

A New Mobile Robot Toolbox for Matlab

Suat Karakaya · Gurkan Kucukyildiz ·
Hasan Ocak

Received: 23 March 2016 / Accepted: 11 January 2017 / Published online: 28 January 2017
© Springer Science+Business Media Dordrecht 2017

Abstract In this study, a wheeled mobile robot navigation toolbox for Matlab is presented. The toolbox includes algorithms for 3D map design, static and dynamic path planning, point stabilization, localization, gap detection and collision avoidance. One can use the toolbox as a test platform for developing custom mobile robot navigation algorithms. The toolbox allows users to insert/remove obstacles to/from the robot's workspace, upload/save a customized map and configure simulation parameters such as robot size, virtual sensor position, Kalman filter parameters for localization, speed controller and collision avoidance settings. It is possible to simulate data from a virtual laser imaging detection and ranging (LIDAR) sensor providing a map of the mobile robot's immediate surroundings. Differential drive forward kinematic equations and extended Kalman filter (EKF) based localization scheme is used to determine where the robot will be located at each simulation step. The LIDAR data and the navigation process are visualized on the developed virtual reality interface. During the navigation of the robot, gap detection, dynamic path planning, collision

avoidance and point stabilization procedures are implemented. Simulation results prove the efficacy of the algorithms implemented in the toolbox.

Keywords Autonomous mobile robots · Navigation · Collision avoidance · Gap detection · Point stabilization

1 Introduction

Mobile robots are finding increasing use in many areas such as education [1–4], military, disaster recovery, home cleaning, advertisement, assistance for people with disability, exploration and transfer of goods [5–7]. Thus, “mobile robot systems” is still a popular topic among students, robotic researchers and educators around the world. Along with increasing interest in mobile robots, simulation tools are becoming more important. A simulation platform is an extremely useful tool for mobile robot manufacturers and developers, allowing them to visualize outcomes of algorithms, robot motion characteristics and environmental effects. Validation of navigation techniques in a physical environment without any simulation is challenging for many applications. Modelling a mobile robot and executing algorithms before physical implementation supply a way to perform feasible and experienced applications.

In mobile robotics, the navigation problem includes several sub-tasks such as path planning [6, 8, 9], and

H. Ocak (✉) · S. Karakaya · G. Kucukyildiz
Mechatronics Engineering Department, Umuttepe Campus,
Kocaeli University, 41380, Kocaeli, Turkey
e-mail: hocak@kocaeli.edu.tr

S. Karakaya
e-mail: suat.karakaya@kocaeli.edu.tr

G. Kucukyildiz
e-mail: gurkan.kucukyildiz@kocaeli.edu.tr

tracking [7, 10], localization [11], collision avoidance [12] and point stabilization [13–16]. These tasks can be considered as independent modules working collectively. The developed toolbox addresses the navigation problem and handles these sub tasks in a multi-function form. Each module of the toolbox is defined as an independent function performing a specified sub task.

Matlab offers a powerful programming environment for developers with wide range of toolboxes. That is why it is widely used for research to prototype algorithms. The functionality of Matlab lets users to extend the proposed toolbox easily by adding new functions or modifying the existing ones. Several Matlab based robot tools are presented in the literature [5, 17, 18]. Although these schemes give satisfactory results in the simulation environment, physical implementation is not feasible since kinematic constraints are not considered. The MATLAB toolbox presented in [4] offers to generate kinematic and dynamic equations required for robot control design. It also generates code for these equations compatible with commonly used programming languages. However, the toolbox is not a complete simulator. Many mobile robot simulators have been developed such as YARS [19], XPERSim [20] and V-REP [21]. These tools provide users an environment to develop algorithms for wheeled mobile robots. There is no absolute superiority between the simulation tools. Advantages and disadvantages of each simulation environment vary depending on the user experience and view.

Several model-driven simulators (e.g. Microsoft Robotics Developer Studio, EasyLab [22]) are released to overcome challenges in the coding phase by using generalized blocks and components to foster reuse. Model and component based approaches cater ease of use but less flexibility [23]. RoboSim, which is a virtual environment for the two-wheeled robots Linkbot and Mobot, is proposed in [24]. The toolkit has a C/C++ controlled basis, which enables to program and control physical robots. Corke [17] has developed a robot toolbox for Matlab. The author focuses on simplicity and user friendliness of the interface by illustrating the mobile robot as a simple triangle and obstacles as circles. Thus, the developed algorithms cannot be implemented in real-time. The proposed toolbox in [5] focuses on generating trajectories for rigid bodies numerically without offering a user interface. One of the most commonly used mobile

robot simulator is Gazebo [25]. Although Gazebo is a powerful tool, it requires robot operating system (ROS) installation as the control interface for the robot. Besides not having a Windows support, installation of Gazebo and ROS, creating the bridge between them and launching Gazebo within ROS is a challenging task and the environment requires a relatively high programming experience. Matlab Robotics System Toolbox released in 2016 provides a powerful environment for mobile robotics applications. Virtual mobile robot examples for Gazebo environment and ROS-enabled mobile robot programming interface are available on the toolbox. Unlike Matlab Robotics System Toolbox, the proposed toolbox provides an all in one simulation environment. An external installation is not required for 3D visualization or code development. In other words, the developed toolbox is a standalone application developed in MATLAB, which is a powerful research tool with wide range of toolboxes most researchers are familiar with. The toolbox offers visual elements such as a user panel, map design tools, 3D obstacle library, a laser imaging detection and ranging (LIDAR) simulator, a 3D navigation monitor, zoom and pan tools. The functionality of Matlab allows users to modify or replace existing algorithms with custom ones. The simulated mobile robot is consistent with differential drive mobile robot kinematics. Thus, prototyped navigation and control algorithms can easily be implemented in real time. In addition, the toolbox presented in this study offers a robust collision avoidance procedure. One can easily integrate custom localization model, kinematics, mobile robot parameters, path planning and point stabilization schemes to the toolbox.

We have used the technique presented in [26] as the default dynamic path planner. The default static path planning method is based on Euclidian distance transform (EDT) [8]. The localization scheme is based on extended Kalman filtering [27] and differential drive mobile robot kinematics [28, 29]. We have integrated these approaches to a LIDAR simulator, which is used to visualize the robot's environment, track changes in the robot's workspace and detect passable gaps. It is possible to change location, scanning resolution, view angle and monitoring options of the LIDAR sensor on the simulated robot.

The following sections highlight graphical user interface (GUI), main functions, the navigation algorithm, simulation results and conclusions.

2 Graphical User Interface (GUI)

The GUI consists of a virtual reality interface (VRI) and a LIDAR monitor (LM). Definitions of the icons and color-codes used in the simulation are given in Table 1.

