DEPTHX Autonomy Software: Design and Field Results

Nathaniel Fairfield George Kantor Dom Jonak David Wettergreen

CMU-RI-TR-08-09

July 2008

Robotics Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

Abstract

This paper describes the control, navigation, and mapping methods that were developed for a hovering autonomous underwater vehicle that explored flooded *cenotes* in Mexico as part of the DEPTHX project. The cenotes of Sistema Zacatón in Tamaulipas, Mexico, are flooded sinkholes, exotic geological formations with unique water chemistry. The deepest, Zacatón, is over 300m deep. None of the cenotes were mapped before this project. The goals of the DEPTHX project were to construct metrically accurate three-dimensional maps of the cenotes, and to collect environmental data, imagery, water samples, and core samples.

The unknown depths of the cenotes, together with the challenging scientific mission, spurred the development of a robotic vehicle which autonomously (with no communications to the surface) built accurate 3D maps using sonar and collected a variety of scientific data, including core samples from the cenote walls. In this paper, we describe the design, implementation, and testing of the robot software, as well as the results from mapping four cenotes of Sistema Zacatón.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation

One of the greatest human endeavors is the attempt to understand life in all its amazing variety. For most of history this search has been constrained to our home planet, and although there is still much to be discovered here, there is a greater possibility: life elsewhere in the universe.

While we have sent probes to many nearby planets and moons, we have yet to search in a place where we would *expect* life as we know it to thrive – a place with liquid water, biogenic elements such as carbon, and a source of free energy [Chyba and Phillips, 2001]. Based on these conditions, Jupiter's moon Europa is one of the most likely places for extraterrestrial life, since it is believed to harbor an ocean of liquid water below an ice mantle [Greenberg, 2005] [Hussmann et al., 2006], and viable sources of energy from tidal forces, radiation in cracks and fissures in and under the ice sheet, or from hydrothermal vents on the ocean floor [Chyba and Phillips, 2002].

Robots can go places people can't and collect measurements of a broader range of phenomena. The current generation of Mars exploration rovers have demonstrated the great potential of long-term, long-distance, robotic presence for exploration and science.

The challenge of robotic exploration is to make it autonomous. Broadly defined, autonomy is the ability of the robot to make decisions and take actions based on its own representation of the world. Autonomy is necessary in cases where communication is infrequent or low bandwidth, such as on other planetary bodies or underwater. But even with good communications, autonomy can relieve the drudge work of teleoperation and make the robot safer. Autonomy can also also aid in scientific data collection. As autonomous robots begin to produce more and more data, they can rapidly exceed the amount of useful information that can be digested by human scientists. Recent work in robotic exploration has attacked this problem from two directions: first, by synthesizing data into a form that is easier to interpret; and second, by adapting the robot's exploration strategy in order to gather more interesting or useful data.

Both of these strategies are based on the insight that the human scientists are actually fitting the data to some (implicit or explicit) model, and then using this model to determine what to do next. Even a very rudimentary representation of this scientific model can provide a framework for autonomous science, which includes the robot's exploration strategy, its sampling strategy, and

even a selective data return strategy.

At a lower level, autonomy assumes that the robot has an accurate representation of its location. The localization problem can be further decomposed into the problem of representing a map of the world, together with the problem of knowing the robot's location within that map. Constructing this map from scratch, especially with sparse sensors, and especially when the map is fully three dimensional (as is the case in underwater caves), is a contribution of this research.

Autonomy also assumes that the robot can reliably control its own actions. Again, this problem becomes more interesting in a three dimensional environment. Beyond basic motion control, there is also the question of more delicate manipulation tasks in which the robot interacts with the environment, for example to collect samples.

In this report, we present the details of the design and operation of an autonomous underwater vehicle that demonstrates a solid foundation of localization and control, and takes the first few steps towards the goal of science autonomy – the ability of the robot to tailor its activity based on scientific objectives. We show the results from the robotic exploration of a system of flooded sinkholes that present a challenging, interesting, and scientifically significant environment.

Sistema Zacatón is a chain of flooded *cenotes*, or sinkholes (Figure 1.1). Zacatón, the world's deepest known limestone sinkhole, is a water-filled cavern that is over 300 m deep. The depths of Zacatón are geothermally heated, have a high sulfur content, lack sunlight or dissolved oxygen, and are an ideal place to search for exotic microbial life Gary [2002]. The robotic exploration and search for microbial life in Zacatón is an analog mission for the search for life in the liquid water ocean beneath the frozen surface of Europa.

The DEPTHX (DEep Phreatic THermal eXplorer) project was a three-year (2004-2007) NASA ASTEP project with the primary objective of using an autonomous vehicle to explore and to characterize the unique biology of the Sistema Zacatón. The DEPTHX vehicle (Figure 1.3) is a hovering autonomous underwater vehicle (AUV) designed to explore flooded caverns and tunnels and to collect environmental data and samples from the water column and cavern walls. Stone Aerospace led the design and fabrication, and Carnegie Mellon created the software and operated the vehicle. Southwest Research Institute designed and built the scientific payload; the Colorado School of Mines provided scientific guidance and performed sample analysis. During three field sessions in Mexico, the vehicle explored four cenotes, creating the first maps of these flooded caverns and collecting precisely localized scientific samples for laboratory analysis.

This paper describes the technical details of the onboard software systems of the DEPTHX vehicle, which performed fully autonomous exploration and sampling missions with no human interaction. The paper is organized as follows: we begin with a high-level overview the vehicle in Section 1.2.2. We then discuss each of the main software systems. First is the control system in Section 2, which directed the vehicle's movement through the water and the delicate task of approaching the walls to take photographs and collect samples. Next, we describe the simultaneous localization and mapping (SLAM) system in Section 3, which uses the information from the onboard sensors to build accurate maps of the cenotes, and simultaneously estimates the vehicle's position within the map. Finally, we describe the executive system in Section 4, which directs the vehicle through the various branches of the mission plan. We finish with conclusions and lessons learned in Section 6.

