NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

ACOUSTIC MOTION ESTIMATION AND CONTROL FOR AUTONOMOUS UNDERWATER VEHICLES

by

Hakkı Çelebioğlu

June, 1997

Thesis Advisor: Roberto Cristi

Approved for public release; distribution is unlimited.

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE June 1997	3. REPORT TYPE AND DATES COVERED Master's Thesis
4. TITLE AND SUBTITLE ACOUSTIC MOTION ESTIMATION AND CONTROL FOR AUTONOMOUS UNDERWATER VEHICLES		5. FUNDING NUMBERS
6. AUTHOR(S) Çelebioğlu, Hakkı		
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000		8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)		10. SPONSORING/MONITORING AGENCY REPORT NUMBER
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.		
12a. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.		12b. DISTRIBUTION CODE

13. ABSTRACT *(maximum 200 words)*

An integrated model of acoustic motion estimation and control is presented. The control system is designed on the basis of the definitions of suitable Lyapunov functions for the different maneuvers in approaching a target. These functions allow the navigation and maneuvering tasks to be performed in a two-layered hierarchical architecture for closed-loop control. The motion estimation algorithm uses pencil beam profiling sonar range and bearing information. The operating environment is modeled with a suitable three-dimensional potential function and its gradient which forms an attractive field. This algorithm provides satisfactory performance for autonomous navigation and obstacle avoidance.

The applicability and robustness of this model are demonstrated with both actual test data obtained with the NPS *Phoenix* submersible and computer generated simulation data. The results show the effectiveness of the combined estimation and control model.

14. SUBJECT TERMS Autonomous Underwater Vehicle, navigation, guidance and control systems, Lyapunov functions, Extended Kalman Filtering			15. NUMBER OF PAGES 104 16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18 298-102

Approved for public release; distribution is unlimited.

ACOUSTIC MOTION ESTIMATION AND CONTROL FOR AUTONOMOUS UNDERWATER VEHICLES

Hakkı Çelebioğlu
Lieutenant Junior Grade, Turkish Navy
B.S., Turkish Naval Academy, 1991

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
June 1997

Author: _____
Hakkı Çelebioğlu

Approved by: _____
Roberto Cristi, Thesis Advisor

_____
Xiaopping Yun, Second Reader

_____
Herschel H. Loomis, Jr., Chairman
Department of Electrical and Computer Engineering

ABSTRACT

An integrated model of acoustic motion estimation and control is presented. The control system is designed on the basis of the definitions of suitable Lyapunov functions for the different maneuvers in approaching a target. These allow the navigation and maneuvering tasks to be performed in a two-layered hierarchical architecture for closed-loop control. The motion estimation algorithm uses pencil beam profiling sonar range and bearing information. The operating environment is modeled with a suitable three-dimensional potential function and its gradient which forms an attractive field. This algorithm provides satisfactory performance for autonomous navigation and obstacle avoidance.

The applicability and robustness of this model are demonstrated with both actual test data obtained with the NPS *Phoenix* submersible and computer generated simulation data. The results show the effectiveness of the combined estimation and control techniques.

TABLE OF CONTENTS

ACKNOWLEDGMENTS

This thesis is dedicated to my parents whose guidance and encouragement inspire me to make them proud.

I would like to express my thanks to Professor Roberto Cristi for his help and encouragement during this study. His technical expertise and insights were invaluable to me. His patience, maturity, and understanding of human nature helped me to overcome many difficulties.

I. INTRODUCTION

A. GOALS

An Autonomous Underwater Vehicle (AUV) is a self-contained unmanned vehicle used for underwater missions such as surveying, pollution detection, or mine detection and neutralization. In order to be deemed truly autonomous the vehicle must be able to independently follow a mission plan, interact with its environment, accomplish a goal and return to a predesignated destination.

This thesis addresses the problem of guidance and control of autonomous vehicles. In particular, we will be focusing on the underwater applications, with the aid of sonar and inertial sensors to detect features in the environment. We assume the environment to be unknown at the beginning of the mission, and the vehicle must be capable of detecting and reconstructing possible obstacles. Then, the next step is to estimate the motion of the vehicle in this environment and keep the vehicle on the desired trajectory by applying a closed-loop guidance and control system.

B. METHOD OF APPROACH

In this study, a guidance and control system based on a two layered hierarchical architecture for closed-loop control has been integrated with an acoustic motion estimator designed for navigation in structured environments. The computer simulations of both the control system and the estimator are presented together with the combined system.

The guidance and control system design is based on the approach which tackles the problems of using control laws based on Lyapunov functions. These allow the navigation and maneuvering tasks to be performed in a closed-loop in the presence of perturbations and modeling errors. The simulation results showing the possible effectiveness of this closed-loop system are also provided.

In the motion estimation part, we address the problem of a vehicle navigating both in an environment which could be entirely or partially known [Ref. 1]. The assumption on the obstacles is that their reflective surfaces are piecewise modeled by segments having constant orientation in two dimensions. We assume the vehicle's initial location and

velocity are known, and we measure its acceleration. The goal of the algorithm is to provide the best estimate of the vehicle's motion and of the segments of the environment, in terms of location and orientation. The environment is modeled by a potential function based on the estimated location of the reflecting surfaces and their orientation. Then, this potential function estimate of the environment is used to correct the vehicle's estimated trajectory. By the potential function approach, through the range and bearing measurements from a scanning sonar, the motion trajectory of the vehicle is estimated using Kalman filtering techniques. The simulation results and program codes are presented.

Finally, the combination of two systems yields a system of acoustic motion estimation and control. The measurements from a scanning sonar are recursively processed in the estimation algorithm, and the result is an estimate of the current position of the vehicle. The estimated vehicle's position is processed in the control algorithm to make it follow a desired trajectory.

C. ORGANIZATION

This thesis is organized in five parts. Chapter II provides the theory behind the design of the vehicle and the environment models, as well as the interaction between the two, provided by the sonar system. The Kalman filter-based algorithm which is used to estimate the motion of the vehicle, is detailed. The simulation results due to this approach are also included.

Chapter III deals with the guidance and control algorithms based on the Lyapunov control techniques. The control algorithm which includes two loops: The outer control loop which corresponds to three phases of the vehicle approach for the target, and the inner control loop which is determined by the vehicle's dynamics, is detailed. The simulation results using MATLAB SIMULINK software package are also included.

Chapter IV is based on the combined model of the vehicle motion estimation and control system. After giving some brief definitions, the MATLAB SIMULINK model of the design is detailed and the results are presented.

Chapter V summarize the results and conclusions of this study and provides several recommendations for follow-on research.

II. ACOUSTIC MOTION ESTIMATION MODELING FOR AN AUV IN A STRUCTURED ENVIRONMENT

A. GENERAL

In this chapter, a nonlinear model for the dynamics of an AUV and the environment in which it operates, is set up. Basically, it is intended to determine a framework, so that we can combine inputs from the inertial navigation system (INS - gyros, accelerometers) with sonar for the localization of the vehicle with respect to the known initial conditions (position and velocity). A mission in which the vehicle explores the surroundings guided by its INS equipment while it maps the environment, is performed.

B. ENVIRONMENT MODEL

We define environment to be the locus E of reflecting surfaces [Ref. 1]. In order to be able to estimate the vehicle's motion, it is assumed that a certain continuity and smoothness conditions are satisfied, at least in a piecewise sense. Therefore at every point, where the sonar beam pings on the reflecting surface E, a vector φ, orthogonal to the surface itself, is defined. This can be expressed as

$$\varphi(\mathbf{q})^T \mathbf{q} = 1, \tag{2. 1}$$

where $\mathbf{q}$ is the tip of the sonar vector on the reflecting surface.

By this, it is assumed that the origin of the coordinate frame does not belong to the reflecting surfaces. For example, the origin is assumed to be the initial location of the vehicle, which is the assumption for beginning its mission in open waters.

Assuming that the sonar head continuously scans the environment, if the vehicle moves smoothly, the point $\mathbf{q}$ on the reflecting surface and the vector φ, normal to the reflecting surface, can be related in a state space equation as

$$\begin{cases} \dot{\varphi}(\mathbf{t}) = 0, & \mathbf{t} \notin \Im \\ \varphi(\mathbf{t})^T q = 1 \end{cases} \tag{2. 2}$$

with $\Im$, denoting the set of times when the slope vector φ changes. Clearly, at every point of the reflecting surfaces, the vector φ, is orthogonal to the surface.

At this point the normal vector φ can be expressed as the gradient of a potential function V [Ref. 1]. The potential function V will be defined over the whole operational space, satisfying the following necessary conditions:

- $V(\mathbf{q}) \geq 0$, for all $\mathbf{q}$;
- $V(\mathbf{q}) = 0$, if and only if the point $\mathbf{q}$ belongs to the reflecting surface defined on the map;
- For any small perturbation $\Delta\mathbf{q}$, the gradient of $V(\mathbf{q})$ is such that $0 \geq \Delta\mathbf{q}^T \nabla V(\mathbf{q}+\Delta\mathbf{q})$, which states that the vector field is attractive towards the reflecting surface, and also its gradient is orthogonal to this surface.

The critical point is the last property above, which shows that the two vectors φ and ∇V are parallel. The reason behind the orhogonality of the gradient has been explained in Ref. 1, and will be recalled later in the chapter.

The sonar returns produced by the sonar installed on the AUV, provide the required interaction between the AUV dynamic model and the environment, through the definition of the potential function. A summary of the operating characteristics of the high frequency scanning sonar installed on the NPS Phoenix AUV is included as Appendix A and a typical potential function for a rectangular pool is presented as Figure 2.1 and Figure 2.2.

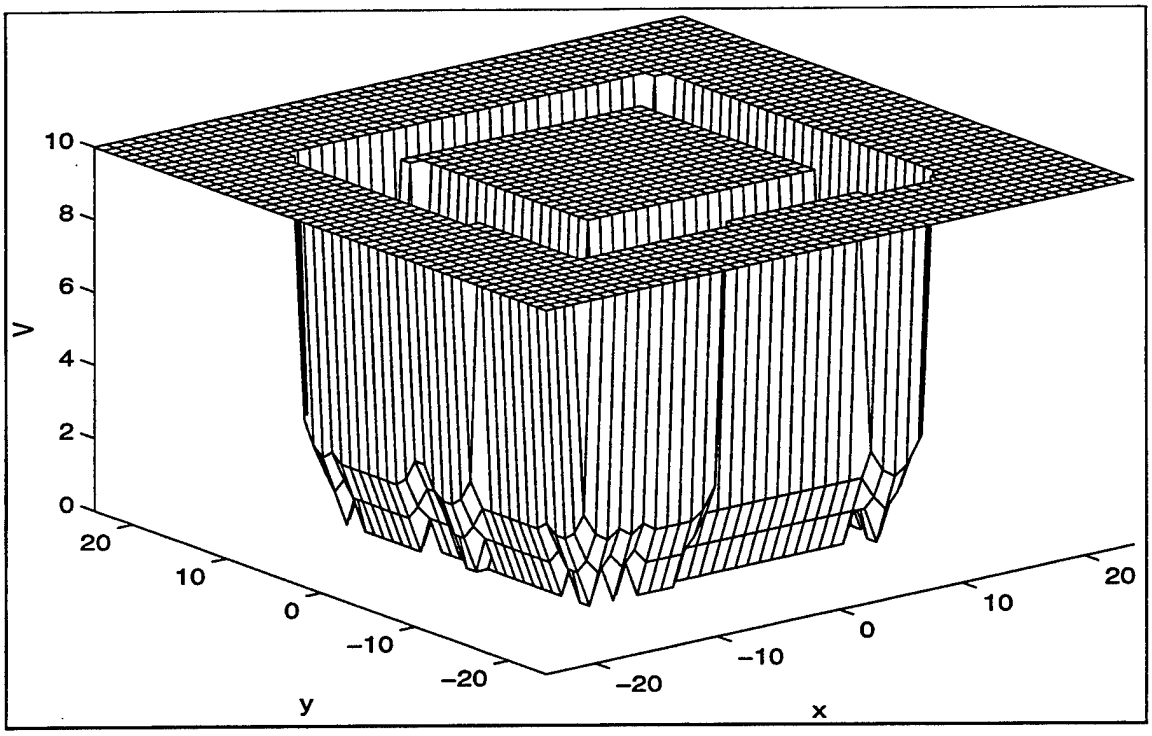

Figure 2.1 Three Dimensional Potential Function of a Rectangular Pool (24x24 ft).

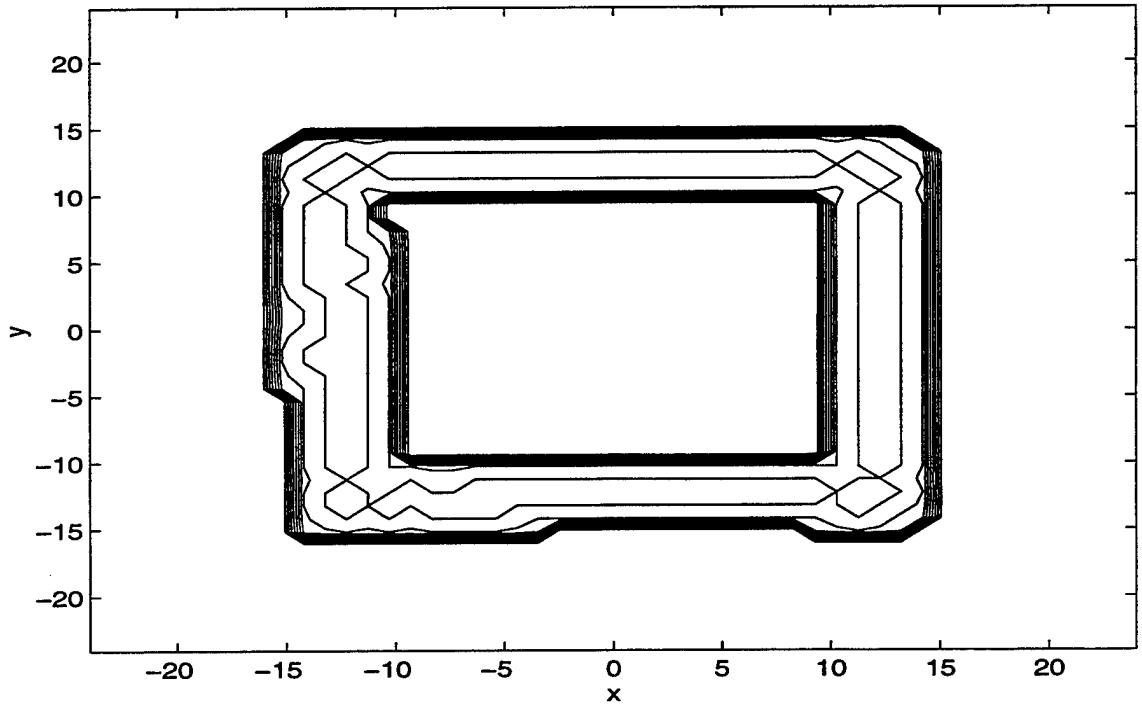

Figure 2.2 The Contours of the Potential Function of a Rectangular Pool(24x24 ft).

C. VEHICLE MODEL

The dynamics of the vehicle moving in an environment is represented by a state vector determined by its position vector **p** with respect to a reference frame fixed with the environment, a velocity and orientation vector **z**, and a command vector **u**, involving the commands given to the actuators of the vehicle. The dynamics can be expressed as a combination of the kinematics and the dynamic equations of the form [Ref. 1],

$$\begin{cases} \dot{\mathbf{p}} = f_1(\mathbf{z}), \\ \dot{\mathbf{z}} = f_2(\mathbf{z}, \mathbf{u}). \end{cases} \tag{2.3}$$

The mappings represent the kinematics and the dynamics relations respectively. In the case of the acceleration being measured by its inertial navigation system (INS), as we would use for our purposes, we can write the vehicle's equations using purely the kinematics. Denoting **a**, the measured acceleration vector and **v** its velocity, this becomes

$$\begin{cases} \dot{\mathbf{p}} = \mathbf{v}, \\ \dot{\mathbf{v}} = \mathbf{a}. \end{cases} \tag{2.4}$$

The main component of the estimation process is based on the dynamic model of the vehicle which is defined in equations (2.3) and (2.4).

The measurement vector ρ, consists of the dynamic observations of the sonar range **r**, at an angle θ, with respect to the longitudinal axis of the vehicle, with **w** indicating measurement noise [Ref. 2].

$$\rho = g(\mathbf{p}, \mathbf{r}, \theta, \mathbf{w}) \tag{2.5}$$

We assume the noise sources **w** to be zero mean, white, and Gaussian, as is standard in this class of problems.

The function g is defined by the map of the environment, and apart from the measurement noise, it is zero when the sonar return information (r,θ) is consistent with the vehicle position on the map.

The criterion by which we choose g is based on the use of the potential function V, which is defined in the previous section. For a vehicle moving on a plane located on the position (x,y) with a heading γ, the function g is defined as

$$g(x, y, \gamma, r, \theta) = V(x + r\,\mathrm{Cos}(\gamma + \theta), y + r\,\mathrm{Sin}(\gamma + \theta)), \tag{2.6}$$

The vector **q**, for the location of the tip of the sonar range in the reference frame can be expressed as the equations below

$$\begin{cases} p = \begin{bmatrix} x & y \end{bmatrix}^T, \\ s = \begin{bmatrix} r\cos\theta & r\sin\theta \end{bmatrix}^T, \\ T = \begin{bmatrix} \cos\gamma & -\sin\gamma \\ \sin\gamma & \cos\gamma \end{bmatrix}, \\ q = p + Ts \end{cases} \tag{2.7}$$

where **p**, is again defining the position of the vehicle at any time, **s** is the sonar information vector consisting of the measured range and angle of the sonar return. The matrix **T** represents the coordinate transformation between the vehicle and the environment reference frames.

In order to design an algorithm for the estimation of the vehicle's position from sonar measurements, using the definitions in (2.7), the overall state space model including the vehicle motion and the environment is considered as

$$\begin{cases} \dot{\mathbf{p}} = \mathbf{v}, \\ \dot{\mathbf{v}} = \mathbf{a}, \\ 0 = \mathbf{V}(\mathbf{p} + \mathbf{Ts}), \end{cases} \tag{2.8}$$

Defining the estimated position of the vehicle as $\hat{\mathbf{p}}$, for the general case where acceleration measurement is available with an inertial navigation which includes accelerometers, the correction can be applied to the position and velocity estimate [Ref.3]. These can be updated as

$$\begin{cases} \dot{\hat{\mathbf{p}}} = \hat{\mathbf{v}} + \mu_1 \nabla \mathbf{V}(\hat{\mathbf{p}} + \mathbf{Ts}), \\ \dot{\hat{\mathbf{v}}} = \mathbf{a} + \mu_2 \nabla \mathbf{V}(\hat{\mathbf{p}} + \mathbf{Ts}), \end{cases} \tag{2. 9}$$

with μ_1 and μ_2 being positive constants.

