A Matlab-based Interactive Simulator for Teaching Mobile Robotics

Ramón González[a,*], Cristian Mahulea[a], Marius Kloetzer[b]

[a]*Aragón Institute of Engineering Research (I3A), University of Zaragoza, Spain*
[b]*Dept. of Automatic Control and Applied Informatics, Technical University of Iasi, Romania*

Abstract

This paper presents an open-source Matlab-based interactive software tool for teaching mobile robotics in introductory courses. In particular, it deals with the subjects: modeling, path planning, and motion control. This simulator offers the advantage of testing different aspects related to these fundamental topics and instantly seeing the result in a Graphical User Interface (GUI). This fact leads to a high realism in interactivity, while no previous knowledge in Matlab or programming is required. Additionally, it constitutes an easily scalable tool, the GUI has been properly designed with just one single window with straightforward and intuitive buttons, icons, and figures. Some illustrative examples demonstrate the benefits of the proposed simulator.

Keywords: Educational GUI, Robot kinematic model, Path planning, Motion control

1. Introduction

Traditionally many courses dealing with mobile robotics are taught through lectures and lab sessions [5, 28, 34, 36]. Lectures focus on introducing and explaining theoretical concepts (e.g. abstract mathematical and physical developments, geometry, algorithms, etc.). Lab sessions are related to assembling, programming, and testing mobile robots often using robotic kits like Lego Mindstorms [14, 20]. However, sometimes the transition from theoretical lectures to lab sessions, where actual robots are employed, is not straightforward, especially for introductory courses. In this regard, it is becoming popular the use of interactive software simulators to drive the theoretical explanations. It is clear that a student will comprehend more quickly a theoretical concept if they can manipulate several parameters and instantly

[*]Corresponding author
Email addresses: `ramongrobot@gmail.com` (Ramón González), `cmahulea@unizar.es` (Cristian Mahulea), `kmarius@ac.tuiasi.ro` (Marius Kloetzer)

see the effect/result [15, 44]. Thus, if such software offers the possibility to compare similar approaches, the learning process will be even worthier.

In order to overcome the gap between theoretical concepts and lab sessions related to mobile robotics some simulators have been proposed in the literature. Many of them have been implemented in Matlab. A pioneering work in the field of robotics simulators was [11]. Here, the popular robotics toolbox for Matlab was first introduced. This toolbox permits to work with serial-link manipulators considering kinematics and dynamics. An extended version of this toolbox adding mobile robotics functionalities appears in [12]. Here a comprehensive set of *Matlab* and *Simulink* scripts deals with mobile robot navigation: motion planning, motion control, as well as localization and mapping. Another pioneering toolbox specifically related to mobile robotics was SIMROBOT [24]. It was a toolbox that allows the user to simulate the behavior of one or more mobile robots. Each robot could be equipped with several virtual sensors. The control came from using Fuzzy Logic and Neural Network toolboxes available for Matlab. However, it constitutes an obsolete tool and no support is available today (it was implemented only for Matlab 5). A more recent simulator extended from SIMROBOT is MRSim [13]. This software is mainly focused on multi-robot simulation. In this sense, interesting features such as inter-robot communication and collective mapping are available. In [37] a Matlab-based simulator, called AMORsim, is presented. In particular, it deals with localization and it permits to evaluate the effects of systematic and non-systematic errors in robot pose estimation. Additionally, obstacle detection is also simulated. A navigation toolbox related to robot localization and Simultaneous Localization and Mapping (SLAM) is detailed in [2]. In this case, previously recorded sensorial data can be loaded to reproduce a desired simulation.

On the other hand, we can also find software simulators implemented in other languages. Such an example is USARsim [7]. This software constitutes the simulation engine used to run the Virtual Robots Competition within the Robocup initiative. Among the main features, it allows simulation of multiple sensors and actuators, several platforms (wheeled and tracked robots, among others), computer vision, and controllers. In [23], the Mobile Robotics Interactive Tool (MRIT) is implemented in Sysquake. MRIT mainly deals with motion planning. There, several global path planning algorithms and reactive navigation approaches can be selected for different robot kinematics. In [8] a virtual and remote laboratory based on Easy Java Simulations (EJS) is described. The main application of this software is to improve the study of sensors in order to obtain a map. We also remark commercial development environments such as Webots [33] and ANVEL [39]. The first one provides a rapid prototyping environment for modelling, programming and simulating any kind of mobile robot. The second one is particularly devoted to mobile robots moving on off-road conditions.

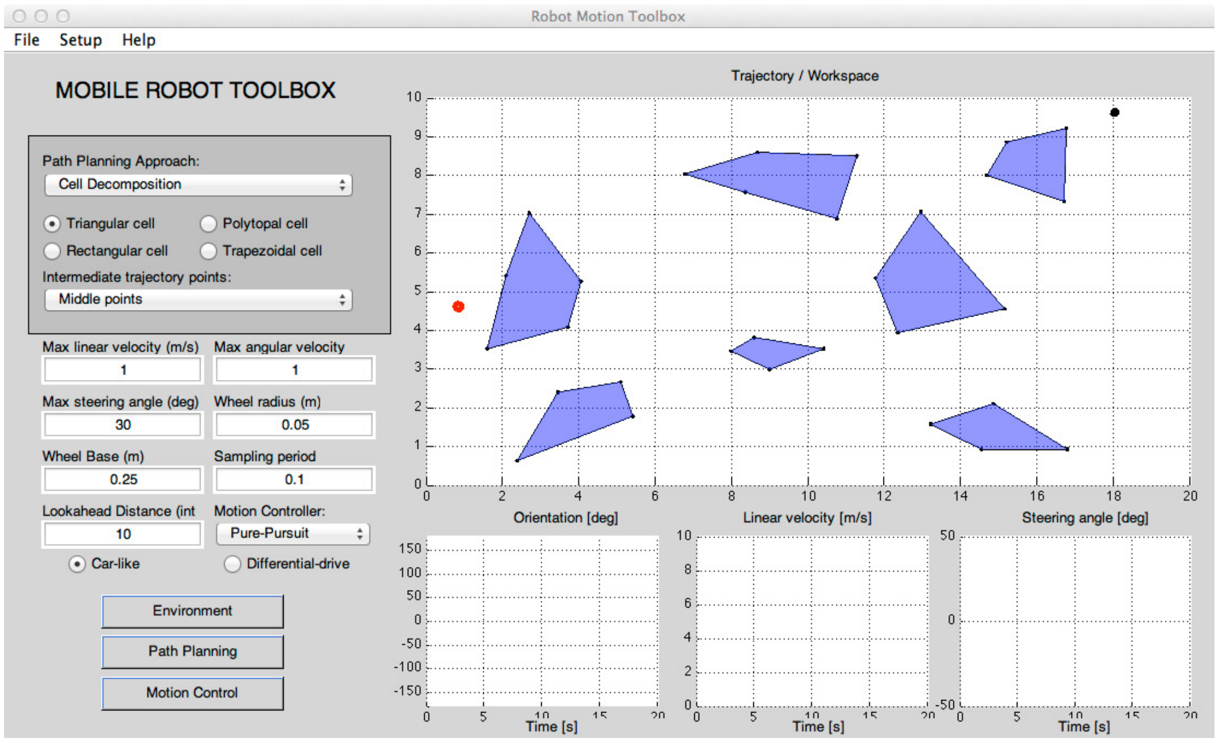
The above-mentioned simulators show some limited options for mobile robotics education, especially in introductory courses: (i) very few packages permit to extend their capabilities by adding new functions (scalability); (ii) some of them are stand-alone applications, so the interested user cannot take advantage of a particular function for his/her own application (closed implementation); (iii) in order to work with some of them, a reasonable knowledge of the programming environment is required (e.g. Matlab, Simulink, C++); (iv) some require a complicated setup before carrying out interesting simulations. It is important to remark that most of the Matlab-based tools rely on running or configuring specific scripts (m-files) and/or typing commands one by one in the Matlab command window. So the feeling of interactivity can be easily lost, while the student becomes bored, especially if one is not familiar with Matlab suite.