The main virtual reality window provides an interface through which users can reach all controls, construct a map by inserting and deleting obstacles from an obstacle library, specify the initial position and the target point, view message panel, start/stop/pause/continue simulation and supply input to the simulation. A custom 3D map can be constructed by inserting and/or deleting obstacles from an obstacle library. Obstacles can be inserted into the map by selecting the “Add” sub-menu item from the “Obstacle” menu. The main virtual reality interface and obstacle library window are shown in Fig. 1. The obstacle library currently includes a box, cylinder, cone, sphere and a table, which can all be scaled in 3D and rotated around the z-axis. After selecting an object from the library, the point where it will be located is determined by left clicking on the map. Similarly, obstacles can be removed by selecting the “Delete” sub-menu item from the “Obstacle” menu. Users can add/delete obstacles into/from the

map both prior and during the simulation. Existing obstacles can also be repositioned during the simulation providing a way to simulate a dynamic environment including non-stationary obstacles. Obstacle repositioning is accomplished through the function *moveObject*. The designed map including both static and dynamic obstacles can be saved to a file for future use. The LIDAR monitor visualizes the obstacles in the robot’s immediate environment depending on where the LIDAR is located on the mobile robot. The virtual LIDAR is assumed to be located at the origin of the monitor. The objects and planar surfaces represented with gray color are assumed to be known static obstacles where the rest of the objects are completely unknown to the simulated robot until they enter the LIDAR’s field of view. In other words, the dynamic path planner has no information about the global map except for the stationary walls covering the configuration space of the mobile robot.

Current LIDAR data, obstacle-free passable gaps, optimal gap and the optimal local target point are visualized on the LIDAR monitor. The monitor is activated after the simulation is started. The main virtual reality window displays the 3D map including static, traversed and predicted trajectories, obstacles and the simulated mobile robot. The real-time behaviors of the robot are observed through this window. A sample zoomed region of a virtual reality interface and the corresponding LIDAR monitor displaying the robot’s immediate surroundings are given in Fig. 2.

The main window also includes a message panel that is used to inform users about certain situations. Sample messages and their descriptions are given below:

- *Message 1:* “Navigating the robot”: This message indicates that the simulation has started successfully and the robot is moving towards the global target.
- *Message 2:* “The mobile robot successfully reached the target”: This message is displayed when the robot reaches the global target. The robot is assumed to reach the target after getting closer to the target than a certain distance.
- *Message 3:* “No obstacles on the path, heading directly towards the target”: This message expresses that the mobile robot can proceed to the global target through a safe corridor.

Table 1 Definitions of the Symbols Used in the GUI
















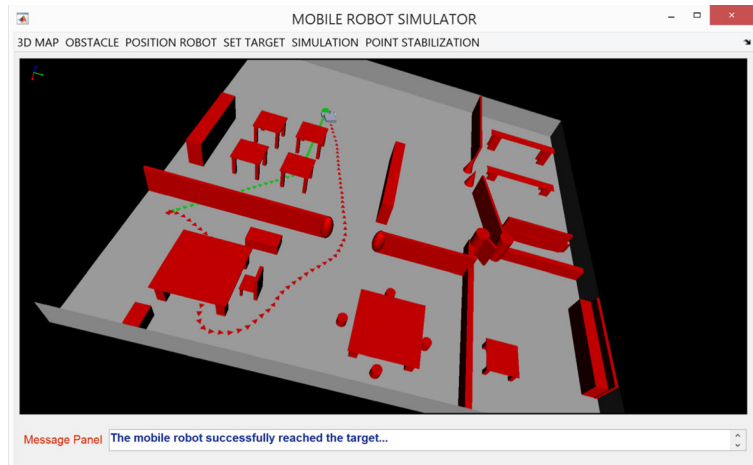
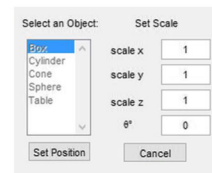
Symbol	Monitor	Definition
	VRI	Mobile robot
	VRI	Region of static obstacles
	VRI	Region of unmapped obstacles
	VRI	Static path
	VRI	Traversed path
	VRI	Predicted path
	VRI	Global target
	LM	Points of obstacles
	LM	Possible collision point
	LM	Local target
	LM	Safe maneuvering point
	LM	Detected gaps
	LM	Filtered gaps
	LM	Optimal gap
	LM	Mobile robot

Fig. 1 The main virtual reality interface (a) and obstacle library window (b)



(a)



(b)

- *Message 4*: “Robot is on collision course”: This message denotes that if the robot keeps its current course, it might collide with an obstacle. In such a case, the collision avoidance scheme updates the local target allowing safe maneuvering.
- *Message 5*: “Cannot assign a local target”: This message is displayed when the robot does not detect any passable gaps. In such a case, the mobile robot makes a zero-radius turn searching for a passable gap.
- *Message 6*: “Target is unreachable”: This message is displayed if the target assigned is unreachable given the static map before the simulation is started or the robot does not sense any gap after a 360° search of its environment during the simulation.

3 Main Functions and the Navigation Algorithm

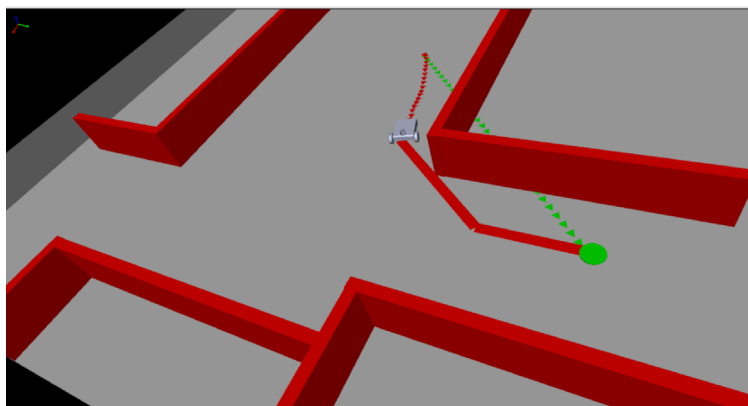
3.1 Main Functions and Variables

Table 2 provides a list of the commonly used input-output parameters.

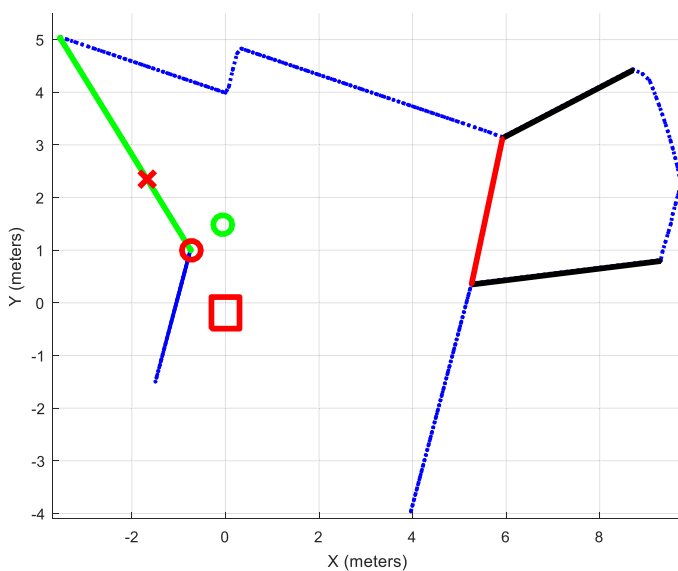
Definitions for the main functions are given below:
 InitSimParam() : This function is used to set the simulation parameters. Size of the mobile robot, LIDAR’s location on the robot, cruise speed, LIDAR settings, localization parameters, EKF and collision avoidance settings, map and display options are set through this function.