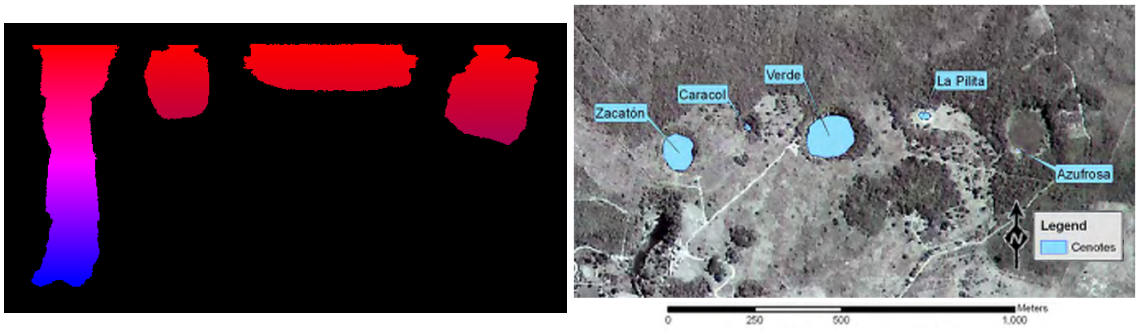


Figure 1.1: Left: a profile view of the cenotes of Sistema Zacatón (distances between cenotes are not to scale) as mapped by the vehicle – maximum depths are Zacatón: 318m; Caracol: 82m; Verde: 48m; La Pilita: 117m. Right: an aerial photograph of the Sistema Zacatón study area. Courtesy Robin Gary [Gary, 2005]

1.2 Approach

1.2.1 Concept of Exploration

We developed the concept of *iterative exploration* in order to reduce the risk of exploring unknown environments with an experimental (and expensive) autonomous vehicle. The primary concern was always to return the vehicle safely to the surface with a safety margin of battery power after every mission. While the vehicle could usually be trimmed to be slightly buoyant, the buoyancy changed significantly with depth – and initially we did not know by how much! Further, when the vehicle was under overhangs, simply floating upwards would trap the vehicle against the ceiling. The use of the fiber optic tether was also a trade-off, since it allowed us to monitor, debug, and control the vehicle, but also increased the risk of fouling on some obstacle.

Iterative exploration involves several intuitive phases that corresponded to our knowledge of the cenote and our confidence in the vehicle's autonomous capabilities (Figure 1.2). Loosely, these phases consist of getting a first look from a safe distance, building a detailed map with accurate spatial registration, and then expanding the frontier of exploration.

Note that once we knew that a cenote was safe, we could have the vehicle explore the cenote without a priori information and without risking loss or damage to the vehicle.

Accordingly, the first step in the exploration of a cenote was the central axis dive, in which the vehicle descended to with a few tens of meters of the bottom, and then ascended, all roughly on the central axis of the cenote. Once we had examined the resulting map, we performed further selective dives in order to fill in gaps in the map.

After just a few dives, we used the complete map to select sites on the walls that were both safe from dangerous overhangs or obstacles, and scientifically promising. The vehicle then approached the walls and took pictures. After we reviewed the pictures, as well the close-up sonar data, final sites were selected for water and solid sampling.

By the end of the project, all of these steps were automated to the point that the robot could explore and sample a cenote without any human intervention.

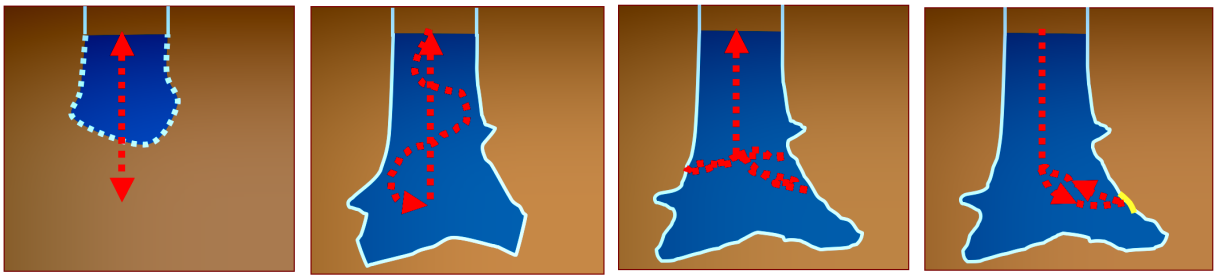


Figure 1.2: Iterative exploration: center drop, exploration, wall inspection, wall sampling.

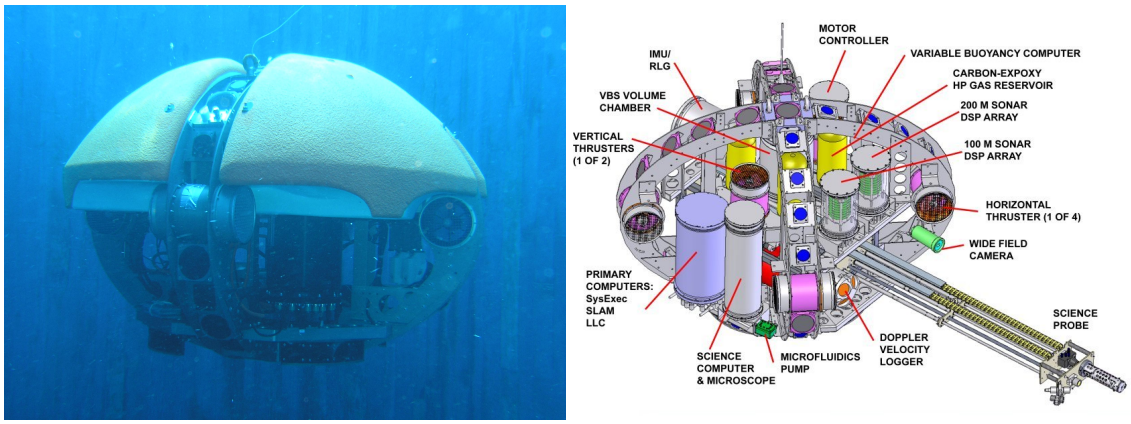


Figure 1.3: The DEPTHX AUV, about 2 m wide and 1.5 m high.

1.2.2 Vehicle

This section provides a brief description of the vehicle and its sensing capabilities. For a more detailed description of the mechanical and electrical design, and of the science payload, see Stone et al. [2005].

The DEPTHX vehicle is a hovering AUV that has been specifically designed for exploration of flooded caverns and tunnels. It has an ellipsoidal shape that measures approximately 1.5 meters in height and 1.9 meters in both length and width (Figure 1.3). Its dry mass is 1500kg. Four large pieces of syntactic foam are mounted on the top half of the vehicle, passively stabilizing the vehicle roll and pitch. The vehicle can control its motion in the three translation axes and its yaw using six thrusters driven by brushless DC motors. The diving and cruising speed of the vehicle is about 0.2 meters per second under thruster power . The vehicle is powered by two 56-volt lithium-ion batteries with a total capacity of 6.2 KWh, enough to power the vehicle during a four-hour exploration mission. The vehicle is depth rated to 1000 m.

1.2.3 Navigation Sensors

The vehicle has a full suite of underwater navigation sensors, including a Honeywell HG2001AC inertial measurement unit (IMU), two Paroscientific Digiquartz depth sensors, and an RDI Naviga-

tor 600kHz Doppler velocity logger (DVL). The specifications for the INS are roll/pitch: 0.2° 2σ, yaw: $0.4^\circ 2\sigma$, for the DVL velocities 0.3 cm/s 1σ, and for the depth sensors 0.01% of full range (10 cm for our 1000m rated sensor). The two depth sensors are tared, or zeroed with respect to atmospheric pressure, at the start of each day. The DVL is mounted to the front of the vehicle facing forward and tilted down 30 degrees from horizontal, a nonstandard configuration for this instrument. The usual DVL configuration points straight down so that it can achieve lock on the ocean floor. In our application, it is difficult to predict the relative direction to surfaces useful for DVL lock. Near the surface, Zacatón was known to be a chimney with a diameter of approximately 80 meters [Fairfield et al., 2005], so the forward-looking configuration is intended to allow the DVL to lock on to one of the vertical walls in most situations. This configuration can cause the DVL to lose bottom lock in more open waters, such as a large lake with a flat bottom. Loss of bottom lock can also occur at extremely short ranges, or when passing over highly irregular terrain.

