Neural Based Autonomous Navigation of Wheeled Mobile Robots

Submitted: 16th February 2016; accepted: 18th June 2016

Mariam Al-Sagban, Rached Dhaouadi

DOI: 10.14313/JAMRIS_2-2016/17

Abstract:

This paper presents a novel reactive navigation algorithm for wheeled mobile robots under non-holonomic constraints and in unknown environments. Two techniques are proposed: a geometrical based technique and a neural network based technique. The mobile robot travels to a pre-defined goal position safely and efficiently without any prior map of the environment by modulating its steering angle and turning radius. The dimensions and shape of the robot are incorporated to determine the set of all possible collision-free steering angles. The algorithm then selects the best steering angle candidate. In the geometrical navigation technique, a safe turning radius is computed based on an equation derived from the geometry of the problem. On the other hand, the neural-based technique aims to generate an optimized trajectory by using a user-defined objective function which minimizes the traveled distance to the goal position while avoiding obstacles. The experimental results demonstrate that the algorithms are capable of driving the robot safely across a variety of indoor environments.

Keywords: *reactive navigation, obstacle avoidance, autonomous ground robots, recurrent neural networks*

1. Introduction

Mobile robots have rapidly evolved over the past years to encompass a wide spectrum of applications: Robots are assisting in driving vehicles, aiding in medical tasks, and taking charge in hazardous rescue missions. Autonomous navigation is a key feature in all of these applications. It deals with the problem of navigating to a target location while avoiding collision with obstacles that may be present in the environment. One approach to autonomous navigation is model-based approach. It uses a model of the environment to generate a safe path to the target location [11]. Classical methods for this type include: Road maps [7], cell decomposition, and potential fields [8]. Although these methods may produce efficient paths, a global and accurate map of the environment is not available when the environment is unknown or is dynamic. Hence, model based methods are used only in artificially controlled environment. A complementary approach to autonomous navigation is obstacle avoidance (also known as reactive navigation). No prior information is required about the environment. Instead, obstacles are discovered in real time while the robot is executing its mission. The main challenges in developing such methods are: computational complexity, sensors uncertainties, robot geometrical shape, and kinematic and dynamic constraints. In addition, because a global map of the environment is not available the robot may produce inefficient paths or converge to a local minimum (trap situation) [5].

Neural Networks-based techniques have been proposed in the literature to solve the motion problem. In [6] the path planning problem is viewed as two subproblems: find space and find path. Two neural networks connected in cascade are used. The first neural network is responsible for finding the C-free space which is the set of all possible robot configurations that avoids collision with obstacles. The second neural network guides the robot through the free space segments to the target location. The navigational methodology used in [3] is a Probabilistic Neural Network (PNN). This type of network facilitates the training process, allows a faster time of response and has a low computational cost. The output of the network represents the steering direction which takes three forms: forward, right turn, and left turn. The autonomous navigation of a mobile robot is considered as a classification problem. The motion of the robot and the sensorial information are the patterns to be classified. The obstacle avoidance problem in [10] is divided into three subproblems: General obstacle avoidance, corridor and wall following, and passing through a door. A separate neural network is designed for each one of those subproblems. The algorithm uses the accessible space as the input to the neural network. The output of the network is the steering angle and velocities. The number of neurons in the hidden layer is determined by using a Bayesian framework which computes the evidence of a set of neural networks with different hidden nodes. The accurate usable accessible space is computed by incorporating the wheel chair dimensions, laser information, and encoder data. The algorithm choice is interesting because it is able to optimize the number of hidden neurons for a given problem. However, the algorithm is considered incomplete from the autonomous navigation perspective because it does not provide a framework that is able to recognize the relevant situation and selects the corresponding output. The authors divide the obstacle avoidance problem into sub-problems because they claim that a single neural network could not provide the desired performance. However, It is not clear whether it is necessary to separate the avoiding obstacle and passing through a door tasks since they do not produce a conflict in training the network. A training conflict is created when the same input pattern is mapped to two

different output values. Another artificial intelligence technique is introduced in [4]. The path planning is done using particle swarm optimization. The desired path is designed to avoid obstacles while maintaining a smooth continuous path. The path is expressed as a 5th order polynomial. Two of the polynomial coefficients are estimated via particle swarm optimization such that obstacles are avoided while the other coefficients are chosen such that a smooth path is generated. The particle swarm optimization is required to estimate the best polynomial coefficients as well as other parameters called the critical points. The algorithm was verified in simulation. However, nonholonomic constrains are not considered. Overall, artificial neural networks provides an interesting platform for the obstacle avoidance problem because of their generalization ability, ability to learn from examples and the ability to extract temporal dependencies. In this paper, we present an obstacle avoidance technique based on recurrent neural networks that takes into consideration the kinematic constraints of differential drive robots. While the common trend is to use more than one neural network, we use a single dynamic neural network. The obstacle avoidance problem is a dynamic problem that should be solved using dynamic methods. In order to guarantee optimal convergence, we require the neural network learning environment to satisfy certain conditions that are derived using Lyapunov stability method. Also, the earlier presented neural networks techniques generated a dataset by manually driving the robot across different scenarios. We automate the process by generating a sub-optimal dataset using a computer algorithm. For optimum performance, the neural network is trained using the real-time recurrent learning algorithm along with a customized objective function to equip the robot with the capability of improving its learning while in motion.

2. Autonomous Navigation Methodology

2.1. Geometrical Navigation Algorithm

The main steps in the obstacles avoidance algorithm are: Identify a reference steering angle, model the environment, compute the configuration space, and select the desired steering angle and radius of curvature [1]. The reference steering angle, γ_{ref} represents the steering angle that the robot takes in the absence of obstacles. It is an intermediate reference angle that will later help us find $\gamma_{desired}$.

