

** Universidad Nacional de Educación a Distancia, Dpto. de Informática y Automática.
Avda. Senda del Rey nº 7, 28040, Madrid, España.
Tel. +34 91 3987151, Fax. +34 91 3986697,
E-mail: sdormido@dia.uned.es

Index

List of Figures

1. Introduction.

MRIT (*Mobile Robotics Interactive Tool*) is used to test and design and test mobile robotics navigation algorithms. This tool main aim are research and teaching, and is used as for settle theoretical concepts as for research new tecnichs and methods in mobile robotics. Since it is an interactive tool, it makes easy to use it and fast experimentation by user.

From teaching point of view, the pupil can put acquired knowledge in subjects into practice by easy simulated examples. The Tool helps pupil to understand the underlaying theory below planning and tracking algorithms, kinematics models and obstacles avoidance. Thanks to the interactivity of the tool, it's possible to observe how the simulation response is affected by the change of the value of the different parameters, such those of the own selected algorithm, the kinematics control signals, mobile robotics own parameters, etc.

This document is a manual of the tool and, in his second chapter presents one description about the different parts of the tool and it way of operation. Once the user is familiarized with the tool, in the third and last chapter, it's showed a set of examples presented with the objetive of make comprehensive and put into practice knowledge about mobile robotics to the user.

2. MRIT. Mobile Robotics Interactive Tool.

2.1. Tool description.

The Tool consists of one only screen in wich is showed all handled by user parameters just as graphics where results are shown. In figure 2.1 is showed the main screen, where can be distinguished two big areas. The graphic one, where results are shown, wich take up central and right screen areas. In this area is shown the work environment, which contains the mobile robot and the different obstacles. The other part, in the left area of the screen, contains the option menu with all parameters of the mobile robot, the obstacles, the algorithms and the kinematics.

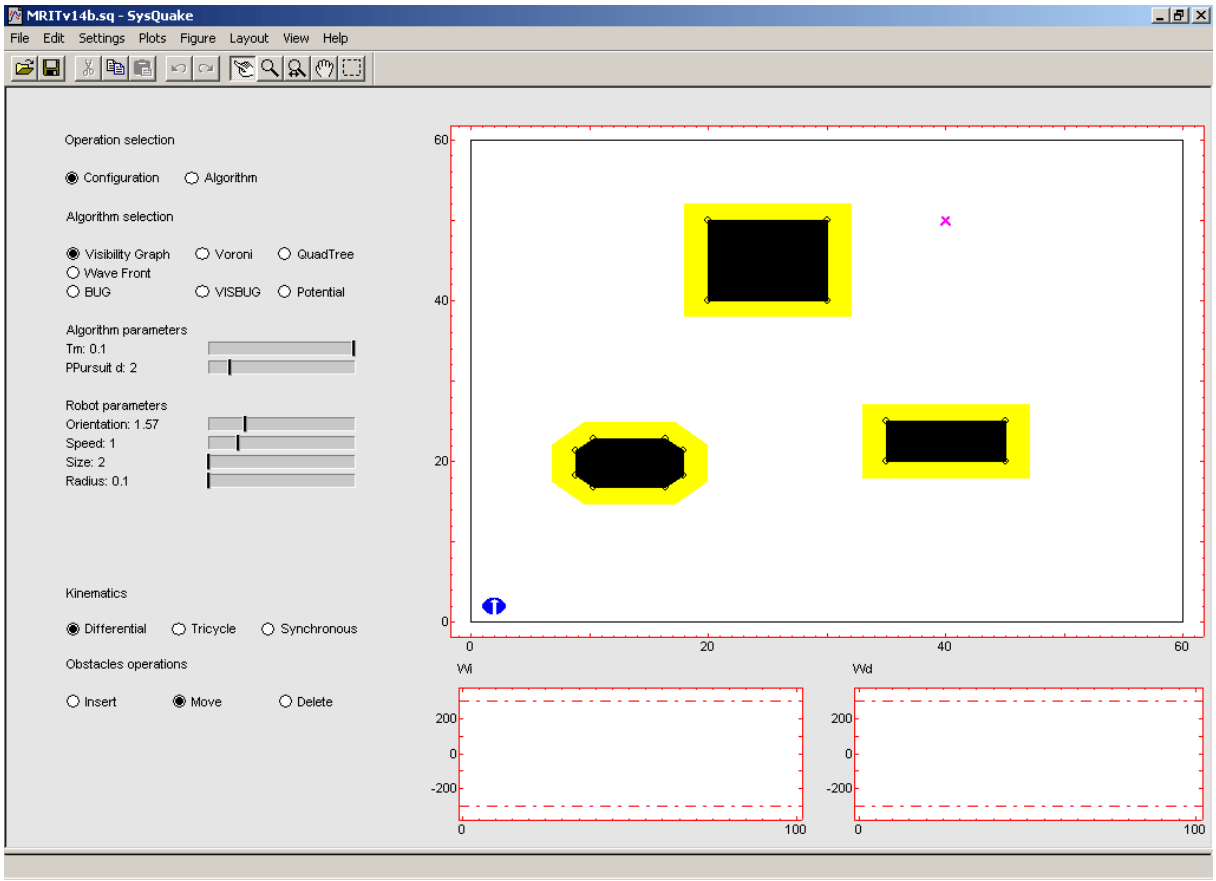


Figure 2.1. Interactive tool main screen.

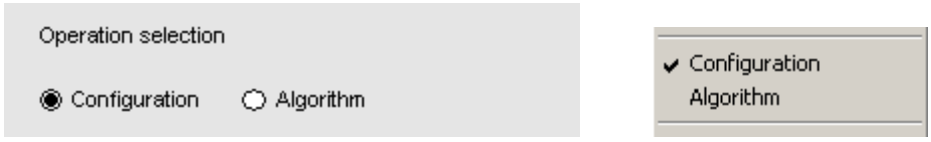
2.2. Options menu.

The options menu is located in the left side of the tool. Because of user interface consistency, options are duplicated, since are available at the same time through *Settings* menu, situated in the upper menu bar, and in the left side of the tool. This menu is divided into various parts:

- **Operation way**

  The tool has two different operation ways: *Configuretion* and *Algorithm*. The first way allows to select the algorithm, its parameters, robot parameters, its kinematics, and to handle work environment obstacles. Although the two operation ways allows same parameters handling, the main difference is that in *Algorithm* mode, the selected algorithm, with its selected parameters, is executed, while in *Configuretion* mode no calculation is made. So, if *algorithm* mode is selected and we modify some parameter, the change will be reflected inmediately into work environment.

  In figure 2.2 *Operation selection* menus can be observed.



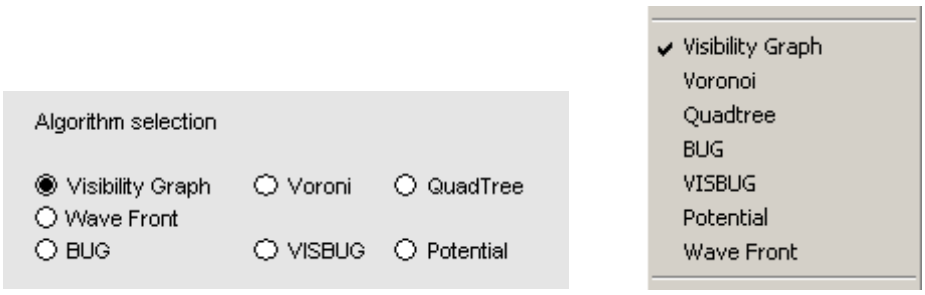
(a) Main menu. **(b)** *Settings* menu.

Figure 2.2. Operation way option menus.

- **Algorithm selection**

  Algorithms included in the tool are divided in *Global Algorithms* and *Local Algorithms*. First ones include: *Visibility Graph*, *Voronoi Diagram*, *Quadtree* and *Wave Front*. Local algorithms are: *BUG*, *VISBUG* and *Potential*.

  In figure 2.3 algorithm options menus are shown.



(a) Main menu. **(b)** *Settings* menu.

Figure 2.3. Algorithm options menus.

- **Algorithm parameters**

  Algorithms included in the tool have a set of characteristic parameters. Only specific parameters of the selected algorithm are showed every time.

  In figure 2.4 is showed one example of algorithm parameters options menu. The different parameters of each of existing algorithms will be described in detail further on.

(a) Main menu. **(b)** *Settings* menu.

Figure 2.4. Malgorithm parameters options menus.

- **Robot parameters**

  Mobile robot have also a set of characteristic parameters. In this case we can modify its orientation, its speed, its size and radius of its wheels.

  In figure 2.5 are shown mobile robot parameters options menus.

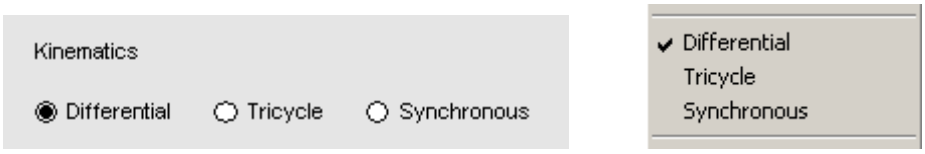
(a) Main menu. **(b)** *Settings* menu.

Figure 2.5. Mobile robot parameters options menus.

- **Kinematics**

  It is possible to select three different kinematics models for path tracking: *Differential*, *Tricycle* or *Ackerman* and *Synchronous*.

  In figure 2.6 Mobile robot kinematics options menus are shown.



(a) Main menu. **(b)** *Settings* menu.

Figure 2.6. Mobile robot kinematics options menus.

- **Obstacles operations**

  At defining or modifying work environment time, user can make one set of operations with its obstacles: *Insert*, *Move* and *Delete*.

  Figure 2.7 shows allowed obstacles operations options menus.

(a) Main menu. **(b)** *Settings* menu.

Figure 2.7. Obstacles operations options menus.

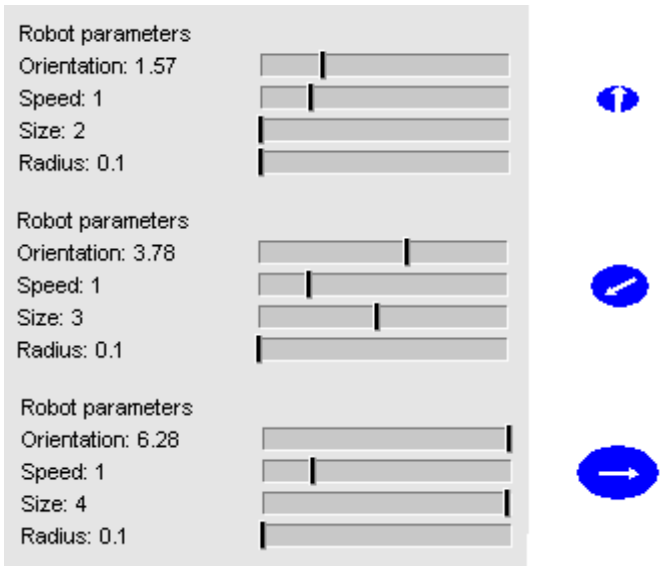
2.3. Work environment.

Work environment is represented by one graphic that take up great part of the toll main screen, on the central and right area. This two dimensional environment has a 6 square metres size and it showed with a bird's-eye view. The environment contains the mobile robot, the obstacles and the goal point.

2.3.1. Mobile robot.

The mobile robot is represented by a blue circle with one white arrow inside it. The blue circle diameter is specified by the maximum robot dimension and the arrow shows the current orientation. In figure 2.8 one robot size and orientation change example can be observed.

Figure 2.9 shows one how to change orientation through popup *Settings* menu example.



(a) Main menu. **(b)** Mobile robot.

Figure 2.8. Robot size and orientation change.

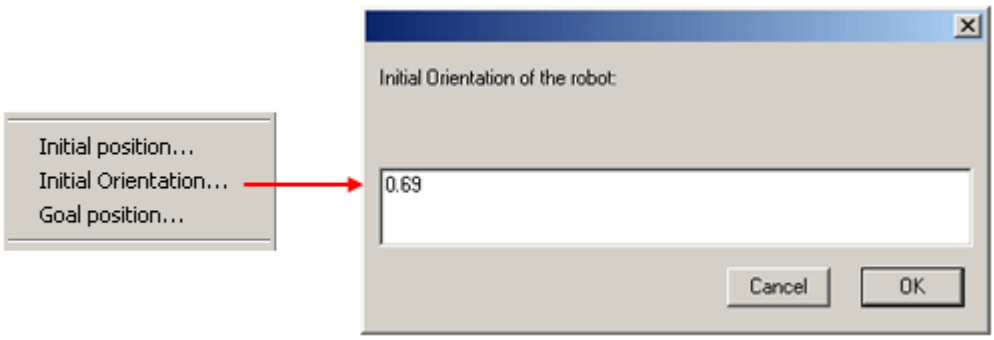
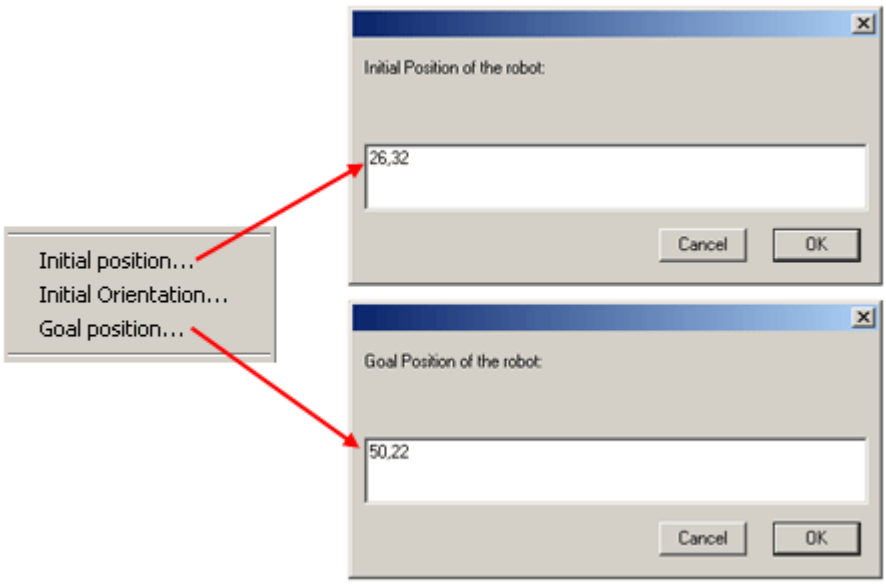


Figure 2.9. Initial orientation change example.

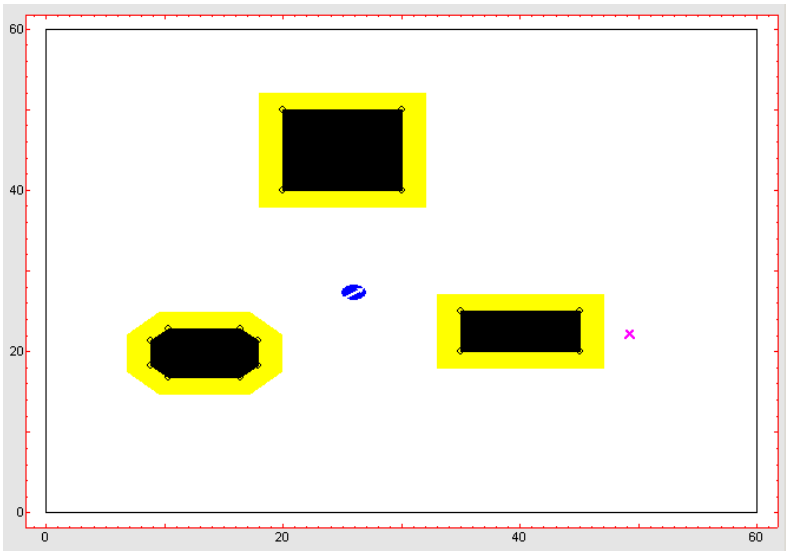
Initially, the robot's situation is the initial point, with a given orientation, and the goal is the destination point, represented by a *X*. Both points can be modified interactively by dragging them to another free position in the work environment, and into a more precisse way by popup *Settings* menu. Figure 2.10 (a) shows how mobile robot initial and goal points can be modified and figure 2.10 (b) shows the results of this change.

2.3.2. Obstacles.

Work environment obstacles are represented by regular polygons. Initially, there are two rectangles and one octogone into it, but there can be up to six obstacles simultaneously into work environment. When it's time to insert new obstacles, the tool counts on with different geometric shapes based on how many edges they have: triangles, rectangles and polygons. These lasts includes pentagons, hexagons, heptagons and octogons. In figure 2.11, object type to insert options menus can be observed.

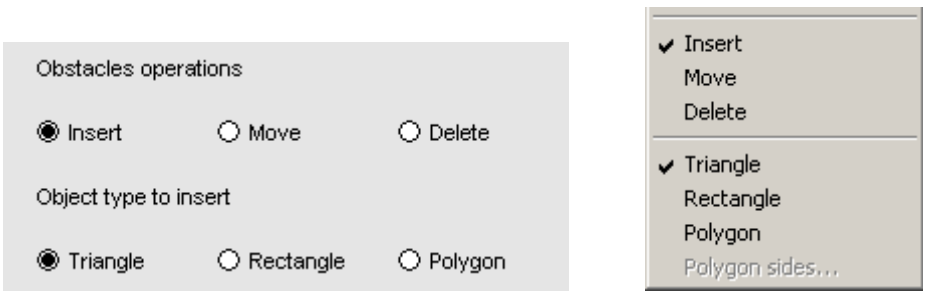


(a) *Settings* menu.



(b) Work environment.

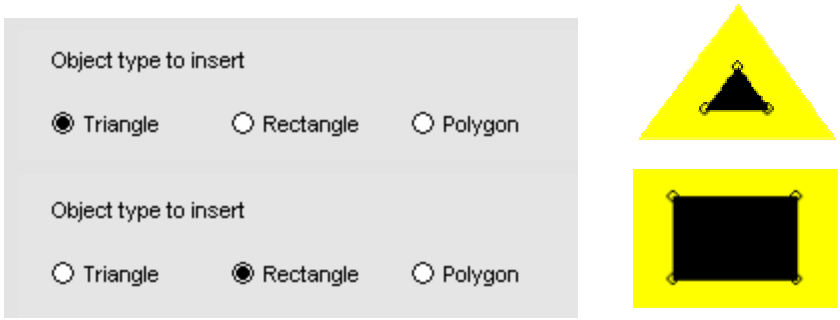
Figure 2.10. Mobile robot initial and goal positions change example.



(a) Main menu. **(b)** *Settings* menu.

Figure 2.11. Object type to insert options menus.

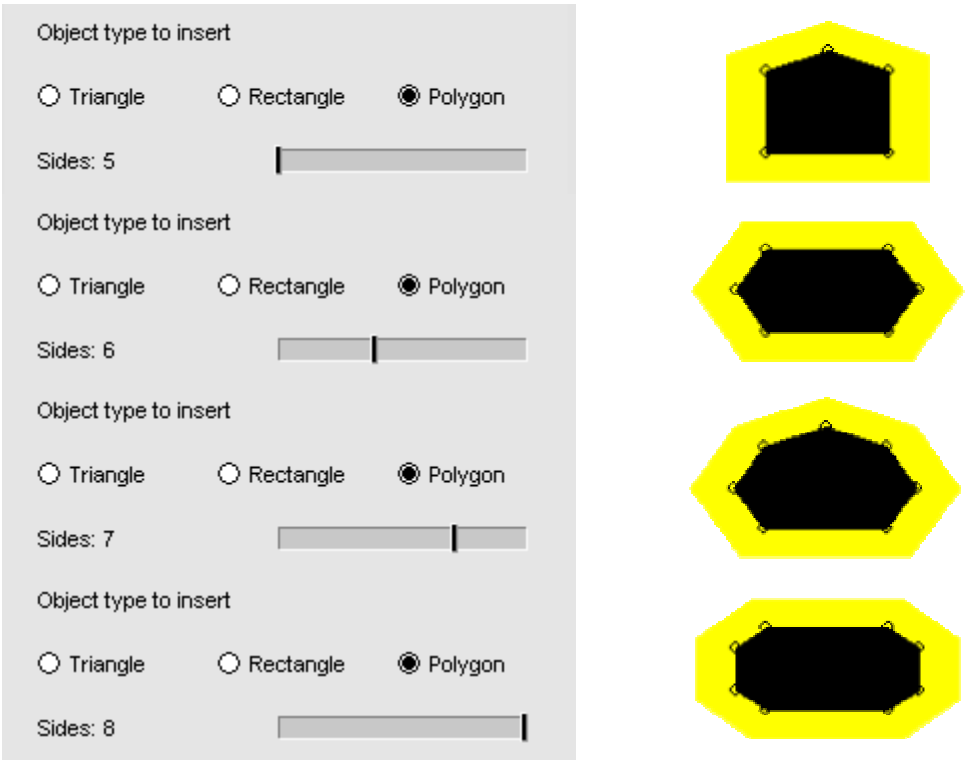
In figure 2.12 a triangle and a rectangle obstacles examples can be observed.



(a) Main menu. **(b)** Obstacles.

Figure 2.12. Triangle and rectangle obstacles selection examples.

Figure 2.13 shows obstacle type to insert options menus and the result of selecting each one of the availaible polygons.



