American Journal of Engineering and Applied Sciences 7 (1): 149-164, 2014
ISSN: 1941-7020

doi:10.3844/ajeassp.2014.149.164 Published Online 7 (1) 2014 (http://www.thescipub.com/ajeas.toc)

IMPLEMENTATION OF AUTONOMOUS NAVIGATION ALGORITHMS ON TWO-WHEELED GROUND MOBILE ROBOT

[1]Stephen Armah, [2]Sun Yi and [3]Taher Abu-Lebdeh

[1,2]Department of Mechanical Engineering,
[3]Department of Civil, Architectural and Environmental Engineering,
North Carolina A and T State University, Greensboro, NC 27411, USA

Received 2014-02-20; Revised 2014-02-23; Accepted 2014-04-07

ABSTRACT

This study presents an effective navigation architecture that combines 'go-to-goal', 'avoid-obstacle' and 'follow-wall' controllers into a full navigation system. A MATLAB robot simulator is used to implement this navigation control algorithm. The robot in the simulator moves to a goal in the presence of convex and non-convex obstacles. Experiments are carried out using a ground mobile robot, Dr Robot X80SV, in a typical office environment to verify successful implementation of the navigation architecture algorithm programmed in MATLAB. The research paper also demonstrates algorithms to achieve tasks such as 'move to a point', 'move to a pose', 'follow a line', 'move in a circle' and 'avoid obstacles'. These control algorithms are simulated using Simulink models.

Keywords: Wheeled Mobile Robots, PID-Feedback Control, Navigation Control Algorithm, Differential Drive, Hybrid Automata

1. INTRODUCTION

The field of mobile robot control has attracted considerable attention of researchers in the areas of robotics and autonomous systems in the past decades. One of the goals in the field of mobile robotics is the development of mobile platforms that robustly operate in populated environments and offer various services to humans. Autonomous mobile robots need to be equipped with appropriate control systems to achieve the goals. Such control systems are supposed to have navigation control algorithms that will make mobile robots successfully 'move to a point', 'move to a pose', 'follow a path', 'follow a wall' and 'avoid obstacles (stationary or moving)'. Also, robust visual tracking algorithms to detect objects and obstacles in real-time have to be integrated with the navigation control algorithms.

A mobile robot is an automatic machine that is capable of movement in given environments; they are not fixed to one physical location. Wheeled Mobile Robots (WMRs) are increasingly present in industrial and service robotics, particularly when flexible motion capabilities are required on reasonably smooth grounds and surfaces. Several mobility configurations (wheel number and type, their location and actuation and single- or multi-body vehicle structure) can be found in different applications (De Luca *et al*., 2001). The most common for single-body robots are differential drive and synchro drive (both kinematically equivalent to a unicycle), tricycle or car-like drive and omnidirectional steering (De Luca *et al*., 2001).

The main focus of the research is the navigation control algorithm that has been developed to enable Differential Drive Wheeled Mobile Robot (DDWMR) to accomplish its assigned task of moving to a goal free from any risk of collision with obstacles. In order to develop this navigation system a low-level planning is used, based on a simple model whose input can be calculated using a PID controller or transform into actual robot input. The research also presents control algorithms that make mobile

Corresponding Author: Stephen Armah, Department of Mechanical Environmental, North Carolina A and T State University, 1601 E. Market Street, Greensboro, NC 27411, USA Tel: (336) 285-3753

robots 'move to a point', 'move to a pose', 'follow a line', 'follow a circle' and 'avoid obstacles' taken from the literature (Corke, 2011; Egerstedt, 2013).

A MATLAB robot simulator is used to implement the navigation control algorithm and the individual control algorithms were simulated using Simulink models. For the navigation control algorithm, the robot simulator is able to move to a goal in the presence of convex and non-convex obstacles. Also, several experiments are performed using a ground robot, Dr Robot X80SV, in a typical office environment to verify successful implementation of the navigation architecture algorithm programmed in MATLAB.

Possible applications of WMR include security robots, land mine detectors, planetary exploration missions, Google autonomous car, autonomous vacuum cleaners and lawn mowers, toxic cleansing, tour guiding, personal assistants to humans, etc. (Jones and Flynn, 1993).

The remainder of this study is organized as follows: In section II the kinematic model of the DDWMR is shown. The control algorithms are presented in section III. In section IV simulations and experiments performed in MATLAB/Simulink are explained. Simulation and experimental results are summarized in section V. Concluding remarks and future work is presented in section VI.

2. KINEMATIC MODEL OF THE DDWMR

The DDWMR setup used for the presented study is shown in **Fig. 1** (top view). The mobile robot is made up of a rigid body and non-deforming wheels and it is assumed that the vehicle moves on a plane without slipping, i.e., there is a pure rolling contact between the wheels and the ground.

The configuration of the vehicle is represented by the generalized coordinates q = (x,y,θ), where (x,y) is the position and θ is the orientation (heading) of the center of the axis of the wheels, C, with respect to a global inertial frame {O,X,Y}. Let $\{O_V, X_V, Y_V\}$ be the vehicle frame. The vehicle's velocity is by definition ν in the vehicle's x-direction, L is distance between the wheels, R is radius of the wheels, ν_r is the right wheel angular velocity, ν_l is the left wheel angular velocity and ω is the heading rate. The kinematic model of the DDWMR based on the stated coordinate is given by:

$$\begin{aligned}\dot{x} &= \frac{R}{2}(\nu_r + \nu_l)\cos\theta \\ \dot{y} &= \frac{R}{2}(\nu_r + \nu_l)\sin\theta \\ \dot{\theta} &= \frac{R}{L}(\nu_r - \nu_l)\end{aligned} \tag{1}$$

For the purpose of implementation the kinematic model of a unicycle is used, which corresponds to a single upright wheel rolling on the plane, with the equation of motion given as:

$$\begin{aligned}\dot{x} &= \nu\cos\theta \\ \dot{y} &= \nu\sin\theta \\ \dot{\theta} &= \omega\end{aligned} \tag{2}$$

The inputs in (Equation 1 and 2) are ν_r, ν_l, ν and ω. These inputs are related as Equation 3:

$$\nu_r = \frac{2\nu + \omega L}{2R} \quad \nu_l = \frac{2\nu - \omega L}{2R} \tag{3}$$

A Simulink model shown in **Fig. 2** have been developed that implements the unicycle kinematic model.

