Journal of Artificial Intelligence
ISSN: 2229–3965 & E-ISSN: 2229–3973, Volume 3, Issue 1, 2012, pp.-49-55
Available online at http://www.bioinfo.in/contents.php?id=71

A COLLISION-FREE ALGORITHM OF A POINT-MASS ROBOT USING NEURAL NETWORKS

BIBHYA SHARMA*, AVINESH PRASAD AND JITO VANUALAILAI

School of Computing, Information & Mathematical Sciences, University of the South Pacific, Suva, Fiji
*Corresponding Author: email- sharma_b@usp.ac.fj

Received: September 05, 2011; Accepted: January 15, 2012

Abstract- In this paper an artificial neural network approach is proposed to solve the findpath problem of a mobile point-mass robot which is required to move safely to a prescribed target in a priori known workspace. A novel velocity algorithm is designed to ensure that the robot moves towards its goal at all times and remains there once reached. Our method in the construction of a collision-free path for the robot amongst a randomized number of arbitrarily configured obstacles is based on learning via the multilayer perceptron. Two neural networks are used to determine the direction and angle of turn of the robot so that it moves in the free space of the workspace while avoiding the obstacles in its path. The training data for the neural network are obtained using computer simulations where the initial path is traced by the user. Finally, simulations using point-mass robot and an anchored 2-link (RP) manipulator highlight the stabilizing controls of the neural network approach.
Keywords- Point-mass robot, neural network, motion planning and control, findpath, and multilayer perceptron.

Citation: Bibhya Sharma, Avinesh Prasad and Jito Vanualailai. (2012) A Collision-free Algorithm of a Point-Mass Robot using Neural Networks. Journal of Artificial Intelligence, ISSN: 2229–3965 & E-ISSN: 2229–3973, Volume 3, Issue 1, 2012, pp.-49-55.

Introduction

Motivational Work

The recent past has witnessed numerous research on machine learning and its applicability on improving the operational capabilities of mobile robots [1–5]. Herein one of the areas that has gained huge support and attention is the work on the findpath problem or the piano mover's problem, which is a geometric problem of finding a collision-free path from an initial configuration to a predefined final configuration, usually in a workspace cluttered with obstacles [6, 7]. This can also be considered as a robot path planning problem when contextualized with mobile robots. In terms of the design and development of autonomous robots, it is the ability of a mobile robot to plan and execute collision free motions within in either fully known or partially known environments [1, 8, 9]. The environments may be very harsh, hazardous or even inaccessible to humans [10] or could involve laborious repetitions. Ideally, robots must be able to fully recognize and understand their workspace or environment so that they can navigate to their goal configurations, satisfying the safety, cost and time constraints tagged to the system. There are many ways this can be realized. The literature has algorithms that can be seen addressing the motion planning and control of mobile robots. This includes the artificial potential field (APF) methods, graph search techniques and road maps (voronoi diagrams, visibility and accessibility graphs) and neural network models [9–11].

The pioneer work on motion planning and control of robots via artificial potential fields was carried out by Khatib in [12]. Since then algorithms based on the artificial potential framework appear abundantly in the literature [9, 13–16]. While this is due to easier analytic representation of system singularities and inequalities, better processing speed and its simplicity and elegance, the meth-

od inherits the problem of traps or local minima. In addition, APF-based smooth controllers of autonomous nonholonomic systems have to prove stabilization (see Brockett's Theorem in [6]). Work in literature show merely a Lyapunov stable system [10, 17].
The graph search techniques and the road maps establish collision -free trajectories by searching for graphs or maps formed out of straight lines via vertices of obstacles or patches of free space decomposed into geometrical primitives. Work using the technique is been continuously carried out and cited from literature [18–21]. Although the algorithms are elegant, they are computationally intensive and can suffer from the problem of closeness [22].
In more recent times, there has been a paradigm shift to using neural networks-based approach for motion planning and control of robots in obstacle-ridden environments. Some relevant results are outlined in the next section.

Background on Neural Networks

The work of McCulloch and Pitts in 1943 [23] is considered a landmark result in the literature of artificial neural networks. They presented neurons as models of biological
neurons and as conceptual components for circuits that could perform computational tasks. The basic model of the neuron is founded upon the functionality of a biological neuron. This has been followed by the development of various types of ANN architectures such as, single/multi-layer perceptron, Kohonen's self-organising map (SOM) and Hopfield-type network, etc. In the last two decades, the artificial neural networks have been applied in many areas of robotics, amongst which the steering, path-planning and control of autonomous robot vehicles garners monumental attention [24]. This area is not only challenging but it inherently harbors real-world practicabilities and possibilities.
The application of ANNs can be grouped into two classes: optimization and associative retrieval/classification. Therefore, most robot problems can be formulated as one of the two classes. For example, stereo vision for task planning, autonomous robot path planning, and position control can be formulated as optimization problems. Several models of ANNs such as: single-layer feedback neural networks, competitive learning neural networks, and multi-layer feed-forward neural networks, etc. have been utilized to solve motion planning and control of different types of mobile robots [1–5, 24, 25].
In the interest of brevity, we present a few relevant work from literature. Bekey & Goldberg in [25] utilized competitive type of neural networks for robot navigation. Moreno et al. in [2] utilized a Multilayer Static Neural Network to consider the inverse kinematics problem of a two link planar robot arm. An iterative technique known as
the Levenberg-Marquardt algorithm was used for the learning. Chohra et al. in [5] used the supervised Gradient Backpropagation for robot navigation in partially known environment. Yang & Meng in [3] proposed a neural network for the real-time collision-free motion planning of mobile robots in a dynamic environment of no prior knowledge. Pham & Sahin in [4] designed an Internal Model Control system for a planar two-degree-of-freedom robot. The system consisted of a forward internal neural model of the robot, a neural controller and a conventional feedback controller, wherein training was by backpropagation. Janglova in [1] dealt with a path planning
and intelligent control of an autonomous robot in partially structured environment using two neural networks, one to determine the free space and the other to find a safe direction.