For our purposes, the roll, pitch, and yaw measurements provided by the IMU and the depth sensor measurements are accurate enough to be considered absolute measurements of those quantities. The task of determining the location of the vehicle is then reduced to the two dimensional problem of estimating its position in the horizontal plane.

1.2.4 Mapping Sonars

For mapping, the vehicle an array of 54 sonars with 2° beam-width. This array is in the shape of three great circles (Figure 1.3), a configuration that was selected after studying the suitability of various sonar geometries for the purposes of SLAM [Fairfield et al., 2005]. The sonars have long ranges (some 100m and others 200m) and the accuracy of the range measurements is fairly high (about 10cm), however the low resolution, slow update rate, and sparse point density makes the mapping problem significantly more difficult than it is with ranging sensors like a laser scanner that provide fast, accurate, high-resolution range images.

1.2.5 Science Payload

The science payload for DEPTHX was designed, built, and programmed by the Southwest Research Institute (SwRI). It included a robotic arm that could be extended about 2 m to bring a mechanical core sampling device into contact with the wall. The corer extracted a cylindrical sample weighing a few grams. Two NTSC cameras, one on the end of the arm and the other on the vehicle body, captured images of sample locations. The payload could also collect up to five individual 1 liter water samples using an intake on the sampling arm.

1.2.6 Software Architecture

The DEPTHX software architecture is heavily influenced by our successful terrestrial rover architectures (Figure 1.4) [Wettergreen et al., 2005b],[Wettergreen et al., 2005a] that support similar robotic exploration under resource constraints.

The DEPTHX architecture organizes primitive and complex behaviors through distributed communicating modules. Each module is a process with one or more threads deployed onto the

various computers onboard the robot. A common communication architecture structures a message dictionary and allows any module to publish and subscribe to messages. The modules subscribe to the message that provide their necessary input information and publish (output) state information, either sensor observations, interpreted data or models, and status information needed by others (although each module is blind to its subscribers).

The main classes of modules onboard the vehicle are the device drivers, the control system, the pose estimation system, and the executive, which interprets the mission plan and monitors the vehicle's status. These modules are briefly introduced here, and described in detail in the following chapters.

At the lowest level, sensors onboard the DEPTHX vehicle provide information about the vehicle state including its depth, velocities and accelerations and, through sonar transducers, the range to obstacles. The state of the robot's actuators, thruster speed and torque, as well as ancillary measurements like temperature, energy, and internal humidity are also recorded.

The Sensor Interface module transforms data from the IMU, DVL, and depth sensors into the correct reference frame for use in pose estimation. The Sonar Interface module filters sonar ranges to produce measurements to reflective objects in the environment.

The Pose Estimation module produces an estimate of the vehicle location using several methods – it selects which method is most appropriate given the sensor data, the model of the environment and the state of the robot. The methods that it uses include: simple dead reckoning of position using only inertial and depth measurements, localization of the robot using sonar measurements and a prior 3D model of the environment, and SLAM, in which case the Pose Estimator also produces a map along with the pose estimate.

The Executive metes out steps in a dive specification based on the state of the system and its location from the Pose Estimator. In the DEPTHX system there is no onboard planning – rather the Executive chooses actions from a primary or cascading set of contingency plans. Dive specifications are composed into mission plan files that encode, in an interpreted command language, the sequence of behaviors to execute. These behaviors may initiate sensing, navigate to locations, trigger on conditions, perform maneuvers, or acquire samples. The primary plan file is backed up by a number of contingency plans that are triggered on conditions encoded in the currently executing plan description. Actions from the primary or contingent plans resolve down to specific commands to the thrusters by the Controller. The coordination of all thrusters is enabled and coordinated motions are reduced to specific thruster velocities that the Thruster Interface uses for motor control.

The Telemetry Manager subscribes to and logs all message traffic in the system. Similarly the State Observer tracks internal module state and exports this for operator awareness. Lastly the Health Monitor compares system state to desired limits and known fault conditions. It is these faults that the Executive monitors to determine if a contingency plan should be initiated.

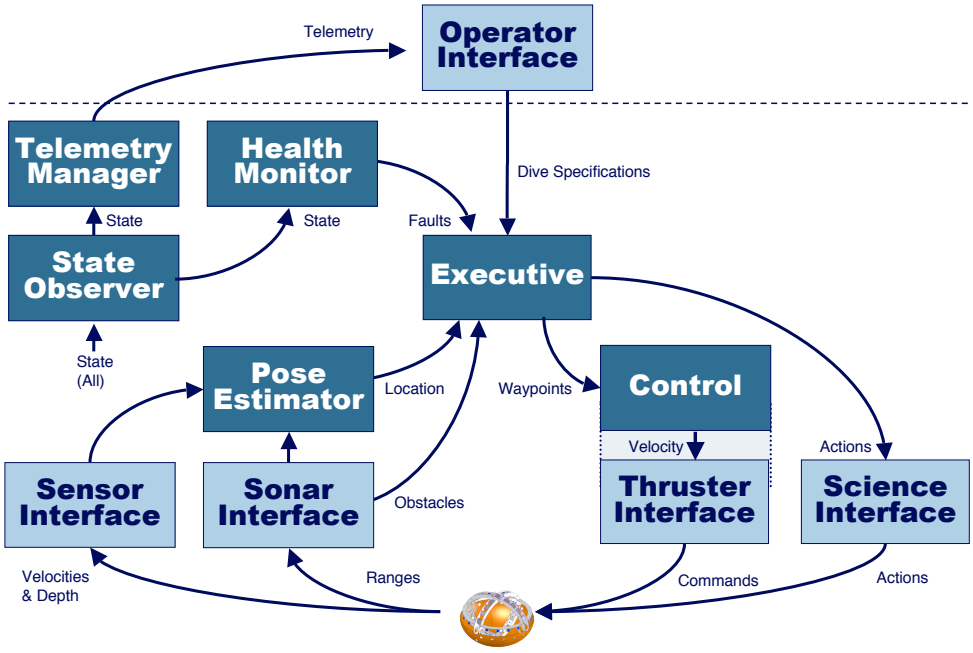


Figure 1.4: DEPTHX software architecture overview

Chapter 2

Control

In this section, we derive the vehicle motion model and then describe our approach to coordinated control of an underwater vehicle.