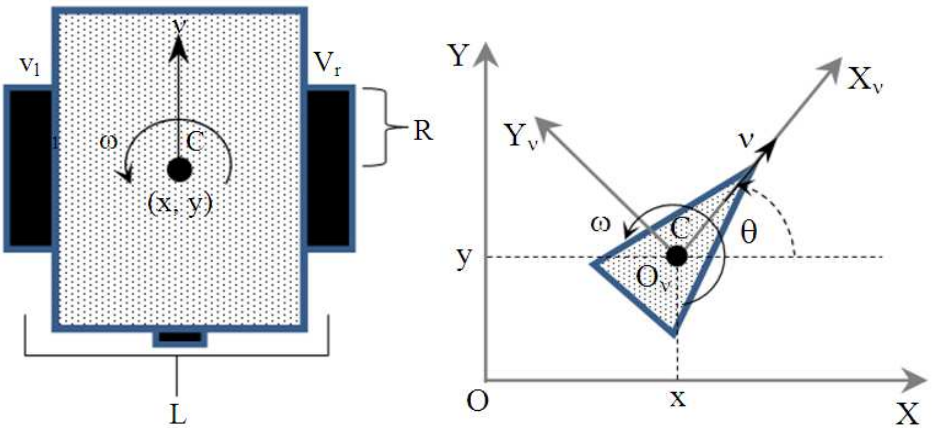

Fig. 1. Coordinates for differential drive wheeled mobile robot

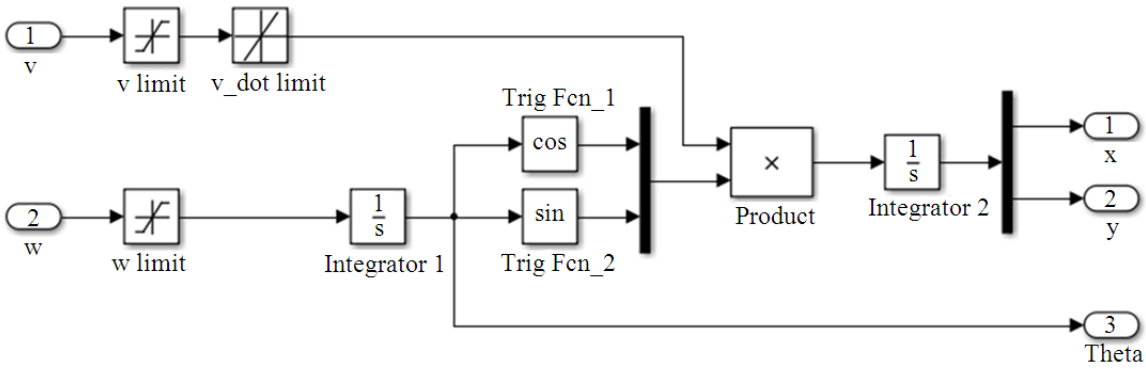

Fig. 2. Simulink block for the unicycle kinematic model

The velocity input has a rate limiter to model finite acceleration and limiters on the velocity and the heading or turn rate.

3. CONTROL ALGORITHMS

Control of the unicycle model inputs is about selecting the appropriate input, $u = (v\omega)^T$ and applying the traditional PID-feedback controller, given by Equation 4:

$$u(t) = PID(e) = K_p e(t) + K_I \int_0^t e(\tau)d\tau + K_D \frac{de(t)}{dt} \quad (4)$$

where, e, define for each task below, is the error between the desired value and the output value, K_p is the proportional gain, K_I is the integrator gain, K_D is the derivative gain and t is time. The control gains used in this research are obtained by tweaking the various values to obtain satisfactory responses. If the vehicle is driven at a constant velocity, $v = v_0$ then the control input will only vary with the angular velocity, ω, thus:

$$\omega = PID(e) \quad (5)$$

3.1. Developing Individual Controllers

This section presents control algorithms that make mobile robots 'move to a point', 'move to a pose', 'follow a line', 'follow a circle' and 'avoid obstacles'.

3.1.1. Moving to a Point

Consider a robot moving toward a goal point, (x_g, y_g), from a current position, (x, y), in the xy-plane, as depicted in **Fig. 3** below.

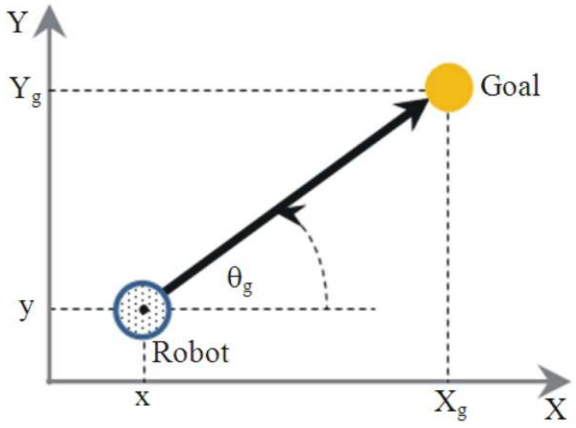

Fig. 3. Coordinates for moving to a point

The desired heading (robot's relative angle), θ_g, is determined as:

$$\theta_g = \theta_{goal} = \arctan\left(\frac{y_g - y}{x_g - x}\right) \quad (6)$$

And the error, e, is defined Equation 7:

$$e = \theta_g - \theta \quad (7)$$

To ensure $e \in [-\pi, \pi]$, a corrected error, e' is used instead of e as shown below Equation 8:

$$e' = \arctan 2(\sin(e), \cos(e)) \in [-\pi, \pi] \quad (8)$$

Thus ω can be controlled using (Equation 5). If the robot's velocity is to be controlled, a proportional

controller gain, K_v, is applied to the distance from the goal, shown below (Corke, 2011) Equation 9:

$$v = K_v\sqrt{\left(x_g + x\right)^2 + \left(y_g - y\right)^2} \tag{9}$$

3.1.2. Moving to a Pose

The above controller could drive the robot to a goal position but the final orientation depends on the starting position. In order to control the final orientation (Equation 5) is rewritten in matrix form as (Corke, 2011):

$$\begin{pmatrix}\dot{x}\\ \dot{y}\\ \dot{\theta}\end{pmatrix} = \begin{pmatrix}\cos\theta & 0\\ \sin\theta & 0\\ 0 & 1\end{pmatrix}\begin{pmatrix}v\\ \omega\end{pmatrix} \tag{10}$$

Equation 10 is then transformed into the polar coordinate form using the notation shown in **Fig. 4** below. Applying a change of variables, we have Equation 11:

$$\begin{aligned}\rho &= \sqrt{\Delta_x^2 + \Delta_y^2}\\ \alpha &= \arctan\left(\frac{\Delta_y}{\Delta_x}\right) - \theta\\ \beta &= -\theta - \alpha\end{aligned} \tag{11}$$

Which results in Equation 12:

$$\begin{pmatrix}\dot{\rho}\\ \dot{\alpha}\\ \dot{\beta}\end{pmatrix} = \begin{pmatrix}-\cos a & 0\\ \frac{\sin a}{\rho} & -1\\ -\frac{\sin a}{\rho} & 0\end{pmatrix}\begin{pmatrix}v\\ \omega\end{pmatrix}, \text{ if} \alpha \in \left(-\frac{\pi}{2}, \frac{\pi}{2}\right] \tag{12}$$

And assumes the goal {G} is in front of the vehicle. The linear control law Equation 13:

$$v = K_\rho\rho \quad \omega = K_\alpha\alpha + K_\beta\beta \tag{13}$$

Drives the robot to unique equilibrium at $(\rho,\alpha,\beta) = (0,0,0)$. The intuition behind this controller is that the terms $K_\rho\rho$ and $K_\alpha\alpha$ drive the robot along a line toward {G} while the term $K_\beta\beta$ rotates the line so that $\beta \rightarrow 0$ (Corke, 2011). The closed-loop system Equation 14:

$$\begin{pmatrix}\dot{\rho}\\ \dot{\alpha}\\ \dot{\beta}\end{pmatrix} = \begin{pmatrix}-K_\rho\cos\alpha\\ K_\rho\sin\alpha - K_\alpha\alpha - K_\beta\beta\\ -K_\rho\sin\alpha\end{pmatrix} \tag{14}$$

Is stable so long as $K_\rho>0$, $K_\beta>0$, $K_\alpha - K_\rho>0$ (Corke, 2011). For the case where the goal is behind the robot, that is $\alpha \notin \left(-\frac{\pi}{2}, \frac{\pi}{2}\right]$ the robot is reverse by negating v and ω in the control law. The velocity v always has a constant sign which depends on the initial value of α (Corke, 2011).