(a) Main menu. **(b)** Obstacles.

Figure 2.13. All different polygonal obstacles type selection example.

Figure 2.14 shows the way to insert one polygonal obstacle into work environment by using popup menu *Settings*.

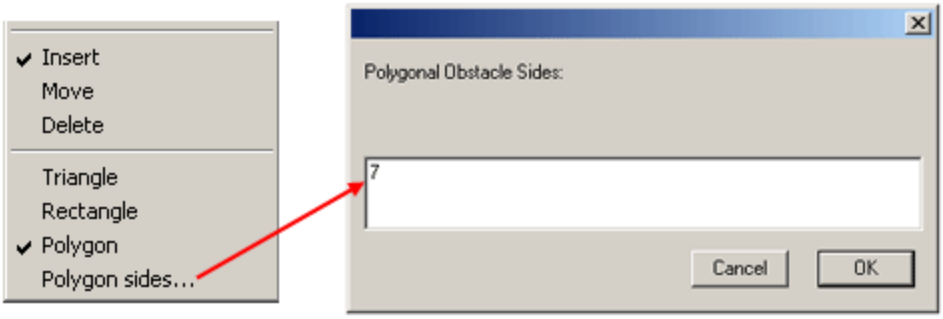


Figure 2.14. Polygonal obstacles type to insert selection example.

When inserting an obstacle into work environment, is located around the point specified by the mouse pointer. Obstacles size is pre-established proportionally according to environment and robot sizes.

Figure 2.15 shows one work environment with one obstacle of each type into it.

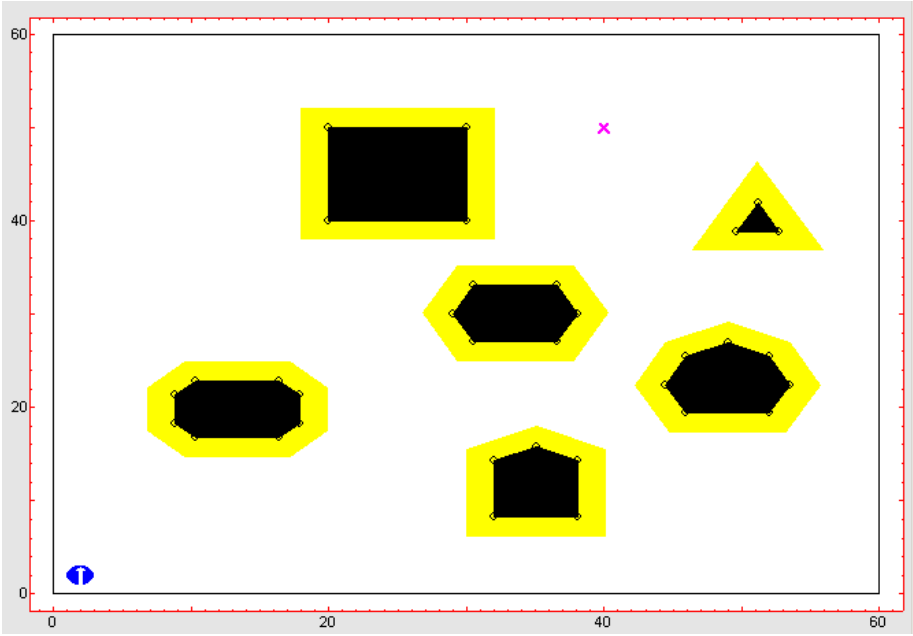


Figure 2.15. Work environment with one obstacle of each type example.

Work environment obstacles can be modified by options menus *Move* option. Obstacles vertexs are interactive, and its position can be modified, taking place to irregular polygons. Figure 2.16 shows obstacles *Move* options menus. Figure 2.7 shows obstacles change example.

(a) Main menu. **(b)** *Settings* menu.

Figure 2.16. Work environment obstacles *move* options menus.

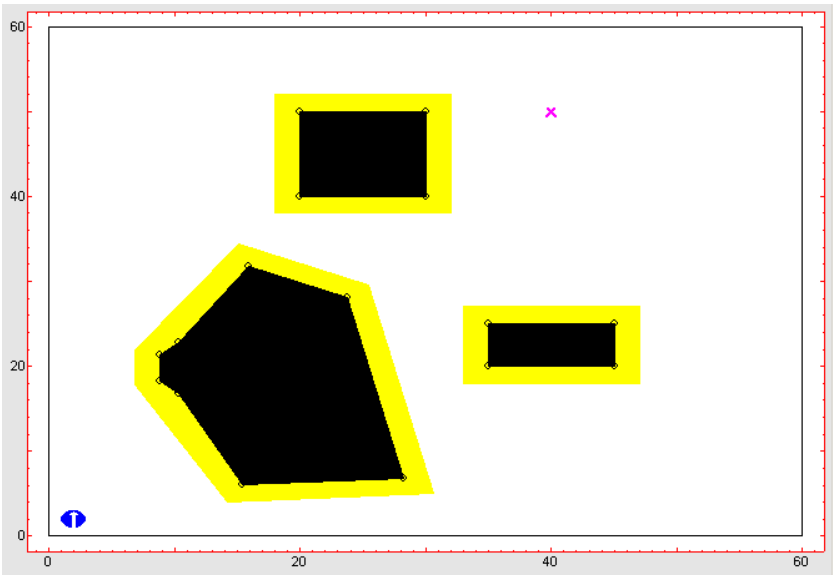


Figure 2.17. Work environment obstacle change example.

If one obstacle intersects with one or more existing obstacles when is modified, all these obstacles merge in to one single obstacle (it depends of what algorithm is selected). Figure 2.18 shows one example of merging obstacles.

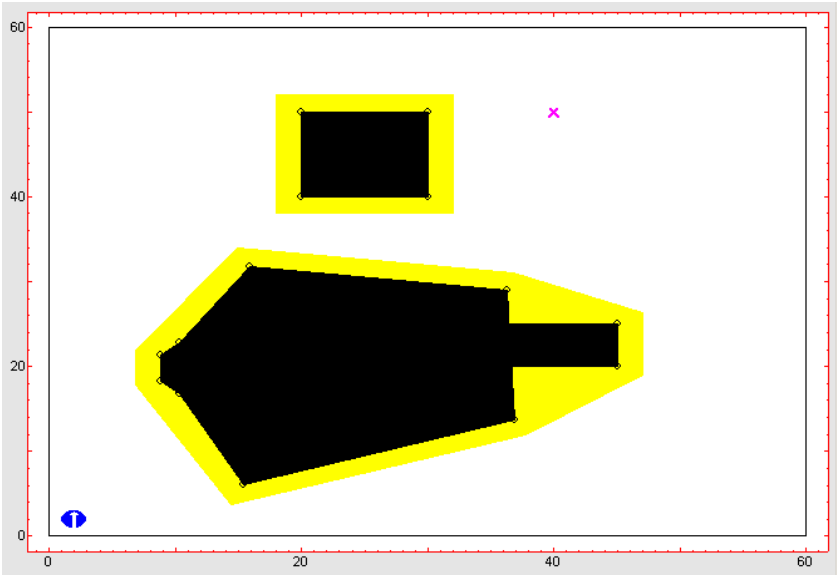


Figure 2.18. Work environment obstacles merge example.

Work environment obstacles can be entire deleted by *Delete* option. Selecting obstacle to erase with mouse left button, it is deleted from work environment at once. Figure 2.19 shows options menus to delete an obstacle. Figure 2.20 shows the results of deleting one obstacle of the same work environment in figure 2.17.

(a) Main menu. **(b)** *Settings* menu.

Figure 2.19. Work environment obstacles *delete* options menus.

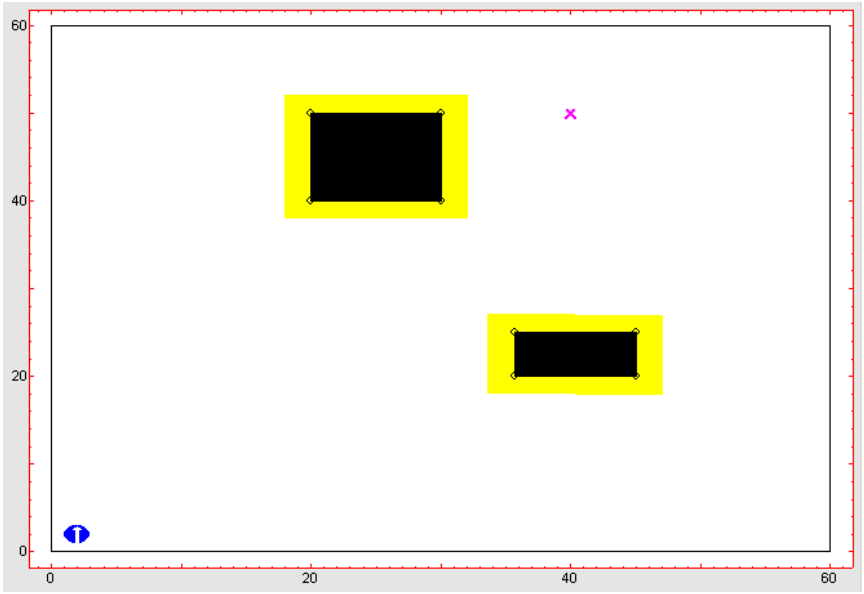


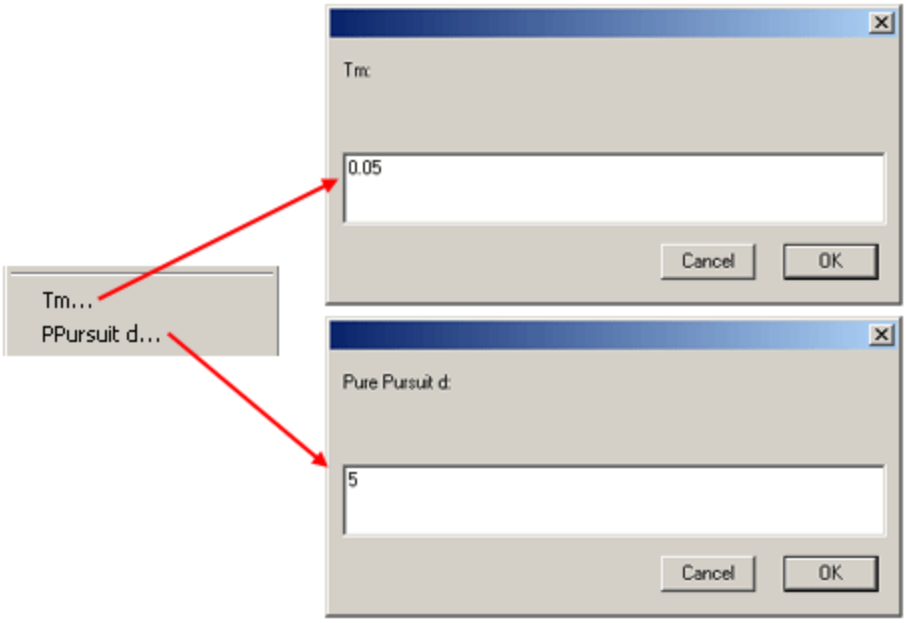
Figure 2.20. Work environment obstacle delete example.

2.4. Algorithms.

2.4.1. Global Algorithms.

Global algorithms first carry out path planning and then it's tracking. All tool global algorithms use *Pure Pursuit* algorithm to path tracking. So, all of them have two characteristic parameters relative to pure pursuit among its own ones: sampling time (*Tm*) and objective point distance (*d*). Figure 2.21 shows options menus of these two parameters and a selection example.

(a) Main menu.



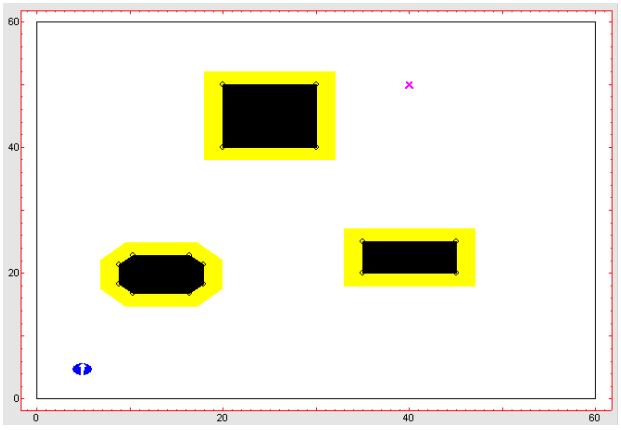
(b) *Settings* menu.

Figure 2.21. *Pure Pursuit* parameters options menus examples.

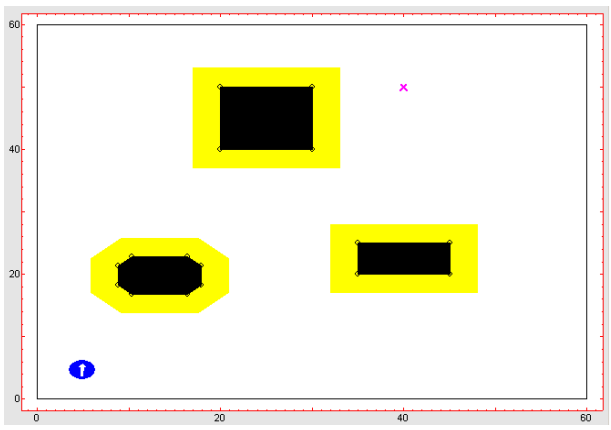
Another characteristic that global algorithms have jointly is that they perform an enlargement of the obstacles perimeter, corresponding to mobile robot maximum dimension, represented by *Size* parameter. This obstacle's wrapper is aimed at make work environment more safe for the mobile robot, so, the bigger the robot, the bigger the enveloped obstacles.

Figure 2.22 shows an example of the effect of the enlargement of the mobile robot on the work environment obstacles.

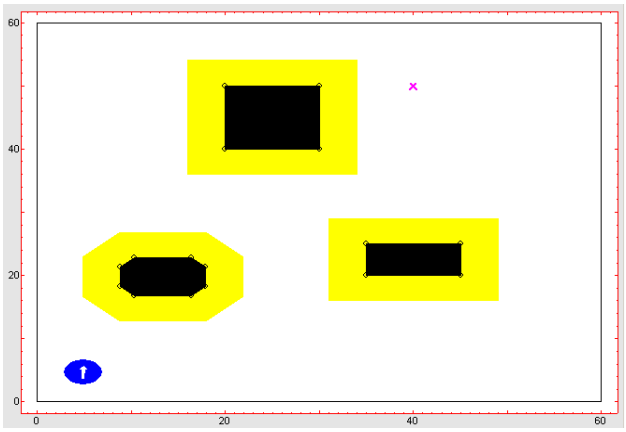
Next, characteristic parameters of some of the global algorithms are described.



(a) Robot size 2.



(b) Robot size 3.



(c) Robot size 4.

Figure 2.22. Mobile robot enlargement effect on work environment example.

- **Quadtree**

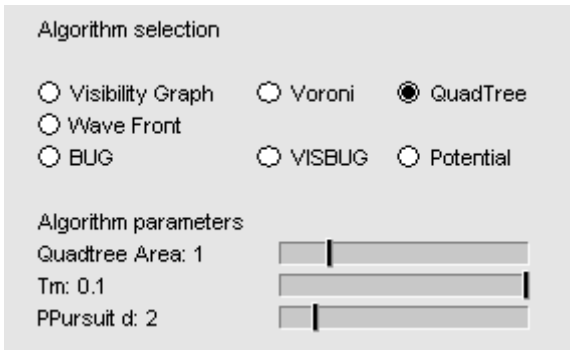
  Its characteristic parameter is the *quadtree area*. Figure 2.23 shows options menus for this parameter.

  Figure 2.24 shows the effect of changing the size of the quadtree area on the algorithm.

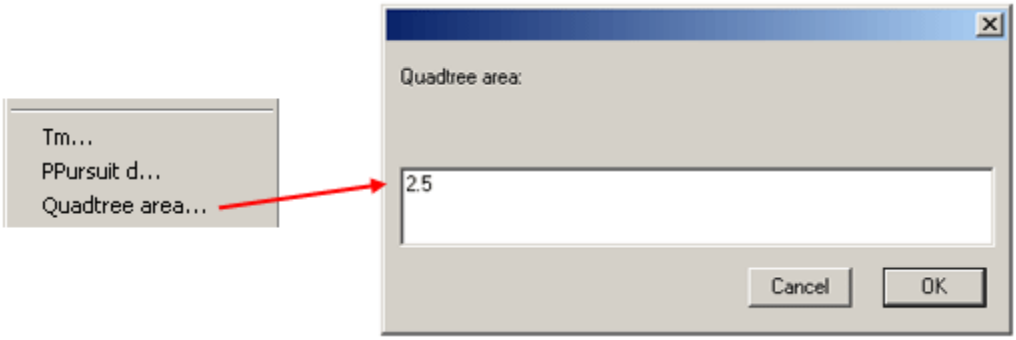
- **Wave front**

  Its characteristic parameters are the size of the cells that divide work environment into a grid and the neighborhood between cells. Figure 2.25 shows the options menus for the two parameters.

  Figure 2.26 shows the effect of changing cell size join with neighborhood between cells.

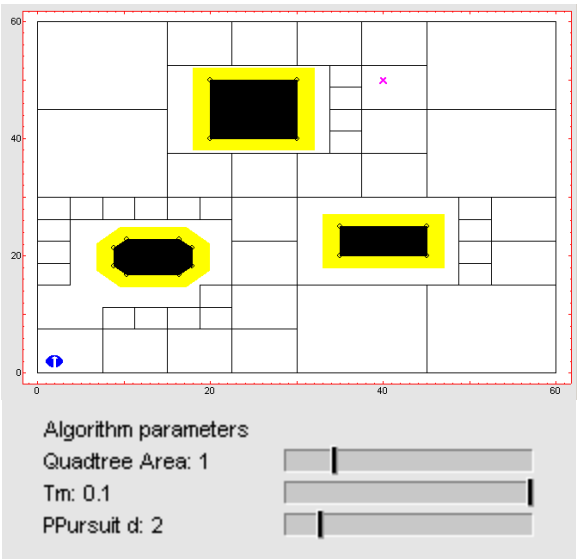


(a) Main menu.

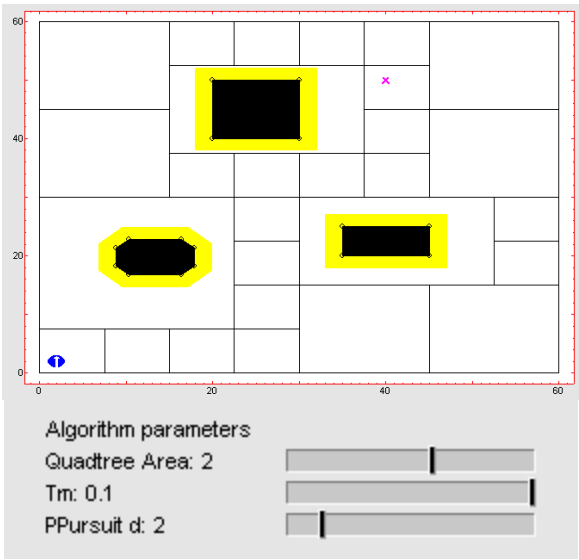


(b) *Settings* menu.

Figure 2.23. *Quadtree area* options menus example.

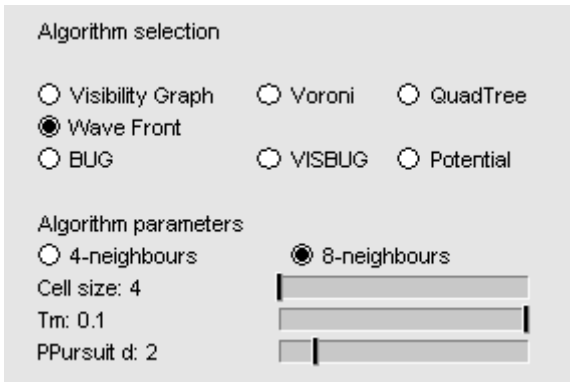


(a) *Quadtree area* 1.

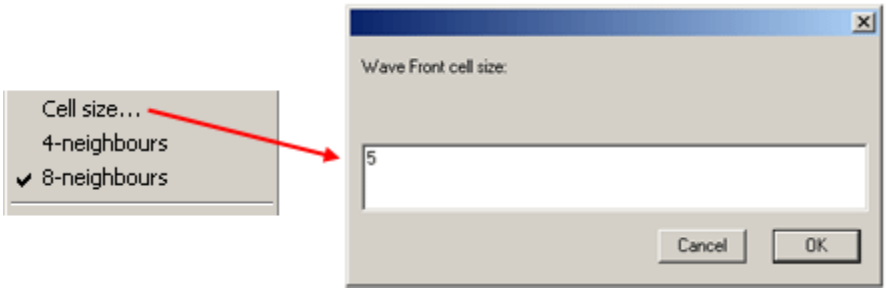


(b) *Quadtree area* 2.

Figure 2.24. *Quadtree area* change effect examples.