Contributions

This paper provides five major contributions:
Velocity Algorithm. Design of a new velocity algorithm for the point-mass robot. The velocity algorithm is significantly different from the ones in literature. Normally constant velocities are used; however, the robot needs to stop after it has achieved its aim. This stop should not be sudden by a truncation of velocity, rather the robot should slow down its motion and then come to rest. The velocity algorithm and a target designed for the robot ensures a safe stop at the target and also ensures that the robot remain there.
Obstacle Avoidance Scheme. For this, we only consider the obstacle closest to the point-mass robot enroute to its target. Therefore, our obstacle avoidance scheme is more simple compared to, for example, the avoidance schemes utilized in the artificial potential methods where all the obstacles are considered in parallel [9, 10]..
Multilayer Perceptron (MLP). While the new velocity algorithm is sufficient to control the motion of the robot, the inclusion of obstacles garners the need to control the direction of motion. For this, two MLPs have been used in this paper: (1) that indicates whether the robot should turn or not, and (2) that determines the precise angle of turning.
Training Data. The training data for the neural network are obtained using computer simulations where the initial path is traced by the user. The data are not obtained from real-life experiments, such as physically driving or maneuvering a real robot. Suppose, we obtain the data by driving a robot (or from using a remote), there is a possibility that we can make a mistake and the robot can collide with an obstacle. This will damage the robot. On the other hand, the procedure in this research provides data in a more simplified and controlled environment. Any slight deviations can be easily and quickly readjusted.
Polar Form. The neural network operates using input variables (R, θ, ϕ) in a polar form. This greatly implifies the algorithm which can be made complex if one is using the positional variables. The training of the network becomes easy.

Organization

This paper is organized as follows. In section 2.0, we give the definition of a point mass robot and derive its kinematic model. The main objective of this paper is given in section 3.0. The motion planning and control problem of the point mass in obstacle free workspace is described in section 4.0, together with introducing a new velocity algorithm and its associated claims. In section 5.0, we have utilized the neural technique to control the motion of the robot in a workspace cluttered with obstacles of arbitrary sizes. In section 6.0, training of the neural network is described with details of the updating rules. We use a supervised multilayer perceptron to model the turning angle of the robot. Stability analysis is carried out in section 7.0 followed by some examples in section 8.0. In section 9.0, we apply this idea of target convergence and obstacle avoidance to control the motion of a planar robot (RP) arm. Finally, in

Section 10.0 the paper closes with a concluding remark on its contributions and lists the future work.

Modelling the Point-Mass Robot

We start with the following definition of a workspace:

Definition 1: The workspace is a fixed, closed and bounded rectangular region for some η1 > 0 and η2 > 0. Precisely, the workspace is the set WS = {(z1, z2) ∈: R2 : 0 ≤ z1 ≤ η1, 0 ≤ z2 ≤ η2}. The boundaries of the region are defined as follows: (a) Left Boundary: B1 = {(z1, z2) ∈: R2 : z1 = 0}; (b) Lower Boundary: B2 = {(z1, z2) ∈: R2 :z2 = 0}; (c) Right Boundary: B3 = {(z1, z2) : R2 : z1 = η1}; (d) Upper Boundary:B4 = {(z1, z2) ∈ R2 : z2 = η2}.

Definition 2: Let A be a point-mass robot in the z1-z2 plane, positioned at (x,y) with a circular protective region of radius of rt, and moving with a velocity of v in the direction
θ at time t ≥ 0. Precisely, it is a set A = {(z1, z2) : R2: (z1 − x)2 + (z2 − y)2 ≤ rt2}

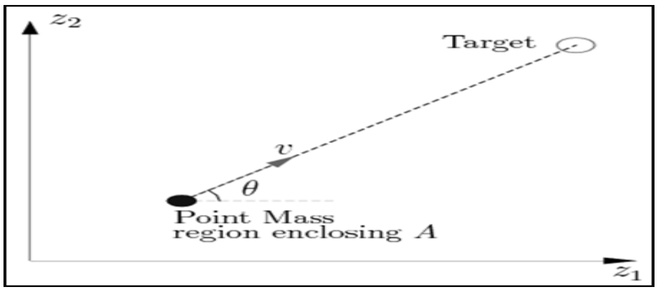


Fig.1- Schematic representation of a point-mass robot A in the z1-z2 plane.