To demonstrate local convergence of this method, let us consider a simpler problem where the velocity vector $\mathbf{v}$ is measured. Defining the position error as $\tilde{\mathbf{p}} = \hat{\mathbf{p}} - \mathbf{p}$, and combining equations (2.8) and (2.9), we obtain the error equation,

$$\dot{\tilde{\mathbf{p}}} = \mu \nabla \mathbf{V}(\hat{\mathbf{p}} + \mathbf{Ts}), \tag{2. 10}$$

and pre-multiplying by the transpose of the error yields,

$$\tilde{\mathbf{p}}^{\mathbf{T}} \dot{\tilde{\mathbf{p}}} = \mu \tilde{\mathbf{p}}^{\mathbf{T}} \nabla \mathbf{V}(\mathbf{p} + \mathbf{Ts} + \tilde{\mathbf{p}}), \tag{2. 11}$$

where we recognize that the point $\mathbf{p}+\mathbf{Ts}$ is on the reflecting surface and that for $\tilde{\mathbf{p}}$ being sufficiently small, the right-hand side of the above equation is non-positive by the property specified for the potential function. From (2.11) we can see that $\|\tilde{\mathbf{p}}\|^2$ is decreasing with time since $\frac{d}{dt}\|\tilde{p}\|^2 = -2\,\tilde{p}^T\tilde{p} < 0$ for all $\hat{p} \neq 0$.

Integration of the equation (2.11) with respect to time yields,

$$\frac{1}{2}(\|\tilde{\mathbf{p}}(\mathbf{t})\|^2 - \|\tilde{\mathbf{p}}(0)\|^2 = \mu\int_0^t \tilde{\mathbf{p}}(\tau)^T \nabla \mathbf{V}(\mathbf{p}+\mathbf{Ts}+\tilde{\mathbf{p}}(\tau))\mathbf{d}\tau \tag{2.12}$$

Since the integrand is non-positive for all τ, it will go to zero as time tends to infinity. This implies that the error vector $\tilde{\mathbf{p}}$ tends to be orthogonal to the gradient of the field, and that the point $\hat{\mathbf{p}}+\mathbf{Ts}$ tends to be on the reflecting surface.

A particular form of the gradient is obtained by defining it as

$$\nabla \mathbf{V}(\mathbf{p}+\mathbf{Ts}+\tilde{\mathbf{p}}) = -\varphi\varphi^T\tilde{\mathbf{p}}, \tag{2.13}$$

where, φ is the orthogonal vector to the reflecting surface. The mathematical verification or the proof of convergence in the special case of acceleration vice velocity measurements is addressed in Ref. 1.

D. COMBINED MOTION AND ENVIRONMENT MODEL

The most important part of this research is to be able to identify the environment in a dynamic fashion, by continuously updating the mathematical model (the potential function). Ideally, we may start with an initial estimate of the environment and refine it during the mission by including the data received from the sonar returns. The identification of the new objects, not present in the local map would be the outcome of the mission.

Firstly, from the known initial position of the vehicle, we build an initial estimate of the environment for our task purposes. The operational area is divided into occupancy cells, as described in Ref. 1. The number and the size of these cells is defined before the construction of the potential function model of the environment. Then the sonar data, using the initial position as a reference, is analyzed within each cell. In this approach, within each cell ξ_{ij} , the reflecting surface is modeled by an ellipsoid defined by its center

vector c_{ij} and covariance matrix P_{ij}, as it were a Gaussian distribution. These parameters are computed as

$$\begin{cases} \mathbf{c}_{ij} = \sum_{t} {}_{(ij)}\, \mathbf{p}(t) + \mathbf{T}\mathbf{s}(t), \\ \mathbf{P}_{ij} = \sum_{t} {}_{(ij)}\, (\mathbf{p}(t) + \mathbf{T}\mathbf{s}(t) - \mathbf{c}_{ij})(\mathbf{p}(t) + \mathbf{T}\mathbf{s}(t) - \mathbf{c}_{ij})^{T} \end{cases} \tag{2. 14}$$

where $\Sigma_{(ij)}$ restricts the summation to terms belonging to the cell $\xi_{(ij)}$. In the cell $\xi_{(ij)}$ where a return is detected, the potential function is constructed as

$$\mathbf{V}_{ij}(\mathbf{q}) = (\mathbf{q}-\mathbf{c}_{ij})^{T}\, \mathbf{P}_{ij}^{-1}\, (\mathbf{q}-\mathbf{c}_{ij}) \quad \text{for } \mathbf{q} \in \xi_{(ij)}. \tag{2. 15}$$

This is consistent with the definition of the potential function on the reflecting surfaces in (2.8). In the cells where no return is detected, the nearest cell containing a reflecting surface is determined.

After initially constructing the environment and defining the initial state, we move on to the task of motion estimation.

The general structure of the sensor based navigation system is shown in Figure 2.3 [Ref. 1].

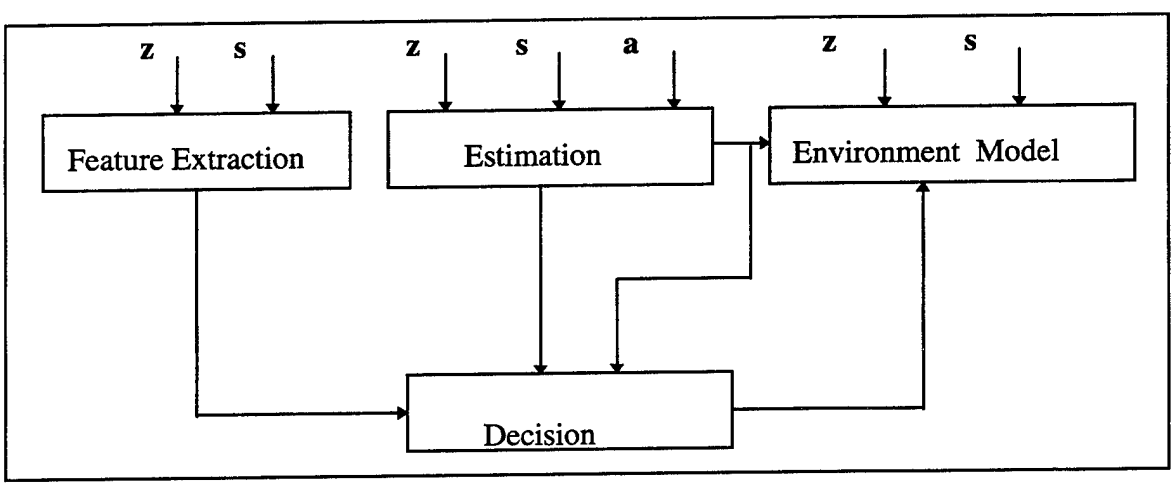

Figure 2. 3 General Structure for Combined Motion and Model of the Environment.

We consider the case of navigating the vehicle within an unknown area using its inertial navigation systems (INS), while collecting data from the scanning sonar. The model of the environment built in this way is subsequently used to correct for the errors in the INS and provide a current estimate of the vehicle's position.

The block 'feature extraction' estimates relevant features of the environment, that will be used to either validate or reject the potential function of the environment. While collecting data from the scanning sonar, we are constructing the potential function model of the environment, to be able to compare and/or update the initially estimated model. In the 'feature extraction' block, the slope of the environment is estimated by using the measured sonar data. As in (2.14), this time using the estimated vehicle position and velocity, we obtain the centroid vector and covariance matrix of the measured data for the time interval. It is considered that these reflecting cells are ellipsoids and are modeled as Gaussian distributions. Since the covariance matrix P_{ij} is Hermitian, in our case where we have a two dimensional problem, we can compute the two eigenvalues λ_{max} and λ_{min}, and the two corresponding orthonormal eigenvectors, e_{max} and e_{min}, satisfying the relation

$$P_{ij}\, e_k = \lambda_k\, e_k \;; \qquad k = 1,2 \tag{2.16}$$

Typical contours of a Gaussian density function are ellipses. If we consider one of the ellipses, it is centered at the coordinates of the mean vector, (centroid vector) and oriented with its axes parallel to the direction of the eigenvectors. The major and minor axes are proportional to the corresponding maximum and minimum eigenvalues, respectively. The slope of this ellipse is the slope of the major axis, which is the eigenvector corresponding to the maximum eigenvalue. In our case, the eigenvector corresponding to the maximum eigenvalue of each covariance matrix P_{ij} , gives the slope of the reflecting cell. This information of the measured data is going to be used in the 'decision block'.

The 'decision block' uses the slope information to validate the current model of the environment by either discarding the measurements which show inconsistencies with the

model, or/and, by updating the mathematical model itself with new segments. Ideally, when the model and the measurements are consistent with each other, then this relation holds

$$e_k P_{ij} e_m^T = \begin{cases} \lambda_k & \text{if } k = m \\ 0 & \text{if } k \neq m \end{cases} \tag{2. 17}$$

where k and m are the specified times and λ_k is the maximum eigenvalue of the covariance matrix of the initially predicted environment cell. This information is going to be used in the 'estimation block' , in order to estimate the position and the velocity of the vehicle from the data and the potential function (model of the environment).

Although the overall above system is not observable, assuming we know the initial conditions of the vehicle in terms of initial position and velocity, these can be applied to the Extended Kalman Filter (EKF) to estimate the state vector **p**.

The Kalman Filter estimates the state of a system given a set of known inputs to the system and a set of measurements. [Ref. 4]. The system is assumed to be driven by both a known input and an unknown random input, in our case

$$\mathbf{p}(k+1) = \mathbf{A}\,\mathbf{p}(k) + \mathbf{B}\,\mathbf{a}(k) + \Delta_1\,\mathbf{w}(k), \tag{2. 18}$$

where **p**(k) is the state vector (position and velocity of the vehicle), **A** is the known state transition matrix, **B** is the known input matrix, **a** is the known acceleration vector, and Δ_1 and **w**(k) are the unknown, random input matrix and vectors. We assume **w**(k) is white noise. The measurements of the system are related to the state by

$$\mathbf{q}(k) = \mathbf{H}\,\mathbf{p}(k) + \eta(k), \tag{2. 19}$$

where **q** (k) is the measurement, **H** is the measurement matrix , and η (k) is the random noise. We assume that η (k) is white noise.

The solution of the estimation problem can be shown to have the following form

$$\hat{\mathbf{p}}(\mathbf{k}+1|\mathbf{k}+1) = \hat{\mathbf{p}}(\mathbf{k}+1|\mathbf{k}) + \mathbf{K}(\mathbf{k}+1)[\mathbf{q}(\mathbf{k}+1) - \hat{\mathbf{q}}(\mathbf{k}+1)], \qquad \textbf{(2. 20)}$$

where **K** (k+1) is the Kalman Filter gain at each time index (k+1).

The form of the Extended Kalman filter equations are essentially similar to those of the Linear Kalman filter. This can find applications in nonlinear systems. The nonlinear model is used to predict the state and the nonlinear measurement is used to correct the prediction. The details and the Extended Kalman Filter Equations can be found in [Ref. 4].

In using the Extended Kalman Filter, the nonlinearities are modeled as plant noise. This means Δ_1 matrix in (2.19) is usually the identity matrix. Furthermore, because the nonlinear effects are not included in the gain matrix, the Extended Kalman Filter can be very sensitive to the **W** and η matrixes. In our case, considering η is the measurement noise of the sensors, we can include the inconsistencies of the measured slope to the model to this term in the equations. As with the measurement noise covariance matrix, if η is large, then the measurements are expected to deviate more from the states being measured, and the Kalman Filter rely more on the predicted state than on the measurements. [Ref. 2]. So, if the slope of the environment obtained from measurement data is inconsistent with the model of the vehicle's position and velocity, the estimates are based on the previous estimates and the model only.

In order to attempt to update dynamically the map of the environment, it is important to have a criterion to establish whether a sonar return comes from a mapped or unmapped object.

By standard Kalman Filtering techniques an estimate of the state **p**(t), **v**(t) and its covariance matrix can be obtained. From [Ref. 1], denoting **W**(t) the covariance matrix relative to the estimated position error, a likelihood function on the consistency of the sonar return at time *t* given all past measurements, can be defined.

In particular, assuming Gaussian errors, this would be given by the expression

$$L(t) = \ln \Pr\{s(t) \mid s(t-1), \ldots, s(0)\} =$$

$$C - \frac{1}{2} \ln \varphi(t)^T \mathbf{W}(t)\varphi(t) - \frac{1}{2} \frac{\mathbf{V}(t)^2}{\varphi(t)^T \mathbf{W}(t)\varphi(t)} \tag{2.21}$$

with Pr indicating the probability, C is a constant and the terms **V** and its gradient φ can be computed using the estimated vehicle position. A threshold can be established, by which low probability status can be assigned for the unmapped object, most likely by trial and error basis.

This model is suitable where the environment is composed of surfaces with a piecewise constant orientation. This would be the case of the vehicle operating in an area of walls or manmade structures.

E. APPLICATIONS

In this section, we address the problem of applying the combined model of the vehicle motion and the potential function approach for the environment. Although we will be using the model of the test tank at the NPS Annex, the results can be generalized to more complex environments, such as harbors, pipelines, etc. The rectangular environment with closed borders is the simplest realistic operating area which can be modeled with the potential function approach and would reflect operations such as acoustic test data-gathering with the NPS Phoenix AUV in the laboratory pool.

The program codes which are written in C for simulation purposes, are included in Appendix B. The program is using real data, that is collected by the AUV during pool testing. Although these programs are tested in simulations, they may be embedded with the original program of the vehicle, and used in real time applications.

In the first set of the simulation we are going to use the real time data collected in the test pool at NPS. For here we assume that the environment is partially known. The operational area, in our case the sizes and the shape of the pool is known, however the initial conditions (position and velocity) with respect to the fixed reference and the presence of two unmarked cylinders are not known.

The environment, we are testing is the pool at the AUV lab. at NPS. Figure 2.4 shows the contours of the potential function of the pool and the two cylinders which have been placed for testing purposes.

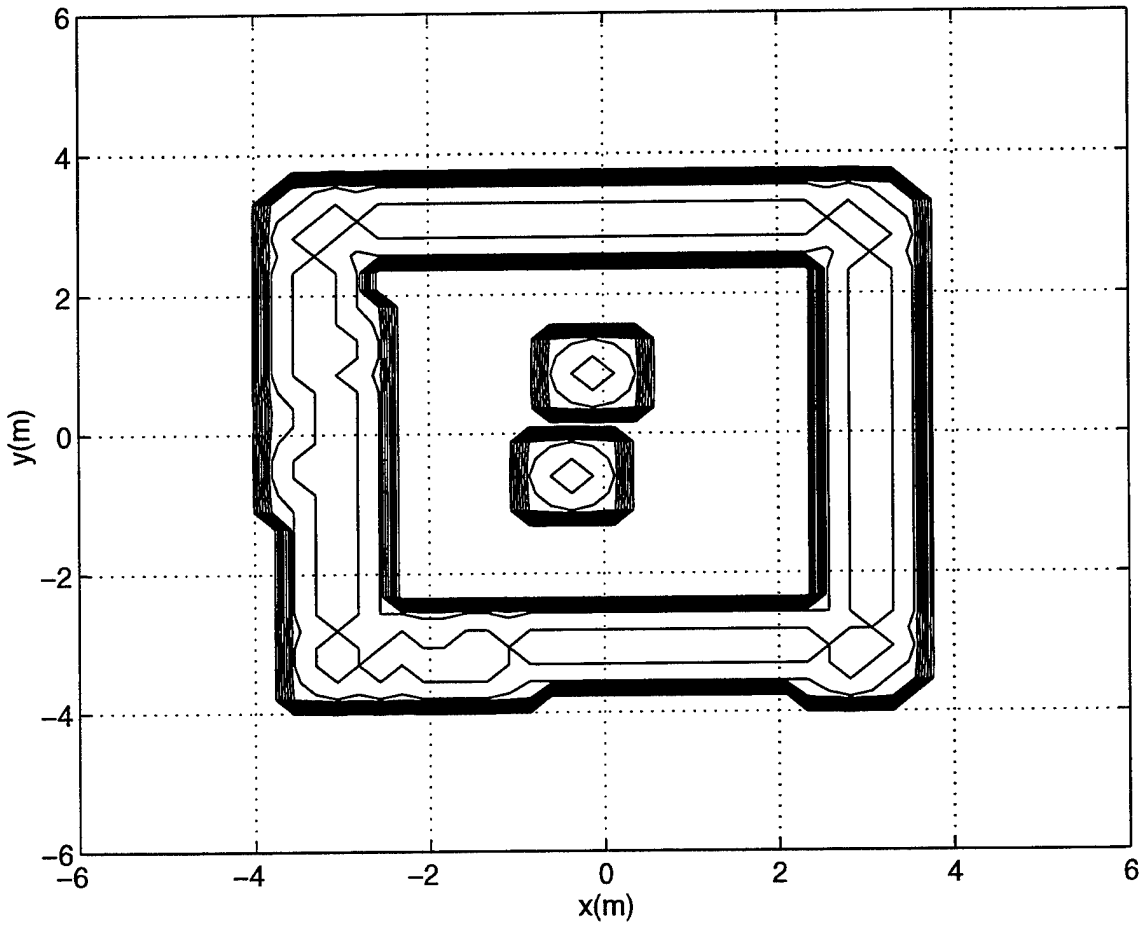

Figure 2. 4 The Contour Plot of The Pool with Two Cylinders.

A set of sonar data (range and bearing) has been collected while the vehicle is moving at a constant rate. The Cartesian plot of the data is given in Figure 2.5, which shows a collection of six consecutive scans. The scans have been taken 0.9 degrees apart. That results in 400 data points per scan.

The data in Cartesian coordinates can be seen in Figure 2.5.

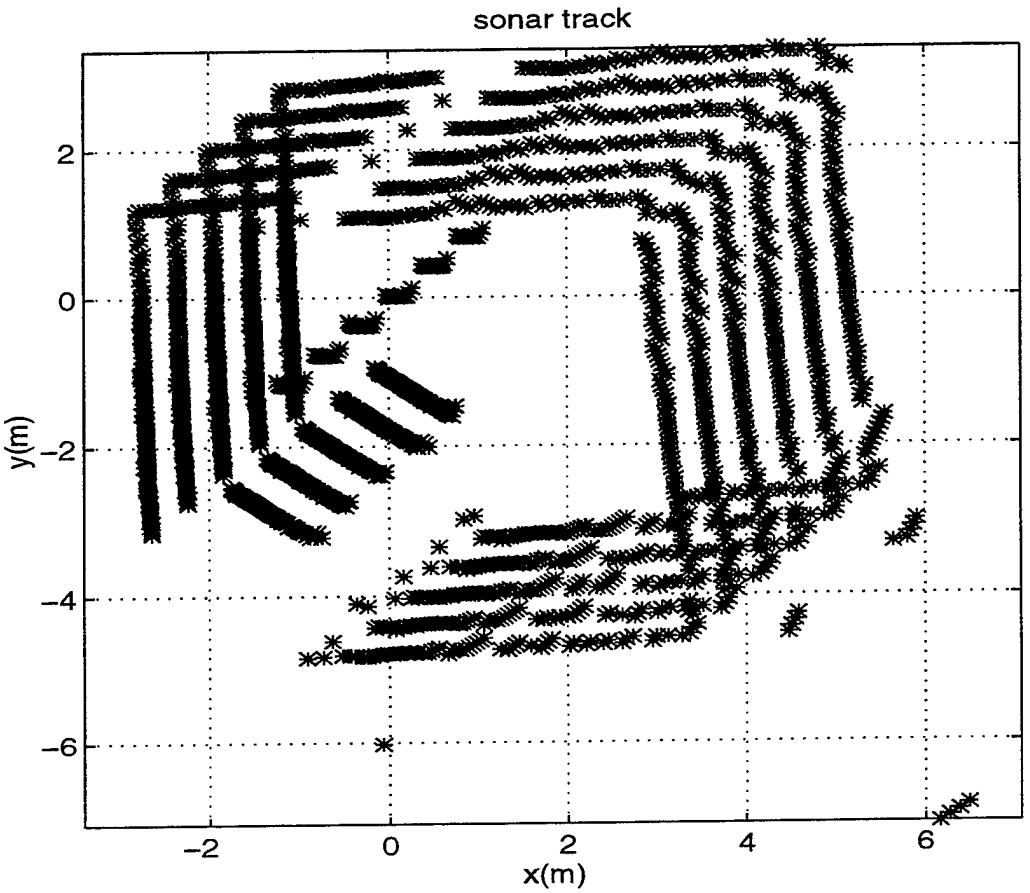

Figure 2.5 The Data in Cartesian Coordinates.