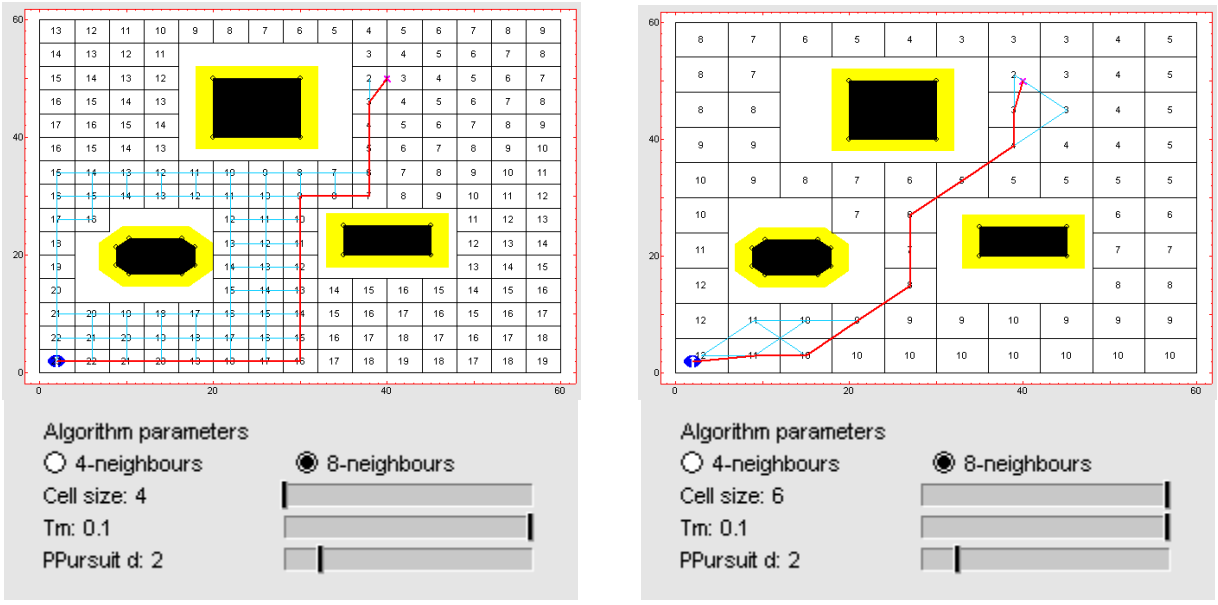


(a) Main menu.



(b) *Settings* menu.

Figure 2.25. *Cell size* options menus example.



(a) *Cell size* 4, *neighborhood* 4.

(b) *Cell size* 6, *neighborhood* 8.

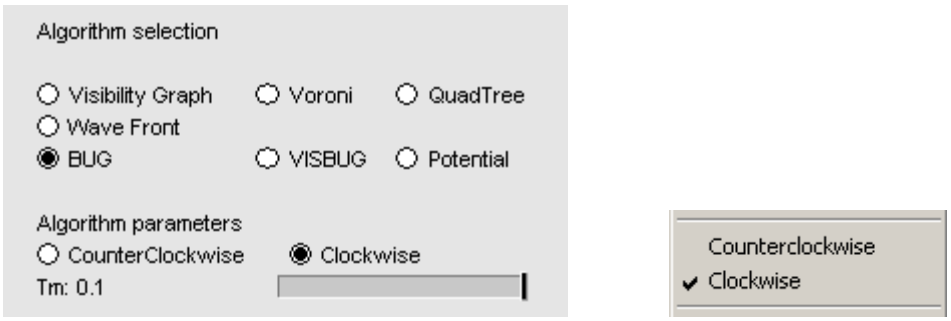
Figure 2.26. *Cell size* and *neighborhood* change effect examples.

2.4.2. Reactive Algorithms.

Reactive algorithms perform path planning and tracking jointly, that is, they reacts environment as they navigate through it. That's the reason why they don't need any algorithm else to perform path tracking.

- **BUG**

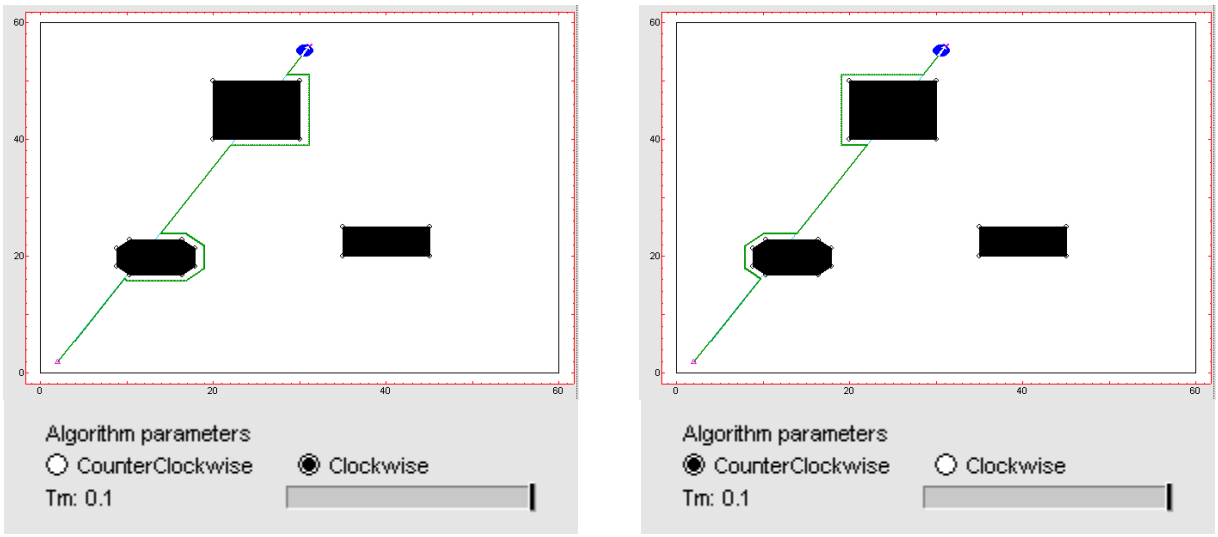
  BUG algorithm only has one characteristic parameter, the pre-established turn direction to avoiding obstacles into work environment when robot finds one of them. Figure 2.27 shows options menus for this parameter.



(a) Main menu. **(b)** *Settings* menu.

Figure 2.27. *BUG* algorithm *turn direction* options menus.

Figure 2.28 shows the effect of changing turn direction when avoiding one obstacle.



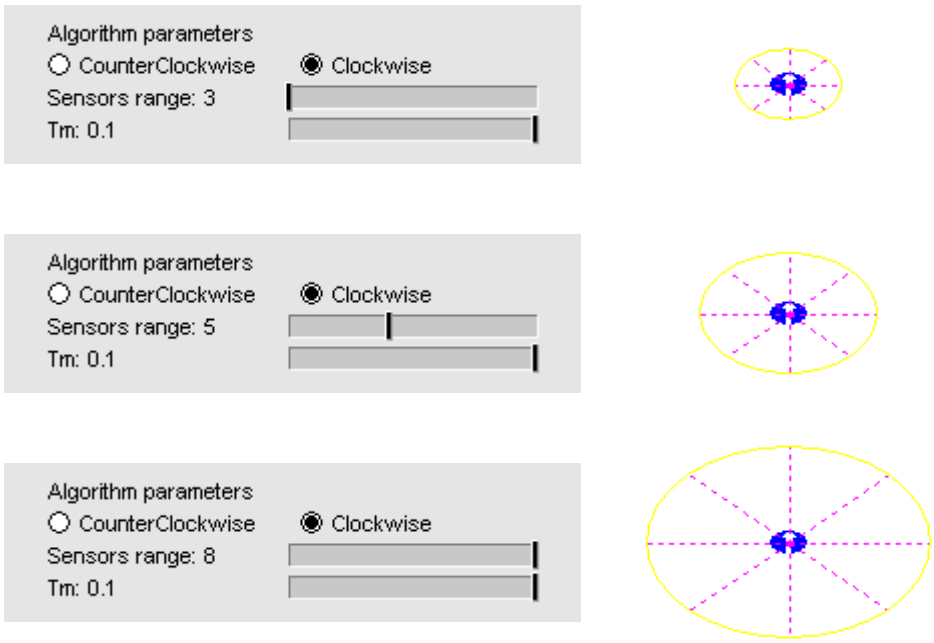
(a) Clockwise *turn direction*. **(b)** Counterclockwise *turn direction*.

Figure 2.28. *BUG* algorithm *turn direction* change effect examples.

- **VISBUG**

  VISBUG algorithm has two characteristic parameters, the pre-established turn direction to avoiding obstacles into work environment and the distance sensors range (sonar ones). Mobile robot shows orientation and range of its different sensors into work environment. Figure 2.29 shows robot appearance change when its sensors range is modified.

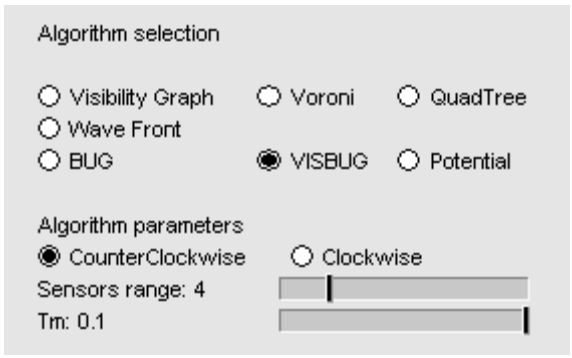
  Figure 2.30 shows turn direction and sensors range options menus.



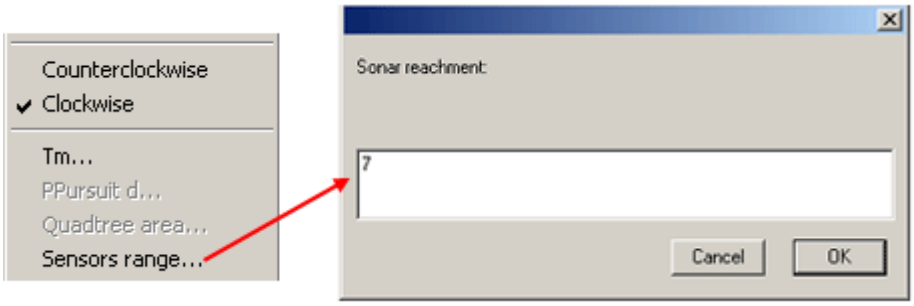
(a) Main menu. **(b)** Mobile robot.

Figure 2.29. *Sensors range* different values examples.

Figure 2.31 shows the effect of changing robot sensors range jointly with turn direction.

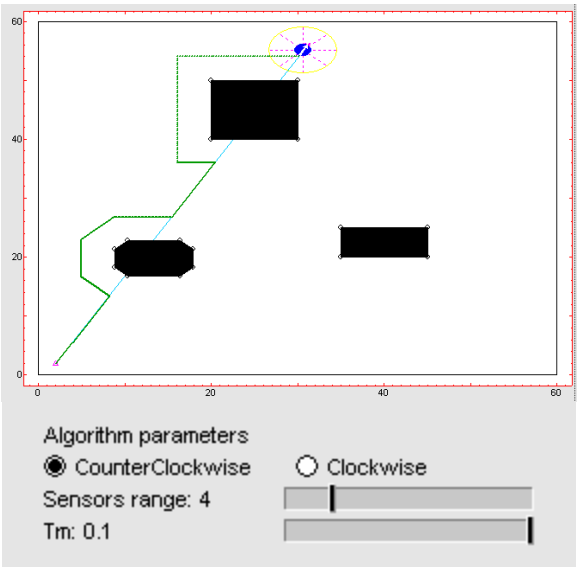


(a) Main menu.

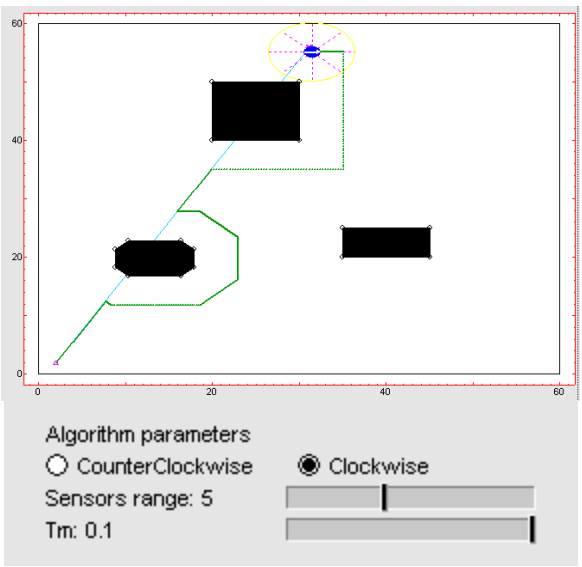


(b) *Settings* menu.

Figure 2.30. *Sensors range* options menus examples.



(a) Counterclokwise *turn direction*, *sensors range* 4.

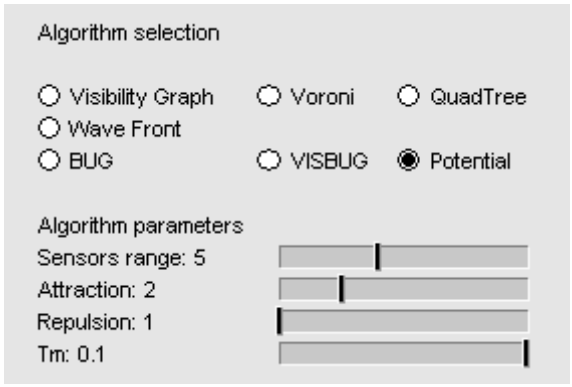


(b) Clockwise *turn direction*, *sensors range* 5.

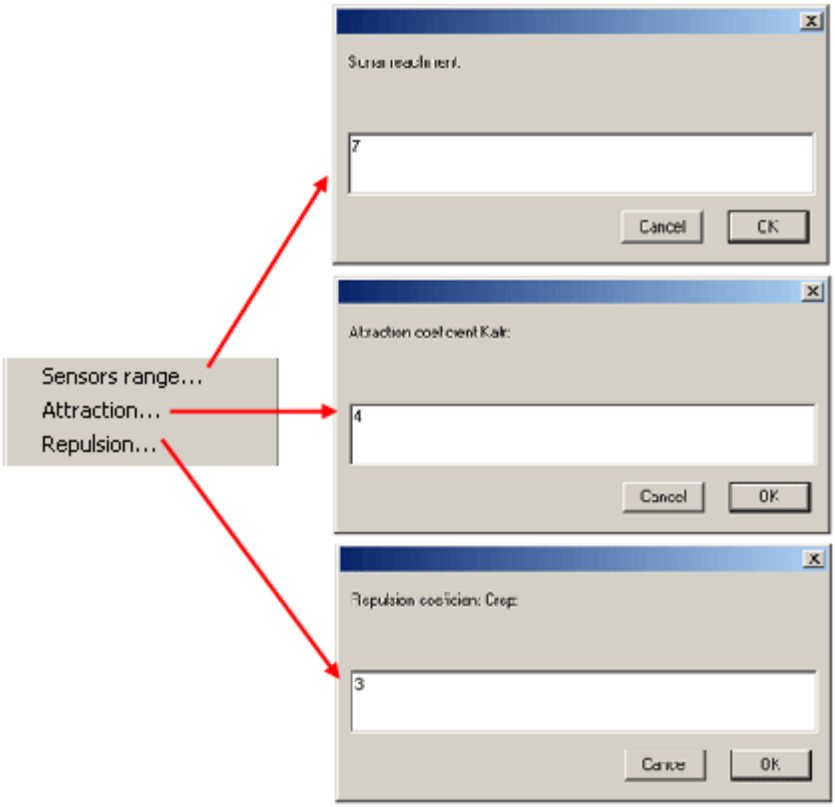
Figure 2.31. *Turn direction* and *sensors range* change effect examples.

- **Potential**

  Potential algorihtm has three characteristic parameters: mobile robot distance sensors range (sonar ones), goal point attraction constant (*Katr*) and obstacles repulsion constant (*Crep*). Figure 2.32 shows all potential algorithm characteristic parameters options menus.



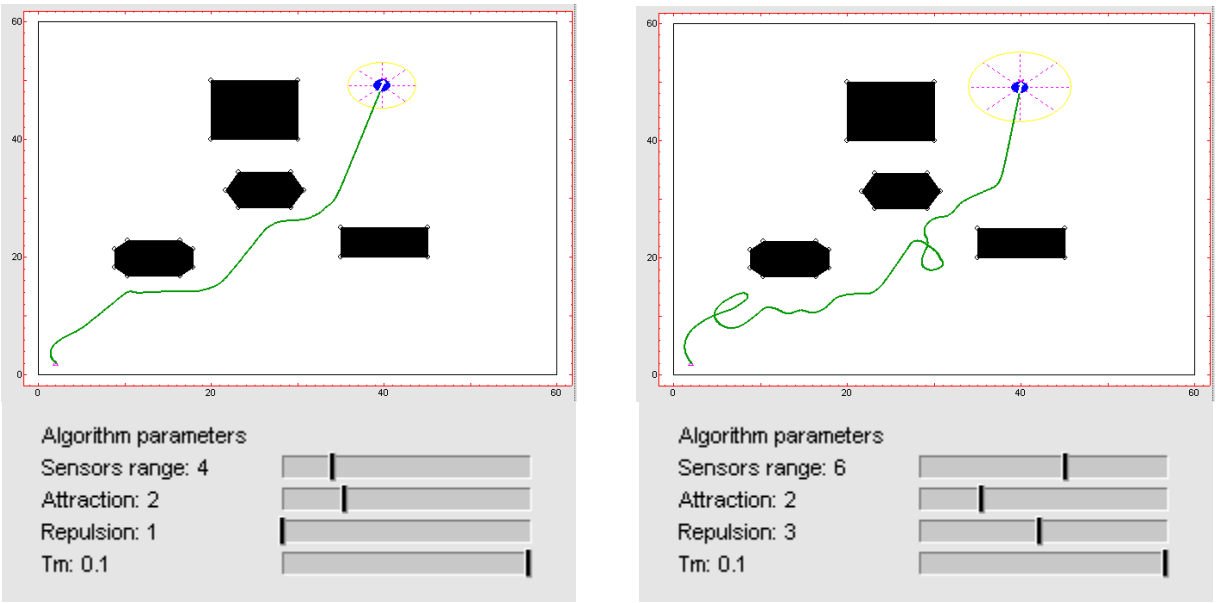
(a) Main menu.



(b) *Settings* menu.

Figure 2.32. *Sensors range*, *attraction* and *repulsion* options menus example.

Figure 2.33 shows the effect of changing jointly *sensors range*, *attraction* and *repulsion*.

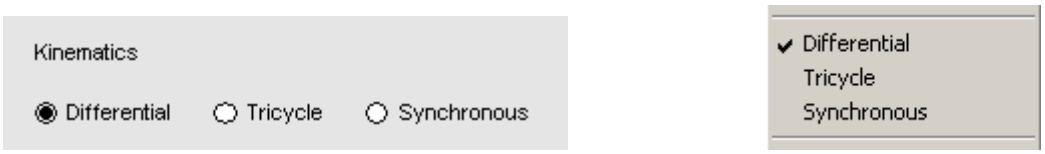


(a) Alcance del sónar 4, atracción 2 y repulsión 1. **(b)** Alcance del sónar 6, atracción 2 y repulsión 3.

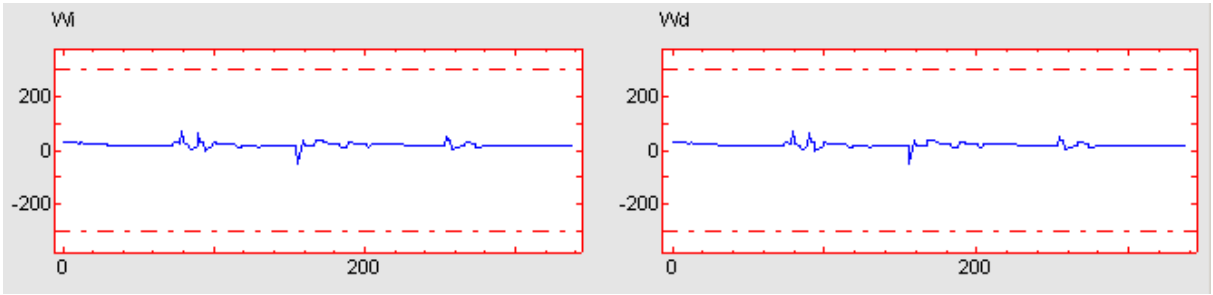
Figure 2.33. *Sensors range*, *attraction* and *repulsion* change effect examples.

2.5. Kinematics.

Tool includes three different kinematics models: *Differential*, *Tricycle* and *Synchronous*. Each kinematics has different characteristic parameters or control signals: differential kinematics has robot wheels angular velocities *Wi* y *Wd*, tricycle kinematics has robot free wheel angle, *alpha*, and synchronous kinematics has all robot wheels angle, *theta*.