In this paper, we consider a mobile robot with a differential drive configuration as shown in Fig. 1. Let the robot configuration be:

$$q_r = (x_r, y_r, \theta_r), \tag{1}$$

where (x_r, y_r) is the position of the robot in the $x-y$ plane, and $\theta_r \in [0, 2\pi)$ is the robot's orientation with respect to the x axis.
Let the target configuration be:

$$q_{target} = (x_{target}, y_{target}, \theta_{target}). \tag{2}$$

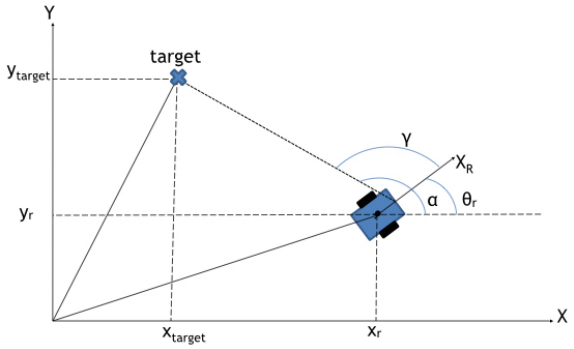

Fig. 1. Target and robot coordinates

Let $\vec{u}_e$ be the vector connecting the robot reference point to the target location. The phase angle of $\vec{u}_e$ is given by:

$$\alpha = \arctan \frac{y_{target} - y_r}{x_{target} - x_r}. \tag{3}$$

To correct the error in orientation, the robot should turn by a reference steering angle γ_{ref}. The instantaneous turning radius r_c can be evaluated by:

$$r_c = L\frac{v_r + v_l}{v_r - v_l}, \tag{4}$$

where v_L and v_R are the translational velocities of the left and right wheels and L is the distance between the wheels.

The second main step is to model the surrounding environment. A partial polar map of the workspace is constructed in the robot local frame. The robot is equipped with a laser range finder that is programmed to scan the 200° front view of the robot in 20 sectors, with 10° angular resolution. The sensor returns a set of points:

$$\mathcal{P}(q(t_i)) = \{p_1, p_2, ..., p_j, ..., p_{20}\}. \tag{5}$$

A point p_j is expressed by a pair (d_j, β_j) where d_j is the distance between the robot and the obstacle at sector j. β_j is the orientation of the j^{th} sector, S_j, with respect to the local x axis. The subset of workspace obstacles seen at configuration $q(t_i)$ is identified by applying a threshold on d_j:

$$\mathcal{O}(q(t_i)) = \{p_j \in \mathcal{P}(q(t_i)) | d_j \leq R_{safe}\}. \tag{6}$$

The third main step of the algorithm is to compute the configuration space C_{obst}. First, consider the case where only a point obstacle exists in the workspace: $\mathcal{O} = \{p_j\}$. To find $\mathcal{C}_{obst}$, we slide the robot around p_j and trace the configurations it went through, as illustrated in Fig. 2. Hence, $\mathcal{C}_{obst}$ is enclosed by a circle C_j of radius R and center $I_j = (I_{j,x}, I_{j,y})$:

$$\mathcal{C}_{obst} = \{q \in \mathcal{C} | (x - I_{j,x})^2 + (y - I_{j,y})^2 \leq R^2\}, \tag{7}$$

$$I_{j,x} = R + d_j \cos\beta_j,$$
$$I_{j,y} = d_j \sin\beta_j, -100^\circ \leq \beta_j \leq 100^\circ.$$

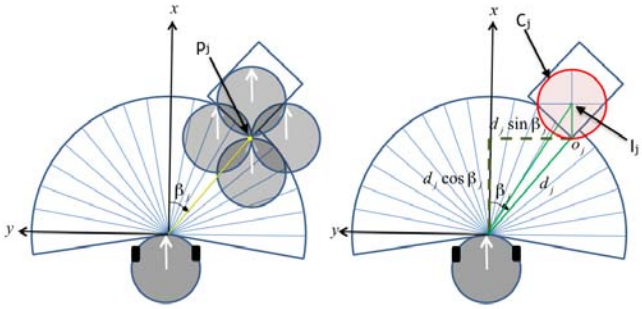

Fig. 2. C-space Algorithm

Next, we find L_i which is the radial distance between the robot and the boundary of $\mathcal{C}_j$ at angle β_i:

$$L_i = \min\{\rho_j \cos(\beta_i - \phi_j) \pm \sqrt{R^2 - \rho_j^2 \sin^2(\beta_i - \phi_j)}\},$$
$$\alpha_{min} \leq \beta_i \leq \alpha_{max}, \alpha_{min} = \min\{\phi_j \pm \sin\frac{R}{\rho_j}\}$$
$$,\alpha_{max} = \max\{\phi_j \pm \sin\frac{R}{\rho_j}\} \quad (8)$$

If $\mathcal{O}$ includes m obstacle points, then $\mathcal{C}_{obst} = \bigcup_{1\leq j\leq m} \mathcal{C}_j$. Now, to select the desired steering angle, the sectors in $\mathcal{C}$ are classified as free or occupied. The j^{th} sector S_j is occupied if $L_j \leq R_{safe}$; otherwise it is free. Adjacent free sectors are grouped together to form gaps. Then, the gaps are classified as wide, medium, and narrow. Every gap edge is a candidate for the desired steering angle. A cost function is defined to aid in the selection process:

$$Cost = c_1(\gamma_{ref} - \beta_j) + c_2\beta_j. \quad (9)$$

The first term in equation (9) represents how close the desired steering direction is to the goal location and the second term represents how close the current steering direction is to the current robot heading.The angle of the gap edge that has the minimum cost is selected as $\gamma_{desired}$. The candidate of the desired steering angle is first considered within the wide gaps. If none is available the search is performed within the medium gaps. The final choice is the narrow gaps.