This paper presents a Matlab-based interactive software tool for teaching mobile robotics, named RMTool. The ultimate goal of this research is that students focus more on the concepts: robot modeling, path planning, and motion control than on the program implementation (they do not need any previous knowledge on Matlab or programming). The proposed toolbox offers a real-time simulation whatever the tested algorithm. This leads to a high realism in interactivity and students do not have to wait to see the effects of their actions. Additionally, the proposed software simulator comprises the following advantages: (i) scalability: new functions/algorithms can be easily added to our toolbox by just modifying a single file (GUI main file); (ii) visual appearance: we have focused on developing a friendly GUI with just one single window with straightforward and intuitive buttons, icons, and figures; (iii) it is open-source so any interested educator or researcher is welcome to use it, and specific functions (scripts) can be employed outside of this toolbox; (iv) it is implemented in Matlab, which constitutes a widespread and well-known suite in the academic community. Some functionalities of the RMTool are using freely-available tools for polyhedral operations, geometry-related tools and cell decompositions from [18, 19, 26, 35, 31, 41, 42]. These packages are properly integrated in RMTool. The interested reader is cordially invited to download RMTool freely from: `http://webdiis.unizar.es/RMTool/`.

The work is organized as follows. Section 2 details the graphical user interface (GUI) dealing with the simulator proposed in this paper. The path planning algorithms implemented within the robot simulator are reviewed in Section 3. Section 4 includes the considered robot kinematic models, while the motion control algorithms are presented in Section 5. Some illustrative examples demonstrating the benefits of the proposed simulator are discussed in Section 6. Finally, conclusions and future work close the paper in Section 7.

2. General description of the simulator

The Mobile Robot Toolbox (MRTool) presents a friendly Graphical user Interface (GUI) that enables the user to easily insert the input parameters in order to perform the simulations (Figure 1). It has several graphical environments where the results of simulations are displayed. The environment in which the robot is moving can be either defined by the user in a Matlab axes object, or it can be imported from a .mat file. There are several editable text box objects and popup menus where the user can specify various parameters for path planning and motion control. The GUI includes five control panels: *Menu Bar* (1), *Drawing Area (Trajectory / Workspace)* (2), *Path Planning Panel* (3), *Motion Control Panel* (4) and *Simulation Panel* (5). In the following, all these panels are described.



Path Planning Approach: Cell Decomposition

[✓ Triangular cell] Polytopal cell Rectangular cell Trapezoidal cell

Intermediate trajectory points: Middle points

Max linear velocity (m/s)	1
Max angular velocity	1
Max steering angle (deg)	30
Wheel radius (m)	0.05
Wheel Base (m)	0.25
Sampling period	0.1
Lookahead Distance (int	10
Motion Controller:	Pure-Pursuit

[✓ Car-like] Differential-drive

Figure 1: The main window of *RMTool*.

2.1. Menu Bar

The *Menu Bar* panel (placed horizontally, on top of the main window of the RMTool) displays a set of three drop-down menus: *File*, *Setup* and *Help*.

The *File* menu offers facilities for file-handling operations and contains the following two commands: *Open* (permitting to open some previously used data) and *Save* (which saves the current data in a .mat file). The *Setup* menu has three submenus enabling the user to introduce different input parameters. In particular, (1) *Environment limits* submenu allows the user to change the limits of the axes object placed on the top-right part of the GUI containing the representation of the environment, (2) *Robot initial and final position* can be used for changing the start and the goal positions of the robot, and (3) *PI tuning parameters* allows access to parameters of the PI controller (see Section 5.2). Finally, the *Help* menu provides credential information.

2.2. Drawing Area Panel (Trajectory / Workspace)

This panel is composed by four axes objects used to represent the evolution of different control variables and the environment. The main axes object placed on top right side of the main window is used to define the obstacles, to represent the trajectory obtained by path planning algorithms and to display the simulation of the controlled motion. On the bottom part of the main axes there are three smaller axes objects used to represent the control actions after a simulation: orientation, linear velocity and steering angle of the robot.

2.3. Path Planning Panel

It is placed in the left-top side of the GUI and it consists of two popup menus and four radio button objects. By using the first popup menu, the user can select the approach used for path planning. The *RMTool* implements three approaches: cell decomposition, visibility graph and generalized Voronoi diagram (see Section 3). If cell decomposition is selected, the used can chose the type of decomposition by using the radio buttons. Moreover, since in this case the trajectory will be given as a sequence of regions that should be followed by the robot, the second popup menu permits the selection of the intermediate points method computation. The following options are available: middle points (no optimization) or norm 1, 2 or infinity (by solving an optimization problem).

2.4. Motion Control Panel

Placed just below the *Path Planning Panel*, it consists of seven editable box objects, one popup menu and two radio buttons. The editable boxes allow the user to select different parameters of the robot and different parameters for the motion control approach, i.e., maximum linear and angular velocity, maximum steering angle, wheel radius and base, sampling period, lookahead distance (see Section 5. By using the popup menu, the user can choose the path following controller from the following two possibilities: *Pure Pursuit* or *PI control*. The robot type can be selected by pressing one of the radio buttons, the *RMTool* allowing kinematic models for car-like or differential-drive robot (see Section 4).

2.5. Simulation Panel

The *Simulation Panel* consists of three push buttons that are used to start the simulations. The *Environment* button starts the routine for defining new obstacles in the environment. A new Matlab window is opened and the number of new obstacles should be introduced. By using the mouse the user can introduce the obstacle in the main axes object. The *Path Panning* button starts automatically the path planning approach (selected in the *Path Planning Panel*) and draws (in the main axes object of the *Drawing Area Panel*) the trajectory that the robot should follow. Finally, the *Motion Control* button simulates the trajectory following by using the robot model and control parameters from the *Motion Control Panel*. The resulted robot trajectory is displayed in the main axes object of the *Drawing Area Panel*, while graphical representations of control actions are represented in the other axes objects.