At the beginning of the run, the vehicle does not know the environment, and it has to build a map in terms of the potential function. The vehicle is moving, but the velocity of the vehicle is not known, and the initial position of the vehicle is arbitrarily considered at the middle of the pool. The orientation and the distance of each wall is computed by applying the slope algorithm (as previously discussed) to the walls of the pool. Figure 2.6 shows the contours of the estimated environment where we can see both the walls of the pool and the effect of the two obstacles. The data at the bottom right corner are due to spurious returns of sonar, and can be easily filtered. These do not belong to the environment.

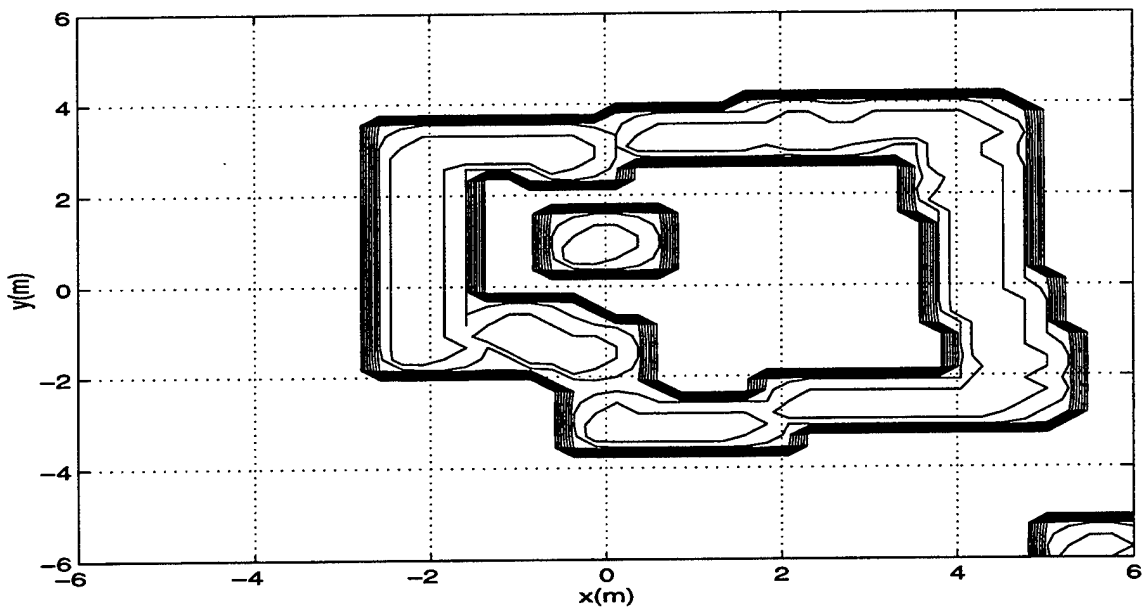

Figure 2.6 The Contours of the Estimated Environment.

In Figure 2.7, we show the estimated trajectory of the vehicle, since we do not have the exact trajectory available.

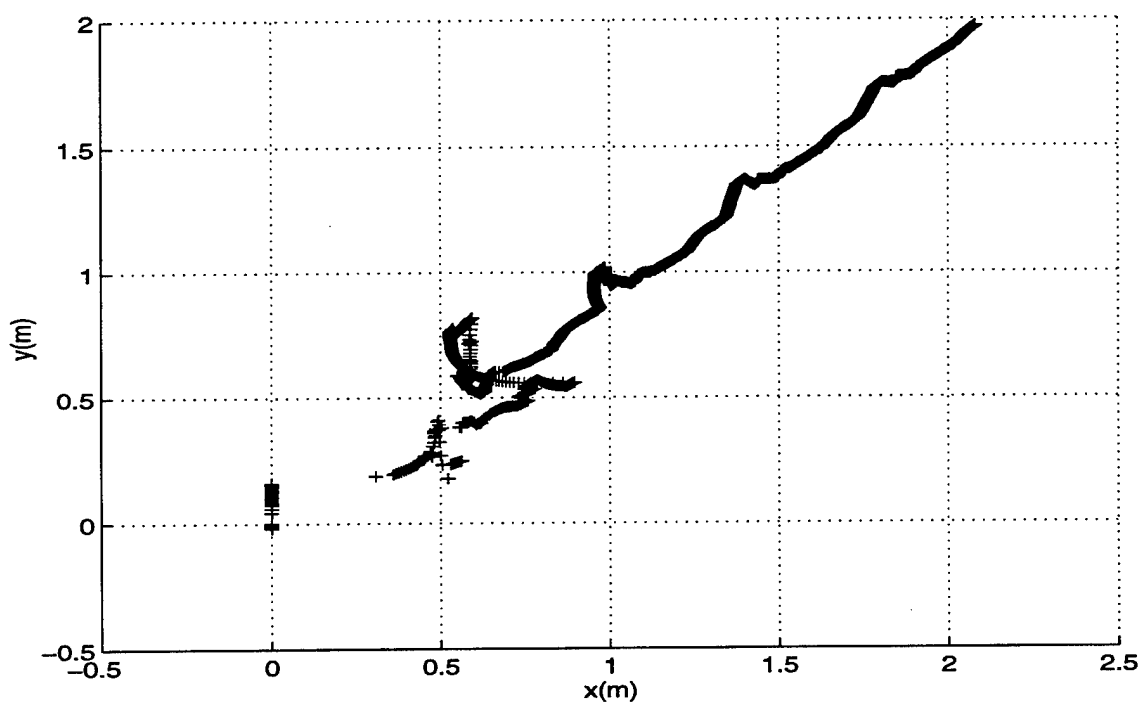

Figure 2. 7 The Trajectory Estimation of the Vehicle.

We show the validity of the estimate by comparing the estimated locations of the walls of the pool with the actual ones. We can see that the two coincide reasonably to infer that the estimated trajectory is close to the actual trajectory. This is shown in Figure 2.8.

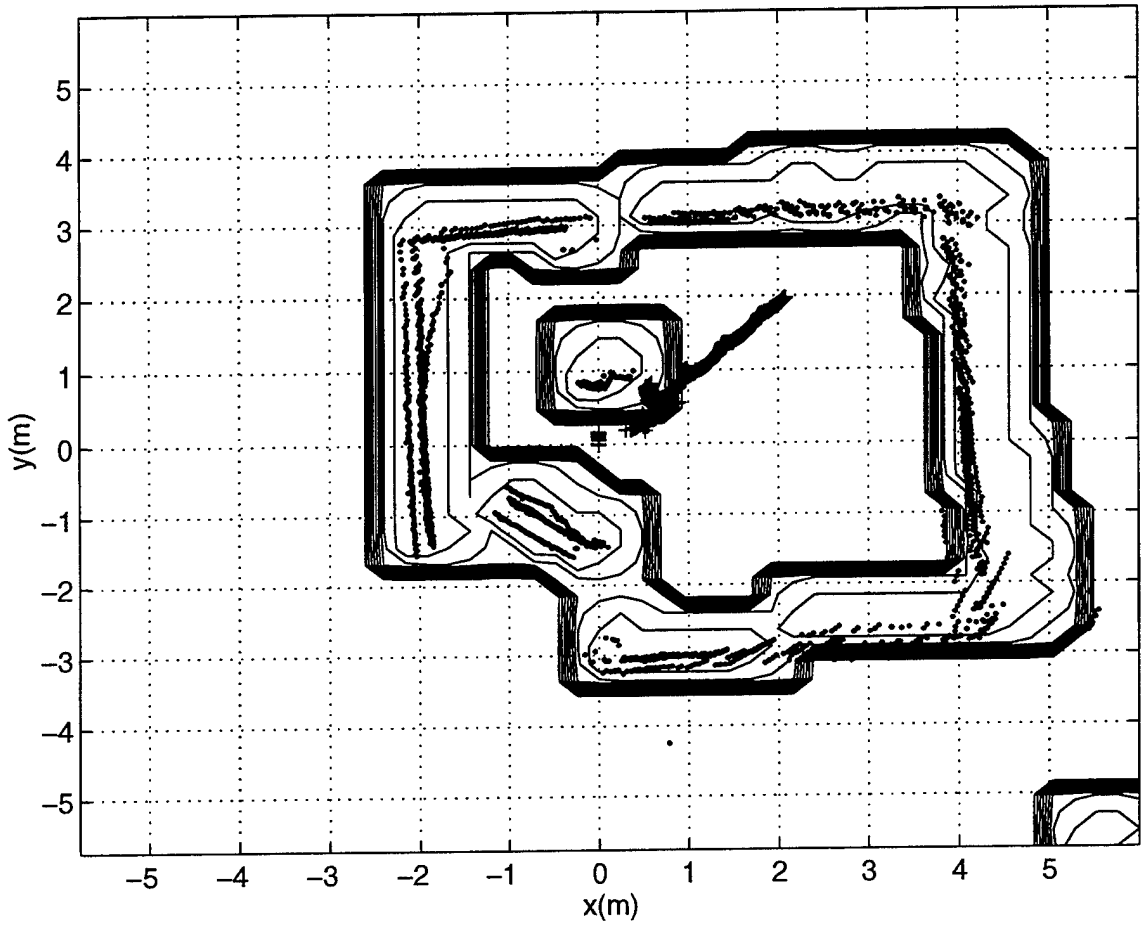

Figure 2. 8 The Estimated Environment with the Actual Environment.

In the second set of the simulations, we are going to use a sonar data set, created by using known position and velocity of the vehicle. This time we can see the consistency of the real trajectory of the vehicle with the estimated one.

The vehicle is assumed to be moving from the initial position coordinates (0,0), to the coordinates (6,6), in a same kind of environment (6 x 6 m rectangular pool), without any obstacle presence. The vehicle velocity is set to be about 0.024 m/sec.

In Figure 2.6, the motion of the vehicle in the x-direction is shown.

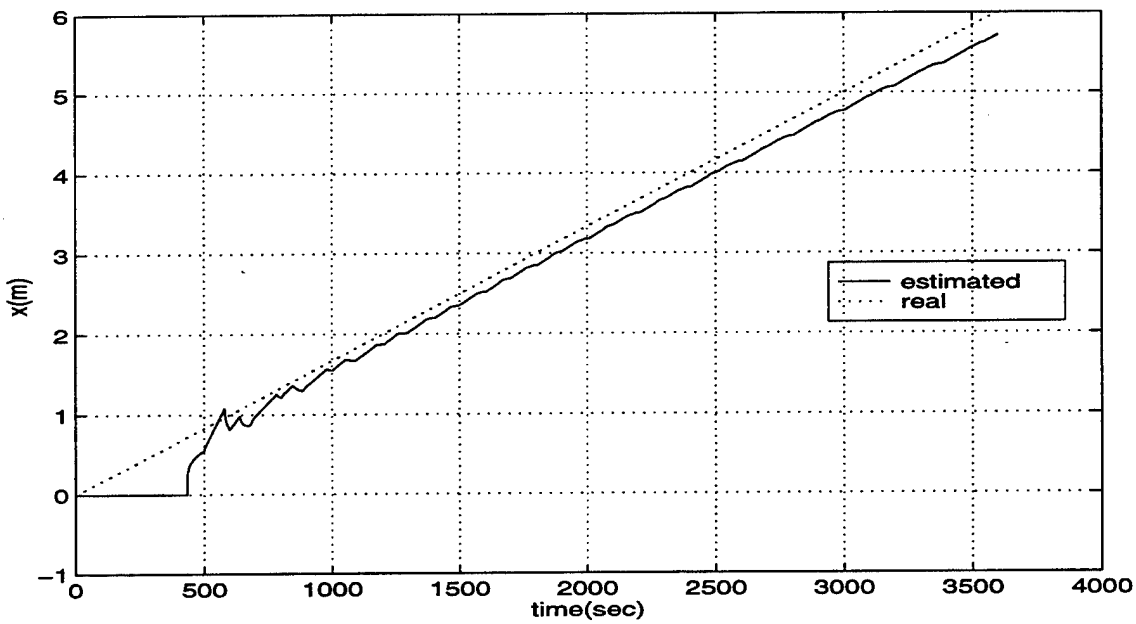

Figure 2.9 The Trajectory of the Vehicle in the X-axis.

Then, the motion of the vehicle in the y-direction is shown in Figure 2.7.

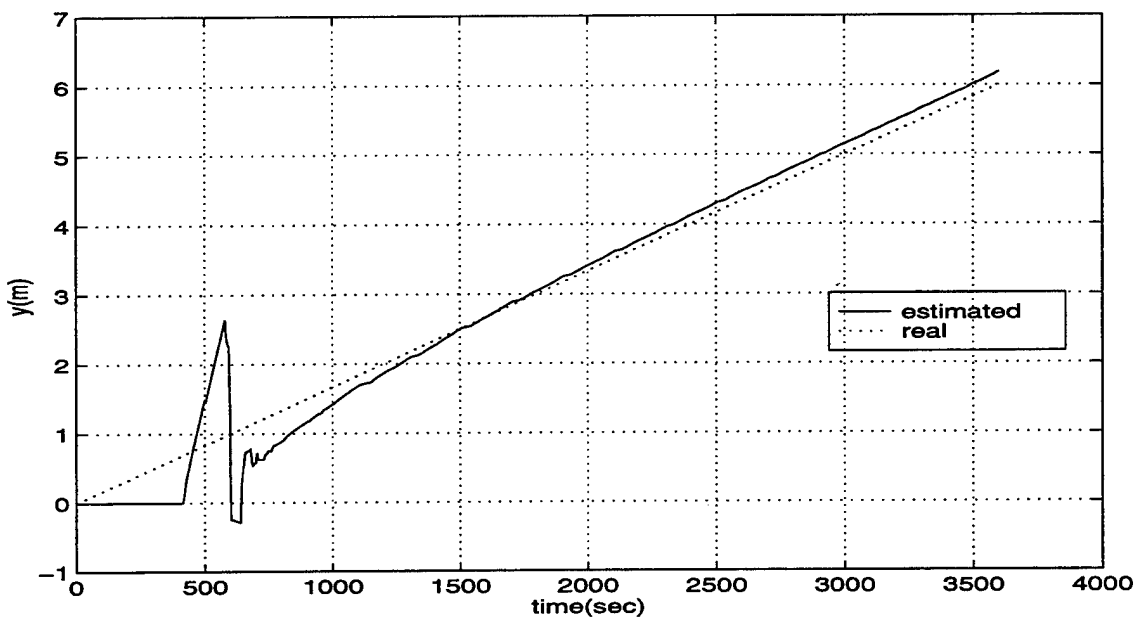

Figure 2.10 The Trajectory of the Vehicle in the Y-axis.

Finally, the whole trajectory of the vehicle and the estimated value for its velocity can be seen in Figure 2.8 and Figure 2.9, respectively

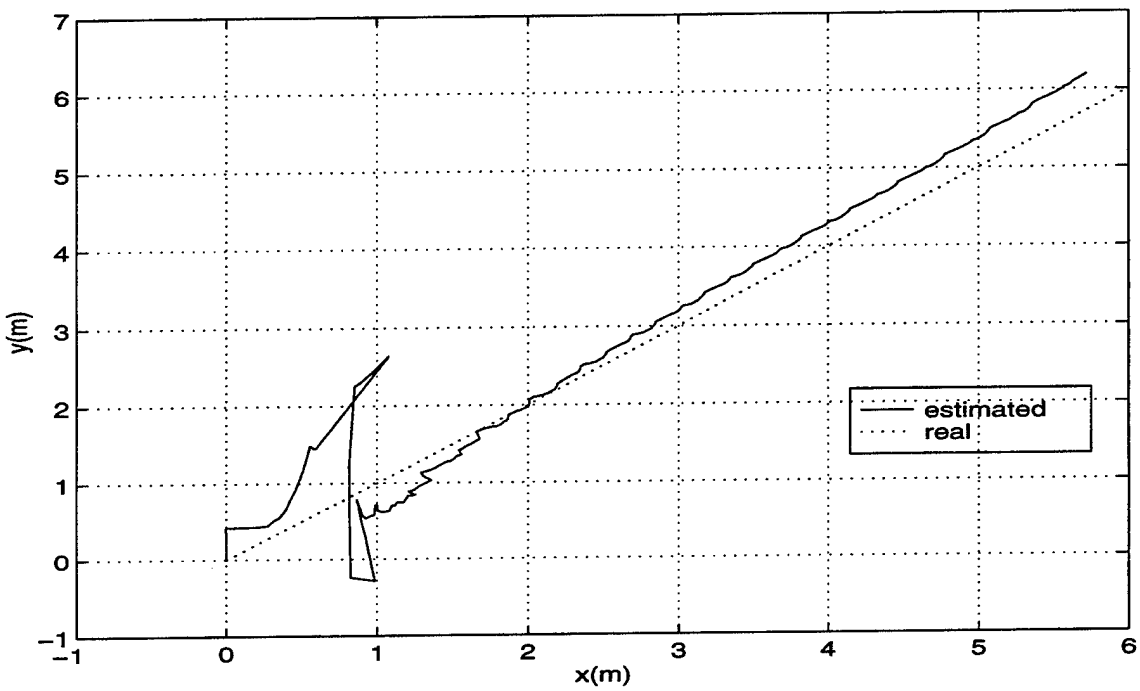

Figure 2.11 The Trajectory of the Vehicle.

III. LYAPUNOV BASED CLOSED-LOOP MOTION CONTROL FOR AN AUV

A. GENERAL

In this chapter, the problems of navigation, guidance and control of the AUV are addressed. In particular, these problems can be summarized as reaching a target frame (position and orientation) and remaining stationary on site, even in the presence of disturbances. It is intended to use the control laws based on the definition of suitable Lyapunov functions [Ref. 5], in order to allow the navigation and maneuvering tasks to be performed in a closed-loop without requiring any off-line preplanning.[Ref. 6]

B. SYSTEM MODELS

The kind of task to be performed governs the choice of the set of the coordinates which will describe the position of a frame fixed on the vehicle with respect to the target frame.

In general for an underwater vehicle, the state vector is not only composed of the linear longitudinal velocity υ and the angular velocity ω (which directly affects vehicle rotation), but also of the lateral velocity ν. The control variables refer to the orthogonal reference frame centered on the vehicle itself. The first set of kinematics equations can be directly obtained from Figure 3.1 [Ref. 7].