The final step of the algorithm is to determine the desired radius of curvature of the robot trajectory. Due to the kinematic constraints, the robot can not achieve the desired steering angle instantly. Instead, the robot follows a circular arc if the wheels' velocities are constant. The path from the initial configuration to the final configuration may intersect with $\mathcal{C}_{obst}(q(t_i))$ causing a collision. Therefore, using a radius of curvature that is a function of the surrounding obstacles is safer than using a fixed radius for all obstacle scenarios. Let S_0 be the sector that contains the local x axis and let $S_{desired}$ be the sector that contains the desired steering angle $\gamma_{desired}$. Let L_j^m be the distance between the obstacle point o_j and the reference point m shown in Fig. 3. The relationship between L_j and L_j^m is given by:

$$L_j^m = \sqrt{(L_j cos\beta_j + a)^2 + (L_j sin\beta_j)^2}, \quad (10)$$

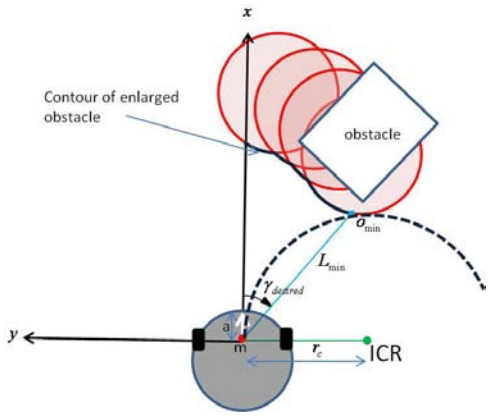

Fig. 3. Turning Radius Selection

where a is the distance between the robot reference point and the reference point m. Define L_{min} as the distance of the nearest obstacle point that exists anywhere between S_0 and $S_{desired}$. The turning radius r_c is chosen such that the trajectory passes through the point $(L_{min}, \gamma_{desired})$ as shown in Fig. 3. From the geometry, the turning radius is obtained as:

$$r_c = \frac{L_{min}}{2sin(\gamma_{desired})}. \quad (11)$$

To include a safety buffer, the turning radius is designed to pass through the point $(L_{min} - d_{safe2}, \gamma_{desired})$ instead. Also, the turning radius r_c saturates if it is greater than a threshold value r_{large}.

2.2. Neural Navigation Algorithm

A neural network is proposed to work in collaboration with the reactive navigation algorithm developed earlier to optimize the navigational capabilities of the overall system. While the navigational algorithm avoids obstacles, it does not contain any constraints on the length of the path to the target position. The neural network is incorporated into the system to minimize the length of the path taken. The neural network selects the optimum trajectory based on the obstacles information, target configuration, and the turning radius proposed by the navigational algorithm. The neural network outputs the most promising turning radius of the robot trajectory.

This paper uses a Diagonal Recurrent Neural Network (DRNN) which despite its simple structure, has the ability to adapt the learning rates such that the network convergence is guaranteed. Fig. 4 depicts the network architecture. For the neural network to achieve this objective, it needs to overcome few challenges. One of those challenges is that the 'correct' turning radius is not available. Hence, the training cannot be conducted in a supervised manner where a dataset is available to help the network form a map between the input data and the desired value. To overcome this obstacle, a hybird training scheme is proposed. First, the neural network will receive supervised training based on a sub-optimal dataset. Second, the neural network will be trained on-line to adjust its

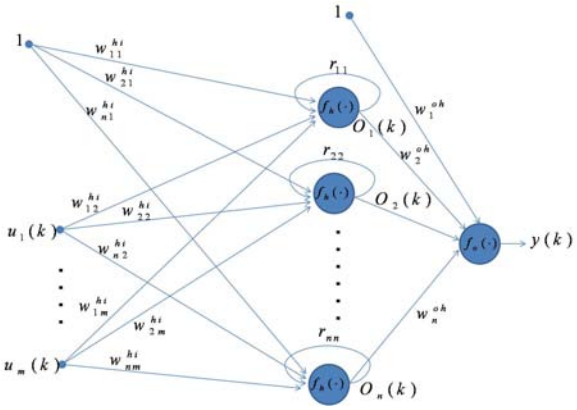

Fig. 4. Neural Network Architecture

weights in order to produce an optimum value based on an evaluation function. In the first training phase, the network is trained to map the input values to a desired value. The dataset is generated using the reactive navigation algorithm. The training is based on backpropogation and is done off-line. The purpose of this training phase is to provide an adequate initial set of weights for the next phase as opposed of starting phase 2 from random variables. In the second training phase, the optimum turning radius is unknown. However, the radius taken by the robot can be evaluated. The network receieves feedback about its performance based on an evaluation function described as:

$$J = \frac{1}{2}e^2 = \frac{1}{2}(e_x^2 + e_y^2) \tag{12}$$
$$e_x = x_t - x, e_y = y_t - y$$

where (x_t, y_t) is the target position. The derivative of the evaluation function with respect to the weights is given by:

$$\frac{\partial J}{\partial W} = -(e_x J_x + e_y J_y)\frac{\partial r}{\partial W} \tag{13}$$

where J_x and J_y are the sensitivities of the system and are given by:

$$J_x = \frac{\partial x}{\partial r} = sign\Big(\frac{x(t) - x(t-1)}{r(t) - r(t-1)}\Big) \tag{14}$$

$$J_y = \frac{\partial y}{\partial r} = sign\Big(\frac{y(t) - y(t-1)}{r(t) - r(t-1)}\Big) \tag{15}$$

$\frac{\partial r}{\partial W}$ is estimated online using Real Time Recurrent Learning (RTRL) [1]. The neural network weights are updated according to the gradient descent technique:

$$\Delta W = -\eta\frac{\partial J}{\partial W} \tag{16}$$

Subsituting eq.12 into 16 gives:

$$\Delta W = -\eta e\frac{\partial e}{\partial W} \tag{17}$$

The training phases are shown in Fig. 5 and 6.