- [LIDAR] = acquireLIDARdata (robotPosition): This function generates virtual LIDAR data given the robot’s current position and orientation. LIDAR data is generated either 2D or 3D according to user’s choice.
- [safeGaps, gapDist] = FindObstacleFreeGaps(robotPosition, LIDAR): This function implements an algorithm to find the gaps between the obstacles through which the robot can safely navigate. Gaps are defined as line segments joining obstacle corners.
- distMtrx = fcn_euclidian_dist_trans(Ireal_BW, target): This function computes EDT with respect to a target point.
- [localTarget, OptimalGap, OptimalGapDist] = SetLocalTarget(safeGaps, gapDist, distMtrx,

Fig. 2 VRI (a) and LIDAR monitor (b)



(a)



(b)

robotPosition, LIDAR): This function determines an optimal local target based on the mobile robot’s current position, gap configuration and the Euclidian distance transform matrix. The cost for a grid point on the map is determined by,

$$C(\mathbf{p}_i) = d(\mathbf{p}_i - \mathbf{p}_r) + D(\mathbf{p}_i) \tag{1}$$

where \mathbf{p}_i and \mathbf{p}_r are the i^{th} grid point on the line passing through the detected gaps and the position vector of the robot, respectively. $d(\mathbf{p}_i)$ is the Euclidian distance between the robot and the grid point \mathbf{p}_i . $D(\mathbf{p}_i)$ is the Euclidean distance transform value for \mathbf{p}_i . The grid point, which minimizes the cost function is assumed as the local target.

- $[V_r, V_l] = \text{SpeedController}(\text{robotPosition}, \text{local-Target}, \text{dist2Collision})$: This function sets the speeds of the left and right wheels. The speeds are set such that the robot is steered towards the given local target.
- $\text{posEst} = \text{LocalizeRobot}()$: This function determines the position of the mobile robot.
- $\text{kalm_filter}(\text{KalmanFilter})$: This function implements Kalman filtering.
- $\text{system_model}(\text{SystemModel})$: This function computes the current position of the mobile robot based on the kinematic equations.

Differential drive kinematic model is the default mechanism used in this study. An illustration of

Table 2 Input/Output Parameters Commonly Used in Functions

Parameters	Definitions
Imap_BW	The grid-based global map containing free cells and static obstacles
Ireal_BW	The grid-based global map containing both static and dynamic obstacles
robotPosition	Position of the mobile robot with respect to the global coordinates.
distMtrx	Euclidian distance transform matrix computed for a given map.
LIDAR	Distance array acquired from the virtual LIDAR.
lidarRes	Angular scanning resolution of the LIDAR (0.5° or 1°).
start and target	Robot’s initial position and target point.
KalmanFilter	A structure of state estimate, input, current measurement value of the output, process covariance, measurement covariance and error covariance.
SystemModel	A structure of state and input vectors.
safeGaps	Free spaces between obstacles through which the robot can safely pass.
OptimalGap	The gap on which the optimal local target is located.
OptimalGapDist	Width of the optimal gap.
gapDist	Array of widths of the detected gaps.
localTarget	2D position of the local target with respect to the global coordinates.
localTarget.upd	Collision avoidance procedure updated local target.
dist2Collision	The distance to the nearest obstacle which is closer than a critical value.
Vl	Velocity of the left wheel (in cm/sn).
Vr	Velocity of the right wheel (in cm/sn).

the model is provided in Fig. 3. Assuming that $v_l, v_r, b, \theta, x, y$ are the left and right wheel velocities, wheel base, orientation angle and x and y position of the mobile robot, respectively. Rotation radius (R), angular velocity (w) and instant curvature (I_c) are given by [28]

$$R = \frac{b v_r + v_l}{2 v_r - v_l}, \quad \omega = \frac{v_r - v_l}{b}, \quad I_c = [x - R \sin \theta, y + R \cos \theta] \tag{2}$$

State-space model of the differential drive mobile robot is then given by,

$$\begin{bmatrix} x_{t+\Delta t} \\ y_{t+\Delta t} \\ \theta_{t+\Delta t} \end{bmatrix} = \begin{bmatrix} \cos(\omega \Delta t) & -\sin(\omega \Delta t) & 0 \\ \sin(\omega \Delta t) & \cos(\omega \Delta t) & 0 \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} x - I_{c_x} \\ y - I_{c_y} \\ \theta \end{bmatrix} + \begin{bmatrix} I_{c_x} \\ I_{c_y} \\ \omega \Delta t \end{bmatrix} \tag{3}$$

Although differential drive mobile robots are considered in this study, the toolbox can be modified to

be used with other types of mobile robots by updating the mathematical model defined in *system_model* function.

- `displayPath(xp, yp, type)`: Function to display a path given the type and points on the path. Type of the path is 1 for optimal path (green-triangular trajectory), 2 for predicted path (red), 3 for traversed

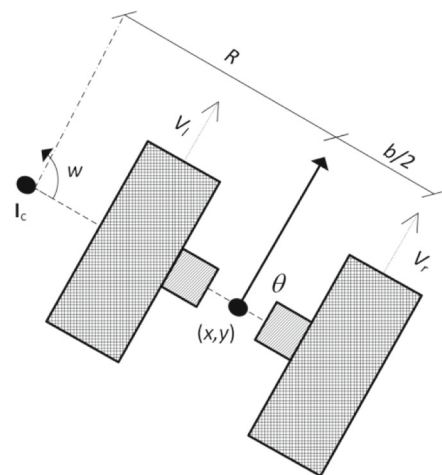


Fig. 3 Differential drive kinematics

path (red-triangular trajectory) and 4 for other use (solid green circles).

- deletePath(type): Function to delete the specified path.
- [localTarget_upd] = CollisionAvoidance(robotPosition, localTarget, OptimalGap, safeGaps, gapDist, OptimalGapDist, LIDAR): This function checks whether there is an obstacle close enough to the direct path to current local target that the robot might collide with or not. In case of an obstacle, the current local target is updated to allow safe maneuvering.

Figure 4 illustrates the collision avoidance procedure used in the simulation. Let t , illustrated by red cross, be the local target determined by the function *SetLocalTarget*. p -line denotes the line from the center of the robot to the local target. Out of the LIDAR measured obstacle points closer the robot than the local target t , the one nearest to p -line is marked as the critical collision point, c , illustrated by a red circle. Let the distance between c and p -line be d_{min} . If d_{min} is less than a pre-determined safe distance d_{tol} , the collision avoidance algorithm updates the local target as the point t^* that is d_{tol} away from c such that the line from c to t^* is perpendicular to p -line. The mobile robot is then steered to t^* to avoid possible collision.

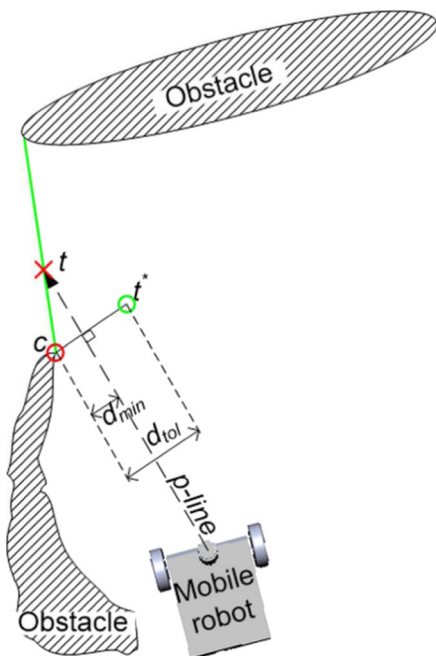


Fig. 4 Collision avoidance procedure

- [xp_exact, yp_exact]=fcn_exact_euclidian_dist_trans(Ireal_BW, x_init, y_init, theta_init, x_target, y_target, theta_target): This function computes the exact Euclidian distance transform (EEDT) given the start and the target points. The function computes the waypoints that are utilized in point stabilization module.
- [X_pt_stab, Y_pt_stab, T_pt_stab]=PointStabilization(): This function performs point stabilization. The point stabilization method proposed in this study is presented in Section 3.2.

