UNIVERSITY OF GENOVA

FACULTY OF ENGINEERING

EUROPEAN MASTER ON ADVANCED ROBOTICS (EMARO)

A Patrol Chaotic Mobile Robots for Hostile Agents Environment

By

Raffaello Camoriano

Gamal El Ghazaly

2012

1. Introduction

During the last two decades, significant research effort has been devoted to developments of mobile robots that solve real world applications. Among these applications are performing floor cleaning tasks [3], executing industrial transportation tasks, [4], and scanning hostile environments or dangerous areas that may have explosive devices like mines [5].

This project is concerned with the planning the motion mobile robots to patrol a defined region subjected to the existence of mobile hostile agents with unpredictable motion. In such applications, the criterion is to plan the motion of the robot to have high unpredictable motion in addition to fast scanning capability.

One of interesting methods that offers these criteria is to make use of the highly unpredictable response of chaotic systems. If a mobile robot is driven by a dynamic system with chaotic behavior, the motion of the robot will be unpredictable. The integration between chaotic systems are mobile robots has been addressed in [2]. Where a chaotic system called Arnold has been successfully used to drive a mobile robot to obtain unpredictable chaotic behavior of the robot.

The main objective of this project is to implement a real-time simulation to a chaotic mobile robot that works as a patrol in a given specified area. This robot will scan the environment chaotically searching for mobile hostile agents that are also have chaotic behavior.

The organization of this report is as follows. In section 2, a theoretical description of mobile robot model and how a chaotic motion can be generated, are covered. Implementation and real-time issues of the will be covered in section 3. A future research direction is given in section 4. Section 5 concludes this report.

2. Chaotic Mobile Robot

2.1 Mobile Robot Model

The mobile robot considered in this project is a differential motion robot with two DOF. Fig. 1 shows the structure of the robot which is composed of two active, parallel, and independent wheels and a third passive wheel for equilibrium and free steering of the robot.

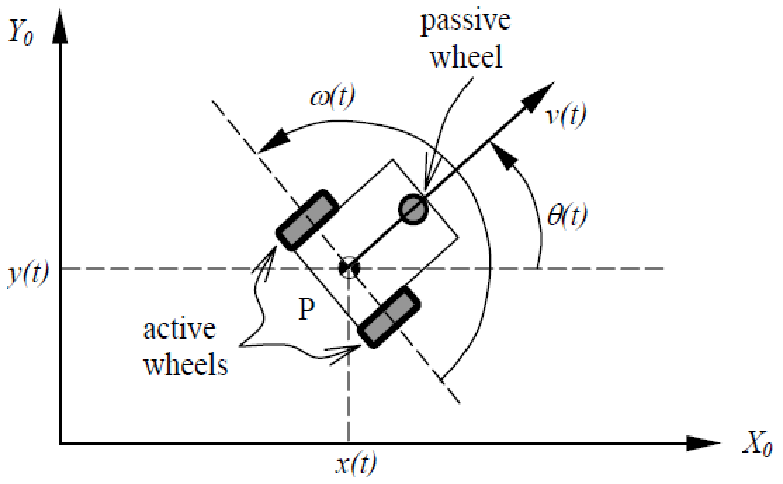

Fig. 1 Mobile Robot

The robot is considered as a rigid body operating horizontally in (x, y) plane, and its motion is obtained by driving the active wheels. The resultant motion is described in terms of linear velocity $v(t)$ and direction $\theta(t)$ describing an instantaneous linear motion of the medium point of the wheel axis and a rotational motion (rotational velocity $\omega(t)$) of the robot body over this same point.

The robot motion in (x, y) plane can be controlled by manipulating the angular speeds of the right and left wheels $\omega_r(t)$ and $\omega_l(t)$ or equivalently $v(t)$ and $\omega(t)$. The mathematical model of the robot is given by

$$\begin{bmatrix} \dot{x}(t) \\ \dot{y}(t) \\ \dot{\theta}(t) \end{bmatrix} = \begin{bmatrix} \cos\theta & 0 \\ \sin\theta & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v(t) \\ \omega(t) \end{bmatrix} \tag{1}$$

A Chaotic mobile robot implies a mobile robot with a controller that ensures chaotic motion. Our objective is to drive the robot by a dynamic system such that the robot scans the patrol area with a chaotic trajectory.