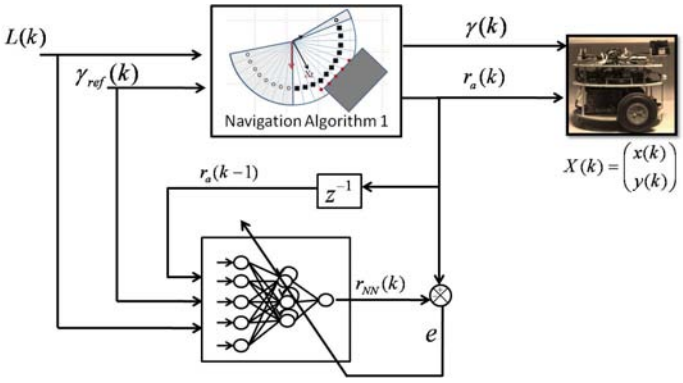

Fig. 5. Phase 1 *Training*

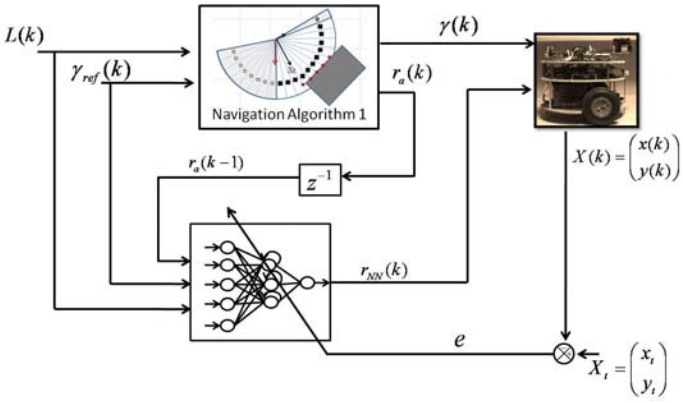

Fig. 6. Phase 2 *Training*

2.3. Adaptive learning and stability analysis

In order to maintain the system stability, we put a restriction on the learning rate using Lyapunov theorem. Define the lyapunov function $V(k) = \frac{1}{2}e^2(k) \geq 0$. To prove system stability, we need to show that $\Delta V = V(k+1) - V(k) \leq 0$. Hence,

$$\Delta V = \frac{1}{2}\Big(e^2(k+1) - e^2(k)\Big) \tag{18}$$

Subsitute $e(k+1) = e(k) + \Delta e(k)$ in eq.18:

$$\Delta V = e(k)\Delta e(k) + \frac{1}{2}\Delta^2 e(k) \tag{19}$$

$$= \Delta e(k)\Big(e(k) + \frac{1}{2}\Delta e(k)\Big) \tag{20}$$

$\Delta e(k)$ is given by:

$$\Delta e(k) = \Big[\frac{\partial e(k)}{\partial W}\Big]^T \Delta W \tag{21}$$

Subsituting eq.17 in 21 gives:

$$\Delta e(k) = -\eta e\Big\|\frac{\partial e}{\partial W}\Big\|^2 \tag{22}$$

In order to find $\frac{\partial e(k)}{\partial W}$,the chain rule is used:

$$\frac{\partial e}{\partial W} = \frac{\partial r}{\partial W}\Big[\frac{\partial e}{\partial x}\frac{\partial x}{\partial r} + \frac{\partial e}{\partial y}\frac{\partial y}{\partial r}\Big] \tag{23}$$

$$\frac{\partial e}{\partial x} = -\frac{e_x}{e}, \frac{\partial e}{\partial y} = -\frac{e_y}{e} \tag{24}$$

Hence eq 23 becomes:

$$\frac{\partial e}{\partial W} = -\frac{\partial r}{\partial W}\left(\frac{e_1 J_x + e_2 J_y}{e}\right) \tag{25}$$

Substituting 25 in 21, gives:

$$\Delta e(k) = -\frac{\eta}{e}\left\|\frac{\partial r}{\partial W}\right\|^2 (e_x J_x + e_y J_y)^2 \tag{26}$$

Substituting 26 in 20:

$$\Delta V = \eta\left\|\frac{\partial r}{\partial W}\right\|^2 (e_x J_x + e_y J_y)^2 * \left(-1 + \frac{\eta}{2e^2}\left\|\frac{\partial r}{\partial W}\right\|^2 (e_x J_x + e_y J_y)^2\right) \tag{27}$$

the term $\eta\|\frac{\partial r}{\partial W}\|^2(e_x J_x + e_y J_y)^2 \geq 0$, hence in order to have $\Delta V \leq 0$, η should be chosen as:

$$\eta \leq \frac{2e^2}{\|\frac{\partial r}{\partial W}\|^2 (e_x J_x + e_y J_y)^2} \tag{28}$$

3. Experimental Results

The performance of the proposed algorithm is verified over a variety of real unstructured indoor environments using an autonomous mobile robot platform [2]. The mobile robot platform is designed to operate in an indoor environment with a solid flat surface. A differential steering system is employed to generate forward and steered motion. The platform provides a rich computing environment consisting of a single board computer and a microcontroller. It is also equipped with a laser range sensor and ultrasonic sensors for obstacle detection as well as a compass and wheel encoders for localization. The laser range sensor is calibrated to scan the 200 degrees front view of the robot in 20 sectors with a 10 degrees angular resolution, The mobile robot platform is shown in Figure 7.The platform has a cylindrical structure with a 35cm diameter and approximately 30cm height. For