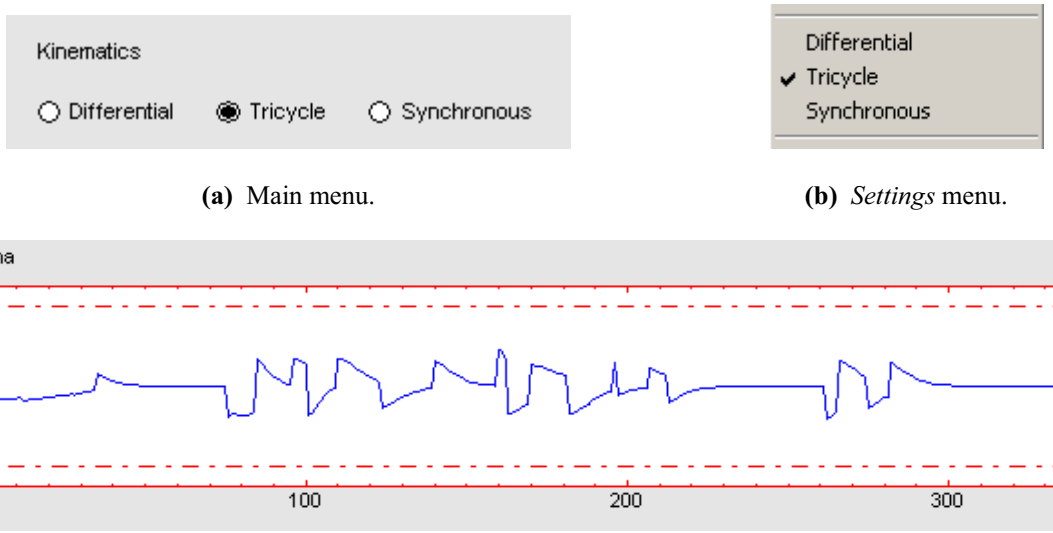
(a) Main menu. **(b)** *Settings* menu.



(c) Differential kinematics control signals ωi and ωd.

Figure 2.34. *Differential* kinematics options menus and effect into user interface.

Graphics corresponding to each kinematics control signals values are located on right-down side of the main screen of the tool, just below the work environment. Figures 2.34, 2.35 and 2.36 show options menus for all kinematics and their control signals graphics.

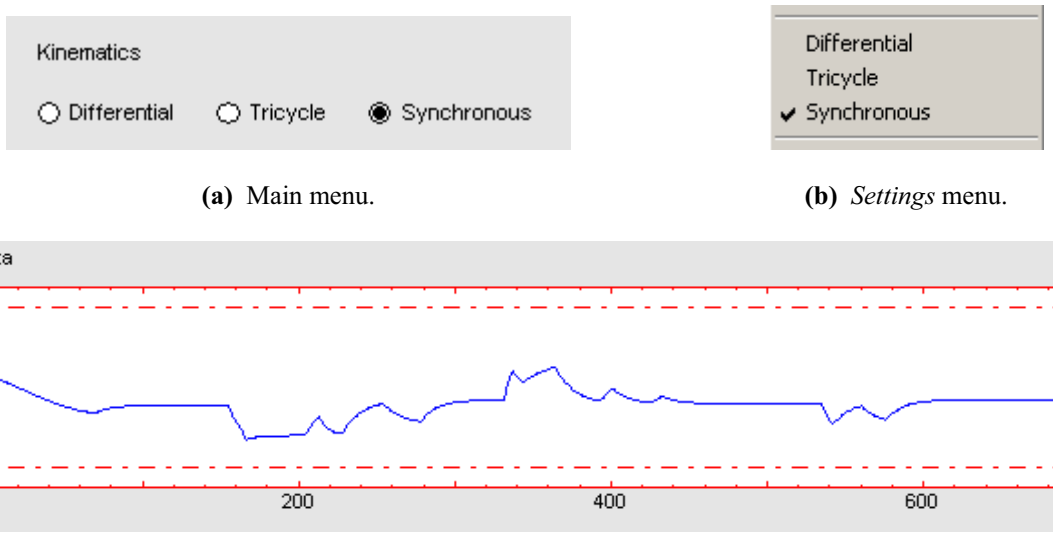


(a) Main menu.

(b) *Settings* menu.

(c) *Tricycle* kinematics control signal α.

Figure 2.35. *Tricycle* kinematics options menus and effect into user interface.



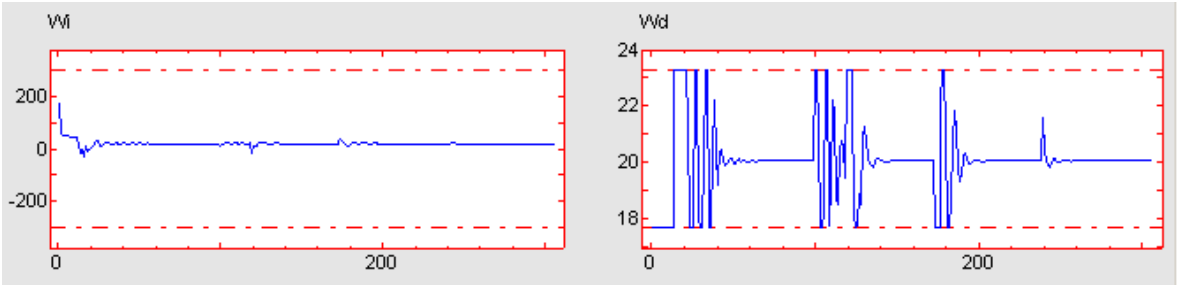
(a) Main menu.

(b) *Settings* menu.

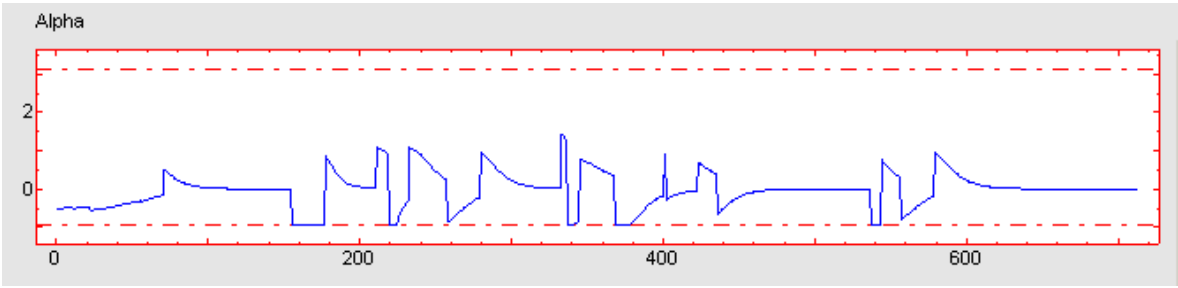
(c) *Synchronous* kinematics control signal θ.

Figure 2.36. *Synchronous* kinematics options menus and effect into user interface.

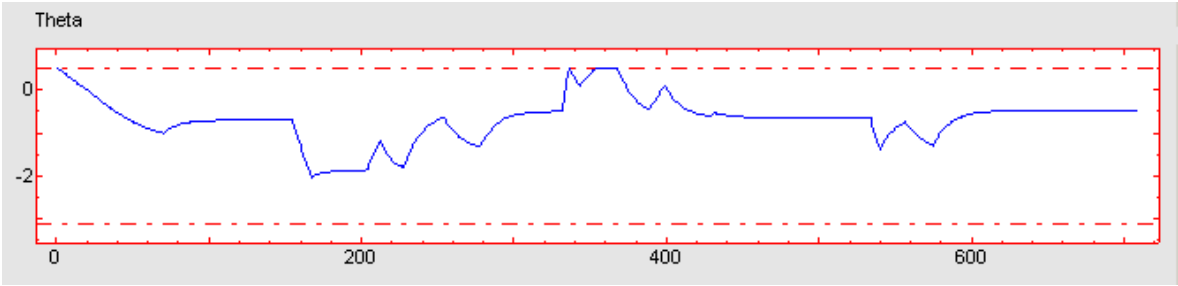
Graphics representing kinematics control signals counts on with an upper and a lower limit. User can set this limits wherever he wants, affecting robot path tracking. The effect of adding constraints to the different kinematics is shown in figure 2.37.



(a) *Differential* kinematics constraints.



(b) *Tricycle* kinematics constraints.



(c) *Synchronous* kinematics constraints.

Figure 2.37. Different kinematics constraints application examples.

2.6. Simulation.

In order to perform simulation with the interactive tool, *Algorithm* mode has to be selected, and next, through *Settings* menu, select *Run* or *Step by Step*.

Run simulation makes robot path tracking over path planning, whereas *Step by Step* simulation shows all steps taken during planning. In this way, *Step by Step* simulation needs *Run* to be selected in order to make robot track path obtained through this steps. *Step by Step* simulation it's only available for global algorithms, which are the only ones who separately make path planning and tracking.

To stop simulation and go back to *Algorithm* mode, just necessary to select again the type of simulation selected through *Settings* menu. Figure 2.38 shows simulation types options menus.

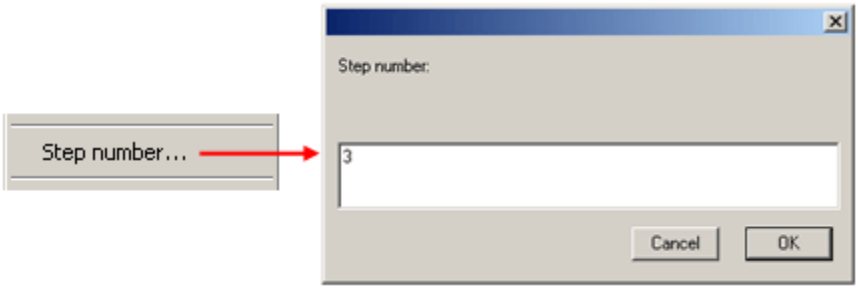
(a) *Run* execution. **(b)** *Step by Step* execution.

Figure 2.38. Simulation mode options menus.

2.6.1 Step by Step simulation.

Simulation steps are shown with each algorithm characteristic parameters, and the desired step can be selected at once. Figure 2.39 shows simulation steps options menus.

(a) Main menu.



(b) *Settings* menu.

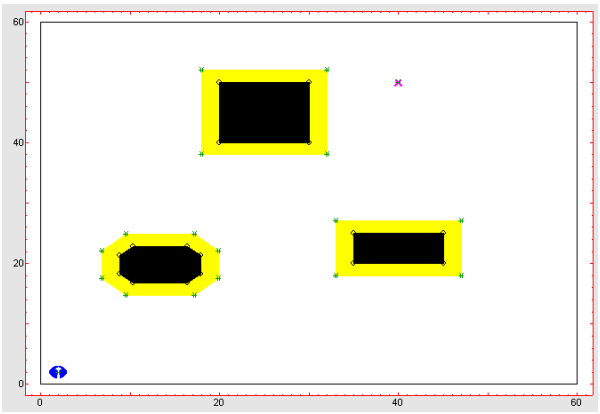
Figure 2.39. *Step number* options menus example.

- **Visibility Graph.**

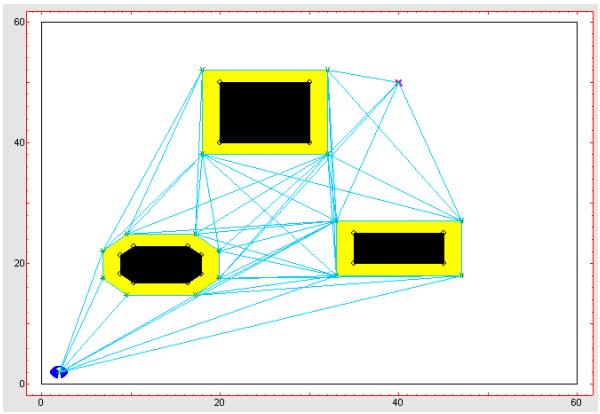
  Visibility Graph algorithm has three simulation steps:

  1) Visibility matrix points.
  2) All possible paths.
  3) Minimal path.

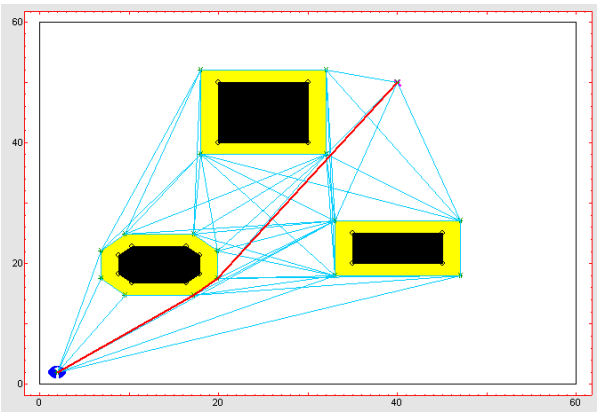
  Figure 2.40 shows all *Visibility Graph* algorithm simulation steps.



(a) Step 1. Clear vertex.



(b) Step 2. All possible paths.



(c) Step 3. Minimal path.

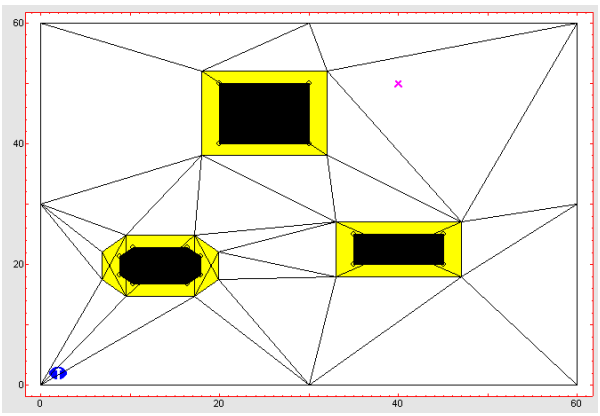
Figure 2.40. *Visibility Graph* simulation steps.

- **Voronoi Diagram.**

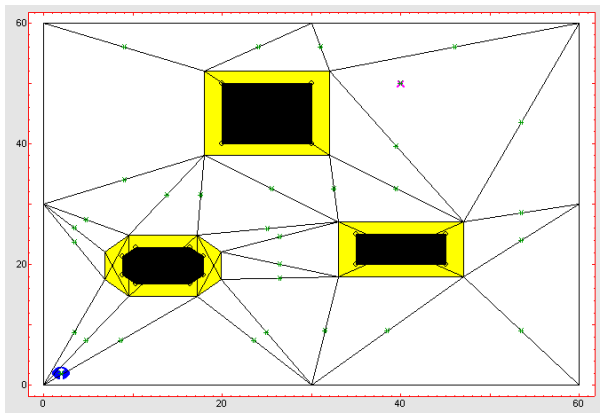
  Voronoi Diagram algorithm has four simulation steps:

  1) Delaunay triangulations.
  2) Triangulations middle grounds.
  3) All possible paths.
  4) Minimal path.

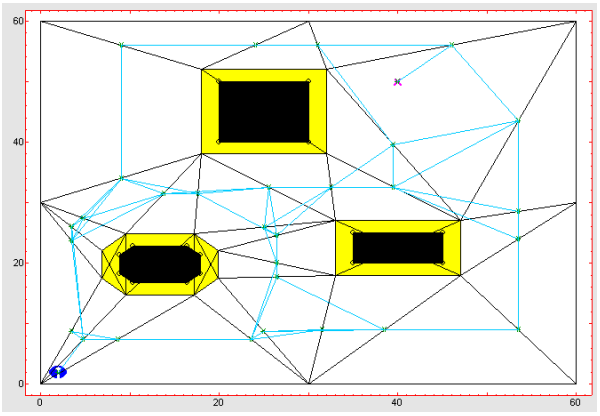
  Figure 2.41 shows all *Voronoi Diagram* algorithm simulation steps.



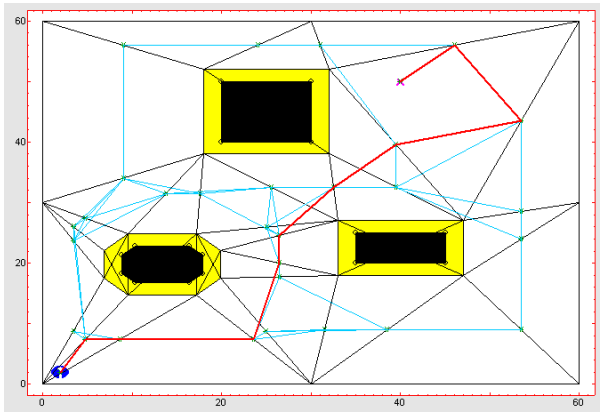
(a) Step 1. Delaunay triangulations.



(b) Step 2. Triangulations middle grounds.



(c) Step 3. All possible paths.



(d) Step 4. Minimal path.

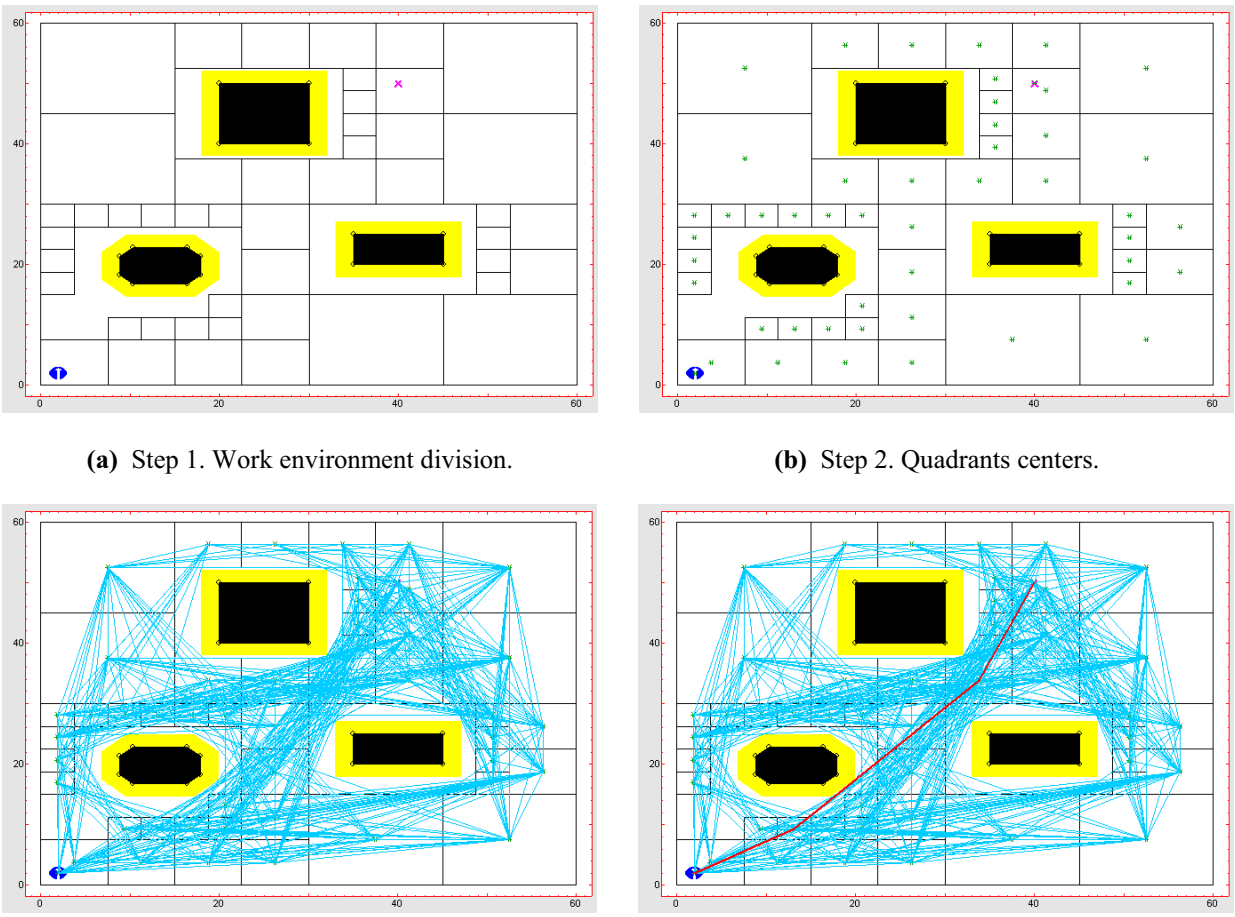
Figure 2.41. *Voronoi Diagram* simulation steps.

- **Quadtree.**

  Quadtree algorithm has four simulation steps:

  1) Work environment quadrants divison.
  2) Quadrants central points.
  3) All possible paths.
  4) Minimal path.

  Figure 2.42 shows all *Quadtree* algorithm simulation steps.



(a) Step 1. Work environment division.

(b) Step 2. Quadrants centers.

(c) Step 3. All possible paths.

(d) Step 4. Minimal path.

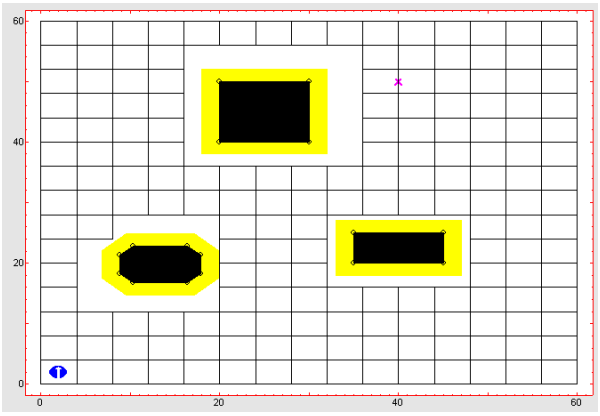
Figure 2.42. *Quadtree* simulation steps.