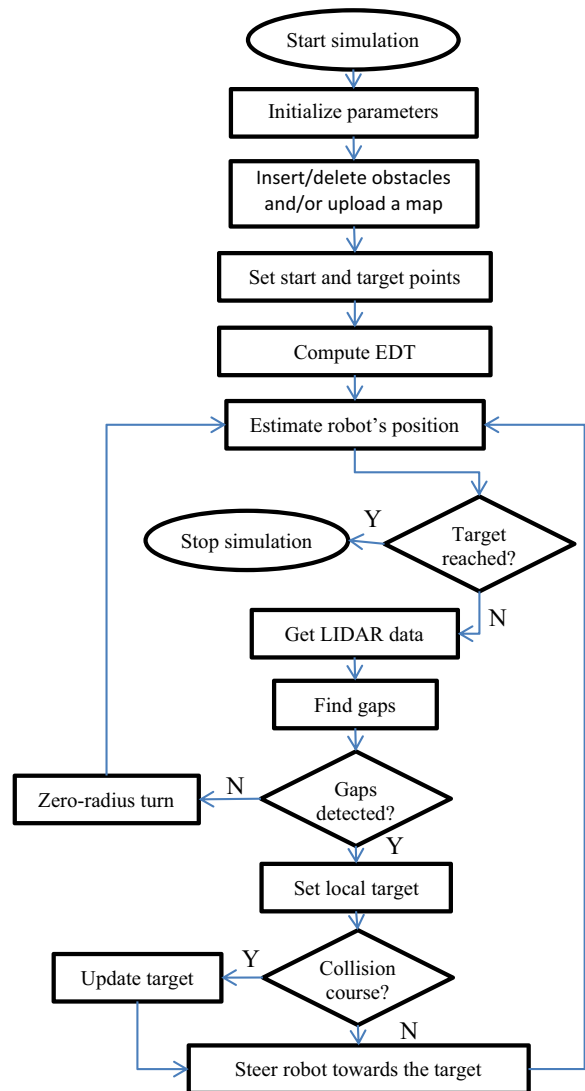


Fig. 5 Flowchart of the navigation algorithm

Fig. 6 Simulation result for the map with mapped and unmapped obstacles

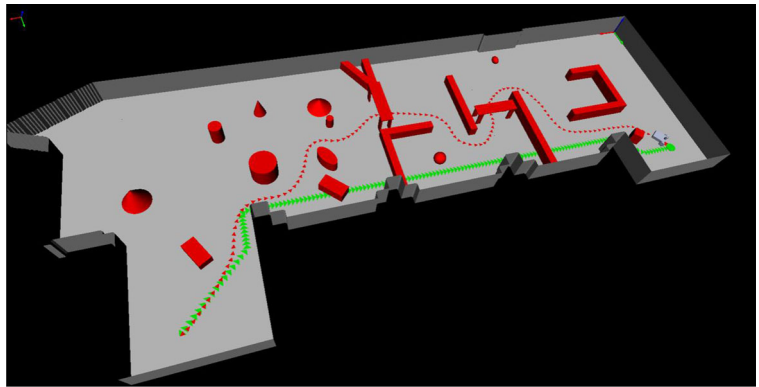
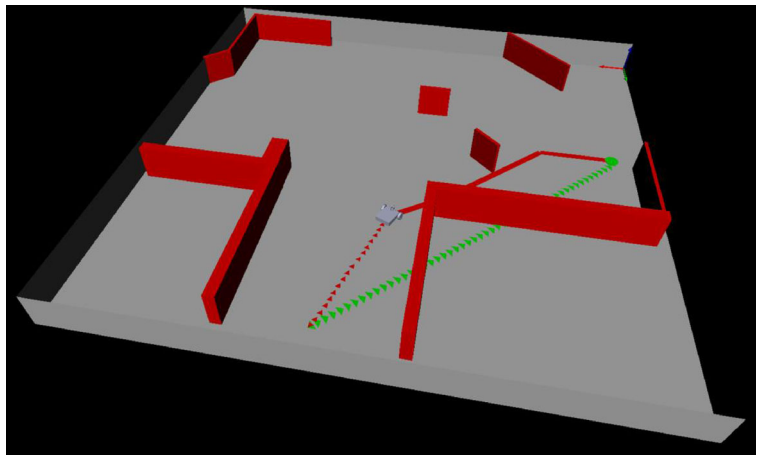
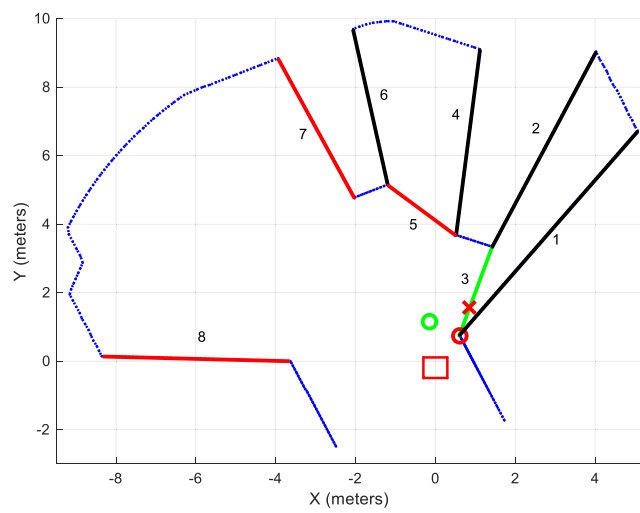


Fig. 7 Sample Scenario-1: VRI (a) and LIDAR (b) monitors



(a)



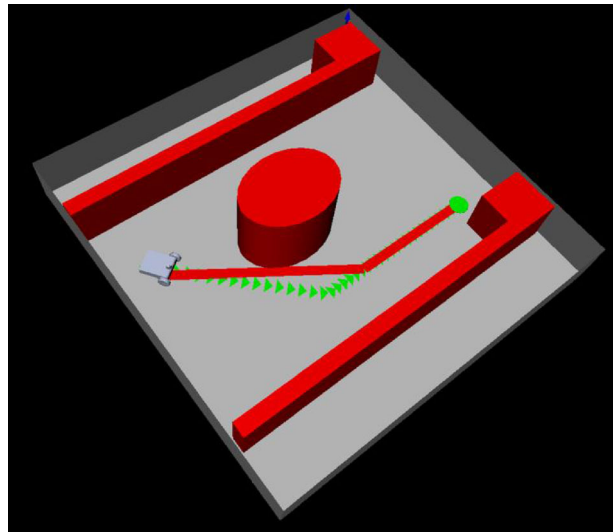
(b)