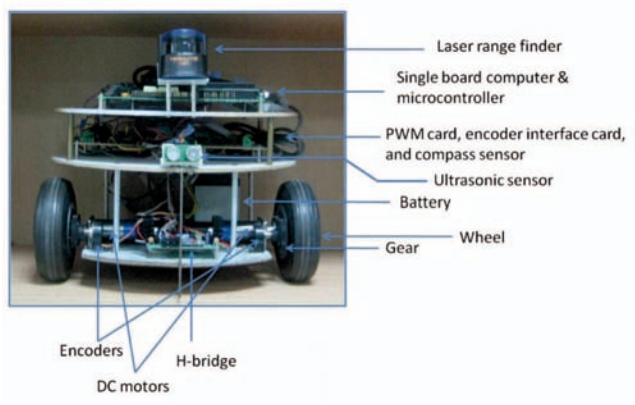

Fig. 7. Target and robot coordinates

all the testing scenarios, the data acquisition is performed with a sample time T=1s. The measured variables consist of the current robot position and orientation (x_r, y_r, θ) and the twenty-sector readings of the laser sensor.

In all experiments, the robots initial position is (0,0) while the goal position is at (1.5,-1.5). In the first experiment, an obstacle is placed along the robot direct path, which is the straight line that connects the robot's initial configuration to the target configuration ($\vec{u}_e$). There were also other obstacles surrounding the robot as shown in Figure 8. Fig. 8b depicts the trajectory obtained from the neural network based controller. The length of the trajectory is 2.4157 m. Figures 8c - 8e are snapshot of the intermediate robot parameters at different instances in time. The obstacles seen by the robot at each instant of time is shown in the Cartesian coordinates as black dots. The solid yellow line is an approximation to the obstacle contour. The polar histogram shows how each sector is classified to either free or occupied. The reference steering angle, γ_{ref}, is illustrated as a solid red line while the desired steering angle, $\gamma_{desired}$, is shown as a dashed green line.

In the second experiment shown in Fig. 9a, the robot successfully avoids the 2 obstacles, and drives its way to q_{target} as shown Fig. 9b. The robot velocities and control actions are shown in Fig. 9c and Fig. 9d respectively. The length of the trajectory is 2.3016 m and it takes 106 s to execute.

In the third experiment shown in Fig. 10a, the gap between the obstacles is reduced. The robot correctly identified the gap as navigable and went in between. However, the left side of the robot touched the obstacle. The trajectory is shown in Fig. 10b and the robot motion and control action are shown in Fig. 10c and Fig. 10d. The length of the trajectory is 2.1349 m and is completed in 90 s.

In the fourth experiment shown in Fig. 11a, the robot avoids the narrow gap by contouring the obstacles, discovers a blocked path, reverses direction and progresses to the target configuration. The trajectory is depicted in Fig. 11b and the robot velocities and control action are depicted in Fig. 11c and 11d. The length of the trajectory is 7.4457 m and the time it take is 227 s.

In the fifth experiment shown in Fig. 12a, the robot discovers the dead end and turns around the obstacles to reach q_{target}. The trajectory is depicted in Fig. 12b. The robot velocities and control action are depicted in Fig. 12c and Fig. 12d. The trajectory exhibited some fluctuations. The length of the trajectory is $5.4778m$ and is completed in $169s$.

There are several metrics that can be used to evaluate the performance of a navigation system [9]. The following performance metrics are used to evaluate the quality of the trajectory while considering the security or proximity to obstacles and the smoothness of the trajectory relative to the control effort.

1) Security Metric-1 (SM1): Mean distance between the robot and the obstacles through the entire mission measured by the laser sensor (20 sectors).
2) Security Metric-2 (SM2): Mean minimum-distance to obstacles. This is taken from the average of the lowest value of the laser sensor data (20 sectors).
3) Path length: distance traveled by the robot to ac-