2.2 Mobile Robot Chaotic Trajectories by Arnold Equation

Chaotic systems are a class of nonlinear systems with a very complex dynamic behavior. One of the main characteristics of chaotic systems is system dependence on initial condition. In this subsection, we will show how the so called Arnold chaotic is integrated to the model of the mobile robot (1) such that a chaotic motion in (x, y) plane is obtained. Arnold equation is a third-order autonomous nonlinear system of the form

$$\begin{bmatrix} \dot{x}_1(t) \\ \dot{x}_2(t) \\ \dot{x}_3(t) \end{bmatrix} = \begin{bmatrix} A\sin(x_3(t)) + C\cos(x_2(t)) \\ B\sin(x_1(t)) + A\cos(x_3(t)) \\ C\sin(x_2(t)) + B\cos(x_1(t)) \end{bmatrix} \tag{2}$$

where A, B, C are constants. Arnold equation is shown to exhibit a chaotic motion with some range of parameters values and initial condition [1]. The system has been successfully integrated with mobile robot in [2].

In order to integrate Arnold equation (2) with mobile robot (1), the following state variables are defined

$$\begin{aligned} \dot{x}_1(t) &= D\dot{y}(t) + C\cos(x_2(t)) \\ \dot{x}_2(t) &= D\dot{x}(t) + B\sin(x_1(t)) \\ x_3(t) &= \theta(t) \end{aligned} \tag{3}$$

where D is a constant. Substituting (1) into (3), we can obtain

$$\begin{aligned}\dot{x}_1(t) &= Dv(t) + C\cos(x_2(t)) \\ \dot{x}_2(t) &= Dv(t) + B\sin(x_1(t)) \\ \dot{x}_3(t) &= \omega(t)\end{aligned} \tag{4}$$

Consequently, if the mobile robot is driven by the following control inputs

$$\begin{aligned} v(t) &= \frac{A}{D} \\ \omega(t) &= C\sin(x_2(t)) + B\cos(x_1(t))\end{aligned} \tag{5}$$

We can obtain the following overall system

$$\begin{bmatrix}\dot{x}_1(t) \\ \dot{x}_2(t) \\ \dot{x}_3(t) \\ \dot{x}(t) \\ \dot{y}(t)\end{bmatrix} = \begin{bmatrix} A\sin(x_3(t)) + C\cos(x_2(t)) \\ B\sin(x_1(t)) + A\cos(x_3(t)) \\ C\sin(x_2(t)) + B\cos(x_1(t)) \\ v\cos(x_3(t)) \\ v\sin(x_3(t))\end{bmatrix} \tag{6}$$

It is clear from (5) and (6) that the robot is driven by a constant linear velocity and the angular velocity depends on the first two states of Arnold equation. Fig. 2 illustrates how mobile robot is driven by chaotic system.

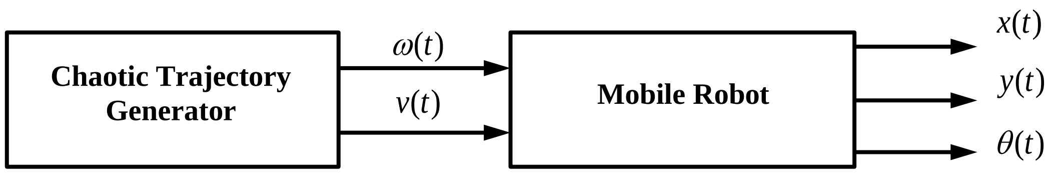

Fig. 2 Chaotification of Mobile Robot

The mobile robot for this application is assumed to be equipped with some sort of proximity sensing system. Once one of the boundaries of the patrol area is detected, the robot should reflect its heading angle $\theta(t)$ according to the mirror condition shown in Fig. 3. This condition allows the robot to be confined inside the patrol area.

$$\theta(t) = \theta(t) - \pi/2 \quad \text{if} \quad (x, y) \text{ on a boundary} \tag{7}$$

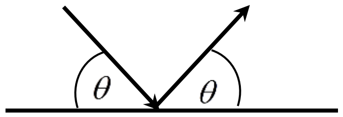

Fig. 3 Mirror Reflecion Condition

Now, in order to see the chaotic motion of the robot and how it scans and homogenously covers the patrol area, the overall dynamics of the mobile robot equipped with the chaotic Arnold system of solved by numerically by MATALB Rung-Kutta solver.