With reference to Fig. (1), one can easily derive the following differential (kinematic) equations, assuming the initial condition at $t = t_0 \geq 0$:

$$\left.\begin{aligned} \dot{x} &= v\cos\theta \\ \dot{y} &= v\sin\theta \\ x_0 &= x(t_0),\ y_0 = y(t_0) \end{aligned}\right\} \qquad (1)$$

In the following sections, we will explore the concept of artificial neural network in the algorithm of the robot motion planning and control problem.

Main Objective: Problem Statement

A point-mass robot moving in WS is represented by A. Let FO1, FO2, …, FOm be stationary obstacles randomly distributed in WS. Assume that both the geometry and the location of A and FO1, FO2, …, FOm is a prior known. The problem statement is:

Given an initial position and orientation of A in WS, generate a path specifying a contiguous sequence of positions and orientations of A avoiding collision with FOi for i = 1, 2, …, m, starting at the initial position and orientation, and terminating at the goal position.

Motion Planning and Control (MPC)

In our MPC problem, we want the point-mass robot A to start from an initial position, move towards a target and finally converge at the center of the target as shown in Fig. (1). We therefore, require for A a predefined target:

Definition 3: The target for the point-mass robot A is a disk of center (p1, p2) and radius rT . Precisely, it is a set

T = {(x, y) : R2 : (x − p1)2 + (y − p2)2 ≤ rT2 }.

Velocity algorithms in literature include mostly constant (either maximum or optimal) velocities which truncate at the goal configuration. However, a sudden switch or truncation of the velocity to force the robot to stop will require infinite accelerations and in turn infinite torques and can also cause physical damage to the robot. Thus we develop a more practical velocity algorithm which is dependent on the initial and final positions of the robot.

The new velocity algorithm proposed is

$$v(t) = |v_0| \frac{(p_1 - x(t))^2 + (p_2 - y(t))^2}{(p_1 - x(0))^2 + (p_2 - y(0))^2} \qquad (2)$$

where v_0 is the initial velocity of the robot *A* at *t* = 0.

With the new velocity algorithm, we have the following Lemma.

Lemma 1. In the absence of obstacles in the workspace, there is a linear movement of the point mass robot from the initial position to the target position.

Proof. The linear movement can be proved by showing that $\dot{\theta}$ = 0. To see this, note that

$\theta = \operatorname{atan2}(p_2 - y(t), p_1 - x(t))$.

Then (upon suppressing *t*)

We can make the following claims from the new velocity algorithm and Lemma 1:

$$\begin{aligned} \dot{\theta} &= \frac{1}{1 + \frac{(p_2 - y)^2}{(p_1 - x)^2}} \cdot \frac{-(p_1 - x)\dot{y} + (p_2 - y)\dot{x}}{(p_1 - x)^2} \\ &= \frac{-(p_1 - x)\dot{y} + (p_2 - y)\dot{x}}{(p_1 - x)^2 + (p_2 - y)^2} \\ &= \frac{-d\cos\theta\sin\theta + d\sin\theta\cos\theta}{(p_1 - x)^2 + (p_2 - y)^2} \\ &= 0 \end{aligned}$$

Claim 1. Given an initial position (x(0), y(0)) ≠ (p1, p2) of the robot in *WS*, if the velocity governed by (2) is applied, the robot *A* will move straight towards its target *T*.

Proof. Let $d = \sqrt{(p_1 - x(t))^2 + (p_2 - y(t))^2}$ be the Euclidian distance between the target *T* and the point-mass robot *A* at any time *t* ≥ 0. Then the time derivative of *d* is

$$\begin{aligned} \dot{d} &= \frac{-1}{d}\left[(p_1 - x)\dot{x} + (p_2 - y)\dot{y}\right] \\ &= \frac{-1}{d}\left[(p_1 - x)v\cos\theta + (p_2 - y)v\sin\theta\right] \\ &= \frac{-1}{d}\left[(p_1 - x)v\frac{p_1 - x}{d} + (p_2 - y)v\frac{p_2 - y}{d}\right] \\ &= \frac{-v}{d^2}\left[(p_1 - x)^2 + (p_2 - y)^2\right] \\ &= \frac{-v}{d^2}(d^2) \\ &= -v < 0 \end{aligned}$$

which means that as the point-mass robot *A* moves, *d* decreases, thus *A* is moving towards its target *T*. Lemma 1 shows that it is doing so in a straight line.

Claim 2. As the point-mass robot moves towards its target *T*, the velocity, *v* will decrease, as $(x(t), y(t)) \to (p_1, p_2)$.

Proof.

This shows that *v* decreases as the point-mass robot *A* moves towards its target *T*.