- **Wave Front.**

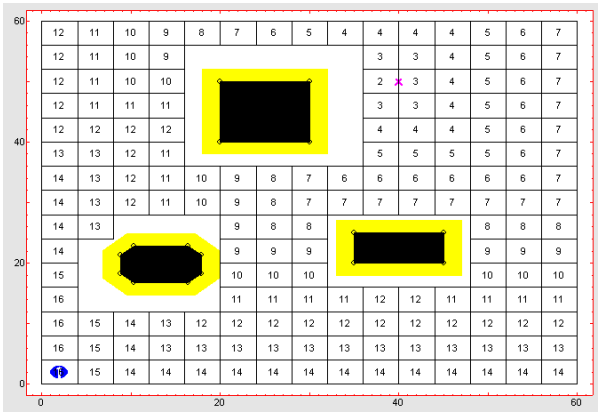
  Wave Front algorithm has four simulation steps:

  1) Work environment cells division.
  2) Cells logical values.
  3) All possible paths.
  4) Minimal path.

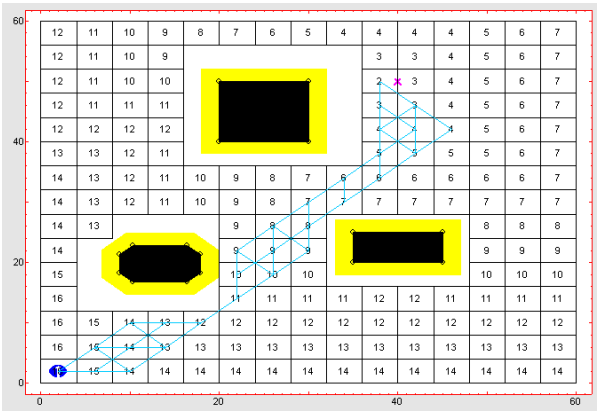
  Figure 2.43 shows all *Wave Front* algorithm simulation steps.



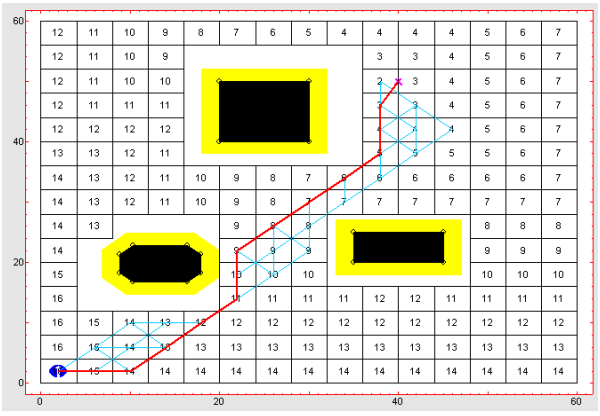
(a) Step 1. Work environment division.



(b) Step 2. Cells values.



(c) Step 3. All possible paths.



(d) Step 4. Minimal path.

Figure 2.43. *Wave Front* simulation steps.

2.6.2. *Run* mode simulation.

This simulation mode will be shown in examples section through different practical assumptions.

2.6.3. Error messages.

Next are summarized all different error messages tool can show and its meaning:

- **Execution error.** It happens when *Configuration* mode is selected and you try to select *Run* mode simulation (figure 2.44).

- **Obstacles excess error.** It happens when there are six obstacles into work environment and you try to insert more obstacles (figure 2.45).

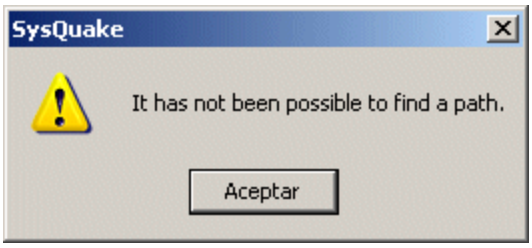


Figure 2.44. Simulation execution error message.

Figure 2.45. Obstacles excess error message.

- **Collision error.** It happens when mobile robot collides with one obstacle (figure 2.46).

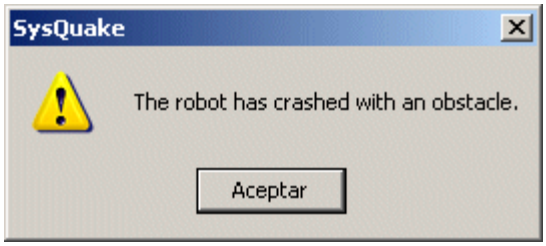


Figure 2.46. Mobile robot collision error message.

- **Local minimum error.** It happens when *Potential* algorithm is performing execution and the mobile robot can't overcome one obstacle (figure 2.47).

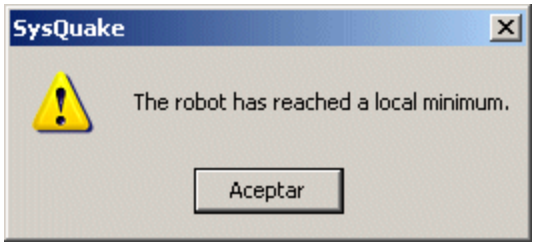


Figure 2.47. Work environment local minimum error message.

3. Examples.

In this chapter are presented some of the results obtained with *MRIT* interactive tool in an example way. Simulation examples of the different path planning and tracking algorithms are shown, as well as of the different kinematics, individually and comparatively.

3.1. Analysis and comparisson of navigation algorithms.

Different algorithms of the tool offer different results for the same planning problem. Along this section, various comparative examples where global and reactive algorithms are used are shown, proving that is possible to study which of them have better solutions in front of the navigation problem inside a specific work environment.

Next, each navigation and path tracking algorithms examples are shown, in which each algorithm characteristic parameters values changing effect can be observed.

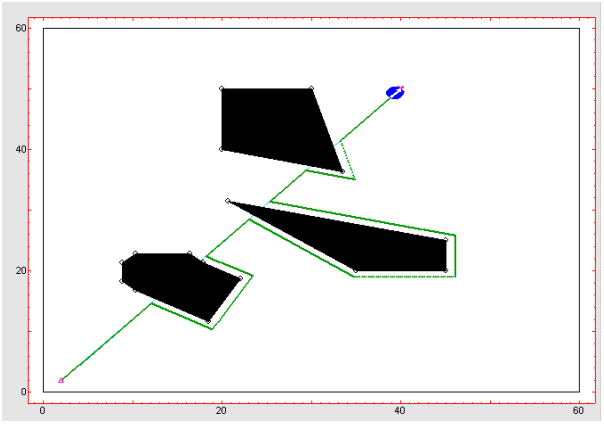
3.1.1 Reactive algorithms.

3.1.1.1. BUG.

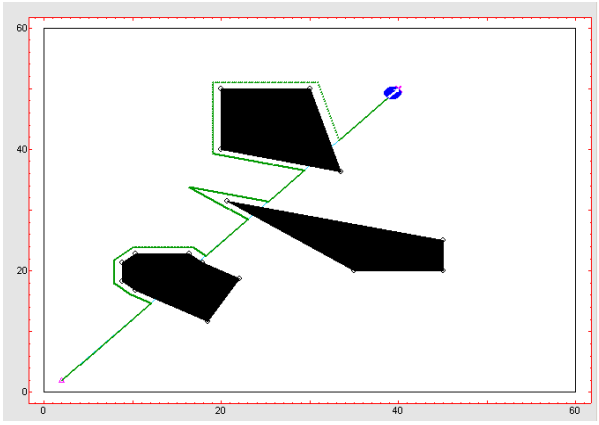
Like it was described previously, *BUG* algorithm has obstacles avoiding turn direction as characteristic parameter. With the aim to observe the influence of this parameter, figures 3.1 to 3.3 shows different examples for the two different turn directions, affected also by changes in robot *size* parameter.

3.1.1.2. VISBUG.

Refering to *VISBUG* algorithm, there exist two characteristic parameters, obstacles avoiding turn direction and distance sonars range (range sensors). Figures 3.4 to 3.6 shows changes of this two parameters effect examples. This graphics shows robot *size* change effect, jointly with *sensors range* and *turn direction* (*sensors range* and robot *size* are modified together because of increasing this last, minimum *sensors range* value increases to). We must point out that, in the case of the figure 3.6 (b), if the object to avoid and *sensors range* are too large, mobile robot can collide with work environment boundaries.

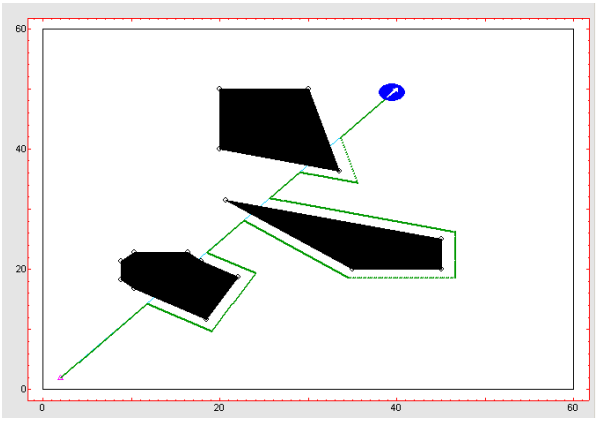


(a) Counterclockwise turn direction.

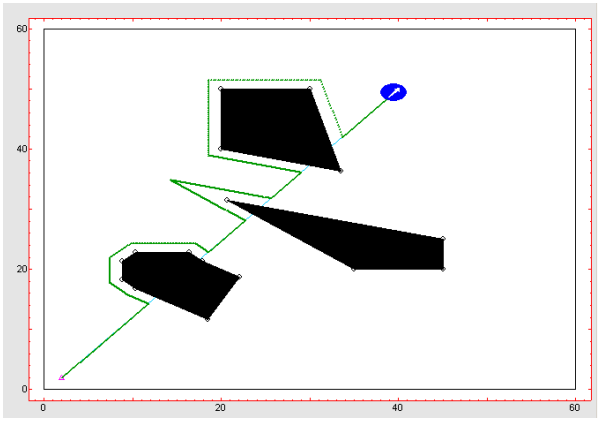


(b) Clockwise turn direction.

Figure 3.1. *BUG* algorithm with robot *size* 2.

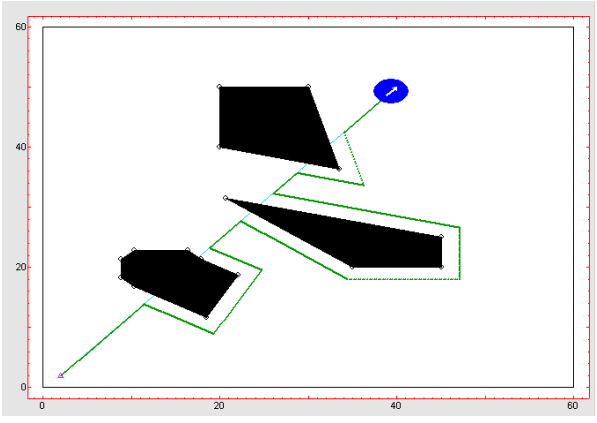


(a) Counterclockwise turn direction.

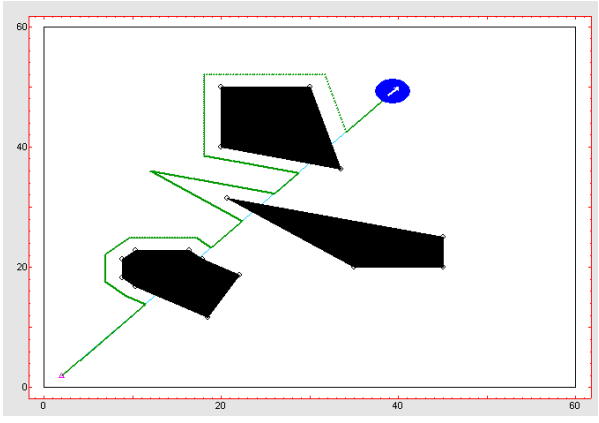


(b) Clockwise turn direction.

Figure 3.2. *BUG* algorithm with robot *size* 3.

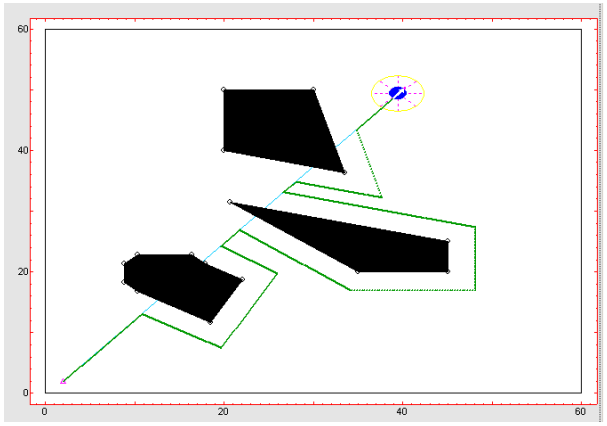


(a) Counterclockwise turn direction.

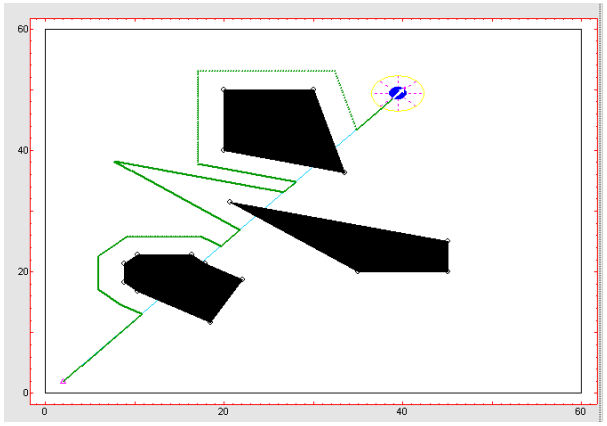


(b) Clockwise turn direction.

Figure 3.3. *BUG* algorithm with robot *size* 4.

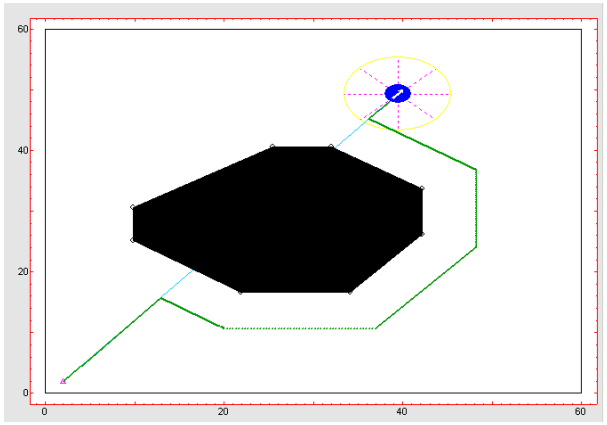


(a) Counterclockwise turn direction.

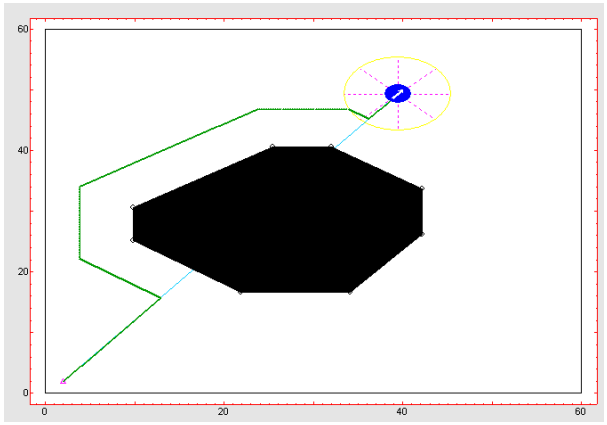


(b) Clockwise turn direction.

Figure 3.4. *VISBUG* algorithm with *sensors range* 3 and robot *size* 2.

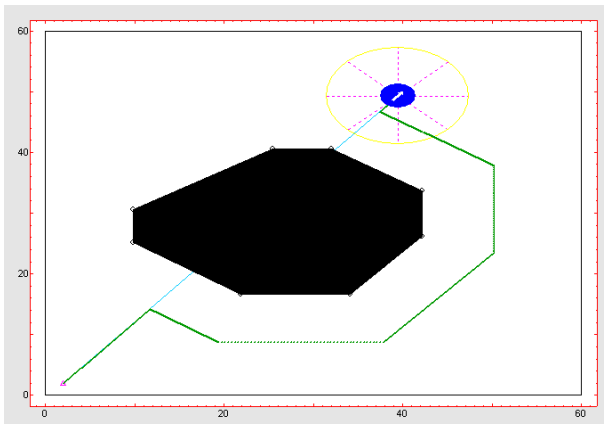


(a) Counterclockwise turn direction.

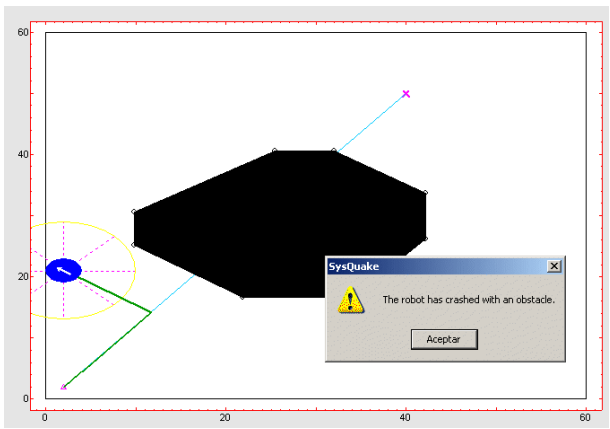


(b) Clockwise turn direction.

Figure 3.5. A *VISBUG* algorithm with *sensors range* 6 and robot *size* 3.



(a) Counterclockwise turn direction.

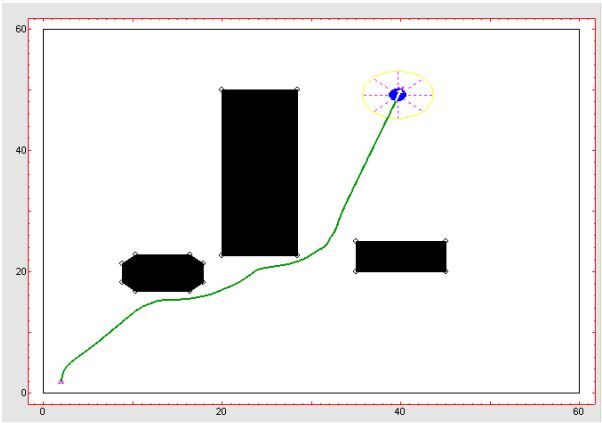


(b) Clockwise turn direction.

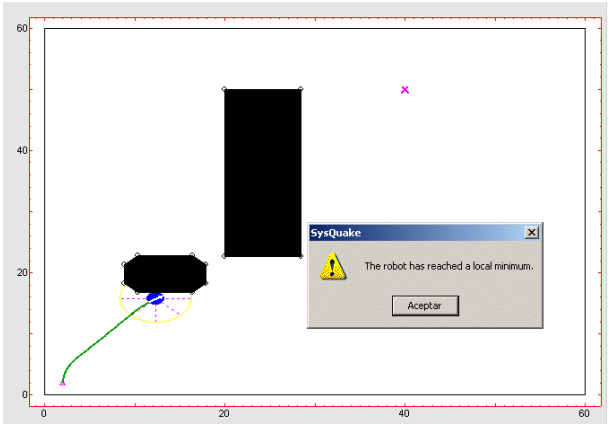
Figure 3.6. *VISBUG* algorithm with *sensors range* 8 and robot *size* 4.

3.1.1.3. Potential.

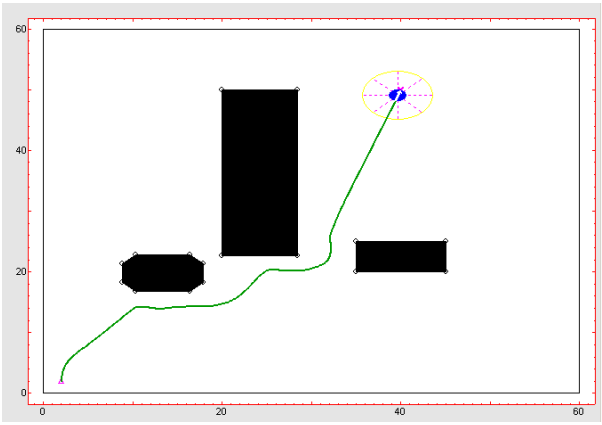
Potential algorithm has distance sensors range, attraction and repulsion constants as characteristic parameters. Figure 3.7 shows the effect of changing attraction and repulsion constants. As it can be observed, constants values can't have a big difference, since it can cause mobile robot to collide, because progress to goal position take priority over obstacles avoidance.



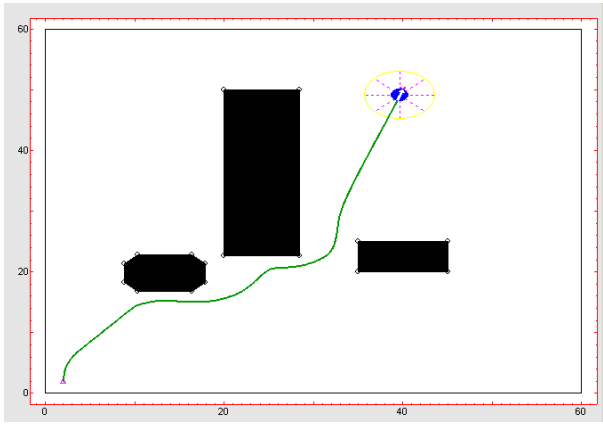
(a) *Attraction* 3 and *repulsion* 1.