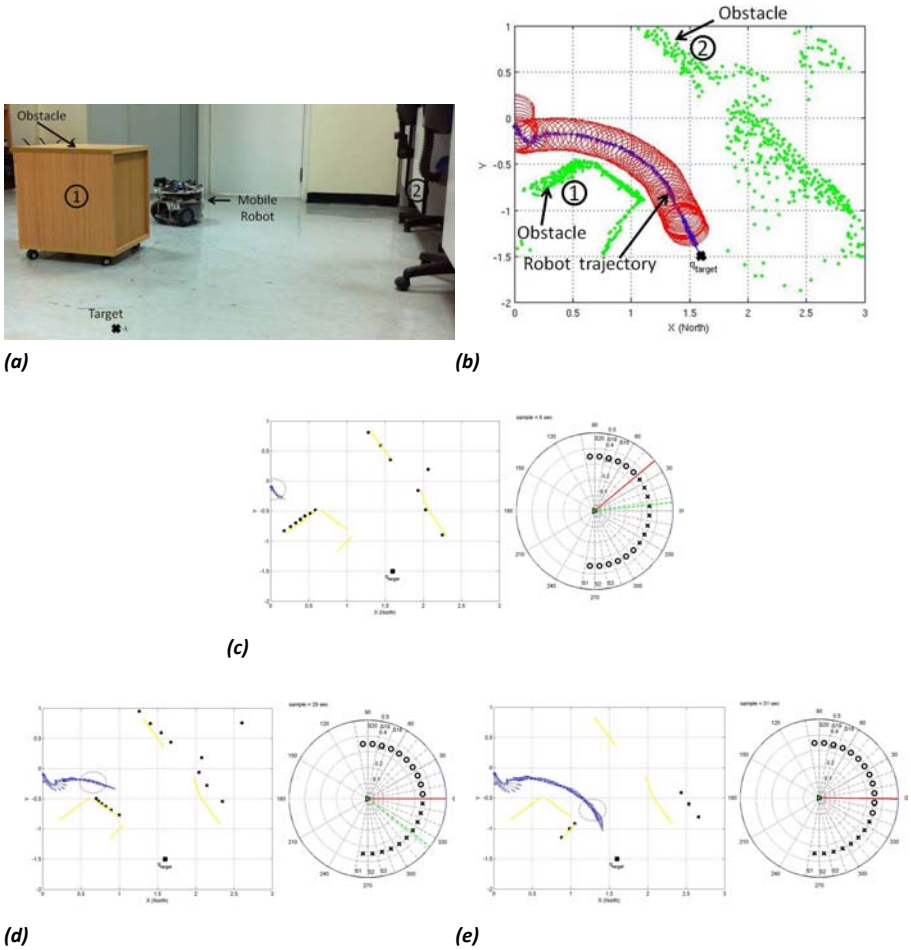

Fig. 8. (b) Robot trajectory using neural network algorithm; (c) Intermediate robot parameters at sample = 8 seconds; (d) Intermediate robot parameters at sample = 29 seconds; (e) Intermediate robot parameters at sample = 51 seconds.

complish the task from the initial position to the target position.

$$P_L = \sum_{i=1}^{n} \sqrt{(x_i - x_{i-1})^2 + (y_i - y_{i-1})^2} \quad (29)$$

where, $(x_i, y_i), i = 1, 2, \ldots, n$ are the n-point Cartesian coordinates of the robot along the given trajectory .

4) Time T_A: time taken to accomplish the task.

5) Smoothness of the trajectory relative to control effort.

$$TB_E = \sum_{i=1}^{n} k^2(x_i, y_i) \quad (30)$$

where $k(x_i, y_i)$ is the curvature at any point (x_i, y_i) across the trajectory.

$$k(x_i, y_i) = \frac{f''(x_i)}{[1 + (f'(x_i))^2]^{\frac{3}{2}}} \quad (31)$$

Table 1 summarizes the experimental results obtained from the 5 different scenarios. The results show that the neural navigation algorithm allows the robot to transit through narrow zones keeping a safe distance from the obstacles while generating smooth trajectories. In the neural navigation algorithm, a threshold of 0.5m was placed on the maximum distance that the robot can view from the laser sensor data. The deviation of the security metrics SM1 from this maximum value (0.5m) is relatively low, which means that the chosen routes passed always through an obstacle free area. The SM2 index gives also an idea about the risk taken by the robot through the different missions in terms of proximity to obstacles. The values of the TB_E index are also low, which is desirable, since the energy requirements are increased according to the increase in the curvature of the trajectory. Scenario 2 shows a very low TBE because the corresponding trajectory is straighter that the other scenarios. Fig. 9b shows a smaller change in the orientation during each control period, with consequent energy saving and less structural effort on the robot.

Tab. 1. Performance Metrics

Scenario	Performance Metric				
	SM1 (m)	SM2 (m)	P_L (m)	T_A (s)	TB_E
1	0.4688	0.1973	2.4157	82	2.3186e-02
2	0.4563	0.1588	2.3016	106	9.2964e-05
3	0.4439	0.1474	2.7349	90	6.8655e-02
4	0.4478	0.1890	5.4778	169	1.1589e-01
5	0.4727	0.2704	7.4458	227	2.4902e-02

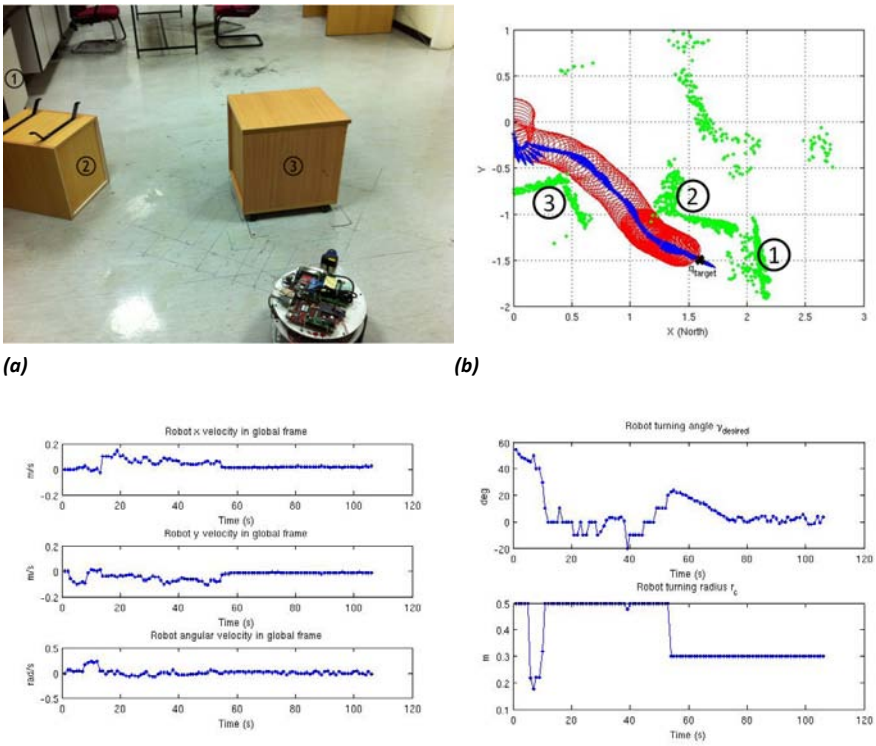

Fig. 9. (a) shows the robot initial position and the surrounding obstacles; (b) shows the obstacle points in green, the area occupied by the robot at each instance in time in red, and the reference point trajectory in blue; (c) describes the robot velocities; (d) describes the robot control vector.

