COOPERATIVE CONTROL OF AUTONOMOUS UNDERWATER VEHICLES

A Thesis

by

ELIZABETH SAVAGE

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

May 2003

Major Subject: Aerospace Engineering

COOPERATIVE CONTROL OF AUTONOMOUS UNDERWATER VEHICLES

A Thesis

by

ELIZABETH SAVAGE

Submitted to Texas A&M University
in partial fulfillment of the requirements
for the degree of

MASTER OF SCIENCE

Approved as to style and content by:

May 2003

Major Subject: Aerospace Engineering

ABSTRACT

Cooperative Control of Autonomous

Underwater Vehicles. (May 2003)

Elizabeth Savage, M.E., Sheffield University, UK

Chair of Advisory Committee: Dr. John E. Hurtado

The proposed project is the simulation of a system to search for air vehicles which have splashed down in the ocean. The system comprises a group of 10+ autonomous underwater vehicles, which cooperate in order to locate the aircraft. The search algorithm used in this system is based on a quadratic Newton method and was developed at Sandia National Laboratories. The method has already been successfully applied to several two dimensional problems at Sandia.

The original 2D algorithm was converted to 3D and tested for robustness in the presence of sensor error, position error and navigational error. Treating the robots as point masses, the system was found to be robust for all such errors.

Several real life adaptations were necessary. A round robin communication strategy was implemented on the system to properly simulate the dissemination of information throughout the group. Time to convergence is delayed but the system still functioned adequately.

Once simulations for the point masses had been exhausted, the dynamics of the robots were included. The robot equations of motion were described using Kane's equations. Path-planning was investigated using optimal control methods. The Variational Calculus approach was attempted using a line search tool "fsolve" found in Matlab and a Genetic Algorithm. A dynamic programming technique was also investigated using a method recently developed by Sandia National Laboratories. The Dynamic Programming with Interior Points (DPIP) method was a very efficient method for path planning and performed well in the presence of system constraints.

Finally all components of the system were integrated. The motion of the robot exactly matched the motion of the particles, even when subjected to the same robustness tests carried out on the point masses. This thesis provides exciting developments for all types of cooperative projects.

To my mother, Isabelle Savage

ACKNOWLEDGMENTS

I would like to thank Professor Peter Fleming, department of automatic controls and systems engineering, Sheffield University, U.K. and Dr. John Junkins, department of aerospace engineering, Texas A&M University whose collaboration made it possible for me to study here. Secondly, I would like to thank my advisor, Dr. John Hurtado for his infective enthusiasm for his work, his excellent teaching abilities and his support over the last eighteen months. Thirdly, thanks to DARPA, who funded my research, to Dr. Ray Robinett III at Sandia National Laboratories who supervised the project and to Nekton Reasearch LLC, who provided the robots. I thank my committee, Dr. John Junkins, Dr.John Valasek and Dr. Darbha Swaroop who were ready with advice and encouragement and also Dr. Andrew Chipperfield of Southampton University (U.K.) to whom I owe all my current knowledge on genetic algorithms. I would also like to recognize the faculty in the aerospace department at Texas A&M for their wisdom and humor and my office mate, Dr. Pavlin Entchev without whom LaTeXwould have remained a mystery.

Finally I thank my family for letting me come 3000 miles to study here, and all my friends at Texas A&M, particularly Daniel Nobles, Edward Caicedo, Maria Morgun, Sheetal Desai, Mitren Chinoy, Michael Quintana and countless others who welcomed me here in College Station, encouraged me and kept me motivated. This thesis would not have been possible without you.

TABLE OF CONTENTS

LIST OF TABLES

TABLE Page

LIST OF FIGURES

FIGURE Page

CHAPTER I

INTRODUCTION AND LITERATURE REVIEW

A. Background to Cooperative Systems

Robots and small remote controlled vehicles have been with us since the seventies, however, over the last decade we have seen an increase in the intelligence and autonomy of such systems. At the same time, it has become increasingly obvious that systems would be more robust and cost-effective if the desired task was performed by a group of small, simple robots rather than one large, complex robot. Cooperative systems have already been proposed for a variety of applications from space experiments [1] to detecting the sources of water pollution [2], and from manufacturing to managing radioactive waste.

To the author's knowledge, underwater cooperative systems have not received as much attention as space, land and air applications. Research is continuing at the Naval military college; Naomi Leonard at the University of Princeton has covered a wide range of features associated with Unmanned Underwater Vehicles (UUV's) [3] and the University of Hawaii are also investigating the interaction of multiple robots and the stability of such systems. Underwater groups around the world have generally focused on large, individual unmanned underwater vehicles (UUV's). These systems tend to be multi-functional, capable of both search and retrieval. Moreover, most current cooperative systems use less than five robots, regardless of the medium through which, or on which, they travel. This project has used ten and is currently pushing the envelope in the field of cooperative robotics. The project is headed up by SANDIA National Labs and the UUV's are provided by Nekton Research, NC.

The journal model is *IEEE Transactions on Automatic Control.*

B. An Underwater Search Strategy

The rationale for this project is to locate aircraft that have splashed-down in the ocean. Given the cost of today's military aircraft, and public concern surrounding the safety of commercial airlines, salvage operations are necessary in the wake of all aircraft crashes. Impact sites can be pinpointed using GPS technology, but the tracking device does not work well underwater. The aircraft can move a large distance from the initial impact area due to the momentum of the aircraft as it hits the water, the submarine dynamics of the vehicle and underwater currents. Aircraft are located via onboard pingers, which give off periodic sound signals once the vehicle hits the water. These typically last up to 30 days and can have a range of up to 1km. Most of the ocean floor lies between 2km and 6km depth, with the deepest trenches reaching some 10km. This means that the search must be performed underwater in order to detect the pinger signals. Most search and rescue operations today are performed by tethered unmanned vehicles. The robots relay visual and sound information back up to the surface via the cable. The ship at the surface uses this information to navigate through the search area.

Since the ship will perform an exhaustive search, this method can be rather time consuming. To minimize the search time, we propose to to use a team of UUV's, able to communicate amongst themselves any signals received from the pinger and to collaboratively locate the aircraft without aid or interference from the mother ship. The underwater robot positions may be mapped using floating a positioning device and transmitted to a screen on board the ship. When the robots have converged, the robots will float to the surface and a retrieval vessel can begin operation.

The cooperative algorithm will be presented in chapter II, followed by the adaptations that were necessary to apply it to this system (chapter III). Much of the work done on the Hurtado algorithm involved point masses; we wanted to extend the simulations to include the dynamics of the UUV. Nekton's "Ranger" UUV's are fully capable of navigation, but are protected via a patent. It was therefore necessary to investigate strategies ourselves. Chapter IV presents the equations of motion for the UUV and chapters V through VII examine two main methods of path planning: optimal control, using variational calculus methods and dynamic programming. A summary of the work covered and conclusions will be drawn in chapter VIII.

CHAPTER II

A COOPERATIVE ALGORITHM

This chapter will present the optimization method used by the robots to locate the aircraft. The cooperative algorithm was developed by John E. Hurtado et al. [4] at Sandia National Laboratories for a two dimensional system. It is based on a quasi-Newtonian method, however the cooperative algorithm ensures all individuals converge on the target, whereas the quasi-Newtonian method uses only one individual [5]. The cooperative algorithm was extended to three dimensions for the aircraft search problem and was tested for robustness to several types of error pertinent to this application: noise in the pinger readings, uncertainty in position updates given to the robots, and navigational error. The latter would occur if the robots had the tendency to veer one way or another due to a hardware flaw or because of underwater currents. The results will be summarized at the end of the chapter.

A. The 2D Algorithm

The robots are initially scattered randomly over the search space. Each robot then takes a reading of the function to be optimized (e.g. heat, light or sound), and measures its position relative to the other robots in the group. In order to determine the position update for robot 'i', a quadratic function is formed thus:

$$\begin{aligned} f(x_i, y_i) &= a_1 + a_2(x_i - x) + a_3(y_i - y) + a_4\frac{1}{2}(x_i - x)^2 \\ &\quad + a_5(x_i - x)(y_i - y) + a_6\frac{1}{2}(y_i - y)^2 \end{aligned} \tag{2.1}$$

where (x, y) is the position of the robot of interest, (x_i, y_i) is the positions of robot (i), $f(x_i, y_i)$ is the sensor readings of robot (i) and a_i are unknown constants.

Expanding equation (2.1)

$$\begin{bmatrix} 1 & (x_1-x_i) & (y_1-y_i) & \frac{1}{2}(x_1-x_i)^2 & (x_1-x_i)(y_1-y_i) & \frac{1}{2}(y_1-y_i)^2 \\ 1 & (x_2-x_i) & (y_2-y_i) & \frac{1}{2}(x_2-x_i)^2 & (x_2-x_i)(y_2-y_i) & \frac{1}{2}(y_2-y_i)^2 \\ 1 & (x_3-x_i) & (y_3-y_i) & \frac{1}{2}(x_3-x_i)^2 & (x_3-x_i)(y_3-y_i) & \frac{1}{2}(y_3-y_i)^2 \\ 1 & (x_4-x_i) & (y_4-y_i) & \frac{1}{2}(x_4-x_i)^2 & (x_4-x_i)(y_4-y_i) & \frac{1}{2}(y_4-y_i)^2 \\ 1 & (x_5-x_i) & (y_5-y_i) & \frac{1}{2}(x_5-x_i)^2 & (x_5-x_i)(y_5-y_i) & \frac{1}{2}(y_5-y_i)^2 \\ 1 & (x_6-x_i) & (y_6-y_i) & \frac{1}{2}(x_6-x_i)^2 & (x_6-x_i)(y_6-y_i) & \frac{1}{2}(y_6-y_i)^2) \end{bmatrix} \begin{pmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \\ a_6 \end{pmatrix} = \begin{pmatrix} f(x_1,y_1,z_1) \\ f(x_2,y_2,z_2) \\ f(x_3,y_3,z_3) \\ f(x_4,y_4,z_4) \\ f(x_5,y_5,z_5) \\ f(x_6,y_6,z_6) \end{pmatrix} \tag{2.2}$$

This time (x_i, y_i, z_i) describes the position of the robot which we want to move. We can write equation 2.2 more compactly as:

$$[\boldsymbol{D}]\{\boldsymbol{a}\} = \{\boldsymbol{f}\} \tag{2.3}$$

Since the positions of all the robots and their sensor readings are known, we need to solve for the unknown coefficients a_i.

$$\{\boldsymbol{a}\} = [\boldsymbol{D}]^{-1}\{\boldsymbol{f}\} \tag{2.4}$$

There are 6 unknown coefficients for the quadratic function, so we require a minimum of 6 robots in the system. If there are more than 6 robots, the least squares algorithm is applied, see (2.5a) below:

$$\begin{aligned}[\boldsymbol{D}]\{\boldsymbol{a}\} &= \{\boldsymbol{f}\} && (2.5a)\\ ([\boldsymbol{D}]^T[\boldsymbol{D}])\{\boldsymbol{a}\} &= [\boldsymbol{D}]^T\{\boldsymbol{f}\} && (2.5b)\\ \{\boldsymbol{a}\} &= ([\boldsymbol{D}]^T[\boldsymbol{D}])^{-1}[\boldsymbol{D}]^T\{\boldsymbol{f}\}. && (2.5c)\end{aligned}$$

The robot computes his new position using the following update rule:

$$\{\boldsymbol{x}_{k+1}\} = \{\boldsymbol{x}_k\} - [\boldsymbol{H}]^{-1}\{\boldsymbol{g}\} \tag{2.6}$$

where $\{\boldsymbol{x}_{k+1}\}$ is a vector of the new position,

$\{\boldsymbol{x}_k\}$ is a vector of the current position,

$[\boldsymbol{H}] = \begin{bmatrix} a_4 & a_5 \\ a_5 & a_6 \end{bmatrix}$ is the Hessian of the quadratic approximation and

$\{\boldsymbol{g}\} = \begin{pmatrix} a_2 \\ a_3 \end{pmatrix}$ is the gradient of the quadratic approximation.

Now if the Hessian is positive definite, the position update is guaranteed to be in the direction of negative gradient. This is a highly desirable characteristic if we want to reach the minimum of the quadratic function. Equally, if the Hessian is negative definite, the position update is guaranteed to be in the direction of positive gradient. Thus we are guaranteed to reach any maximum. Problems arise if the Hessian is indefinite or singular, since the position update requires an infinitely large step. A simple solution to this problem is to force the Hessian to be positive definite by reversing the sign of any non-positive eigenvalues. The Hessian is then reconstructed using spectral decomposition. This is shown in the pseudo code below.

```
$tol = 0.01;$
$if min(eig(\boldsymbol{H})) < tol$
    $[v, d] = eig(\boldsymbol{H});$
    $\boldsymbol{H} = v * abs(d) * v;$
$end$
```

Since we are interested in the minimum of the function, this reconstruction guarantees we find the solution. Once H has been corrected accordingly, we move robot 'i' and go through the same procedure with the next robot.

B. Converting To 3D

When considering 3D systems we need to expand (2.1) thus:

$$\begin{aligned} f(x_i, y_i, z_i) &= a_1 + a_2(x - x_i) + a_3(y - y_i) + a_4(z - z_i) + a_5\frac{1}{2}(x - x_i)^2 + \\ &\quad a_6(x - x_i)(y - y_i) + a_7(x - x_i)(z - z_i) + a_8\frac{1}{2}(y - y_i)^2 + \\ &\quad a_9(y - yi)(z - z_i) + a_{10}\frac{1}{2}(z - z_i)^2. \end{aligned} \tag{2.7}$$

Note this time we have 10 unknown coefficients so we require a system of at least 10 robots in order to solve for them.

$$\{\boldsymbol{a}\} = [\boldsymbol{D}]^{-1}\{\boldsymbol{f}\}$$

[**D**] is an nx10 matrix, **a** is a 10x1 vector, **F** is nx1. The position update is again calculated using the update rule:

$$\{\boldsymbol{x}_{k+1}\} = \{\boldsymbol{x}_k\} - [\boldsymbol{H}]^{-1}\{\boldsymbol{g}\}$$

But this time $$[\boldsymbol{H}] = \begin{bmatrix} a_5 & a_6 & a_7 \\ a_6 & a_8 & a_9 \\ a_7 & a_9 & a_{10} \end{bmatrix} \quad \{\boldsymbol{g}\} = \begin{pmatrix} a_2 \\ a_3 \\ a_4 \end{pmatrix}$$

We apply the same technique as before to maintain a positive definite Hessian matrix.

C. System Robustness

A number of studies were undertaken to examine the robustness of the system to noise. Three main sources of error were identified:

- Noise in the sensor reading - distorts readings and therefore the approximation of the objective function.
- Error in the position updates - A robot may think it is at position x but is in fact at $x + \Delta x$. One position error will affect the update positions of all the robots since the readings will be misplaced, which in turn, will distort the quadratic approximation of the objective function.
- Navigational/heading error - Due to hardware fault or underwater currents, the robot may not arrive at the update position exactly. If the system can identify the position of each robot exactly, the heading error is harmless. The robot will be correctly located at the next update.

  As will be seen, the algorithm is robust to all the errors. Position error is the most disruptive of the three.

All simulations were performed in a pool $50\times50\times8$ with a team of 10 robots. Each of the simulations was run for 50s starting in a ring of radius 15m around the source. The light source is assumed to be a fake plume i.e. the light dissipates according to the relation $1/r^2$. Two starting configurations were considered. First, the robots were all placed in random positions up to a meter around the point $(25, 25, 6)$ as shown in Fig. 1. This will be called the "corner" configuration. Secondly, the robots were placed in a ring 15m radius from the source, see Fig. 2, which will be named the "surround" configuration. For both configurations, the effect of each error will be considered individually followed by the effect of different combinations of errors.

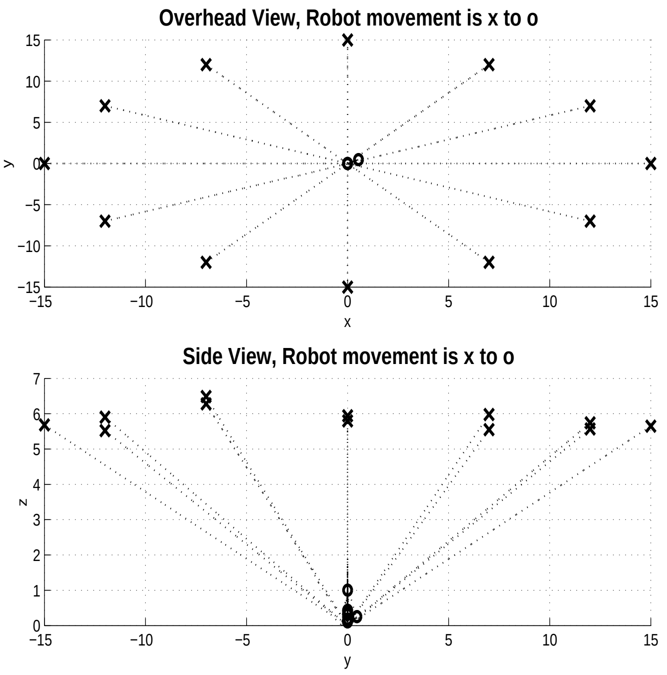

Fig. 1. Surround configuration

1. Corner Configuration

The corner configuration simulates a clustered initial configuration. The robots cannot move more than 1m at each update and the simulation terminates when all 12 robots are within 5m radius of the light source, which is placed at $(0, 0, -6)$. For each case presented below, 20 simulations were run and the average number of iterations

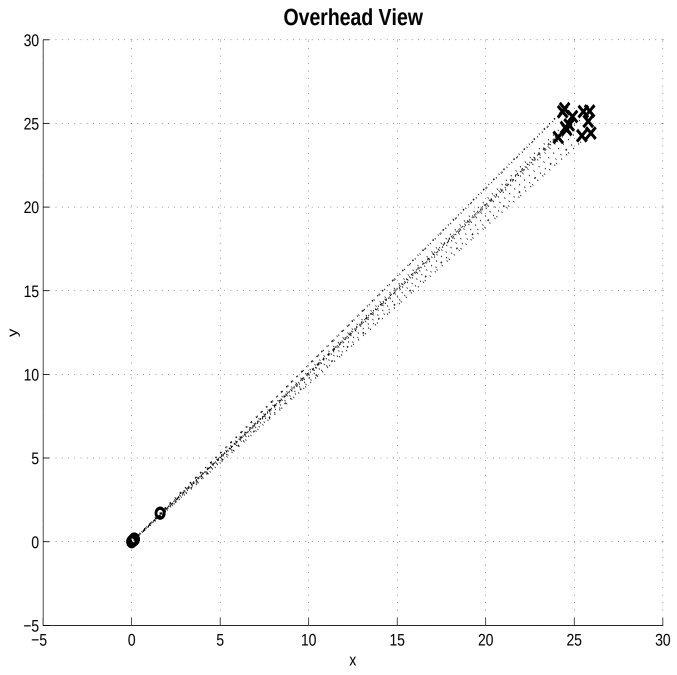

Fig. 2. Corner configuration

recorded. This value will be used to determine the performance of the system.

a. Single Errors

Table I and Fig. 3 show the effects of sensor noise; Table II and Fig. 4 show the effects of navigational error, Table III and Fig. 5 show the effects of position error.

Table I. Effects of sensor noise for corner configuration

Sensor Noise	Average No. updates
$\pm1\%$	47
$\pm3\%$	50
$\pm5\%$	52
$\pm10\%$	59

Table II. Effects of heading error for corner configuration

Heading Error	Average No. updates
5°	42
10°	43
15°	45
20°	49
25°	53
30°	56

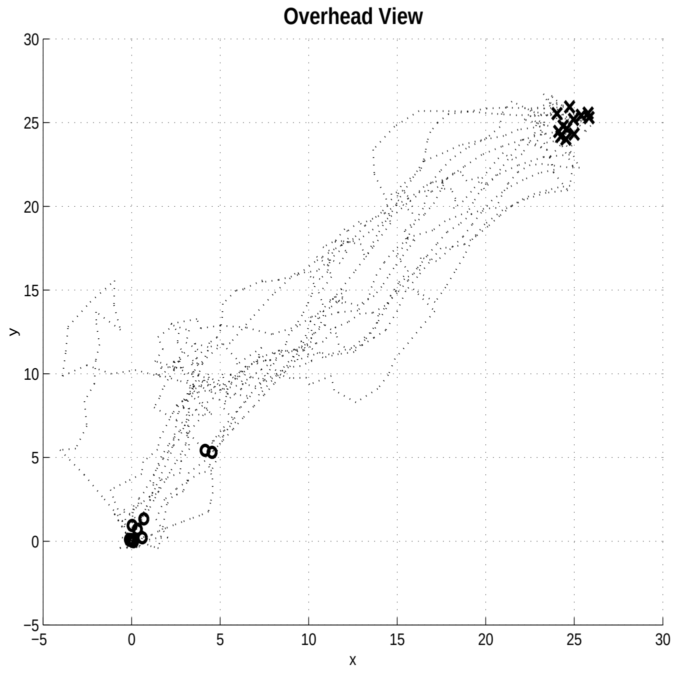

Fig. 3. 10% sensor noise

Table III. Effects of position error for corner configuration

Position Error	Average No. updates
$\pm 0.2m$	47
$\pm 0.6m$	52
$\pm 1.0m$	56
$\pm 1.5m$	73
$\pm 2.0m$	129

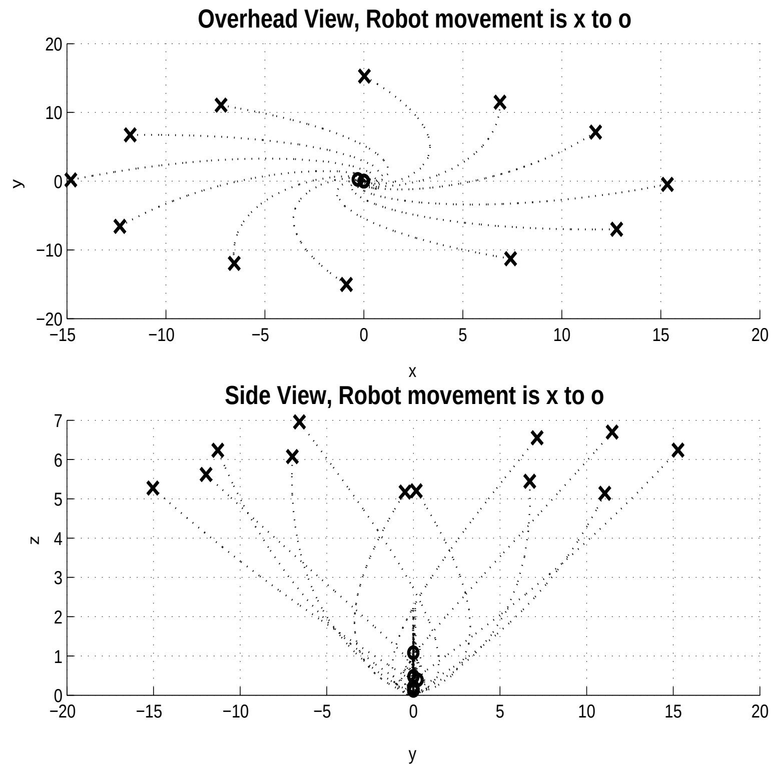

Fig. 4. 30^{o}m heading error

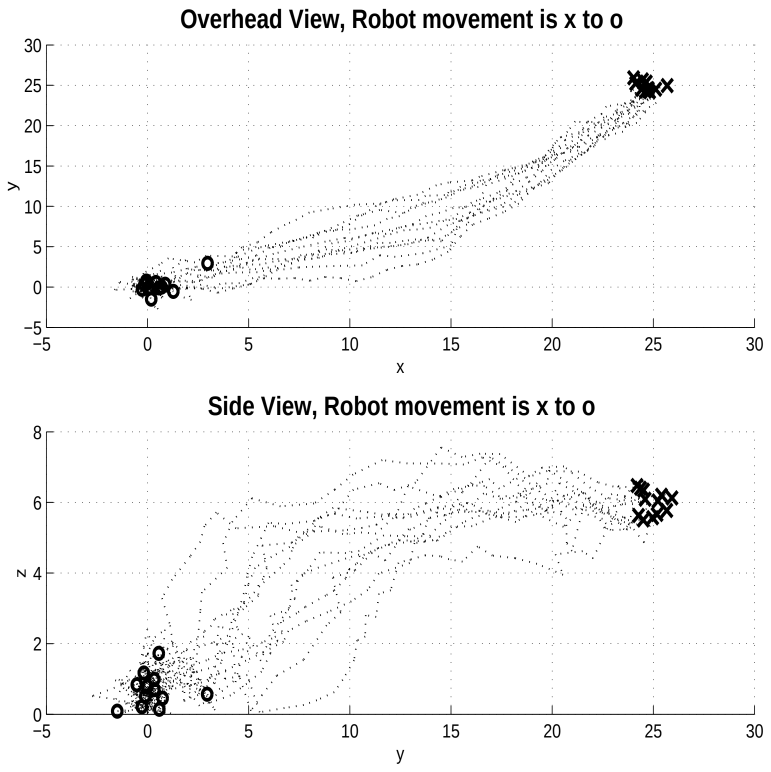

Fig. 5. Effects of position error

With all three errors, the number of updates needed for convergence increased as the error increased. The position error caused the longest delay, especially as the error exceeded the maximum update length (1m). If we consider 1m to be the maximum position error, all three cases produce similar results.

b. Combining 2 Errors

It is unlikely that there will be just one error present. Table IV investigates effects of different combinations of sensor noise and heading error, Table V presents combinations of position and heading error and Table VI presents combinations of sensor noise and position error.