(b) *Attraction* 4 and *repulsion* 1.



(c) *Attraction* 4 and *repulsion* 2.

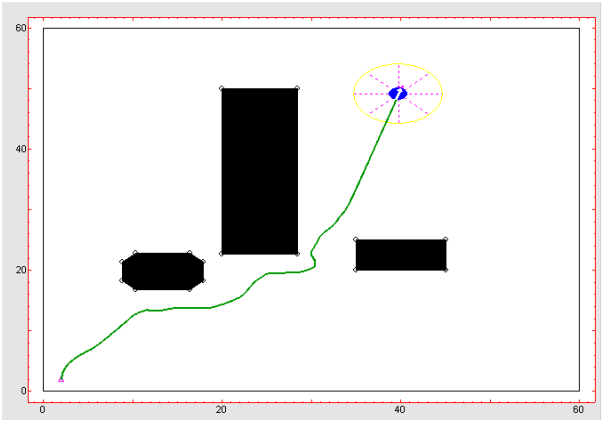


(d) *Attraction* 5 and *repulsion* 2.

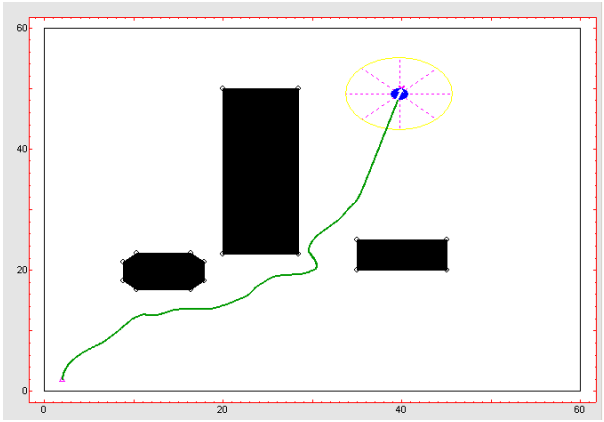
Figure 3.7. *Potential* algorithm with *sensors range* 4.

Figure 3.8 shows the effect of changing mobile robot *sensors range*. One must point out that, the higher the sensors range, the better the robot reacts in obstacles presence, because path tracked is more away from them, and therefore more safety.

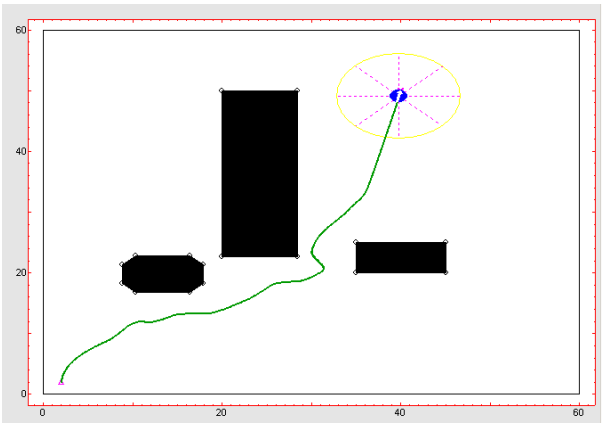
With aim to shows possible difficulties, figure 3.9 shows one example where, in spite of having a very great sensors range, the mobile robot is too large and collides when try to move through two close obstacles.



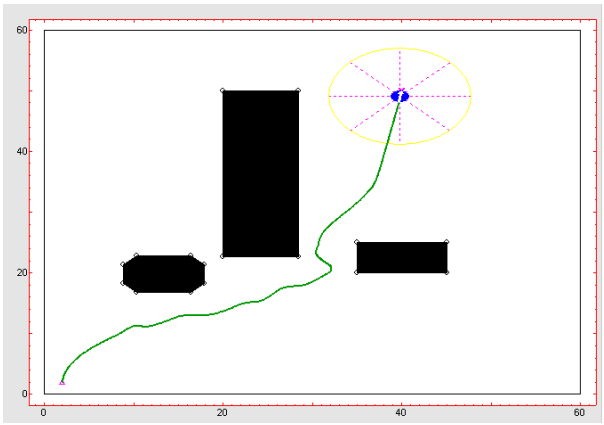
(a) *Sensors range* 5.



(b) *Sensors range* 6.



(c) *Sensors range* 7.



(d) *Sensors range* 8.

Figure 3.8. *Potential* algorithm with *attraction* 2 and *repulsion* 1.

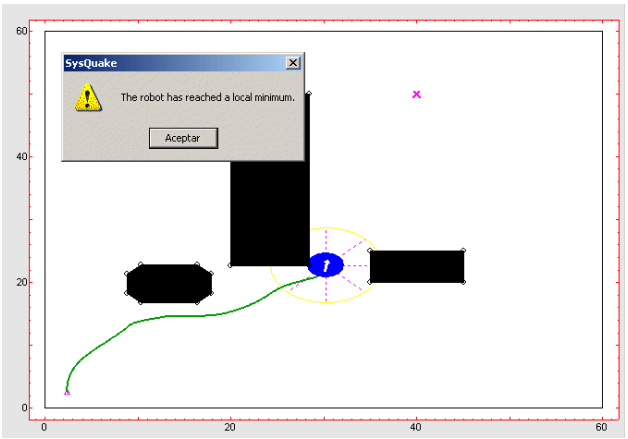
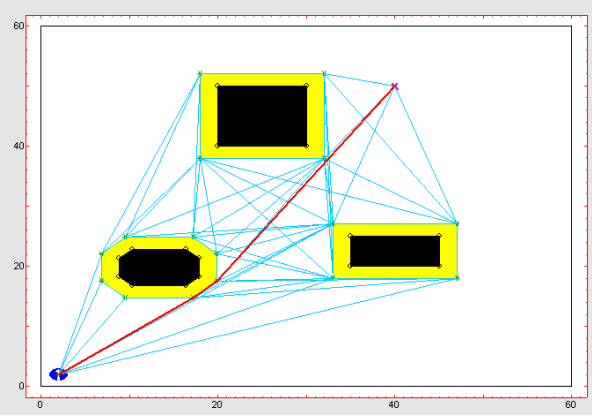


Figure 3.9. *Potential* algorithm with *sensors range* 6, *attraction* 5, *repulsion* 2 and robot *size* 4.

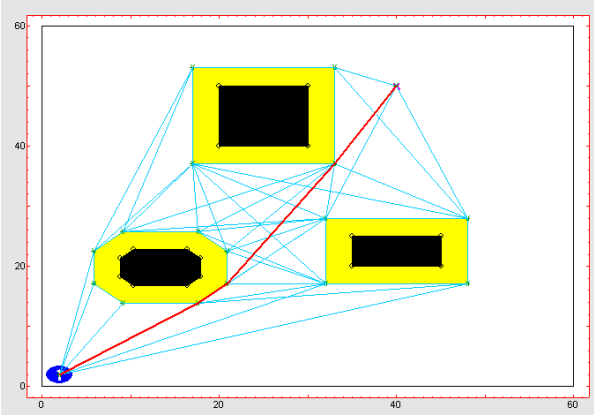
3.1.2. Global Algorithms.

3.1.2.1. Visibility Graph.

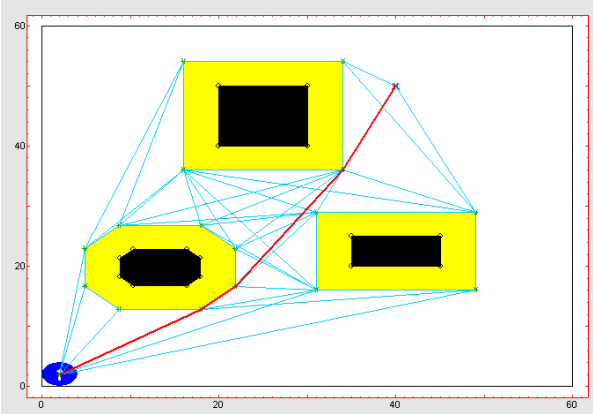
Visibility Graph algorithm has no characteristic parameters; therefore, with sights of showing its operation, figure 3.10 shows robot *size* change effect over algorithm results. From this examples can be showed that the larger the robot size, the larger the obstacles are increased, and therefore, the sharper the calculated trajectory curvature.



(a) Robot *size* 2.



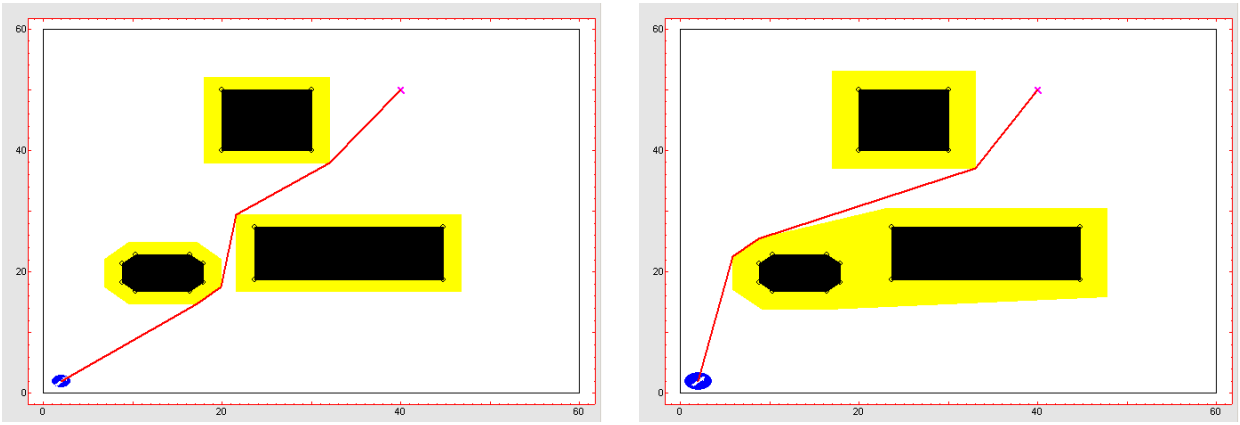
(b) Robot *size* 3.



(c) Robot *size* 4.

Figure 3.10. *Visibility Graph* algorithm with different robot sizes.

The effect of augmenting robot size, jointly with corresponding obstacles enlargement, can take place to obstacles merge, as can be observed in figure 3.11 example. Notice as lower work environment obstacles proximity makes them merge when increasing robot *size*, taking place to different trajectories.



(a) Robot *size* 2. **(b)** Robot *size* 3.

Figure 3.11. *Visibility Graph* algorithm obstacles merge.

3.1.2.2. Voronoi Diagram.

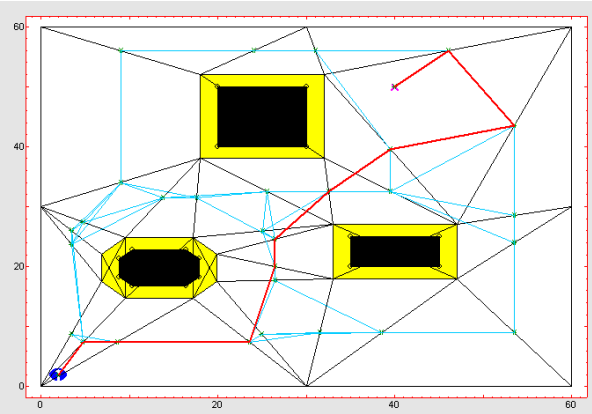
Just like *Visibility Graph* algorithm, *Voronoi Diagram* based on one has not characteristic parameter either. Figure 3.12 shows robot size enlargement effect. In examples can be observed that algorithm calculate, each case, one trajectory that fits obstacles and mobile robot wides, achieving that it keeps farest away from obstacles and work environment boundaries.

One must point out that figure 3.12 (b) path is totally different to other ones in the same figure. That's due to, when enlarging obstacles, possible paths quantity varies and, therefore, minimum path can be totally opposite to the previous to this change one.

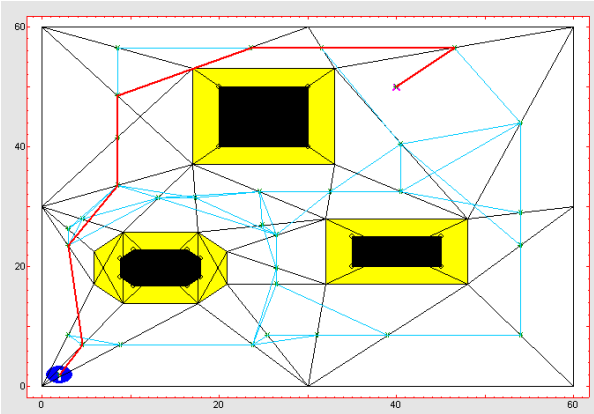
3.1.2.3. Quadtree.

Quadtree algorithm has grid size as characteristic parameter. Figures 3.13 to 3.15 show *quadtree area* and robot *size* jointly increasing effect.

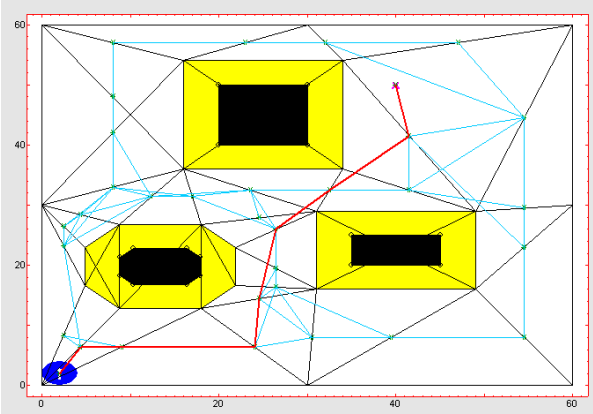
It can be telled that, the larger the *quadtree area*, the better calculated path efficiency, whenever work environment not containing too many obstacles. Same way, it can be observed that *quadtree area* enlarging inneficiency is reflected in paths with abrupt trajectory changes.



(a) Robot *size* 2.

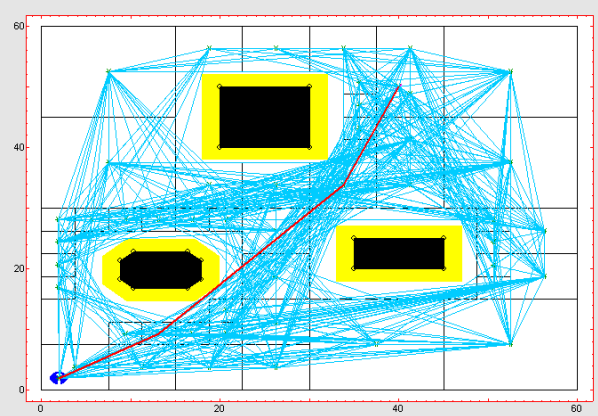


(b) Robot *size* 3.

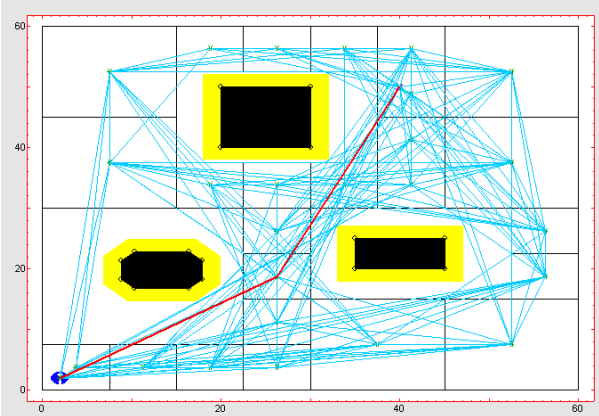


(c) Robot *size* 4.

Figure 3.12. *Voronoi Diagram* algorithm with different robot sizes.

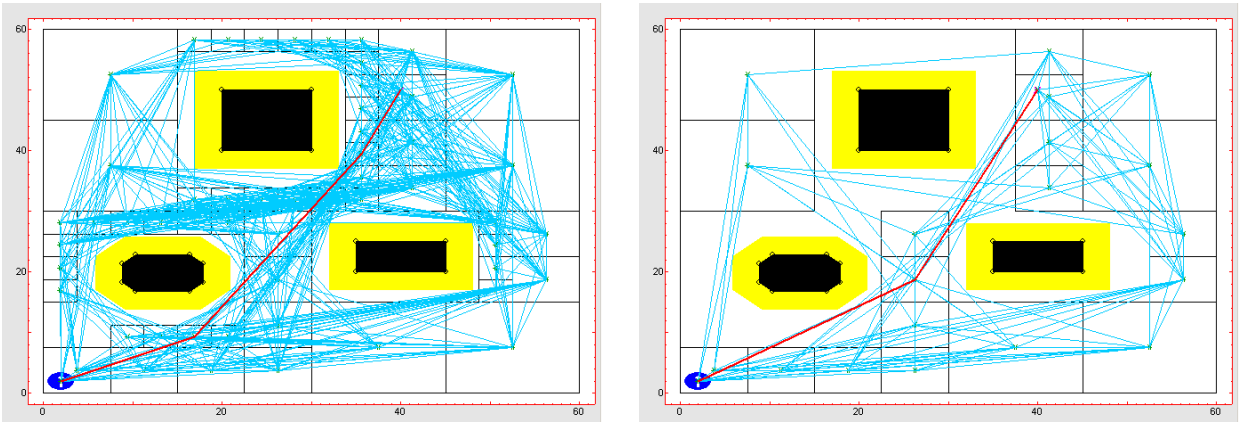


(a) *Quadtree area* 1.



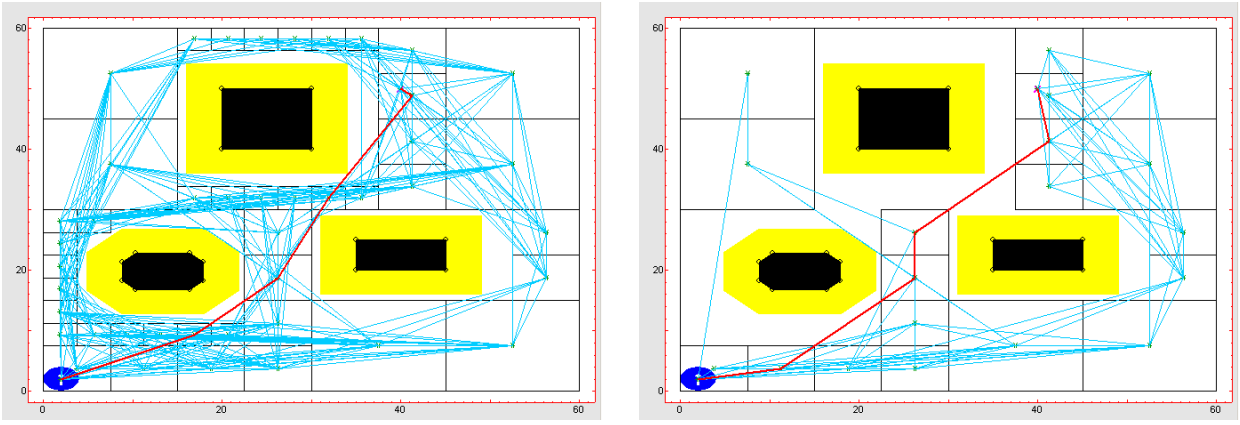
(b) *Quadtree area* 2.

Figure 3.13. *Quadtree* algorithm with different *quadtree area* and robot *size* 2.



(a) *Quadtree area* 1. **(b)** *Quadtree area* 2.

Figure 3.14. *Quadtree* algorithm with different *quadtree area* and robot *size* 3.



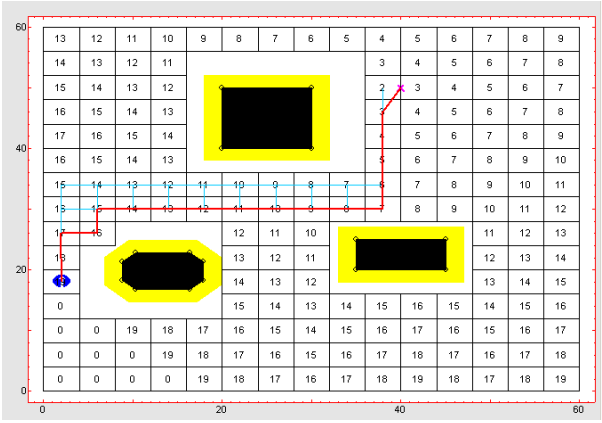
(a) *Quadtree area* 1. **(b)** *Quadtree area* 2.

Figure 3.15. *Quadtree* algorithm with different *quadtree area* and robot *size* 4.

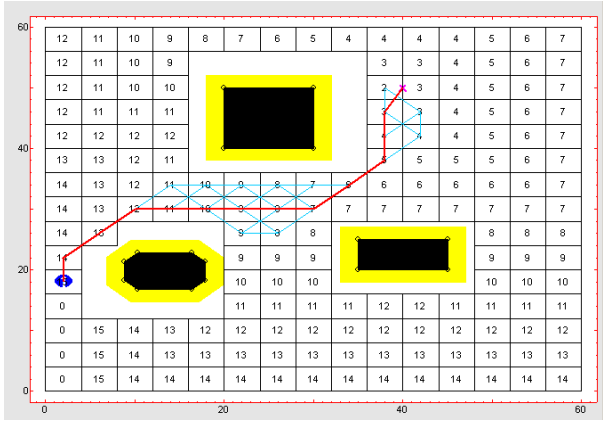
3.1.2.4. Wave Front.