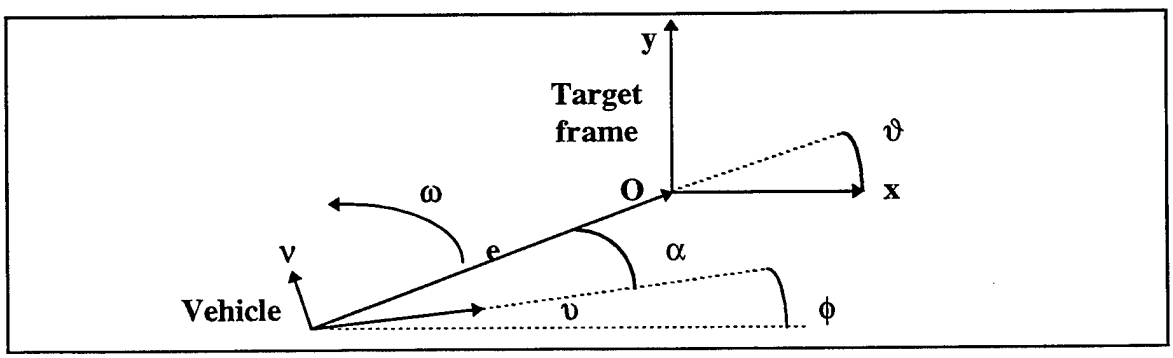

Figure 3. 1 System Coordinates.

In Figure 3.1, the vehicle's position x, y and the orientation angle ϕ, are all measured with respect to the target frame center **O**, then the first set of equations are

$$\begin{cases} \dot{x} = \upsilon \cos\phi - \nu \sin\phi \\ \dot{y} = \upsilon \sin\phi + \nu \cos\phi \\ \dot{\phi} = \omega \end{cases} \tag{3. 1}$$

where υ,ν and ω are the speeds defined above.

The second set defines the state vector as [e α ϑ], which is chosen and assumed to be completely measurable at any time. The variable **e** represents the distance error from the target frame, α denotes the angle between advanced orientation and the vector **e** while ϑ is the angle between advanced orientation and the x axis of the target frame.

From their definitions and the geometry in Figure 3.1, the following equations can be derived

$$\begin{cases} \dot{e} = -\upsilon \cos\alpha - \nu \sin\alpha \\ \dot{\vartheta} = \dfrac{\upsilon \sin\alpha}{e} - \dfrac{\nu \cos\alpha}{e} \\ \dot{\alpha} = -\omega + \dot{\vartheta} = -\omega + \dfrac{\upsilon \sin\alpha}{e} - \dfrac{\nu \cos\alpha}{e} \end{cases} \tag{3. 2}$$

It should be noted that this set of equations is only valid for non-zero value of the distance errors **e**, otherwise, both angles α and ϑ simply would be undefined quantities.

Both models (3.1) and (3.2) are kinematics, so the linear (υ,ν) and angular (ω) velocities need to be imposed. To these ends, it is necessary to use a dynamic system [Ref. 7]. Let us assume that all the masses and inertial moments are equal to the unitary value. New variables describing the forces and moments are introduced.

In this case, the dynamics of the vehicle becomes

$$\begin{cases} \dot{\upsilon} = f + v\omega \\ \dot{v} = g + \upsilon\omega \\ \dot{\omega} = M \end{cases} \tag{3. 3}$$

where f and g are the forces of the thrusts of the frontal and lateral screw propeller and M is the angular moment.

C. CONTROL ARCHITECTURE

The control loop is designed based on the approach described in Ref. 6. This is set as two layered hierarchical control architecture seen in Figure 3.2.

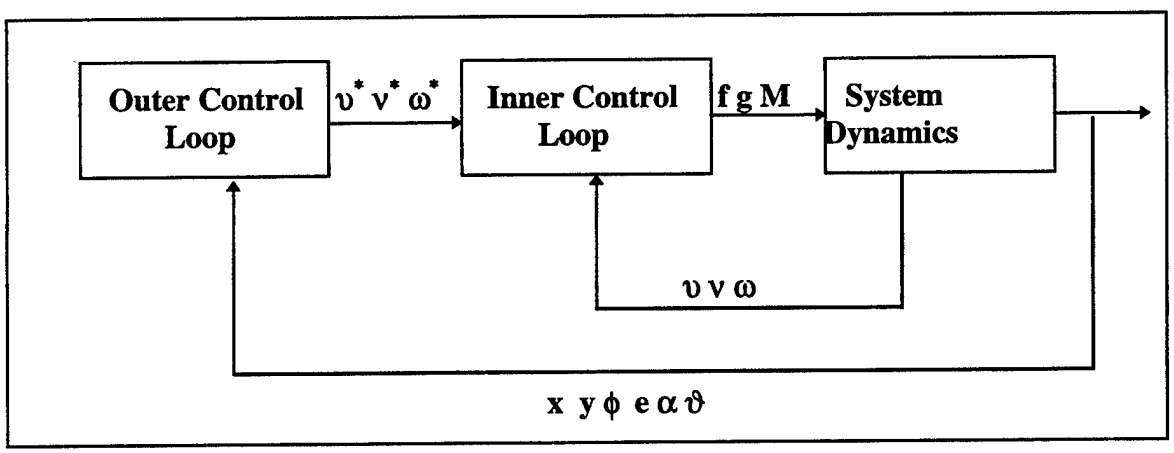

Figure 3. 2 Two Layered Hierarchical Control Loop.

Defining the Lyapunov function V= V(z(t)), with z is the state [Ref. 6], the control law is designed by imposing $\dot{V} \leq 0$. The particular operational variables of motion are designated for each task, and the outer loop eliminates any operational error by computing the reference velocities. These will be a feedback for the inner control loop on the next turn.

The inner control loop is task independent and implements a velocity servo loop, based on the vehicle dynamics. However, the outer control loop lets the reference trajectory based on the task to be followed.

D. OUTER CONTROL LOOP

By the outer control loop, we set the desired motion of the vehicle in order to perform the given task. As mentioned above, we define different vehicle trajectories to satisfy different requirements during the approach to the target. Three main tasks are defined as the maneuvers of the vehicle [Ref. 6].

1. Long Range Navigation

Regardless of its orientation, this task is performed by the vehicle, when it is sufficiently distant from the target area. The heading of the vehicle, yaw rate and speed can be measured by a gyrocompass and a Doppler sonar.

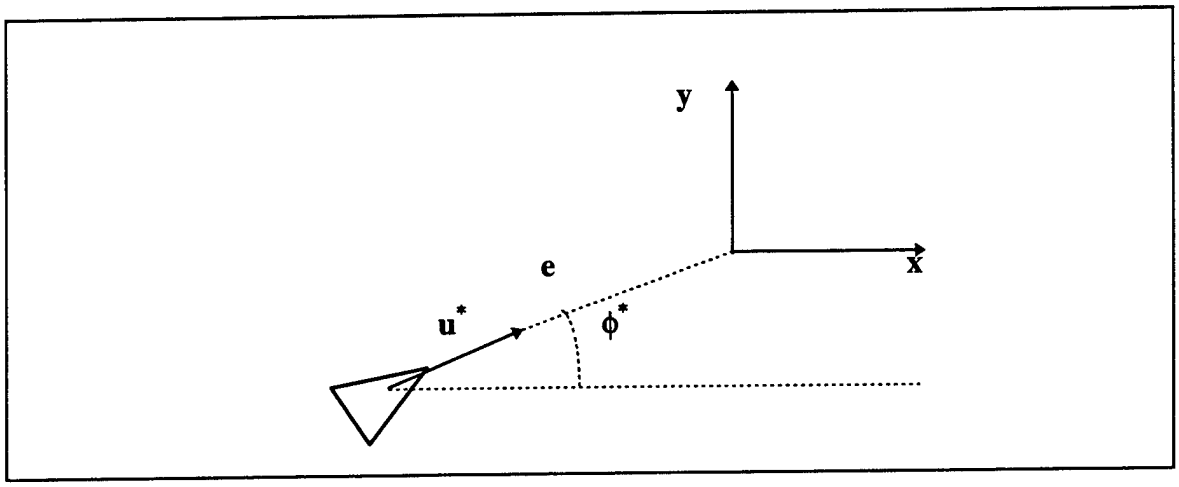

Figure 3. 3 Long Range Navigation.

The Lyapunov function associated to the task of approaching the target and its first time derivative can be expressed as [Ref. 6]

$$\begin{cases} V(x,y) = 1/2\, e^2 = 1/2\,(x^2 + y^2), \\ \dot{V}(x,y) = x\,\dot{x} + y\,\dot{y}. \end{cases} \tag{3. 4}$$

Now, the problem is to find a control which makes its derivative negative semidefinite.

Plugging the values for $\dot{x}$ and $\dot{y}$ from (3.1), and collecting the terms, we obtain the expression below

$$\dot{V}(x,y) = \upsilon\,(x\cos\phi + y\sin\phi) + v\,(y\cos\phi - x\sin\phi) \tag{3.5}$$

From here we are going to define the reference values for the outer control loop variables, linear and angular velocities, and the heading angle, as $\upsilon^*, v^*, \omega^*$, and ϕ^*. The reference values for v and ω, are set to be $v^* = \omega^* = 0$ [Ref. 6]. Also, by the fact that the vehicle is heading to the target, ϕ^* would be

$$\phi^* = \tan^{-1}(-y,-x) \tag{3.6}$$

Then, using the reference variables, (3.5) can be expressed as

$$\dot{V}(x,y) = \upsilon^*\,(x\cos\phi^* + y\sin\phi^*), \tag{3.7}$$

setting υ^* as

$$\upsilon^* = -\,k_L\,(\,x\cos\phi^* + y\sin\phi^*\,) \tag{3.8}$$

where υ^*, describes the reference value for the linear velocity, υ, and ϕ^* is the desired heading, makes the first time derivative of the Lyapunov function negative definite.

2. Medium Range Navigation

At this phase, the vehicle is sufficiently close to the target, and it begins to approach with the required orientation. For the simple case, it is assumed that the desired orientation is $\phi = 0$. The state vector [$e\ \alpha\ \vartheta$], which is defined in Section B, will be used to describe the task function. The state variables are shown in Figure 3.1. For this operation the following Lyapunov function and its derivative can be defined as [Ref. 6]

$$\begin{cases} V(e,\alpha,\vartheta) = 1/2\,\lambda e^2 = 1/2\,(\alpha^2 + h\vartheta^2), \\ \dot{V}(e,\alpha,\vartheta) = \alpha\dot{\alpha} + h\vartheta\dot{\vartheta}. \end{cases} \tag{3.9}$$

In general the lateral velocity ν can be either neglected, due to the low term speed, or compensated by the lateral thrusters. In this study we assume the lateral velocity to be zero, i.e. $\nu^* = 0$. Plugging the values for $\dot{\alpha}$ and $\dot{\vartheta}$ from (3.2), and collecting the terms, we obtain the derivative expression as

$$\dot{V}(e,\alpha,\vartheta) = -\alpha\omega + \upsilon\frac{\sin\alpha}{e}(\alpha + h\vartheta) - \nu\frac{\cos\alpha}{e}(\alpha - h\vartheta) \qquad \textbf{(3. 10)}$$

Setting, the reference values for linear and angular velocities, υ^* and ω^* as in the following, and $\nu^*=0$, we guarantee that the derivative of the Lyapunov function is negative definite.

$$\begin{cases} \upsilon^* = k_M e\cos\alpha \\ \nu^* = 0 \\ \omega^* = \eta\alpha + k_M\dfrac{\cos\alpha\sin\alpha}{\alpha}(\alpha + h\vartheta) \end{cases} \qquad \textbf{(3. 11)}$$

In these equations η, k_M are positive constants.

3. Short Range Navigation / Hovering

At this phase, when the vehicle is very close to the target, it hovers above it. The vehicle must be fully controllable, so it must be equipped with traverse thrusters as well as longitudinal ones. When the area is structured, the position and the speed can be estimated by a combination of sonar, visual techniques and its INS. The fine maneuvering Lyapunov function [Ref. 6] and its first time derivative can be defined as

$$\begin{cases} V(x,y,\phi) = 1/2\,\lambda(x^2 + y^2) + 1/2\,h\,\phi^2, \\ \dot{V}(x,y,\phi) = \lambda(x\dot{x} + y\dot{y}) + h\phi\dot{\phi}. \end{cases} \qquad \textbf{(3. 12)}$$

Plugging the values for $\dot{x}$, $\dot{y}$ and $\dot{\phi}$ from (3.1), and collecting the terms, we obtain the derivative expression as

$$\dot{V}(x,y,\phi) = \lambda\, \upsilon\, (x \cos\phi + y \sin\phi) + \lambda\, \nu\, (-x \sin\phi + y \cos\phi) + h\, \phi\, \omega \qquad \textbf{(3. 13)}$$

Setting, the reference values for linear and angular velocities, υ^*,ν^* and ω^* as in the following, we guarantee that the derivative of the Lyapunov function is negative definite [Ref. 6].

$$\begin{cases} \upsilon^* = -k_u (x \cos\phi + y \sin\phi), & k_u > 0 \\ \nu^* = -k_v (-x \sin\phi + y \cos\phi), & k_v > 0 \\ \omega^* = -\eta\phi, & \eta > 0 \text{ and } k_u = k_v \end{cases} \qquad \textbf{(3. 14)}$$

where η, k_u and k_v are positive constants.

E. INNER CONTROL LOOP

As described above, the inner control loop structure in Figure 3.3 depends on the system dynamics and is independent of the particular task performed. Thus, the dynamic equations, (3.3), together with (3.1) and (3.2), globally yields a second order system, with the control variables **f**, **g**, **M** as inputs, and either the state vector $[x\ y\ \phi\ \upsilon\ \nu\ \omega]'$ for the long range and short range navigation phase or the state vector $[e\ \alpha\ \vartheta\ \upsilon\ \nu\ \omega]'$ for the medium range navigation phase as the outputs, as seen in Figure 3.2. The task of the internal loop is to follow a desired vector $w^*=[\upsilon^*\ \nu^*\ \omega^*]'$. Finally, a Lyapunov function [Ref. 6] composed of the velocity terms is defined as

$$V_d = \tfrac{1}{2} \lambda_u (\upsilon - \upsilon^*)^2 + \tfrac{1}{2} \lambda_v (\nu - \nu^*)^2 + \tfrac{1}{2} \lambda_\omega (\omega - \omega^*)^2 \qquad \textbf{(3. 15)}$$

After differentiating and substituting the expressions for the angular and linear velocities in (3.3),

$$\begin{aligned}\dot{V}_d &= \lambda_u(\upsilon-\upsilon^*)(\dot{\upsilon}-\dot{\upsilon}^*)+\lambda_v(v-v^*)(\dot{v}-\dot{v}^*)+\lambda_w(\omega-\omega^*)(\dot{\omega}-\dot{\omega}^*) \\ &= \lambda_u(\upsilon-\upsilon^*)(f+v\omega-\dot{\upsilon}^*)+\lambda_v(v-v^*)(g-\upsilon\omega-\dot{v}^*)+\lambda_w(\omega-\omega^*)(M-\dot{\omega}^*)\end{aligned} \tag{3.16}$$

the control variables are chosen to force the derivative of the task function to be negative definite [Ref. 7]. Then, the control variables can be defined as follows

$$\begin{cases} f = -v\omega+\dot{\upsilon}^* - p_f(\upsilon-\upsilon^*) \\ g = \upsilon\omega+\dot{v}^* - p_g(v-v^*) \\ M = \dot{\omega}^* - p_M(\omega-\omega^*) \end{cases} \tag{3.17}$$

This choice implies that the error on velocity tends toward zero ($\Delta\omega \rightarrow 0$).

The coefficients p_f , p_g, p_M are the gains of the proportional part and are free parameters that can be tuned to achieve optimum performance.

F. SIMULATION RESULTS

All simulations were based on the hydrodynamic model explained in previous sections. The scenario is designed by a MATLAB SIMULINK model shown in Figure 3.4.

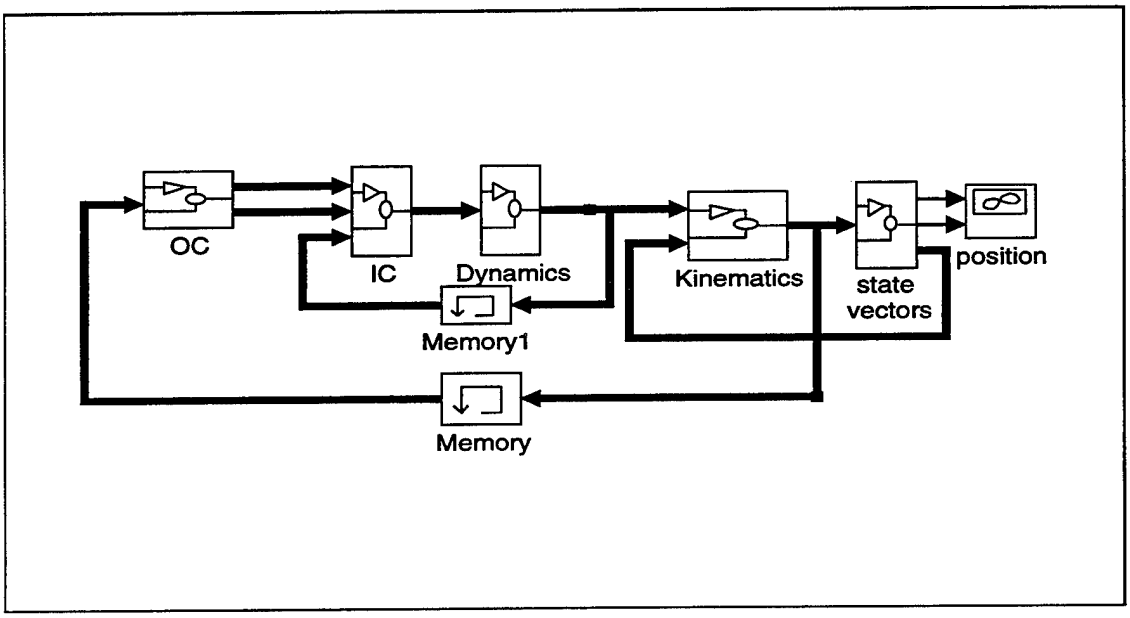

Figure 3. 4 SIMULINK Model of the Control Algorithm.

The corresponding MATLAB files used in above model are presented in Appendix C.

In this simulation the target is positioned in (10,10) with the final required orientation of 0 degrees. The initial position of the vehicle is assumed to be in (1,1). All of the constants and gains in the equations in previous sections are determined by trial and error methods, in order to be able to get a sufficient and desired results.

As shown in Figure 3.5 below, when only the Long Range Navigation task is scheduled, the vehicle moves in a straight line, towards the target and reaches it without controlling its orientation.