Table IV. Effects of combined sensor noise and heading error

Sensor Noise	±1%	±3%	±5%	±10%
Heading Error				
10°	52	54	55	65
20°	57	57	61	66
30°	63	65	69	75

As a rule, the greater the total error, the slower the convergence. It appears that the combination of sensor noise with any other disturbance causes the greatest delay. This is most clearly seen in Table VI. The sensor noise also causes a large variation about the mean number of updates. This is also noticeable in the noise/position error Table VI, particularly when the position error is larger than the maximum step i.e. for errors above 1m.

Table V. Effects of combined heading and position error

Position Error Heading Error	$\pm 0.2m$	$\pm 0.6m$	$\pm 1.0m$	$\pm 1.5m$	$\pm 2.0m$
10°	43	46	46	52	53
20°	48	48	51	59	63
30°	55	58	59	64	67

Table VI. Effects of combined sensor noise and position error

Position Error Sensor noise	$\pm 0.2m$	$\pm 0.6m$	$\pm 1.0m$	$\pm 1.5m$	$\pm 2.0m$
±1%	46	52	60	75	142
±3%	50	55	59	79	112
±5%	51	57	62	79	147
±10%	60	59	65	77	138

c. Combining All Errors

Finally all the errors were combined to create the worst-case scenario. For this section, I have added the standard deviations from the mean number of iterations-to-convergence in order to demonstrate how noise error introduces uncertainty in time to convergence. The tables show position error vs heading error for different amounts of noise in the sensor readings.

Table VII. Effects of all errors: 1% sensor noise

Position Error	$\pm0.2m$		$\pm0.6m$		$\pm1.0m$		$\pm1.5m$		$\pm2.0m$	
Heading Error	mean	std	mean	std	mean	std	mean	std	mean	std
10°	47	5.3	48	5.3	49	2.8	49	6.3	48	5.7
20°	54	7.6	53	6.0	54	7.1	54	6.1	56	5.8
30°	60	4.1	59	7.2	63	8.1	63	9.2	61	6.1

Table VIII. Effects of all errors: 3% sensor noise

Position Error	$\pm0.2m$		$\pm0.6m$		$\pm1.0m$		$\pm1.5m$		$\pm2.0m$	
Heading Error	mean	std	mean	std	mean	std	mean	std	mean	std
10°	53	5.3	53	6.9	52	3.4	51	7.8	54	8.9
20°	58	11.7	58	8.2	60	9.2	60	8.0	54	6.7
30°	66	8.6	64	7.9	64	7.1	64	10.4	61	6.8

As we scan Tables VII through X, we note that there is no correlation between the increase in position error and the number of updates. The result is surprising

Table IX. Effects of all errors: 5% sensor noise

Position Error	$\pm 0.2m$		$\pm 0.6m$		$\pm 1.0m$		$\pm 1.5m$		$\pm 2.0m$	
Heading Error	mean	std	mean	std	mean	std	mean	std	mean	std
10°	57	12.9	55	9.6	56	8.9	55	9.1	54	11.6
20°	60	7.9	58	7.6	62	10.4	61	8.1	59	6.9
30°	61	7.8	66	14.4	64	10.5	69	12.3	65	8.8

Table X. Effects of all errors: 10% sensor noise

Position Error	$\pm 0.2m$		$\pm 0.6m$		$\pm 1.0m$		$\pm 1.5m$		$\pm 2.0m$	
Heading Error	mean	std	mean	std	mean	std	mean	std	mean	std
10°	63	12.1	61	10.6	64	16	64	17.5	62	10.1
20°	72	11.6	64	9.7	71	17	68	10.9	64	11.5
30°	73	13	76	19.7	66	10.3	68	10.7	77	14.6

even though the relationship between heading and position error was weaker than the other two combinations. Even with maximum sensor noise see Table X, the mean number of updates is significantly lower than previously seen with just sensor noise and position error. The mean number of updates increases both with heading error and with sensor noise.

The standard deviation does indeed increase as the sensor noise goes up, however, there is no correlation with either the position error or the heading error.

2. Surround Configuration

The surround configuration simulates a more scattered initial configuration. Convergence is generally faster using this set-up and the system is more robust to noise. This is because the robots can capture more of the objective function and so produce a more accurate approximation. As before, the robots cannot move more than 1m at each update and the simulation terminates when all 12 robots are within 5m radius of the light source. The light source is placed at $(0, 0, -6)$. For each case presented below, 20 simulations were run and the average number of iterations recorded. This value will be used to determine the performance of the system.

a. Single Errors

The effect of each error was considered separately. Tables XI to XIII show that the average number of updates is significantly lower for the surround configuration than for the corner configuration. In each case, an increase in error still causes an increase in the number of required updates, but it is much less noticeable than in the previous simulations.

Table XI. Effects of sensor noise for corner configuration

Sensor Noise	Average No.updates
$\pm 1\%$	19
$\pm 3\%$	21
$\pm 5\%$	23
$\pm 10\%$	23

Table XII. Effects of position error for corner configuration

Position Error	Average No.updates
$\pm 0.2m$	18
$\pm 0.6m$	21
$\pm 1.0m$	25
$\pm 1.5m$	35
$\pm 2.0m$	84

Table XIII. Effects of heading error for corner configuration

Heading Error	Average No.updates
5°	16
10°	16
15°	17
20°	17
25°	18
30°	18

The heading error has very little effect on the system. The robots generally converge faster than with the other two errors and there is little variation in number of updates as the error is increased. The position error, again, has a more pronounced effect than the heading error and the sensor noise for single errors.

b. Two Errors Combined

The errors were combined in the same way as before. Table XIV demonstrates the effects of heading and position error, Table XV the effects of sensor noise and heading error and Table XVI the effects sensor noise and heading error.

The relationship between the error and the mean number of updates is very weak for the cases involving heading error. This is extremely surprising in the case of heading and positioning error as it would seem that the position error would aggravate the effects of the drift. As with the corner configuration, the position and sensor noise combination produced the most dramatic effects.

Table XIV. Effects of combined heading and position error

Position Error Heading Error	$\pm 0.2m$	$\pm 0.6m$	$\pm 1.0m$	$\pm 1.5m$	$\pm 2.0m$
10°	17	17	17	17	17
20°	18	17	18	17	18
30°	19	18	18	18	19

Table XV. Effects of combined sensor noise and heading error

Sensor Noise Heading Error	±1%	±3%	±5%	±10%
10°	18	19	20	21
20°	18	19	21	23
30°	21	21	22	23

Table XVI. Effects of combined sensor noise and position error

Position Error Sensor noise	$\pm 0.2m$	$\pm 0.6m$	$\pm 1.0m$	$\pm 1.5m$	$\pm 2.0m$
±1%	17	21	26	36	82
±3%	18	20	25	35	104
±5%	20	22	25	36	115
±10%	21	22	27	39	110

c. Combining All Errors

Again, all the errors were combined to create the worst-case scenario. Fig.s XVII through XVIII show the effects of position and heading error for a certain amount of noise. The standard deviations from the mean number of iterations-to-convergence features in order to show the noise introducing uncertainty in time to convergence.

Table XVII. Effects of all errors: 1% sensor noise

Position Error	$\pm 0.2m$		$\pm 0.6m$		$\pm 1.0m$		$\pm 1.5m$		$\pm 2.0m$	
Heading Error	mean	std	mean	std	mean	std	mean	std	mean	std
10°	19	2.8	18	2.8	17	1.7	18	2.6	18	2.6
20°	19	2.1	18	2.3	18	3.1	19	2.4	18	2.4
30°	20	2.9	20	2.6	20	2.8	19	2.6	19	2.8

Table XVIII. Effects of all errors: 3% sensor noise

Position Error	$\pm 0.2m$		$\pm 0.6m$		$\pm 1.0m$		$\pm 1.5m$		$\pm 2.0m$	
Heading Error	mean	std	mean	std	mean	std	mean	std	mean	std
10°	19	2.5	19	2.8	19	2.7	20	2.7	19	2.7
20°	19	2.5	20	2.9	18	2.7	19	2.6	20	3.2
30°	21	2.8	22	2.7	20	2.8	20	2.6	21	3.2

The effects of the combined system on the surround configuration are similar to that on the corner configuration. They are generally less noticeable due to the wide spread of the robots in this configuration and the rapid convergence characteristics

Table XIX. Effects of all errors: 5% sensor noise

Position Error	$\pm 0.2m$		$\pm 0.6m$		$\pm 1.0m$		$\pm 1.5m$		$\pm 2.0m$	
Heading Error	mean	std	mean	std	mean	std	mean	std	mean	std
10°	20	3.3	20	3.8	19	2.7	19	3.4	21	2.7
20°	19	3.2	20	2.3	21	3.2	20	3.1	21	2.4
30°	21	2.8	21	2.0	21	3.0	21	3.0	21	3.0

Table XX. Effects of all errors: 10% sensor noise

Position Error	$\pm 0.2m$		$\pm 0.6m$		$\pm 1.0m$		$\pm 1.5m$		$\pm 2.0m$	
Heading Error	mean	std	mean	std	mean	std	mean	std	mean	std
10°	22	5.1	23	4.0	21	2.9	23	3.9	21	3.8
20°	22	3.5	23	6.5	22	4.4	21	4.3	21	4.4
30°	23	3.3	24	4.4	23	3.5	23	3.2	23	3.4

which arise from it. To summarize:

- The mean update is dependent on the sensor noise and heading error but not position noise.
- The mean update is kept below 25. A significant improvement on the combination of sensor noise and position error.
- The variation about the mean increases as the sensor noise increases.

Generally, the corner configuration is more robust than the surround configuration. The larger disparity between readings taken by each robot means that gradient information is less likely to be perturbed by noise and/or errors. It also means the robots can quickly find the local minimum. It is therefore recommendable to spread the robot cluster as wide as is feasible within constraints imposed by the search area and the communications system.

CHAPTER III

THE REAL SYSTEM

A. The Robots

The UUV's used to test the system were Nekton's Ranger/UMAP UUV's, shown in Fig. 6. The robots weigh 3.5 Kg, (1kg in the water) and have dimensions: 91 cm long, 9 cm diameter. The robot moves by spinning the propeller at the back of the vehicle. The propeller is shielded from the metal casing seen at the back of the UUV. The sensors are built into the nose of the robot, and are protected by the wire cage shown.

B. Locating the Source

To test the algorithm, a light source was used. All light dissipates according to the relation $L \propto \frac{1}{r^2}$ however the cooperative algorithm models the light using a quadratic approximation; this discrepancy produces singularities. To accommodate the algorithm, the inverse quadratic function is inverted before updating the robot positions.

$$\boldsymbol{G}(x,y,z) = \frac{1}{r^2} \tag{3.1a}$$

$$\boldsymbol{F}(x,y,z) = -\frac{1}{G}. \tag{3.1b}$$

Both functions have a minimum at zero, so the robots will still converge at the correct location.

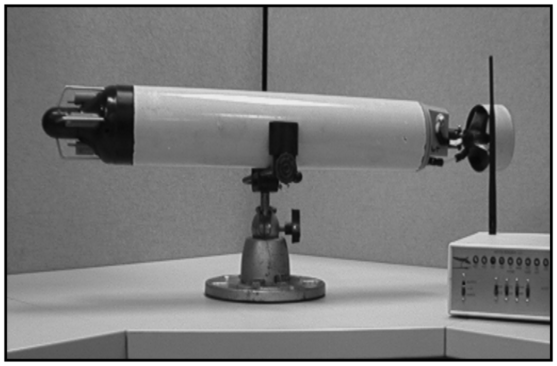

Fig. 6. Nekton's Ranger/Underwater Multi-Agent Platform (UMAP)

C. Sensor Issues

As we have seen, the light emitted by the light source decreases at a rate of $\frac{1}{r^2}$, however, as Fig. 7 shows, the light seen by the robot is actually

$$\begin{aligned} \boldsymbol{F} &= \frac{|\sin\gamma|}{r^2} \qquad (3.2) \\ \gamma &= \arctan\frac{\sqrt{x^2+y^2}}{z} \qquad (3.3) \end{aligned}$$

Fortunately, for small distances from the light source, small angle approximations can be employed to justify using $\frac{1}{r^2}$. The algorithm works well for distances up to 40m.

The measured light also depends on the pitch of the robot relative to the light source. Consider robots 1 and 2 in Fig. 8. Although both robots are in the vicinity of the light source, robot 2 will record a much stronger signal than robot 1. This will make it much harder for the robots to find the source. It was assumed that the robots would travel at a constant depth in order to negate these effects.

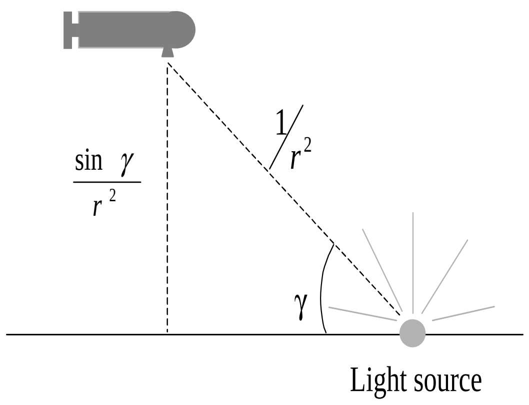

Fig. 7. Robot reading light from light sensor

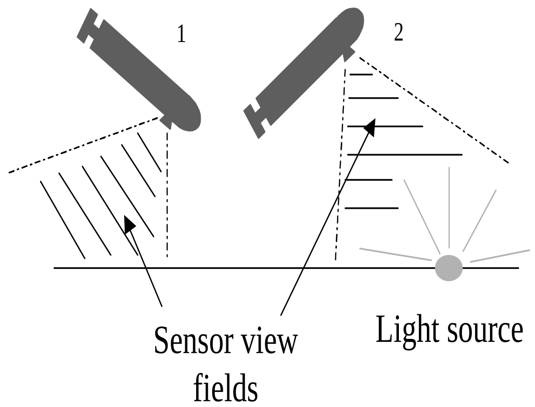

Fig. 8. Sensor reading depends on pitch of robot

D. Exhaustive Search

In the introduction, it was mentioned that the pinger is limited to a 1 km range [6]. While this covers a large area, it is likely that the robots will not hear it initially (see Fig. 9)

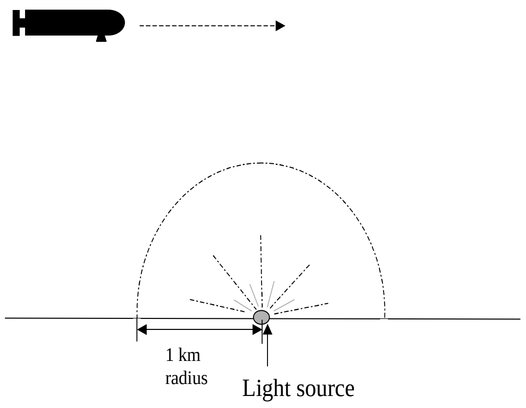

Fig. 9. The robot is out of range and so cannot locate the pinger or use the cooperative algorithm

The search therefore requires two stages. The first involves a simple exhaustive search as depicted in Fig. 10. Once one or more of the robots have a reading, the robots switch to the cooperative algorithm. The manner in which the cooperative algorithm is used is slightly different to the way explained before because the robots do not have real time information about the readings of the other robots or their locations. The next section will explain it more fully.

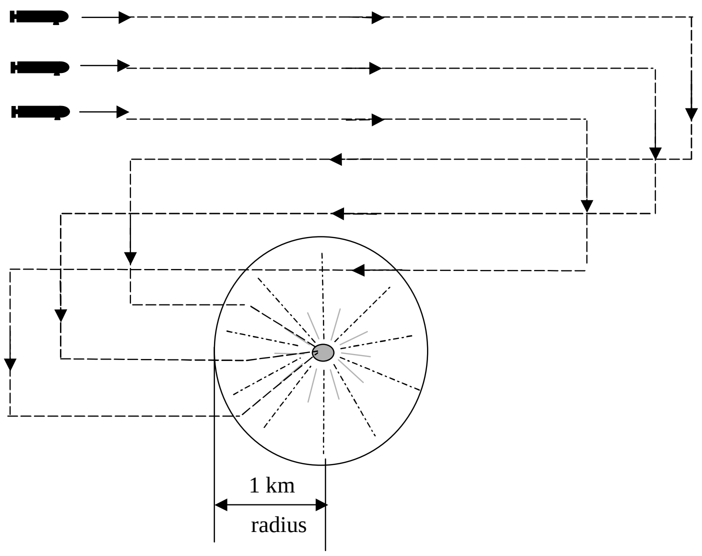

Fig. 10. Exhaustive search configuration. The Rangers are dropped to a suitable depth and then made to swim in straight lines until they locate the pinger

E. Communications

The communications strategy is based on a seven second round robin method. The protocol is shown graphically in Fig. 11

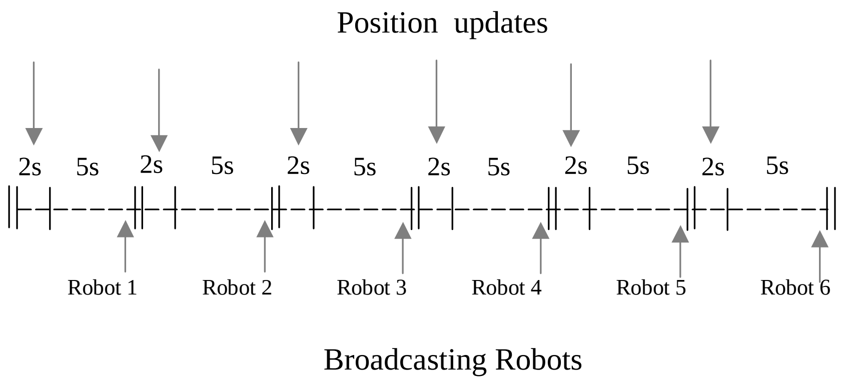

Fig. 11. Round robin communications for AUV team

During the first 2 seconds all the robots receive their own current positions from an external positioning device. For the next 5 seconds, the robots each collect light readings at 5Hz frequency. Any positive data readings are stored in memory along with the position at which the data was found. The position is calculated using dead-reckoning (assuming travel is in a straight line at constant speed). At the end of the 7 second cycle, one of the robots broadcasts any useful information to the rest of the robots. The other robots check incoming data for positive light data and store it in memory if it appears.

Each of the robots can broadcast, but only in turn. Once the last robot has broadcast his information, robot 1 is allowed to broadcast again. This means that any robot can only broadcast every 7 N seconds, for an N robot system. This method is certainly the simplest and most robust form of communication, however it does incur considerable delays. A looser protocol may be implemented to give the system a faster response time, but this is beyond the scope of the project.

With the system implemented, broadcasting robots are restricted to sending 4 pieces of data. These are typically the 4 best sensor readings in memory. Each robot runs the cooperative algorithm on board. However, instead of obtaining its information from the 10 robots in real time, it uses 10 data points stored in memory. These maybe all from the same robot provided the readings are sufficiently different for the cooperative algorithm to solve the simultaneous equations. Thus, a robot swimming straight through the light source can pick up 10 readings in 2 seconds and switch from exhaustive search to cooperative search immediately. The robots that do not pass through the light source or that pass through the periphery of the source rely on information communicated by the other robots in order to obtain the 10+ data points needed to run the algorithm. Since any robot can only broadcast 4 pieces of information at a time, the fewer robots passing through the light source, the longer it takes before the other robots can start to converge on the target area.

Fig. 12 through Fig. 14 show the group in action.