2.1 Vehicle Model

2.1.1 Coordinate Frames

In this discussion, the *world frame* is defined to be a right-handed inertial reference frame aligned north-east-down with some fixed origin, and the *body frame* is defined to be a right-handed frame rigidly attached to the vehicle with:

- origin at the geometric center of the vehicle
- x axis aligned with the forward direction of the vehicle, (forward is defined by the sampling arm on the science package on the vehicle)
- y axis aligned with the starboard direction of the vehicle
- z axis aligned with the downward direction of the vehicle

In addition to the world frame $\{W\}$ and the body frame $\{B\}$, we also use the IMU frame $\{I\}$, and the DVL frame $\{D\}$. Since the vehicle model deals with the motion of the body frame, transform velocity and acceleration measurements taken from the IMU and DVL into velocities and accelerations of the body frame. The transformation matrix $T_j^i \in SE(3)$ is the homogeneous matrix that transforms points in the i frame into points in the j frame, e.g., T_B^W transforms body frame points into world frame. Superscripts and subscripts are used in the same manner to describe rotation matrices R_j^i and coordinate frame origins p_j^i, i.e.,

$$T_j^i = \begin{bmatrix} R_j^i & p_j^i \\ 0 & 1 \end{bmatrix}$$

We use a similar notation for velocities and accelerations, see Table 2.1.

W	world frame
B	body frame
I	IMU instrument frame
D	DVL instrument frame
T_j^i	4×4 matrix that transforms points in the i frame into points in the j frame
R	3×3 rotation matrix
p	3×1 origin vector
v_j^i	velocity of the origin of the j frame expressed in the i frame
a_j^i	acceleration of the origin of the j frame expressed in the i frame
ω_j^i	angular velocity of the j frame expressed in the i frame
α_j^i	angular acceleration of the j frame expressed in the i frame

Table 2.1: Vehicle Model Notation

We assume that velocity and acceleration are always defined with respect to the world frame. Because of this, velocity and the time derivative of position are not the same thing. For example, p_B^B is the location of the origin of the body frame expressed in the body frame and is always zero. Its derivative $\dot{p}_B^B$ is also always zero. But its velocity with respect to the world frame is not necessarily zero. The quantity v_B^B means the velocity of the origin of the body frame, expressed in the body frame, and it is not necessarily zero. Hence $\dot{p}_B^B \neq v_B^B$!

However, it is true that $\dot{p}_i^W = v_i^W$ for any frame i, so when using differentiation to compute velocities, we first convert into world frame, then differentiate, then convert back. [1]

2.1.2 Finding Body Frame Velocities and Accelerations

The objective is to find the body frame velocities and accelerations $(v_B^B, \omega_B^B, a_B^B, \alpha_B^B)$ from the IMU and DVL measurements $(v_D^D, \omega_I^I, a_I^I, \alpha_I^I)$. The angular velocity and accelerations are the same at

[1] Proof that differentiation correctly yields acceleration:

$$\omega_I^W = R_I^W \omega_I^I$$

so

$$\begin{aligned} \alpha_I^W &= \dot{\omega}_I^W \\ &= \dot{R}_I^W \omega_I^I + R_I^W \dot{\omega}_I^I \\ &= R_I^W \hat{\omega}_I^I \omega_I^I + R_I^W \dot{\omega}_I^I \\ &= R_I^W \omega_I^I \times \omega_I^I + R_I^W \dot{\omega}_I^I \\ &= R_I^W \dot{\omega}_I^I \end{aligned}$$

then converting back to IMU frame we get

$$\alpha_I^I = R_W^I \alpha_I^W = R_W^I R_I^W \dot{\omega}_I^I = \dot{\omega}_I^I.$$

all points on a rigid body. This means that the IMU measurements can be converted to the desired by simply using the coordinate transformations. The fact that velocity and acceleration are free vectors means that the coordinate transformation is just a rotation:

$$\omega_B^B = R_I^B \omega_I^I$$

and

$$\alpha_B^B = R_I^B \alpha_I^I$$

The linear components are more complicated since the velocities and accelerations of points on a rigid body vary with position due to rotational motion. The position of the DVL is given as

$$p_D^W = p_B^W + R_B^W p_D^B$$

Differentiating this to get velocities yields

$$\dot{p}_D^W = \dot{p}_B^W + \dot{R}_B^W p_D^B$$

$$= \dot{p}_B^W + R_B^W \hat{\omega}_B^B p_D^B,$$

where $\hat{\omega}$ is defined to be

$$\hat{\omega} \triangleq \begin{bmatrix} 0 & -\omega_z & \omega_y \\ \omega_z & 0 & -\omega_x \\ -\omega_y & \omega_x & 0 \end{bmatrix}$$

Switching from $\dot{p}$ to v, using $\omega_B^B = R_I^B \omega_I^I$, and the hat function property that $\widehat{R\omega} = R\hat{\omega}R^T$ gives:

$$v_D^W = v_B^W + R_B^W R_I^B \hat{\omega}_I^I \left(R_I^B\right)^T p_D^B$$

Solving for v_B^W then converting to body frame yields the answer:

$$v_B^B = R_D^B v_D^D - R_I^B \omega_I^I \left(R_I^B\right)^T p_D^B$$

To find the acceleration, we start with p_I^W and use the same procedure as above to get:

$$v_B^W = R_I^W v_I^I - R_I^W \hat{\omega}_I^I \left(R_I^B\right)^T p_I^B$$

Differentiating this and converting to body frame yields (after a few steps):

$$a_B^B = R_I^B \hat{\omega}_I^I v_I^I + R_I^B a_I^I - R_I^B \left(\hat{\omega}_I^I\right)^2 \left(R_I^B\right)^T p_I^B - R_I^B \hat{\alpha}_I^I \left(R_I^B\right)^T p_I^B.$$

Everything on the right hand side of this equation is known except v_I^I, which can be found setting the v_B^B that results from the DVL equal to the v_B^B that results from converting the above expression for v_B^W into body frame. The resulting v_I^I is

$$v_I^I = \left(R_I^B\right)^T R_D^I v_D^D + \hat{\omega}_I^I \left(R_I^B\right)^T \left(p_I^B - p_D^B\right).$$

2.1.3 Vehicle State

The state of the vehicle is described by the pair $x = (G, \xi)$. Here $G = T_B^W$ is the pose of the vehicle represented as the homogeneous matrix that is the coordinate transform from body frame to world frame, and $\xi \in \mathbb{R}^6$ are the vehicle velocities. The 4×4 matrix G is of the form

$$G = T_B^W = \begin{bmatrix} R & p \\ 0 & 1 \end{bmatrix}$$

Where the 3×3 rotation matrix R represents the orientation of the body frame relative to the world frame and the 3×1 vector p is the location of the origin of the body frame expressed in the world frame.

The vehicle velocities ξ are represented as by the six-vector

$$\xi = \begin{bmatrix} \omega \\ v \end{bmatrix} = \begin{bmatrix} \omega_x \\ \omega_y \\ \omega_z \\ v_x \\ v_y \\ v_z \end{bmatrix}$$

where ω is the angular velocity of the vehicle expressed in the body frame and v is the linear velocity of the vehicle expressed in the body frame.