3. Path planning algorithms

In this section we focus on path planning for a fully-actuated point mobile robot. Furthermore, we assume that the robot evolves in a 2D environment cluttered with convex polygonal obstacles.

In the context of mobile robots, the path planning problem refers to finding a path (reference trajectory) from a given initial position to a desired goal position. This reference path must avoid any obstacle in the environment. The path optimality generally deals with minimum-length routes, but in some occasions the shortest distance does not guarantee reaching the goal point, for instance, due to terrain unevenness [25], or constrained routes where the robot must pass through some waypoints [32].

Generally, in order to accomplish the path planning objective, a (geometric) map of the environment is required [16, 40]. Depending on the considered geometrical map, two general strategies can be identified [40]:

- Cell decomposition: partition the free environment space. In particular, rectangular, triangular, polytopal, and trapezoidal decompositions are employed in this work (see Section 3.1).

- Road map: identify a set of fast routes within the free space. Visibility graphs and generalized Voronoi diagrams constitute two examples (see Sections 3.2, 3.3).

The geometric map represents a finite abstraction of the environment (either via a set of cells, or a set of fast routes). Once the geometric map is available, planning strategies require some form of graph searching algorithm in order to identify the path from the initial position to the goal one. The toolbox presented in this paper uses Dijkstra's algorithm, which is generally employed in shortest-path graph searches [16]. In situations when a particular heuristic is of interest, optimal route algorithms such as A^* or E^* may be preferred [16, 38], but this is not the case for the currently assumed setup.

3.1. Cell decomposition

Cell decomposition techniques partition the free space (not covered by obstacles) into a set of regions having the same geometrical shape [30, 4]. Our toolbox includes decompositions of environments with polygonal and convex obstacles into cells having triangular, trapezoidal, polytopal or rectangular shape. More details on the software implementation of decomposition routines and supporting algorithms can be found in [26] and the references therein.

A finite graph is constructed by assigning each cell to a node and transitions based on adjacency between cells. The Dijkstra algorithm is run on this finite abstraction of the free space, with the start and goal nodes corresponding to cells that include the initial and final robot positions. The output is a sequence of cells to be followed, and any path belonging to this sequence is valid, since it avoids obstacles.

We construct a piece-wise linear robot trajectory by linking the start point with a point on the line segment shared by first and second cells, and so on until the last cell. The points on the line segments shared by two successive cells can be chosen to be the middle point of the segment, or can be computed via an optimization problem. The convex optimization problem minimizes a sum of norms of all linear segments of the obtained path, and our toolbox supports norm one, two or infinity. Although the obtained path is not optimal from the point of view of the distance traveled by the robot, the cell decomposition methods provide advantages that can be further used in more complicated planning algorithms [3, 27].

Due to space constraints, formal descriptions of the involved optimization problems are omitted here, and they will constitute the focus of another study. Illustrative examples showing the results of these procedures are included in Section 6.

3.2. Visibility graph

The well-known approach Visibility graph planning or V-graph planning solves a minimum-length path by following a sequence of edges (straight lines). These line segments connect a set of obstacle vertices visible one from the other, as well as the start and goal points [40, 9]. The V-graph has nodes corresponding to vertices and edges corresponding to line segments, while each arc cost is given by the traveled distance along each line segment. The Dijkstra's graph search algorithm yields the shortest path from the start to the goal, but the reference trajectory in close to obstacles, since it contains some obstacle vertices.

Notice that we have assumed that the robot is represented by a point and it is capable of omnidirectional motion. One mechanism for compensating this assumption is to consider that the robot has a circular cross section and to *dilate* all obstacles in the environment by an amount equal to the robot's radius [29, 16]. We have followed this approach in our simulator, and hence, polygonal objects are dilated by the robot's radius.

3.3. Generalized Voronoi diagram

Contrary to the V-graph planner, the algorithm based on the Generalized Voronoi Diagram (GVD) tends to maximize the distance between the robot and obstacles in the map. In particular, for each point in the free space, the distance to the nearest obstacle is computed. The GVD contains only equidistant points from two or more closest obstacles, and this leads to straight and parabolic segments (when polygonal obstacles are considered) [9, 30]. The GVD has the useful property of maximizing the distance between the robot and the obstacles [16]. As in previous case, once the graph with the set of paths maximizing the clearance between points and obstacles is calculated, the graph search algorithm returns the reference route. The resulted robot's path accesses the GVD from the initial position, follows some of its segments and leaves the GVD near the goal point.

In the next sections we will apply the path planning result to more complex kinematic models for mobile robots. Note that in some situations errors may appear, since the path was designed for a simple (fully-actuated point robot) and the actual robot may go off the reference trajectory, thus leaving the environment or hitting an obstacle. However, designing a reference trajectory for a specific robot would require different tailored planning algorithms, which in general are more complex and conservative [10, 17, 19]. Such modified planning procedures would not be appropriate for the introductory and educational purpose of RMTool. Our toolbox signals the situations when the considered robot cannot follow close enough the prescribed trajectory.

4. Robot kinematic models

In the context of robotics, a model is defined as a set of mathematical differential equations that represents the behavior of a robot. In this sense, kinematic and dynamic modelling constitutes a key issue that can be employed for estimating robot location and implementing software simulators, among other issues. In this work, two kinematic models are considered in order to simulate the motion of a car-like robot and a differential-drive robot. Furthermore, such models will be employed for estimating the robot position at each sampling instant (integrated from the initial robot position). This position will be fed back to the selected motion controller in order to ensure path following. Notice that car-like and differential-drive models have been selected because they are the most common configurations in wheeled/tracked mobile robots [16, 22, 40].

4.1. Car-like robot

Four-wheeled car-like robots (Ackerman vehicles) are the most popular ones [40], and this configuration was the first choice to include in RMTool. In general, such a robot is modeled in terms of a bicycle model. The bicycle has a rear wheel fixed to the body and the plane of the front wheel rotates about the vertical axis to steer the vehicle (Figure 2a) [12]. The vehicle is assumed to only move forward, and the wheels cannot slip sideways.

The car-like configuration leads to circular paths when the vehicle turns, these circular arcs are centered at a point known as the Instantaneous Centre of Rotation (ICR). Therefore, the angular velocity is given by

$$\dot{\theta} = \frac{v}{R_1}, \qquad (1)$$

where θ is the vehicle orientation with respect to the x-axis, v is the forward speed of the robot (imposed by the real wheels). R_1 is the turning radius, which is defined by $R_1 = b_l/\tan\alpha$ where $b_l \in \mathbb{R}^+$ is the length of the vehicle or wheel base. The steering angle, $\alpha \in \mathbb{R}$, is mechanically constrained and its maximum value influences robot motion. Notice that this important feature can be analyzed with the software tool proposed in this work because the user can change the maximum angle that the steering wheel can turn (see Section 6).