The overall system (7) is an autonomous system. The system with following parameters and initial condition exhibits chaotic motion.

$$v(t) = 0.7 \quad m/\sec$$
$$A = 0.5, \; B = 0.25, \; C = 0.25$$
$$x_1(0) = 1, \; x_2(0) = 1, \; x_1(0) = 0$$
$$x(0) = 6.5, \; y(0) = 6.5$$

Fig. 4 – Fig. 6 shows the chaotic trajectory of the mobile robot in [10 m x 10 m] map with the given parameters and initial conditions but with different simulation period. The mirror condition will be applied at the borders *x = 0, x = 10 and y = 0, y = 10*. These given results show the trajectory of the mobile robot inside that borders of the map. However, the robot many cross the borders of the map by small distance and return again to the map. This phenomena happens due to the fact that robot has a dynamic nature and rotation caused by mirror.

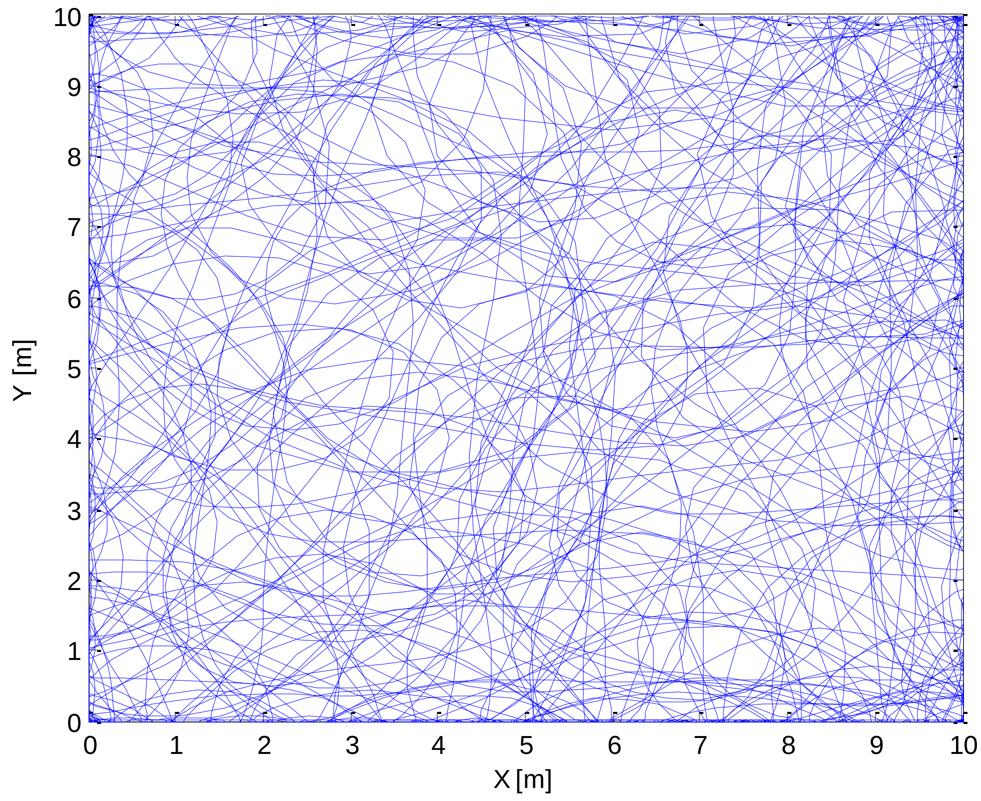