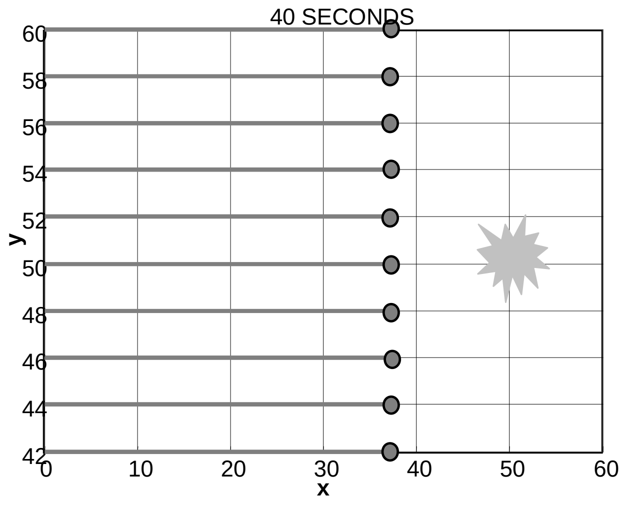

Fig. 12. No robot has detected a signal

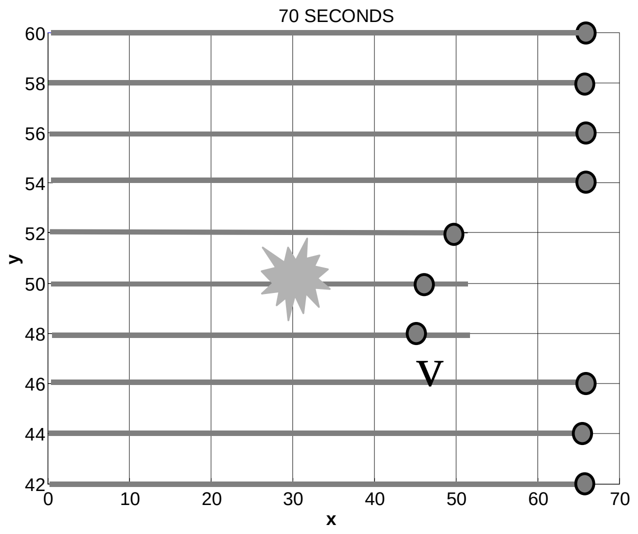

Fig. 13. Robots 4,5 and 6 have each detected sufficient information to switch to cooperative search. The other robots have neither detected nor received data

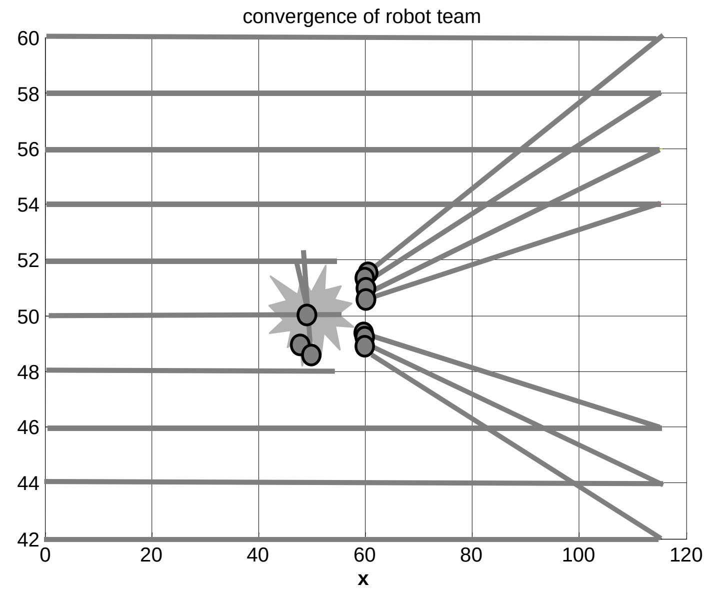

Fig. 14. All robots are converging on target. Robots 4, 5 and 6 use their own data to perform the cooperative search, the others use received data

1. Memory

For simplicity of use, the memory was divided into 3 areas:

- Mylightdata (4x4) - stores best light readings and their positions for broadcasting
- Optdata (20x4) - stores data for cooperative algorithm
- Scratch (4x4) - stores incoming data before sorting occurs

Each memory has 'r' rows of information, which each contain one sensor reading and the (x,y,z) co-ordinates where the reading was taken. During exhaustive search, the robot places any positive light readings in the next available row in Optdata. If there are 10 or more filled rows in Optdata, the robot switches from exhaustive search to cooperative search and uses the update provided by the quadratic method to move. Optdata may hold more than 10 readings, the algorithm increases in effectiveness as the number of points used increases. This has to be weighed against computational time. Once Optdata is full, it will only accept data that improves on data already stored there.

The best 4 light readings taken by the robot are kept in Mylightdata ready for broadcast. When broadcasting, the data in Mylightdata is moved to the Scratch and broadcast to the rest of the robots. All data in Mylightdata is then wiped to avoid broadcasting the same data twice. The receiving robots check the Scratch for positive readings and paste these into the next available row in Optdata. Optdata will not accept any repeat readings.

F. Summary

- In converting the 3D cooperative algorithm to better simulate the robot system, there are issues concerning the type of source used and the positioning of the light sensor on the UUV.
- The Nekton Ranger UUV has been introduced; its equations of motion will be presented in the next section.
- I have explained the need for a two search strategies: first an exhaustive search in order to locate the light or noise and then the cooperative search to locate the source of the light/the pinger.
- Finally, the communications set-up has been laid out and discussed.

CHAPTER IV

EQUATIONS OF MOTION

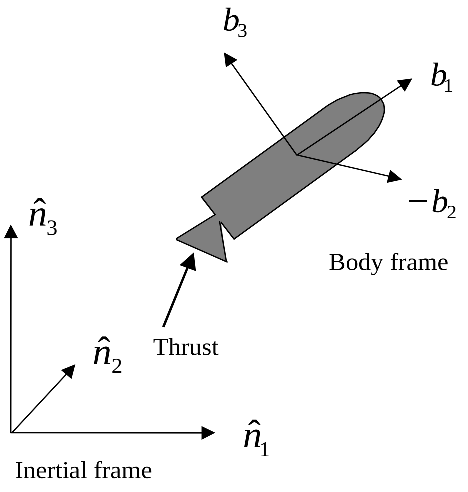

Fig. 15. Robot with inertial and body axes

Consider the robot in Fig. 15. It is described using the states:

$$\boldsymbol{q} = \{x, y, z, \alpha, \beta\}^T \tag{4.1}$$

x,y and z are the positions of the robot in the body frame. α is the heading or yaw angle (positive left) and β is the pitch angle (positive up). The robot is bottom heavy and therefore cannot roll. The total velocity of the vehicle in the inertial axis is

$$\boldsymbol{v}_{cm} = \dot{x}\,\hat{\boldsymbol{n}}_1 + \dot{y}\,\hat{\boldsymbol{n}}_2 + \dot{x}\,\hat{\boldsymbol{n}}_3 \tag{4.2}$$

The cosine matrix which converts between the inertial and body frames is formed by a 3-2 Euler transformation in equation 4.3

$$\{\boldsymbol{b}\} = \begin{bmatrix} \cos\beta & 0 & -\sin\beta \\ 0 & 1 & 0 \\ \sin\beta & 0 & \cos\beta \end{bmatrix} \begin{bmatrix} \cos\alpha & \sin\alpha & 0 \\ -\sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix} \{\boldsymbol{n}\} \tag{4.3}$$

$$\{\boldsymbol{b}\} = \begin{bmatrix} \cos\beta\cos\alpha & \cos\beta\sin\alpha & -\sin\beta \\ -\sin\alpha & \cos\alpha & 0 \\ \sin\beta\cos\alpha & \sin\beta\sin\alpha & \cos\beta \end{bmatrix} \{\boldsymbol{n}\} \tag{4.4}$$

$$\{\boldsymbol{n}\} = \begin{bmatrix} \cos\beta\cos\alpha & -\sin\alpha & \sin\beta\cos\alpha \\ \cos\beta\sin\alpha & \cos\alpha & \sin\beta\sin\alpha \\ -\sin\beta & 0 & \cos\beta \end{bmatrix} \{\boldsymbol{b}\} \tag{4.5}$$

Using (4.5) the velocity in the inertial frame is

$$\begin{aligned} \boldsymbol{v}_{cm} &= \dot{x}\left[(\cos\beta\cos\alpha)\,\hat{\boldsymbol{b}}_1 - (\sin\alpha)\,\hat{\boldsymbol{b}}_2 + (\sin\beta\cos\alpha)\,\hat{\boldsymbol{b}}_3\right] \\ &\quad + \dot{y}\left[(\cos\beta\sin\alpha\,)\hat{\boldsymbol{b}}_1 + (\cos\alpha)\,\hat{\boldsymbol{b}}_2 + (\sin\beta\sin\alpha)\,\hat{\boldsymbol{b}}_3\right] \\ &\quad + \dot{z}\left[(-\sin\beta)\,\hat{\boldsymbol{b}}_1 + (\cos\beta)\,\hat{\boldsymbol{b}}_3\right] \end{aligned} \tag{4.6}$$

Collecting terms

$$\begin{aligned} \boldsymbol{v}_{cm} &= (\dot{x}\cos\beta\cos\alpha + \dot{y}\cos\beta\sin\alpha - \dot{z}\sin\beta)\,\hat{\boldsymbol{b}}_1 \\ &\quad + (-\dot{x}\sin\alpha + \dot{y}\cos\alpha)\,\hat{\boldsymbol{b}}_2 \\ &\quad + (\dot{x}\sin\beta\cos\alpha + \dot{y}\sin\beta\sin\alpha + \dot{z}\cos\beta)\,\hat{\boldsymbol{b}}_3 \end{aligned} \tag{4.7}$$

A. Constraints

The position of the propeller at the tail of the UAV means that all motion will be through the nose, i.e. along the $\hat{\boldsymbol{b}}_1$ axis. All motion in the other axes is zero. We therefore have *nonholonomic* constraints:

$$0 = -\dot{x}\sin\alpha + \dot{y}\cos\alpha \tag{4.8a}$$

$$0 = \dot{x}\sin\beta\cos\alpha + \dot{y}\sin\beta\sin\alpha + \dot{z}\cos\beta \tag{4.8b}$$

Nonholonomic constraints are constraints expressed in terms of velocities (Pfaffian form), which cannot be integrated to give configuration constraints [7]. *Holonomic* constraints can be written as both configuration and velocity constraints.

B. Generalized Speeds

When writing equations of motion we choose some *generalized coordinates* which will express the system in the simplest form. For instance, expressing a pendulum in terms of its length L and rotation angle θ give cleaner expressions than if (x, y) coordinates were used. *Generalized velocities* are formed by differentiating generalized co-ordinates with respect to time. It is always possible to get back the generalized coordinates by integrating the generalized velocities.

Generalized speeds are formed from non-generalized coordinates. In a conventional (x, y, z) inertial coordinate system they are often angular velocity components or combinations of generalized velocities (e.g. total velocity). Because they do not have a corresponding generalized coordinate, generalized speeds are similar to nonholonomic constraints. In both cases we only have meaningful expressions at the velocity level.

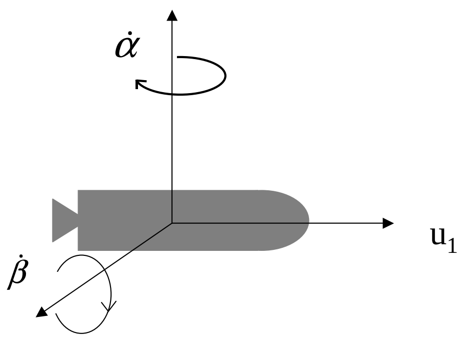

Fig. 16. Partial velocities for AUV

We define 3 generalized speeds for the AUV. These are shown in Fig. 16

$$u_1 = \dot{\alpha} \tag{4.9a}$$

$$u_2 = \dot{\beta} \tag{4.9b}$$

$$u_3 = \dot{x}\cos\beta\cos\alpha + \dot{y}\cos\beta\sin\alpha - \dot{z}\sin\beta \tag{4.9c}$$

Next,we define some *partial velocities.* These can be viewed as new velocity unit vectors positioned in direction of the generalized speeds

$$\frac{\partial \boldsymbol{v}_{cm}}{\partial u_1} = 0 \tag{4.10a}$$

$$\frac{\partial \boldsymbol{v}_{cm}}{\partial u_2} = 0 \tag{4.10b}$$

$$\frac{\partial \boldsymbol{v}_{cm}}{\partial u_1} = \hat{\boldsymbol{b}}_3 \tag{4.10c}$$

The angular velocity of the vehicle is obtained from Fig. 17 and Fig. 18 obtaining

$$\boldsymbol{\omega} = -\dot{\alpha}\sin\beta\,\hat{\boldsymbol{b}}_1 + \dot{\beta}\,\hat{\boldsymbol{b}}_2 + \dot{\alpha}\cos\beta\,\hat{\boldsymbol{b}}_3 \tag{4.11}$$

$$\boldsymbol{\omega} = -u_1\sin\beta\,\hat{\boldsymbol{b}}_1 + u_2\,\hat{\boldsymbol{b}}_2 + u_1\cos\beta\,\hat{\boldsymbol{b}}_3 \tag{4.12}$$

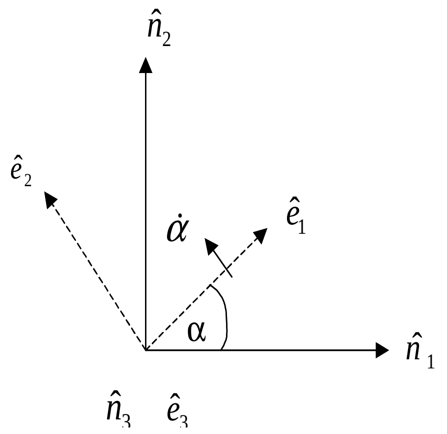

Fig. 17. 3 rotation

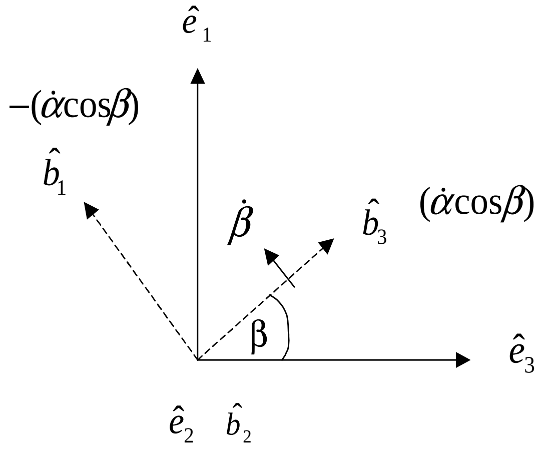

Fig. 18. 2 rotation

And the partial angular velocities. Again, these can be viewed as the directions of the unit vectors in the direction of the generalized speeds.

$$\frac{\partial \boldsymbol{\omega}}{\partial u_1} = -\sin\beta\,\hat{\boldsymbol{b}}_1 + \cos\beta\hat{\boldsymbol{b}}_3 \tag{4.13a}$$

$$\frac{\partial \boldsymbol{\omega}}{\partial u_2} = \hat{\boldsymbol{b}}_2 \tag{4.13b}$$

$$\frac{\partial \boldsymbol{\omega}}{\partial u_3} = 0 \tag{4.13c}$$

The thrust force can be broken down into 3 body axis components

$$\boldsymbol{F} = f_1\,\hat{\boldsymbol{b}}_1 + f_2\,\hat{\boldsymbol{b}}_2 + f_3\,\hat{\boldsymbol{b}}_3 \tag{4.14}$$

where $\boldsymbol{F}$ is the total force and f_i are the force components

The moment about the center of mass is given by

$$\begin{aligned}\boldsymbol{M} &= -L\,\hat{\boldsymbol{b}}_1 \times \boldsymbol{F}\\ &= -L\,\hat{\boldsymbol{b}}_1 \times \left(f_1\,\hat{\boldsymbol{b}}_1 + f_2\,\hat{\boldsymbol{b}}_2 + f_3\,\hat{\boldsymbol{b}}_3\right)\\ &= -f_2L\,\hat{\boldsymbol{b}}_3 + f_3L\,\hat{\boldsymbol{b}}_1\end{aligned} \tag{4.15}$$

To get the forces and moments in the directions denoted by the partial velocities, we take the dot product of the vectors, thus:

$$\frac{\partial \boldsymbol{v}_{cm}}{\partial u_i} \cdot \boldsymbol{F} + \frac{\partial \boldsymbol{\omega}}{\partial u_i} \cdot \boldsymbol{M} \tag{4.16}$$

so

$$\begin{aligned}
\frac{\partial \boldsymbol{v}_{cm}}{\partial u_1} \cdot \boldsymbol{F} + \frac{\partial \boldsymbol{\omega}}{\partial u_1} \cdot \boldsymbol{M} &= 0 \cdot \boldsymbol{F} - f_2 L \cos\beta &\text{(4.17a)}\\
\frac{\partial \boldsymbol{v}_{cm}}{\partial u_2} \cdot \boldsymbol{F} + \frac{\partial \boldsymbol{\omega}}{\partial u_2} \cdot \boldsymbol{M} &= 0 \cdot \boldsymbol{F} + f_3 L &\text{(4.17b)}\\
\frac{\partial \boldsymbol{v}_{cm}}{\partial u_3} \cdot \boldsymbol{F} + \frac{\partial \boldsymbol{\omega}}{\partial u_3} \cdot \boldsymbol{M} &= f_1 + 0 \cdot \boldsymbol{M} &\text{(4.17c)}
\end{aligned}$$

Finally, we want to balance the forces with the kinetics of the robot. From the transport theorem

$$\begin{aligned}
\boldsymbol{a}_{cm} &= \dot{\boldsymbol{v}}_{cm}\\
&= \dot{u}_3 \hat{\boldsymbol{b}}_1 + \boldsymbol{\omega} \times u_3 \hat{\boldsymbol{b}}_1\\
&= \dot{u}_3 \hat{\boldsymbol{b}}_1 + u_1 u_3 \cos\beta \hat{\boldsymbol{b}}_2 - u_2 u_3 \hat{\boldsymbol{b}}_3 &\text{(4.18)}
\end{aligned}$$

The angular momentum of the system is

$$\boldsymbol{H}_{cm} = \begin{bmatrix} I_a & 0 & 0 \\ 0 & I_t & 0 \\ 0 & 0 & I_t \end{bmatrix} \begin{pmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \end{pmatrix} \tag{4.19}$$

Applying the transport theorem again

$$\dot{\boldsymbol{H}}_{cm} = I\dot{\boldsymbol{\omega}} + \boldsymbol{\omega} \times I\boldsymbol{\omega} \tag{4.20}$$

$$\begin{aligned}\dot{\boldsymbol{H}}_{\boldsymbol{cm}} &= \begin{pmatrix} I_a(-\dot{u}_1 \sin\beta - u_1 u_2 \cos\beta) \\ I_t \dot{u}_2 \\ I_t(\dot{u}_1 \cos\beta - u_1 u_2 \sin\beta) \end{pmatrix} \\ &\quad + \begin{pmatrix} 0 \\ (I_a - I_t)(-u_1^2 \sin\beta\cos\beta) \\ (I_t - I_a)(-u_1 u_2 \sin\beta) \end{pmatrix} \qquad (4.21) \\ &= \begin{pmatrix} I_a(-\dot{u}_1 \sin\beta - u_1 u_2 \cos\beta) \\ I_t \dot{u}_2 + (I_a - I_t)(-u_1^2 \sin\beta\cos\beta) \\ I_t(\dot{u}_1 \cos\beta - u_1 u_2 \sin\beta) + (I_t - I_a)(-u_1 u_2 \sin\beta) \end{pmatrix} \qquad (4.22)\end{aligned}$$

Express the left hand side of the kinetic equations of motion in the correct unit directions

$$LHS = \frac{\partial \boldsymbol{v}_{cm}}{\partial u_i} \cdot m\boldsymbol{a}_{cm} + \frac{\partial \boldsymbol{\omega}}{\partial u_i} \cdot \dot{\boldsymbol{H}}_{cm} \qquad (4.23)$$

Expanding

$$\begin{aligned} LHS_1 &= 0 \cdot m\boldsymbol{a}_{cm} + \begin{pmatrix} -\sin\beta & 0 & \cos\beta \end{pmatrix} \cdot \begin{pmatrix} \dot{H}_1 \\ \dot{H}_2 \\ \dot{H}_3 \end{pmatrix} \\ &= \sin\beta\, I_a(-\dot{u}_1 \sin\beta - u_1 u_2 \cos\beta) \\ &\quad + \cos\beta\, I_t(\dot{u}_1 \cos\beta - u_1 u_2 \sin\beta) \\ &\quad + \cos\beta\, (I_t - I_a)(-u_1 u_2 \sin\beta) \\ &= \dot{u}_1\, (I_a \sin^2\beta + I_t \cos^2\beta) \\ &\quad + 2\sin\beta\cos\beta\, u_1\, u_2\, (I_a - I_t) \qquad (4.24) \end{aligned}$$

$$\begin{aligned} LHS_2 &= 0 \cdot m\boldsymbol{a}_{cm} + \hat{\boldsymbol{b}}_2 \cdot \begin{pmatrix} \dot{H}_1 \\ \dot{H}_2 \\ \dot{H}_3 \end{pmatrix} \\ &= I_t \dot{u}_2 + (I_a - I_t)(-u_1^2)\sin\beta\cos\beta \qquad (4.25) \end{aligned}$$

$$\begin{aligned} LHS_3 &= \hat{\boldsymbol{b}}_{\mathbf{1}} \cdot m\boldsymbol{a}_{cm} + 0 \cdot \dot{\boldsymbol{H}}_{cm} \\ &= m\dot{u}_3 \qquad (4.26) \end{aligned}$$

Combining equation 4.22 with equation 4.24 through equation 4.26 we obtain kinetic equations of motion for the UUV

$$\begin{aligned} -f_2 L\cos\beta &= \dot{u}_1 \left(I_a \sin^2\beta + I_t \cos^2\beta\right) \\ &\quad +2\sin\beta\cos\beta\, u_1\, u_2\, (I_a - I_t) \qquad (4.27a) \\ f_3 L &= I_t \dot{u}_2 + (I_a - I_t)(-u_1^2)\sin\beta\cos\beta \qquad (4.27b) \\ f_1 &= m\dot{u}_3 \qquad (4.27c) \end{aligned}$$

OR

$$\begin{aligned} f_1 &= m\dot{u}_3 \qquad (4.28a) \\ f_2 &= \frac{-\dot{u}_1(I_a \sin^2\beta + I_t\cos^2\beta) + 2\sin\beta\cos\beta\, u_1\, u_2\, (I_a - I_t)}{L\cos\beta} \\ f_3 &= \frac{I_t\dot{u}_2 + (I_a - I_t)(-u_1^2)\sin\beta\cos\beta}{L} \qquad (4.28b) \end{aligned}$$

The actual controls: Thrust, vertical deflection, ψ and horizontal deflection ϕ can be obtained by manipulating the force components (see Fig. 19).

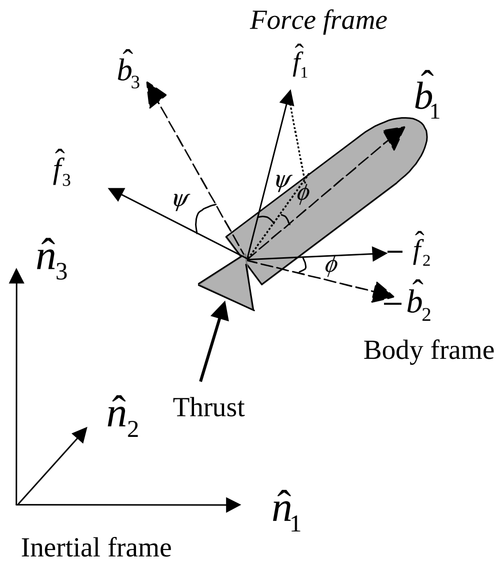

Fig. 19. Converting from body axis to force axis

Note that

$$f_1 = Th\cos\phi\cos\psi \tag{4.29a}$$

$$f_2 = Th\sin\phi\cos\psi \tag{4.29b}$$

$$f_3 = -Th\sin\psi \tag{4.29c}$$

Thus

$$\psi = \arctan\left(\frac{f_2}{f_1}\right) \tag{4.30a}$$

$$Th = \frac{-f_3}{\sin\psi} \tag{4.30b}$$

$$\phi = \arccos\left(\frac{f_2}{f_3}\right) \tag{4.30c}$$

CHAPTER V

OPTIMAL CONTROL

A. Introduction

This chapter gives a detailed introduction to optimal control and its application to the robots used in the cooperative system. Section A deals with the basic principles underlying all optimal control methods. Section B looks at variational methods and the infamous two point boundary value problem. For those familiar with these concepts, the author suggests turning to section C where the techniques mentioned in the first two sections are applied to the robot.

Optimal control is the search for a control input sequence which will deliver the best performance given certain system constraints. The three main measures of performance are:

- control - minimize the fuel consumption or actuator effort
- time - minimize the time needed to perform the task
- states - maximize the distance travelled in a given time

Taking the position updates of the robots as an example. The cooperative algorithm gives us the current and desired positions of each robot. We would like the robots to move between these points minimizing the control effort so as to maximize the search time. For search efficiency, the position update must occur within a certain time, and to avoid too many collisions, we would prefer to use the shortest path.