Based on Figure 2a, one can write the following equations dealing with the car-like robot kinematics [30]

$$\begin{aligned} \dot{x} &= v\cos\theta, \\ \dot{y} &= v\sin\theta, \\ \dot{\theta} &= \frac{v}{b_l}\tan\alpha, \end{aligned} \tag{2}$$

where $[x \quad y \quad \theta]^T \in \mathbb{R}^3$ represents the pose (position and orientation) of the mobile robot.

4.2. Differential-drive robot

Perhaps the second most used configuration in mobile robotics is the differential-drive one. In fact, this is the easiest mechanical configuration because only two parallel driving wheels mounted on an axis are enough to move the robot (Figure 2b). In particular, forward spin of right wheel (v_r) and stopped left wheel ($v_l = 0$) result in counterclockwise rotation around the point where left wheel touches the ground, while equal wheel speeds result in straight motion. Different combinations of v_r and v_l yield instantaneous rotations around points on the line segment linking the contact points between wheels and ground surface.

As known, in absence of wheel slip, the linear velocity of the wheels is given by [40]

$$\begin{aligned} v_r(t) &= \rho\omega_r(t), \\ v_l(t) &= \rho\omega_l(t), \end{aligned} \tag{3}$$

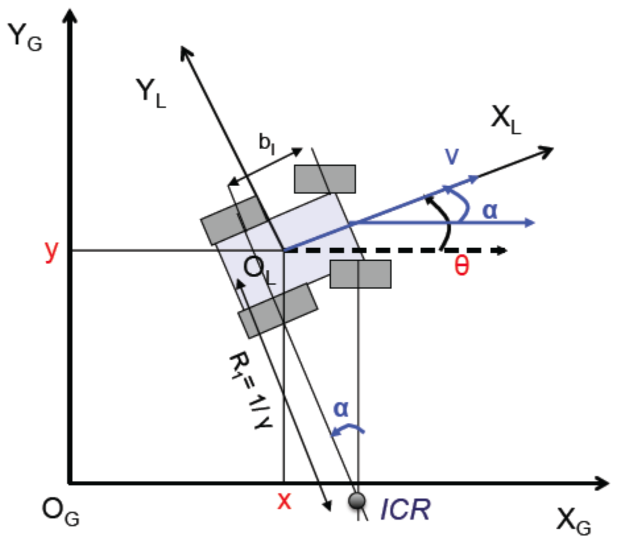
where $\omega_r, \omega_l \in \mathbb{R}$ are the angular velocities of the right and left wheels, respectively, and ρ is the wheel radius. In this way, the kinematic model for a differential-drive mobile robot is given by [6], [40]

$$\begin{aligned} \dot{x} &= \frac{v_r + v_l}{2}\cos\theta, \\ \dot{y} &= \frac{v_r + v_l}{2}\sin\theta, \\ \dot{\theta} &= \frac{v_l - v_r}{b_w}, \end{aligned} \tag{4}$$

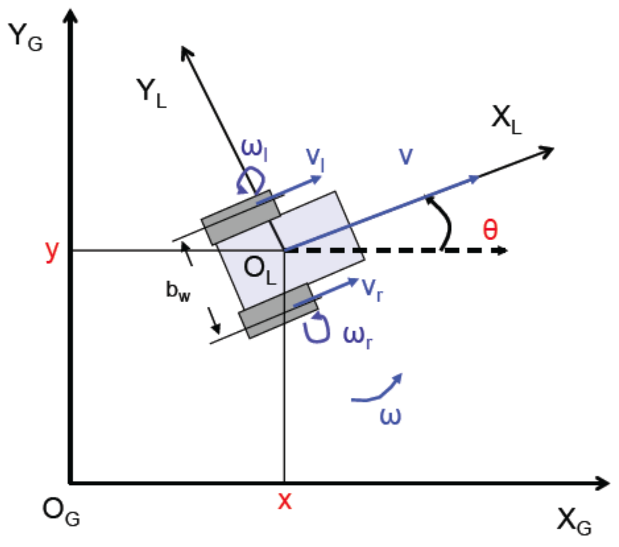
where $b_w \in \mathbb{R}^+$ is the distance between wheel centers.

5. Motion control algorithms

Mobile robots must have effective motion controllers that generate proper control actions to successfully steer the robot such that it follows the reference trajectory given by the path planner. Closed-loop



(a) Car-like robot (control inputs: linear velocity, v and steering wheel angle, α)



(b) Differential-drive robot (control inputs: angular wheel velocities ω_r and ω_l)

Figure 2: Robot configurations considered in the RMTool software.

motion controllers use the current robot position to make new decisions (control inputs) that will eventually drive the robot to a desired goal or waypoint [21, 22, 40, 43]. Notice that accurate robot localization plays a key role in the performance of the motion controller. In particular, in this work we are interested in the path following problem. In this case, a reference point on the robot must follow a path in the Cartesian space $(x^{ref}(t), y^{ref}(t))$ starting from a given initial configuration. This path may come from a sequence of coordinates generated by the path planner, as in Section 3. Examples of trajectory following approaches are the Pure Pursuit algorithm [1], reviewed in Subsection 5.1, and a form of Proportional Integral (PI) controller introduced in [12] and explained in Subsection 5.2.

As previously mentioned, a robot with a specific kinematic model as in Section 4 may sometimes not be able to closely follow a reference path designed for a generic robot as in Section 3. If the simulated trajectory leaves the environment or hits an obstacle, the user is informed. Such situations serve the educational purpose of the RMTool, since a student easier understands the errors that may appear when the planning procedure does not account a specific robot.

5.1. Pure Pursuit algorithm

The Pure Pursuit is a well-known strategy for the mobile robotics community. It was formulated in the framework of the CMU Navlab project [1]. The idea behind the Pure Pursuit control law is based on repeatedly fitting different circular arcs (geometrical approach) to different waypoints as the robot moves forward until the final goal point is reached. These waypoints are obtained as a constant lookahead distance away from the closest point between the current robot position and the reference path (Figure 3). The output of the controller is the desired steering angle of the driving wheel of the robot (in case of car-like robot) or the wheel velocities (for differential-drive robots).

From Figure 3, the following mathematical equations hold

$$\begin{aligned} r &= \Delta x + d, & (5)\\ r^2 &= d^2 + (\Delta y)^2, & (6) \end{aligned}$$

solving for the turning radius, r,

$$r^2 = (r - \Delta x)^2 + (\Delta y)^2, \tag{7}$$

leads to

$$r = \frac{(\Delta x)^2 + (\Delta y)^2}{2\Delta x}. \tag{8}$$

Finally, the curvature that the robot must follow is

$$\gamma = \frac{1}{r} = -\frac{2\Delta x}{L^2}. \tag{9}$$

Notice that the Pure Pursuit is a proportional controller of the steering angle using the x-displacement as the error and $\frac{2}{L^2}$ as the gain. In this sense, tuning the lookahead distance, L, plays a major role in the performance of the controller. In Section 6 some examples highlight the significance of the lookahead distance.