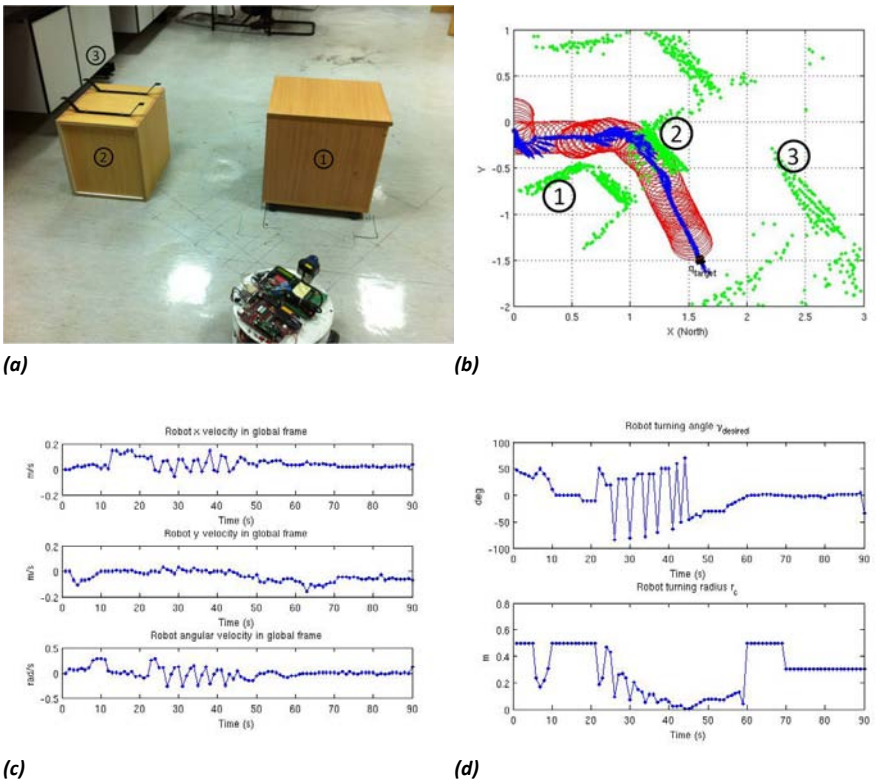

Fig. 10. (a) shows the robot initial position, target position, and the surrounding obstacles; (b) shows the obstacle points in green, the area occupied by the robot at each instance in time in red, and the reference point trajectory in blue; (c) describes the robot velocities; (d) describes the robot control vector.

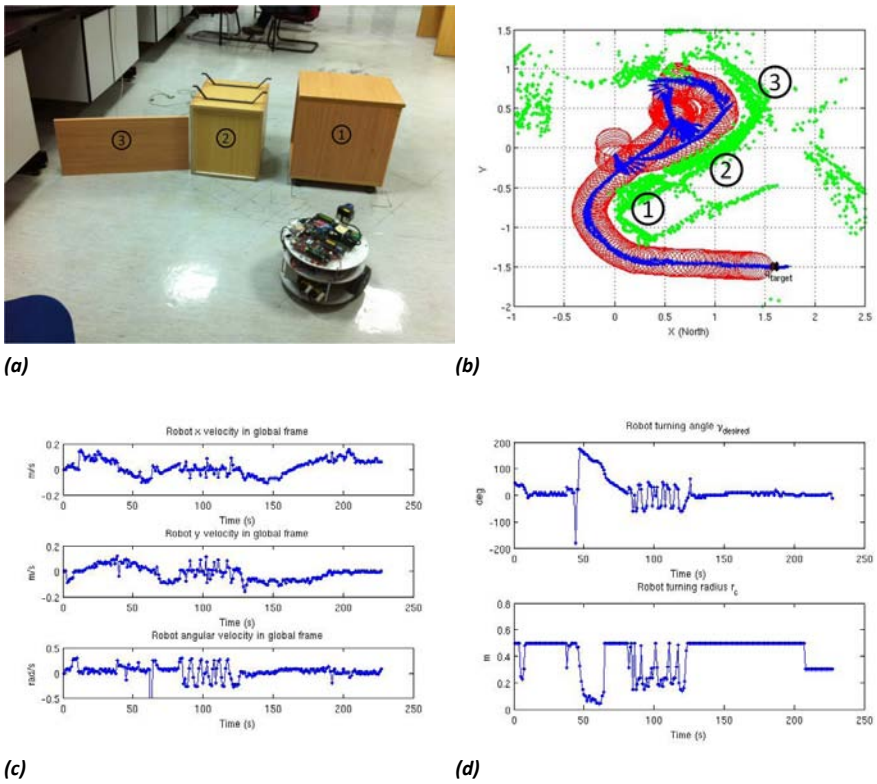

Fig. 11. (a) shows the robot initial position and the surrounding obstacles; (b) shows the obstacle points in green, the area occupied by the robot at each instance in time in red, and the reference point trajectory in blue; (c) describes the robot velocities; (d) describes the robot control vector.