- [objNo] = insertObject(pos, scale, strObj): This function is used to insert an object to a specified position. The parameter 'pos' represents a vector with 3 elements: x-position (in cm), y-position (in cm) and orientation angle (in degrees), respectively. Object type is specified as a string ('Box', 'Cone', 'Cylinder', 'Sphere' or 'Table'). The physical size of the object can be scaled with the resizing factor 'scale'. 'objNo' is

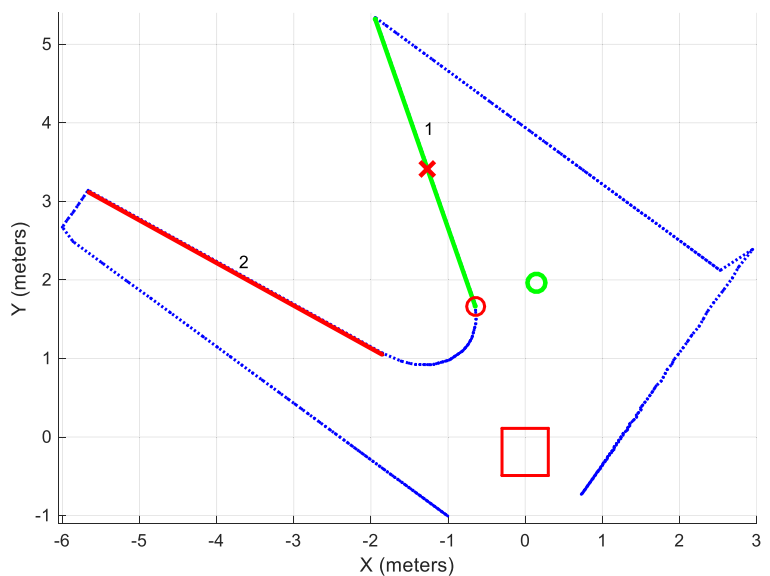
a positive integer ID which is assigned to each inserted object used later for deleting and repositioning the objects.

- deleteObject(objNo): This function is used to delete the object specified by 'objNo'.
- moveObject(objNo, pos): This function is used to move an object to a specified position.
- setGlobalTarget(pos): This function is used to set the global target point to a specified position.

Fig. 8 Sample Scenario-2: VRI (a) and LIDAR (b) monitors

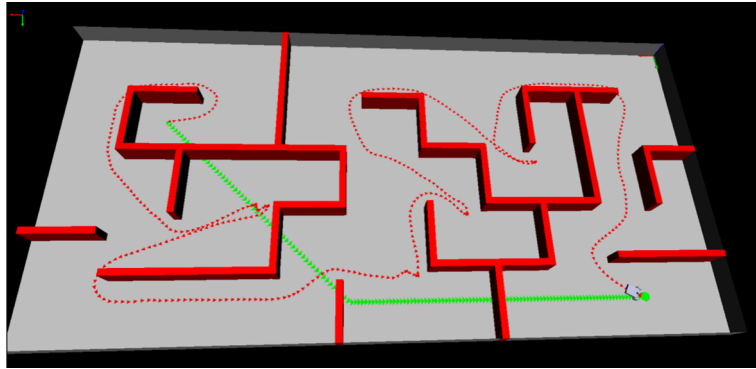


(a)



(b)

Fig. 9 Simulation result for a maze-type environment



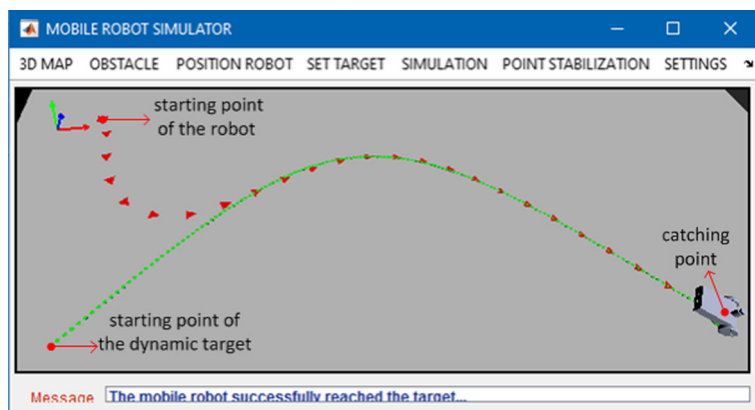
By calling `moveObject` and `setGlobalTarget` function, one can simulate dynamic a dynamic environment consisting of moving objects and target.

- `NAV_tangentBug()`: This function is used for executing the tangent bug algorithm when the simulation is started. One can switch this mode on by selecting the tangent bug as the navigation algorithm.
- `NAV_dynaBug()`: This function implements the navigation algorithm presented in this paper. When selected as the navigation mode through the simulation parameters settings, the mobile robot is navigated from its current position to the specified target point using the proposed navigation algorithm.
- `NAV_customCode()`: One can develop a custom navigation algorithm under this procedure. When the user-defined procedure is selected as the valid navigation procedure, this function is evoked to navigate the mobile robot right after the simulation is started.

3.2 Navigation Algorithm

The flowchart describing the navigation algorithm is given in Fig. 5. The user first designs the workspace by loading a map and/or adding/deleting obstacles and sets the start and the target points on the map. When the simulation is started, the parameters are initialized. EDT is computed with respect to the target point provided by the user. The position of the robot is estimated through the function *LocalizeRobot* which implements the Kalman based localization scheme presented in [28]. Next, the algorithm computes the Euclidean distance between the robot's current position and the global target. If the distance is smaller than a pre-defined threshold and there is no obstacle between the robot and the target, the robot is assumed to reach the target and the simulation is stopped. Otherwise, range data is acquired from the virtual LIDAR sensor. Based on the LIDAR data, obstacle free gaps are detected by the function *FindObstacleFreeGaps*. In the gap detection scheme, gaps are initially located

Fig. 10 Simulation result for dynamic global target-1



by tracking the differences between two consecutive LIDAR measurements that are bigger than a certain threshold. The gaps are then post-processed by the algorithm. If the robot cannot proceed to a given gap without passing through another gap, the algorithm keeps the gap the robot must pass first while deleting the other(s). An example scenario is presented in the results section. If the algorithm does detect any gaps, the robot starts making a zero-radius turn searching for gaps. Otherwise, the dynamic path planning function *SetLocalTarget* computes the costs for the grid points on the detected gaps based on the cost function given in Eq. 1 and sets the point with the minimum cost as the optimal local target. The optimal local target is marked with a red cross on the LIDAR monitor. The function *CollisionAvoidance* checks the direct path to the local target for obstacles close enough to the path that the robot might collide with. The local target is updated with a safe one under the presence of such an obstacle. The robot is finally steered towards the updated local target by the function *SpeedController*.

The proposed navigation algorithm also includes a point stabilization method that is a combination of exact Euclidian distance transform (EEDT) [8] and

model predictive control (MPC) [14, 15]. Once the user positions the mobile robot on the map and provides a target, EEDT algorithm generates waypoints from the initial position to the target point. Then MPC based point stabilization is applied on consecutive waypoint pairs starting from the initial point to the goal point. Initial and final orientation angles of the mobile robot are requested from the user through an external message box. The orientation angles at the waypoints are calculated by averaging the gradients of the line segments constructed by consecutive waypoints. MPC based algorithm utilizes the waypoints and the corresponding orientation angles in order to generate sub-trajectories. The sub-trajectories are then combined to obtain the optimized path from the initial position to target.