Claim 3. At the center of the target *T*, the point-mass robot *A* will stop and remain there as $t \to +\infty$.

$$\begin{aligned}
\dot{v} &= -2|v_0|\frac{(p_1-x)\dot{x}+(p_2-y)\dot{y}}{(p_1-x(0))^2+(p_2-y(0))^2} \\
&= -2|v_0|\frac{(p_1-x)v\cos\theta+(p_2-y)v\sin\theta}{(p_1-x(0))^2+(p_2-y(0))^2} \\
&= -2|v_0|\frac{(p_1-x)v\frac{p_1-x}{d}+(p_2-y)v\frac{p_2-y}{d}}{(p_1-x(0))^2+(p_2-y(0))^2} \\
&= \frac{-2v|v_0|}{d}\frac{(p_1-x)^2+(p_2-y)^2}{(p_1-x(0))^2+(p_2-y(0))^2} \\
&= \frac{-2v}{d}(v) \\
&= \frac{-2v^2}{d} < 0
\end{aligned}$$

Proof. At the center of the target *T*,

$$v = |v_0|\frac{(p_1-p_1)^2+(p_2-p_2)^2}{(p_1-x(0))^2+(p_2-y(0))^2} = 0$$

Injection of Stationary Obstacles

Let us fix $i > 0$ stationary obstacle within workspace *WS*.

Definition 4. The *m*th stationary obstacle is a disk-shaped obstacle with center $(o_{l1}, o_{l2})$and radius ro_l. Precisely, the *l*th stationary obstacle is the set

$$FO_l = \{(z_1, z_2) \in: R^2 : (z_1 - o_{l1})^2 + (z_2 - o_{l2})^2 \le ro_j^2\},$$

for $l = 1, 2, \dots, q$.

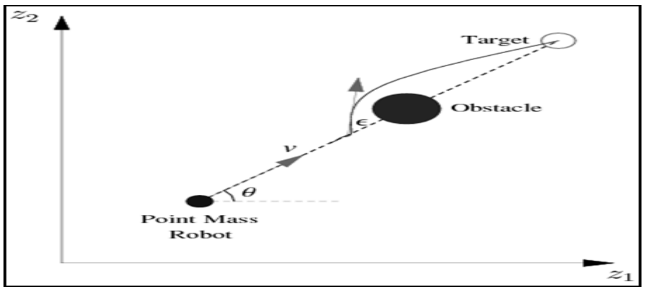


Fig.2- Schematic representation of the obstacle avoidance of A in the z1-z2 plane.

In order for the point-mass robot *A* to avoid the stationary obstacles, we need to make the following adjustment to the ODE in system (1):

$$\left.\begin{aligned}
\dot{x} &= v\cos(\theta+\varepsilon) \\
\dot{y} &= v\sin(\theta+\varepsilon) \\
x_0 &= x(t_0),\ y_0 = y(t_0)
\end{aligned}\right\} \quad (3)$$

where ε determines the direction in which the point-mass robot *A* will turn to avoid the obstacles. The inclusion of ε is explained in Fig (2). If $\varepsilon > 0$, the point mass will turn left; if $\varepsilon < 0$, it will turn right; and if $\varepsilon = 0$, it will move straight towards target. Thus controlling the value of ε will enable the robot to avoid obstacles and reach its target safely.

We will use a neural networks-based approach to control the motion of the point-mass robot A amongst a prior known stationary obstacles in a structured and bounded workspace *WS*. Our approach basically deploys two neural networks in order to solve the findpath problem. The first neural network will be used to find the direction (left or right) that the robot must turn to avoid collision with the nearest obstacle, while a second one will be used to determine the angle by which the robot should turn.

Artificial Neural Network (ANN) Technique

Our method in the construction of a collision-free path for a point-mass robot amongst stationary obstacles is based on a standard multilayer perceptron (see Fig. (3)).

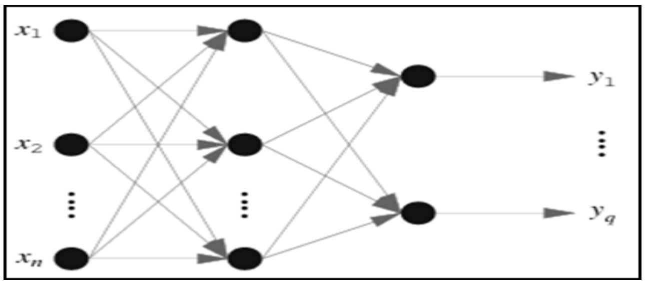


Fig.3- Architecture of a multilayer perceptron with inputs (x1, x2, . . . , xn) and outputs (y1, y2, . . . , yq).

ANNs are made up an input layer, hidden layer(s) and an output layer. The input layer receives data from outside the network, the output layer sends data out of the network, while for the hidden layer, the input and output signals remain within the network. In this research, we shall use only one hidden layer in the network while the number of neurons in that hidden layer will be determined from the training of the network. Fig. (4) shows a summary of our proposed methodology for the collision-free algorithm, where R, ϕ and θ are shown in Fig. 5.