5.2. PI controller

For comparison purposes, we have also selected the path following algorithm implemented in [12], where a Proportional Integral (PI) controller is employed for controlling the robot position. Again, the

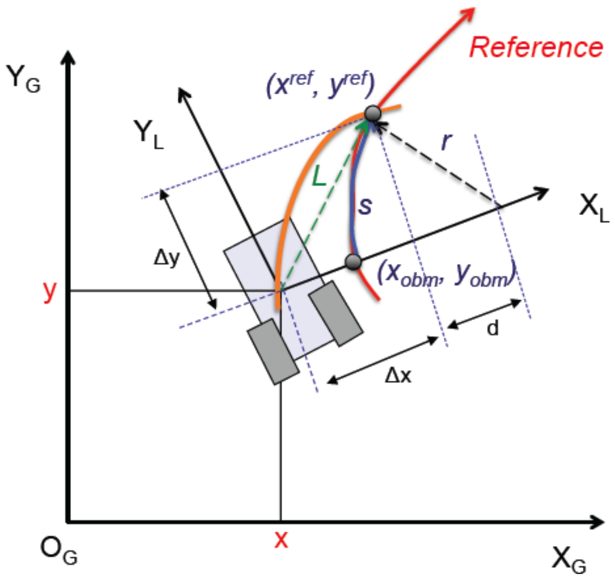


Figure 3: Pure Pursuit strategy

robot maintains a distance behind a given waypoint. First, the Euclidean distance between the current robot position and the desired waypoint is calculated as

$$e_d \quad = \quad \sqrt{(x^{ref} - x)^2 + (y^{ref} - y)^2} - d^*. \tag{10}$$

The goal is to regulate (10) to zero by controlling the robot's velocity using a PI controller

$$v^c = K_v e_d + K_i \int e_d dt, \tag{11}$$

where v^c is the control input dealing with the linear velocity of the robot. Notice that the integral term avoids an offset error when a constant error appears. On the other hand, a second controller steers the robot towards the target, which is at the relative angle

$$e_\theta = \tan^{-1}\left(\frac{y^{ref} - y}{x^{ref} - x}\right). \tag{12}$$

Then a simple proportional controller turns the steering wheel of a car-like robot for driving the robot towards the target, that is,

$$\alpha^c = K_h(e_\theta \ominus \theta), \ K_h > 0. \tag{13}$$

It is important to point out that the performance of the PI controller depends highly on the values of the gains K_v, K_i, K_h (tuning). In this sense, this software tool permits to adjust such values, and hence, the user can understand in a better way how they affect the trajectory following.

Notice that the main difference between Pure Pursuit and this PI-based controller is that the latter control approach produces the control action as for the linear velocity as for the steering wheel. On the contrary, for the Pure Pursuit approach the linear velocity of the robot is manually fixed and does not change along the experiment. As shown in Section 6, this software tool allows to evaluate the performance of fixing the linear velocity of the robot (Pure Pursuit) against being able to change that velocity depending on the Euclidean error (PI controller).

6. Illustrative Examples

This section discusses some examples that demonstrate the main capabilities of RMTool. Furthermore, in order to check two key advantages of this software tool, that is, interactivity and simple GUI, the reader is cordially invited to visit the website `http://webdiis.unizar.es/RMTool/` to experience these features.

6.1. Examples about path planning aspects

The examples shown in Figure 4 addresses all the steps related to the application of the path planning algorithms implemented in this software: cell decomposition, visibility graph, and generalized Voronoi diagram. The minimum length trajectory is shown in red color, the actual path followed by the mobile robot is in black, obstacles are in blue, and the grey segments represents the different cells in the cell decomposition approach, the different segments connecting the edges in case of V-graph, and the equidistant routes in case of Voronoi. The tracking of this trajectory is performed using the Pure Pursuit (PP) method, except in Figure 4c where Proportional Integral (PI) controller was employed. In these experiments the robot always starts with orientation $\theta = 0$.

Figures 4a, b deals with cell decomposition using two different approaches to estimate the reference path: triangular versus polytopal cell decomposition. RMTool allows us to observe a possible outcome of the polytopal approach, where the returned trajectory is fairly close to obstacles. Notice that RMTool detects this situation and remarks it to the user via a popup window.

Figures 4c, d are related to the visibility graph approach. The V-graph is constructed by dilating the obstacles, such that it is less likely to hit an obstacle while a motion controller is used for following the angular reference path. This aspect can be observed at the beginning of the trajectory using the PI controller (Figure 4c). Another interesting conclusion can be extracted from these figures. Notice that in the first experiment the car-like model is used, where the steering angle was constrained to 15 [^{o}]. In the second example, where a differential-drive robot is used, the steering angle does not influence the robot motion because the differential-drive configuration is not affected by a steering mechanism (recall that differential-drive mechanism achieves robot turning by changing the relative velocity of the driving wheels).

Finally, the generalized Voronoi diagram is employed in Figures 4e,f. In particular, with RMTool we can observe the importance of a parameter dealing with the interpolation between two consecutive points, that is, the parameter ϵ. Smaller the value of Epsilon smoother the output will be but more computation time it will take. Notice that in the first experiment a high value leads to unconnected segments, since not enough points have been considered. On the contrary, Figure 4f shows a continuous trajectory that is safely followed by the robot, while in Figure 4e the outcome of trajectory following was unsatisfactory.

6.2. Examples about motion control aspects

The examples shown in this section allow us to easily and visually understand the main properties of motion control and the importance of tuning properly the controller. For that reason, we focus mainly on the lower part of the GUI, that is, the plots dealing with orientation, linear velocity, and steering angle.

Thanks to Figures 5a,b,c we may understand the importance of the lookahead parameter in the pure pursuit approach. In particular, a higher value ($L = 10$) leads to a smoother trajectory than a smaller value ($L = 2$). This is especially noticeable in the orientation and steering wheel angle plots. Notice the oscillatory behavior after 10 seconds in the steering wheel angle in Figure 5b. This fact is also observed in Figure 5c. In this case, sudden changes in the robot orientation are noticed from the beginning of the experiment. Such sudden changes are produced by the incorrect control actions (see the plot dealing with steering wheel angle).