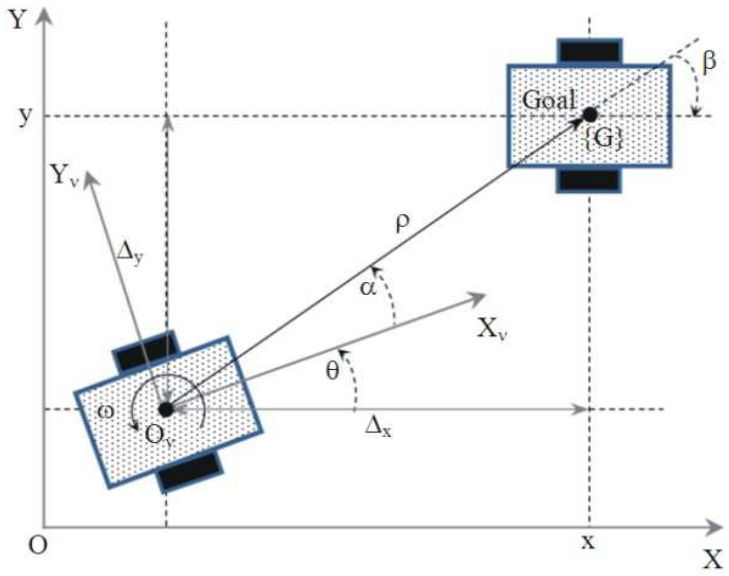

Fig. 4. Coordinates for moving to a pose

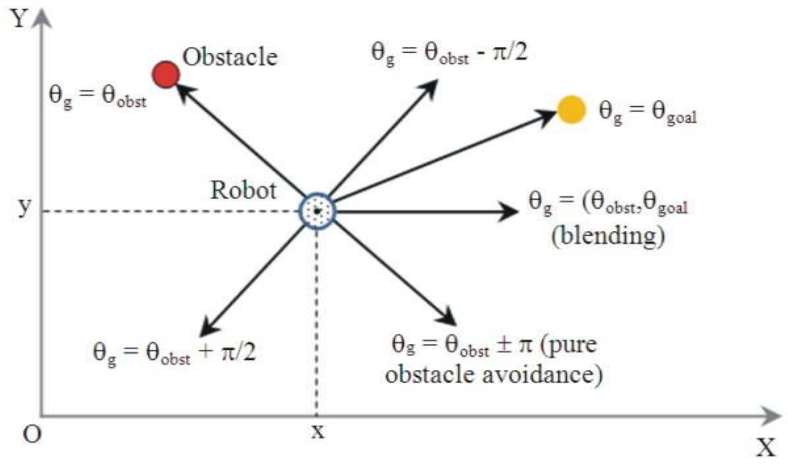

Fig. 5. Coordinates for avoiding obstacle (Egerstedt, 2013)

3.1.3. Obstacle Avoidance

In a real environment robots must avoid obstacles in order to go to a goal. Depending on the positions of the goal and the obstacle (s) relative to the robot, the robot need move to the goal using θ_g from a 'pure go-to-goal' behavior or blending the 'avoid obstacle' and the 'go-to-goal' behaviors. In pure obstacle avoidance the robot drives away from the obstacle and move in the opposite direction. The possible θ_g that can be used in the control law discussed in section III-A.1 are shown in **Fig. 5** below, where θ_{obst} is the obstacle heading.

3.1.4. Following a Line

Another useful task for a mobile robot is to follow a line on a plane defined by $a_x+b_y+c = 0$. This requires two controllers to adjust the heading. One controller steers the robot to minimize the robot's normal distance from the line given by Equation 15:

$$d = \frac{(a,b,c).(x,y,1)}{\sqrt{a^2+b^2}} \tag{15}$$

The proportional controller Equation 16:

$$\alpha_d = -K_d d, \quad K_d > 0 \tag{16}$$

Turns the robot toward the line. The second controller adjust the heading angle to be parallel to the line Equation 17:

$$\theta_g = \arctan\left(-\frac{a}{b}\right) \tag{17}$$

Using the proportional controller:

$$\alpha_h = K_h\left(\theta_g - \theta\right), \quad K_h > 0 \tag{18}$$

The combined control law Equation 19:

$$\omega = -K_d d + K_h\left(\theta_g - \theta\right) \tag{19}$$

Turns the wheel so as to drive the robot toward the line and moves along it (Corke, 2011).

3.1.5. Following a Circle

Instead of a straight line the robot can follow a defined path on the xy-plane and in this section the robot follows a circle. This problem is very similar to the control problem presented in section III-A.1, except that this time the point is moving. The robot maintains a distance d_d behind the pursuit point and an error, e, can be formulated as (Corke, 2011) Equation 20:

$$e = \sqrt{\left(x_g - x\right)^2 + \left(y_g - y\right)^2} - d_d \tag{20}$$

That will be regulated to zero by controlling the robot's velocity using a PI controller Equation 21:

$$v_d = PI(e) = K_p e(t) + K_I \int_0^\tau e(\tau)d\tau \tag{21}$$

The integral term is required to provide a finite velocity demand v_d when the following error is zero. The second controller steers the robot toward the target which

is at the relative angle given by (Equation 6) and a controller given by (Equation 18).

3.2. Developing Navigation Control Algorithm

This section introduces how the navigation architecture, that consist of go-to-goal, follow-wall and avoid obstacle behaviors, was developed. In order to develop the navigation system a low-level planning was used, by starting with a simple model whose input can be calculated by using a PID controller or transform into actual robot input, depicted in **Fig. 6** (Egerstedt, 2013). For this simple planning a desired motion vector, x, is picked and set equal to the input, u, (Equation 22).

$$\dot{x} = u = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} x + \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} u,\ x \in \Re^2 \tag{22}$$

This selected system is controllable as compared to the unicycle system which is non-linear and not controllable even after it has been linearized. This layered architecture makes the DDWMR act like the point mass model shown in (Equation 22) (Egerstedt, 2013).

3.2.1. Go-To-Goal (GTG) Mode

Consider the point mass moving toward a goal point, x_g, with current position as x in the xy-plane. The error, $e = x_g - x$, is controlled by the input $u = Ke$, where K is gain matrix. Since $\dot{e} = -Ke$ the system is asymptotically stable if K>0. An appropriate K is selected to obey the function shown in **Fig. 7a** above such that $\dot{e} = -K(\|e\|)e$, where a and b are constants to be selected; in this way the robot will not go faster further away from the goal (Egerstedt, 2013).

3.2.2. Obstacle Avoidance (AO) Mode

Let the obstacle position be x_0, then $e = x_0 - x$ is controlled by the input $u = Ke$ and since $\dot{e} = -Ke$ the system is desirably unstable if K>0. An appropriate K is selected to obey the function shown in **Fig. 7b** above such that $\dot{e} = -K(\|e\|)e$, where c and ε are constants to be selected (Egerstedt, 2013).

3.2.3. Blending AO and GTG Modes

In a 'pure GTG' mode, u_{GTG}, or 'pure AO' mode, u_{AO}, or what is termed as hard switches, performance can be guaranteed but the ride can be bumpy and the robot can encounter the zeno phenomenon (Egerstedt, 2013). A control algorithm for blending the u_{GTG} and u_{AO} modes is given by (Equation 23). This algorithm ensures smooth ride but does not guarantee performance (Egerstedt, 2013):

$$u_{GTG,AO} = \alpha(\Delta)u_{GTG} + (1-\alpha(\Delta))u_{AO}, \alpha(\Delta) \in [01] \tag{23}$$

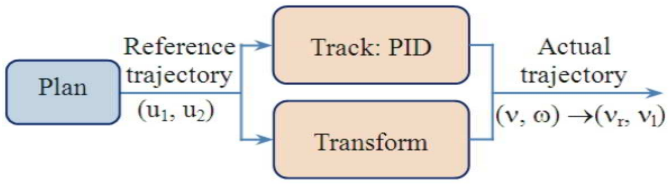

Fig. 6. Planning model input to actual robot input

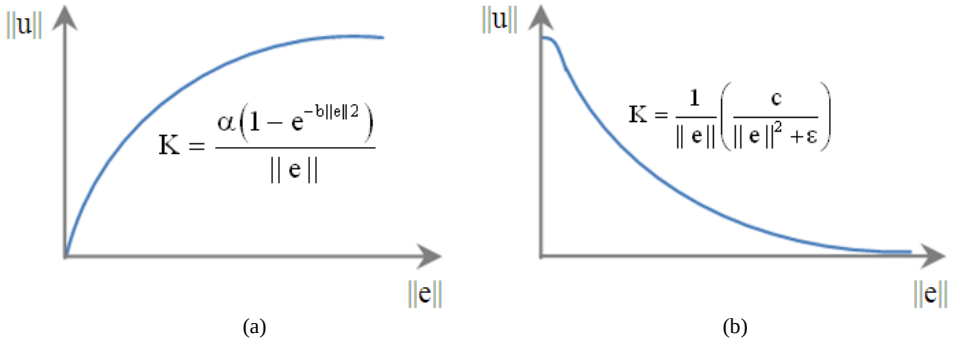

Fig. 7ab. Suitable graph for an appropriate K (a) Go-to-goal mode (b) Obstacle avoidance mode

where, Δ is a constant distance to the obstacle/boundary and α is the blending function to be selected, giving appropriately as an exponential function by:

$$\alpha(\Delta) = 1 - e^{-\beta\Delta} \tag{24}$$

where, β is a constant to be selected.