2.1.4 Vehicle Kinematics

The kinematics take the form of a differential equation that describes how the pose G changes as a function of the velocities in ξ. (for an introduction see Chapter 2 of Murray et al. [1994]). The differential equation describing the vehicle kinematics is:

$$\dot{G} = G \begin{bmatrix} \hat{\omega} & v \\ 0 & 0 \end{bmatrix}$$

If ξ is constant, the solution to the kinematic differential equation is

$$G(t - t_0) = e^{\Omega(t-t_0)} G(t_0),$$

where e represents the matrix exponential and $G(t_0)$ is the pose at the initial time t_0. The matrix exponential can be computed efficiently using Rodrigues' formula [Murray et al., 1994].

2.2 Vehicle Dynamics

The vehicle dynamics are a generalization of Newton's law $f = ma$. Following Fossen [2002], this gives the differential equation

$$\dot{\xi} = M^{-1} \left(F_{\text{dyn}} + F_{\text{g}} + F_{\text{b}} + F_{\text{drag}} + F_{\text{thrust}} + F_{\text{dist}} \right),$$

M	6×6 mass/inertia matrix with added virtual terms
F_{dyn}	6×1 vector of body frame torques and forces resulting from the rigid body dynamic
F_{g}	6×1 vector of body frame torques and forces due to gravity
F_{b}	6×1 vector of body frame torques and forces due to vehicle buoyancy
F_{drag}	6×1 vector of body frame torques and forces due to drag
F_{thrust}	6×1 vector of body frame torques and forces due to the thrusters
F_{dist}	6×1 vector of body frame torques and forces due to external disturbances

Table 2.2: Vehicle Dynamics Notation

(see Table 2.2 for notation).

We assume that the mass matrix is diagonal, which is roughly true for the round, symmetric vehicle. The mass matrix is then of the form

$$M = \begin{bmatrix} m + m_{vx} & 0 & 0 & 0 & 0 & 0 \\ 0 & m + m_{vy} & 0 & 0 & 0 & 0 \\ 0 & 0 & m + m_{vz} & 0 & 0 & 0 \\ 0 & 0 & 0 & J_{xx} + J_{vx} & 0 & 0 \\ 0 & 0 & 0 & 0 & J_{yy} + J_{vy} & 0 \\ 0 & 0 & 0 & 0 & 0 & J_{zz} + J_{vz} \end{bmatrix}$$

The moments of inertia can be computed theoretically by assuming that the vehicle is a uniform ellipsoid. Similarly, virtual terms can also be computed theoretically [Fossen, 2002]. However in practice, all mass and inertia terms were determined experimentally.

The rigid body forces are

$$\begin{aligned} F_{\text{dyn}} &= \text{ad}^*_\xi M\xi \\ &= \begin{bmatrix} -\hat{\omega} & -\hat{v} \\ 0 & -\hat{\omega} \end{bmatrix} M\xi \end{aligned}$$

The gravitational force is

$$F_{\text{g}} = \begin{bmatrix} x_{\text{cg}} \times R^T \begin{bmatrix} 0 \\ 0 \\ -mg \end{bmatrix} \\ R^T \begin{bmatrix} 0 \\ 0 \\ -mg \end{bmatrix} \end{bmatrix}$$

where x_{cg} is a 3×1 vector describing the location of the vehicle center of mass in the body frame, m is the mass of the vehicle (its the same m that shows up in the mass matrix m), g is the acceleration due to gravity, and R is the rotation matrix part of G.

The force due to buoyancy is

$$F_{\mathrm{b}} = \begin{bmatrix} x_{\mathrm{cb}} \times R^T \begin{bmatrix} 0 \\ 0 \\ \rho V g \end{bmatrix} \\ R^T \begin{bmatrix} 0 \\ 0 \\ \rho V g \end{bmatrix} \end{bmatrix}$$

where x_{cb} is a 3×1 vector describing the location of the vehicle center of buoyancy in the body frame, V is the volume of water displaced by the vehicle, ρ is the density of water, 1000 kg/m^3, g is acceleration due to gravity, 9.8 m/s(^{2}).

The drag forces are simplified because we assume that the vehicle moves slowly enough that the flow is laminar and the drag is well modeled by viscous friction. Formally, we assume that

$$F_{\mathrm{drag}} = -\mathrm{diag}(\gamma_{\mathrm{roll}}, \gamma_{\mathrm{pitch}}, \gamma_{\mathrm{yaw}}, \gamma_x, \gamma_y, \gamma_z)\xi$$

where the γs are the drag coefficients in the six degrees of freedom. The rotational gammas are in units of Nms/rad and the translational gammas are in units of Ns/m.

Thruster Forces The force that results from a command to the ith thruster is

$$F_i = \begin{bmatrix} x_{ti} \times (\tau u_{ti} v_{ti}) \\ \tau u_i v_{ti} \end{bmatrix}$$

where

- x_{ti} is a 3×1 vector describing the location of the ith thruster in the body frame.
- v_{ti} is a 3×1 unit vector describing the direction of positive thrust for the ith thruster, represented in the body frame.
- τ is the coefficient that maps thruster shaft torque to thrust.
- u_{ti} is the control input to the ith thruster. For now it is assumed to be a commanded shaft torque.

The total force due to the action of all of the thrusters is simply the sum of the individual thruster forces:

$$F_{\mathrm{thrust}} = \sum_{i=1}^{N_t} F_i,$$

where N_t is the number of thrusters.

parameter	description	nominal value	units
m	vehicle mass	1360	kg
$m + m_{vx}$	effective mass, forward	3750	kg
$m + m_{vy}$	effective mass, sideways	3750	kg
$m + m_{vz}$	effective mass, vertical	6222	kg
$J_{xx} + J_{vx}$	effective moment of inertia, forward axis	1008	kg·m^2
$J_{yy} + J_{vy}$	effective moment of inertia, sideways axis	1008	kg·m^2
$J_{zz} + J_{vz}$	effective moment of inertia, vertical axis	1114	kg·m^2
x_{cg}	center of gravity	$[0 \quad 0 \quad 0.04]^T$	m
x_{cb}	center of buoyancy	$[0 \quad 0 \quad 0.66]^T$	m
V	displacement	$= \frac{m}{\rho} = 1.5$ (neutral)	m^3
γ_{roll}	laminar viscous drag, roll	280	Nms/rad
γ_{pitch}	laminar viscous drag, pitch	280	Nms/rad
γ_{yaw}	laminar viscous drag, roll	318.2	Nms/rad
γ_x	laminar viscous drag, forward	500	Ns/m
γ_y	laminar viscous drag, sideways	500	Ns/m
γ_z	laminar viscous drag, vertical	777.8	Ns/m
τ	thrust to shaft torque ratio	50	m^{-1}
x_{t1}	position of thruster 1	$[0.6443 \quad 0.6443 \quad 0]^T$	m
x_{t2}	position of thruster 2	$[-0.6443 \quad 0.6443 \quad 0]^T$	m
x_{t3}	position of thruster 3	$[0 \quad 0.3683 \quad -0.3810]^T$	m
x_{t4}	position of thruster 4	$[0.6443 \quad -0.6443 \quad 0]^T$	m
x_{t5}	position of thruster 5	$[-0.6443 \quad -0.6443 \quad 0]^T$	m
x_{t6}	position of thruster 6	$[0 \quad -0.3683 \quad -0.3810]^T$	m
v_{t1}	direction of thruster 1	$[0.7071 \quad -0.7071 \quad 0]^T$	NA
v_{t2}	direction of thruster 2	$[0.7071 \quad 0.7071 \quad 0]^T$	NA
v_{t3}	direction of thruster 3	$[0 \quad 0 \quad 1]^T$	NA
v_{t4}	direction of thruster 4	$[0.7071 \quad 0.7071 \quad 0]^T$	NA
v_{t5}	direction of thruster 5	$[0.7071 \quad -0.7071 \quad 0]^T$	NA
v_{t6}	direction of thruster 6	$[0 \quad 0 \quad 1]^T$	NA