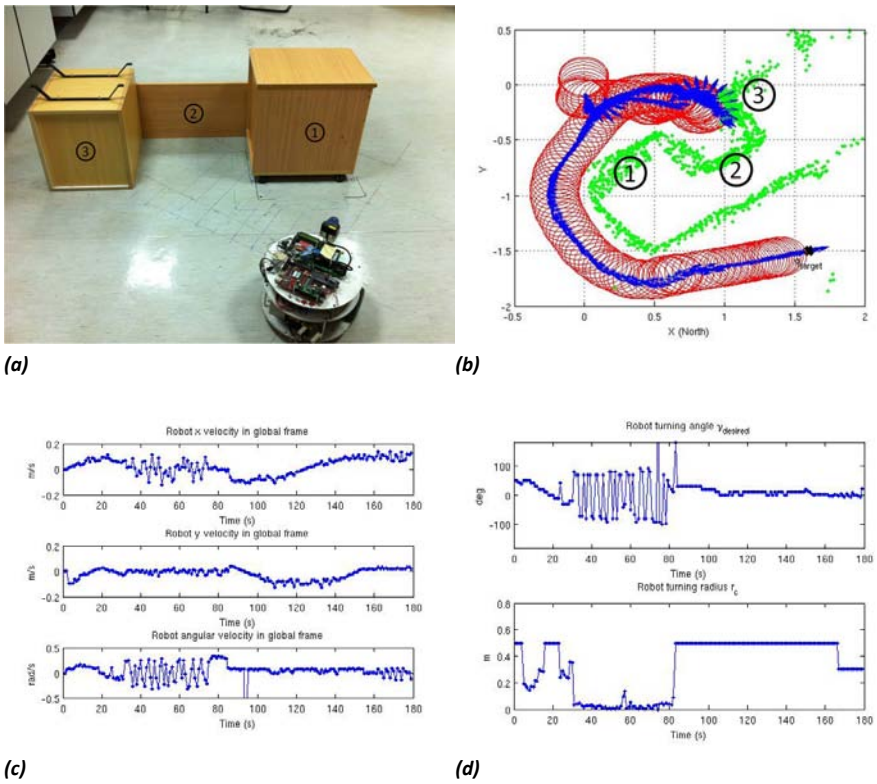

Fig. 12. (a) shows the robot initial position and the surrounding obstacles; (b) shows the obstacle points in green, the area occupied by the robot at each instance in time in red, and the reference point trajectory in blue; (c) describes the robot velocities; (d) describes the robot control vector.

4. Conclusions

The paper presents two reactive navigation algorithms for a wheeled mobile robot under non-holonomic constraints and in unknown environments. The mobile robot travels to a pre-defined goal position safely and efficiently without any prior map of the environment. The first method is based on a reactive navigation algorithm which incorporates the dimensions and shape of the robot to determine the set of all possible collision-free steering angles. The steering angle that falls in the widest gap and is closest to the target is selected. The algorithm also takes into account the non-holonomic constraints of differentially steered robots by computing circular trajectories with adaptive radius of curvature. The second navigation algorithm introduces a neural network based reactive navigation. The algorithm aims to generate an optimized path by using a user-defined objective function which minimizes the traveled distance to the goal position while avoiding obstacles. To this end, a diagonal recurrent neural network (DRNN) has been employed to achieve the necessary generalization capability across a variety of indoor environments. The network is trained through off-line learning followed by an on-line learning algorithm with guaranteed convergence. The performances of the algorithms are verified over a variety of real unstructured indoor environments using an autonomous mobile robot platform. The results demonstrated that the algorithm is capable of driving the robot safely through different obstacle arrangements.

AUTHORS

Mariam Al-Sagban* – American University of Sharjah, UAE, e-mail: g00006931@aus.edu.
Rached Dhaouadi* – American University of Sharjah, UAE, e-mail: rdhaouadi@aus.edu.

*Corresponding author

REFERENCES

[1] M. Al-Sagban and R. Dhaouadi, "Neural-based navigation of a differential-drive mobile robot", *12th International Conference on Control Automation Robotics Vision*, 2012, 353–358.

[2] M. Al-Sagban. "Autonomous robot navigation based on recurrent neural networks". Master's thesis, American University of Sharjah, 2012.

[3] V. Castro, J. Neira, C. Rueda, J. Villamizar, and L. Angel, "Autonomous navigation strategies for mobile robots using a probabilistic neural network (pnn)", *33rd Annual Conference of the IEEE Industrial Electronics Society*, 2007, 2795–2800.

[4] X. Chen and Y. Li, ""smooth formation navigation of multiple mobile robots for avoiding moving obstacles"", *International Journal of Control, Automation, and Systems*, vol. 4, no. 4, 2006, 466–479.

[5] W. Chung, L.-C. Fu, and S.-H. Hsu. "Motion control". In: *"Springer Handbook of Robotics"*, 133–159. 2008.

[6] D. Janglová, "Neural networks in mobile robot motion", *Inernational Journal of Advanced Robotic Systems*, vol. 1, no. 1, 2004, 15–22.

[7] L. Kavraki, P. Svestka, J.-C. Latombe, and M. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces", *IEEE Transactions on Robotics and Automation*, vol. 12, no. 4, 1996, 566 –580.

[8] O. Khatib, "Real-time obstacle avoidance for manipulators and mobile robots", *International Journal of Robotics Research*, vol. 5, no. 1, 1986, 90–98.

[9] N. D. Munoz, J. A. Valencia, and N. Londono, "Evaluation of navigation of an autonomous mobile robot", *Proceedings of the 2007 Performance Metrics for Intelligent Systems Workshop, Maryland USA*, 2007, 15–21.

[10] H. T. Trieu, H. T. Nguyen, and K. Willey, "Advanced obstacle avoidance for a laser based wheelchair using optimised bayesian neural networks", *30th Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, 2008, 3463 –3466.

[11] J. Yen and N. Pfluger, "A fuzzy logic based extension to payton and rosenblatt's command fusion method for mobile robot navigation", *IEEE Transactions on Systems, Man and Cybernetics*, vol. 25, no. 6, 1995, 971 –978.