3.2.4. Follow-Wall (FW) Mode

As pointed out in section III-B.2, in a pure obstacle avoidance mode the robot drives away from the obstacle and move in the opposite direction, but this is overly cautious in a real environment where the task is to go to a goal. The robot should be able to avoid obstacles by going around its boundary and this situation leads to what is termed as the follow-wall or an induced or sliding mode, u_{FW}, between the u_{GTG} and u_{AO} modes; this is needed for the robot to negotiate complex environments (Egerstedt, 2013).

The FW mode maintains Δ to the obstacle/boundary as if it is following it and the robot can clearly move in two different directions, clockwise (c) and counter-clockwise (cc), along the boundary, **Fig. 8**. This is achieved by rotating u_{AO} by $\pi/2$ and $-\pi/2$ to obtain u_{AW}^{cc} and u_{AW}^{c} respectively and then scaled by δ to obtain a suitable induced mode, (Equation 25-27), where $R(\varnothing)$ is a rotation matrix (Egerstedt, 2013):

$$R(\varnothing) = \begin{bmatrix} \cos\varnothing & -\sin\varnothing \\ \sin\varnothing & \cos\varnothing \end{bmatrix} \tag{25}$$

$$u_{FW}^{cc} = \delta R(\pi/2) u_{AO} = \delta \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} u_{AO} \tag{26}$$

$$u_{FW}^{c} = \delta R(-\pi/2) u_{AO} = \delta \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} u_{AO} \tag{27}$$

The direction the robot selects to follow the boundary is determined by the direction of u_{GTG} and it is determined using the dot product of u_{GTG} and u_{FW}, as shown in (Equation 28 and 29) (Egerstedt, 2013):

$$\left\langle u_{GTG}, u_{FW}^{cc} \right\rangle = \left(u_{GTG}\right)^{T} u_{FW}^{cc} > 0 \Rightarrow u_{FW}^{cc} \tag{28}$$

$$\left\langle u_{GTG}, u_{FW}^{c} \right\rangle = \left(u_{GTG}\right)^{T} u_{FW}^{c} > 0 \Rightarrow u_{FW}^{c} \tag{29}$$

Another issue to be addressed is when the robot releases u_{FW}, that is when to stop sliding. The robot stops sliding when "enough progress" has been made and there is a "clear shot" to the goal, as shown in (Equation 30-32), where τ is the time of last switch (Egerstedt, 2013):

$$\text{enough progres}: \| x - x_g \| < d_\tau \tag{30}$$

$$\text{where } d_\tau = \| x(\tau) - x_g \| \tag{31}$$

$$\text{clear shot}: \left\langle u_{AO}, u_{GTG} \right\rangle = \left(u_{AO}\right)^{T} u_{GTG} > 0 \tag{32}$$

3.2.5. Implementation of the Navigation Algorithms

The behaviors or modes discussed above are put together to form the navigation architecture shown in **Fig. 9** below. The robot started at the state x_0 and arrived at the goal x_g, switching between the three different operation modes; this system of navigation is termed the hybrid automata where the navigation system has been described using both the continuous dynamics and the discrete switch logic (Egerstedt, 2013).

An illustration of this navigation system is shown in **Fig. 10**, where the robot avoids a rectangular block as it moves to a goal.

3.2.6. Tracking and Transformation of the 'Simple' Model Input

The simple planning model input, $u = (u_1\ u_2)^T$ can be tracked using a PID controller or clever transformation can be used to transform it into the unicycle model input, $u = (\nu\omega)^T$ (Egerstedt, 2013). These two approaches are discussed below.

Method 1: Tracking Using a PID Controller

Let the output from the planning model be $u = (u_1\ u_2)^T$ and the current position of the point mass be $x = (x\ y)^T$, **Fig. 11a** below, then the input, $u = (\nu\ \omega)^T$, to the unicycle model can be determined as shown below (Egerstedt, 2013) Equation 33-35:

$$\theta_g = \arctan\left(\frac{u_2}{u_1}\right) \tag{33}$$

$$\omega = \text{PID}\left(\theta_g - \theta\right) \tag{34}$$

From Equation 2:

$$\sqrt{\dot{x}^2 + \dot{y}^2} = \sqrt{\nu^2 \cos^2\theta + \nu^2 \sin^2} \Rightarrow \nu = \sqrt{u_1^2 + u_2^2} = \| u \| \tag{35}$$

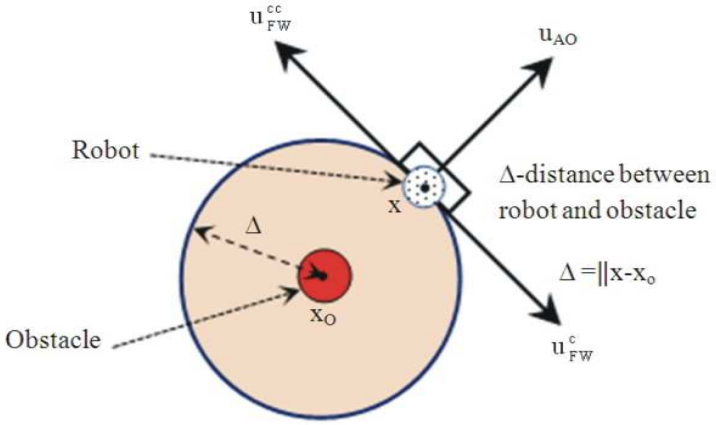

Fig. 8. Setup for follow-wall mode (Egerstedt, 2013)