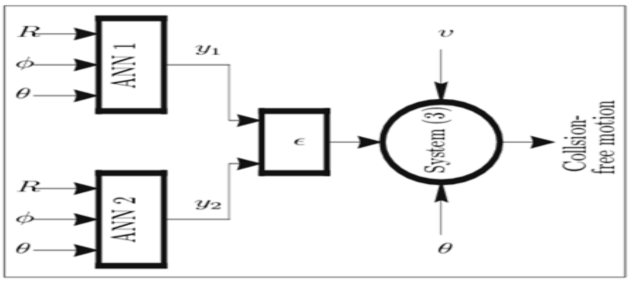


Fig.4- of the proposed methodology for the collision-free algorithm.

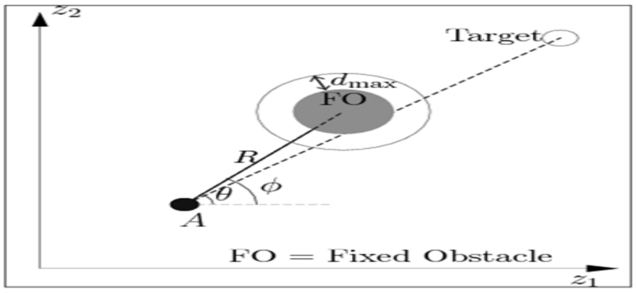


Fig.5- Schematic representation of the collision avoidance with reference to new parameters R and dmax

Modelling using ANN

Let R be the distance from the robot to the circumference of the nearest stationary obstacle and ϕ be the angular position of this obstacle center relative to the robot position (see Fig. (5)). To model the motion of the robot with ANN we assume that our collision avoidance scheme depends entirely on R, ϕ and θ.

We enact the following rules:

- Rule 1: R > 0.
- To avoid collision with the nearest stationary obstacle.
- Rule 2: If robot is approaching the nearest obstacle, it should change direction if R < dmax, where dmax is the maximum distance from the nearest obstacle at which the robot should turn to avoid collision.
- Rule 3: If R < dmax, the robot should turn right if $\theta \geq \phi$, otherwise turn left.
- This is to ensure that it follows the shortest path.
- Rule 4: If $|\theta - \phi|$ is small, the magnitude of the turning angle is large but small if $|\theta - \phi|$ is large.

We use two multi-layer perceptron ANNs. For both ANN's the input are $x_1 = R$, $x_2 = \phi$ and $x_3 = \theta$, and the outputs:

- For ANN1: The output is y_1 which determines the direction the point-mass should turn. -1 = turn right, 1 = turn left and 0 = no turning.
- For ANN2: The output is y_2 which determines the angle the point-mass should turn.
- It follows that $\varepsilon = y_1 y_2$.

Training the Network

Learning takes place when the network is trained by establishing the weighted connections between the input neurons and output neurons via the hidden neurons. Weights are continuously modified until the neural network is able to predict the outputs from the given set of inputs within an acceptable user-defined error level.

Let $w_{ij}^{(1)}(p)$ be the weight from the ith input neuron to the jth hidden neuron and $w_{jk}^{(2)}(p)$ b be the weight from the jth hidden neuron to the kth output neuron at iteration p. We use the standard backpropagation training algorithm which was first described by Rumelhart and McClelland in 1986 [26]. The weights are updated as follows:

$$\begin{aligned} w_{jk}^{(2)}(p+1) &= w_{jk}^{(2)}(p) + \eta\delta_k^{(2)}(p)\cdot\Phi\left(\sum_{i=1}^{n} w_{ij}^{(1)}(p)x_i\right) \\ w_{ij}^{(2)}(p+1) &= w_{ij}^{(2)}(p) + \eta\delta_j^{(1)}(p)\cdot x_i \end{aligned}$$

where

$$\begin{aligned} \delta_k^{(2)}(p) &= (d_k - y_k(p))\cdot\Phi'\left(\sum_{j=1}^{m}\sum_{i=1}^{n} w_{ij}^{(1)}(p)w_{jk}^{(2)}(p)x_i\right) \\ \delta_j^{(1)}(p) &= \Phi'\left(\sum_{i=1}^{n} w_{ij}^{(1)}(p)x_i\right)\cdot\sum_{k=1}^{q} w_{jk}^{(2)}(p)\delta_k^{(2)}(p) \end{aligned}$$

Here n is the number of inputs (n = 3 in our case), m is the number of neurons in the hidden layer, η is the learning rate and $\Phi(.)$ is the activation function. For ANN1, we use the hyperbolic tangent activation functions

$$\Phi_1(X_j) = \tanh(X_j) \quad \text{and} \quad \Phi_1(Y_k) = \tanh(Y_k)$$

and for the ANN2, we use the sigmoid activation functions

$$\Phi_2(X_j) = \frac{1}{1+e^{-X_j}} \quad \text{and} \quad \Phi_2(Y_k) = \frac{1}{1+e^{-Y_k}}$$

$$\text{where } X_j = \sum_{i=1}^{n} w_{ij}^{(1)}(p)x_i \text{ and } Y_j = \sum_{j=1}^{m} w_{jk}^{(2)}(p)X_j.$$

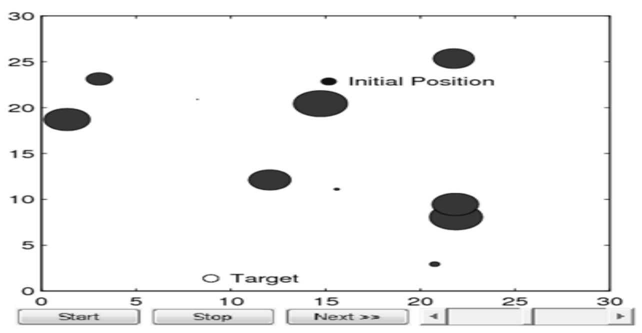


Fig.6- The Training environment.