Table 2.3: Control Model Parameters Summary

2.3 Simplified Model

We derived the full 6 DOF model of the vehicle described above, and through simulation we recognized a number of design considerations and mission restrictions that could simplify vehicle dynamics such that the vehicle could be well modeled as a decoupled 4 DOF system with linear dynamics. Since we did this early in the project, we were able to use these discoveries to influence the vehicle design and mission specification, making the job of controlling the actual vehicle significantly easier. The simplifying assumptions/restrictions are

1. maximum vehicle linear velocity due to thrusters is 0.1 m/s
2. roll and pitch dynamics are passively stabilized
3. linear velocity commands in the lateral plane and yaw velocity commands are mutually exclusive – don't turn while moving
4. fluid dynamic effects are well modeled by direction dependent virtual mass and viscous damping terms
5. a low-level "thrust mixer" exists

The ramifications of each assumption and the resulting model are discussed in detail below, but it is important to note that in practice we often exceeded these limits and simplifications. For example, the vehicle's cruising speed was bumped up to 0.2 m/s after we experimentally verified that control performance was not degraded.

Maximum Linear Velocity Imposing a maximum velocity constraint of 0.1 m/s simplifies system dynamics in two important ways. First, the assumptions on the fluid dynamic effects (list item 4 above) hold well for low velocities. Second, the thrusters are approximately linear for low speeds. Specifically, the output thrust is a linear function of the thruster current.

Passively Stable Roll/Pitch The vehicle's center of mass is directly below the center of buoyancy. This effect, together with the damping provided by the water, causes the roll and pitch degrees of freedom to be asymptotically stable about zero, meaning that when the roll or pitch is perturbed away from zero it will converge back to zero. We assume that pitch and roll converge to zero fast enough that we (1) will not have to actively control them and (2) can ignore their effect on the system dynamics. This reduces the number of degrees of freedom we need to consider from 6 to 4.

Mutually Exclusive Linear and Angular Motion We restrict motion in the lateral plane to be sequences of pure linear translations and pure rotations. This restriction causes the dynamics in the three lateral degrees of freedom in the vehicle fixed coordinate system. The dynamics in the vertical (z) direction are also decoupled from the x, y, and θ dynamics due to the fact that roll and pitch are assumed to be zero. As a result, we can model the vehicle as four decoupled systems, one for each of the remaining degrees of freedom (x, y, z, θ).

Simplified Fluid Dynamics At low velocities, the effect that the surrounding fluid has on the vehicle dynamics can be modeled as coming from two sources: virtual mass and drag. We assume that the drag can be modeled as viscous damping, where the damping coefficient is constant for each of the directions. In other words, the forces due to drag in the available vehicle frame

directions are

$$\begin{aligned} F_{dx} &= -\gamma_x v_x \\ F_{dy} &= -\gamma_y v_y \\ F_{dz} &= -\gamma_z v_z \\ F_{d\theta} &= -\gamma_\theta v_\theta \end{aligned}$$

Virtual mass is used to model the mass of fluid that the vehicle must displace as it moves. It is added to the vehicle mass when deriving the equations of motion, and it will be different in different directions. We will denote the virtual masses in the x, y, and z directions as M_x, M_y, and M_z, respectively. We assume that the virtual mass in the θ direction is zero due to the symmetry of the vehicle about the z axis. Likewise, we assume that the symmetry of the vehicle will let the "off diagonal" virtual mass (coupling) terms be zero.

Both virtual masses and drag coefficients should be determined experimentally through a series of system identification experiments that are outlined below. For now we will assume that all drag coefficients and virtual masses are constant, and we hope that this turns out to be the case when the actual vehicles. However, we can allow these "constants" to fluctuate somewhat by using look-up tables to smooth out the nonlinearities. This will be discussed in more detail in the subsequent discussion of the low level controller design.

Thrust Mixing The vehicle has six thrusters mounted at various locations on the vehicle and pointing in various direction. We assume the existence of a "thrust mixer," described below, that will allow us to specify desired torques and forces in the available vehicle fixed directions $(F_x, F_y, F_z, F_\theta)$. From these desired thrusts, the thrust mixer generates the inputs (either desired current or desired shaft torque) to the six individual thrusters necessary to generate the desired torques and forces. This allows us to treat forces and torques applied to the vehicle as inputs to our vehicle model.

Tuning the Thrust Mixer Matrix

The purpose of the thrust mixer is a matrix that maps from desired frame thrust $T = [\, T_{\text{yaw}}, T_x, T_y, T_z \,]^T$ to the thruster shaft torque commands necessary to achieve this thrust. The mixer can be theoretically determined from the placements of the thrusters, but it will be necessarily to tune the mixer experimentally to account for unmodelled vehicle hydrodynamics.

First, we define the following parameters for $i = 1, 2, \ldots, 6$:

- x_{ti} the position of the ith thruster in body frame coordinates. (this can be the position of any point along the line of thrust).
- ν_{ti} a unit vector describing the direction of positive thrust for the ith thruster, represented in body frame.
- τ_i the thrust to shaft torque ratio for the ith thruster (nominal value is 50 from the data sheet).