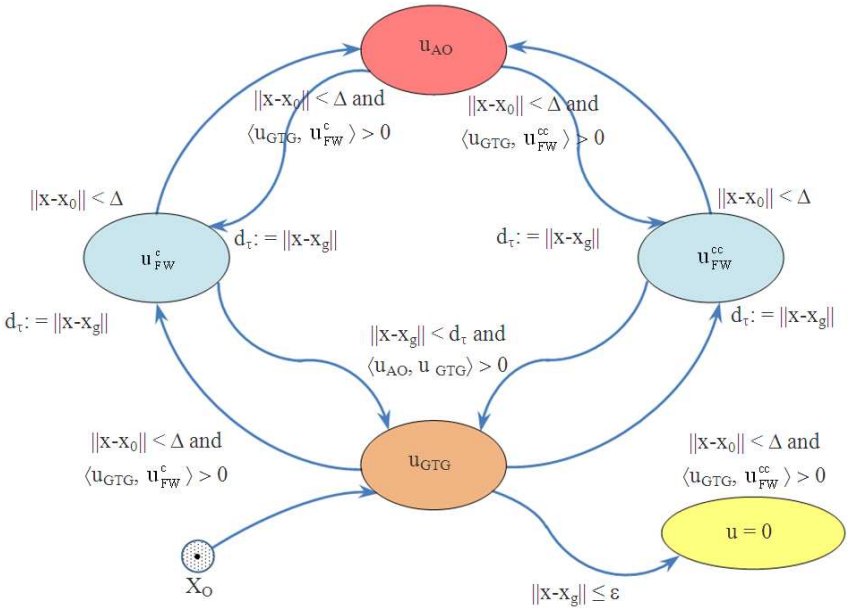

Fig. 9. Setup for navigation architecture

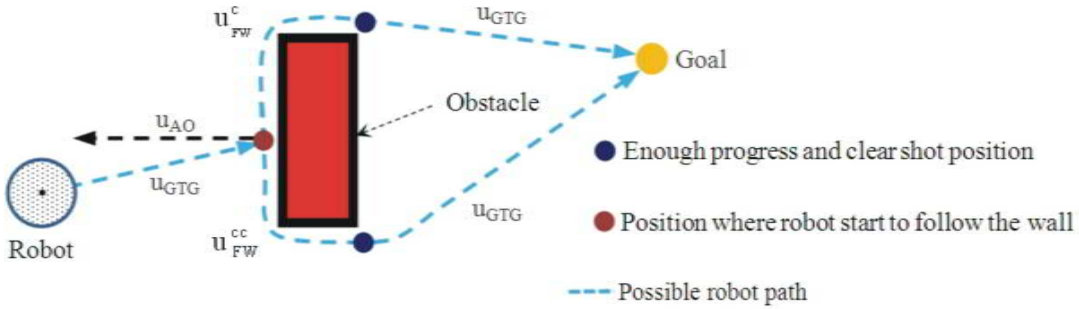

Fig. 10. Illustration of the navigation system

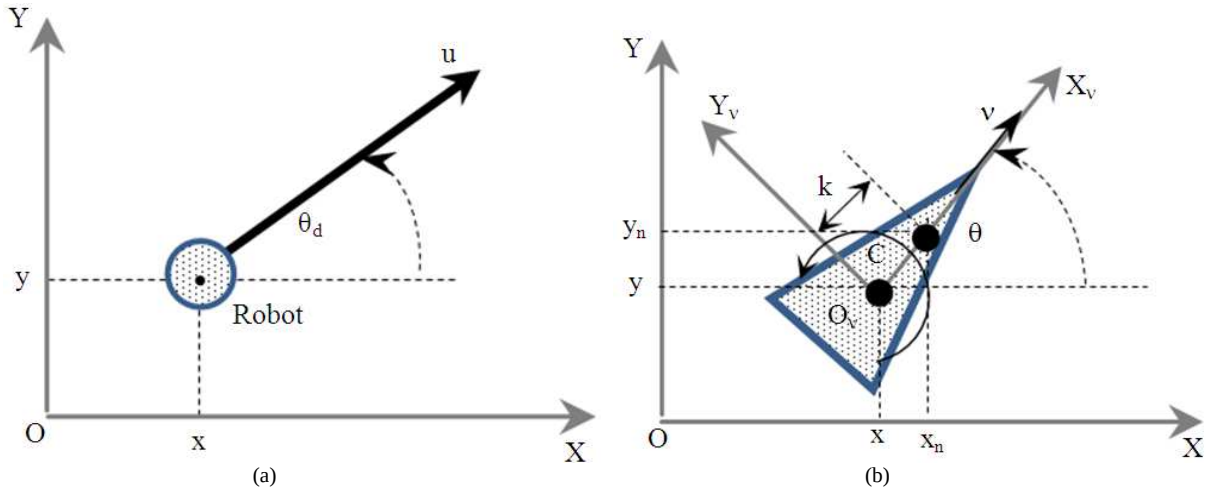

Fig. 11. (a) Coordinates for point mass in a specific direction (b) Coordinates for DDWMR model showing the new point

Method 2: Transformation

In this clever approach a new point (x_n, y_n), of interest is selected on the robot at a distance k from the center of mass, (x, y), as shown in **Fig. 11b** (Egerstedt, 2013), where $x_n = (x_n\ y_n)^T$ and $\dot{x}_n = u$.

If the orientation is ignored then (Egerstedt, 2013) Equation 36:

$$\begin{aligned} x_n &= x + k\cos\theta \\ y_n &= y + k\sin\theta \end{aligned} \tag{36}$$

Thus:

$$\begin{aligned} \dot{x}_n &= \dot{x} - k\dot{\theta}\sin\theta \\ \dot{y}_n &= \dot{y} + k\dot{\theta}\cos\theta \end{aligned} \tag{37}$$

Substituting (Equation 2) into (Equation 37) and using $\dot{x}_n = u_1$ and $\dot{y}_n = u_2$, we have (Egerstedt, 2013) Equation 38:

$$\begin{aligned} &\dot{x}_n = \nu\cos\theta - k\omega\sin\theta = u_1 \\ &\dot{y}_n = \nu\sin\theta + k\omega\cos\theta = u_2 \\ &\Rightarrow \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} \nu \\ k\omega \end{bmatrix} = \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} \\ &\Rightarrow R(\theta) \begin{bmatrix} 1 & 0 \\ 0 & k \end{bmatrix} \begin{bmatrix} \nu \\ \omega \end{bmatrix} = \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} \\ &\Rightarrow \begin{bmatrix} \nu \\ \omega \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & k^{-1} \end{bmatrix} R(-\theta) \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} \end{aligned} \tag{38}$$

4. SIMULATIONS AND EXPERIMENS

4.1. Simulations of Individual Controllers

Simulink models developed, by modifying similar models in (Corke, 1993-2011), that implement the control algorithms discussed in section III are presented in **Fig. 12-15**. These models are based on the unicycle model in **Fig. 2**.

4.2. Simulations of the Navigation System

A MATLAB robot simulator introduced in (MATLAB Robot Simulator (Software), 2013) was used to simulate the navigation architecture control algorithms presented in section III; the control algorithm combines the GTG, AO and FW controllers into a full navigation system for the robot. The robot simulator mimics the Khepera III (K3) mobile robot, whose model is based on the unicycle model presented in section II. The K3 is equipped with 11 Infrared (IR) range sensors, of which nine are located in a ring around it and two are located on the underside of the robot. The IR sensors are complemented by a set of five ultrasonic sensors (Corke, 2011). The K3 has a two-wheel differential drive with a wheel encoder for each wheel.

The MATLAB algorithm that controls the simulator implements Finite State Machine (FSM) to solve the full navigation problem. The FSM uses a set of if/elseif/else statements that first check which state (or behavior) the robot is in and then based on whether an event (condition) is satisfied, the FSM switches to another state or stays in the same state, until the robot reaches its goal (Egerstedt, 2013).