There are two main types of optimal control: Variational Control, which is based on the calculus of variations, and Dynamic Programming, which discretizes the search area in order to find the most effective path. I applied two search methods to the Variational Control problem and then used Dynamic Programming with Interior Points (DPIP), a method recently developed by Dohrmann and Robinett [8] at Sandia National Laboratories. The methods will be compared and contrasted in the conclusions.

The first step in any optimization problem is to form the performance index, a scalar formed from an expression containing the variables we would like to minimize. To simplify matters, we require all the variables to be greater than zero. An obvious solution to this problem is to square all the variables so that they form quadratic functions. A typical performance index is shown below:

$$J = \frac{1}{2}x(t_f)^T S x(t_f) + \frac{1}{2}\int_{t_0}^{t_f} \left[x(t)^T Q\, x(t) + u(t)^T R\, u(t)\right] dt. \tag{5.1}$$

where $x(t) \in R^n$ is a vector of the states, $x(t_f) \in R^n$ is a vector of the final states, $u(t) \in R^m$ is a vector of the control inputs, S and Q are square, positive semi-definite matrices and R is a square, positive definite matrix. J is a functional, that is to say, a function of a function.

Usually we define a general cost function L(x(t),u(t)), which simplifies the performance index

$$L(x(t), u(t)) = \left[x(t)^T Q x(t) + u(t)^T R u(t)\right] dt \tag{5.2}$$

$$J = \frac{1}{2}\int_{t_0}^{t_f} L(x(t), u(t)) dt \tag{5.3}$$

$L(x(t), u(t))$ is problem dependent and so will change according to the criteria for optimal performance. For minimum time problems,

$$L = t_f \tag{5.4}$$

for minimum fuel problems

$$L = |u_k| \tag{5.5}$$

and for minimum energy problems, we return to the more familiar

$$L(x(t), u(t)) = \left[x(t)^T Q x(t) + u(t)^T R u(t)\right] dt \tag{5.6}$$

In one dimensional optimization, we locate the local extrema by taking the derivative of the function and finding the point where it goes to zero. Recall, the derivative is the slope of the function and that at extrema the slope is zero. More information on the type of extremum can be found by taking the second derivative. If

$\partial^2 J \partial x^2$	> 0	extremum is minimum
	< 0	extremum is maximum
	≥ 0 or ≤ 0	extremum is a saddle-point
	indefinite	we require further derivatives to be able to conclude anything.

In the same way, we can determine the minimum of a functional by taking variations of J with respect to each variable and setting it to zero. The calculus of variations will be discussed in the next section.

The problem constraints may be included in the minimization procedure by augmenting the performance index.

$$J_a = \Phi(x(t_f), t_f) + \frac{1}{2}\int_{t_0}^{t_f} \left(L(x(t), u(t)) + \lambda^T \left[f(x(t), u(t), t) - \dot{x}(t)\right]\right) dt. \tag{5.7}$$

where $f(x(t), u(t), t)$ represents the time-varying system constraints (typically the system equations of motion) and λ is a vector of Lagrange multipliers. We apply one Lagrange multiplier to each constraint. These allow us to decouple the states and the controls, unfortunately since the Lagrange multipliers are unknown, they add complication. Φ represents states we want minimized at the final time.

A useful tool in optimal control is the Hamiltonian function.

$$H = L(x(t), u(t)) + \lambda^T f(x(t), u(t)) \tag{5.8}$$

Note it is similar to the integral term in equation (5.7) but there are no derivative terms ($\dot{x}$). Putting this back in to the equation for the performance index, we arrive at

$$J_a = \Phi(x(t_f), t_f) + \frac{1}{2}\int_{t_0}^{t_f} \left(H(x(t), u(t)) - \lambda^T \dot{x}(t)\right) dt. \tag{5.9}$$

B. Variational Control

In the last section, we showed that the first derivative of a function would indicate the presence of local extrema. We would like to apply the same method to the performance index. Now, $J = J(x(t), u(t), t)$ and $x(t), u(t)$ are continuous functions of t, thus dx and dt are not independent. In mathematics, we define a *variation*

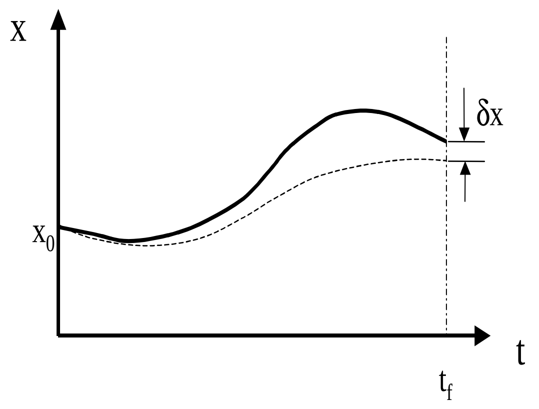

Fig. 20. An optimal path, shown as a thick black line, with an instantaneous variation δx, shown as a dotted line.

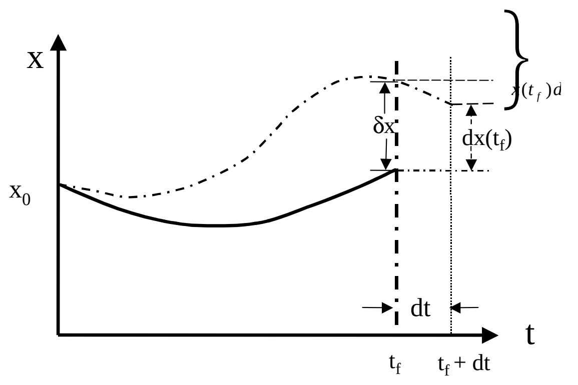

Fig. 21. An optimal path, shown with a time varying variation

which describes a small, instantaneous change in $x(t)$. This is seen in Fig. 20. Fig. 21 demonstrates the more general case. Again, the thick black line is the optimal path and the dotted line is a path deviating from it. Noting that the last section may be approximated by the slope at t_f,

$$dx_f = \delta x(t_f^*) + \dot{x}(t_f)dt_f + O(2) \tag{5.10a}$$

$$dx_f = \delta x(t_f^*) + \left[\dot{x}(t_f^*) + \delta\dot{x}(t_f)\right] dt_f + O(2) \tag{5.10b}$$

$$dx_f = \delta x(t_f^*) + \dot{x}(t_f^*)dt_f + O(2) \tag{5.10c}$$

This is a very useful result, as we will see later.

Now for the most general problem, we need to impose constraints at the final time. These constraints ($\psi(x(t_f), t_f)$) may constitute a certain point, area or trajectory, on which the system *must* terminate. Like their time-varying counterparts, each of these constraints also has a Lagrange multiplier associated with it. These final time constraints should not be confused with the final weighting function $\Phi(x(t_f), t_f)$. $\psi(x(t_f), t_f)$ must be set to zero and is therefore a hard constraint, whereas Φ, on the other hand, need only be minimized.

Consider the case of a space vehicle visiting Mars, the hard constraint is that it arrives on the Martian surface; a soft constraint may be that it land near a particular canyon. The distance from the canyon is something we would like to minimize, but it is not of great importance to the overall success of the trip. On the other hand, if the spacecraft misses the planet altogether, the mission will have been a complete waste of time.

All the constraints considered so far have been equality constraints. Inequality constraints may also be included by expressing them as less than zero

$$\begin{array}{ccc} x \geq 2 & \text{becomes} & 0 \geq 2 - x \\ u < 3x + 6 & \text{becomes} & 0 > u - 3x - 6 \end{array}$$

The inequality constraints are then appended to the performance index with their own set of Lagrange multipliers.

$$\begin{aligned} J_a &= \Phi(x(t_f), t_f) + \nu_E^T \psi_E(x(t_f), t_f) + \nu_I^T \psi_I(x(t_f), t_f) && (5.11) \\ &+ \frac{1}{2} \int_{t_0}^{t_f} \left(H(x(t), u(t)) - \lambda^T \dot{x}(t) \right) dt. && (5.12) \end{aligned}$$

In order to take the variation of the performance index, we will need Leibniz's rule for functionals. For a given functional

$$J(x) = \int_{t_0}^{t_f} h(x(t), t) dt \tag{5.13}$$

$$dJ = h(x(t_f), t_f) dt_f - h(x(t_0), t_0) dt_0 + \int_{t_0}^{t_f} \left[h_x^T(x(t), t)\, \delta x \right] dt \tag{5.14}$$

we define $h_x = \frac{\partial h}{\partial x}$. Thus, applying Liebniz's rule to the performance index and using the same short hand notation

$$\begin{aligned} dJ &= (\Phi_x + \psi_x^T \nu)^T dx|_{t_f} + (\Phi_t + \psi_{t_f}^T \nu) dt|_{t_f} \\ &\quad + \psi^{t_f}|_T d\nu + (H - \lambda^T \dot{x}) dt|_{t_f} - (H - \lambda^T \dot{x}) dt|_{t_0} \\ &\quad + \int_{t_0}^{t_f} \left[H_x^T \delta x + H_u^T \delta u - \lambda^T \delta \dot{x} + (H_\lambda - \dot{x})^T \delta \lambda \right] dt \end{aligned} \tag{5.15}$$

To get rid of the variation of $\dot{x}$, we integrate by parts

$$\int_{t_0}^{t_f} -\lambda^T \delta \dot{x} = -\lambda^T \delta x|_{t_f} + \lambda^T \delta x|_{t_0} + \int_{t_0}^{t_f} \dot{\lambda}^T \delta x \, dt \tag{5.16}$$

From 5.10c

$$\lambda^T \delta x(t_f) = \lambda^T dx|_{t_f} - \lambda^T \dot{x}(t_f) dt|_{t_f} \tag{5.17a}$$

$$\lambda^T \delta x(t_0) = \lambda^T dx|_{t_0} - \lambda^T \dot{x}(t_f) dt|_{t_0} \tag{5.17b}$$

Substitute (5.16) and (5.17b) into (5.15)

$$\begin{aligned} dJ &= (\Phi_x + \psi_x^T \nu - \lambda)^T dx|_{t_f} + (\Phi_t + \psi_t^T \nu + H - \lambda^T \dot{x} + \lambda^T \dot{x}) \, dt|_{t_f} \\ &\quad + \psi^T|_{t_f} d\nu + (H - \lambda^T \dot{x} + \lambda^T \dot{x}) \, dt|_{t_0} + \lambda^T dx|_{t_0} - (H - \lambda^T \dot{x}) \, dt|_{t_0} \\ &\quad + \int_{t_0}^{t_f} \left[(H_x^T + \dot{\lambda}) \delta x + H_u^T \delta u + (H_\lambda - \dot{x})^T \delta \lambda \right] dt \end{aligned} \tag{5.18}$$

At the critical point $dJ = 0$, therefore all terms must either cancel each other out or be set to zero. Consider the terms within the integral, the variations $\delta x, \delta u, \delta \lambda$ are all arbitrary so the necessary conditions at the critical point are:

$$\begin{array}{llll} \dot{x} = \frac{\partial H}{\partial \lambda} & = f(x(t), u(t)) & \geq t_0 & \textit{State Equation} \\ \dot{\lambda} = \frac{\partial H}{\partial x} & = \left(\frac{\partial f}{\partial x}\right)^T \lambda + \frac{\partial L}{\partial x} & t \leq t_f & \textit{Costate Equation} \\ 0 = \frac{\partial H}{\partial u(t)} & = \left(\frac{\partial f}{\partial u}\right)^T \lambda + \frac{\partial L}{\partial u} & & \textit{Stationarity Condition} \end{array}$$

The state equation simply returns the equations of motion. The Costate equation allows us to calculate the Lagrange multipliers. The state and costate equations are coupled differential equations. From the boundary conditions we have an initial value for our states, thus $x(t)$ can be determined from the initial conditions. On the other hand, $\lambda(t_f)$ can be found from the boundary conditions at the final time, so the costate equation is a backwards integration. These equations are difficult to solve because of the need to integrate in different directions. This is the *two point boundary value problem.* The third equation is known as the optimality or stationarity condition. It allows us to determine the control at the critical point.

The boundary conditions result from setting the boundary terms to zero. From (5.18) (ignoring the inequality constraints)

$$x(t_0) \quad specified \tag{5.19a}$$

$$(\phi_x + \psi_x^T \nu - \lambda)^T)|_{t_f} \; dx(t_f) + (\phi_t + \psi_t^T \nu + H)|_{t_f} \; dt_f = 0 \tag{5.19b}$$

The initial time and states are already known so $dt(0) = dx(t_0) = 0$. Since the final states and final time are arbitrary, the terms in the brackets must go to zero.

$$\phi_x + \psi_x^T \nu - \lambda \;\; = \;\; 0 \tag{5.20a}$$

$$\phi_t + \psi_t^T \nu + H \;\; = \;\; 0 \tag{5.20b}$$

However for fixed final states, $dx(t_f) = 0$ and the boundary conditions reduce to

$$(\phi_t + \psi_t^T \nu + H)|_{t_f} \; dt_f = 0 \tag{5.21}$$

Likewise, if the final time is fixed, $dt_f = 0$ and the boundary conditions become

$$(\phi_x + \psi_x^T \nu - \lambda)^T)|_{t_f} \; dx(t_f) = 0 \tag{5.22}$$

With these equations in hand, it is now possible to tackle the path planning problem for the UUV's.

C. Robot Path Planning

To set up the problem, we require the equations of motion, the performance index and the system boundary conditions.

1. Time-varying Constraints

We will use the Gibbs-Appell equations of motion presented in the last chapter to simulate the system. The system is under-actuated; there are eight equations and only three control inputs: Thrust, ϕ and ψ. The kinetic equations $\dot{u_1} = \ddot{\alpha}$, $\dot{u_2} = \ddot{\beta}$, $\dot{u_3}$ can be obtained by differentiating the kinematic equations. We therefore concern ourselves with the kinematic equations.

$$\dot{\alpha} = u_1 \tag{5.23a}$$

$$\dot{\beta} = u_2 \tag{5.23b}$$

$$\dot{x} = u_3 \cos\alpha \cos\beta \tag{5.23c}$$

$$\dot{y} = u_3 \sin\alpha \cos\beta \tag{5.23d}$$

$$\dot{z} = -u_3 \sin\beta \tag{5.23e}$$

This reduces the problem to five equations and three inputs. The five kinematic equations form the time varying constraints and have five Lagrange multipliers associated with them.The force components F_1, F_2, F_3 can be computed from the kinetic equations and in turn can be manipulated to give the actual input. Therefore, for the purposes of the optimal control problem, the generalized speeds u_1, u_2, u_3 may be considered as the inputs to the system.

2. Cost Function

Fuel is limited, since the robots are autonomous, however the energy of the states is not a great concern with this system. Thus, the cost is represented by:

$$L(u) = u(t)^T R u(t) \tag{5.24}$$

where u is a (3x1) vector of the generalized speeds and R is a (3x3) weighting matrix. Combining the cost function with the right hand side of the equations of motion gives the Hamiltonian function.

$$H(x, u, \lambda) = u(t)^T R u(t) + \lambda^T f(x, u) \tag{5.25}$$

x represents the states, u the controls and a the kinematic equations.

3. Boundary Conditions

At each iteration, the cooperative algorithm gives new update positions; we also specify the time taken for each step since the robots travel at 1m/s. Thus $dx(t_f) = dt_f = 0$ so the boundary conditions do not appear in the performance index.

4. Performance Index

The performance index is built combining equations (5.22) through (5.25)

$$J = \int_{t_0}^{t_f} H(x, u, \lambda) - \lambda^T \dot{x} dt \tag{5.26}$$

Taking variations in J and applying Leibniz's rule

$$\begin{aligned} dJ &= \left(H - \lambda^T \dot{x}\right) dt|_{t_f} - \left(H - \lambda^T \dot{x}\right) dt|_{t_0} \\ &\quad + \int_{t_0}^{t_f} \left[H_x^T \delta x + H_u^T \delta u - \lambda^T \delta \dot{x} + \left(H_\lambda - \dot{x}\right) \delta \lambda\right] dt \end{aligned} \tag{5.27}$$

Integrate by parts to remove variation in $\dot{x}$

$$\begin{aligned} dJ &= \lambda\, dx|_{t_0} - \lambda\, dx|_{t_f} + H\, dt|_{t_f} - H\, dt|_{t_0} \\ &\quad + \int_{t_0}^{t_f} \Big[(H_x + \dot{\lambda})^T \delta x + H_u^T \delta u + (H_\lambda - \dot{x})\, \delta\lambda \Big]\, dt \end{aligned} \tag{5.28}$$

From the integral term, we have three necessary conditions. These are not generally affected by changes in the performance index.

$$\begin{array}{llll} \dot{x} = \frac{\partial H}{\partial \lambda} & = f(x(t), u(t)) & \geq t_0 & \textit{State Equation} \\ \dot{\lambda} = \frac{\partial H}{\partial x} & = \left(\frac{\partial f}{\partial x}\right)^T \lambda + \frac{\partial L}{\partial x} & t \leq t_f & \textit{Costate Equation} \\ 0 = \frac{\partial H}{\partial u} & = \left(\frac{\partial f}{\partial u}\right)^T \lambda + \frac{\partial L}{\partial u} & & \textit{Stationarity Condition} \end{array}$$

5. Lagrange Multipliers

The Lagrange multipliers show how changing a constraint will change the performance index. Consider the Hamiltonian:

$$H(x, u, \lambda) = L(x, u, \lambda) + \lambda^T f(x, u) \tag{5.29a}$$

$$dH = H_x\, dx + H_u\, du + H_\lambda\, d\lambda \tag{5.29b}$$

$$= \big(L_x + \lambda^T f_x\big)\, dx + \big(L_u + \lambda^T f_u\big)\, du + f(x, u)\, d\lambda \tag{5.29c}$$

where L is the cost function, λ is the Lagrange multiplier and f is the system constraint. At the critical point, the Hamiltonian is zero, which means that for arbitrary changes in x, u and λ, all the other terms must go to zero. From the first term,

$$\lambda^T = -L_x^T f_x^{-1} \tag{5.30}$$

OR

$$\lambda^T = -\frac{\partial L}{\partial f} \tag{5.31}$$

All Lagrange multipliers are constants so λ's with large magnitudes indicate that small increases in the constraint will produce large changes in the cost function: $L(x, u) = u^2$ in our case. On the other hand, a small λ will mean that large increases will only produce small changes in the cost function. Therefore, constraints with large Lagrange multipliers are important to the performance of the system and constraints with small Lagrange multipliers are not so important. Moreover, the sign of the Lagrange multiplier determines whether the cost function increases or decreases.

The time-varying constraints are the equations of motion, as seen previously. The Lagrange multipliers will therefore be good indicators of the velocities (and subsequently the types of controls) required to perform a maneuver.

D. Line Search Using 'fsolve'

Fsolve is a Matlab function, which solves nonlinear algebraic equations. The necessary conditions are applied to find the differential equations for the states and the costates. Given the initial states and a guess of the Lagrange multipliers, Fsolve will integrate these equations. The function compares the final value of the states it computed with the desired final position. If there is an error, Matlab will change the values of the λ_i and run the program again. The Line search technique used to find the costates is rather simple. Fsolve will take a 2D slice of the objective function and find the minimum. It will then choose another slice in another direction, say perpendicular to the original slice and perform the same linear search again. The algorithm is deterministic and converges quickly to a solution for simple problems.

The syntax used is:

$$[p, fval, exitflag] = fsolve(@robotopt, p, options, xf, x0) \tag{5.32}$$

where *robotopt* is the function we are evaluating, *p* is a vector of Lagrange multipliers, i.e. the input to robotopt,

options gives the user options for integration like step size, number of function evaluations

xf, x0 are parameters that are passed to robotopt.

robotopt in turn integrates a third function *robotodes* using the Matlab tool ode45. *robotodes* are the *Hamiltonian System.*

The Hamilton System is the coupled state and costate equations using the optimal control, which is obtained from the stationarity condition. From the necessary conditions, the stationarity condition gives

$$\frac{\partial H}{\partial u} = Ru + \lambda^T \frac{\partial f}{\partial u} \tag{5.33a}$$

$$u^* = R^{-1}\lambda^T \frac{\partial f}{\partial u} \tag{5.33b}$$

$$u^* = R \cdot \begin{pmatrix} \lambda_\alpha & \lambda_\beta & \lambda_x & \lambda_y & \lambda_z \end{pmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & \cos\beta\cos\alpha \\ 0 & 0 & \cos\beta\sin\alpha \\ 0 & 0 & -\sin\beta \end{bmatrix} \tag{5.33c}$$

$$\begin{pmatrix} u_1^* \\ u_2^* \\ u_3^* \end{pmatrix} = \begin{pmatrix} \lambda_\alpha \\ \lambda_\beta \\ \lambda_x \cos\beta\cos\alpha + \lambda_y \cos\beta\sin\alpha - \lambda_z \sin\beta \end{pmatrix} \tag{5.33d}$$

The state equations are:

$$\begin{aligned}
\dot{\alpha} &= u_1^* &(5.34a)\\
\dot{\beta} &= u_2^* &(5.34b)\\
\dot{x} &= u_3^* \cos\alpha\cos\beta &(5.34c)\\
\dot{y} &= u_3^* \sin\alpha\cos\beta &(5.34d)\\
\dot{z} &= -u_3^* \sin\beta &(5.34e)
\end{aligned}$$

The costate equations are $\dot{\lambda} = -(\frac{\partial f}{\partial x})^T \lambda$

$$\begin{pmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \\ \frac{\partial f}{\partial x_3} \\ \frac{\partial f}{\partial x_4} \\ \frac{\partial f}{\partial x_5} \end{pmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ -u_3^*\sin\alpha\cos\beta & -u_3^*\cos\alpha\sin\beta & 0 & 0 & 0 \\ u_3^*\cos\alpha\cos\beta & -u_3^*\sin\alpha\sin\beta & 0 & 0 & 0 \\ 0 & -u_3^*\cos\beta & 0 & 0 & 0 \end{bmatrix} \tag{5.35}$$

$$\begin{pmatrix} \dot{\lambda}_1 \\ \dot{\lambda}_2 \\ \dot{\lambda}_3 \\ \dot{\lambda}_4 \\ \dot{\lambda}_5 \end{pmatrix} = \begin{pmatrix} \lambda_3 u_3 \cos\beta\sin\alpha - \lambda_4 u_3 \cos\beta\cos\alpha \\ \lambda_3 u_3 \sin\beta\cos\alpha + \lambda_4 u_3 \sin\beta\sin\alpha + \lambda_5 u_3 \cos\beta \\ 0 \\ 0 \\ 0 \end{pmatrix} \tag{5.36}$$

The maximum step taken by a robot is 1m, and the robots travel at 1m/s therefore the equations are integrated over a period of 1s. The final states produced by the integration are compared to the desired positions and the error is minimized by changing the Lagrange multipliers. The process is iterated until either a solution is found, the algorithm exceeds the maximum function evaluations or fsolve fails to find a solution.

These three conditions are identified by the exitflag and a message is printed to the screen.

1. Straight Lines

The simplest case is a position update in a straight line along one of the axes. Results of motion along each axis are shown in Tables XXI and XXII.

There are two important things to note

- the number of iterations needed is highly dependent on the initial conditions and
- the final Lagrange multiplier in the direction of motion is always near one whereas the others are much smaller terms. In fact, as Table 2 shows, the following rule provides very good initial guesses for the Lagrange multipliers. Convergence requires very few iterations.

$$delx = x(t_f) - x(t_0) \tag{5.37}$$

2. Simple Turns

Equation 5.37 is also a good estimate for the next type of maneuver: a simple turn. In this case, the robot is given a final position that requires change along two axes and an angle change so that the nose remains pointed in the direction of motion. For example, from an initial position $(0, 0, 0)$ with no heading or pitch angle, the desired final position is $(0.3, 0.3, 0)$ with a final heading angle $\alpha = -\pi/2$. Optimal trajectories are found consistently within 70 iterations when using the difference rule, whereas the random initial conditions could take anything between 60 and 800 iterations.