Fig. 4 Mobile Robot Trajectory with Period T = 5x10^{3}

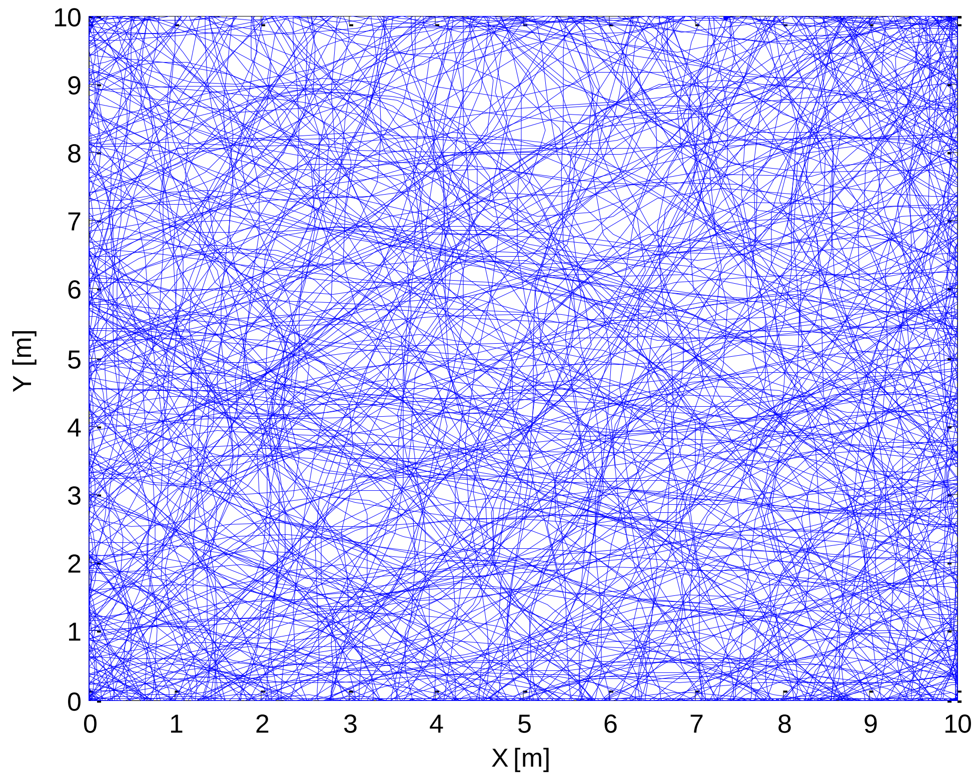

Fig. 5 Mobile Robot Trajectory with Period T = 10^{4}

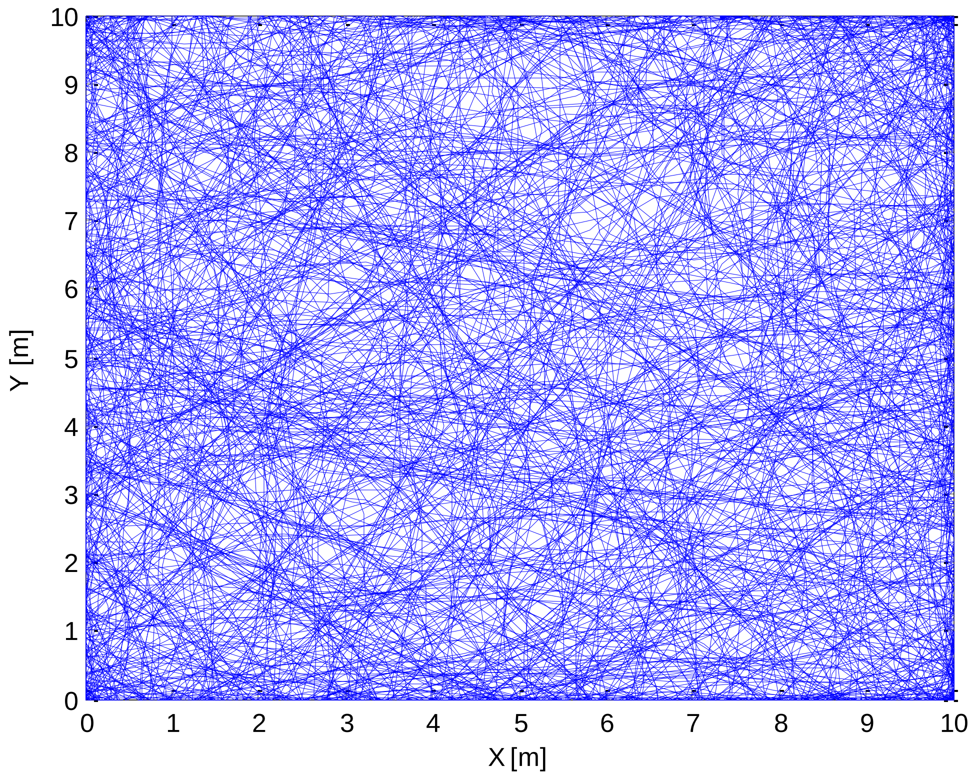

Fig. 6 Mobile Robot Trajectory with Period T = 5x10^4

It is clear from Fig. 4 to Fig. 6 that the mobile robot can scan the patrolling area homogenously and even increasing the patrolling period *T* homogeneity also is maintained with better coverage. In [2] a comparison between the coverage achieved by chaotic motion and random motion for the robot is done [2]. It was shown that chaotic motion of robots guarantee homogeneity and rapid coverage compared to ones with random motion That is why mobile robot can be used for patrol application against hostile agents that could appear the area.

2.3 Chaotic Mobile Robot as a Patrol in Hostile Environment

In this section, we will show how a chaotic mobile robot can be used as a patrol to a given area which contains hostile agents also with chaotic motion. As stated before, one of the main characteristics of chaotic systems is the sensitive dependency on initial condition. Even s small difference between the initial states, the chaotic trajectory of the robot will be different. Consider the following two close initial condition and the same previous parameters

<u>First Initial Conditions</u>

$x_1(0) = 1$, $x_2(0) = 1$, $x_1(0) = 0$, $x(0) = 6.5$, $y(0) = 6.5$

<u>Second Initial Conditions</u>

$x_1(0) = 1.01$, $x_2(0) = 0.99$, $x_1(0) = 0$, $x(0) = 6.5$, $y(0) = 6.5$

Fig. 7 shows two chaotic trajectories with the given different initial conditions. Obviously, even the difference between initial conditions is very small, the trajectories are completely different.