4 Simulation Results

In this section, simulation results based on the 3D map of the 3rd floor of the Mechatronics Engineering Department of Kocaeli University and custom 3D maps are presented. For the simulation result given in

Fig. 11 Simulation result for dynamic global target-2

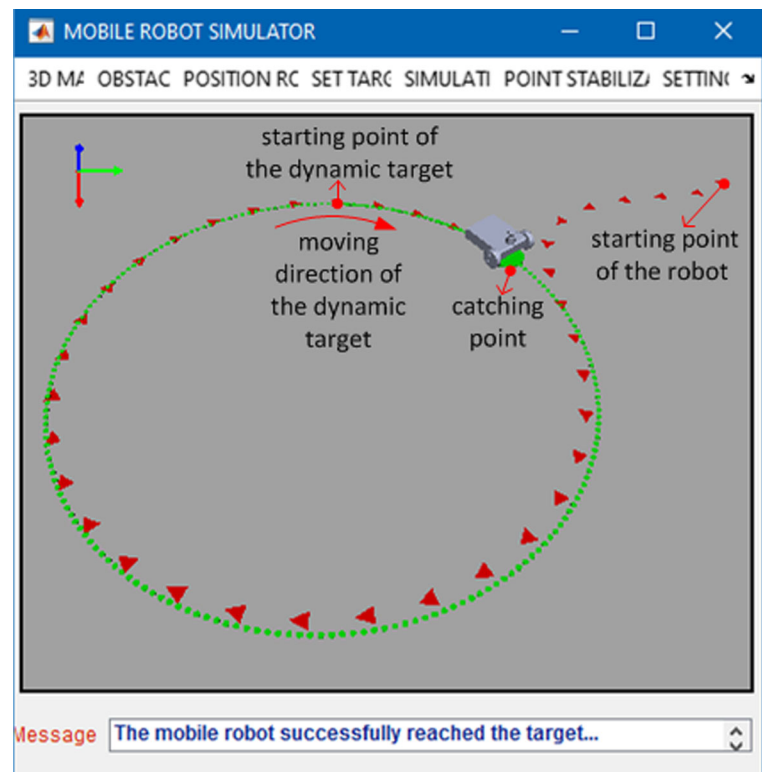


Fig. 6, a certain number of random shaped unmapped (dynamic) obstacles are placed on the map. The map containing static and dynamic obstacles, simulated mobile robot, static, traversed and predicted paths between the start and the global target points are given in the figure. The static path generated by the EDT algorithm is depicted with the green trajectory. It is possible for an optimal static path to cross over an unmapped obstacle since unmapped obstacles are not taken into account when computing the distance transform. Sampling period of the simulation is set to 80 ms. Localization, LIDAR simulation, gap detection, local target setting and monitoring processes are executed within a single period. The map size is 600 pixels x 600 pixels with one pixel corresponding to a 5 cm x 5 cm area. Consequently, the map illustrates a 30 m x 30 m indoor area.

The virtual mobile robot has differential drive kinematic model. The size of the robot is 60 cm x 60 cm with a front track width of 50 cm. Minimum, maximum, and cruise velocities are set to 0 cm/s, 100 cm/s and 50 cm/s, respectively. Maximum LIDAR range is simulated to be 10 m with 270 degrees angle of view and 1 degrees of angular resolution. LIDAR is located at (0, -2) cm coordinates with respect to mobile robot's origin (the center of the wheel-base). The simulation stops when the robot reaches the destination with 30 cm tolerance. Process and measurement noise covariances used in Kalman filtering are set to 15 cm/s and 30 cm, respectively. If an obstacle gets closer than 2 m interfering with the robot's planned path, an action is taken to eliminate the possibility of a collision.

Figures 7 and 8 illustrate gap detection, local target setting and collision avoidance procedures under two different scenarios. Both scenarios are illustrated by giving immediate VRI screenshots and the corresponding LIDAR monitors. Figure 7 depicts the virtual reality window and the LIDAR monitor for the first scenario. In the gap detection scheme, gaps are initially located by tracking the differences between two consecutive LIDAR measurements that are bigger than a certain threshold. Gaps numbered 1, 2, 4, 6, 7 and 8 are initially detected by the function *FindObstacleFreeGaps*. After the post-processing step, gaps 1 and 2 are replaced by the 3rd gap since the mobile robot cannot proceed to either gap without passing through the 3rd gap. Similarly, the 4th and the 6th gaps are replaced by the 5th gap. The optimal local target determined by the function *SetLocalTarget* is

marked with a red cross on the LIDAR monitor. The collision avoidance procedure detects a critical collision point marked with a red circle and then updates the local target as the point illustrated with the green circle. Figure 8 depicts the virtual reality window and the LIDAR monitor for the second scenario. The gaps detected by the gap detection scheme are numbered as 1 and 2. The optimal local target set by the dynamic path planner is shown with a red cross on the 1st gap.

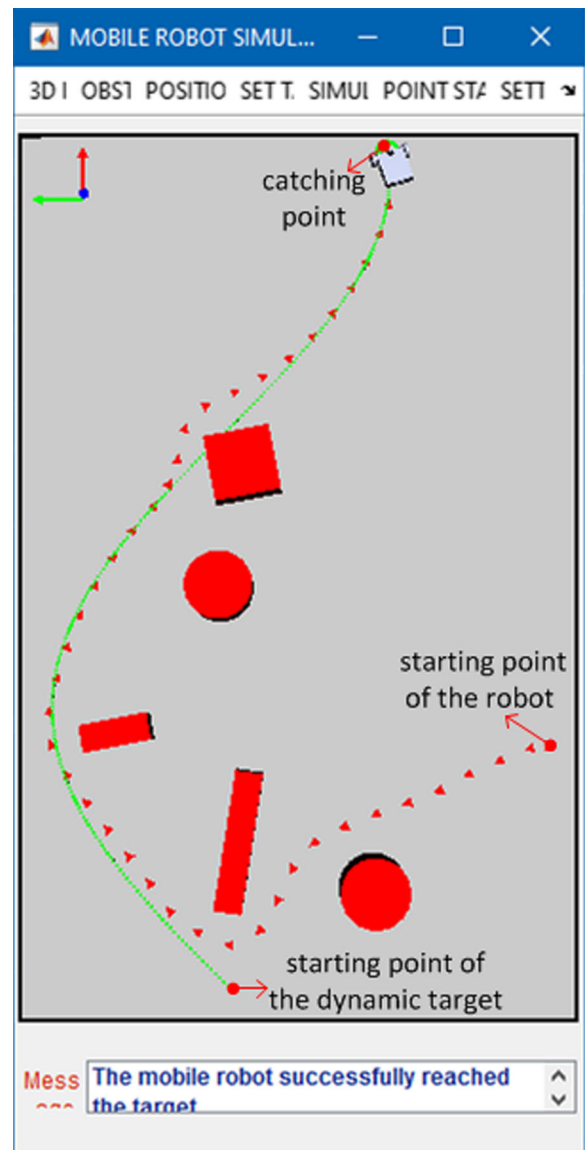
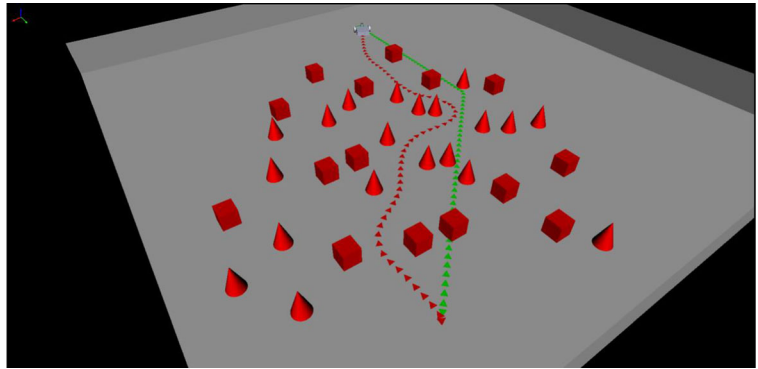