Table XXI. Motion along the x-axis using random choices for initial Lagrange multipliers

initial pos	desired pos	initial guess	final lambdas	No. iterations
$[0,0,0,0,0]$	$[0,0,1,0,0]$	$\begin{pmatrix} 0.4186 \\ 0.8462 \\ 0.5252 \\ 0.2026 \\ 0.6721 \end{pmatrix}$	$\begin{pmatrix} 0.0030 \\ -0.0198 \\ 0.9996 \\ -0.0069 \\ 0.0459 \end{pmatrix}$	403
$[0,0,0,0,0]$	$[0,0,1,0,0]$	$\begin{pmatrix} 0.5028 \\ 0.7095 \\ 0.4289 \\ 0.3046 \\ 0.1897 \end{pmatrix}$	$\begin{pmatrix} -0.0001 \\ -0.0002 \\ 1.0001 \\ -0.0002 \\ 0.0004 \end{pmatrix}$	169
$[0,0,0,0,0]$	$[0,0,1,0,0]$	$\begin{pmatrix} 0.7271 \\ 0.3093 \\ 0.8385 \\ 0.5681 \\ 0.3704 \end{pmatrix}$	$\begin{pmatrix} 0.0199 \\ -0.0030 \\ 1.0002 \\ 0.0462 \\ 0.0073 \end{pmatrix}$	61

Table XXII. Motion along the y-axis using random choices for initial Lagrange multipliers

initial pos	desired pos	initial guess	final lambdas	No. iterations
$[-pi/2, 0, 0, 0, 0]$	$[0, 0, 0, 1, 0]$	$\begin{pmatrix} 0.7948 \\ 0.9568 \\ 0.5226 \\ 0.8801 \\ 0.1730 \end{pmatrix}$	$\begin{pmatrix} -0.0011 \\ -0.0001 \\ 0.0025 \\ 1.0000 \\ 0.0003 \end{pmatrix}$	61
$[-pi/2, 0, 0, 0, 0]$	$[0, 0, 0, 1, 0]$	$\begin{pmatrix} 0.9797 \\ 0.2714 \\ 0.2523 \\ 0.8757 \\ 0.7373 \end{pmatrix}$	$\begin{pmatrix} -0.0011 \\ -0.0131 \\ 0.0021 \\ 1.0007 \\ 0.0305 \end{pmatrix}$	175
$[-pi/2, 0, 0, 0, 0]$	$[0, 0, 0, 1, 0]$	$\begin{pmatrix} 0.6614 \\ 0.2844 \\ 0.4692 \\ 0.0648 \\ 0.9883 \end{pmatrix}$	$\begin{pmatrix} -0.0117 \\ -0.0085 \\ 0.0277 \\ 1.0003 \\ 0.0193 \end{pmatrix}$	127

Table XXIII. Straight line path planning for robots using intelligent first guess for λ's. The final λ's are exactly the same as the initial guess

motion	initial pos	desired pos	final λ's	Number of iterations
x	$[0, 0, 0, 0, 0]$	$[0, 0, 1, 0, 0]$	[0,0,1,0,0]	7
x	$[0, 0, 0, 0, 0]$	$[0, 0, 0.75, 0, 0]$	[0,0,0.75,0,0]	7
x	$[0, 0, 0, 0, 0]$	$[0, 0, 0.5, 0, 0]$	[0,0,0.5,0,0]	7
x	$[0, 0, 0, 0, 0]$	$[0, 0, 0.25, 0, 0]$	[0,0,0.25,0,0]	7
y	$[-\pi/2, 0, 0, 0, 0]$	$[-\pi/2, 0, 0, 1, 0]$	[0,0,0,1,0]	7
y	$[-\pi/2, 0, 0, 0, 0]$	$[-\pi/2, 0, 0, 0.75, 0]$	[0,0,0,0.75,0]	7
y	$[-\pi/2, 0, 0, 0, 0]$	$[-\pi/2, 0, 0, 0.5, 0]$	[0,0,0,0.5,0]	7
y	$[-\pi/2, 0, 0, 0, 0]$	$[-\pi/2, 0, 0, 0.25, 0]$	[0,0,0,0.25,0]	7
-y	$[\pi/2, 0, 0, 0, 0]$	$[\pi/2, 0, 0, -1, 0]$	[0,0,0,-1,0]	7
-y	$[\pi/2, 0, 0, 0, 0]$	$[\pi/2, 0, 0, -0.75, 0]$	[0,0,0,-0.75,0]	7
-y	$[\pi/2, 0, 0, 0, 0]$	$[\pi/2, 0, 0, -0.5, 0]$	[0,0,0,-0.5,0]	7
-y	$[\pi/2, 0, 0, 0, 0]$	$[\pi/2, 0, 0, -0.25, 0]$	[0,0,0,-0.25,0]	7
z	$[0, \pi/2, 0, 0, 0]$	$[0, \pi/2, 0, 0, 1]$	[0,0,0,0,1]	7
z	$[0, \pi/2, 0, 0, 0]$	$[0, \pi/2, 0, 0, 0.75]$	[0,0,0,0,0.75]	7
z	$[0, \pi/2, 0, 0, 0]$	$[0, \pi/2, 0, 0, 0.5]$	[0,0,0,0,0.5]	7
z	$[0, \pi/2, 0, 0, 0]$	$[0, \pi/2, 0, 0, 0.25]$	[0,0,0,0,0.25]	7

3. Multiple Turn Maneuvers

Once we start to combine turns the coupled nature of the equations soon becomes noticeable. Consider Table XXIV. It depicts a robot performing a 90° turn left and a 90° pitch so that it ends up on its nose. The robot initially moves 0.3m in each direction. The Lagrange multipliers needed to perform this maneuver are shown in the final column. The final positions are varied along each axis separately to investigate which λ's should be varied for each maneuver so as to minimize the number of iterations needed. To generalize the results, random initial values for each λ were generated in the range [0..1]. Each case was run several times to ensure the Lagrange multipliers obtained were reliable.

As before, $delx = x_f - x_0$ is a more appropriate initial guess for the Lagrange multipliers. The number of iterations was smaller than with random initial conditions

4. Varying the Update Step

Update steps of up to 7m were investigated and the previous method for choosing the initial Lagrange multipliers failed in several areas. The following adaptation was successful for all types of position updates

```
for  $i = 1:5$
          if     $delx > 0$,   $lam(i) = 1$
                 $delx = 0$,   $lam(i) = 0$
                 $delx < 0$,   $lam(i) = -1$
          end    %if
end              % for
```

The number of iterations increased slightly, but the method was more robust.

Table XXIV. Combined turn

initial pos $[\alpha, \beta, x, y, z]$	desired pos $[\alpha, \beta, x, y, z]$	change	final λ's $[\lambda_\alpha, \lambda_\beta, \lambda_x, \lambda_y, \lambda_z]$
$[0, 0, 0, 0, 0]$	$[\pi/2, \pi/2, 0.3, 0.3, -0.3]$		[2.0, 1.2, -0.1, 2.1, 0.4]
$[0, 0, 0, 0, 0]$	$[\pi/2, \pi/2, 0.4, 0.3, -0.3]$	x	[2.0, 1.2, 0.3, 1.7, 0.4]
$[0, 0, 0, 0, 0]$	$[\pi/2, \pi/2, 0.5, 0.3, -0.3]$	x	[1.9, 1.2, 0.8, 1.3, 0.3]
$[0, 0, 0, 0, 0]$	$[\pi/2, \pi/2, 0.6, 0.3, -0.3]$	x	[1.8, 1.15,1.2, 0.9, 0.3]
$[0, 0, 0, 0, 0]$	$[\pi/2, \pi/2, 0.7, 0.3, -0.3]$	x	[1.5, 1.1, 1.6, 0.5, 0.3]
$[0, 0, 0, 0, 0]$	$[\pi/2, \pi/2, 0.8, 0.3, -0.3]$	x	[1.3, 1.0, 1.9, 0.2, 0.3]
$[0, 0, 0, 0, 0]$	$[\pi/2, \pi/2, 0.9, 0.3, -0.3]$	x	[1.1, 0.9, 2.2,-0.1, 0.3]
$[0, 0, 0, 0, 0]$	$[\pi/2, \pi/2, 0.3, 0.4, -0.3]$	y	[2.2, 1.0, -0.5, 2.9, 0.8]
$[0, 0, 0, 0, 0]$	$[\pi/2, \pi/2, 0.3, 0.5, -0.3]$	y	[2.3, 0.9, -0.8, 3.2, 1.0]
$[0, 0, 0, 0, 0]$	$[\pi/2, \pi/2, 0.3, 0.6, -0.3]$	y	[2.4, 0.8, -1.0, 3.4, 1.2]
$[0, 0, 0, 0, 0]$	$[\pi/2, \pi/2, 0.3, 0.7, -0.3]$	y	[2.5, 0.7, -1.2, 3.5, 1.3]
$[0, 0, 0, 0, 0]$	$[\pi/2, \pi/2, 0.3, 0.8, -0.3]$	y	[2.6, 0.6, -1.3, 3.5, 1.4]
$[0, 0, 0, 0, 0]$	$[\pi/2, \pi/2, 0.3, 0.9, -0.3]$	y	[2.7, 0.5, -1.4, 3.6, 1.4]
$[0, 0, 0, 0, 0]$	$[\pi/2, \pi/2, 0.3, 0.3, -0.4]$	z	[1.90, 1.37, 0.06, 1.70, -0.03]
$[0, 0, 0, 0, 0]$	$[\pi/2, \pi/2, 0.3, 0.3, -0.5]$	z	[1.81, 1.51, -0.04, 1.29, -0.44]
$[0, 0, 0, 0, 0]$	$[\pi/2, \pi/2, 0.3, 0.3, -0.6]$	z	[1.75, 1.62, -0.06, 0.95, -0.80]
$[0, 0, 0, 0, 0]$	$[\pi/2, \pi/2, 0.3, 0.3, -0.7]$	z	[1.67, 1.76, -0.07, 0.57, -1.14]
$[0, 0, 0, 0, 0]$	$[\pi/2, \pi/2, 0.3, 0.3, -0.8]$	z	[1.63, 1.88, -0.12, 0.24, -1.42]
$[0, 0, 0, 0, 0]$	$[\pi/2, \pi/2, 0.3, 0.3, -0.9]$	z	[1.59, 1.97, -0.18, -0.01, -1.66]

E. Conclusions

Fsolve solves the optimal control problem by solving a set of algebraic equations. The number of iterations taken to find the solution depend on the initial conditions. Some rules of thumb can be used to find a solution with fewer iterations. This is desirable in robotics where onboard computation is often limited.

Fsolve cannot implement inequality constraints. Simulations are therefore not smooth and sometimes show the robot moving backwards. Backwards motion can be achieved by reversing the propeller, however, many of the more complicated maneuvers require switching between forwards and reverse motion. This is not the most efficient use of the propeller.

The constraints on thrust angles ϕ and ψ have not been imposed either. Thus, many of the optimal paths chosen may not be feasible.

CHAPTER VI

GENETIC ALGORITHM

A. Why Genetic Algorithms?

Genetic algorithms attempt to solve optimization problems using the basic theories behind genetics. These types of search methods have proved superior for some classes of problems when compared to traditional search and optimization methods due to their random nature. Genetic algorithms are also parallel in nature, which typically makes them more effective as a search tool. They have excelled at solving problems where traditional methods break down, particularly

- Multi-modal problems (i.e. when there is more than one extremum). Most traditional methods can identify an extremum, but will not be able to verify if it is a global extremum or a local one. The stochastic nature of genetic algorithms ensures the whole search space is covered and thereby guarantees that the final solution is indeed the global maximum or minimum.

- Pareto-optimal systems where an optimal solution is required in the presence of several, conflicting constraints. For example, a car has to be built according to the designer's taste, with certain safety features, having suitable creature comforts and all to budget!

In the UUV path planning problem, we are looking to find 5 Lagrange multipliers in order to achieve the optimal performance for our system. Since the problem is extremely sensitive to initial conditions, I was interested to see if a genetic algorithm would help in the search process. One of the most striking aspects of genetic algorithms is the number of variations available. Each problem requires a different approach in order to obtain the optimal solution. I have therefore presented a number of techniques. Much of the material comes from a well referenced set of lecture notes and features with permission of the author [9].

B. Background

Most people are familiar with Charles Darwin and his controversial book *The Origin of the Species* (abbr. title). In it he proposed that within each species there was variation and that this variation was brought about through reproduction. From every day life we know that every child will inherit characteristics from his parents but these will not be the same as those inherited by his siblings. Darwin also noted that while an animal species will produce many offspring, very few survive to adulthood. He described this as natural selection - the survival of those who were best suited to their environment. As the environment changes, so do the criteria needed for an individual to survive. Later Hugo de Varis noted that plants would occasionally throw up completely new variants whose characteristics had no resemblance to the parent plants. This phenomenon was named mutation and is due to a random or unexplained replacement of one gene with another. The effects of mutation can be dramatic or unnoticeable depending on the importance of the gene. A certain characteristic, say eye color, may be dependent on one single gene. A mutation in this case would cause dramatic effects. The length of a person's arm, however, is dependent on many genes so a mutation in the cell of a fingernail is unlikely to cause a noticeable change. Work on genetic algorithms and its affiliates: evolutionary algorithms and evolutionary programming began in the late 50's. The first attempts to use genetic algorithms focused on mutation to produce the variation at each new generation, however the rate of variation was far too slow to effectively find a solution. In 1975, John Holland at the University of Michigan successfully modelled the mixing of chromosomes, which occur in Nature during mating and went on to create the first functioning genetic algorithm.

C. A Simple Genetic Algorithm

The basic genetic algorithm has 6 steps:

1. Representing problem and constructing objective function
2. Forming an initial population
3. Evaluating the fitness of each individual in the population
4. Selecting the individuals for mating
5. Mating/Recombination to produce offspring
6. Reinsertion (forming a new population)

Steps 3 to 6 are iterated until a satisfactory solution has been found.

1. Representing the Problem

First, produce a number of possible solutions to the problem. These are called individuals and together the individuals form the population. The size of the population depends on the problem. Secondly, encode the individuals in a form that resemble genes. In the natural world, genetic information is held in chromosomes, which are strings of genes. The closest analogy is to use binary encoding. This means converting the individuals to a string of ones and zeros.

Example 1:

We want to find the solution of $y = x_2$. The initial population has 4 individuals at 1, -2, 4, -6. Fig. 22 shows the individuals in binary format.

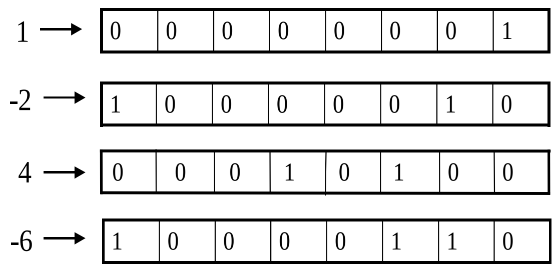

Fig. 22. Binary representation of individuals 1, -2, 4 and -6 respectively

Here the first bit is used to represent the sign of the integer. '1' represents a negative sign, '0' a positive sign.

For multiple parameter problems, each parameter is encoded in the same way and then concatenated to form a longer chromosome.

Other forms of encoding are:

- Gray scale
- Integer labels (good for discrete problems like the travelling salesman)
- Real - valued genes. This type of encoding is said to be much more computationally efficient.

2. Initializing the Population

Populations of 30 - 100 are typical. The more individuals used to search the objective function, the faster the solution is likely to be found. On the other hand, large populations require a greater amount of computation since there are more children produced. Micro populations of 10 or so have also been investigated. Initializing the individuals is best achieved using a random number generator at the gene level. For a population of 10 individuals with chromosomes 8 bits long, we create 80 bits (0 or 1). This means the designer has no control over the initial population, which ensures a more even spread of the population over the search space.

3. Evaluating the Fitness of the Individuals

First we need a criterion to judge how near to the solution each individual is. This is known as the raw fitness of an individual. Sometimes, the designer knows what the solution should be, but doesn't know how to achieve it. In this case, it is simply a matter of comparing the desired solution to that of the individual. The individuals with the least error have the highest fitness.

Example 3:

Going back to Example 1, the objective function is: $y = x_2$. It has a solution at x=0. The fitness of the initial population can be summarized in Table XXV.

Table XXV. Evaluate the fitnesses of individuals from Example 1

Individual	Initial Value	Absolute Error	Relative Fitness
1	1	1	1
2	-2	2	2
3	4	4	3
4	-6	6	4

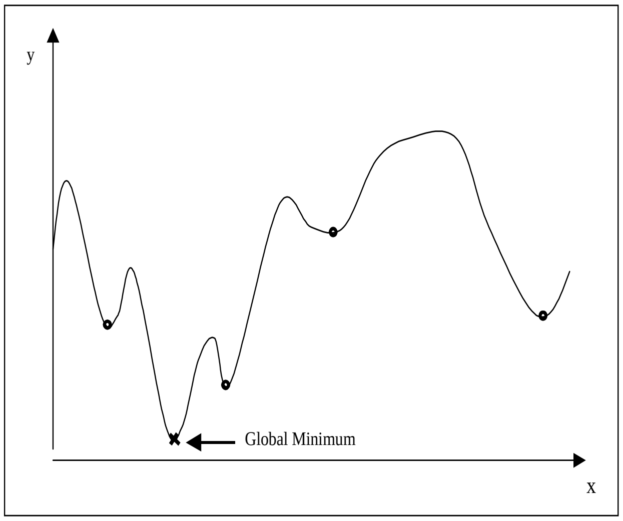

Fig. 23. Multi-modal graph showing local and global minima

Usually the objective is to minimize a quantity. In this case, the raw fitness of an individual will be determined relative to the other individuals, which increases the chances of finding the global minimum. In Fig. 23 there are 5 minima. Most gradient search methods will locate the minimum nearest to their initial position, but this is not usually the best solution. By comparing the fitnesses of individuals randomly scattered over the whole search space, the GA can successfully locate the global minimum.

The raw fitness of each individual is not quite in the form we would like. First, for minimization problems, we would like to have the individual with the smallest solution to have the highest fitness. Secondly, we would like to be able to accurately assess the fitness of each individual regardless of the sign of the raw fitness. We achieve this through a fitness function. There exist a number of methods to achieve a suitable fitness function. Proportional fitness assigns a fitness value to each individual by comparing its raw fitness value to the sum of the others in the population.

$$F(x_i) = \frac{f(x_i)}{\sum_{i=1}^{No.Ind} f(x_i)} \tag{6.1}$$

For minimization problems, we can use the following conversion to achiever the correct raw fitness:

Let $r = obj_{max} - obj_{min}$ be the range for the current generation

$$f(x_i) = 1.001 - \frac{obj_i - obj_{min}}{r} \tag{6.2}$$

Proportional fitness is a popular method but does not account for the negative objective function values. We can, instead use the absolute values of the raw fitness.

An alternative method is the power scaling function.

$$F(x_i) = f(x_i)^k \tag{6.3}$$

where k is a varying function allowing the range of fitness to expand or contract, as the user wishes. Typically k is a decaying exponential function. The advantage of letting k vary is that we can avoid premature convergence or unnecessarily long searches.

4. Selecting the Individuals for Mating

How do we choose a mate? It is generally true that animals with the right attracting characteristics will be more likely to mate, however, as most of us will agree that courting (even if we have the right characteristics) has an uncertain outcome. Indeed, male spiders need to be particularly careful else they may end up as the female's lunch! To summarize, we want selection procedure that is biased towards the most fit individuals, but has a degree of uncertainty built in.

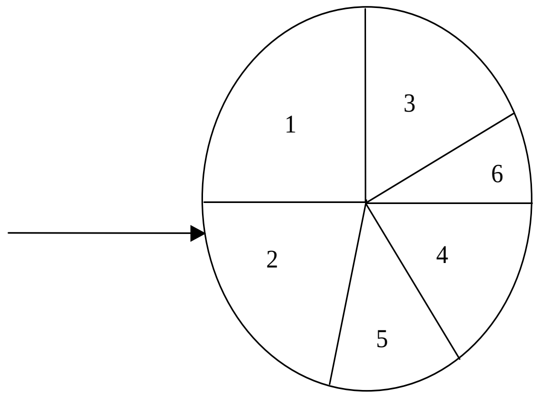

Fig. 24. Roulette wheel used for Stochastic Sampling Replacement (SSR) selection method

Most selection techniques use some form of the roulette wheel, shown in Fig. 24. Here, each individual is assigned a slice of the wheel according to his fitness. The wheel is spun and the individual chosen by the arrow is selected for mating. Its pair and indeed all the mating pairs are chosen in a similar fashion. (A pair obviously must consist of two different individuals.) This technique is also known as the stochastic sampling replacement (SSR) method.

The random nature of the SSR method means that one individual could potentially be represented in all the pairs, even though its fitness is fairly small. Equally, an individual with a relatively large fitness may not be chosen for mating at all. This leads to a narrowing of the gene pool and thereby early convergence at an undesirable location. Many solutions have been presented to overcome this problem, but the simplest and most popular, is the stochastic universal sampling (SUS) method.

The SUS is very similar to the SSR method, except it uses '2N' equally spaced pointers to obtain the individuals for mating. (N is the number of pairs required for the next generation). Fig. 25 shows the roulette wheel needed for four pairs.

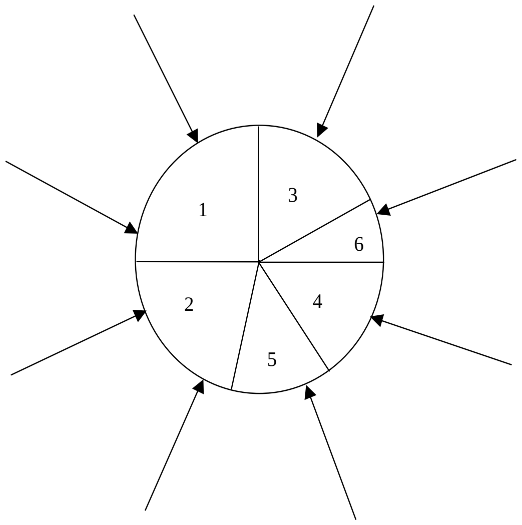

Fig. 25. Stochastic Universal Sampling (SUS) Method

The population is initially shuffled and the individuals selected. The SUS has many advantages:

- fitnesses are accurately represented,
- gene pool remains sufficiently varied,
- roulette wheel needs only one spin in order to collect all the individuals for mating.

The latter can reduce computational time.

5. Mating/Recombination and Mutation

a. Recombination

In nature children are produced when genetic material is swapped between the parents. Crossover uses the same basic principle to create new individuals for the genetic algorithm. The method is best illustrated by an example.

Example 4: Simple or Single-point Crossover

Consider Fig. 26.