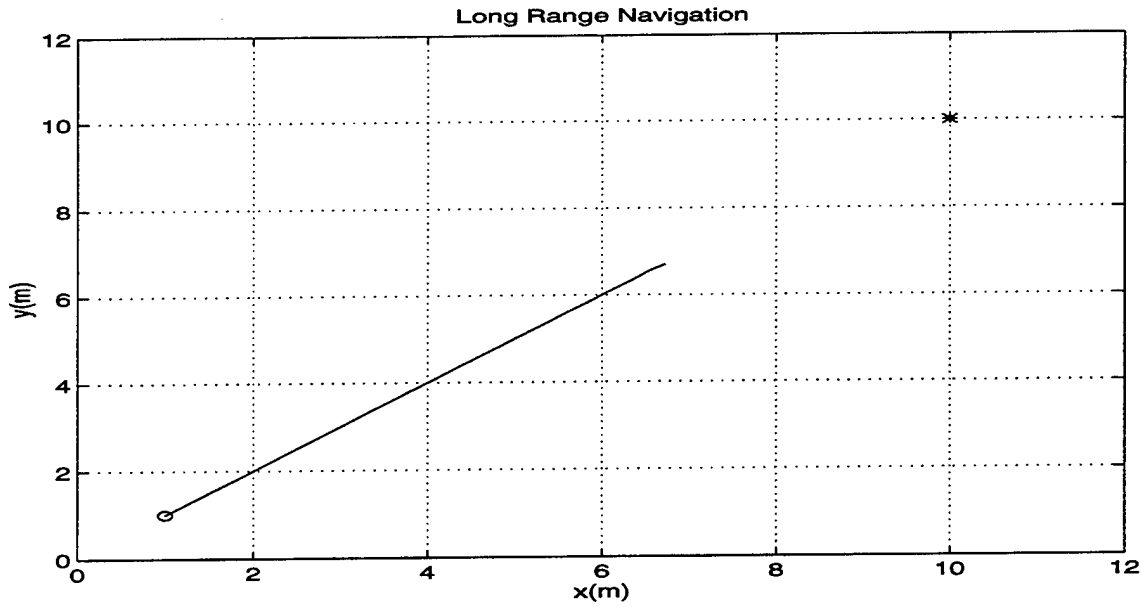

Figure 3. 5 Long Range Navigation Trajectory.

When the vehicle is 5m away from the target, the Medium Range Navigation Phase starts, and a constraint on the vehicle orientation is introduced in the Lyapunov function. The trajectory of the vehicle including this phase can be seen in Figure 3.6.

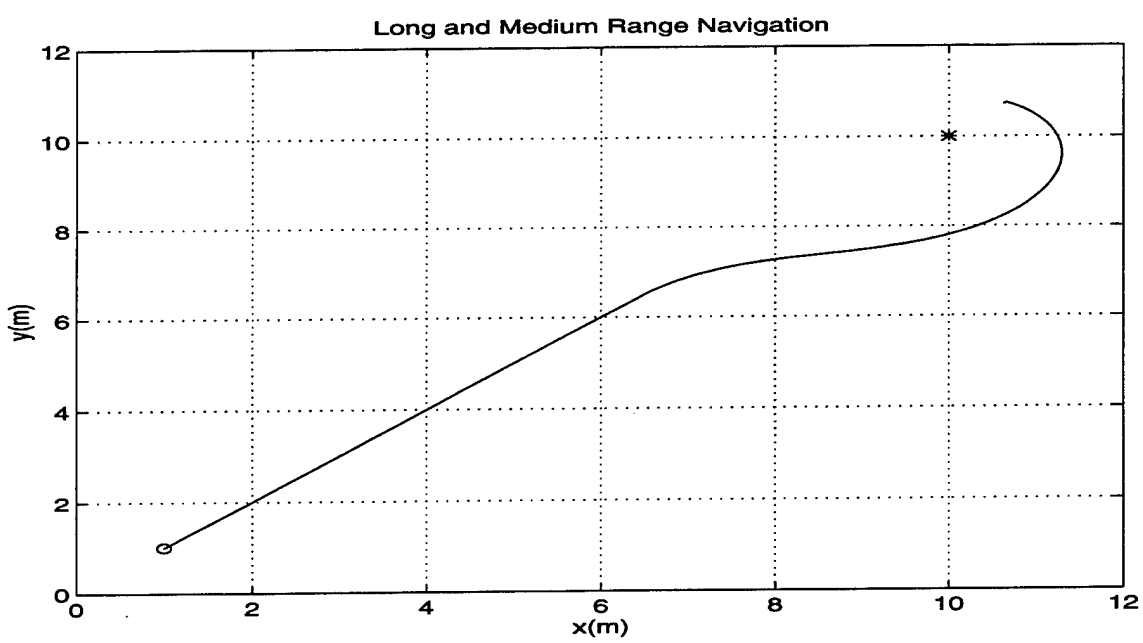

Figure 3.6 Long and Medium Range Navigation Trajectory.

In Figure 3.6 we see the vehicle taking a bend to approach the target. When the vehicle is close to the target, since **e** goes to zero α and ϑ change very quickly for small movements of the vehicle.

In our simulation, the vehicle begins the Short Range Navigation/Fine Maneuvering Phase, when it reaches the distance of 1 m. from the target. The resulting trajectory is shown in Figure 3.7.

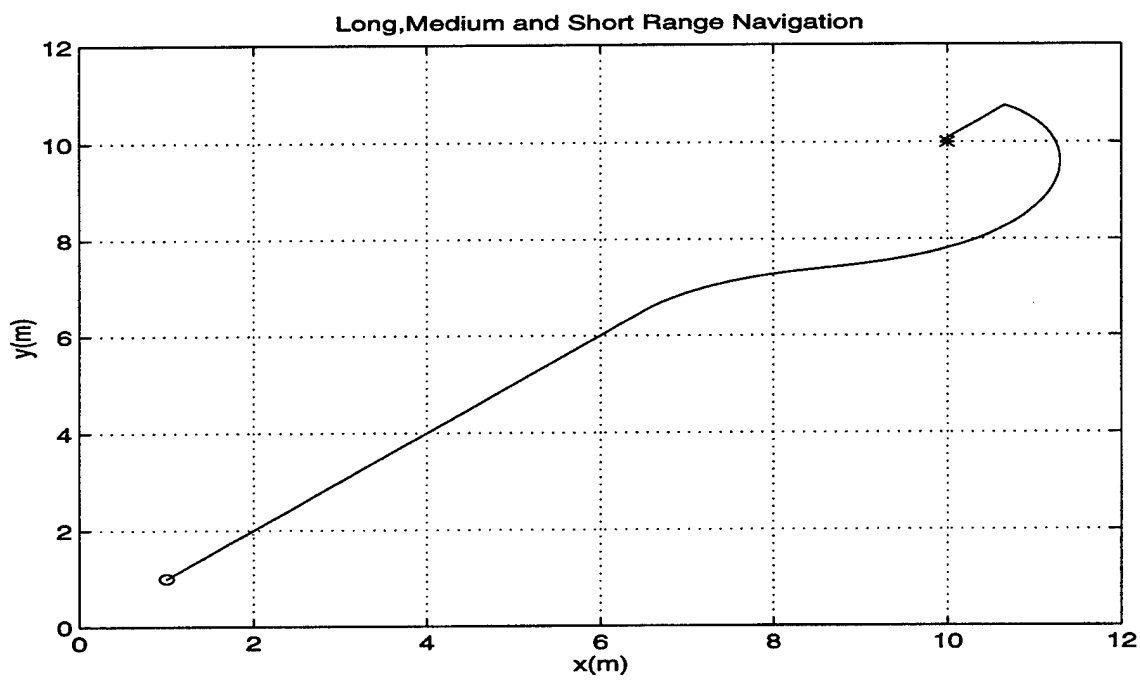

Figure 3. 7 Long, Medium and Short Range Navigation Trajectory.

The Long Range Navigation phase makes the vehicle head straight for the target. Its distance **e** from the target decreases and the angle α goes to zero. These are seen in Figure 3.8.

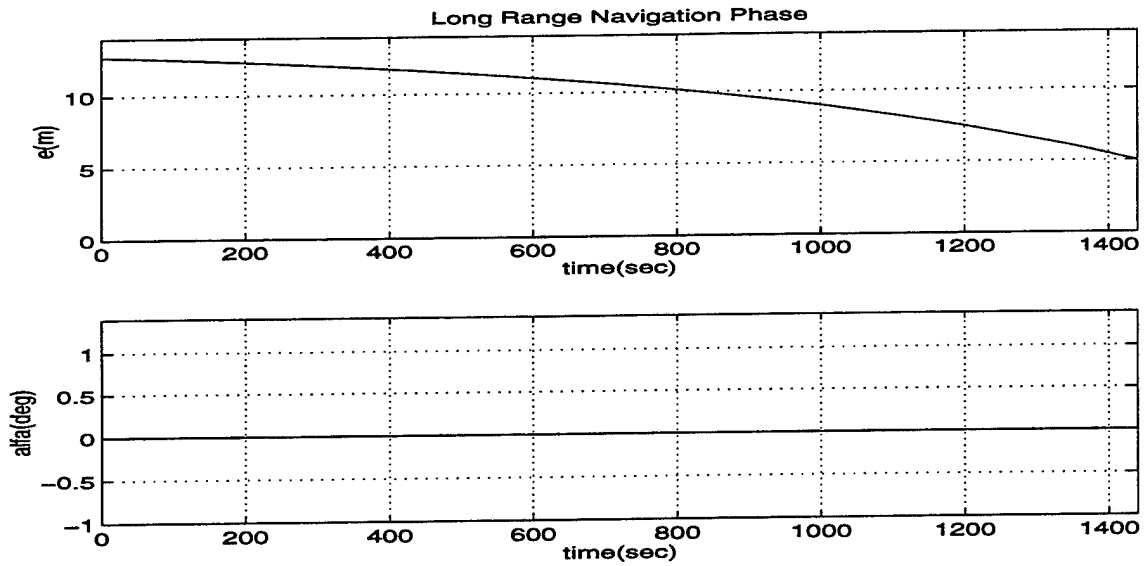

Figure 3. 8 Long Range Navigation Operational Variables (e (t) and α (t)).

When executing the Medium Range Navigation phase, the vehicle approaches the target and lines up with the desired orientation, so minimizing the operational variables **e**, (α+ϑ). These variables can be seen in Figure 3.9.

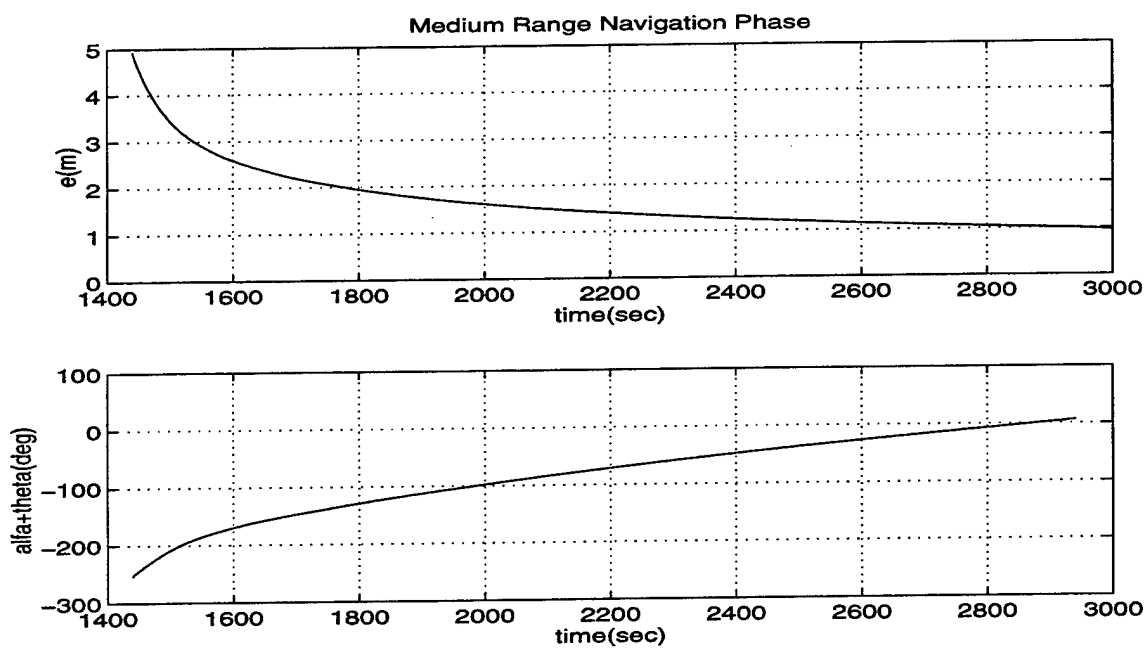

Figure 3. 9 Medium Range Navigation Operational Variables (e(t) and α(t)+ϑ(t)).

In the fine maneuvering phase the distance e from the target goes to zero, and the orientation angle ϕ will do small oscillations but finally goes to the desired orientation.

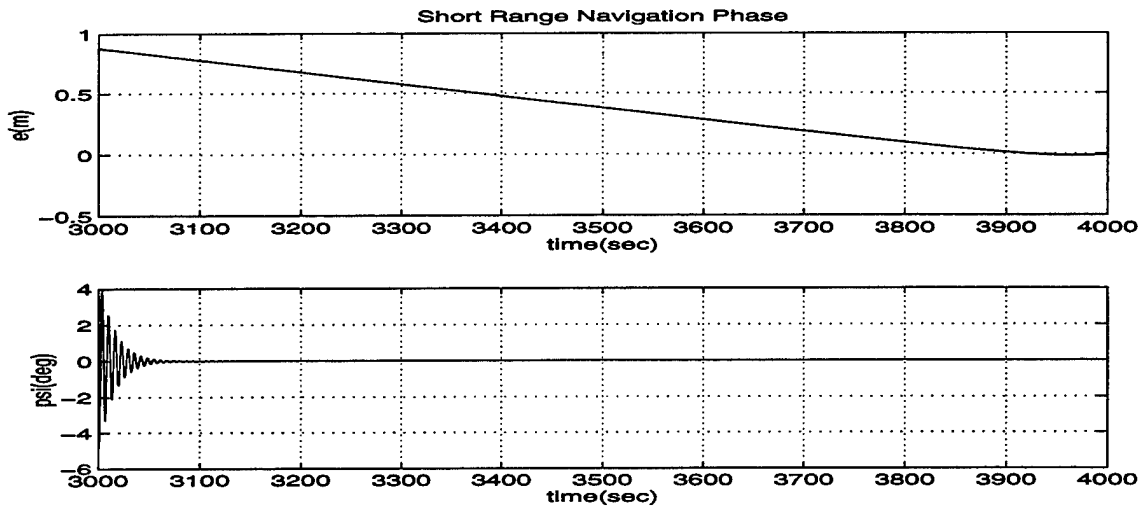

Figure 3. 10 Fine Maneuvering Operational variables (e(t) and ϕ(t)).

IV. SONAR BASED MOTION ESTIMATION AND CONTROL FOR AN AUV

A. GENERAL

In this chapter, a combined model of navigation, guidance and control in an AUV is addressed. In particular, a Lyapunov based control system for guidance and an acoustic motion estimation module are integrated in a two layered hierarchical architecture for closed-loop control [Ref. 8]. The guidance system is designed, as mentioned in Chapter III, on the basis of the definition of suitable Lyapunov functions for the different maneuvers in approaching a target. The motion estimation algorithm, as anticipated in Chapter II, is based on scanning sonar returns and uses the sonar range and bearing information in an Extended Kalman Filter-based structure. The overall estimation and control model is simulated in MATLAB SIMULINK model and the results are presented.

B. THE ACOUSTIC MOTION ESTIMATION AND CONTROL MODEL

As previously discussed, we are going to use a nested-loop Lyapunov based guidance algorithm of the integrated acoustic motion estimation model. A two layered hierarchical architecture for closed-loop control which does not require any planning has been designed through the definitions in Chapter III. This approach is suitable for handling the different target approach maneuvers, which require different precision in motion control, and so it introduces different phases of approaching the target based on the distance to it. On the other hand, this method includes an independent inner control loop for different tasks.

For the motion estimation module in the integrated model, we are going to use the low speed AUV motion estimation model as previously discussed in Chapter II. This method allows high precision motion estimation in the direction orthogonal to a tracked linear reflecting surface for the case of structured environments. The vehicle's planar motion is estimated through the range and bearing measurements of a scanning sonar, using Kalman Filtering techniques.

Finally, a Kalman Filter based acoustic navigation module and a two nested loop guidance and control module have been combined to control the motion of an underwater vehicle. Chapter III dealt with the control algorithms based on Lyapunov techniques and the phases in the approaching a target for the vehicle. This will be our guidance and control module. On the other hand, the dynamic model of the Extended Kalman Filter based motion estimation technique from the scanning sonar measurements were discussed in Chapter II. This will be our motion estimation module for the combined model.

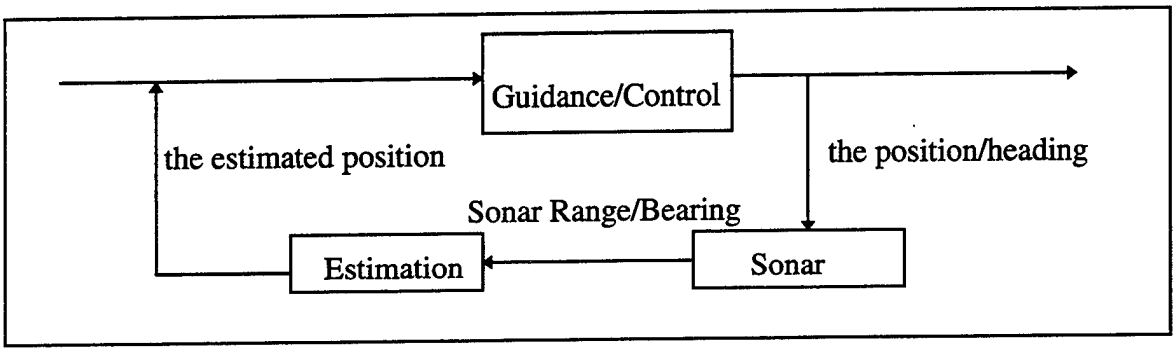

Figure 4. 1 The Acoustic Motion Estimation and Control Model.

In Figure 4.1, the 'Guidance/Control' module represents the Lyapunov based control model, previously designed in Figure 3.2, and the 'Estimation' module represents the potential function based environment and motion estimation model, previously designed in Figure 2.3.

C. SIMULATIONS AND RESULTS

In this section we present the application of combined model of the vehicle motion and the potential function approach for the environment with two layered Lyapunov based control/guidance system. As an operating area we will assume a rectangular closed environment described by a suitable potential function. We are going to simulate the combined model with a MATLAB SIMULINK model shown in Figure 4.2. The corresponding MATLAB files can be found in Appendix C.