The network is trained in the Matlab environment shown in Fig. (6) where the point-mass is driven manually and ε is controlled manually using the slider. This training is done at a low speed so that we can control the motion efficiently. Ten sets of data are obtained, in each case the size and position of obstacles and the initial and target positions are chosen randomly. These data are then fed into the networks which are trained using backpropagation.

We use the slider (bottom right of Figure 6) to control the motion of the point-mass robot and guide it to its target. The goal of the training process is to find the set of weight values that match the output of the neural network with the actual target values as closely as possible.

When the slider is kept at the center, the value of ε is zero so the robot moves straight (without turning) to the target. Moving the slider to the right will increase the value of ε and thus turning the robot to the right and moving the slider to the left will decrease the value of ε so that robot turns to the left.

Analysis: Asymptotic Stability

Stability analysis of our dynamic model is carried out via the Lyapunov's Second Method. The method provides one of the most powerful means of analyzing nonlinear systems because of the qualitative information it is able to provide concerning the stability of an equilibrium state of a nonlinear dynamical system.

Theorem 1. Let $x_e = (p_1, p_2) \neq (x(0), y(0))$ be the equilibrium point of system (3). Then x_e is a global asymptotic stable equilibrium point of system (3).

Proof. Consider the Lyapunov function

$$L(x,y) = \left(\dot{x}^2 + \dot{y}^2\right)/2 = v^2/2,$$

which has the following properties:

$L(x, y)$ is continuous and has first partial derivatives in the neighborhood of the equilibrium point (p_1, p_2) of system (3);

$L(p_1, p_2) = 0$ since $v = 0$ at the target;

$L(x, y) > 0$ for all $(x, y) \neq (p_1, p_2)$;

$$\dot{L}(x,y) = \frac{-2v^2 \,|\, v_0 \,|\, \sqrt{(p_1 - x)^2 + (p_2 - y)^2}}{(p_1 - x(0))^2 + (p_2 - y(0))^2} < 0$$

for all $(x, y) \neq (p_1, p_2)$ and $(x(0), y(0)) \neq (p_1, p_2)$;

$$\dot{L}(p_1, p_2) = 0$$

since $v = 0$ at the target.

Hence, it can be concluded that (p_1, p_2) is an equilibrium point of system (3), and $L(x, y)$ is a Lyapunov function that intrinsically guarantees its asymptotic stability.

Computer Simulations

In this section, we demonstrate the simulation results for the point -mass robot navigating from an initial to a final configuration in a constrained workspace cluttered with randomly fixed obstacles. The nonlinear controllers v, ε are simulated to generate feasible robot trajectories, as seen in Fig. (7). With the initial conditions and final positions selected randomly, the control laws ensured a collision-free trajectories and nice convergence of the system state to the equilibrium state, whilst satisfying all the underlying constraints. For the numerical integration of system (3), a fourth order Runge-Kutta method is utilized.

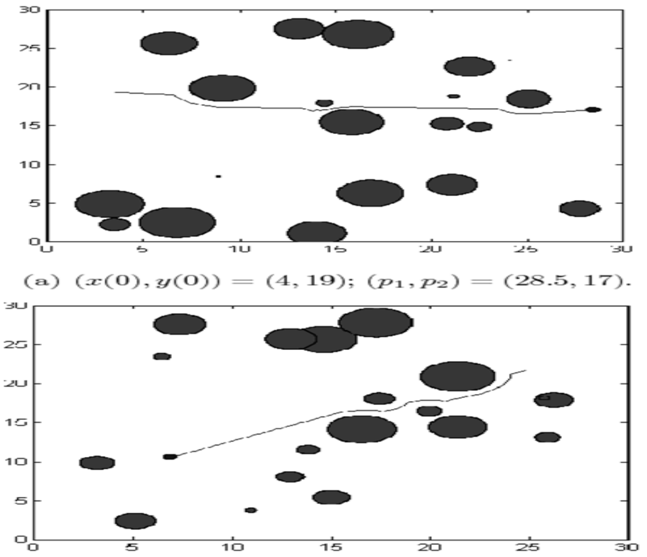


Fig.7- The time evolution of the trajectories of the point-mass robots in two different scenarios.