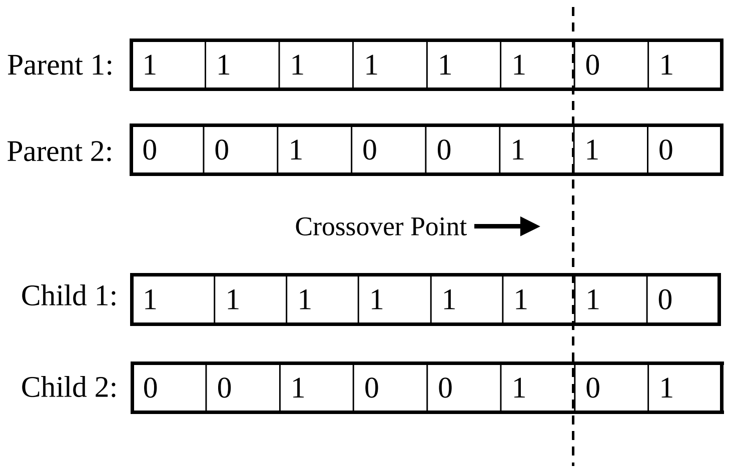

Fig. 26. Single-point crossover

Comments:

The crossover point is picked using a random number generator from the range [1..L-1], where L is the length of the chromosome.

- Child 1 consists of genes from parent 1 lying to the left of the crossover point and the genes of parent 2 lying to the right of the crossover point.
- Child 2, on the other hand, is made from the genes of parent 1 lying to the right of the crossover point and the genes of parent 2 lying to the left of the crossover point.
- A criticism of single-point crossover is that it is not disruptive enough and therefore does not encourage a particularly wide search of the objective function. If the crossover point does not fall in the most appropriate place, the children

will not be very different from the parents. This defeats the object of mating in the first place! Let us convert the chromosomes of Example 4 back to integer values:

Parent 1: -125, Parent 2: 38, Child 1: -126, Child 2: 37

If instead, the crossover point had been chosen after the 3rd bit, the children would be a better mix of the two parents:

Parent 1: -125, Parent 2: 38, Child 1: -102, Child 2: 61

There are several variations of simple crossover, which all attempt to overcome this problem. One of the variations available is the multi-point crossover, a direct extension of the single-point crossover method. In Multi-point crossover, p points are chosen in the range [1.. L-1] using a random number generator, ensuring each point is different. Leaving the first section of each parent where it is, successive sections between the following crossover points are exchanged between the two chromosomes to produce the children. Again, this is best seen with an example.

Example 5: Multi-point Crossover

Consider again the chromosomes of Example 4 this time with crossover points after bits 1, 3, and 5 (as shown in Fig. 27). Converting back to integer values, we have:

Parent 1: -125, Parent 2: 38, Child 1: -64, Child 2: 101 Now consider the crossover points 2, 4, 7. The chromosomes now have the values

Parent 1: -125, Parent 2: 38, Child 1: -108, Child 2: 45 The two methods both have advantages, the multi-point crossover operator ensures a more thorough search, but will delay the time to convergence. Its disruptive nature may also cause good genetic material to be lost. The designer must choose which is most appropriate to

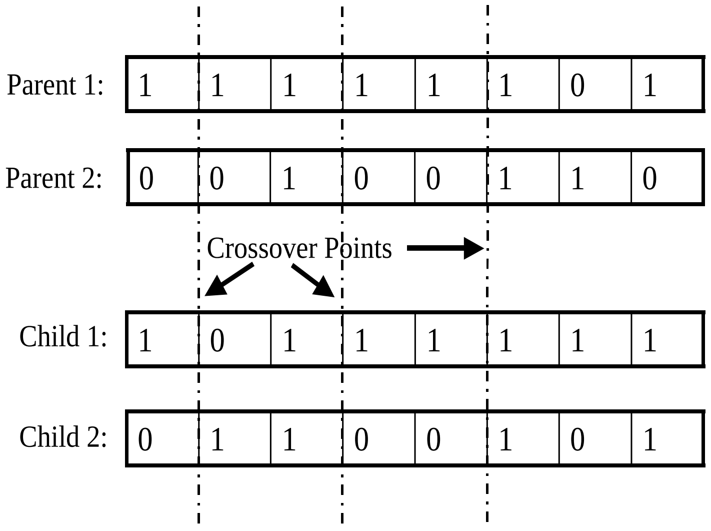

Fig. 27. Multi-point crossover

the problem at hand.

Example 6: Arithmetic Crossover [10]

In this method a constant ‘f’ is randomly chosen in the range [0,1]. New individuals are formed by linearly combining the two parents in the following way:

$$\begin{aligned}
P_1 &= [x_1, x_2, x_3, ..., x_N] \\
P_2 &= [y_1, y_2, y_3, ..., y_N] \\
C_1 &= [f \cdot x_1 + (1-f) \cdot y_1, f \cdot x_2 + (1-f) \cdot y_2, f \cdot x_3 + (1-f) \cdot y_3, ... \\
C_2 &= [f \cdot y_1 + (1-f) \cdot x_1, f \cdot y_2 + (1-f) \cdot x_2, f \cdot y_3 + (1-f) \cdot x_3, ...
\end{aligned} \tag{6.4}$$

where P_i are the parents and C_i are the children.

Clearly, this method can only be used with non-binary numbers, however most problems deal with real numbers or integers, so arithmetic crossover may be applied once the individuals have been reconverted back into their original form.

b. Mutation

As we have seen previously, mutation is the random replacement of one gene with another. It does not occur very frequently in nature, or its effects would have been discovered much earlier. Programming mutation in genetic algorithms is very easy, particularly when the binary representation has been used. Simply use a random number generator in the range [1..L] and flip the bit so that $1 \longrightarrow 0$ and $0 \longrightarrow 1$. The mutation operator is applied with very low probability, usually in the range $0.1\% - 1\%$. [9]. Higher mutation rates increase the disruptive nature of the process.

6. Reinsertion

Once the new generation has been produced, it has to be integrated into the current population. The algorithm calculates the fitness of all the individuals - current and new. It is then up to the designer to decide how to use this information to update the population. The Elitist strategy states only the fittest individuals will survive to the next generation, however it is often the weaker individuals who provide the genetic material to cover the whole search space, see Fig.28.

Point 1 is the fittest individual, but points 6 and 7 are closer to the global optimum. In this case, the fittest individual does not represent the best genetic information. The Elitist strategy would at best hinder the search and at worst cause the GA to converge to a local maximum. It also means that the GA is unnecessarily dependent on initial conditions.

Fogarty [9] performed several studies into reinsertion techniques and asserts that the best scheme is not a replacement of the least fit individuals, but of the oldest individuals. As in nature, even the fittest individuals will grow weak and die, so in

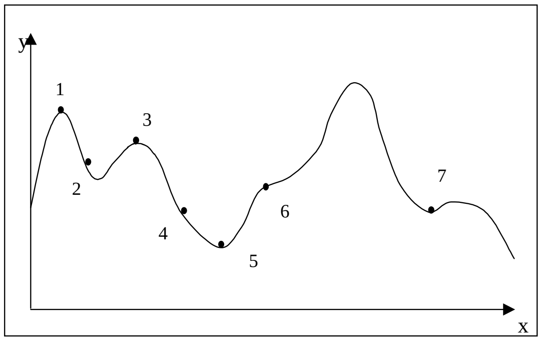

Fig. 28. Solutions produced by new generation

the genetic algorithm, the earlier estimates become less useful as the search progresses.

D. Termination of the Genetic Algorithm

The designer can specify the algorithm to run for a number of generations and then compare the performance of the GA to his/her desired performance OR can simply state a certain target for the GA to obtain. The advantage with the first technique is that if the GA is obviously going in the wrong direction, or has converged on some non-optimal solution, the designer can stop the program and start again with a new set of initial conditions. The disadvantage is that it requires more user-intervention.

E. Using GA's to Find the Lagrange Multipliers

1. Representing problem and constructing objective function

Each individual represents a solution to the problem. The problem at hand is to find the initial 5 Lagrange multipliers that will solve the optimal control problem and move the robot to the desired location. The Lagrange multipliers are initially real valued and are converted to a 21 bit binary representation of each number. This allows for a range [-4, 4] to an accuracy of 4 decimal places.

2. Forming an initial population

   Populations of 10, 30, 40 and 60 were investigated.

3. Evaluating the fitness of each individual in the population

   This was done by using each set of Lagrange multipliers as an initial guess to the costate equations. The state and costate equations were then integrated for one second and the computed final position was compared to the desired final position. The total error was computed by summing the absolute errors for each state α, β, x, y, z. This was the raw fitness and is proportional to the error. The actual fitness of each individual should be inversely proportional, as mentioned previously. I therefore used the following equation

   $$f(x_i) = 1.001 - \frac{obj_i - obj_{min}}{obj_{max} - obj_{min}} labeleqn : 6.5 \tag{6.5}$$

4. Selecting the individuals for mating

   Only two children were produced at each new generation, regardless of the size of the population. Larger numbers of children tended to cause early convergence. The Stochastic Universal Sampling (SUS) method was used to choose the parents. Since the GA only uses two parents, this method does not have much advantage over the simpler Stochastic Sampling replacement method (SSR).

5. Recombination and Mutation

   Recombination was achieved using multiple crossover technique. Initially 5 points were chosen because there are 5 Lagrange multipliers.

   Mutation was applied every 10 iterations. Each individual was assigned a random number ‘mut’ in the range [0,1]. If ‘mut’ exceeded 0.99, one gene from every Lagrange multiplier was flipped. This is a very drastic mutation, almost as disruptive, if not more than recombination. It was vital in this scheme to help avoid early convergence.

6. Reinsertion (forming a new population)

   Initially the oldest individuals were thrown away, but this strategy tended to eliminate the better individuals and caused convergence at a suboptimal point. An elitist strategy was therefore applied, but as the literature indicates, this increased the likelihood of genetic dominance by one or two individuals. Large mutation rates were applied to prevent early convergence. Certainly, the performance of the GA was significantly better with the Elitist strategy than with the "new generation" strategy.

7. Termination

   The GA was left to run its course. Correct termination occurred if a solution had been found with a combined error of 0.1. Suboptimal convergence occurred if all the individuals converged on a point before reaching the desired error. For steps of 1m, this is already a very loose constraint on the final position.

F. Results

1. Comparing population sizes: Population sizes of 10,20,30 and 40 were investigated and compared. Three trajectories were considered: Motion in a straight line, left turn and right turn. All trajectories remained in the x-y plane. Micro-populations ran and terminated extremely quickly, but the results obtained were very poor. The quality of results increased as the population got larger but the time to run each case increased exponentially. The smaller populations

were subject to less mutation since convergence occurred within fewer iterations. This was still true if the probability was raised. The results are shown

Table XXVI. Effect of population size on GA performance

Population	Straight Line		Turn Left		Turn Right	
size	error	time	error	time	error	time
10	0.8860	25.6	1.7799	33	1.6904	29
20	0.3690	87	1.3479	70	1.1353	96
30	0.1973	117	1.1784	146	1.0925	148
40	0.1697	231	1.000	194	1.1740	239

in table XXVI. Because of the computational restrictions, a population of 20 individuals was considered the best option for further study.

2. Comparing numbers of Crossover points: If a population does not converge, it seems intuitive to introduce more crossover points and thus increase the randomness in the system in order to carry out a wider search. In fact, having too many crossover points can destroy the good material that is already there. The fewer crossover points there are, the greater the probability that good material will remain in the population. Single-point crossover is a generally a very popular tool. In this case, it was not enough given the length of the individual. An investigation was carried out with a population of 20 individuals. Results of this investigation are shown in table XXVII. Five crossover points produced the best performance. This constitutes roughly 1 crossover point per λ. It also had the longest average convergence time. For a population of 20 individuals, none of the test cases gave a satisfactory result.

Table XXVII. Effect of number of crossover points on GA performance

Crossover	Time to Convergence		Final Error	
Points	mean	std	mean	std
1	58.9	10.2	0.4786	0.2248
3	76.9	20.2	0.3529	0.1412
5	102.8	25	0.2628	0.0659
7	97.1	16.4	0.2759	0.1293
10	85	19.3	0.2923	0.1300
15	82.2	20.6	0.3817	0.2015

G. Conclusions

GA's are not a good tool for real time path planning. Owing to their random nature, convergence requires many iterations and there was no guarantee of actually finding a solution. For a "good" solution, they are computationally expensive, however, Matlab is much slower than C or Fortran so it is more useful to compare iterations. The amount of initial programming needed is much higher than with the fsolve method. Even using a toolbox, there are so many random variables contributing to the performance of the genetic algorithm, it almost requires a genetic algorithm to train a genetic algorithm!

CHAPTER VII

DYNAMIC PROGRAMMING

A. Introduction

The Two Point Boundary Value problem is extremely difficult to solve for nonlinear systems. It has a number of disadvantages

- 2 point BVP is dependent on having good initial guesses for the costates (λ_i).
- Most real systems have constraints, which tend to make it harder to find a solution
- Finding the solution, even with good guesses, is very time consuming

There are a number of modifications to 2 point BVP, which can aid solution. One is the multiple-shooting method, in which the search space is divided up into sections and the 2 point BVP is solved for each section. This is advantageous if the quantities you wish to minimize are large, however this is not the case for the Ranger UUV. Another method is Pontryagin's minimum principle, which gives some guidelines for optimizing controls in the presence of constraints. Even with these modifications, variational control is still an awkward tool to use and many people prefer to use dynamic programming methods.

Dynamic Programming was invented in 1953 by Richard E. Bellman. He subsequently published several papers on the subject in the late 50's and early 60's. This discrete method is well suited to implementation on digital computer and has also proved to be more computationally efficient than the traditional variational approach to optimal control because constraints make the problem easier and not harder to solve [11].

This chapter will begin with a general review of dynamic programming. Section C will present dynamic programming with Interior Points (DPIP). The DPIP method was implemented using a toolbox developed at Texas A&M and Sandia National Laboratories [12], which is described in section D. The results of the simulations can be found in section E.

B. Review of Dynamic Programming

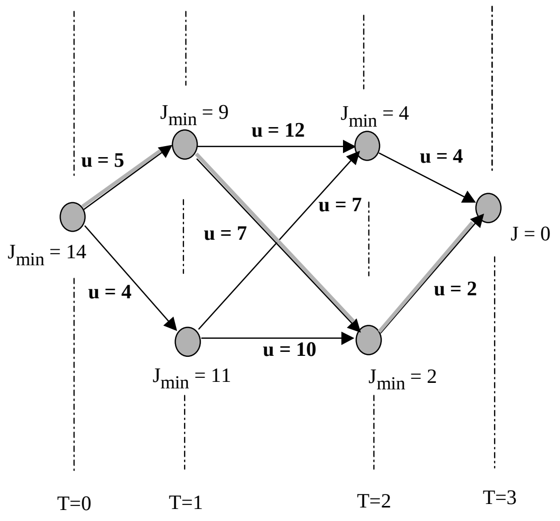

Fig. 29. Method of dynamic programming (the optimal path is highlighted in grey)

Dynamic Programming is applied backwards in time. First, the states and controls are given a range of discrete, finite values (also known as admissible values). Then we define the cost function J, a function of the variables we want to minimize

$$J = \frac{1}{2}x(T)^2 + \frac{1}{2}\int_{t_0}^{t_f} u(t)^2 dt \tag{7.1}$$

where $x(T)$ is the state at the final time and $u(t)$ is a scalar control input. Consider

Fig. 29. The cost of the system at the final time (T=3) is calculated using equation 7.1 and applying all the admissible controls, we work backwards to find the previous admissible states (T=2). Here we calculate the path, which will give us the minimum cost to/from the final state or set of states i.e. the optimal path. The cost at node (T=2) is calculated using the control applied over the optimal path and the final position x(T=3). We then apply the admissible controls again to get the states (T=1). The whole process is iterated until the initial states, the total (minimum) cost and therefore the optimal path are obtained.

Dynamic Programming is very simple and can be implemented using the steps overpage. To illustrate the method I have applied it to a linear system. The example is based on one given by Lewis and Syrmos [11]. It features with kind permission of the authors. This method can also be applied successfully to nonlinear systems.

1. State system equations, e.g. for a linear system

$$\dot{x} = A\,x + B\,u \tag{7.2}$$

2. Form the performance index. For a minimum control effort problem subject to final state constraints,

$$J(0) = \frac{1}{2}\,x^2(T) + \frac{1}{2}\int_0^T u^2(t)\,dt \tag{7.3}$$

   where J(0) is the cost from the initial time to the final time.

3. Define the state and control constraints, e.g.

$$0\ \le x(t) \le 2 \quad |u(t)| \le 1$$

4. Discretize the system equations and the performance index

$$x_{k+1} = A_k\,x_k + B_k\,u_k \tag{7.4}$$

where $A_k = e^{A\tau}$, $B_k = \int_0^{\tau} e^{A\tau}\, B\, dt$, τ is the sample in seconds. Thus, for a scalar system, if $a = 1$, $\tau = 0.5$ and $b = 1$

$$\begin{aligned} x_{k+1} &= e^{0.5}\, x_k + (e^{0.5} - 1)\, u_k && (7.5) \\ &= c\, x_k + (c-1)\, u_k && (7.6) \end{aligned}$$

where $c = e^{0.5} = 1.645$. The performance index is

$$J_0 = \frac{1}{2}\, x_N^2 + \frac{1}{2} \sum_{k=0}^{N-1} ((c-1)u_k)^2 \tag{7.7}$$

where $N = T/\tau$, T is the final time and τ is the sampling period.

5. Form a set of discrete admissible controls within the bounds previously stated. e.g. -1,0,1. This is a perfectly reasonable set of controls, particularly for bang-off-bang control systems.

6. Form a set of discrete admissible states, see Fig. 30. This is also a reasonable assumption if each admissible state can be obtained with at least one of the admissible controls. For discrete problems, like the travelling salesman, the discrete states will already be defined by the problem. For continuous problems, the more admissible states we have, the closer the model resembles the true system and therefore the better the final control. As always, the finer the mesh, the more computational effort needed.

7. Calculate the performance index at the final states. With k = N = 2,

$$J_0 = \frac{1}{2} x_N^2 \tag{7.8}$$

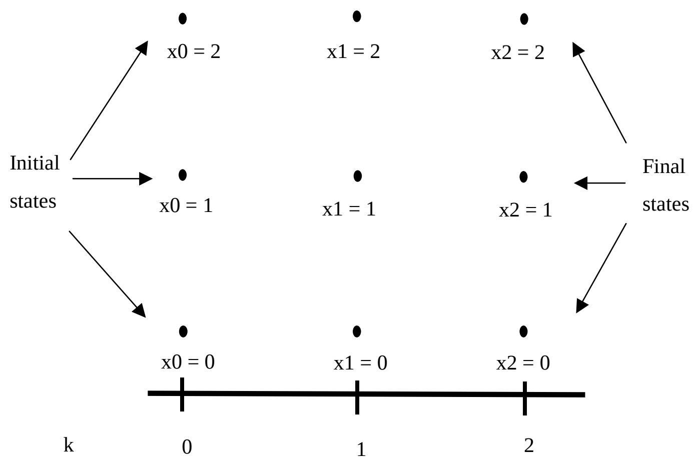

Fig. 30. Grid of admissible states

Thus

$$J_2(x_2 = 2) = \frac{2^2}{2} = 2 \tag{7.9}$$

$$J_2(x_2 = 1) = \frac{1^2}{2} = 0.5 \tag{7.10}$$

$$J_2(x_2 = 0) = \frac{0^2}{2} = 0 \tag{7.11}$$

8. Move left to the set of previous admissible states at x_1 and apply each of the admissible controls. These are shown in Fig. 31. Note that only the controls which end in the range of the admissible states are used. In fact, if a control results in a state lying between two admissible states it is still feasible, but it is not defined. It is therefore necessary to interpolate between the two states to find the cost of reaching this point. The cost of using each control is calculated

using a simplified form of the performance index.

$$J_k = J_{k+1} + \frac{(c-1)^2}{2} u_k \tag{7.12}$$

This equation is key, forming the basis for most extensions to dynamic programming. The optimal path between any state at k=1 and a state at k=2 is the one with the minimum cost. The cost of that state is therefore solely dependent on the cost of the state we came from and the control needed to move between the states. These numbers are shown in Fig. 31. The cost of each path is shown above each line with the control at the end of the line. The minimum cost of each state at k=1 is shown above it.

9. Jump back a state again and repeat procedure to find minimum cost for each state. The results are shown in Fig. 32. Since J_0 is the set of initial states, the iterative process ends here. The cost for each of the initial states represents the total cost of reaching one of the final states. The optimal state trajectory is found by using the optimal controls. The optimal trajectory from $x0 = 2$ is shown as a thick grey line in Fig. 32. When using this control weighting, all the optimal state trajectories tend towards zero. If instead, the performance index was

$$J_0 = \frac{1}{2}\, x_k + 5 \sum_{k=0}^{N-1} u_k^2 \tag{7.13}$$

The rate of decay would be significantly slower. This effect represents using less control effort in the actual system. The process may seem tedious for two steps but the numerical efficiency quickly becomes apparent as the steps increase. On average, the number of computations needed for dynamic programming is of the order

$$No.states(n) \times No.controls(m) \times N \tag{7.14}$$

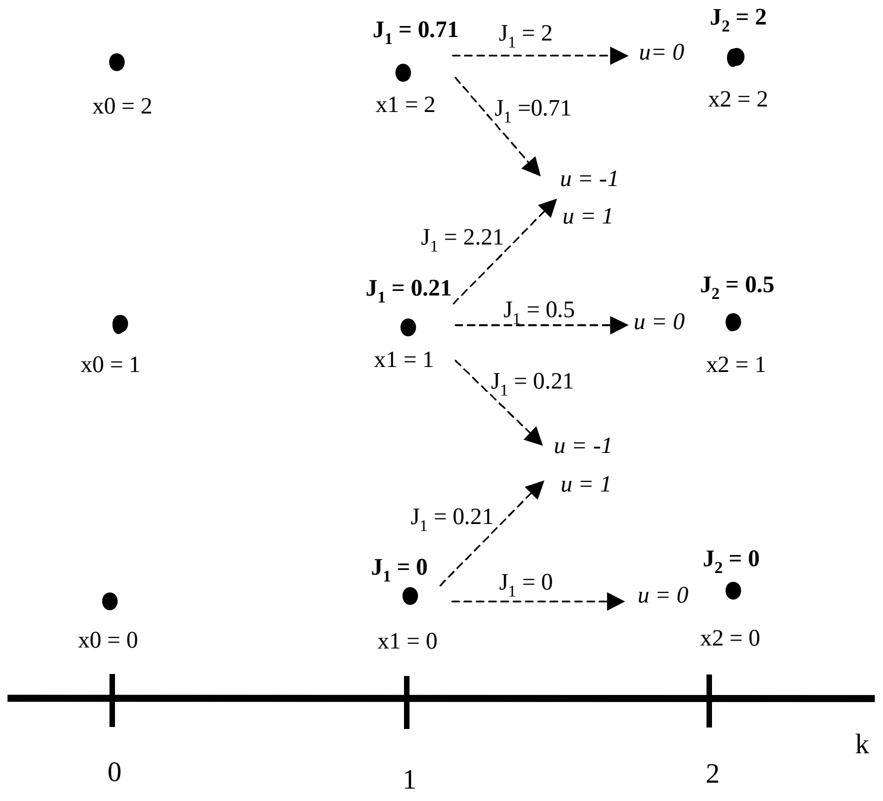

Fig. 31. Dynamic Programming: Computing the cost of states at k = 1.