Fig. 12 Simulation result for dynamic global target with unmapped obstacles

Fig. 13 Simulation result for a complex environment



The local target updated by the collision avoidance module is depicted with a green circle.

Figure 9 illustrates the simulation result for a maze-type environment. Following the optimal local target point, the mobile robot initially enters dead-end rooms in the maze since the simulated LIDAR sensor does not sense the inside walls. When the walls inside the rooms fall within the simulated LIDAR's field of view, the robot realizes that there is no exit. Consequently, it turns back and follows an alternative path that leads to the global target.

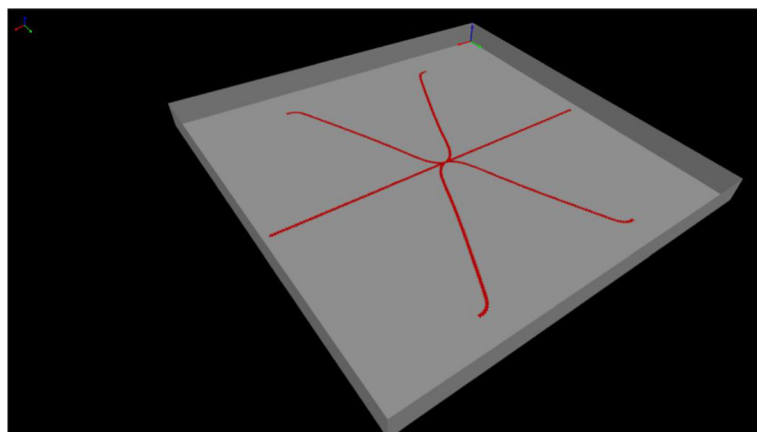
Figures 10–12 present the performance of the navigation algorithm for non-stationary targets. Figure 10 illustrates that the mobile robot and the dynamic target start from different positions and the dynamic target tracks a sinusoidal trajectory where the mobile robot follows the target. The mobile robot catches the dynamic target at the “catching point”. Similarly, as

seen in Fig. 11, the dynamic target starts its motion through a circular trajectory in CW manner and the mobile robot reaches the target at “catching point”.

Figure 12 highlights both dynamic target tracking and obstacle avoidance procedures simultaneously. The mobile robot does not only follow the moving target but also tracks sudden changes in its immediate environment to avoid collision with unmapped obstacles. The average velocities of the mobile robot and the dynamic target are tuned to be close to each other. Thus, catching the dynamic obstacle takes a certain time.

Figure 13 illustrates the simulation result for a complex environment. The map consists of several small-sized unmapped obstacles. As depicted in the figure the mobile robot successfully planned a collision-free dynamic path, navigated through the obstacles and eventually reached the global target.

Fig. 14 Point stabilization in obstacle free environment



Message Panel Point stabilization completed

Fig. 15 Point stabilization in the presence of an obstacle

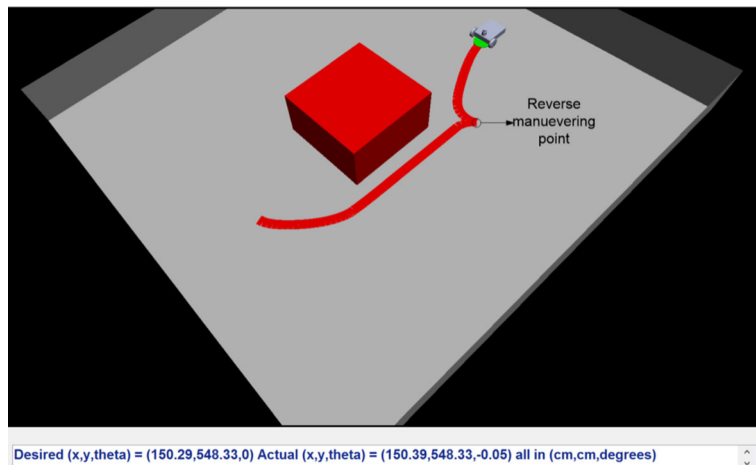


Figure 14 shows the point stabilization results for an obstacle free environment whereas Fig. 15 illustrates a point stabilization example in the presence of an obstacle. The maps depicted in both figures demonstrate 15m x 15m sized workspace area.

The origins of both maps are the top right corners. In Fig. 14, the target point is assigned to be (7.5m, 7.5m) with 0° heading angle. The figure demonstrates that the simulated mobile robot finally settles at the target point, starting from six different initial positions. The starting positions are located on a circle centered on the target point with 7.5m radius.

The scenario given in Fig. 15 demonstrates point stabilization procedure with a 90° initial and 0° target orientations. As clearly seen in the figure, the mobile robot starts its motion from an arbitrary position with the initial heading angle towards the given target coordinates. The mobile robot makes a reverse maneuvering motion on the trajectory to achieve the 0° final

heading orientation. A comparison between desired position and actual position at the target point is given in the message panel as seen in Fig. 15. The final positioning errors are 0.1 cm, 0 cm and 0.05° for the x position, y position and orientation angle, respectively.

Simulation results prove the efficacy of the static/dynamic path planning, navigation, gap detection, collision avoidance and point stabilization algorithms utilized in the toolbox. Although the methods

provide satisfactory results, the users can choose to replace any algorithm with a custom one and test its performance. This allows users to effectively prototype algorithms.

5 Conclusion

A wheeled mobile robot simulator is presented in this study that offers a 3D virtual reality interface. Unlike the tools presented in the literature, the proposed simulator provides an all in one simulation environment. An external installation is not required for 3D visualization or code development. The toolbox incorporates static/dynamic path planning, point stabilization, localization, navigation and collision avoidance algorithms. Each of these algorithms is defined in a separate function which caters flexibility. Researchers and engineering students can use this toolbox to test and verify custom navigation algorithms, kinematic models, and localization systems prior to physical implementation. The algorithms in the toolbox consider the mobile robot kinematic and physical constraints allowing for real-time implementation of prototyped algorithms.

Acknowledgments This study was supported by the Scientific and Technical Research Council of Turkey (TUBITAK) under the grants TUBITAK-113E777 and TUBITAK BIDEB-2211.