Last of global algorithms is *Wave Front*, that has *cell size* and *neighborhood* as characteristic parameters. Figures 3.16 and 3.18 show different examples in which *cell size* and *neighborhood* are jointly modified.

As conclusion it has to be mentioned that when using *neighborhood* 8, obtained trajectory is smoother than using *neighborhood* 4, since that last don't count on with path possible diagonal segments.

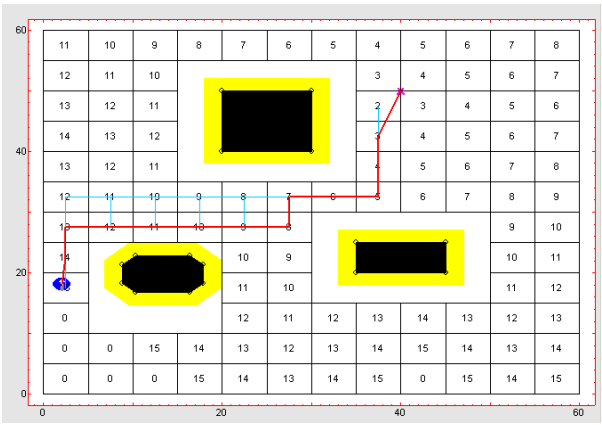


(a) *Neighborhood* 4.

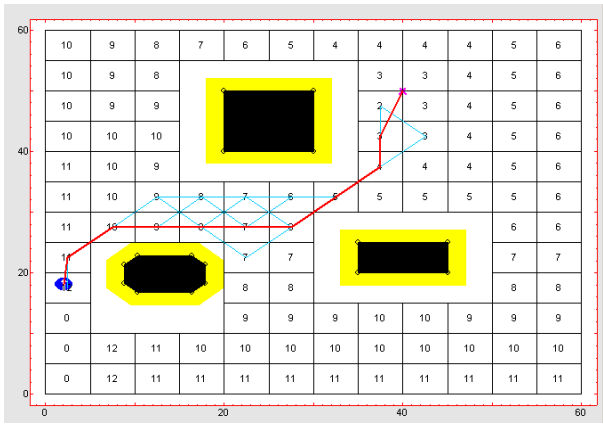


(b) *Neighborhood* 8.

Figure 3.16. *Wave Front* algorithm with *cell size* 4 and robot *size* 2.

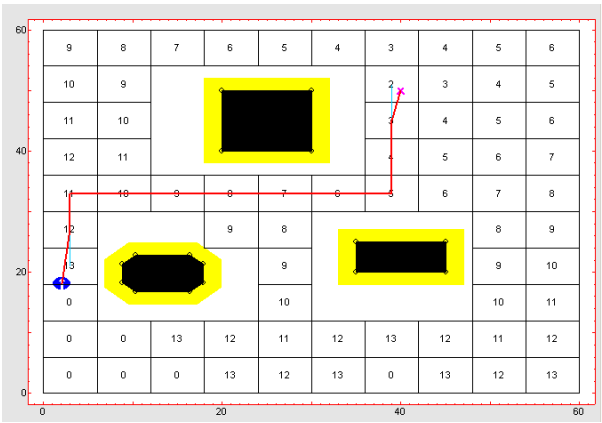


(a) *Neighborhood* 4.

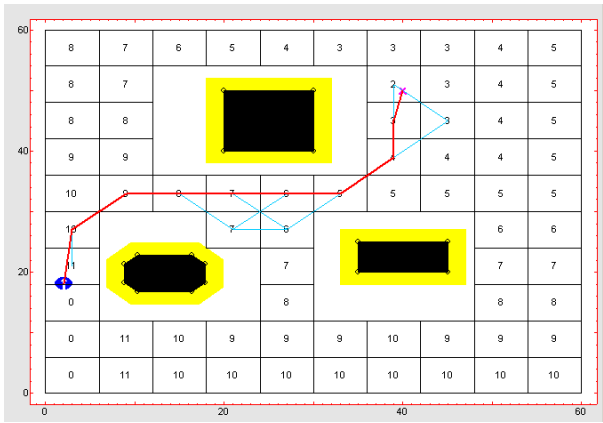


(b) *Neighborhood* 8.

Figure 3.17. *Wave Front* algorithm with *cell size* 5 and robot *size* 2.



(a) *Neighborhood* 4.



(b) *Neighborhood* 8.

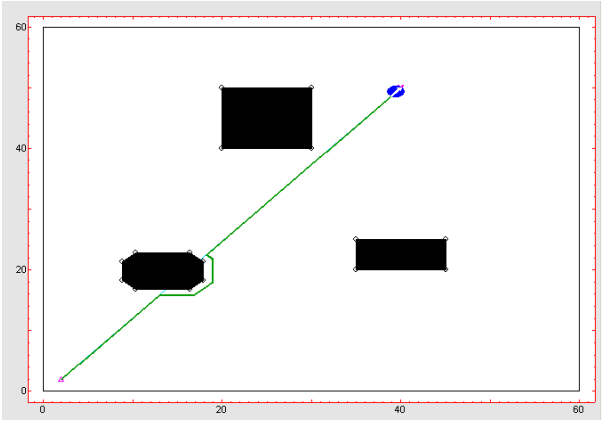
Figure 3.18. *Wave Front* algorithm with *cell size* 6 and robot *size* 2.

3.1.3. Navigation algorithms comparisson.

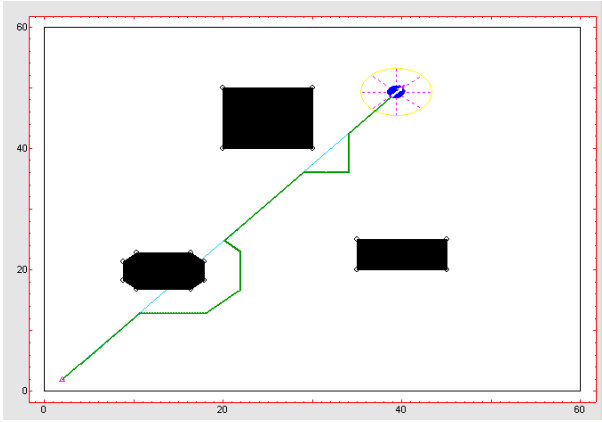
This section shows the different results that can be obtained with the different navigation algorithms for the same given work environment. One given work environment is showed and global and reactive algorithm results are compared.

When it's results comparisson time and the most suitable algorithm has to be determined, it's necessary to set results evaluation criterions. In Mobile Robotics field, most used criterions are basis on path's length, curvature smoothness and obstacles distance (more safety). In this way, examples shown results are compared in a heuristic way, counting on with these characteristics, since exact calculation have not be done (except global algorithms length case, where minimum lentgh path is always calculated).

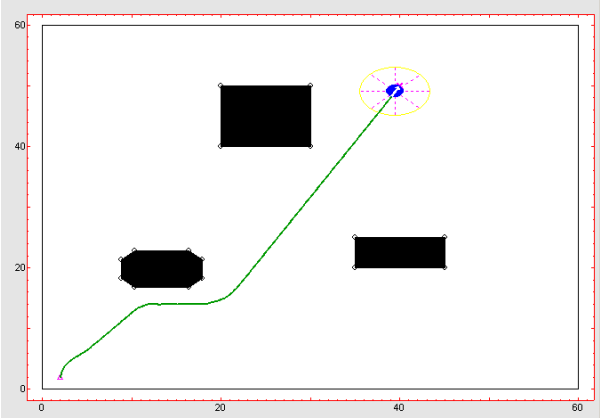
First, reactive algorithms use obtained results are shown. All results can be observed through figure 3.19.



(a) *BUG*.



(b) *VISBUG*.

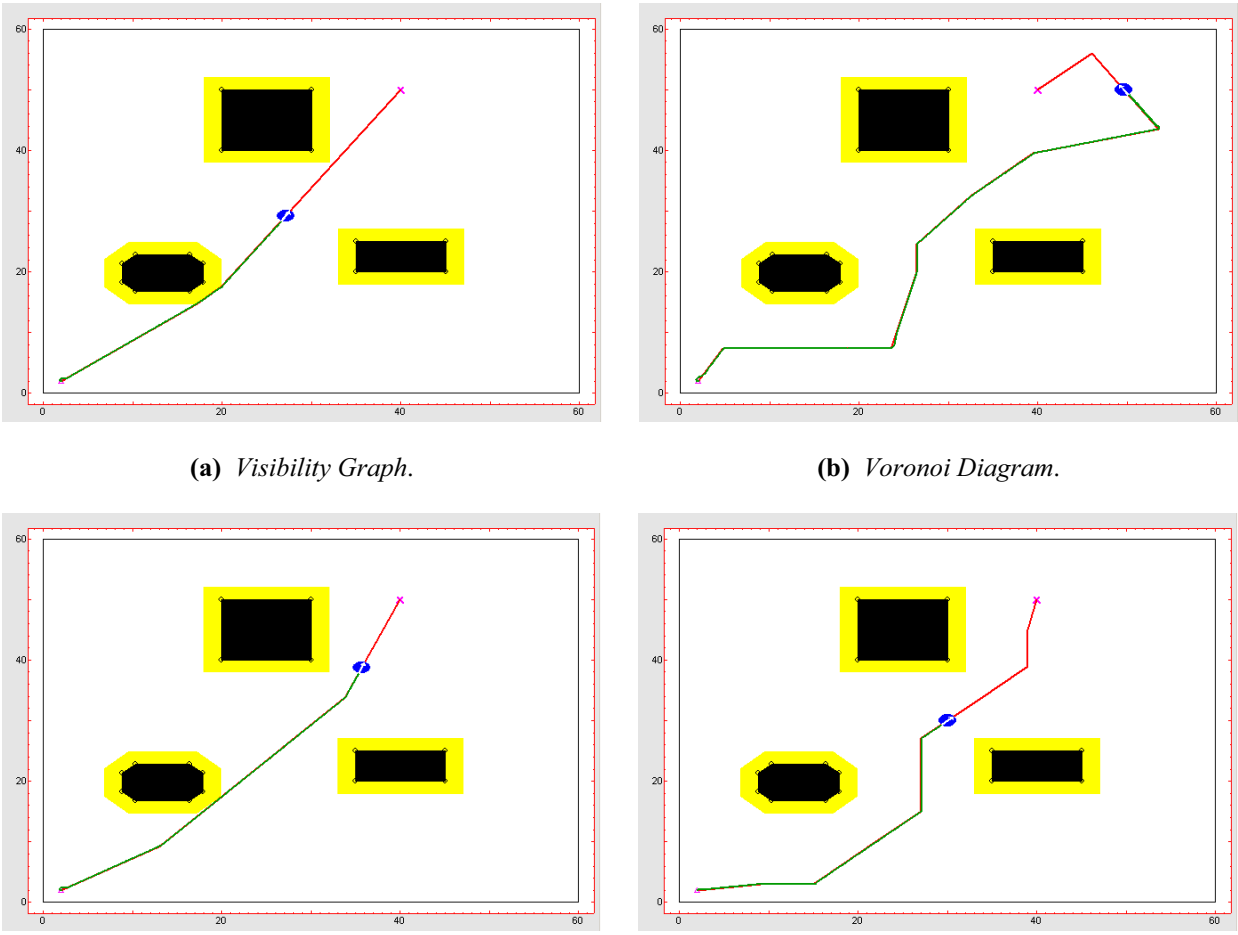


(c) *Potential*.

Figure 3.19. Local planning algorithms example.

Taking in account those previously mentioned criterions, it seems like *BUG* algorithm is the one with the minimum path length, *Potential* one is the smoother and *VISBUG* one the more safety from work environment obstacles. When it's time to select the more suitable algorithm, work environment layout has to be taked in account.

Figure 3.20 shows one comparative example with all global planning algorithms inside the same work environment. It can be observed how minimum paths calculated hereby verify work environment layout previous knowledge that global algorithms have. *Visibility Graph* and *Quadtree* ones seems to be the shortest and smoother paths, while *Voronoi Diagram* achieve farest away from obstacles path.



(a) *Visibility Graph*. **(b)** *Voronoi Diagram*.

(c) *Quadtree*. **(d)** *Wave Front*.

Figure 3.20. Global planning algorithms example.

Jointly comparing global and reactive algorithms, *Visibility Graph* is the one who calculates the shortest and smoother path, while *Voronoi Diagram* gets the more safety one.

3.1.4. Pure Pursuit research.

Pure Pursuit algorithm is used by global planning algorithms to make mobile robot track the calculated trajectory. This path tracking algorithm has *sampling time* (T_m) and *objective point distance* (d) as characteristic parameters.

When changing *sampling time*, more control signals are available to track the calculated trajectory and, in consecuence, robot will keep a smoother path the lower the *sampling time*. Figure 3.21 shows one example with 0.1 and 0.03 *sampling time* values.

(a) *Sampling time* 0.1. **(b)** *Sampling time* 0.03.

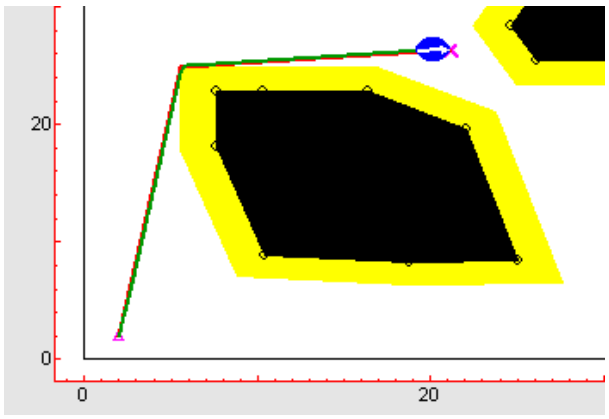
Figure 3.21. *Pure Pursuit* example with different *sampling time* T_m.

The second characteristic parameter, *objective point distance* (d), makes the robot trajectory tighter or straighter at curves. As can be observed in figure 3.22, the lower the value of d, lower smotthness when taking a curve and, the greater the value, greater circumference arch and, so, greater smoothness.

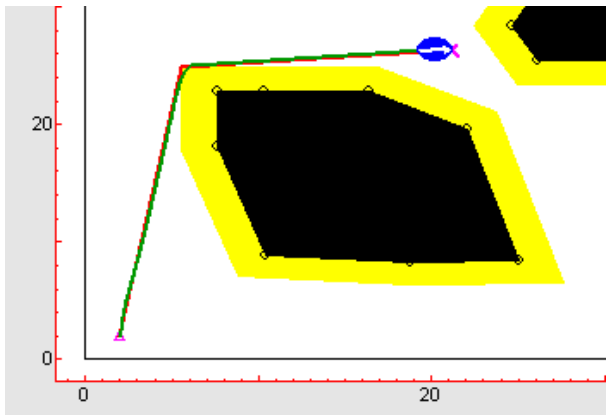
When tracking one trajectory with *Pure Pursuit* algorihtm, mobile robot characteristic parameters values affects this one to. Figure 3.23 shows one example with different initial orientations, and figure 3.24 shows another one where different *speed* values effect can be observed.

3.2. Kinematics behavior comparisson.

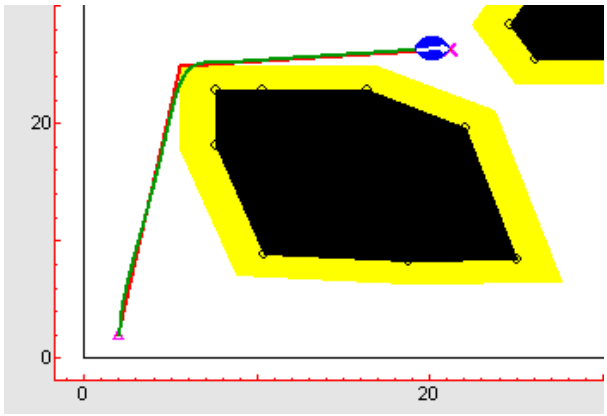
Tool kinematics models are: *Differential*, *Tricycle* and *Synchronous*. Each one has different control signals, which determines it behavior. In the following sections are presented different examples aimed to observe the effect that each different kinematics control signals have.



(a) *Objective point distance* 2.

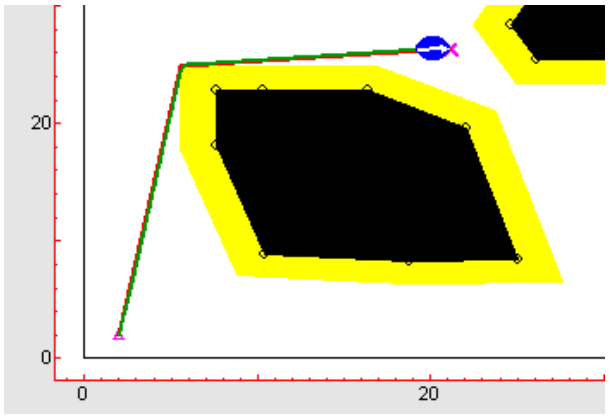


(b) *Objective point distance* 5.

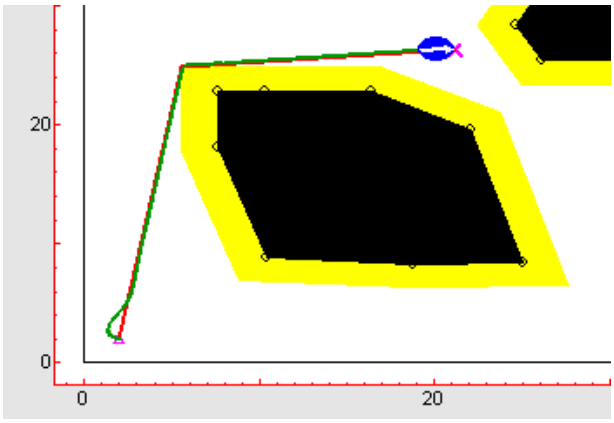


(c) *Objective point distance* 8.

Figure 3.22. *Pure Pursuit* examples with different *objective point distance d*.

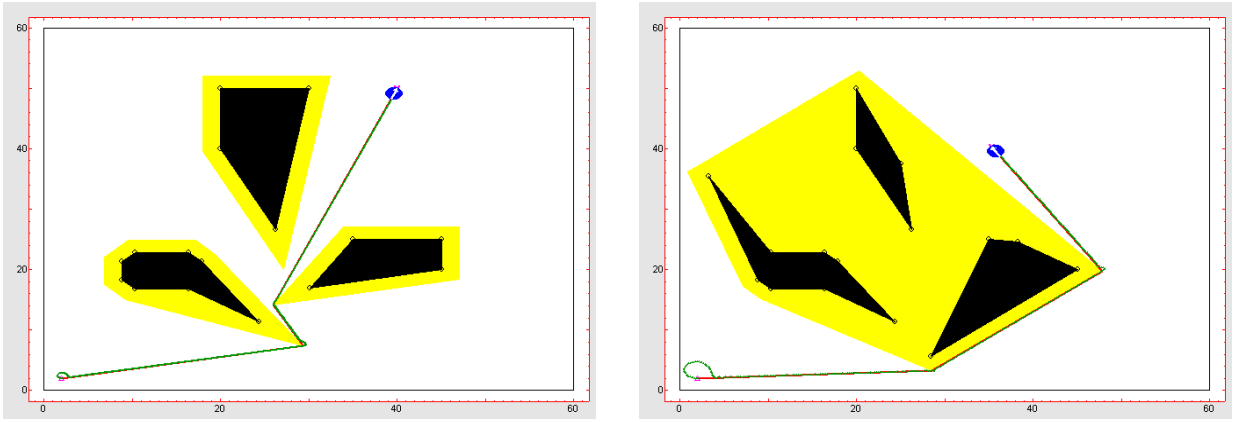


(a) *Initial orientation* 1.57.



(b) *Initial orientation* 2.6.

Figure 3.23. *Pure Pursuit* examples with different *initial orientation*.

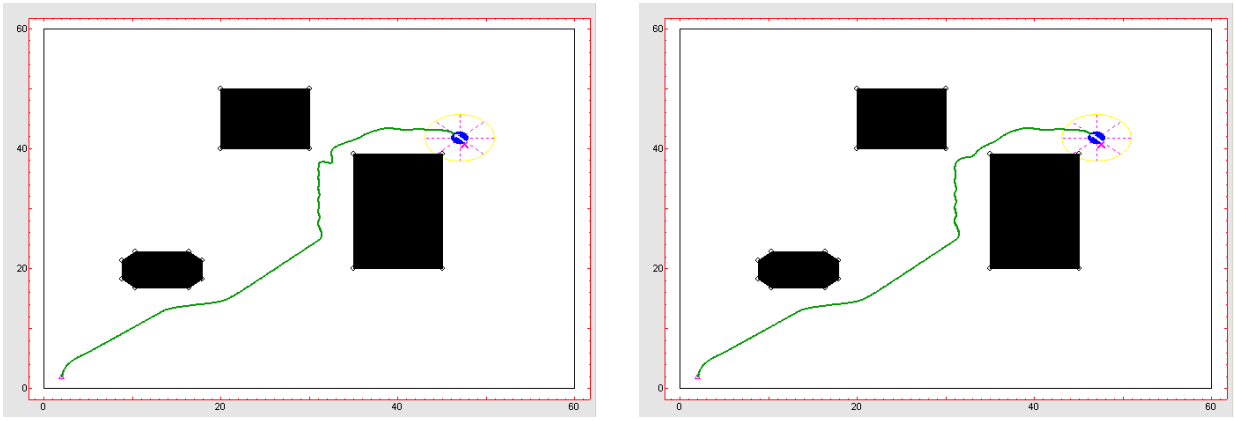


(a) *Speed* 1. **(b)** *Speed* 2.