All numbers are computes to 2 decimal places

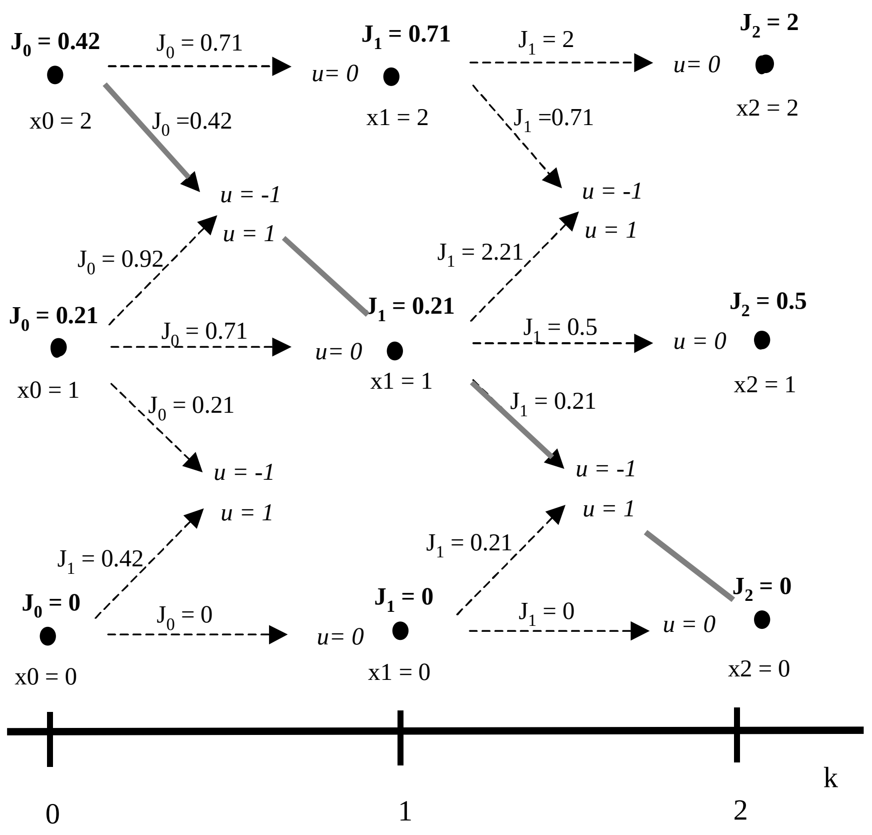

Fig. 32. Dynamic Programming: Total cost of moving between states 0 and 2. All numbers are computed to 2 decimal places

Where N is the number of discrete time steps taken.

Methods which require all possible paths to be calculated have much larger computational demands of the order

$$\sum_{k=1}^{N}(n \times m)^k \tag{7.15}$$

.

C. Dynamic Programming with Interior Points

The interior-point method is used to cope with inequality constraints. Dohrmann and Robinett [8] combined this technique with *sequential quadratic programming* to tackle optimal control problems for constrained, nonlinear systems. The interior points method has been applied to the reorienting of spacecraft in another paper by Hurtado, Eisler, Dohrmann and Robinett [13], where it was compared with other dynamic programming techniques. The DPIP method found all solutions within a very few number of iterations compared with the other methods.

Consider a system with known initial states and quantized controls

$$\dot{x}(t) = f(x(t), u(t)), \quad x(t_1) = x_1 \tag{7.16}$$

$$u(t) = u_i \quad t_i \le t \le t_{i+1} \tag{7.17}$$

This time we work forwards rather than backwards since we know the initial states, not the final states. At each time step 'i', the next states are calculated using 7.18

$$x_{i+1} = g_i(x_i, u_i) \tag{7.18}$$

The augmented performance index is given by

$$\begin{aligned}\Gamma &= \sum_{i=1}^{N-1}[\Gamma_i(x_i,u_i) + c_{iE}(x_i,u_i)^T\,\lambda_{iE} + c_{iI}(x_i,u_i)^T\,\lambda_{iI} && (7.19)\\ &\quad +\Gamma_N(x_N) + c_{NE}^T\lambda_{NE} + cNI^T\lambda_{NI} && (7.20)\end{aligned}$$

where $\Gamma_i(x_i,u_i)$ is the performance index and the equality and inequality constraints have been appended to the performance index by use of Lagrange multipliers λ_{iE}, λ_{iI}, λ_{NE} and λ_{NI}.

This form is called the *Lagrangian function* and is denoted P1. A similar problem P2 approximates P1, using a second order Taylor Series expansion. P2 is expressed as a set of nominal values (steady state conditions) with perturbed values of state, controls and Lagrange multipliers. The state update equations, and constraints are linearized about the nominal values. P2 is generally easier to solve than P1. The discrepancy between the solutions of P1 and P2 is evaluated and based upon the error, P2 is updated to produce a new approximation. This procedure is iterated until the solution to the original nonlinear problem is found. The procedure is illustrated in Fig. 33. This type of technique is known as *Sequential Quadratic Programming.*

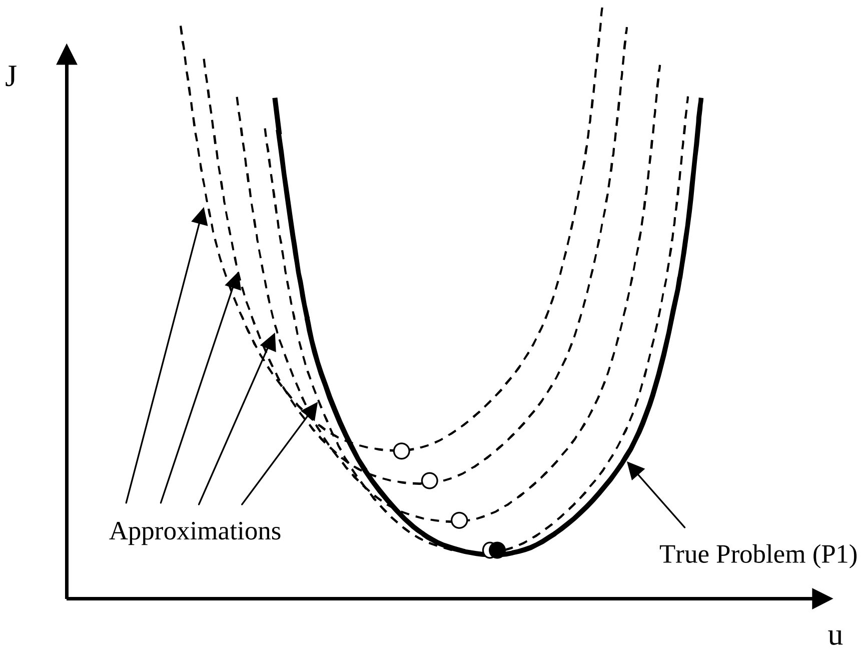

Fig. 33. Quadratic approximations converging on solution to real problem

D. DPIP Toolbox

The DPIP toolbox was developed for Matlab implementation by Robinett, Dohrmann and Hurtado [12]. It requires 5 files to run.

1. STATE EQUATIONS, which will be integrated numerically using a Runge-Kutta method. I used the kinematic equations for the AUV and treated the generalized speeds as the controls.

2. COST FUNCTION calculates the performance index J and returns the first and second partial derivatives of the performance index with respect to the controls and the states.

3. INEQUALITY CONSTRAINTS returns the constraints, the first and second partial derivatives of the inequality constraints with respect to the controls and the states. The only inequality imposed was that the motion must be forwards i.e. $u_3 \geq 0.01$

4. EQUALITY CONSTRAINTS returns the constraints, the first and second partial derivatives of the equality constraints with respect to the controls and the states. The equality constraints for the AUV are the state equations and the final position/pitch of the robot. The heading was not constrained.

5. MAIN FUNCTION, which calls all the above files. Here we initialize the necessary parameters:

    - the number of states (5)
    - the number of controls (3)
    - the number of nodes used to quantize the parameters (21).

- the initial and final states
- initial and final times (tspan = [0..1] second)
- initial guesses for Lagrange multipliers (zeros)
- initial guesses for control history
  - if turning left $u_1 = 1,\ u_2 = 0\ ,\ u_3 = 1\ \forall t$
  - if turning right $u_1 = -1,\ u_2 = 0\ ,\ u_3 = 1\ \forall t$

6. ITERATE. Each iteration uses sequential quadratic programming to solve that particular problem. The number of subproblems used depends on the initial guesses for the Lagrange multipliers and controls.

E. Results

All simulations were performed with the robot center of mass at the origin (0,0,0). The robot has zero initial heading and pitch angles $\alpha = 0$, $\beta = 0$. All final positions were 1m magnitude from the origin. This was determined by the cooperative algorithm, which has a maximum step size of 1m. The final pitch angle was set to be zero because we want to ensure the robot maximizes its chances of detecting a signal from the source; the heading angle, on the other hand, does not have any prescribed final value.

With such a small step size, all maneuvers investigated were attainable within 30 iterations. Larger step sizes prove more difficult. The most difficult maneuvers for the robot are those with the final position located behind the robot i.e. $(-\Delta x)$ and maneuvers requiring a change in z $(\pm\Delta z)$.

Table XXVIII shows the results for changes in positions along one axis. They can be seen in Fig.34 through Fig.36 The first row $(+\Delta x)$ is simply straight line motion, row 2 is a 180° maneuver and rows 3 and 4 necessitate a 135° turn. The last two rows ($\pm\Delta z$) require a complicated S-shaped maneuver. In row two, the integration time was extended to allow the robot more time to get into position. For all 3 figures, the initial and final positions of the robot are shown in black. For all times in between, the robot body is in light grey and the propeller is shown as dark grey line.

Table XXVIII. Position changes along one axis

Final Position	Number of Nodes	Integration Time (sec)	Number of iterations	Error
$[0,0,1,0,0]$	25	1	3	0
$[0,0,-1,0,0]$	25	3	5	0.01
$[0,0,0,1,0]$	25	1	4	0
$[0,0,0,-1,0]$	25	1	4	0.04
$[0,0,0,0,1]$	25	1	13	0.02
$[0,0,0,0,-1]$	25	1	13	0.02

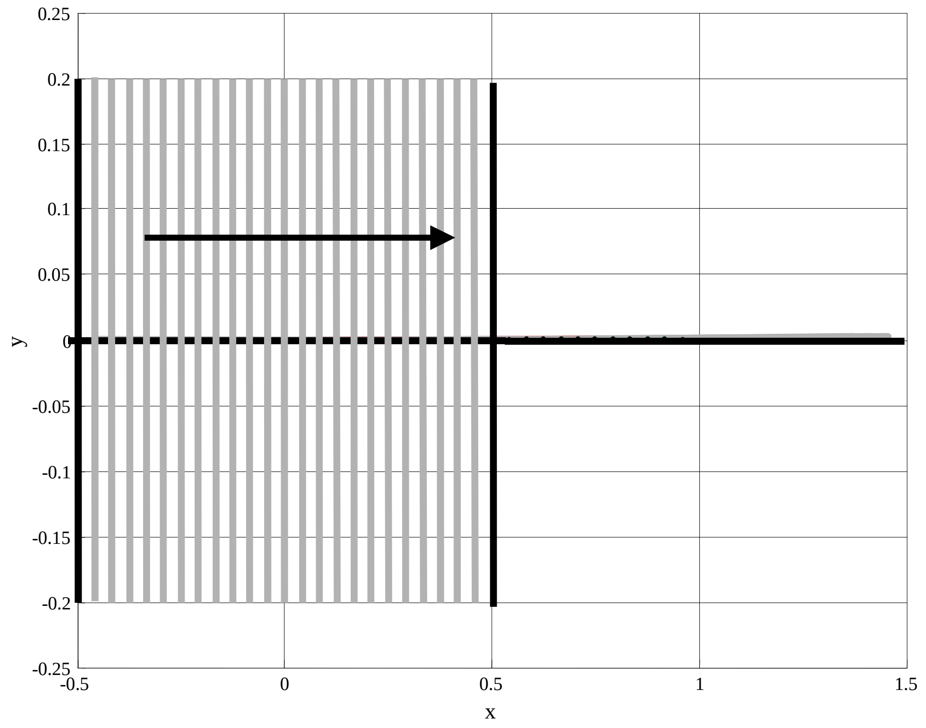

Fig. 34. Robot performing $+1m$ Δx maneuver.

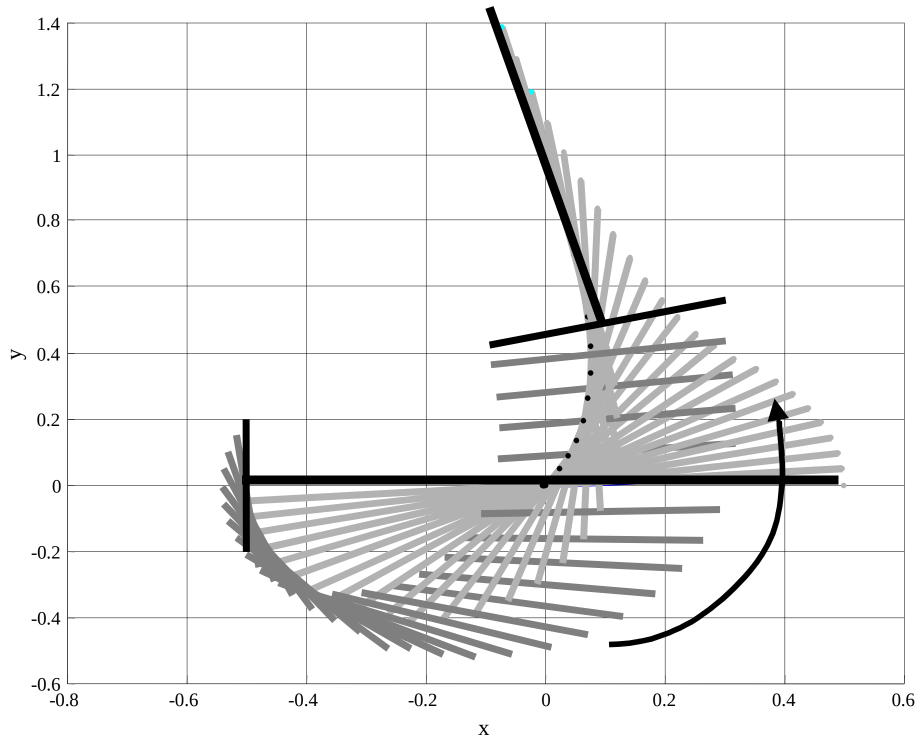

Fig. 35. Robot performing $+1m\ \Delta y$ maneuver.

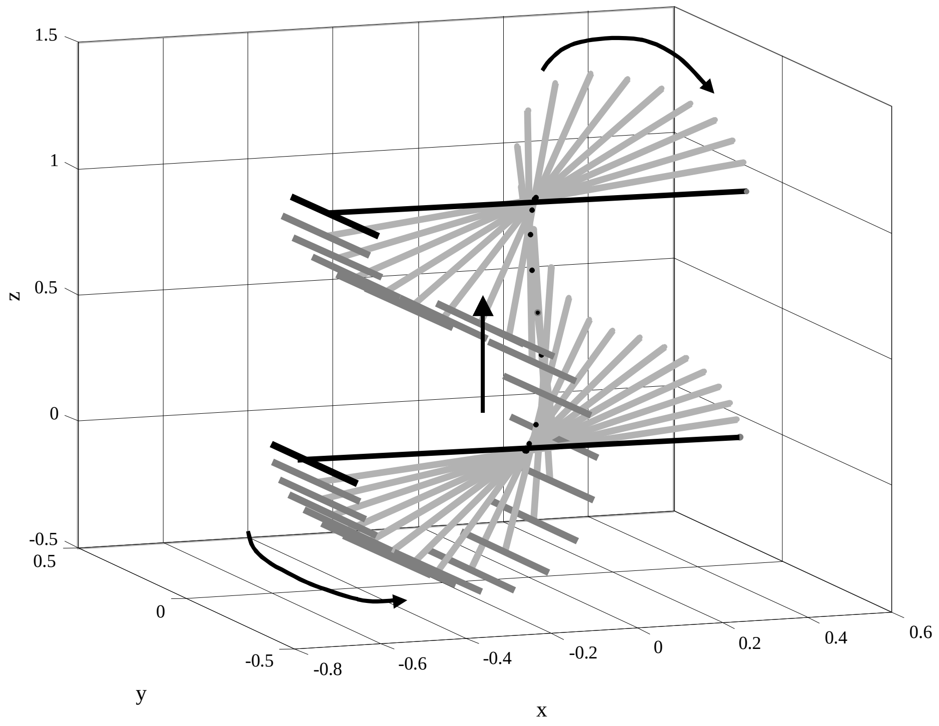

Fig. 36. Robot performing $+1m\ \Delta z$ maneuver.

Table XXIX shows the robot performing maneuvers with position changes prescribed along 2 axes. I have only presented one combination of the two axes in most cases, since turning left produced the same number of iterations as turning right. As in table XXVIII, final positions located behind the robot prolonged the integration time and so I have also presented these. Turns demanding changes in the z-axis required the greatest number of iterations. When both $-\Delta x$ and Δz were combined, it became necessary to change the number of nodes in order to find an optimal path.

Table XXIX. Position changes in two axes

Final Position	Number of Nodes	Integration Time (sec)	Number of iterations	Error
$[0, 0, 0.7, 0.7, 0]$	25	1	5	0.03
$[0, 0, 0.7, 0, 0.7]$	25	1	5	0.02
$[0, 0, 0, 0.7, -0.7]$	25	1	30	0.05
$[0, 0, -0.7, 0, 0.7]$	10	4	17	0.04
$[0, 0, -0.7, 0.7, 0]$	25	3	9	0.04

Finally, table XXX gives the position updates prescribing changes in all three axes. As before, the updates lying behind the initial position of the robot required changes in the number of nodes used.

Table XXX. Position changes in three axes

Final Position	Number of Nodes	Integration Time (sec)	Number of iterations	Error
$[0, 0, 0.57, 0.57, 0.57]$	25	1	7	0.04
$[0, 0, 0.57, -0.57, 0.57]$	25	1	6	0.03
$[0, 0, 0.57, 0.57, -0.57]$	25	1	7	0.04
$[0, 0, 0.57, -0.57, -0.57]$	25	1	10	0.01
$[0, 0, -0.57, 0.57, 0.57]$	10	4	21	0.01
$[0, 0, -0.57, -0.57, 0.57]$	10	4	21	0.01
$[0, 0, -0.57, -0.57, -0.57]$	10	4	21	0.01

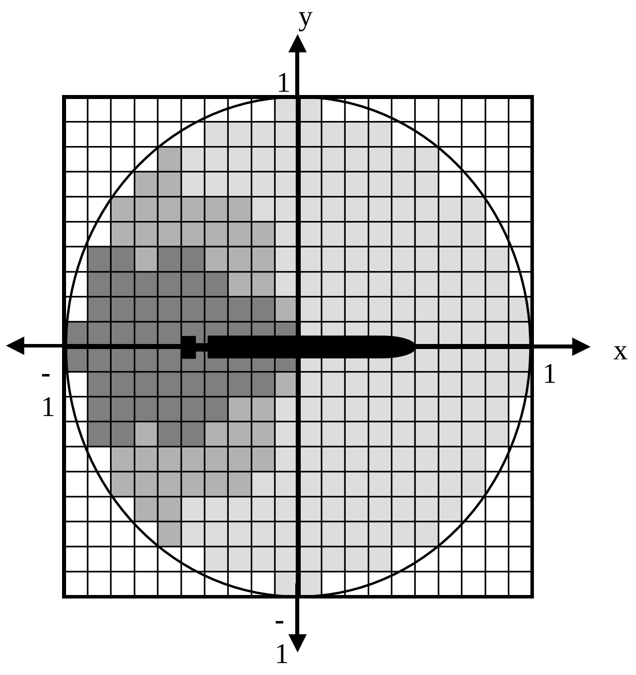

Fig. 37. Grid of integration times needed for each maneuver. Light grey represents 1 second, medium grey represents 2 seconds and dark grey represents 3 seconds

Different maneuvers take different lengths of time. In order to implement this path planning scheme onto the robot, we need to automate the choice for the t_f, the integration time. To simplify matters, we will assume the Ranger will only move in 2 dimensions: x and y. Fig. 37 shows a map of all the possible final positions within a 1m radius of the center of mass. The colors represent the integration times needed to reach them. Everything forward of the center of mass only requires 1 second. The other two regions can be described using parabolas. $x = -y^2$ approximately bounds the light grey zone (2s integration time). $x = -3y^2$ approximately bounds the dark grey zone (3s integration time). These bounds guarantee very fast convergence for the DPIP algorithm. All points can be reached within 6 iterations! This is incredibly fast compared to the variational methods.

F. Conclusions

DPIP apparently requires fewer iterations than fsolve or the genetic algorithm, however, this does not account for the subproblems computed by the interior points. fsolve gives the fastest overall solution, particularly if we use intelligent first guesses. On the other hand, fsolve was not able to impose the necessary constraints and many of the optimal paths involved switching between forwards and reverse motion. In DPIP all motion is forward, as seen from the body frame axis, which is a more efficient use of the propeller and is more realistic.

The DPIP code is written in Matlab; if implemented in Fortran or C, DPIP would be comparable to the fsolve method and almost certainly has the capacity to work in real time on the robots.

CHAPTER VIII

COOPERATIVE CONTROL WITH ROBOTS

To complete the system, DPIP was integrated into the cooperative/exhaustive search algorithm as a means of performing path planning. The results performed in the previous chapter were first extended to perform a series of maneuvers. Simulations of the complete system will convince the reader that the system performs extremely well.

A. The Importance of Heading Angle α

The positions of the robots are expressed in an inertial frame. The most lengthy maneuvers lie in a region behind the robot, therefore it is important to know where the next position is with respect to the tail. More importantly, the direction of turn is determined by where the next position lies. If the direction of turn is not correctly identified, the algorithm may not converge. First, it was important to convert the regions where integration-time transition occurred. A body-fixed reference frame was attached to the center of mass. The regions remain constant with respect to the body frame. A 3-rotation cosine matrix is used to convert all points in the body frame to the inertial frame.

$$\{\boldsymbol{b}\} = \begin{bmatrix} \cos\alpha & \sin\alpha & 0 \\ -\sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix} \{\boldsymbol{n}\} \tag{8.1}$$

First define the change in position for the center of mass: $\Delta x = x_{final} - x_{current}$ and $\Delta y = y_{final} - y_{current}$. When the robot center of mass is at (0,0) the region $\Delta x_b = -(\Delta y_b)^2$ approximately bounds the 2s integration time zone. Using equation 8.1 the area for maneuvers requiring 2s integration is defined as

$$(\cos\alpha\Delta x_N + \sin\alpha\Delta y_N) \leq -(-\sin\alpha\Delta x_N + \cos\alpha\Delta y_N) \tag{8.2}$$

Likewise, when the robot center of mass is at (0,0) the region $\Delta x_b = -3(\Delta y_b)^2$ approximately bounds the 3s integration time. So the area for maneuvers requiring 3s is defined as:

$$(\cos\alpha\Delta x_N + \sin\alpha\Delta y_N) \leq -3(\sin\alpha\Delta x_N - \cos\alpha\Delta y_N). \tag{8.3}$$

B. Simulations with the Robots

Since all code is currently written in Matlab, simulations were only performed with one robot to reduce computational time. This was superimposed on the original particle cooperative method (see Fig. 38). The algorithm was allowed to run ten times to check for consistency. Simulations were repeated starting in each corner of the search area to ensure the heading algorithm worked (Fig. 39. through Fig. 42.). The initial heading angle did not affect the performance of the system. In every case, the robot followed the trajectory traced out by the particle mass.