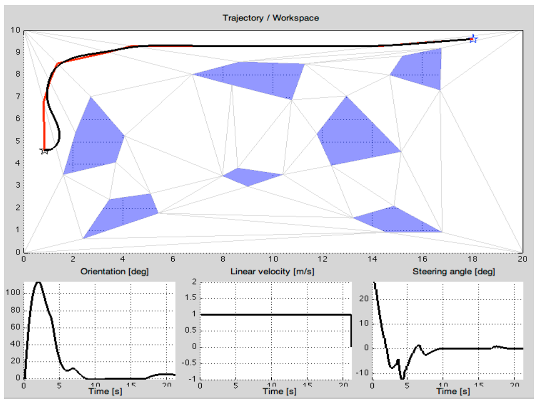
Figure 5d addresses the performance of the PI controller versus PP approach that was used for following the same reference path in Figure5b. RMTool permits to understand the main difference between these controllers that is in case of PP the linear velocity is always fixed no matter the circumstances. In case of the PI approach linear velocity depends on the longitudinal error, and hence, it varies along time. The importance of this fixed linear velocity in the pure pursuit approach is clearly shown in Figures 5e,f. Observe that in the first plot, with a fixed linear velocity of 1 [m/s], the robot follows closely the reference. On the contrary, when the linear velocity is increased to 2 [m/s], the robot cannot reach the goal point.

7. Conclusions and future work

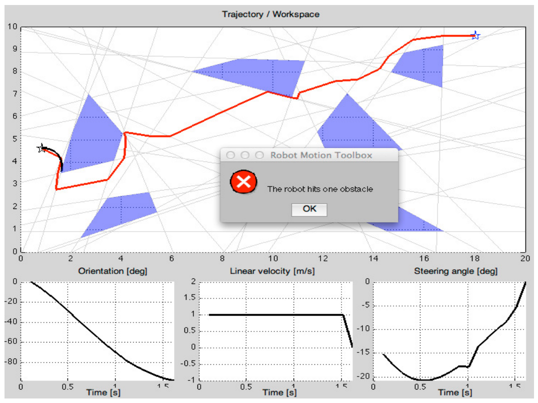
In our personal experience of teaching robotics, the use of software simulators during theory lessons constitutes a key element. Even before application hours, it constitutes an excellent idea in order to quickly review theoretical concepts. This work shows an interactive tool aimed at helping professors and students to understand basic concepts dealing with "wheeled" mobile robots. Among other issues, RMTool permits to clearly understand the importance of path planning and motion control. As shown through some examples the choice of a path planner can lead to trajectories closer to obstacles, shorter paths (V-graph), or longer paths but with larger distance from obstacles (Voronoi diagram).

With the proposed software simulator, a professor can easier illustrate the influence of a bad tuned motion controller. As shown with the pure pursuit approach, the values of the lookahead distance and the fixed linear velocity can either lead to closely following the reference path, or can cause undesirable outcomes.

Future efforts will be focused on several directions. An envisioned extension is to include algorithms that deal with planning and controlling a team of mobile robots, rather than a single agent. Also, we aim to embed in RMTool various approaches that enable mobile robots to execute much richer specifications than obstacle avoidance, e.g. as linear temporal tasks over regions of interest. Related to robot models and motion controllers, we aim to consider more complicated models, e.g. including slipping effects.

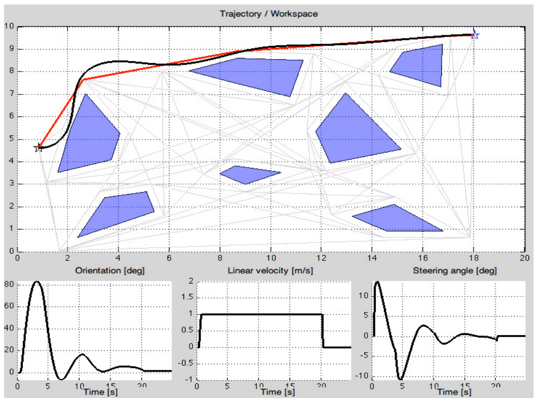


(a) Cell decomp. (triangular cell and middle points), car-like, PP ($L = 10$)

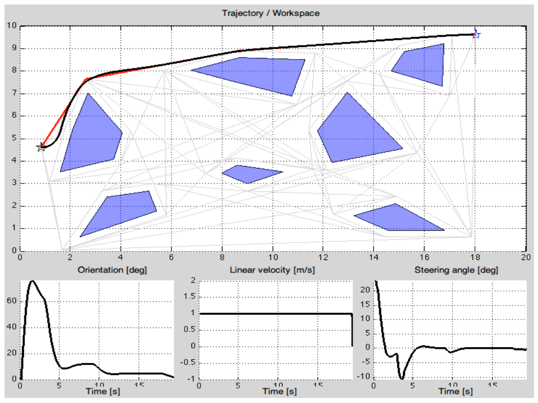


The robot hits one obstacle

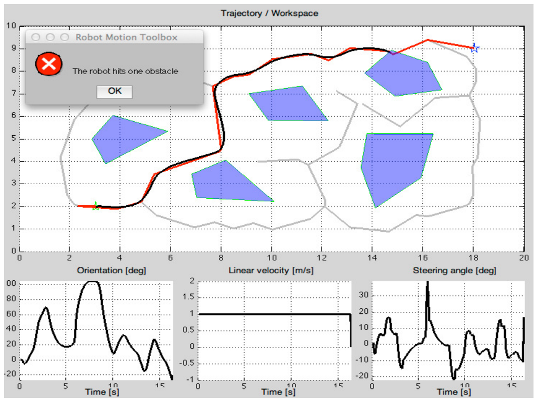
(b) Cell decomp. (polytopal cell and middle points), diff-drive, PP ($L = 10$)



(c) V-graph, car-like, PI ($\alpha^{max} = 15^o$)

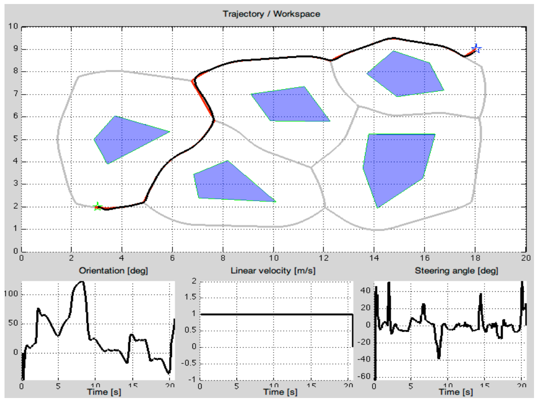


(d) V-graph, diff-drive, PP ($L = 10, \alpha^{max} = 15^o$)



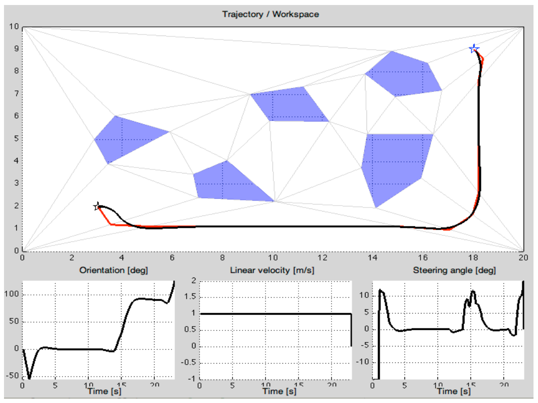
The robot hits one obstacle

(e) Voronoi ($\epsilon = 1.5$), diff-drive, PP ($L = 10$)