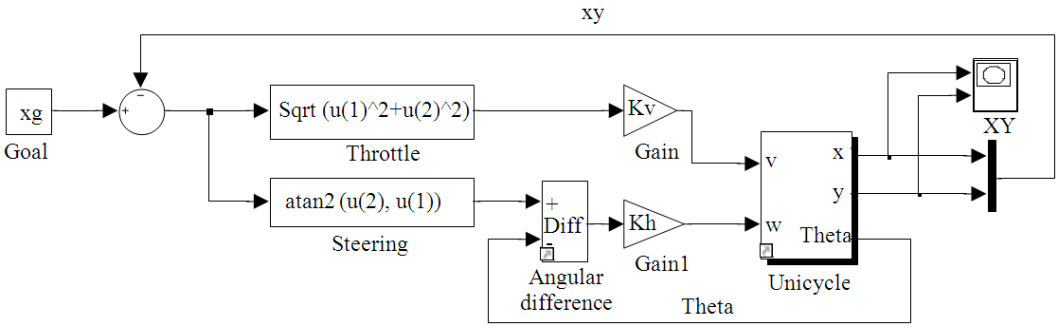

Fig. 12. Model that drives the robot to a point

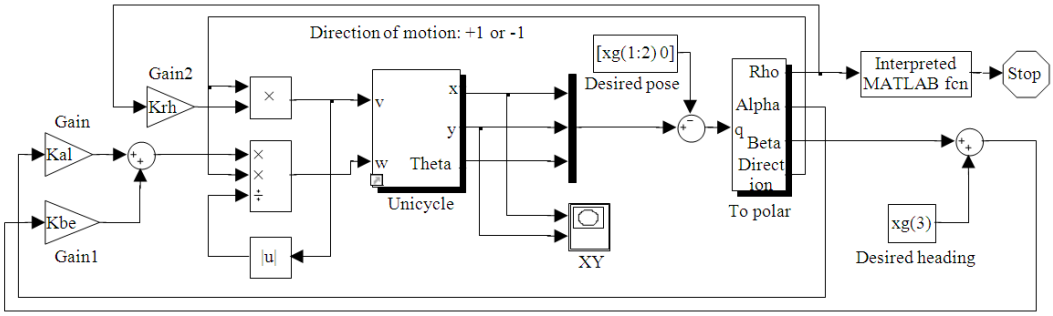

Fig. 13. Model that drives the robot to a pose

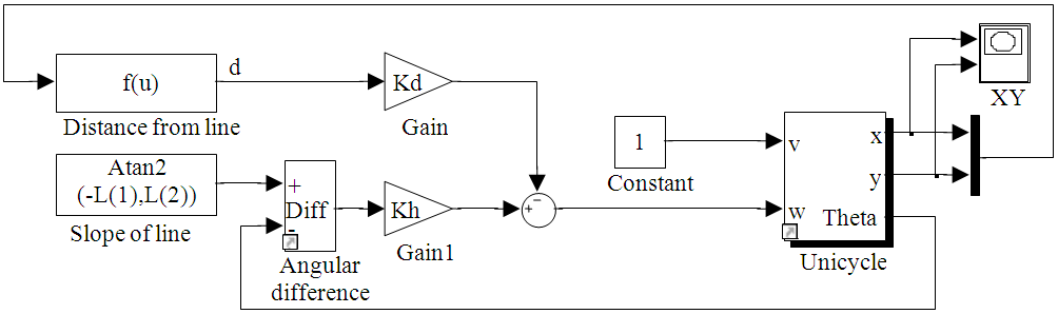

Fig. 14. Model that drives the robot along a line

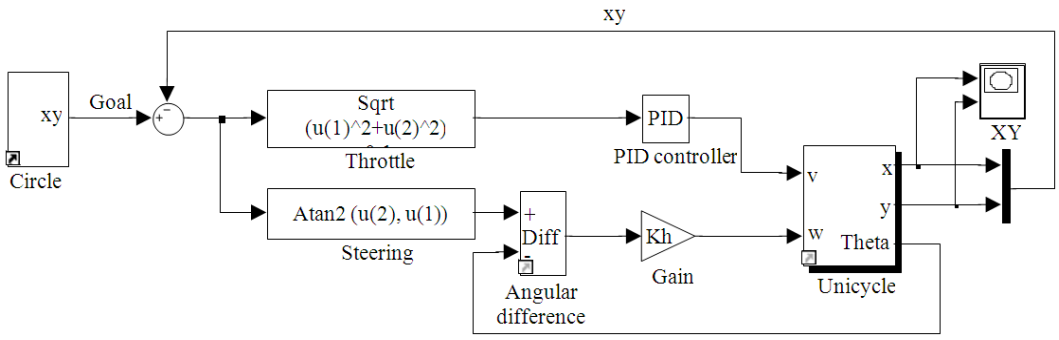

Fig. 15. Model that moves the robot in a unit circle (Corke, 1993-2011)

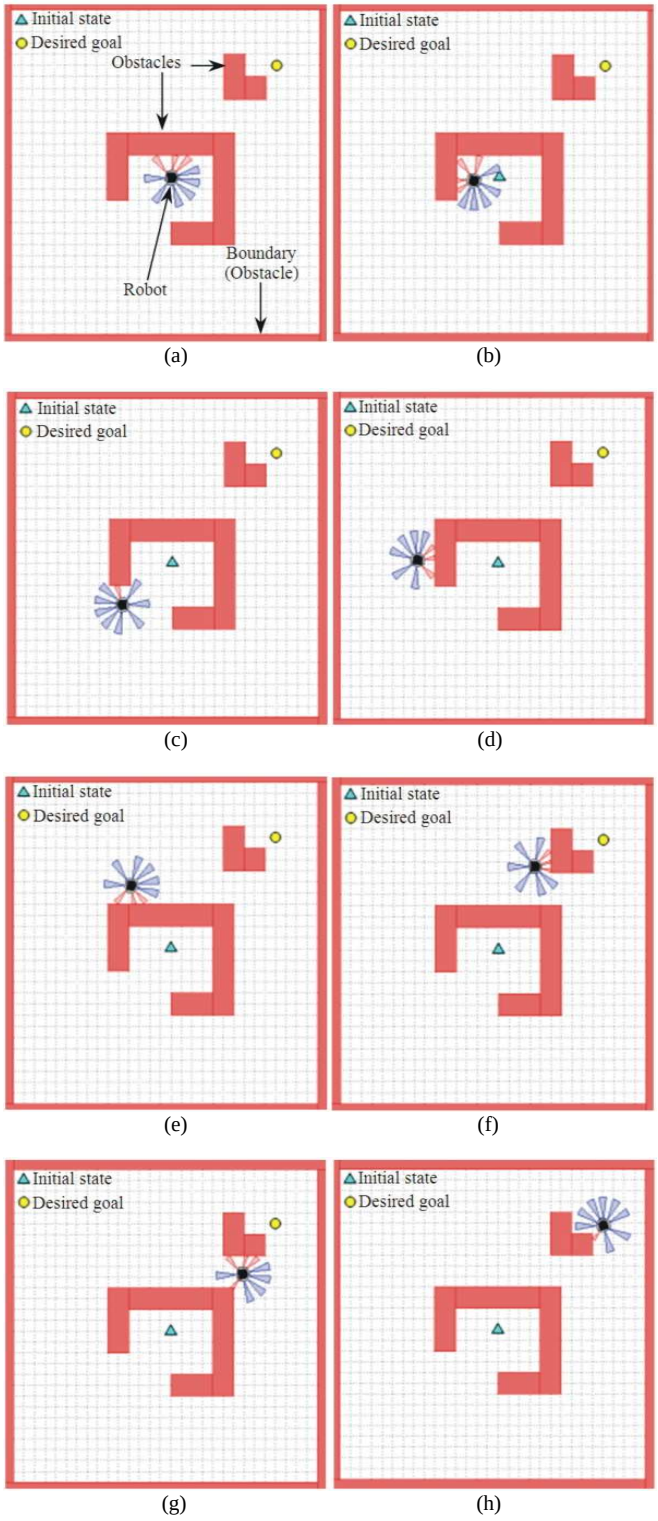

Fig. 16. (a-h) Sequence showing the MATLAB robot simulator implementing the navigation system

Figure 16a-h below shows a sequence of movement of the MATLAB robot simulator implementing the navigation system. The robot navigates around a cluttered, complex environment without colliding with any obstacles and reaching its goal location successfully.

4.3. Experiments Using Dr Robot X80SV

The control algorithm built for the navigation architecture presented in Section III has been experimented on Dr Robot X80SV, programmed using MATLAB GUI, in an office environment. The Dr Robot X80SV can be made to move to a goal whiles avoiding obstacles in front of it.

The Dr Robot X80SV, shown in **Fig. 17a** below, is a fully wireless networked that uses two quadrature encoders on each wheel for measuring its position and seven IR and three ultrasonic range sensors for collision detection. It has 2.6× high resolution Pan-Tilt-Zoom CCD camera with two-way audio capability, two 12V motors with over 22 kg.cm torque each and two pyroelectric human motion sensors.

The Dr Robot X80SV has a dimension of 38cm (length) ×35 cm (width) ×28 cm (height), maximum payload of 10kg (optional 40 kg) with robot weight of 3 kg. Its 12V 3700 mAh battery pack has three hours nominal operation time for each recharging and can drive up to a maximum speed of 1.0 m s^{-1}. The distance between the wheels is 26cm and the radius of the wheels is 8.5cm.

The PID-feedback system depicted in **Fig. 17b** above shows how the DC motor system of the robot is controlled. **Figure 18** shows the setup used for the experiments. After a connection is established between the host PC and the robot through the wireless router the MATLAB program receives and sends the motion/sensors signals using ActiveX control. The program directly exchange multimedia data with the Pan-Tilt-Zoom camera also through an ActiveX control.