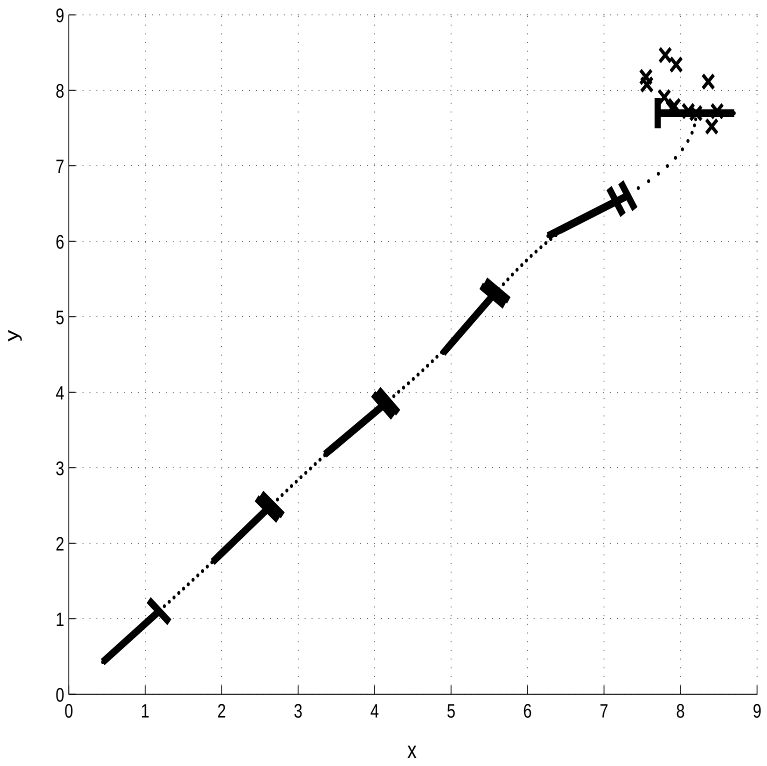

Fig. 38. Motion of robot as it swims in to the source

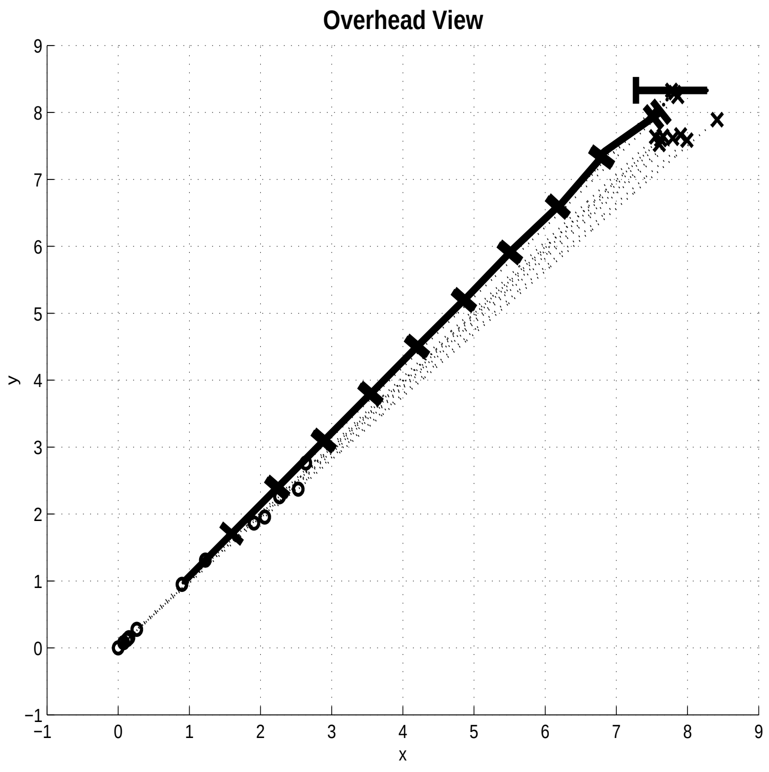

Fig. 39. Motion of robot from top right corner superimposed on the trajectories of the 10 particles.

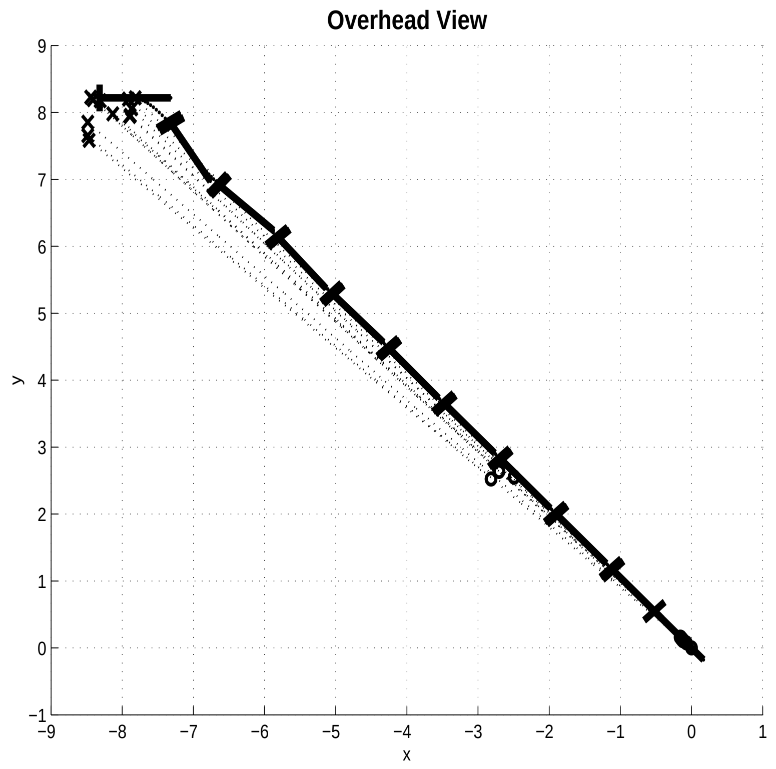

Fig. 40. Motion of robot from top left corner superimposed on the trajectories of the 10 particles.

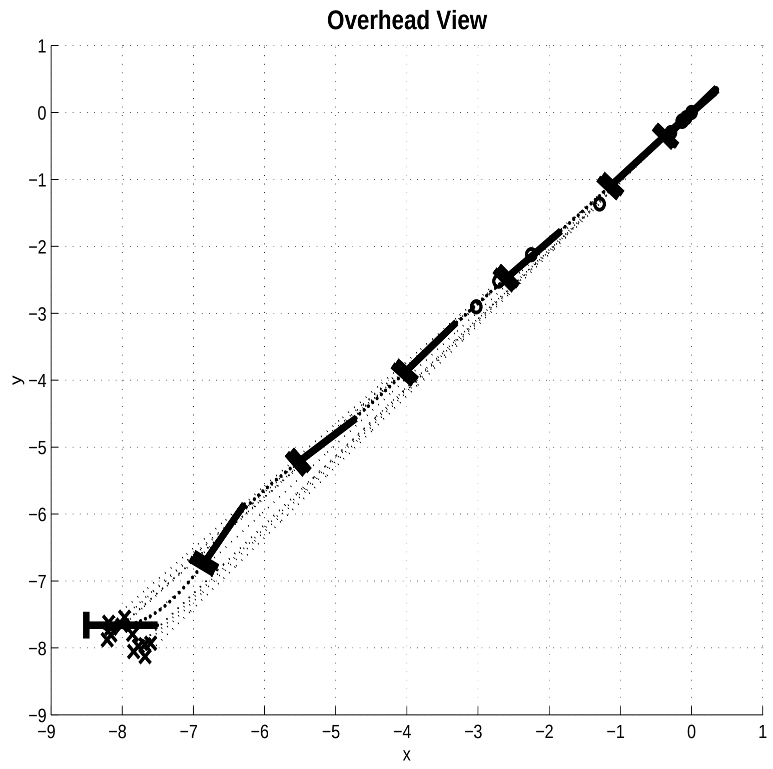

Fig. 41. Motion of robot from bottom right corner superimposed on the trajectories of the 10 particles.

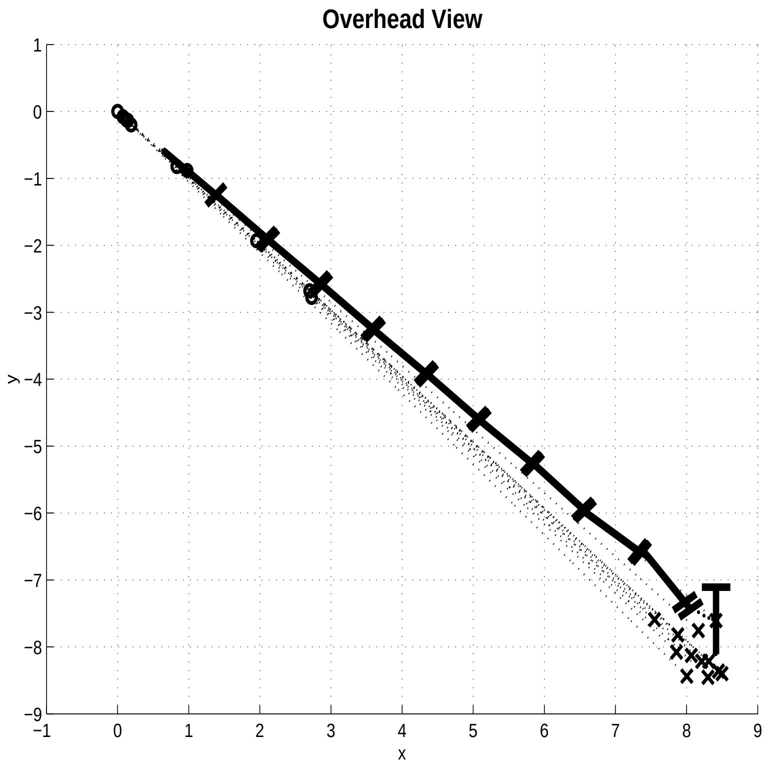

Fig. 42. Motion of robot from bottom left corner superimposed on the trajectories of the 10 particles.

C. Robustness Simulations

The effects of navigational error, position error and sensor noise were investigated in order to fully test the capacities of the DPIP method. For these simulations, the robots were placed in a surround configuration 8 meters from the origin.

Heading error caused no problems to the system. Even with a heading error of 30°, the robot follows his trajectory perfectly. This is clearly seen in Fig. 43.

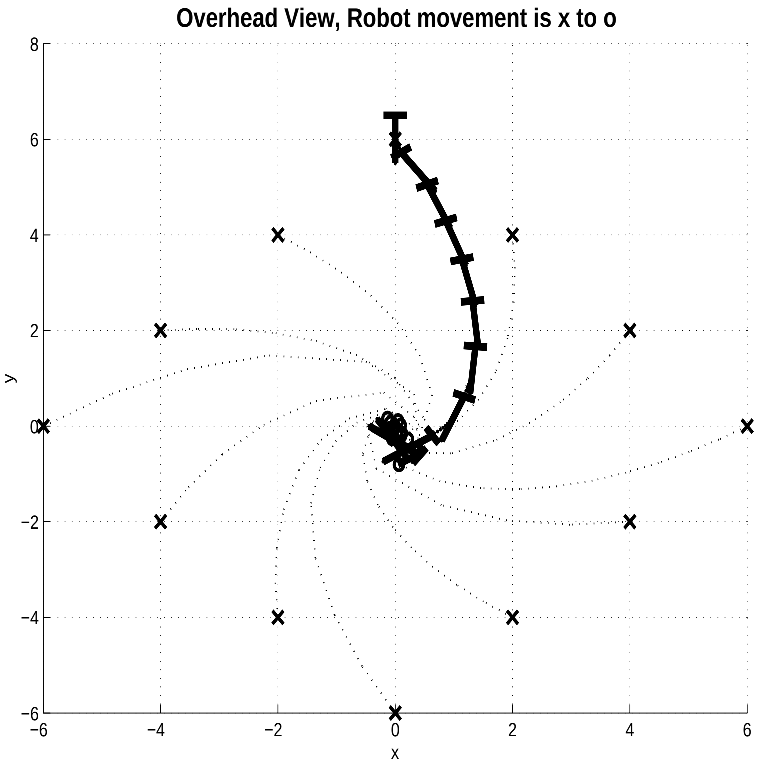

Fig. 43. Full system with 30° heading error applied

Position error of $\pm 0.1m$ was applied in the x and y directions. The robot again followed his trajectory perfectly. Even though the maximum step in the cooperative algorithm remained at 1m, DPIP still managed to reach the desired position at each update. A simulation can be seen in Fig. 44.

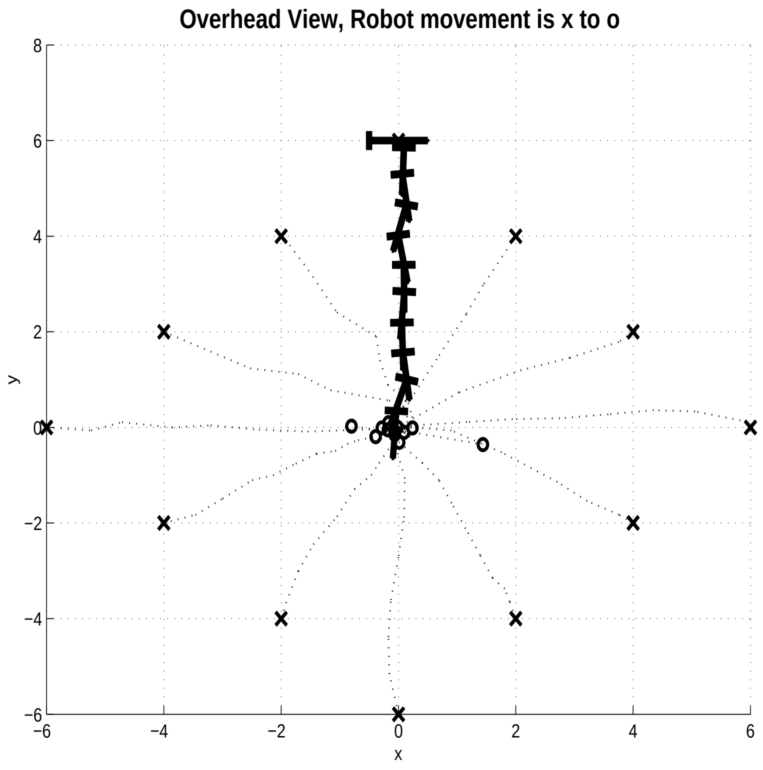

Fig. 44. Full system with $\pm 0.1m$ position error applied in x and y directions

10% sensor noise was introduced into the system. The errors mainly occurred along the z axis, as shown in Fig. 45 and Fig. 46 Again, the paths coincide exactly,

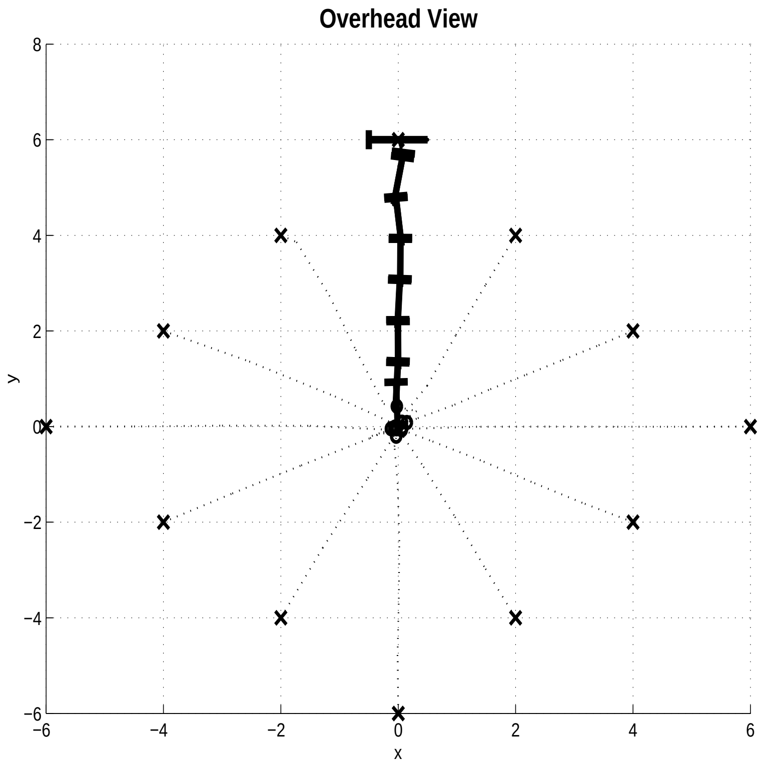

Fig. 45. Full system with ±10% noise (top view)

even with the maximum errors in each case. Further experimentation was not deemed necessary since this will replicate the particle experiments.

D. Conclusions

Dynamic Programming with Interior Points is an excellent path planning method and integrates well with the cooperative algorithm. Test simulations confirm the robot would be able to follow the paths of the particle masses both for an ideal system and in the presence of system errors. The robustness tests also confirm the validity of the

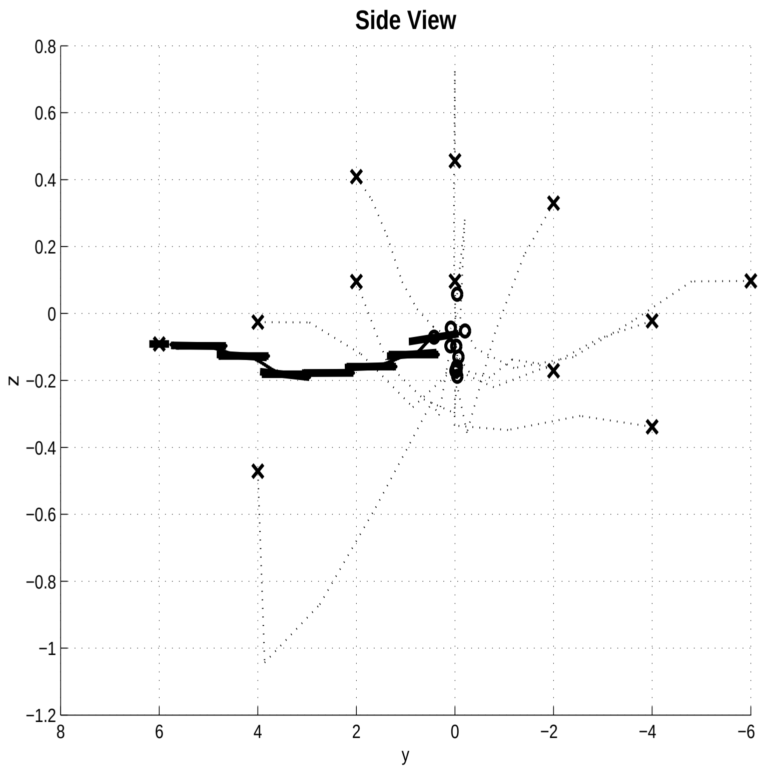

Fig. 46. Full system with ±10% noise (side view)

particle simulations, presented in chapter II. The system shows a lot of promise and is an exciting development for cooperative groups.

CHAPTER IX

SUMMARY

A. Results

1. Particle simulations

- The 2D cooperative algorithm has been converted to 3D and is proven robust to all the errors in the underwater system.
- A number of alterations were made to adapt the 3D algorithm to the robots and the light sensor.
- The communications strategy was simulated and works effectively, although it does significantly extend the time to convergence.

2. Robot simulations and path planning

- The equations of motion were developed using Kane's equations.
- Optimal control was introduced and 3 methods proposed in order to achieve the path planning. These were fsolve and Genetic Algorithms, (which both used variational control to perform the path planning) and Dynamic Programming with Interior Points or DPIP. Each method was required to move the robot to any point within a 1m sphere of the center of mass.
- fsolve performed well, and found solutions even with randomized initial guesses for the costates. A more intelligent guess was proposed and this dropped the number of iterations dramatically. This was the most efficient method, but it has no way of imposing constraints and therefore could not guarantee a realistic path.

- The genetic algorithm provided very disappointing results. It is unsuited to path planning using this method. (Other groups have successfully applied different methods using genetic algorithms).
- DPIP converged with few iterations and had the advantage of being able to cope with both equality and inequality constraints. A strategy was applied to enable the robot to determine the integration time and therefore the speed of the vehicle for each 2D maneuver. The effects of the heading angle were also discussed.
- DPIP was subsequently incorporated into the cooperative algorithm and exactly followed the paths traced out by the original particle simulations. The full system was tested for robustness and again matched the particle simulations.

REFERENCES

[1] S. R. Vadali, S.S. Vaddi, and K.T. Alfriend, "An intelligent control concept for formation flying satelites," *International Journal of Robust and Nonlinear Control*, vol. 12, pp. 97–115, 2000.

[2] L. L. Whitcomb and D. R. Yoerger, "Preliminary experiments in the model-based dynamic control of marine thrusters," in *Special Session on Underwater Robotics, Sensing, Navigation and Control, IEEE International Conference on Robotics and Automation*, Minneapolis, MN, September 1996, vol. 3, pp. 2166–2173.

[3] P. Ogren, E. Fiorelli, and N. Ehrich Leonard, "Formations with a mission: Stable coordination of vehicle group maneuvers," in *Proc. Symposium on Mathematical Theory of Networks and Systems, University of Notre Dame*, August 2002, Located: http://graham.princeton.edu/auvlab/.

[4] J.E. Hurtado, R.D. Robinett III, C.R. Dohrmann, and S.Y. Goldsmith, "Decentralized control for a swarm of vehicles performing source location," *Journal of Intelligent and Robotic Systems*, 2003, Submitted.

[5] A. Ogden, "Genetic algorithms used as a distributed search method for a team of mobile robots," M.S. thesis, University of New Mexico, Albuquerque, 2001.

[6] R. Austin, "Deployable flight incident recorders: Avoiding, not just surviving, a crash," http://www.aviationtoday.com/reports/avionics/previous, March 2003.

[7] H. Baruh, *Analytical Dynamics*, New York:WCB/McGraw-Hill, 1999.

[8] C.R. Dohrmann and R.D. Robinett III, "A dynamic programming method for a constrained discrete-time optimal control," *Journal of Optimization Theory and Applications*, vol. 101, no. 2, pp. 259–283, 1999.

[9] A. Chipperfield, "Introduction to genetic algorithms," Class Notes, 2000, Personal collection, E. Savage. Contact Dr. Chipperfield directly for copies: A.J.Chipperfield@soton.ac.uk.

[10] S. Mondoloni, "A genetic algorithm for determining optimal flight trajectories," in *AIAA Guidance, Navigation, and Control Conference and Exhibit, Collection of Technical Papers*, August 1998, vol. 3, pp. 1646–1656.

[11] F. L. Lewis and V. L. Syrmos, *Optimal Control*, 2nd Edition, New York: John Wiley and Sons, Inc., 1995.

[12] J.E. Hurtado, *Matlab Code and Manual for DPIP*, Texas A&M University, 2002.

[13] J.E. Hurtado, G.R. Eisler, C.R. Dohrmann, and R.D. Robinett III, "Time optimal maneuvers of a rigid symmetric spacecraft," *Journal of Rockets and Spacecraft*, to appear 2003.

VITA

Elizabeth Louise Savage was born on 15th September 1978 in Nottingham, U.K. She passed her GCSE's at Ipswich High School G.P.D.S.T. in 1995 and her A-Levels at Ipswich School, U.K in 1997. Elizabeth continued her education at Sheffield University, U.K. where she pursued a combined Bachelor's and Master's of Engineering Honors program. She graduated in 2001 from the department of Automatic Controls and Systems Engineering. In 2001, Elizabeth arrived at Texas A&M. She has just completed a Master's of Science in Aerospace Engineering.

Elizabeth may be reached at the following address:

c/o Mrs. Isabelle Savage
62 Murray Road
Ipswich
Suffolk IP3 9AH
U.K.

The typist for this thesis was Elizabeth Savage.