References

1. Gerecke, U., Wagner, B.: The challenges and benefits of using robots in higher education. *Intelligent Automation & Soft Computing* **13**, 29–43 (2007)
2. Stormont, D.P., Chen, Y.Q.: Using mobile robots for controls and mechatronics education. *International Journal of Engineering Education* **2**, 1039–1042 (2005)
3. Chen, C., Chai, W., Roth, H.: A single frame depth visual gyroscope and its integration for robot navigation and mapping in structured indoor environments. *J. Intell. Robot. Syst.* **80**, 365–374 (2015)
4. Dean, E., Nair, S., Knoll, A.: User-friendly Matlab-toolbox for symbolic robot dynamic modeling used for control design. In: *Proceedings of the IEEE International Conference on Robotics and Biomimetic*. Guangzhou, China (2012)
5. Yoshida, K.: The SpaceDyn: A Matlab toolbox for space and mobile robots. In: *Proceedings of the IEEE International Conference on Intelligent Robots and Systems*. Sendai, Japan (1999)
6. Shokraneh, K.M., Masehian, E.: Planning robot navigation among movable obstacles (NAMO) through a recursive approach (2016). doi:[10.1007/s10846-016-0344-1](https://doi.org/10.1007/s10846-016-0344-1)
7. Zhengcai, C., Yingtao, Z., Qidi, W.: Genetic fuzzy + PI path tracking control of a non-holonomic mobile robot. *Chin. J. Electron.* **20**, 31–34 (2011)
8. Carlos, E.J., Federico, P.E., Gabriel, R.J.: The exact Euclidean distance transform: A new algorithm for universal path planning. *Int. J. Adv. Robot. Syst.* **10**, 1–10 (2013)
9. Wu, Z., Feng, L.: Obstacle prediction-based dynamic path planning for a mobile robot. *International Journal of Advancements in Computing Technology* **4**, 118–124 (2012)
10. Silijak, H.: Inverse matching-based mobile robot following algorithm using fuzzy logic. *Int. J. Robot. Autom.* **29**, 369–377 (2014)
11. Ssebazza, L., Pan, Y.J.: DGPS-based localization and path following approach for outdoor wheeled mobile robots. *Int. J. Robot. Autom.* **30**, 13–25 (2015)
12. Alejo, D., Cobano, J.A., Heredia, G., Ollero, A.: A Reactive method for collision avoidance in industrial environments. *J. Intell. Robot. Syst.* (2016). doi:[10.1007/s10846-016-0359-7](https://doi.org/10.1007/s10846-016-0359-7)
13. Sun, S., Cui, P.: Path tracking and a practical point stabilization of mobile robot. *Journal of Robotics and Computer-Integrated Manufacturing* **20**, 29–34 (2004)
14. Alves, J.A.V., Lages, W.F.: Real-time point stabilization of a mobile robot using model predictive control. In: *Proceedings of the 13th IASTED International Conference of Robotics and Applications*. Würzburg, Germany (2007)
15. Kühne, F., Lages, W.F., Silva, J.M.G.: Point stabilization of mobile robots with nonlinear model predictive control. In: *Proceedings of the IEEE International Conference on Mechatronics and Automation*. Niagara Falls, Canada (2005)
16. Pazderski, D.: Waypoint following for differentially driven wheeled robots with limited velocity perturbations. *J. Intell. Robot. Syst.* (2016). doi:[10.1007/s10846-016-0391-7](https://doi.org/10.1007/s10846-016-0391-7)
17. Corke, P.: A robotics toolbox for Matlab. *IEEE Robot. Autom. Mag.* **3**, 24–32 (1996)
18. Mirats, J.M., Pfeiffer, C.F.: Mobile robot design in education. *IEEE Robot. Autom. Mag.* **13**, 69–75 (2006)
19. Zahedi, K., Twickel, A.V., Pasemann, F.: Yars: A physical 3D simulator for evolving controllers for real robots. *Lecture Notes in Computer Science*, 75–86. Springer (2008)
20. Awaad, I., Len, B.: Xpersim: A simulator for robot learning by experimentation. *Lecture Notes in Computer Science*, 5–16. Springer (2008)
21. Freese, M., Singh, S., Ozaki, F., Matsuhira, N.: Virtual robot experimentation platform V-Rep: a versatile 3D robot simulator. In: *Proceedings of the 2nd International Conference on Simulation, modelling and Programming for Autonomous Robots*. Berlin, Germany (2010)
22. Barner, S., Geisinger, M., Buckl, C., Knoll, A.: EasyLab: Model-based development of software for mechatronic systems. In: *Proceedings of the IEEE International Conference on Mechatronic and Embedded Systems and Applications*. Beijing, China (2008)
23. Pons, C., Perez, G., Giandini, R., Baum, G.: A model-driving approach to constructing robotic systems. *J. Comput. Sci. Technol.* **14**, 1–8 (2014)
24. Gućwa, K.J., Cheng, H.H.: RoboSim for Integrated Computing and STEM Education. In: *Proceedings of the 121st ASSE Annual Conference & Exposition*. Indianapolis, IN, USA (2014)
25. Ivaldi, S., Padois, V., Nori, F.: Tools for dynamics simulation of robots: a survey based on user feedback. *Tools for Dynamics Simulation of Robots-Extended Report*, 1–15 (2014)
26. Karakaya, S., Ocać, H., Küćüküydüz, G.: A bug-based local path planning method for static and dynamic environments. *International Symposium on Innovative Technologies in Engineering and Science*. Valencia, Spain (2015)
27. Chen, S.Y.: Kalman filter for robot vision: a survey. *IEEE Trans. Ind. Electron.* **59**, 4409–4420 (2012)
28. Karakaya, S., Küćüküydüz, G., Ocać, H.: A hybrid indoor localization system based on infra-red imaging and odometry. *International Conference of Computer Vision and Pattern Recognition*. Las Vegas, NV, USA (2015)
29. Kim, J., Chung, W.: Efficient placement of beacons for localization of mobile robots considering the positional uncertainty distributions. *Int. J. Robot. Autom.* **30**, 119–127 (2015)

Suat Karakaya is a research assistant at Kocaeli University, Turkey. He received his B.S. degree in Mechatronics Engineering from Kocaeli University in 2011. Then, he joined the Machine Vision Laboratory of Mechatronics Engineering Department in Kocaeli University. In 2014, he participated as an intern researcher in an R&D project supported by The Scientific and Technological Research Council of Turkey (TUBITAK) under the grant number TUBITAK-113E777. His research areas include mobile robotics, collision avoidance and control theory.

Gurkan Kucukyildiz received his B.S and M.S. degrees from Kocaeli University, Kocaeli, Turkey in 2009 and 2012, in Communication and Electronics Engineering and Mechatronics Engineering, respectively. He is currently a research assistant at Mechatronics Engineering Department at Kocaeli University. His research interests include pattern recognition, biomedical signal processing, computer vision and embedded systems.

Hasan Ocak received his B.S., M.S. and Ph.D. degrees from Case Western Reserve University, Cleveland, OH, USA in 1997, 1999 and 2004, respectively, all in Electrical Engineering. He is a Full Professor of Mechatronics Engineering Department at Kocaeli University, Kocaeli, Turkey. Dr. OCAK is currently the department chair and the coordinator for the Machine Vision Laboratory. His research interests include pattern recognition, biomedical signal processing, computer vision and intelligent systems.