A screenshot of the main MATLAB interface used for the Dr Robot X80SV control is shown in **Fig. 19** below. The interface was developed by mimicking a similar interface developed in C# by Dr Robot Inc. The motivation for using MATLAB instead of building upon the provided C# interface is to take advantage of the ease of simulation, quick and ease of developing GUI and making use of the in-built control strategies libraries in MATLAB for this research and future studies.

The main GUI interface has three sections: Information about the robot settings and sensors, multimedia and the vision and control. The robot settings information includes the IP addresses of the robot and the camera, the robot wheel radius, distance between the robot wheels and the encoder count per revolution. The sensors information, updated in real-time, includes the IR, ultrasonic, motor, human, temperature, battery and the position of the robot.

The multimedia section include real-time video stream from the robot, which can be controlled using a pan and tilt tools. The section also has tools for capturing images and recording the video stream. In addition, the section also has a tool to capture live audio from the robot.

The vision and control section has tools for performing ‘Basic Motion Control’, ‘Individual Motion Control (PID and MPC)’, ‘Navigation System (PID and MPC) and ‘Object Recognition and Tracking’.

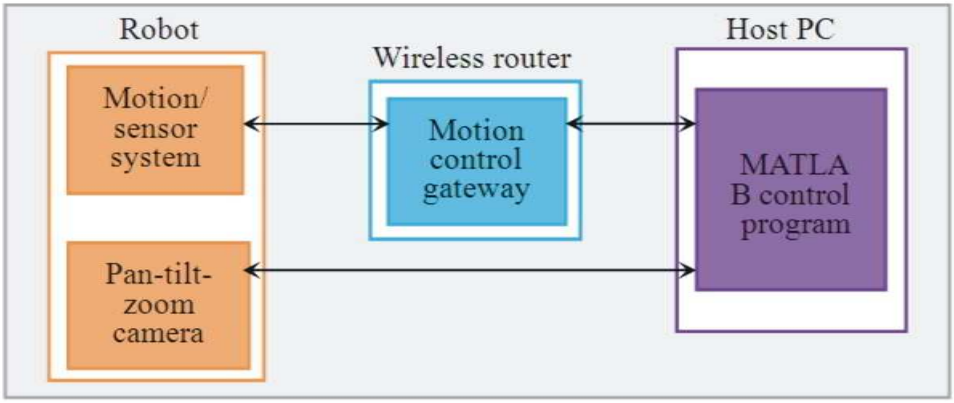

Fig. 18. Setup used for the experiments (Robot, 2008)

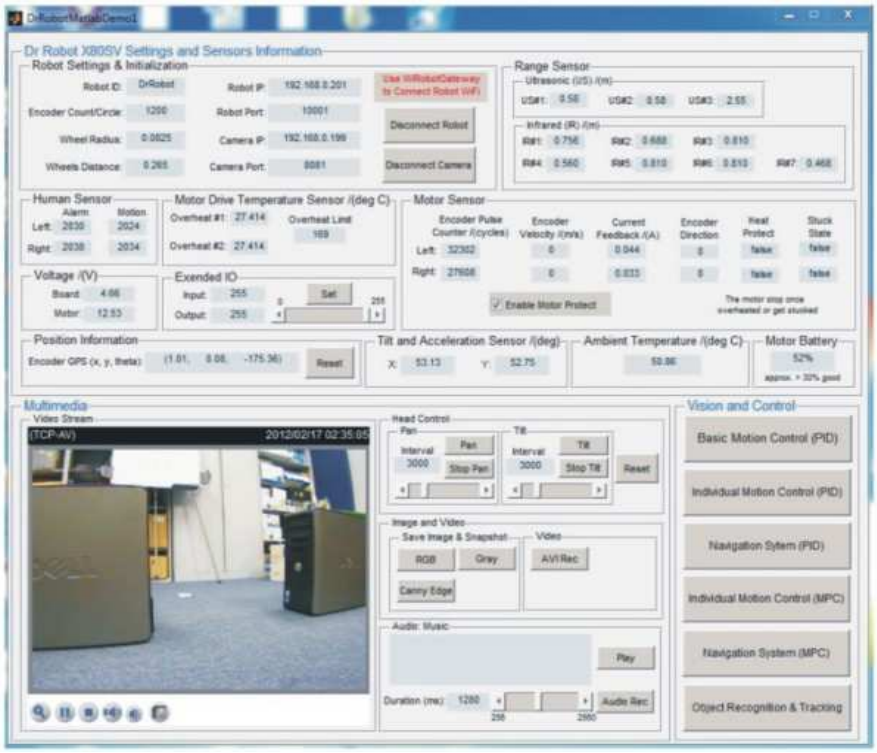

Fig. 19. MATLAB GUI showing the Dr robot X80SV settings and sensors information, multimedia and control

(a) (b)

(c) (d)

(e) (e)

Fig. 20. (a-f) Experimental setup showing sequence of the Dr robot X80SV movement

Currently, the Model Predictive Control (MPC) and the 'Object Recognition and Tracking' tools are not working; these are future research work. The 'Basic Motion Control' makes the robot perform operations such as move forward, backward, rotate, etc. The 'Individual Motion Control (PID)' makes the robot 'move to a point', 'move to a pose', 'follow a path' and 'avoid obstacles. The 'Navigation System (PID)' makes the robot to move to a goal in the presence of obstacles.

Figure 20a-f above shows an experimental setup showing a sequence of movement of the Dr Robot X80SV implementing the navigation control algorithm.

5. RESULTS AND DISCUSSION

5.1. Simulation of Individual Controllers Results

The time domain Simulink simulations were carried out over a 10 sec duration for each model (refer to the simulation setups in **Fig. 12-15**). The trajectories in **Fig. 21** were obtained by using proportional gains of 0.5 and 4.0 for K_v and K_h respectively; the final goal point was (4.9920, 5.0036), compared to the desired goal of (5,5) for the (5,9,π) initial state.

Trajectories in **Fig. 22** were obtained using K_ρ = 3.0, K_α = 8.0 and K_β =-3.0; the final pose was (4.9451, 5.0004, 2.7693), compared to the desired pose of (5,5,π) for the (5,9,π) initial state.

The trajectories in **Fig. 23** were obtained with K_d = 0.5 and K_h = 1.0, driving the robot at a constant speed of 1.0. The trajectory in **Fig. 24** was obtained using K_h = 5, PID controller gains of K_p = 1.0, K_I = 0.5 and K_D = 0.0 and the goal was a point generated to move around the unit circle with a frequency of 0.2 Hz.

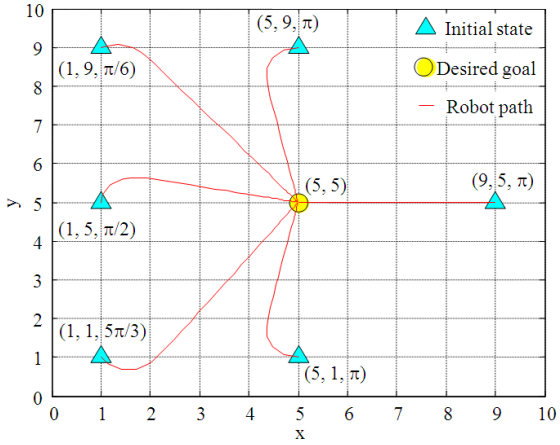

Fig. 21. Move to a point

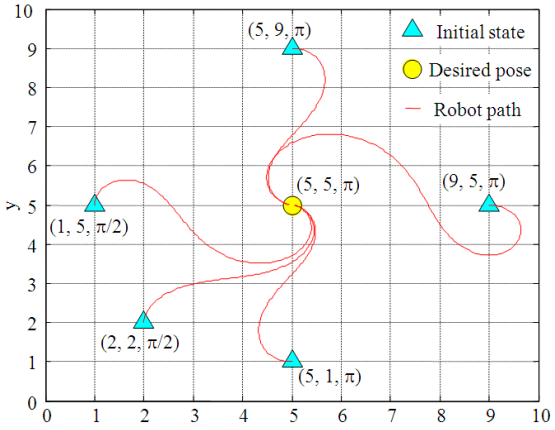

Fig. 22. Move to a pose

5.2. Simulation of Navigation System Results

The trajectory shown in **Fig. 25a** (refer to the simulation setup in **Fig. 16**) was obtained by using PID

controller gains of K_p = 5.0, K_I = 0.01 and K_D = 0.1, α = 0.6, ϵ = 0.05, ν = 0.1 m s^{-1}, initial state of (0,0,0) and desired goal of (1, 1,π/2). The final goal point associated with the simulation was (1.0077, 0.9677, 1.1051) and the average stabilization time was about 35s.