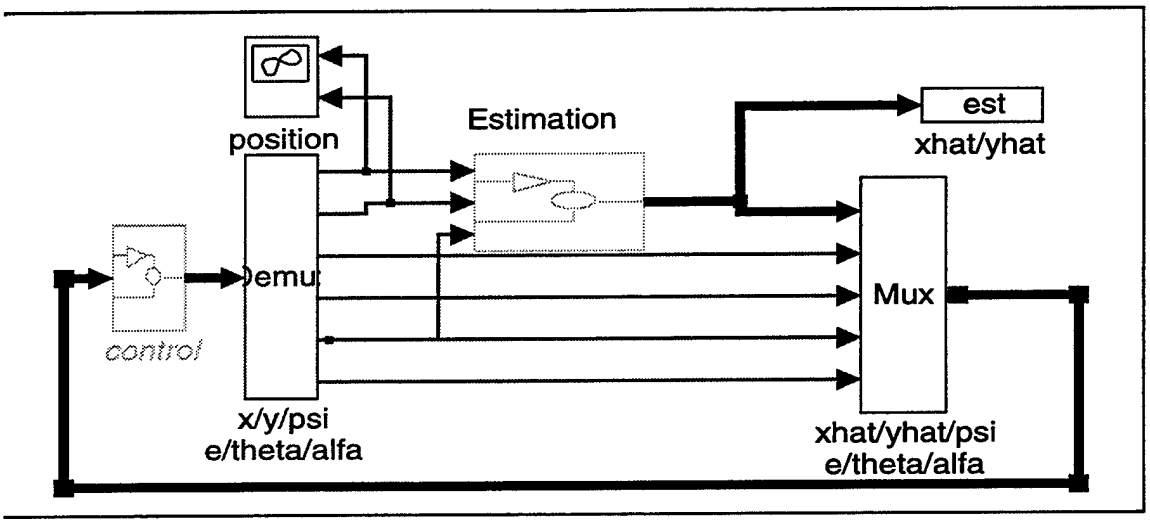

Figure 4.2 The SIMULINK Model of the Motion Estimation and Control.

Each simulation is organized into two phases: initialization and navigation between the initial point and the target. At the beginning, the vehicle builds a local map of the operating environment, which is a rectangular shaped pool with perpendicular walls. To do this, the vehicle stands virtually still (constant depth and orientation) and it scans the walls with its sonar. The orientation and the distance of each wall is computed using the 'slope estimation algorithm', previously discussed in Chapter II. The Three dimensional plot of the potential function is shown in Figure 4.3.

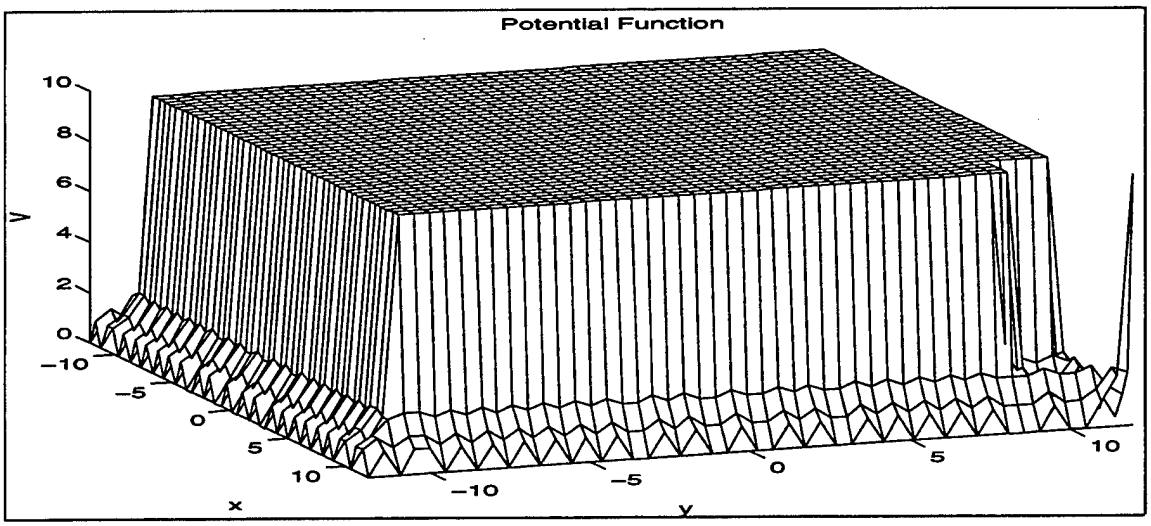

Figure 4.3 Potential Function of the Rectangular Pool Environment (24x24 m).

Upon completion of the initialization phase, the potential function model of the environment is constructed. Then, the vehicle moves between its initial position and the position of the target, estimating its motion on the basis of the algorithm presented in Chapter II, while updating the potential function model of the environment. The estimated positions are processed in the Control/Guidance module for each phase of the approach to the target. The Control/Guidance module computes the desired trajectory to be followed by the vehicle. A good performance of the algorithm relies on a consistent estimate of the vehicle's location by the estimator.

For our case the vehicle is moving in a 24x24 m pool, starting at the origin (the (0,0) position) to a final target located at position (10,10). The desired orientation of the vehicle at the target is zero degrees in the target fixed frame. The contour plot of the potential function is shown in Figure 4.4, where we can see the location of the target at position (10,10).

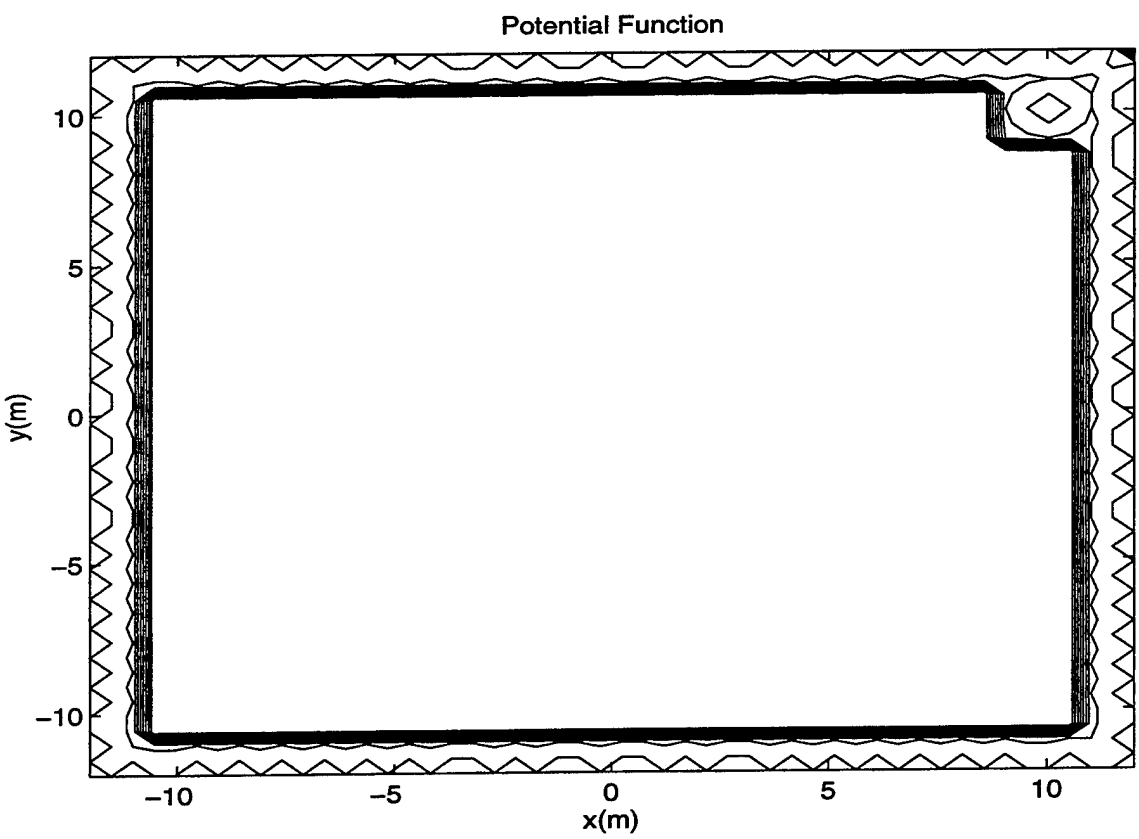

Figure 4.4 The Contour Plot of the Potential Function Model of the Pool (24x24m).

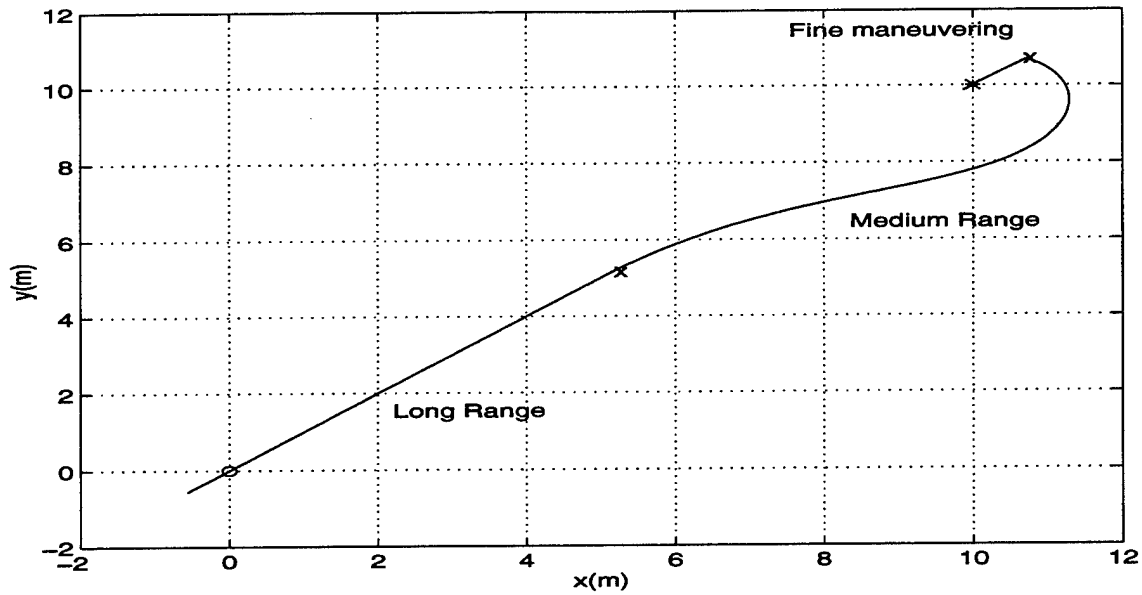

Figure 4.5 Vehicle's Trajectory from Its Initial Position to the Target.

The trajectory of the vehicle including all of the phases of approaching to the target, is shown in Figure 4.5. Firstly, the vehicle adjusts its heading to the target, then approaches it a distance of 7 m. on a straight line. When it is about 7 m. from the target, it maneuvers in order to be able to reach the target at the desired orientation. Finally, at the fine maneuvering phase, about 1 m. apart from the target, it moves to the target keeping its orientation. At the target, the resulting orientation is ϕ=0 degrees and the vehicle is hovering above the target.

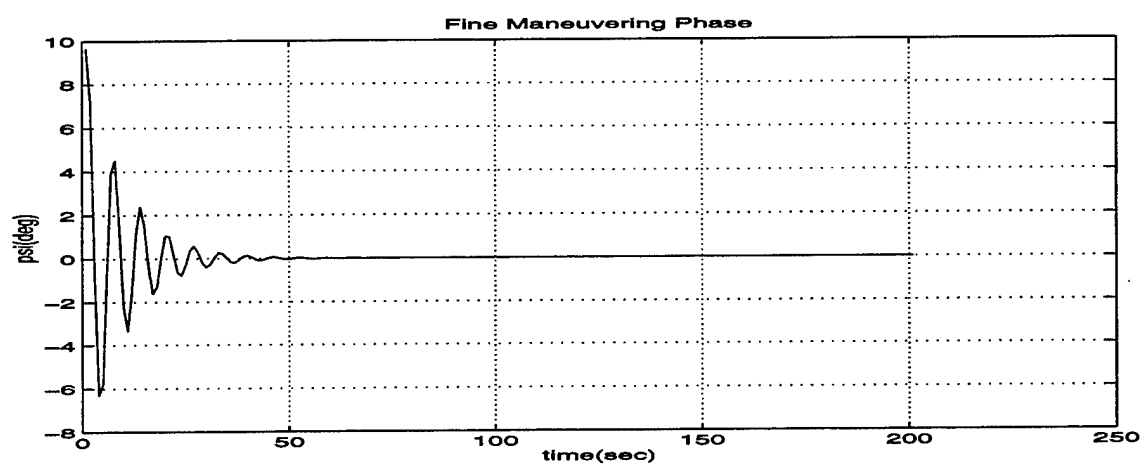

Figure 4.6 Vehicle's Orientation Angle During the Fine Maneuvering Phase.

V. SUMMARY, CONCLUSIONS AND RECOMMENDATIONS

A. SUMMARY

This study investigated the problem of navigation, guidance and control in Autonomous Underwater Vehicles. In particular, a Lyapunov function based guidance and control system and an acoustic motion estimation module are integrated in a two-layered closed-loop control architecture. The control system is designed on the basis of the definition of suitable Lyapunov functions for the different maneuvers in approaching a target. The motion estimation algorithm uses the range and bearing information of a scanning sonar in an Extended Kalman Filter based structure. The combined system is tested in computer simulations, and the results are presented in previous chapters.

B. CONCLUSIONS

The following conclusions have been reached as a result of this study:

- Estimation of vehicle position and velocity is possible with the motion estimation algorithm.
- The algorithm performed adequately through the actual sonar data gathered by the NPS *Phoenix* AUV.
- Lyapunov function approach and defining appropriate constants result in smooth control actions during vehicle motion for the control system algorithm.
- For the combined system, the Lyapunov based guidance system performed adequately in presence of uncertainty in position estimation. On the other hand the performance of the whole system relies on getting better estimates of the vehicle position and velocity.
- The algorithm is applicable to the NPS *Phoenix* AUV. The programs for the acoustic estimation module are written in C programming language and can be directly embedded to the original program of the AUV.

C. RECOMMENDATIONS

The following recommendations are made for follow-on research in the on-going AUV development effort at NPS:

- Convert the control algorithm programs which are written with MATLAB SIMULINK software package to the C programming language for direct implementation into the NPS *Phoenix* original program.
- Generate sonar test data for scenarios of greater complexity to allow testing of the algorithm which goes beyond simulation/modeling.

APPENDIX A. SONAR CHARACTERISTICS

The primary sonar currently installed aboard the NPS *Phoenix* AUV is the Tritech ST1000 Profiling System. This profiling sonar is a high performance, high resolution compact and lightweight system which produces accurate underwater measurements. The following list provides a summary of the sonar performance characteristics and specifications:

Frequency..... 1 MHz.

Beamwidth..... 0.9/1.8 degree conical

Maximum Range..... 50 meters (160 feet)

Minimum Range..... 0.3 meters (1 foot)

Weight ..... 1.5 kg.

View Sector..... 360 degrees

Scan Rate..... 40 degrees/second

APPENDIX B. PROGRAM LISTINGS FOR SIMULATIONS IN CHAPTER II

A. GENERAL

This appendix contains the files generated for the simulations in Chapter II. They are written in C software language. The main program consists of C modules and their corresponding header files. These can be connected through a 'Make file' and run to get results. Basically, whole program reads data which is the real data collected from the scanning sonar of the AUV in the 6x6 feet rectangular test pool. The data length in the header file for initializing the variables, can be changed due to the length of the collected data.