Given a vector of desired shaft torques $u = [\, u_1, \cdots, u_6 \,]^T$, the resulting vehicle frame thrust $T^* = [\, T_{\text{roll}}, T_{\text{pitch}}, T_{\text{yaw}}, T_x, T_y, T_z \,]^T$ is

$$T^* = \underbrace{\begin{bmatrix} x_{t1} \times \nu_{t1} & \cdots & x_{t6} \times \nu_{t6} \\ \nu_{t1} & \cdots & \nu_{t6} \end{bmatrix} \begin{bmatrix} \tau_1 & 0 & 0 & 0 & 0 & 0 \\ 0 & \tau_2 & 0 & 0 & 0 & 0 \\ 0 & 0 & \tau_3 & 0 & 0 & 0 \\ 0 & 0 & 0 & \tau_4 & 0 & 0 \\ 0 & 0 & 0 & 0 & \tau_5 & 0 \\ 0 & 0 & 0 & 0 & 0 & \tau_6 \end{bmatrix}}_{\triangleq A^*} u$$

We invert this relationship in order to map vehicle frame thrusts to desired shaft torques. Since we only care about the yaw, x, y, and z directions, define T to be the vector composed of the last 4 elements of T^* and define A to be a matrix composed of the last for rows of A^*. Then we use the pseudo-inverse to get the mixing matrix $\mathcal{T}_{\text{mix}}$:

$$u = \underbrace{A^T \left(A A^T \right)^{-1}}_{\triangleq \mathcal{T}_{\text{mix}}} T$$

2.3.1 Resulting Vehicle Model

Using Newton's second law of motion together with the drag coefficients, virtual masses, and the assumptions outlined above yields that following equations of motion in the vehicle fixed frame:

$$(m + M_x)\dot{v}_x = F_x + F_{dx} = F_x - \gamma_x v_x$$

$$(m + M_y)\dot{v}_y = F_y - \gamma_y v_y$$

$$(m + M_z)\dot{v}_z = F_z - \gamma_z v_z$$

$$I_z \dot{v}_\theta = F_\Theta - \gamma_\theta v_\theta$$

where m is the mass of the vehicle and I_z is the moment of inertia about the z axis. These are each first order ordinary differential equations, and as such they can be converted into transfer functions, e.g., for the x direction:

$$\tilde{H}_x(s) = \frac{1}{(m + M_x)s + \gamma_x}.$$

Note that by making the substitution

$$r_x = \frac{1}{\gamma_x}$$

and

$$\tau_x = \frac{m + M_x}{\gamma_x}$$

we get the standard first order linear transfer function

$$\tilde{H}_x(s) = \frac{r_x}{\tau_x s + 1}.$$

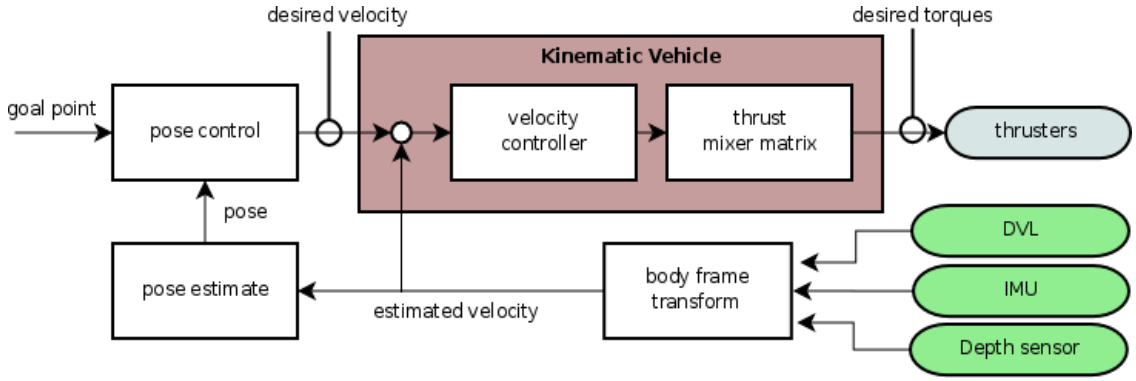


Figure 2.1: Control diagram, showing the control flow through the components described in the text.

If we redefine our input to be $u_x = \frac{F_x}{r_x}$ on the F_x then the net transfer function from u_x to v_x is

$$H_x(s) = \frac{1}{\tau_x s + 1}.$$

This system has the following response when a step input of magnitude u_x is applied:

1. $v_x(t)$ exponentially approaches a steady state value u_x.
2. after τ_x seconds, v_x will have reached 63% of its final value. After $4\tau_x$ seconds, will have reached 98%.

The parameters τ_x and r_x can be determined using experiments that will be described in more detail later. The input u can be easily realized by inserting a scaling factor of $\frac{1}{r_x}$ into the thrust mixer for each degree of freedom.

Using the same reasoning, we can obtain $H_y(s)$, $H_z(s)$, and $H_\theta(s)$. Defining $v = [v_x\, v_y\, v_z\, v_\theta]^T$ and $u_t = [u_x\, u_y\, u_z\, u_\theta]^T$, we can write the model as a single 4-input, 4-output system:

$$Q(s) = H(s)U_t(s),$$

where

$$H(s) = \begin{bmatrix} H_x(s) & 0 & 0 & 0 \\ 0 & H_y(s) & 0 & 0 \\ 0 & 0 & H_z(s) & 0 \\ 0 & 0 & 0 & H_\theta(s) \end{bmatrix}$$

2.4 Control

Using the simplified model derived above, the relationship between vehicle thrust and vehicle velocity can be represented as four decoupled first order linear systems (one for each remaining degree of freedom). The kinematic relationship between velocity and position is still coupled and nonlinear. This leads naturally to a two-level controller: a lower level velocity controller that uses

four independent PID loops to achieve velocity tracking and a higher level pose controller that issues velocity commands to the lower lever in order to achieve desired robot pose (Figure 2.1). The pose controller must handle the nonlinear, coupled kinematics, but its job is simplified by the fact that it does not need to take robot dynamics into account.

2.4.1 Velocity Control

The objective of the velocity controller is to create a system that accepts desired velocities as inputs and outputs thruster torque commands to make the actual vehicle velocities match the desired velocities. This allows higher levels of planning and control to treat the system as a kinematic vehicle; the higher levels specify desired velocities and rely on the low level control system to implement them.

The velocity controller performs two basic functions: it uses velocity feedback to convert a vehicle frame velocity command into the vehicle frame thrust needed to track that velocity and it implements a mixing table in order to convert the vehicle frame thrust command into the necessary shaft torque [2] commands to each of the individual thrusters.

Velocity feedback is implemented in four independent loops, one for each of the vehicle's four degrees of freedom. Each loop contains an experimentally tuned PI controller. This structure assumes that the components of vehicle frame velocity are not coupled by the dynamics of the vehicle, an assumption which not true: the forward and sideways velocity components of the vehicle will be highly coupled when the vehicle simultaneously rotates and moves in the lateral plane. Hence we can enforce the decoupled assumption by simply avoiding these type of motions, a restriction that is compatible with the slow, deliberate types of missions that the vehicle will undertake. In practice, however, the controller performs well even when motions with some such coupling are executed.