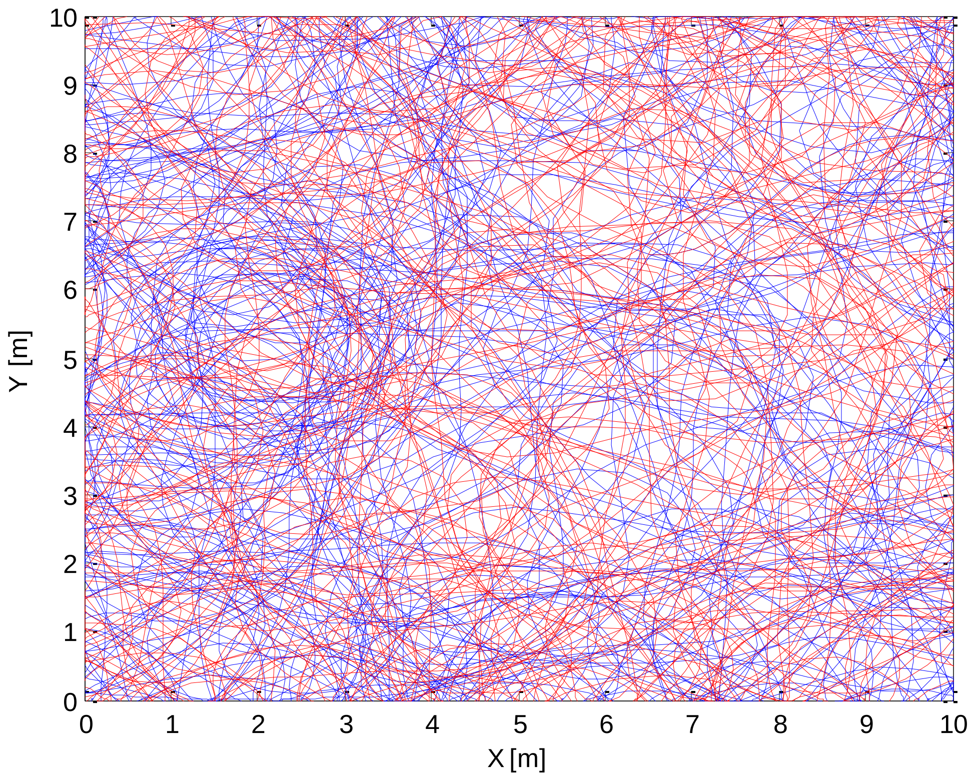

Fig. 7 Chaotic Trajectory with Two Different Initial Conditions

The main of this project to generate a number of agents with chaotic trajectory with slightly different initial conditions that run autonomously in the environment. Each trajectory represents the motion of an agent. All agents including the robot have the same dynamical power in terms of unpredictable motion. Except, the robot is assumed to be equipped with a remote sensing facility to be able to detect agents. Once an agent enters the combat range of the mobile robot it will be killed. The details of real-time implementations of this idea are covered in the section 3.

3. Real Time Simulation of Chaotic Mobile Robot

3.1 Overview

The simulator has been implemented using POSIX threads, time structures and synchronization techniques. The main program configures all the necessary tools and initializes the threads corresponding to the agents. Then, it enters a "proximity routine" which verifies at each step if one or more hostile agents enter the area patrolled by the mobile robot. If one hostile agent is captured, the routine modifies the mode of execution of the corresponding thread, which becomes a "ghost". The modes of execution of all the agent threads are stored in a shared variable, which is protected by a dedicated mutex. A thread in "ghost" mode does not interact with other threads, and the only operation it performs is freeing the barriers to avoid blocking other threads.

On the other hand, if more than a single hostile agent enters the proximity area of the mobile robot, the mobile robot thread is killed and the simulation terminates. The threads are terminated by changing their mode, like in the "ghost" case, so that they terminate the execution by calling the *pthread_exit* function. This allows the main program to safely join the threads.

Agent thread execution modes:

- **Alive [0]:** compute the next position (see section 3.2).

- **Ghost [1]:** free the barriers.
- **Terminate [2]:** call *pthread_exit* in order to be joined by the main program.

Each thread computes the new position of the corresponding agent at each step, and, consequently, it updates its state in the shared variable X. After the computation, it waits for the other threads to complete the computation and then triggers the proximity routine synchronization barrier. The agent threads must wait for the completion of the routine's execution. For this purpose, the previous barrier is called for a second time.

3.2 Implementation of the chaotic Robot's Dynamic Equation

As previously anticipated, each thread corresponds to an agent (either the mobile robot or a hostile agent) and computes its next state at each step. In this section, the implementation of the solution of the chaotic robot dynamic equation (6) in the agent threads will be covered.

The dynamic equation is a set of differential equations, which need to be solved at each step using an approximated numerical method. In our case, the 4th order Runge-Kutta method has been used. The implementation of this method can be found in function *RK4*, which is called by each agent thread for numerically solving the dynamic equation expressed in function *DE*.

After the computation has been completed, the agent thread stores the new state in the shared variable, so that it can be checked by the proximity routine.

3.3 Tools

In this project, a wide use of shared variables for thread synchronization has been made. For instance, the position of the agents on the map is contained in the shared bidimensional array X, which is read by the proximity routine only after all the new positions have been computed. The synchronization between threads, and between threads and the proximity routine, has been implemented by using a barrier with strength N+1 (number of agents plus proximity routine).

A barrier has also been used to synchronize the GUI thread and the main program at the beginning of the simulation, starting the displaying of agents on the screen only after the associated threads have been initialized and the welcome message shown.

The previously mentioned shared variable, and the one containing the mode of the agent threads and of the GUI thread, together with the final result of the simulation to be passed to the GUI thread, are protected by mutexes.