B. PROGRAM LISTINGS

```
/*********************************************************************
  FILENAME:  terms.h
  PURPOSE:   'Header' file to initiate all the program variables
  AUTHOR:    Hakki Celebioglu
  DATE :     1996
*********************************************************************/
#ifndef TERMS_H
#define TERMS_H
#define min(a,b) (((a)< (b)) ? (a):(b))
#define max(a,b) (((a)< (b)) ? (b):(a))
#define MULT3(a,b,c) (matrix_multiply(matrix_multiply(a,b),c))
#define ADDMULT2(a,b,c,d) (matrix_add(matrix_multiply(a,b),matrix_multiply(c,d)))
#define OUT(a) (output_matrix(a))

int I;
int J; /*cell indicators*/
double data[2168][2];
```

```
/*enviroment parameters*/

double xmin; /*pool dimensions*/
double ymin;
double xmax;
double ymax;
int Nx; /*number of the cells*/
int Ny;
double dt; /*time interval*/
matrix neigbors_matrix;
matrixl potential_matrixV;
matrixl potential_matrixVx;
matrixl potential_matrixVy;
matrixl potential_matrixVp1;
matrixl potential_matrixVp2;
matrixl potential_matrixVp3;
matrixl point_i;
matrixl point_j;
matrix covariance_matrix;
matrix covariance_matrixP;
matrix centroid_vector;

/*kalman parameters*/
matrix A;
matrix B;
matrix acceleration_vector;
matrix initial_covariance;
matrix state;
```

```
int window_length;
int I0;
int J0;     /*initial cells*/
matrix data_set;
matrix posest;

/*slope algorithm parameters*/
double eigenvalue;
matrix eigenvector;

#endif

/*********************************************************************
  FILENAME:  matrix.h
  PURPOSE:   'Header' file for matrix operators. In order to minimize the memory usage
             two different type is defined and used according to size of the operation.
*********************************************************************
/
#ifndef MATRIX_H
#define MATRIX_H
/* defines the matrix data structure type */
typedef struct{
      double m[30][30];
      int row, col;
}matrix;

/* defines the large matrix data structure type */
typedef struct{
      double m[50][50];
```

```
    int row, col;
}matrixl;
/* function prototypes for matrix.c */
matrix matrix_multiply(matrix mat1, matrix mat2);
matrix matrix_add(matrix mat1, matrix mat2);
matrix matrix_subtract(matrix mat1, matrix mat2);
matrix matrix_transpose(matrix mat1);
void output_matrix(matrix input_matrix);
void output_matrixl(matrixl input_matrix);
matrix zeros_matrix(int N1,int N2);
matrixl zeros_matrixl(int N1,int N2);
matrix ones_matrix(int N1,int N2);
matrixl ones_matrixl(int N1,int N2);
matrix scalar_matrix(double C,matrix input_matrix);
matrixl scalar_matrixl(double C,matrixl input_matrix);
matrix matrix_diagonal(matrix mat1);
matrix matrix_mean(matrix input_matrix);
matrix identity_matrix(int N1);
matrix matrix_col(int N1,matrix input_matrix);
matrix matrix_row(int N1,matrix input_matrix);

#endif
/*********************************************************************
  FILENAME:  list.h
  PURPOSE:   'Header' file for including operational functions
  AUTHOR:    Hakki Celebioglu
  DATE :     1996
*********************************************************************/
#ifndef LIST_H
```

```
#define LIST_H
/* functions in list.c */
void read_data(int Length);
void initial(double x_min,double y_min,double x_max,double y_max,int NX,int NY);
void initial_kalman(int L,double initial_velx,double initial_vely);
matrix initialize_data(int step,int L);
matrix cartesian_conv(matrix input_matrix);
void estimate(matrix input_matrix);
int round(double number);
void cell(matrix q);
void eig(matrix mat1);
void write_result(char filename[20],matrix mat1);
void write_resultl(char filename[20],matrixl mat1);
void potential(void);
void slope(int length_data);
#endif

/*********************************************************************
  FILENAME:  matrix.c
  PURPOSE:   Create matrix operators to include addition, subtraction,
             multiplication, inverse and gauss_elimination, and
             create a rotation matrix, zero matrix, ones matrix, scalar matrix
             multiplication, diagonal matrix, mean of matrix colons,
             recover row and colons from matrix, identity matrix
  FUNCTIONS: matrix_multiply()
             matrix_add()
             matrix_subtract()
             matrix_transpose()
             output_matrix()
```

```
        output_matrixl()
        zeros_matrix()
        zeros_matrixl()
        ones_matrix()
        ones_matrixl()
        scalar_matrix()
        scalar_matrixl()
        matrix_diagonal()
        matrix_mean()
        matrix_row()
        matrix_col()
        identity_matrix()
*********************************************************************
*/
#include <stdio.h>
#include <math.h>
#include "matrix.h"
/*********************************************************************
  FUNCTION:  matrix_multiply()
  PURPOSE:   Multiplies two matrix's together
  RETURNS:   Matrix1 * Matrix2 in a matrix data structure
*********************************************************************
*/
matrix matrix_multiply(matrix mat1, matrix mat2)
{
int row, col, i;
matrix answer;
/* conducts multiplication */
for (row=0; row<mat1.row; row++) {
```

```
  for (col=0; col<mat2.col; col++) {
      answer.m[row][col]=0.0;
      for (i=0; i <mat1.col; i++){
          answer.m[row][col] += mat1.m[row][i] * mat2.m[i][col];
      }
  }
}
/* assigns new row and col number to matrix data structure */
answer.row = mat1.row;
answer.col = mat2.col;
return answer;
}
/*********************************************************************
  FUNCTION:  matrix_add()
  PURPOSE:   Adds two matrix's together
  RETURNS:   Matrix1 + Matrix2 in a matrix data structure
*********************************************************************
*/
matrix matrix_add(matrix mat1, matrix mat2)
{
int row, col;
matrix answer;
/* conducts addition */
for (row=0; row<mat1.row; row++) {
   for (col=0; col<mat1.col; col++) {
       answer.m[row][col] = mat1.m[row][col] + mat2.m[row][col];
   }
}
/* assigns new row and col number to matrix data structure */
```

```
answer.row = mat1.row;
answer.col = mat1.col;
return answer;
}
/*********************************************************************
  FUNCTION:  matrix_subtract()
  PURPOSE:   Subtracts two matrix's from each other
  RETURNS:   Matrix1 - Matrix2 in a matrix data structure
*********************************************************************
*/
matrix matrix_subtract(matrix mat1, matrix mat2)
{
int row, col;
matrix answer;
/* conducts subtraction */
for (row=0; row<mat1.row; row++) {
   for (col=0; col<mat1.col; col++) {
        answer.m[row][col] = mat1.m[row][col] - mat2.m[row][col];
   }
}
/* assigns new row and col number to matrix data structure */
answer.row = mat1.row;
answer.col = mat1.col;
return answer;
}
/*********************************************************************
  FUNCTION:  matrix_transpose()
  PURPOSE:   Creates the transpose of a matrix
  RETURNS:   transpose(Matrix) in a matrix data structure
```

```
*********************************************************************
*/
matrix matrix_transpose(matrix mat1)
{
int row, col;
matrix answer;
/* conducts transpose */
for (row=0; row<mat1.row; row++) {
   for (col=0; col<mat1.col; col++) {
        answer.m[col][row] = mat1.m[row][col];
   }
}
/* assigns new row and col number to matrix data structure */
answer.row = mat1.col;
answer.col = mat1.row;
return answer;
}
/*********************************************************************
  FUNCTION:  output_matrix()
  PURPOSE:   Prints the contens of a matrix
  RETURNS:   Void
*********************************************************************
*/
void output_matrix(matrix input_matrix)
{
int i,j;
for (i=0;i<input_matrix.row;i++){
  for(j=0;j<input_matrix.col;j++){
     printf("%lf  ",input_matrix.m[i][j]);
```

```
  }
  printf("\n");
}
printf("\n");
}
/*********************************************************************
  FUNCTION:  output_matrixl()
  PURPOSE:   Prints the contens of a matrix l
  RETURNS:   Void
*********************************************************************
*/
void output_matrixl(matrixl input_matrix)
{
int i,j;
for (i=0;i<input_matrix.row;i++){
  for(j=0;j<input_matrix.col;j++){
     printf("%lf  ",input_matrix.m[i][j]);
  }
  printf("\n");
}
printf("\n");
}
/*********************************************************************
  FUNCTION:  zeros_matrix()
  PURPOSE:   Creates a matrix of zeros at given sizes
  RETURNS:   zero(Matrix) in a matrix data structure
*********************************************************************
*/
matrix zeros_matrix(int N1,int N2)
```

```
{
int row, col;
matrix answer;
answer.row=N1;
answer.col=N2;
/* conducts zero */
for (row=0; row<N1; row++)  for (col=0; col<N2; col++)
        answer.m[row][col] = 0.0;
return answer;
}
/*********************************************************************
  FUNCTION:   zeros_matrixl()
  PURPOSE:    Creates a matrix of zeros at given sizes
  RETURNS:    zero(Matrix)l in a matrix data structure
*********************************************************************
*/
matrixl zeros_matrixl(int N1,int N2)
{
int row, col;
matrixl answer;
answer.row=N1;
answer.col=N2;
/* conducts zero */
for (row=0; row<N1; row++)  for (col=0; col<N2; col++)
        answer.m[row][col] = 0.0;
return answer;
}
/*********************************************************************
  FUNCTION:   ones_matrix()
```

```
 PURPOSE:   Creates a matrix of ones at given sizes
 RETURNS:   ones(Matrix) in a matrix data structure
*********************************************************************
*/
matrix ones_matrix(int N1,int N2)
{
int row, col;
matrix answer;
answer.row=N1;
answer.col=N2;
/* conducts ones */
for (row=0; row<N1; row++)  for (col=0; col<N2; col++)
        answer.m[row][col] = 1.0;
return answer;
}
/*********************************************************************
  FUNCTION:  ones_matrixl()
  PURPOSE:   Creates a matrix of ones at given sizes
  RETURNS:   ones(Matrix)l in a matrix data structure
*********************************************************************
*/
matrixl ones_matrixl(int N1,int N2)
{
int row, col;
matrixl answer;
answer.row=N1;
answer.col=N2;
/* conducts ones */
```

```
for (row=0; row<N1; row++)  for (col=0; col<N2; col++)
        answer.m[row][col] = 1.0;
return answer;
}
/*********************************************************************
  FUNCTION:   matrix scalar_matrix()
  PURPOSE:    multiplies a scalar with matrix
  RETURNS:    result(Matrix) in a matrix data structure
*********************************************************************
*/
matrix scalar_matrix(double C,matrix input_matrix)
{
int row, col;
matrix answer;
/* conducts multiplication */
for (row=0; row<input_matrix.row; row++) {
   for (col=0; col<input_matrix.col; col++) {
        answer.m[row][col] = C*input_matrix.m[row][col];
          }
}
answer.row=input_matrix.row;
answer.col=input_matrix.col;
return answer;
}
/*********************************************************************
  FUNCTION:   matrix scalar_matrixl()
  PURPOSE:    multiplies a scalar with matrix
  RETURNS:    result(Matrix)l in a matrix data structure
```

```
*********************************************************************
*/
matrixl scalar_matrixl(double C,matrixl input_matrix)
{
int row, col;
matrixl answer;
/* conducts multiplication */
for (row=0; row<input_matrix.row; row++) {
   for (col=0; col<input_matrix.col; col++) {
       answer.m[row][col] = C*input_matrix.m[row][col];
        }
}
answer.row=input_matrix.row;
answer.col=input_matrix.col;
return answer;
}
/*********************************************************************
  FUNCTION:  matrix_diagonal()
  PURPOSE:   writes elements of one dimensional matrix to diagonal
  RETURNS:   Diagonal(Matrix) in a matrix data structure
*********************************************************************
*/
matrix matrix_diagonal(matrix mat1)
{
int row, col;
matrix ans;
/* conducts diagonal */
if (mat1.row==1 || mat1.col==1){
   if (mat1.row!=1)
```

```
  mat1=matrix_transpose(mat1);
  for (col=0; col<mat1.col; col++)
  ans.m[col][col] = mat1.m[0][col];
  ans.row = mat1.col;
  ans.col = mat1.col;
  }
 for(row=0;row<ans.row;row++)for(col=0;col<ans.col;col++)
  {
  if(row!=col)
  ans.m[row][col]=0.0;
  }
return ans;
}
/*********************************************************************
  FUNCTION:  matrix_mean()
  PURPOSE:   Gets mean of the colons of the matrix
  RETURNS:   mean (matrix) in a matrix data structure
*********************************************************************
*/
matrix matrix_mean(matrix input_matrix)
{
int  row,col;
double a=0.0;
matrix answer;
/* getting mean of colons */
  for (col=0; col<input_matrix.col; col++){
     for(row=0; row<input_matrix.row;row++){
        a+= input_matrix.m[row][col]/(double) input_matrix.row ;
        answer.m[0][col]=a;
```

```
        }
      a=0.0;
      }
answer.row=1;
answer.col=input_matrix.col;
return answer;
}
/*********************************************************************
  FUNCTION:  matrix_row()
  PURPOSE:   Gets given rows of the matrix
  RETURNS:   result(in given sizes) in a matrix data structure
*********************************************************************
*/
matrix matrix_row(int N1,matrix input_matrix)
{
int  col;
matrix answer;
/* getting wanted row */
   for (col=0; col<input_matrix.col; col++) {
        answer.m[0][col] = input_matrix.m[N1-1][col];
        }
answer.row=1;
answer.col=input_matrix.col;
return answer;
}
/*********************************************************************
  FUNCTION:  matrix_col()
  PURPOSE:   Gets given col of the matrix
  RETURNS:   result(one col) in a matrix data structure
```

```
*********************************************************************
*/
matrix matrix_col(int N1,matrix input_matrix)
{
matrix answer;
/* getting wanted col */
answer=matrix_transpose(input_matrix);
answer=matrix_transpose(matrix_row(N1,answer));
return answer;
}
/*********************************************************************
  FUNCTION:  identity_matrix()
  PURPOSE:   establish given size identity matrix
  RETURNS:   identity matrix in Matrix structure
*********************************************************************
*/
matrix identity_matrix(int N1)
{
matrix answer;
answer=ones_matrix(1,N1);
answer=matrix_diagonal(answer);
return answer;
}
```

```
/*********************************************************************
  FILENAME:  list.c
  PURPOSE:   All operations of the algorithm
 AUTHOR:     Hakki Celebioglu
  DATE :     1996
*********************************************************************/
#include <stdio.h>
#include <math.h>
#include "matrix.h"
#include "terms.h"
#include "list.h"
#define pi 3.14159265358979
/*
                Functions
              read_data()
              initial()
              initial_kalman()
              initialize_data()
              cartesian_conv()
              estimate()
              round()
              cell()
              eig
              write_result()
              write_resultl()
              potential()
              slope()
/*********************************************************************
  FUNCTION:  read_data()
```

```
  PURPOSE:   Reading file to an array defined globally
  RETURNS:   void
*********************************************************************/
void read_data(int Length)
{
 int row,col;
 FILE *fp;
  if((fp=fopen("test.txt","r"))==NULL)
  {
   fprintf(stderr,"Error openning");
  }
 for(row=0;row<Length;row++)for(col=0;col<2;col++){
          fscanf(fp,"%lf ",&data[row][col]);
    }
 fclose(fp);
 }
/*********************************************************************
  FUNCTION:  initial()
  PURPOSE:   initilize neighbors_matrix,I,J,I0,J0,potential matrices
          point_i,point_j,xmin,xmax,ymin,ymax,Nx,Ny
  RETURNS:   void ( all are globally decleared)
*********************************************************************/
void initial(double x_min,double y_min,double x_max,double y_max,int NX,int NY)
{
double a,b,c;
a=sqrt(2.0);
b=2.0*a;
c=sqrt(5.0);
xmin=x_min;
```

```
ymin=y_min;
xmax=x_max;
ymax=y_max;
Nx=NX;
Ny=NY;
dt=0.1;
potential_matrixV=zeros_matrixl(Nx,Ny);
potential_matrixVx=potential_matrixV;
potential_matrixVy=potential_matrixV;
potential_matrixVp1=potential_matrixV;
potential_matrixVp2=potential_matrixV;
potential_matrixVp3=potential_matrixV;
potential_matrixV=scalar_matrixl(10.0,ones_matrixl(NX,NY));
point_i=scalar_matrixl(-1.0,ones_matrixl(NX,NY));
point_j=point_i;
neigbors_matrix.m[0][0]=b;
neigbors_matrix.m[0][1]=c;
neigbors_matrix.m[0][2]=2.0;
neigbors_matrix.m[0][3]=c;
neigbors_matrix.m[0][4]=b;
neigbors_matrix.m[1][0]=c;
neigbors_matrix.m[1][1]=a;
neigbors_matrix.m[1][2]=1.0;
neigbors_matrix.m[1][3]=a;
neigbors_matrix.m[1][4]=c;
neigbors_matrix.m[2][0]=2.0;
neigbors_matrix.m[2][1]=1.0;
neigbors_matrix.m[2][2]=0.0;
neigbors_matrix.m[2][3]=1.0;
```

```
neigbors_matrix.m[2][4]=2.0;
neigbors_matrix.m[3][0]=c;
neigbors_matrix.m[3][1]=a;
neigbors_matrix.m[3][2]=1.0;
neigbors_matrix.m[3][3]=a;
neigbors_matrix.m[3][4]=c;
neigbors_matrix.m[4][0]=b;
neigbors_matrix.m[4][1]=c;
neigbors_matrix.m[4][2]=2.0;
neigbors_matrix.m[4][3]=c;
neigbors_matrix.m[4][4]=b;
neigbors_matrix.row=5;
neigbors_matrix.col=5;
 }
/*********************************************************************
  FUNCTION:  initial_kalman()
  PURPOSE:   initilize Kalman dynamics(initial_state,initial_covariance,
             window_length,A,B matrices
  RETURNS:   void ( all are globally decleared)
  *********************************************************************/
 void initial_kalman(int L,double initial_velx,double initial_vely)
{
I0=0;
J0=0;
A=zeros_matrix(4,4);
A.m[0][2]=1.0;
A.m[1][3]=1.0;
A.row=4;
A.col=4;
```

```
A=matrix_add(identity_matrix(4),scalar_matrix(dt,A));
B.m[0][0]=0.0;
B.m[0][1]=0.0;
B.m[1][0]=0.0;
B.m[1][1]=0.0;
B.m[2][0]=1.0;
B.m[2][1]=0.0;
B.m[3][0]=0.0;
B.m[3][1]=1.0;
B.row=4;
B.col=2;
B=scalar_matrix(dt,B);
state.m[0][0]=0.0;
state.m[1][0]=0.0;
state.m[2][0]=initial_velx;
state.m[3][0]=initial_vely;
state.row=4;
state.col=1;
initial_covariance.m[0][0]=0.0;
initial_covariance.m[0][1]=0.0;
initial_covariance.m[0][2]=0.01;
initial_covariance.m[0][3]=0.01;
initial_covariance.row=1;
initial_covariance.col=4;
initial_covariance=matrix_diagonal(initial_covariance);
window_length=L;
acceleration_vector.m[0][0]=0.0;
acceleration_vector.m[1][0]=0.0;
acceleration_vector.row=2;
```

```
acceleration_vector.col=1;
}
/*********************************************************************
  FUNCTION:  initialize_data()
  PURPOSE:   Reading file to a matrix (size*2(angle&range) windowing)
  RETURNS:   data matrix in Matrix structure
*********************************************************************
*/
matrix initialize_data(int step,int L)
{
 int row,i;
 matrix answer1;
 int size=L*2+1;
  i=0;
  for(row=step*size;row<step*size+size;row++){
  answer1.m[i][0]=data[row][0];
  answer1.m[i][1]=data[row][1];
  i++;
   }
 answer1.row=size;
 answer1.col=2;
 return answer1;
 }
/*********************************************************************
  FUNCTION:  cartesian_conv()
  PURPOSE:   making conversian (theta&rho) to (x&y)
  RETURNS:   matrix in Matrix structure
*********************************************************************
*/
```

```
matrix cartesian_conv(matrix input_matrix)
{
 int row,col;
 matrix answer,answer1;
 matrix s1,s2,theta,rho;
 double a=pi/180.0;
 theta=matrix_transpose(scalar_matrix(a,matrix_col(1,input_matrix)));
 for(row=0;row<theta.row;row++) for(col=0;col<theta.col;col++){
    s1.m[row][col]=cos(theta.m[row][col]);
    s2.m[row][col]=sin(theta.m[row][col]);  }
 s1.row=theta.row;
 s2.row=theta.row;
 s1.col=theta.col;
 s2.col=s1.col;
 for(col=0;col<theta.col;col++){
    answer1.m[0][col]=s1.m[0][col];
    answer1.m[1][col]=s2.m[0][col];
    }
answer1.row=2*theta.row;
answer1.col=theta.col;
rho=matrix_diagonal(matrix_col(2,input_matrix));
answer=matrix_multiply(answer1,rho);
 answer.row=input_matrix.col;
 answer.col=input_matrix.row;
 return answer;
 }
/*********************************************************************
  FUNCTION:  estimate()
  PURPOSE:   estimates covariance matrix and centroid vector
```

```
  RETURNS:    void (parameter would be defined global
              so these will be initialize in a matrix data structure(P and c)
              P~covariance_matrix, c~centroid_vector(one colon)
**********************************************************************
*/
 void estimate(matrix input_matrix)
{
matrix Q,QQ,PP;
centroid_vector=matrix_transpose(matrix_mean(matrix_transpose(input_matrix)));
QQ=matrix_multiply(centroid_vector,ones_matrix(1,input_matrix.col));
Q=matrix_subtract(input_matrix,QQ);
PP=matrix_multiply(Q,matrix_transpose(Q));
covariance_matrix=scalar_matrix(1.0/(double)input_matrix.col,PP);
}
/**********************************************************************
  FUNCTION:   round()
  PURPOSE:    rounds the number
  RETURNS:    integer rounding
**********************************************************************
*/
int round(double number)
 {
   int no;
   if (number>0)
      {
      if(number+0.5>ceil(number))
      number = ceil(number);
      else
      number=floor(number);
```

```
    no=(int)number;
    }
  else
    {
    number=-1.0*number;
    if(number+0.5>ceil(number))
    number = ceil(number);
    else
    number=floor(number);
    no=(int)number;
    no=-1*no;
    }
  return no;
}
/*********************************************************************
 FUNCTION:  cell()
 PURPOSE:   calculates the values of cell indicators I,J
            input would be a vector
  RETURNS:  void ( I&J are globally decleared)

/*********************************************************************
/
 void cell(matrix q)
 {
 double qq,qqq,deltax,deltay;
 int v,vv;
 deltax=(xmax-xmin)/(double)Nx;
 deltay=(ymax-ymin)/(double)Ny;
 if (q.row>q.col){
```

```
  qq=(q.m[0][0]-xmin)/deltax;
  v=round(qq);
  qqq=(q.m[1][0]-ymin)/deltay;
  vv=round(qqq);
  if(v<1)v=1;
  if(vv<1)vv=1;
  if(v>Nx)  v=Nx;
  if(vv>Ny) vv=Ny;
  I=v;
  J=vv;
 }
else{
  qq=(q.m[0][0]-xmin)/deltax;
  v=round(qq);
  qqq=(q.m[0][1]-ymin)/deltay;
  vv=round(qqq);
  if(v<1)v=1;
  if(vv<1)vv=1;
  if(v>Nx)  v=Nx;
  if(vv>Ny) vv=Ny;
  I=v;
  J=vv;
  }
}
/*********************************************************************
 FUNCTION:  eig()
 PURPOSE:   computes eigenvalue and eigenvector of 2*2 matrix
 RETURNS:   void(globally)
```

```
**********************************************************************
*/
void eig (matrix mat1)
{
 double tol,l,lsav,kk;
 matrix x,z;
if(mat1.m[0][0]==0.0    &&    mat1.m[0][1]==0.0    &&    mat1.m[1][0]==0.0    &&
mat1.m[1][1]==0.0)
   {
   eigenvalue=0.0;
   eigenvector.m[0][0]=0.0;
   eigenvector.m[1][0]=1.0;
   eigenvector.row=2;
   eigenvector.col=1;
   }
 else
    {
 l=0.0;
 lsav=1.0;
 tol=0.00001;
 x=ones_matrix(2,1);
 while(fabs(lsav-l)> tol)
     {
     lsav=l;
     z=matrix_multiply(mat1,x);
     l=max(fabs(z.m[0][0]),fabs(z.m[1][0]));
     x=scalar_matrix((1.0/l),z);
     }
eigenvalue=l;
```

```
eigenvector=x;
kk=eigenvector.m[0][0]/eigenvector.m[1][0];
eigenvector.m[1][0]=sqrt(1.0/(1+kk*kk));
eigenvector.m[0][0]=sqrt(1-eigenvector.m[1][0]*eigenvector.m[1][0]);
if (kk<0)
eigenvector.m[0][0]=-1.0*eigenvector.m[0][0];
eigenvector.row=2;
eigenvector.col=1;
  }
}
/*********************************************************************
  FUNCTION:  potential()
  PURPOSE:   Estimates the potential function of the enviroment(cells)
  RETURNS:   fills all the globally decleared variables
*********************************************************************
*/
void potential(void)
{
int iter,i,tt,ttt,ii,jj,is,js,ns;
 matrix s,sL,tip_sonar,tempq,temp;
 double D1,D2;
 for (iter=2*window_length+1;iter<=400;iter++)
    {
    s=cartesian_conv(initialize_data(iter-1,0));
    state=matrix_add(matrix_multiply(A,state),matrix_multiply(B,acceleration_vector));
    if (data_set.row>data_set.col)
    ns=data_set.row;
    else
    ns=data_set.col;
```

```
    for(ttt=0;ttt<data_set.row;ttt++)
      {
       sL.m[ttt][0]=data_set.m[ttt][ns-window_length-1];
       sL.row=data_set.row;
       sL.col=1;
      }
    tempq.m[0][0]=state.m[0][0];
    tempq.m[1][0]=state.m[1][0];
    tempq.row=sL.row;
    tempq.col=sL.col;
    tip_sonar=matrix_add(tempq,sL);
    cell(tip_sonar);
/*printf("%d\t%d\t%d\t%d\n",I,J,I0,J0);        */
    if (I==I0 && J==J0)
       {
       data_set.col=data_set.col+s.col;
       data_set.m[0][data_set.col-1]=s.m[0][0];
       data_set.m[1][data_set.col-1]=s.m[1][0];
       posest.col=posest.col+s.col;
       posest.m[0][posest.col-1]=state.m[0][0];
       posest.m[1][posest.col-1]=state.m[1][0];
       }
    else
       {
       estimate(matrix_add(posest,data_set));
       covariance_matrixP=covariance_matrix;
       tt=0;
        for(i=(ns-2*window_length)-1;i<ns;i++)
          {
```

```
    tempq.m[0][tt]=data_set.m[0][i];
    tempq.m[1][tt]=data_set.m[1][i];
    temp.m[0][tt]=posest.m[0][i];
    temp.m[1][tt]=posest.m[1][i];
    tt++;
    }
 data_set=tempq;
 posest=temp;
 data_set.row=2;
 data_set.col=2*window_length+1;
 posest.row=2;
 posest.col=2*window_length+1;
I0=I;
J0=J;
if (I0!=0 && J0!=0)
   {
   potential_matrixVx.m[I0-1][J0-1]=centroid_vector.m[0][0];
   potential_matrixVy.m[I0-1][J0-1]=centroid_vector.m[1][0];
   potential_matrixVp1.m[I0-1][J0-1]=covariance_matrixP.m[0][0];
   potential_matrixVp2.m[I0-1][J0-1]=covariance_matrixP.m[1][1];
   potential_matrixVp3.m[I0-1][J0-1]=covariance_matrixP.m[0][1];
   potential_matrixV.m[I0-1][J0-1]=0.0;
   for (ii=0;ii<5;ii++){for(jj=0;jj<5;jj++)
      {
      is=ii-2;
      js=jj-2;
      if(I0+is>0&&J0+js>0&&I0+is<Nx+1&&J0<Ny+1)
        {
        D1=potential_matrixV.m[I0+is-1][J0+js-1];
```

```
                D2=neigbors_matrix.m[ii][jj];
                if(D2<D1)
                  {
                  potential_matrixV.m[I0+is-1][J0+js-1]=D2;
                  point_i.m[I0+is-1][J0+js-1]=(double)I0;
                  point_j.m[I0+is-1][J0+js-1]=(double)J0;
                  }
                }
              }}
            point_i.m[I0-1][J0-1]=-1.0;
            point_j.m[I0-1][J0-1]=-1.0;
            }
      }
   }
}
/*********************************************************************
  FUNCTION:  slope()
  PURPOSE:   Estimates the slope of the enviroment
  RETURNS:   filling the the slope variables
*********************************************************************
*/
void slope(int length_data)
{
 int iter,tt,ttt,i,slope_window,i1,j1;
 matrix s,tempq,covariance_matrixW,temp;
 matrix eigenvectorw,eigenvectorp,phi,H,K;
 matrixl Q;
 double eigenvaluep,indice,R,den;
 FILE *fp;
```

```
  if((fp=fopen("result1.m","w"))==NULL)
  fprintf(stderr,"Error openning");
  fprintf(fp,"xm=[");

 slope_window=10;
  state=matrix_add(matrix_multiply(A,state),matrix_multiply(B,acceleration_vector));
for (tt=401;tt<=400+slope_window;tt++)
    {
    s=cartesian_conv(initialize_data(tt-1,0));
    if(tt==401)
      {
      data_set=s;
      posest.m[0][0]=state.m[0][0];
      posest.m[1][0]=state.m[1][0];
      posest.row=2;
      posest.col=1;
      }
    else
      {
      data_set.col=data_set.col+1;
      data_set.m[0][data_set.col-1]=s.m[0][0];
      data_set.m[1][data_set.col-1]=s.m[1][0];
      posest.col=posest.col+1;
      posest.m[0][posest.col-1]=state.m[0][0];
      posest.m[1][posest.col-1]=state.m[1][0];
      }
 fprintf(fp,"%lf\t%lf\t%lf\t%lf\n",state.m[0][0],state.m[1][0],state.m[2][0],state.m[3][0]);
     }
     Q=zeros_matrixl(Nx,Ny);
```

```
for (iter=401+slope_window;iter<=length_data;iter++)
   {
   estimate(matrix_add(posest,data_set));
   covariance_matrixW=covariance_matrix;
   cell(centroid_vector);
   while ((point_i.m[I-1][J-1]!=-1.0) && (point_j.m[I-1][J-1]!=-1.0))
       {
       i1=(int) point_i.m[I-1][J-1];
       j1=(int) point_j.m[I-1][J-1];
       I=i1;
       J=j1;
       }
    Q.m[I-1][J-1]=1.0;
    centroid_vector.m[0][0]=potential_matrixVx.m[I-1][J-1];
    centroid_vector.m[1][0]=potential_matrixVy.m[I-1][J-1];
    covariance_matrixP.m[0][0]=potential_matrixVp1.m[I-1][J-1];
    covariance_matrixP.m[0][1]=potential_matrixVp3.m[I-1][J-1];
    covariance_matrixP.m[1][0]=potential_matrixVp3.m[I-1][J-1];
    covariance_matrixP.m[1][1]=potential_matrixVp2.m[I-1][J-1];
    covariance_matrixP.row=2;
    covariance_matrixP.col=2;
    eig(covariance_matrixP);
    eigenvectorp=eigenvector;
    eigenvaluep=eigenvalue;
    eig(covariance_matrixW);
    eigenvectorw=eigenvector;
    if (eigenvaluep!=0.0)
      {
```

```
tempq=MULT3(matrix_transpose(eigenvectorw),covariance_matrixP,eigenvectorw);
      indice=tempq.m[0][0]/eigenvaluep;
      }
    R=0.1;
    if (indice < 0.9 || covariance_matrixP.m[0][0]+covariance_matrixP.m[1][1]==0.0)
      R=100000.0;
    phi.m[0][0]=eigenvectorp.m[1][0];
    phi.m[1][0]=-1.0*eigenvectorp.m[0][0];
    phi.row=2;
    phi.col=1;
    H.m[0][0]=phi.m[0][0];
    H.m[0][1]=phi.m[1][0];
    H.m[0][2]=0.0;
    H.m[0][3]=0.0;
    H.row=1;
    H.col=4;
    tempq=MULT3(H,initial_covariance,matrix_transpose(H));
    den=R+tempq.m[0][0];
    K=scalar_matrix(1.0/den,MULT3(A,initial_covariance,matrix_transpose(H)));
    temp=matrix_subtract(centroid_vector,s);
    temp=matrix_multiply(matrix_transpose(temp),phi);
    temp=matrix_subtract(temp,matrix_multiply(H,state));
state=matrix_add(ADDMULT2(A,state,B,acceleration_vector),matrix_multiply(K,temp))
;
initial_covariance=matrix_subtract(MULT3(A,initial_covariance,matrix_transpose(A)),m
atrix_multiply(scalar_matrix(den,K),matrix_transpose(K)));
    s=cartesian_conv(initialize_data(iter-1,0));
    for(ttt=0;ttt<data_set.row;ttt++) for (i=2;i<=slope_window;i++)
```

```
        {
        data_set.m[ttt][i-2]=data_set.m[ttt][i-1];
        posest.m[ttt][i-2]=posest.m[ttt][i-1];
        }
    data_set.row=data_set.row;
    data_set.col=slope_window-1;
    posest.col=slope_window-1;
    data_set.col=data_set.col+1;
    data_set.m[0][data_set.col-1]=s.m[0][0];
    data_set.m[1][data_set.col-1]=s.m[1][0];
    posest.col=posest.col+1;
    posest.m[0][posest.col-1]=state.m[0][0];
    posest.m[1][posest.col-1]=state.m[1][0];
  fprintf(fp,"%lf\t%lf\t%lf\t%lf\n",state.m[0][0],state.m[1][0],state.m[2][0],state.m[3][0]);
      }
fprintf(fp,"];figure(1),plot(xm(:,1));figure(2),plot(xm(:,2));figure(3),plot(xm(:,3));figure(4
),plot(xm(:,4));");
}
/*********************************************************************
  FUNCTION:   write_result()
  PURPOSE:    Writing a file
  RETURNS:    creates a file for pltting
*********************************************************************
*/
void write_result(char filename[20],matrix mat1)
{
 int row,col;
 FILE *fp;
  if((fp=fopen(filename,"w"))==NULL)
```

```
  fprintf(stderr,"Error openning");
  for(row=0;row<mat1.row;row++)
    {
     for(col=0;col<mat1.col;col++)
       fprintf(fp,"%lf\t ",mat1.m[row][col]);
    fprintf(fp,"\n");
      }
      fclose(fp);
      }
/*********************************************************************
  FUNCTION:  main()
  PURPOSE:   main program file
  RETURNS:   resulting
*********************************************************************
*/
#include <stdio.h>
#include <math.h>
#include "matrix.h"
#include "list.h"
#include "terms.h"
#define pi 3.14159265358979
int main(void)
{
 int data_length;
 printf("Enter the data length");
 scanf("%d",&data_length);
 read_data(data_length);
 initial(-6.0,-6.0,6.0,6.0,50,50);
 initial_kalman(2,0.0,0.0);
```

```
data_set=cartesian_conv(initialize_data(0,window_length));
posest=zeros_matrix(data_set.row,data_set.col);
potential();
slope(data_length);
return 1;
}
```