5.3. Experimental Results

The trajectory shown in **Fig. 25b** (refer to a similar experimental setup in **Fig. 20**) was obtained by using PID controller gains of K_p = 1000, K_I = 1000 and K_D = 5 for the position control and K_p = 10, K_I = 0 and K_D = 1 for the velocity control, ϵ = 0.01, ν = 0.5 m s^{-1}, initial state of (0,0,0) and desired pose of (2,0,0). The final pose associated with the experiment was (200197, 0.0266,-0.0096) and the average stabilization time was about 50s.

Even though there was steady-state errors in the values obtained, the result was encouraging. Possible causes of the errors are friction between the robot wheels and the floor, imperfection in the sensors and/or unmodeled factors (e.g., friction and backlash) in the mechanical parts of the DC motor. Moreover, despite the apparent simplicity of the kinematic model of a WMR, the existence of nonholonomic constraints (due to state or input limitations) turns the PID-feedback stabilizing control laws into a considerable challenge; due to Brockett's conditions (Brockett, 1983), a continuously differentiable, time-invariant stabilizing feedback control law cannot be obtained.

Note that during the experiments the robot sometimes got lost or wandered around before arriving at the desired pose. This is because the navigation system is not robust. It was built using a low-level planning based on a simple model of a point mass and the application of a linear PID controller.

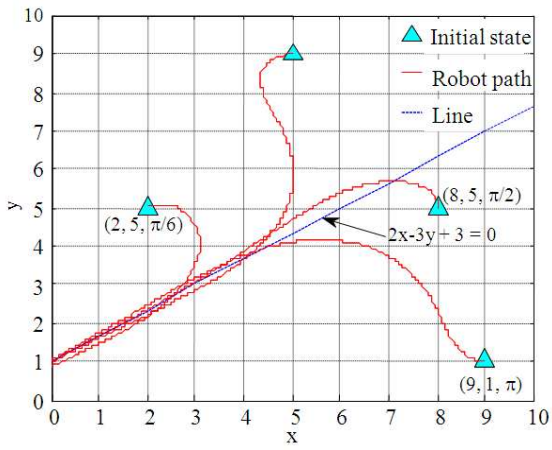

Fig. 23. Follow a line

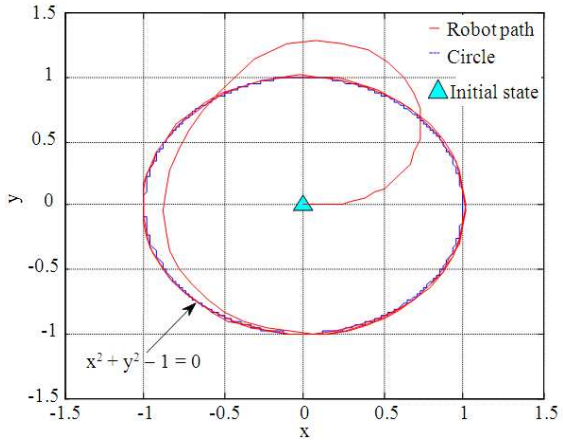

Fig. 24. Follow a circle

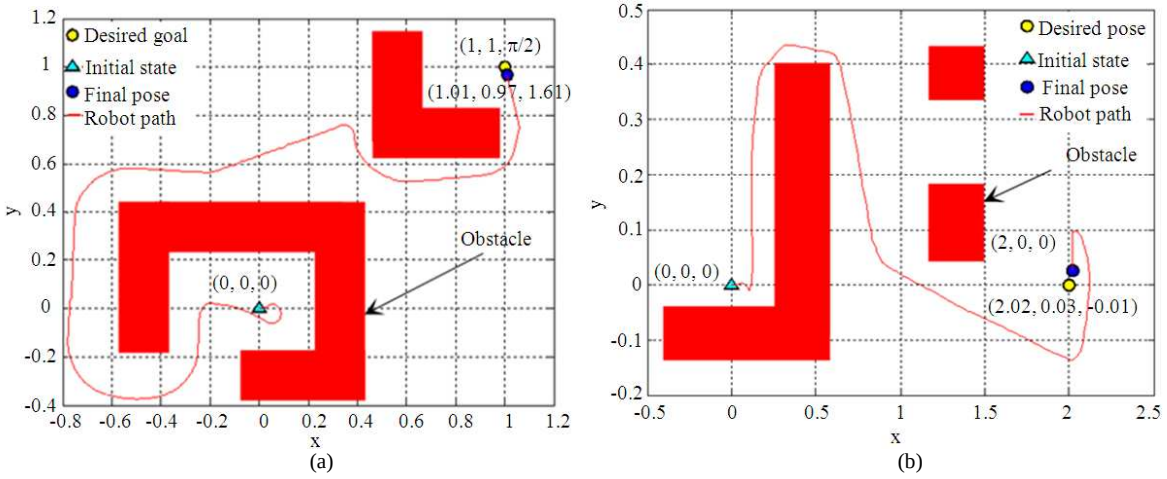

Fig. 25. Trajectory in xy-plane (a) Simulation (b) Experiment

6. CONCLUSION

In this study, an effective navigation control algorithm was presented for a DDWMR, simulated using a MATLAB robot simulator that mimics the K3 robot and implemented on the Dr Robot X80SV platform using a developed MATLAB GUI. The algorithm makes the robot move to a pose, whiles avoiding obstacles in the way. Even though the final steady-state values obtained from the experiments could not be stabilized, the results were encouraging.

This research paper has also demonstrated how to make a unicycle, kinematically equivalent to DDWMR, 'move to a point', 'move to a pose', 'follow a line', 'move in a circle' and 'avoid obstacles'. These were simulated using Simulink models.

In future, the authors will integrate real-time vision-based object detection and recognition to address the imperfection of the IR and ultrasonic range sensors. In addition, application of an optimal control strategy such as MPC to handle the nonholonomic constraints of the WMR will be studied. Furthermore, parameters such as length of path or journey time optimization will be considered.

7. ACKNOWLEDGMENT

This study was supported by the NC space grant.

8. REFERENCES

Brockett, R.W., 1983. Asymptotic stability and feedback stabilization: Differential geometric control theory. Birkhauser, Boston.

Corke, P., 2011. Robotics, Vision and Control: Fundamental Algorithms in MATLAB. 1st Edn., Springer, Berlin Heidelberg Springer, ISBN-10: 3642201431, pp: 570.

Corke, P., 1993-2011. Robotics Toolbox for MATLAB (Version 9.8) (Software).

De Luca, A., G. Oriolo and M. Vendittelli 2001. Control of Wheeled Mobile Robots: An Experimental Overview. In: RAMSETE: Articulated and Mobile Robotics for Services and Technology, Nicosia, S. (Ed.), Springer, Berlin, ISBN-10: 3540420908, pp: 181-226.

Egerstedt, M., 2013. Control of mobile robots.

Jones, J.L. and A.M. Flynn, 1993. Mobile Robots: Inspiration to Implementation. 1st Edn., A K Peters, Wellesley, ISBN-10: 1568810113, pp: 349.

MATLAB Robot Simulator (Software), 2013. Georgia Institute of Technology, Georgia: Georgia Tech Research Corporation.

Robot, 2008. C# Advance X80 demo program (Software.