Readers-writer locks could be used if the number of working agent threads was higher, in order to increase performance and avoid blocking threads on mutexes when they only have to read a position. However, for a relatively small number of threads as ours, mutexes proved themselves to be more than sufficient for guaranteeing the functioning of the overall system.

As far as time primitives are concerned, The nanosleep function has been used to introduce a delay at the beginning of the simulation, to let the GUI display a welcome message.

3.2 The GUI

The GUI is implemented in a dedicated thread, which has a lower priority than the agent threads and the main program. This thread must interfere as little as possible with the execution of the simulation. Its main task is to retrieve the position of the agents in the map (not necessarily at each step) and draw them on the screen. In order to avoid slowing down the simulation, the access to the shared variable containing the positions of the robots and their state has been implemented using a *pthread_trylock*, rather than the classic *pthread_lock*.

The 2D graphic library SDL (Simple DirectMedia Layer) has been chosen as the main tool for implementing the GUI thread. Installing *SDL 1.2* is necessary for compiling the source code of the program. At each cycle, the thread reads the positions and places the markers corresponding to the alive agents on the map. Then, it refreshes the screen and checks if the user has triggers the “close window” event. In that case, the GUI thread terminates, but the simulation goes on and the results are printed in the shell.

Note: The gcc command for compiling the project is the following:

```
gcc -pthread CMR.c -lm -lrt -lSDL
```

4. Future Work

As an extension to this work, one of the main interesting directions for future implementation is to use chaos synchronization schemes to control the chaotic mobile robot to follow the agent instead of scanning the patrol area. Synchronization of chaotic systems is one of the challenging tasks in control of nonlinear systems.

In chaos synchronization, there are two chaotic systems, one is called master system and the other is called slave system. If the two systems have different initial condition and if they run autonomously, they will have different trajectories. If the master system has access to the state vector of the slave, it can be forced by a control input to follow the slave system.

For patrol application, if we consider the chaotic mobile robot as a master system and one of the hostile agents as a slave system. We design a control input to the chaotic mobile robot to force it to follow the hostile agent i.e. to kill it.

However, the main problem is that neither the state vector of the hostile agent nor its model is known. A possible solution to this problem which we are going to consider as our future work is

1- Use a localization mechanism to estimate the coordinates of the agents which represent two states.
2- Using the estimated position we can design an observer to estimate the whole state vector of the agent.
3- Once the state vector becomes available, we can identify the model of the agent using one of the computational intelligent techniques such as neural networks or fuzzy systems.

5. Conclusion

In this project, a real-time simulation of a chaotic mobile robot for patrolling an environment with mobile agents has been successfully implemented. The dynamics of the chaotic mobile robot and agents are composed of an Arnold chaotic system and a mobile robot dynamics. The chaotic mobile robot and agents are initialized with slightly different initial conditions to have a different trajectory for each agent. It has been show by real-time simulation that the chaotic mobile robot can scan the patrol area in a reasonable amount of time and kill all hostile agents.

6. References

[1] H. Okamoto and H. Fujii, *Nonlinear Dynamics, Iwanami Lectures of Applied Mathematics* (in Japanese) Iwanami, Tokyo, 1995, vol. 14.

[2] Y. Nakamura and A. Sekiguchi, The Chaotic Mobile Robot, *IEEE Transactions on Robotics and Automation*, Vol. 17 No. 6, 2001.

[3] J. Palacin, J. A. Salse, I. Valganon, and X. Clua, Building a mobile robot for a floor-cleaning operation in domestic environments, *IEEE Transactions on Instrumentation and Measurement*, vol. 53, no. 5, pp. 1418–1424, 2004.

[4] J. H. Suh, Y. J. Lee, and K. S. Lee, Object-transportation control of cooperative AGV systems based on virtual-passivity decentralized control algorithm, *Journal of Mechanical Science and Technology*, vol. 19, no. 9, pp. 1720–1730, 2005.

[5] B. Rooks, Robotics outside the metals industries, *Industrial Robot*, vol. 32, no. 3, pp. 205–208, 2005.