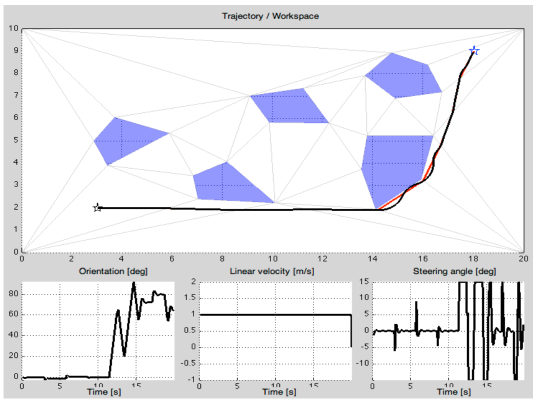


(f) Voronoi ($\epsilon = 0.2$), diff-drive, PP ($L = 10$)

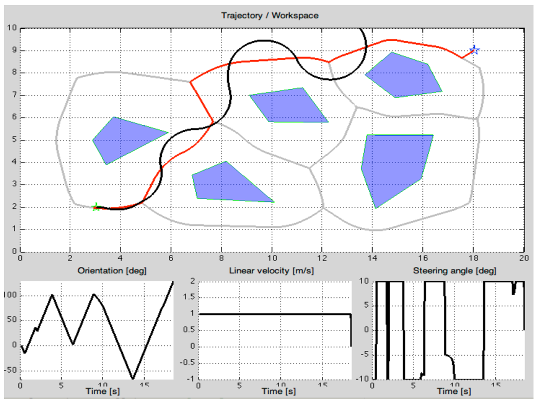
Figure 4: Examples related to path planning aspects.



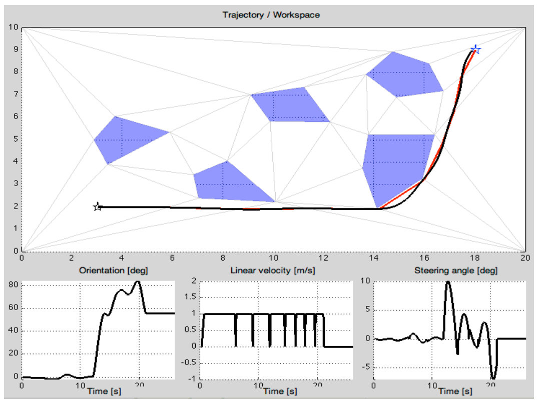
(a) Cell decomp. (tri. cell and mid. points), car-like, PP ($L = 10, \alpha^{max} = 15^o$)



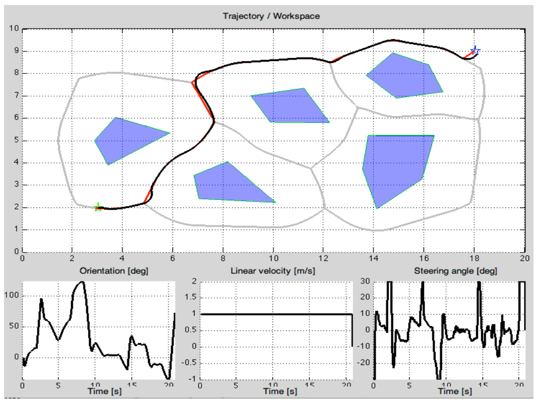
(b) Cell decomp. (tri. cell and norm-2), car-like, PP ($L = 2, \alpha^{max} = 15^o$)



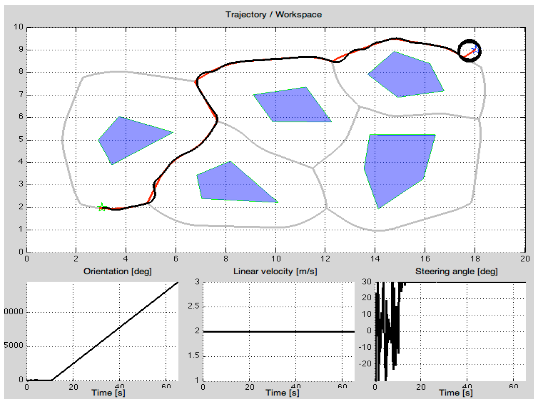
(c) Voronoi, car-like, PP ($L = 2, \alpha^{max} = 10^o$)



(d) Cell dec. (tri. cell and norm-2), car-like, PI ($\alpha^{max} = 15^o$)



(e) Voronoi, car-like, PP ($L = 10, \alpha^{max} = 30^o, v^{max} = 1$)



(f) Voronoi, car-like, PP ($L = 10, \alpha^{max} = 30^o, v^{max} = 2$)

Figure 5: Examples related to motion control aspects: performance of the Pure Pursuit (PP) and Proportional Integral (PI) controllers in terms of maximum steering angle (α^{max}), maximum linear velocity (v^{max}), and lookahead distance (L).

References

[1] O. Amidi. Integrated mobile robot control. Technical Report CMU–RI–TR–90–17, Robotics Institute, Carnegie Mellon University, USA, 1990.

[2] K. O. Arras. *The CAS Robot Navigation Toolbox: Users Guide and Reference.* Center for Autonomous Systems, KTH, September 2004.

[3] C. Belta, A. Bicchi, M. Egerstedt, E. Frazzoli, E. Klavins, and G. J. Pappas. Symbolic planning and control of robot motion. *IEEE Robotics and Automation Magazine, special issue on grand challenges for robotics*, 14(1):61–71, 2007.

[4] M. De Berg, O. Cheong, and M. van Kreveld. *Computational Geometry: Algorithms and Applications.* Springer, 3rd edition, 2008.

[5] C. Buiu. Hybrid Educational Strategy for a Laboratory Course on Cognitive Robotics. *IEEE Transactions on Education*, 51(1):100–107, 2008.

[6] G. Campion, G. Bastin, and B. D‘ Andréa-Novel. Structural Properties and Classification of Kinematic and Dynamic Models of Wheeled Mobile Robots. *IEEE Transactions on Robotics and Automation*, 12(1):47–62, 1996.

[7] S. Carpin, M. Lewis, W. Jijun, and S. Balakirsky. USARSim: A Robot Simulator for Research and Education. In *IEEE Int. Conf. on Robotics and Automation (ICRA)*, pages 1400 – 1405. IEEE, April 10-14 2007.

[8] D. Chaos, J. Chacon, J.A. Lopez-Orozco, and S. Dormido. Virtual and Remote Robotic Laboratory Using EJS, Matlab and LabVIEW. *Sensors*, 13(2):2595 – 2612, 2013.