APPENDIX C. PROGRAM LISTINGS FOR CONTROL ALGORITHM SIMULATIONS

A. GENERAL

This appendix contains the MATLAB files generated for the simulations in Chapter III and IV. These files are used in MATLAB SIMULINK simulation models for the control system. The constants used are defined in a trial error basis on the occasion of the intended trajectory of the vehicle, and will have to be determined for every different mission.

B. PROGRAM LISTINGS

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  FILENAME:  occont.m
  PURPOSE:   MATLAB function giving the outputs of the outer control-loop
  AUTHOR:    Hakki Celebioglu
  DATE :     1996
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function uvwstar=occont(xxx)
% Target positions
xo=10;
yo=10;
x=xxx(1);  % estimated x-axis vehicle position
y=xxx(2);  % estimated y-axis vehicle position
sie=xxx(3); % orientation of the vehicle with respect to target frame
e=xxx(4);  % the distance between the vehicle and the target frame
theta=xxx(5);  % the angle θ defined in Chapter II.
alfa=xxx(6);  % the angle α defined in Chapter II.
% The constant denoted for the Long Range Phase
kLR=0.12;
%The constant denoted for the Medium Range Phase
```

```
kMR=0.0173;
%The constant denoted for Fine Maneuvering Phase
kF=0.0001;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Long Range Navigation
if(e>=7.1)
  siestar=atan2(yo-y,xo-x);
  ustar=-kLR*(x*cos(siestar)+y*sin(siestar));
  vstar=0;
  wstar=0;
  uvwstar=[ustar vstar wstar];
end
% Medium Range Navigation
if(e<7.1 & e>=1)
  ustar=kMR*e*cos(alfa);
  vstar=0;
  wstar=.01*alfa+kMR*(cos(alfa)*sinc(alfa/pi)*(alfa-1.26*theta));
  uvwstar=[ustar vstar wstar];
end
% Fine Maneuvering
if(e<1)
  ustar=-kF*(x*cos(sie)+y*sin(sie));
  vstar=-kF*(-x*sin(sie)+y*cos(sie));
  wstar=-0.8*sie;  % here gamma is 0.8.
  uvwstar=[ustar vstar wstar];
end
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  FILENAME:   iccont.m
  PURPOSE:    MATLAB function giving the outputs of the inner control-loop
  AUTHOR:     Hakki Celebioglu
  DATE :      1996
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function fgM=iccont(stars)
% Outputs of the outer control loop will be inputs for inner control loop
ustar=stars(1);
vstar=stars(2);
wstar=stars(3);
% The linear and angular velocities
u=stars(7);
v=stars(8);
w=stars(9);
% Time drivatives of the output of the outer control loop
ustard=stars(4);
vstard=stars(5);
wstard=stars(6);
%Inner Control Loop adjusting parameters
px=0.1;
py=0.04;
pm=0.001;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% The lateral trusts and the angular moment being computed
f=-v*w+ustard-px*(u-ustar);
g=u*w+vstard-py*(v-vstar);
M=wstard-pm*(w-wstar);
fgM=[f g M u v w]; % The computed forces and the velocities will be outputs.
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
 FILENAME:  dinamic.m
 PURPOSE:   MATLAB function for dynamics of the vehicle
 AUTHOR:    Hakki Celebioglu
 DATE :     1996
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function uvw=dinamic(xxx)
% the outputs of the inner control loop will be inputs for vehicle's dynamics.
f=xxx(1);
g=xxx(2);
M=xxx(3);
u=xxx(4);
v=xxx(5);
w=xxx(6);
%The state space parameters of the vehicle's dynamics
ud=f+v*w;
vd=g-u*w;
wd=M;
uvw=[ud vd wd];
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
 FILENAME:  xysie.m
 PURPOSE:   MATLAB function for kinematics of the vehicle
 AUTHOR:    Hakki Celebioglu
 DATE :     1996
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function xysie=scont(xxx)
% Second set of system states and system kinematics
u=xxx(1);
v=xxx(2);
```

```
w=xxx(3);
sie=xxx(4);
alfa=xxx(5);
e=xxx(6);
xd=u*cos(sie)-v*sin(sie);
yd=u*sin(sie)+v*cos(sie);
sied=w;
ed=-u*cos(alfa)-v*sin(alfa);
thetad=(u*sin(alfa)/e)-(v*cos(alfa)/e);
alfad=-w+thetad;
xysie=[xd yd sied ed thetad alfad];
```

LIST OF REFERENCES

1. R.Cristi, M.Caccia, G.Veruggio, "Motion Estimation and Modeling of the Environment for Underwater Vehicles," paper presented at *Proceedings of 6th IARP in Underwater Robotics,* Toulon, France , 1996.

2. E.Percin, *Sonar Localization of an Autonomous Underwater Vehicle,* Master's Thesis, Naval Postgraduate School, Monterey, CA, December 1993.

3. K.D.Conowitch, *Sensor Based Navigation and Localization Methods for Autonomous Underwater Vehicles,* Master's Thesis, Naval Postgraduate School, Monterey, CA, June 1995.

4. C.K.Chui, G.Chen, *Kalman Filtering with Real-Time Applications,* Springer-Verlag Series in Information Science, 2nd Edition, New York, NY, 1991.

5. J.-J.E. Slotine, W. Li, *Applied Nonlinear Control,* Prentice-Hall Inc., Englewood Cliffs, New Jersey, NY, 1991.

6. G.Casalino, G.Cannata, M.Caccia, "Lyapunov Based Closed-Loop Motion Control for UUV's," paper presented at *Proceedings of 3rd International Symposium on Methods and Models in Automation and Robotics,* Miedzyzdroje, Poland, 1996.

7. M.Caccia, A.C.Colombo, G.Casalino, M.Decia, G.Veruggio, "Closed-loop Approach Algorithm Based on Lyapunov Techniques for an Autonomous Underwater Vehicle," paper presented at *Proceedings of 3rd IFAC Workshop on Control Applications in Marine Systems,* Trondheim, Norway, May 1995.

8. M.Caccia, G.Casalino, R.Cristi, G.Veruggio, "Acoustic Motion Estimation and Control for an Unmanned Underwater Vehicle in a Structural Environment," paper presented at *MMC 97,* Brijuni, Croatia, September 1997.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Technical Information Center 8725 John J. Kingman Rd., STE 0944 Ft. Belvoir, VA 22060-6218	2
2.	Dudley Knox Library Naval Postgraduate School 411 Dyer Rd. Monterey, CA 93943-5101	2
3.	Chairman, Code EC Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, CA 93943-5121	1
4.	Professor Roberto Cristi, Code EC/Cx Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, CA 93943-5121	2
5.	Professor Xiapping Yun, Code EC/Xi Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, CA 93943-5121	1
6.	Deniz Kuvvetleri Komutanlığı Personel Daire Başkanlığı Bakanlıklar, Ankara, TURKEY	2
7.	Deniz Harp Okulu Komutanlığı Tuzla, İstanbul- 81704, TURKEY	1
8.	Birol Zeybek Deniz Kuvvetleri Komutanlığı Gemi İnşa Dairesi Bşk. Bakanlıklar, Ankara, TURKEY	1
9.	Hakkı Çelebioğlu 1730 Sokak No: 69/3 Karşıyaka, İzmir-35530, TURKEY	2