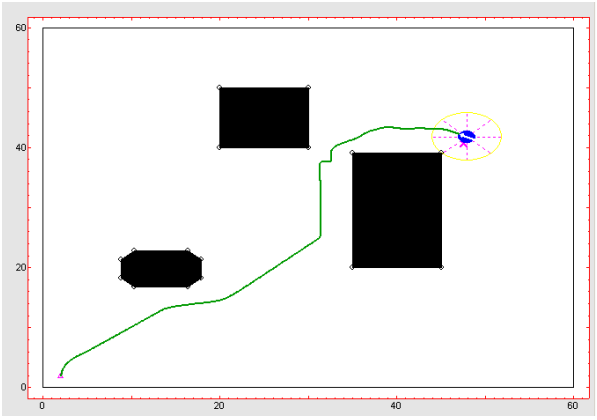
Figure 3.24. *Pure Pursuit* examples with different *speed* and the same *initial orientation*.

3.2.1. Kinematics examples.

Figure 3.25 shows one example where the results of tracking the same path into the same work environment using different kinematics can be observed.



(a) *Differential*. **(b)** *Tricycle*.

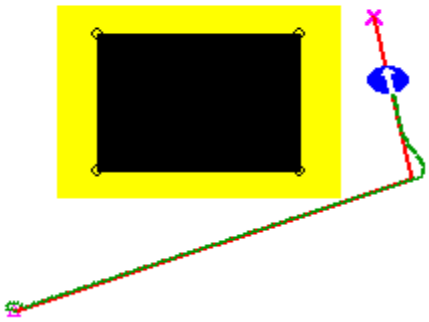


(c) *Synchronous*.

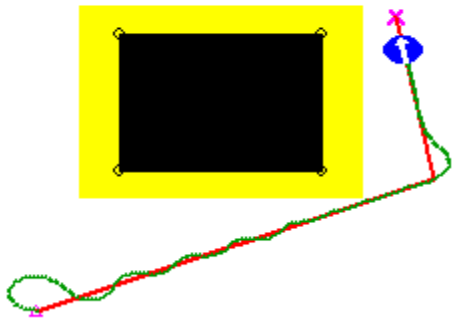
Figure 3.25. *Potential* algorithm with different kinematics.

In the last example, *tricycle* kinematics directional wheel turn angle is supossed to be constrained between 1 and -1 radians, and *synchronous* kinematics wheels orientation angle is supossed to be constrained between 1.5 and -1.5 radians. It can be observed how curves tracked by mobile robot are different in each different kinematics model case, due to each model control signals basis constraints.

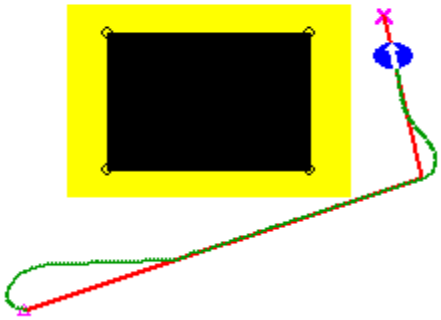
Figure 3.26 shows a clearly example of kinematics comparisson where planned trajectory has a ninety degrees curvature.



(a) *Differential* kinematics.



(b) *Tricycle* kinematics.

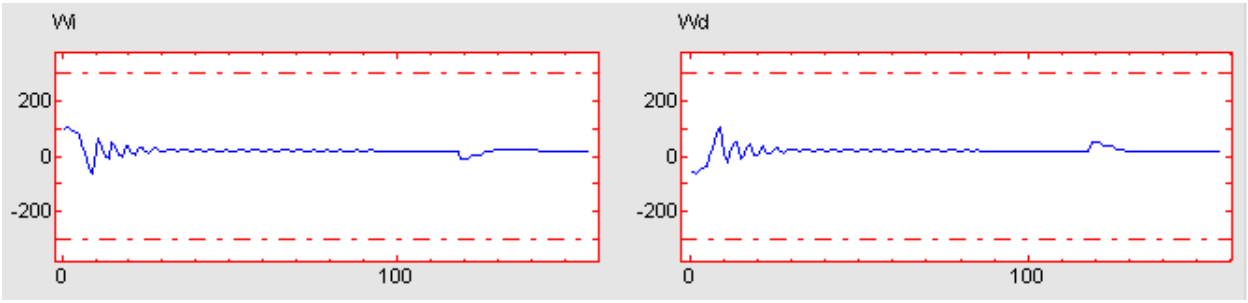


(c) *Synchronous* kinematics.

Figure 3.26. *Quadtree* algorithm with different kinematics.

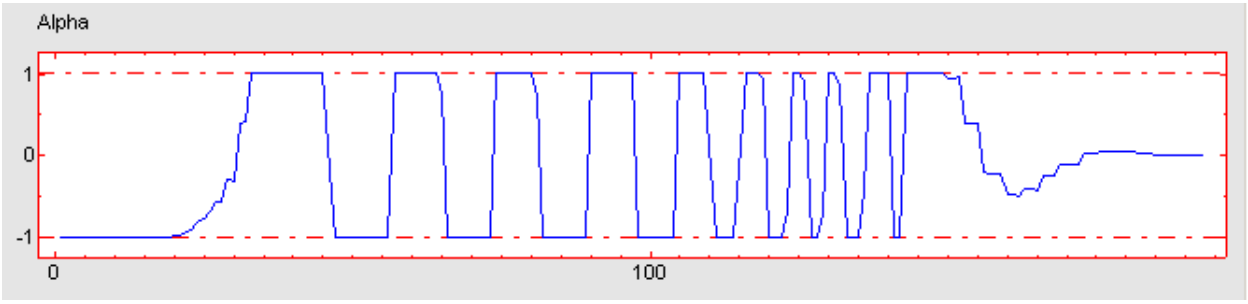
Point out that *differential* kinematics, the only one with null turn angle, minimizes angles during turns, especially into ninety degrees curve. *Tricycle* kinematics needs a bigger angle at start as inside the curve, even zigzaging when it reaches planned trajectory, due to mobile robot speed. Finishing, *synchronous* kinematics makes the robot do slower and curved turns, due to mobile robot wheels orientation constraint.

Figure 3.27 shows figure 3.26 kinematics characteristic control signals values graphics.

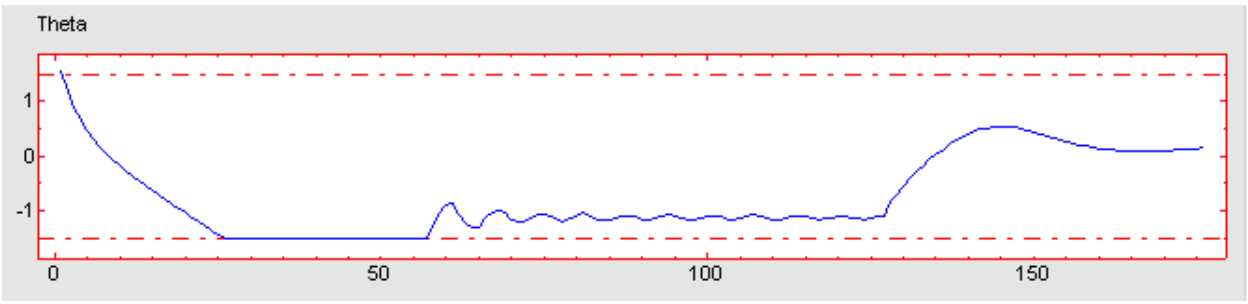


(a) *Differential* kinematics control signals.

.



(b) *Tricycle* kinematics control signal.



(c) *Synchronous* kinematics control signal.

Figure 3.27. Kinematics control signals.

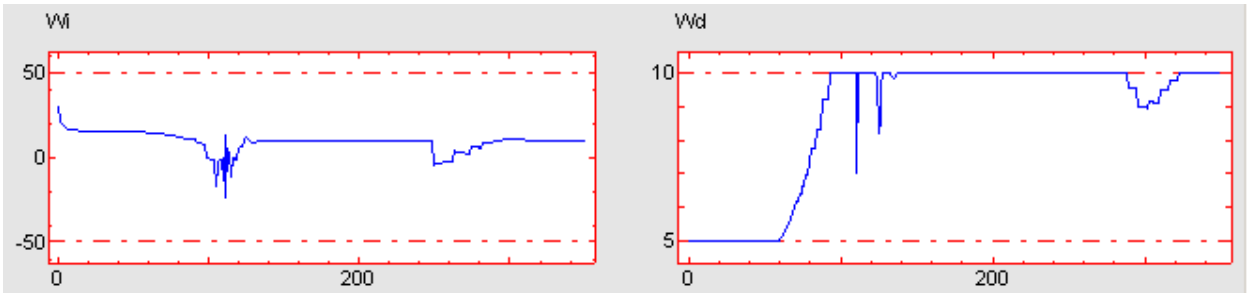
3.2.2. Constraints research.

As it was mentioned previously, tool graphics where each kinematics control signals are shown are interactive ones. That means that value upper and lower limits can be modified to, in example, simulating mobile robot wheels blockade problems.

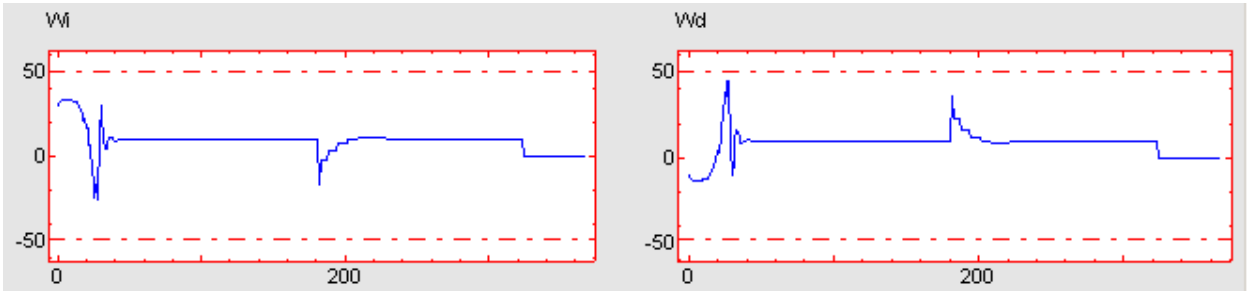
Figure 3.28 shows one example where a simulated mobile robot right wheel blockade problem is done through one ω_d control signal constraint. Figure 3.29 shows graphics for *differential* kinematics control signals of figure 3.28 example.

(a) Constraints. **(b)** No constraints.

Figure 3.28. *Differential* kinematics with and without constraints.



(a) Constraints.

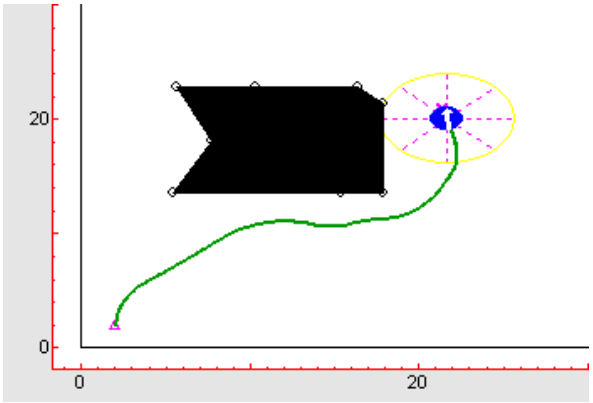


(b) No constraints.

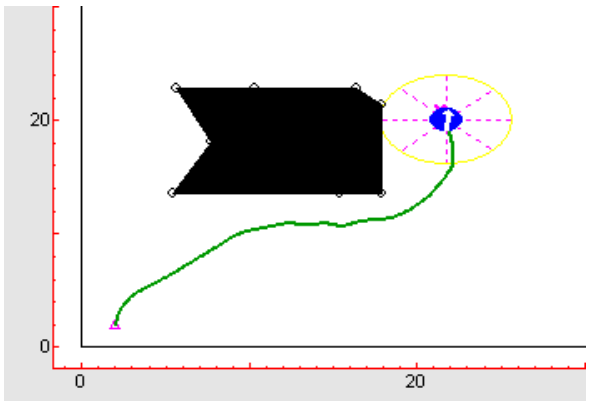
Figure 3.29. *Differential* kinematics control signals with and without constraints.

It can be observed that this simulated right wheel blockade problem makes mobile robot turn capacity to be limited, and that's don't allow it to take a little sharp curves, needing more time and more angle than in constraints free case to do this.

In *tricycle* kinematics case, its control signal is the directional wheel turn angle (α). Figure 3.30 shows an example where theres a mobile robot with *tricycle* kinematics where a simulated directional wheel blockade is done by modifying α control signal basis constraint. Figure 3.31 shows *tricycle* kinematics control signal graphics for this example.

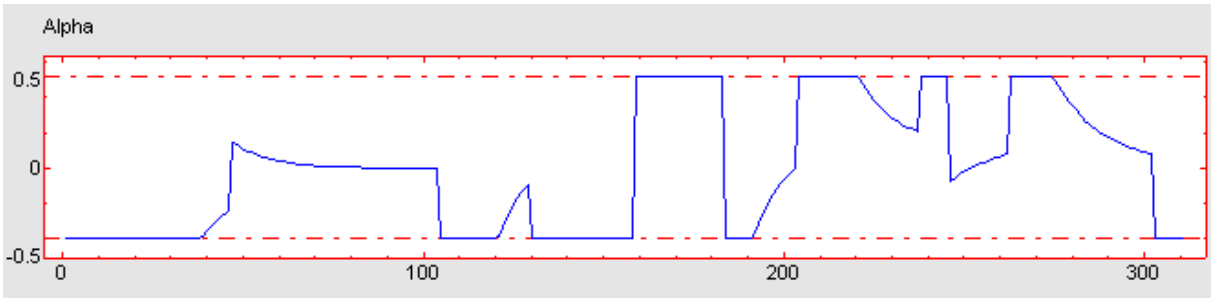


(a) Con restricción entre 0.5 y -0.5 radianes.

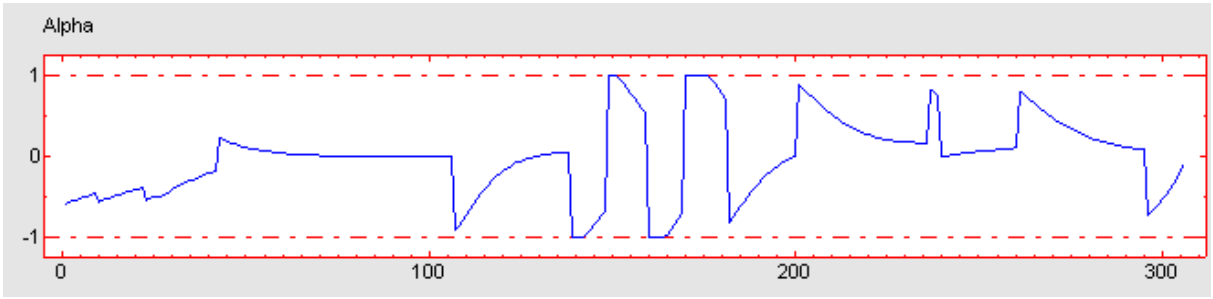


(b) Con restricción base entre 1 y -1 radianes.

Figure 3.30. *Tricycle* kinematics constraints examples.



(a) Constraint between 0.5 and -0.5 radians.

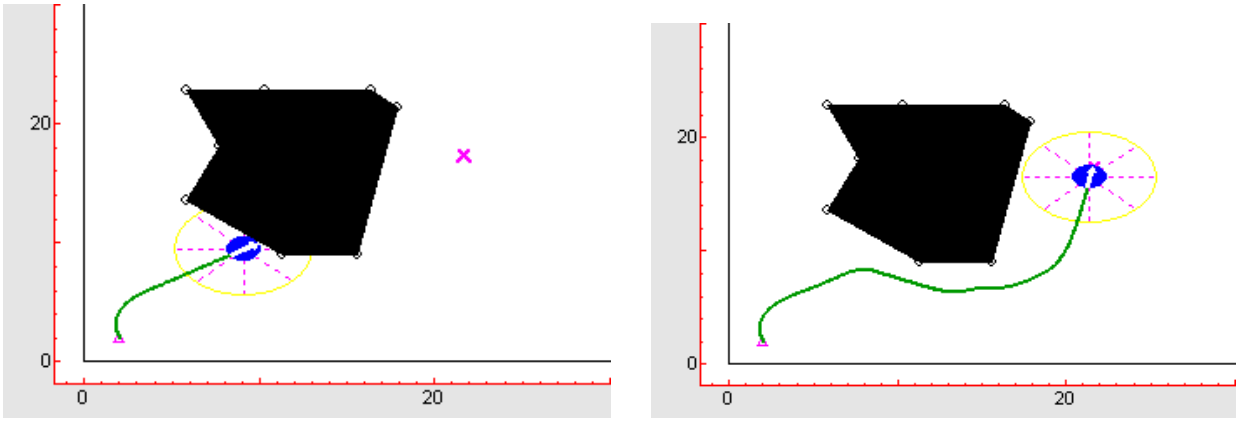


(b) Basis constraint between 1 and -1 radians.

Figure 3.31. *Tricycle* kinematics control signal constraints.

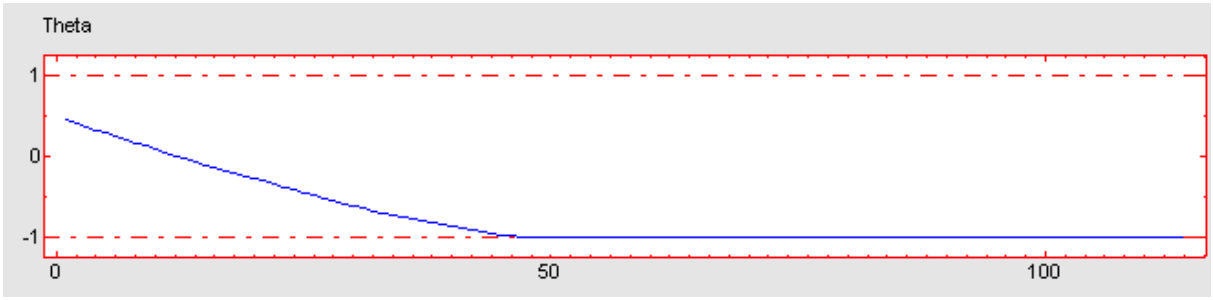
Previous graphics show that the simulated directional wheel blockade problem makes mobile robot turn angle to be limited and so, it has to take more open than necessary curves.

Synchronous kinematics has wheels orientation angle (ϕ) as control signal. Figure 3.32 shows an example where theres a mobile robot with *synchronous* kinematics where a simulated wheels orientation blockade problem is done by modifying ϕ control signal basis constraint. Figure 3.33 shows graphics for *synchronous* kinematics characteristic control signal for this example.

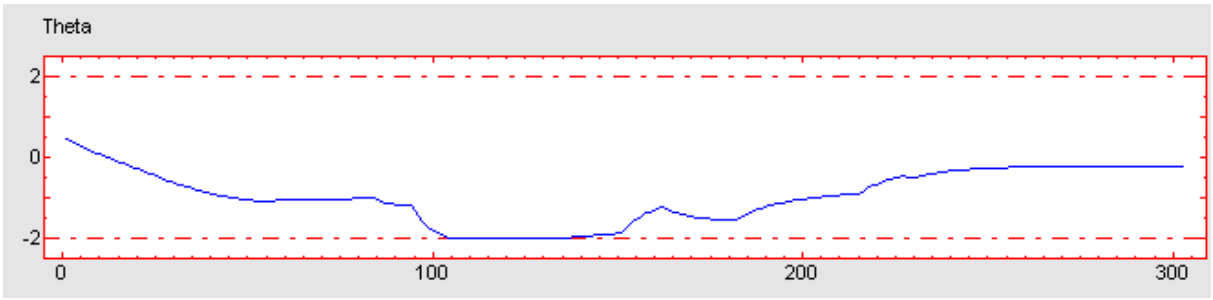


(a) Constraint between 1 and -1 radians. **(b)** Constraint between 2 y -2 radians.

Figure 3.32. *Synchronous* kinematics constraints examples.



(a) Constraint between 1 and -1 radians.



(b) Constraint between 2 and -2 radians.

Figure 3.33. *Synchronous* kinematics control signal constraints.

Through obtained results can observe that, when constraining robot wheels orientation angle it can't overcome the obstacle, colliding eith it due to wheels can't turn enough to avoiding it.