[9] H. Choset, K. M. Lynch, S. Hutchinson, G. Kantor, W. Burgard, L. E. Kavraki, and S. Thrun. *Principles of Robot Motion: Theory, Algorithms, and Implementations.* MIT Press, Boston, 2005.

[10] D.C. Conner, A.A. Rizzi, and H. Choset. Integrated planning and control for convex-bodied nonholonomic systems using local feedback control policies. In *Proceedings of Robotics: Science and Systems*, 2006.

[11] P. Corke. A Robotics Toolbox for Matlab. *IEEE Robotics & Automation Magazine*, 3(1):24 – 32, 1996.

[12] P. Corke. *Robotics, Vision and Control. Fundamental algorithms in Matlab.* Springer Tracts in Advanced Robotics. Springer, Germany, 2011.

[13] M.S. Couceiro. MRSim - Multi-Robot Simulator. `http://www2.isr.uc.pt/~micaelcouceiro/media/help/helpMRSim.htm`, 2012.

[14] A. Cruz-Martin, J.A. Fernandez-Madrigal, C. Galindo, J. Gonzalez-Jimenez, C. Stockmans-Daou, and J.L. Blanco-Claraco. A Lego Mindstorms NXT approach for teaching at data acquisition, control systems engineering and real-time systems undergraduate courses. *Computers & Education*, DOI: 10.1016/j.compedu.2012.03.026:1–49, 2012.

[15] S. Dormido. Control Learning: Present and Future. *Annual Reviews in Control*, 28(1):115–136, 2004.

[16] G. Dudek and M. Jenkin. *Computational Principles of Mobile Robotics.* Cambridge University Press, United Kingdom, second edition, 2010.

[17] G. E. Fainekos, A. Girard, H. Kress-Gazit, and G. J. Pappas. Temporal logic motion planning for dynamic robots. *Automatica*, 45(2):343–352, 2009.

[18] K. Fukuda. CDD/CDD+ package. `http://www.inf.ethz.ch/personal/fukudak/cdd_home/`.

[19] N. Ghita and M. Kloetzer. Trajectory planning for a car-like robot by environment abstraction. *Robotics and Autonomous Systems*, 60(4):609–619, 2012.

[20] J.M. Gómez de Gabriel, A. Mandow, J. Fernandez-Lozano, and A.J. Garcia-Cerezo. Using Lego NXT Mobile Robots with Labview for Undergraduate Courses on Mechatronics. *IEEE Transactions on Education*, 54(1):41–47, 2011.

[21] R. González, M. Fiacchini, J. L. Guzmán, T. Álamo, F. Rodríguez. Robust Tube-based Predictive Control for Mobile Robots in Off-Road Conditions. *Robotics and Autonomous Systems*, 59(10):711–726, 2011.

[22] R. González, F. Rodríguez, and J. L. Guzmán. *Autonomous Tracked Robots in Planar Off-Road Conditions. Modelling, Localization and Motion Control.* Series: Studies in Systems, Decision and Control. Springer, Germany, 2014.

[23] J. L. Guzmán, M. Berenguel, F. Rodríguez and S. Dormido. An Interactive Tool for Mobile Robot Motion Planning. *Robotics and Autonomous Systems*, 56(5):396–409, 2008.

[24] J. Hrabec. Autonomous Mobile Robotics Toolbox SIMROBOT. Master's thesis, Department of Control, Measurement and Instrumentation. Brno University of Technology, Brno, Czech Republic, 2001.

[25] K. Iagnemma and S. Dubowsky. *Mobile Robots in Rough Terrain. Estimation, Motion Planning, and Control with Application to Planetary Rovers.* Springer Tracts in Advanced Robotics. Springer, Germany, 2004.

[26] M. Kloetzer and N. Ghita. Software tool for constructing cell decompositions. In *IEEE Conference on Automation Science and Engineering*, pages 507–512, 2011.

[27] M. Kloetzer and C. Mahulea. A Petri net based approach for multi-robot path planning. *Discrete Event Dynamic Systems*, 2013. in press.

[28] A.N. Kumar. Three years of using robots in an artificial intelligence course - lessons learned. *ACM Journal on Educational Resources in Computing*, 4(3):1–15, 2004.

[29] J. C. Latombe. *Robot Motion Planning.* Kluger Academic Pub., 1991.

[30] S. M. LaValle. *Planning Algorithms.* Cambridge, 2006. Available at `http://planning.cs.uiuc.edu`.

[31] D. Legland. *Geometry Library for Matlab (MatGeom).* 2014. Available at `http://matgeom.sourceforge.net`.

[32] M. Maimone, J. Biesiadecki, E. Tunstel, Y. Cheng, and C. Leger. Surface navigation and mobility intelligence on the Mars Exploration Rovers. In *Intelligence for Space Robotics*, pages 45–69. TSI Press, 2006.

[33] O. Michel. Webots: Professional Mobile Robot Simulation. *International Journal of Advanced Robotic Systems*, 1(1):39 – 42, 2004.

[34] D.P. Miller, I.R. Nourbakhsh, and R. Siegwart. Robots for Education. In *Handbook of Robotics*, pages 1283–1301. Springer, 2008.

[35] A. Makhorin. GNU linear programming kit. `http://www.gnu.org/software/glpk/`, 2012.

[36] D. Obdrzalek and A. Gottscheber, editors. *Research and Education in Robotics - EUROBOT 2011*, volume 161 of *Communications in Computer and Information Science*. Springer, Germany, 2011.

[37] T. Petrinic, E. Ivanjko, and I. Petrovic. AMORsim - A Mobile Robot Simulator for Matlab. In *Int. Workshop on Robotics in Alpe-Adria-Danube Ragion*, Balatonfred, Hungary, June 15-17 2006.

[38] R. Philippsen. *Motion Planning and Obstacle Avoidance for Mobile Robots in Highly Cluttered Dynamic Environments.* PhD Thesis, École Polytechnique Fédérale de Lausanne, Lausanne, Switzerland, 2004.

[39] Quantum Signal, LLC. ANVEL Simulator. `http://anvelsim.com`, 2014.

[40] R. Siegwart and I. Nourbakhsh. *Introduction to Autonomous Mobile Robots.* A Bradford book. The MIT Press, USA, First edition, 2004.

[41] P. B. Solanki, G.H.V. Reddy and A. Mukerjee. *Voronoi Diagram based Roadmap Motion Planning.* Tech. Report. IIT Kanpur Institue, India, 2013.

[42] F. Torrisi and M. Baotic. Matlab interface for the CDD solver. `http://control.ee.ethz.ch/~hybrid/cdd.php`.

[43] S. G. Tzafestas. *Introduction to Mobile Robot Control.* Elsevier, 2013.

[44] L. Zlajpah. Simulation in Robotics. *Mathematics and Computers in Simulation*, 79(4):879 – 897, 2008.