Application: A Planar Robot Arm

We apply the approach to a simple planar robot arm. The robot arm has a translational joint and a rotational joint in the z1-z2 plane as shown in the Fig. (8). The arm consists of two links made up of uniform slender rods; the revolute first link with fixed length and the prismatic second link which caries the payload at the gripper.

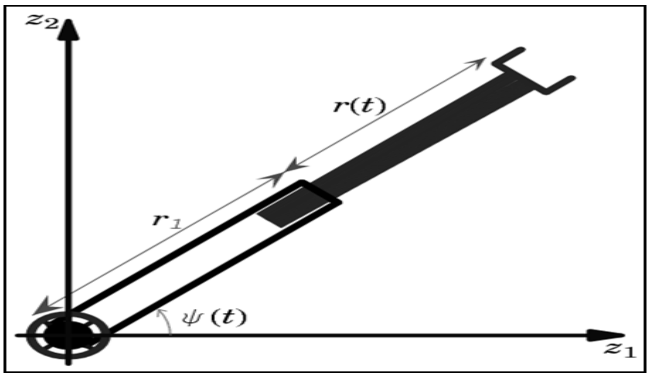


Fig.8- A Planar (RP) Manipulator in the z1-z2 plane.

With the help of Figure 8, we assume:

- the planar robot arm is anchored at the origin;
- the coordinate of the gripper is (x, y);
- the first link has a fixed length r_1;
- the second link has length $r(t)$ at time t; and
- the manipulator has angular position $\psi(t)$ at time t.

We now look for the kinematic model of this planar robot arm. We need to model $r(t)$ and $\psi(t)$. Note that $r(t) = \sqrt{x^2 + y^2} - r_1$ so we get $\dot{r}(t) = \frac{xv\cos(\theta+\varepsilon)+yv\sin(\theta+\varepsilon)}{\sqrt{x^2+y^2}}$. Similarly, $\tan\psi(t) = \frac{y(t)}{x(t)}$ so we get $\dot{\psi}(t) = \frac{xv\sin(\theta+\varepsilon)-yv\cos(\theta+\varepsilon)}{x^2+y^2}$

where v, θ and ε are as per the earlier definitions. Thus the kinematic equations for the planar (RP) arm is

$$\left.\begin{aligned} \dot{r}(t) &= \frac{xv\cos(\theta+\varepsilon) + yv\sin(\theta+\varepsilon)}{\sqrt{x^2+y^2}} \\ \dot{\psi}(t) &= \frac{xv\sin(\theta+\varepsilon) - yv\cos(\theta+\varepsilon)}{x^2+y^2} \end{aligned}\right\} \quad (4)$$

Convergence to the Target and Obstacle Avoidance

For the convergence of the end effector to the target position (p_1,p_2), we will use the velocity algorithm described in section 4.0

$$v(t) = |\, v_0 \,| \frac{(p_1 - x(t))^2 + (p_2 - y(t))^2}{(p_1 - x(0))^2 + (p_2 - y(0))^2}, \quad v_0 := v(0).$$

Assumption 1. To avoid collisions between the fixed obstacles and the end-effector, we treat the end-effector as a point mass.

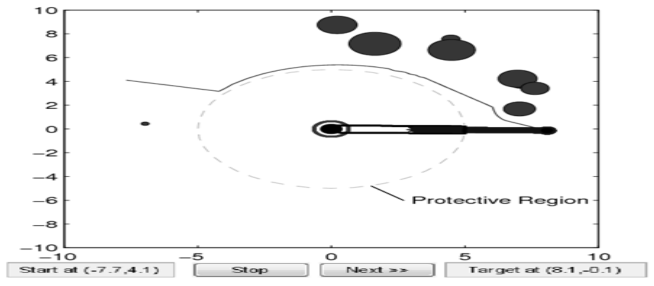
Remark 1. The assumption facilitates the use of the multilayer perceptron algorithm described in section 6.0 for potential collision avoidances.

Mechanical Singularities

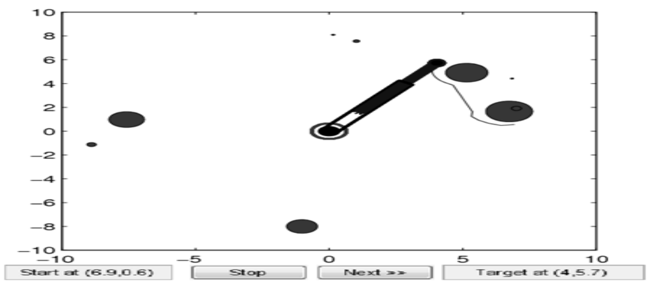
The circular region with the origin (0, 0) as its center that encloses the first link is treated as an artificial obstacle for the end-effector (see Fig. 9(a)). This mechanical singularity can be avoided using the collision avoidance scheme described above.

Simulations

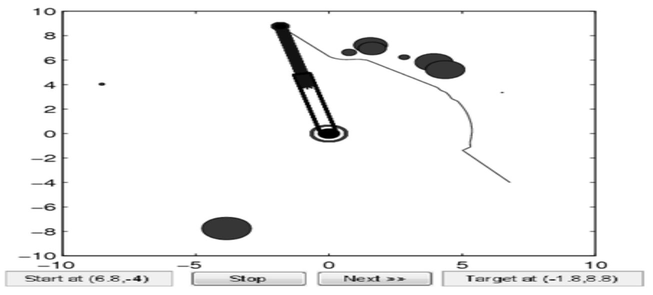
To demonstrate the effectiveness of our method, we show computer simulations of four different scenarios. Each shows the evolution of the trajectory of the end-effector in a workspace fixed with stationary obstacles of random numbers and sizes. These are shown in Fig. 9(a) - (c).



(x(0),y(0)) = (-7.4,4.1); (p1,p2) = (8.1,-0.1)



(x(0),y(0)) = (6.9,0.6); (p_1,p_2) = (4,5.7)



(x(0),y(0)) = (6.8,-4); (p1,p2) = (-1.8,8.8)

Fig.9- The time evolution of the trajectories of the end-effectors in various scenarios.

Concluding Remarks

The paper presents a new neural network approach for the motion planning and control of a point-mass robot. Our method in the construction of a collision-free path in an environment fixed with randomized multiple obstacles, of arbitrary sizes, is based on learning via the multilayer perceptron. The architecture utilizes two neural systems: the first neural network indicates whether the robot should turn or not, while the second determines the precise angle of turning. A unique velocity algorithm is designed to ensure that the robot moves towards its goal and remains there once reached.

The training data for the neural networks are obtained using computer simulations where the initial paths are traced by the user. The user plays the role of the supervisor and trains the robot to make its way intelligently toward its target and to avoid obstacles enroute its target. Ten sets of data were obtained, in each case the size and position of obstacles and the initial and target positions were chosen randomly. These data were then feeded into the networks which were trained using backpropagation. After the networks were trained, it was then used to control the motion in the workspace with obstacles, target and initial position placed randomly. It worked to perfection.G

Finally, simulations using point-mass robot and an anchored 2-link (RP) manipulator highlight an asymptotic stabilization of the system controlled by this neural network approach.

References

[1] Janglova D. (2004) *International Journal of Advanced Robotic Systems*, 1(1), 15–22a.

[2] Moreno P.O., Ruiz S.I. and Valenzuela J.C. (2007) *Electronics, Robotics and Automotive Mechanics Conference*.

[3] Yang S.X. and Meng M. (2000) *Neural Networks*, 13 (2), 143-148.

[4] Pham D.T. and Sahin Y. (2000) *Robotica*, 8 (5), 505–512.

[5] Chohra A.D., Sif F. and Talaoubrid S. (1995) *IAV'*95, 238–243 .

[6] Latombe J.C. (1991) *Robot Motion Planning.*

[7] Sharma B. (2008) *New Directions in the Applications of the Lyapunov-based Control Scheme to the Findpath Problem. PhD thesis, University of the South Pacific, Suva, Fiji Islands.*

[8] Fuller J.L. (1998) *Introduction, Programming, and Projects. Prentice Hall.*

[9] Sharma B., Vanualailai J. and Chand U. (2009) *European Journal of Pure and Applied Mathematics*, 2 (3), 401–425.

[10] Sharma B., Vanualailai J. and Prasad A. (2011) *Journal of Mathematics*, 41(3), 900–940.

[11] Skowronski J.M. (1990) *Nonlinear Lyapunov dynamics*.

[12] Khatib O. (1986) *International Journal of Robotics Research*, 7 (1), 90–98.

[13] Lee L.F. and Krovi K. (2006) *Metrics for Intelligent Systems Workshop*.

[14] Nam Y.S., Lee B.H. and Ko N.Y. (1995) *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2, 482-487.

[15] Rimon E. (1992) *IEEE Transactions on Robotics and Automation*, 8(5), 501–517.

[16] Song P. and Kumar V. (2002) *IEEE International Conference on Robotics & Automation*.

[17] Vanualailai J., Sharma B. and Ali A. (2007) *Journal of Pure and Applied Mathematics*, 3(2), 175–190.

[18] Fujimura K. and Samet H. (1989) *IEEE Transactions on Robotics and Automation*, 5(1):61–69.

[19] Jarvis R.A. (1983) *Australian Computer Journal*, 15(3), 103-111.

[20] Zeghloul S., Helguera C. and Ramirez G. (2006) *Robotica,* 24 (5), 539–548.

[21] Roy D. (2005) *Journal of Intelligent & Robotic Systems*, 43 (2-4), 111–145.

[22] Edelstein-Keshet L. (2001) *International Symposium on Nonlinear Theory and its Applications*, 1-7.

[23] McCulloch W.S. and Pitts W.H. (1943) *Bulletin of Mathematical Biophysics*, 5, 115–133.

[24] Kung S and Hwang J. (1989) *IEEE Transactions on Robotics and Automation*, 5 (5), 641–657.

[25] Bekey G.A. and Goldberg K.Y. (1993) *Neural Networks in Robotics*.

[26] Rumelhart D. and McClelland J. (1986) *MIT Press, Cambridge.*