The thrust mixer maps the vehicle frame thrust command vector $F_v = [F_\omega,\, F_x,\, F_y,\, F_z]^T$ into a vector of n individual thruster commands $T = [\tau_1,\, \tau_2,\, \tau_3,\, \ldots,\, \tau_n]^T$, where n is the number of thrusters. It is implemented as a matrix multiplication, i.e., $T = MF_v$, where M is computed as follows. First, let A be the $6 \times n$ matrix whose ith column is given by

$$A_i = \begin{bmatrix} p_i \times D_i \\ D_i \end{bmatrix},$$

where p_i is a vector describing the location of the ith thruster in the vehicle frame, and D_i is a vector describing the positive thrust direction of the ith thruster in the vehicle frame. Now let B be the $4 \times n$ matrix that is the bottom four rows of A. B is the matrix that maps the individual thruster thrusts T into the resulting vehicle frame thrust vector F_v. Assuming that the rows of B are linearly independent, M can then be found by taking the pseudo-inverse of B:

$$M = B^T \left(BB^T\right)^{-1}.$$

Note that in the nominal case, the number of thrusters is $n = 6$. However, this formulation allows the mixing matrix to easily be recomputed in the event of thruster failure.

[2] The relationship between shaft torque and thrust is very nearly linear, and we rely on the DriveBlok^{TM} controller produced by MTS Systems Corp to implement the desired shaft torque on the brushless DC thruster motors.

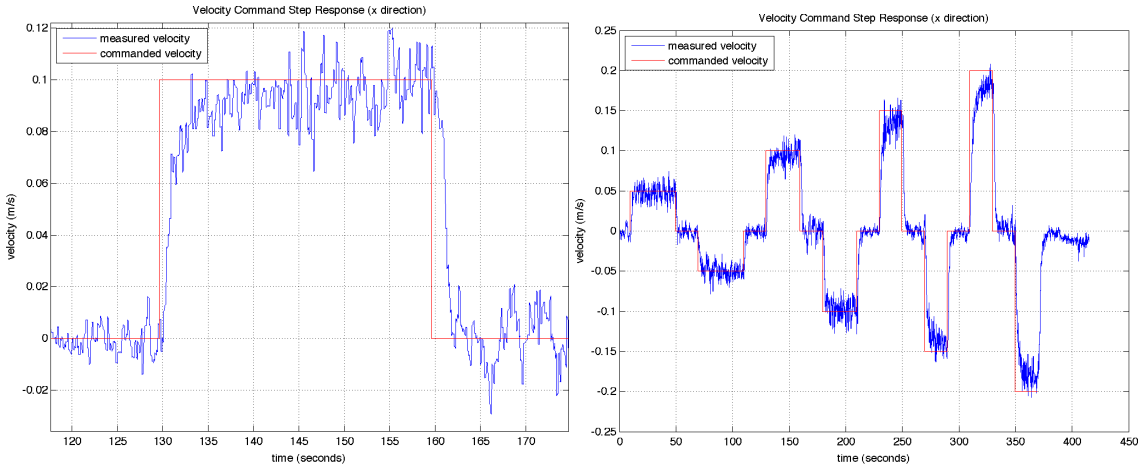


Figure 2.2: Velocity tracking performance of the velocity controller for various commanded velocity pulses. The duration of the leading and falling edge transients is on the order of 4 seconds.

Signal Flow

The signal flow for the thruster control system will is as follows. Measurements v_x, v_y, v_z (from DVL) and v_θ (from IMU) will be subtracted from desired velocity $u_d = (u_{dx}, u_{dy}, u_{dz}, u_{d\theta})$ to produce and error signal $e = (e_x, e_y, e_z, e_\theta)$. The error signals will be passed through the PID controllers to produce the input signal $u = (u_x, u_y, u_z, u_\theta)$. The input signal u will be passed into the thrust mixer which will scale the inputs as discussed in Section 2.3.1 and mix the signals to generate the desired shaft torque to be applied to each of the six thrusters, $T = (T_1, T_2, T_3, T_4, T_5, T_6)$. T is then passed to the thruster controller electronics, which employs torque feedback to track T.

2.4.2 Pose Control

The pose controller sends velocity commands to the velocity controller (which converts them to thruster commands, as described above) in order to drive the vehicle to a desired pose. We differentiate the operation of the pose controller into Open Water, Proximity, and Reactive modes, which achieve higher level motion control by sequentially composing multiple simple behaviors [Burridge et al., 1999]. Open Water mode drives directly to a waypoint and is used for traveling long distances. Proximity mode is used near walls for sampling. Reactive mode is used as an abort strategy.

Open Water Mode

The most common mode is open water, which includes station keeping and waypoint motion. In this mode the vehicle uses a "virtual forward" to keep the vehicle oriented for optimal thruster use, which is 45 degrees from the vehicle forward defined by the sampling arm. Pose homing serves as a base for all open water motion. This algorithm requires a 4D goal (position and heading) and tolerance, and independently computes lateral, vertical and rotation velocities to drive the vehicle

toward that goal using PID laws. Station keeping is a special case where the goal configuration is the initial position and heading. Another special case is rotation to a heading, the goal configuration merely contains the current position and desired heading.

Homing is decomposed into descent, rotation and lateral phases. The vehicle first moves to the target depth. During descent a slight rotation is also applied to collect better sonar data. Once at the target depth, the vehicle then rotates so that the virtual forward direction is pointing at the goal. The final phase moves the vehicle laterally to the goal.

In the ideal case, both the descent and lateral phases would be straight lines. However currents and hydrodynamics push the vehicle off-course. To correct for this, the pose controller computes a trackline it attempts to follow using a form of pure pursuit Coulter [1992]. This trackline is defined by start and end points x_s and x_e, such that

$$l = x_e - x_s$$

At each step it uses the current position x_c to compute how far along the ideal path it has traveled ($\epsilon_{\text{downtrack}}$) and how far from the path is has drifted ($\epsilon_{\text{crosstrack}}$)

$$\epsilon_{\text{downtrack}} = \frac{l^T(x_c - x_s)}{||l||_2}$$

which we can use to compute the closest point x_{closest} on the trackline

$$x_{\text{closest}} = x_s + \epsilon_{\text{downtrack}} \frac{l}{||l||_2}$$

and thus the crosstrack error

$$\epsilon_{\text{crosstrack}} = ||x_c - x_{\text{closest}}||_2$$

We then compute a target point x_t on the trackline that is ahead of the vehicle by a distance that is inversely proportional to the cross track error

$$x_t = x_c + \left(\frac{1}{\epsilon_{\text{crosstrack}}}\right) \frac{l}{||l||_2}$$

This point is then used as the target for naive homing. Thus as the vehicle drifts further away, the target point is brought closer to the vehicle making it move more aggressively back onto the trackline. See Figure 2.3 for a schematic of this procedure, and Figures 2.4 and 2.5 for plots showing actual control performance.

Proximity Mode

In Proximity mode, the pose controller is capable of approaching a wall and moving along it based on sonar and inertial sensors. Goals are specified as distance d from the wall, lateral motion ℓ along the wall, and heading θ from perpendicular to the plane of the wall (Figure 2.6) (changes in depth

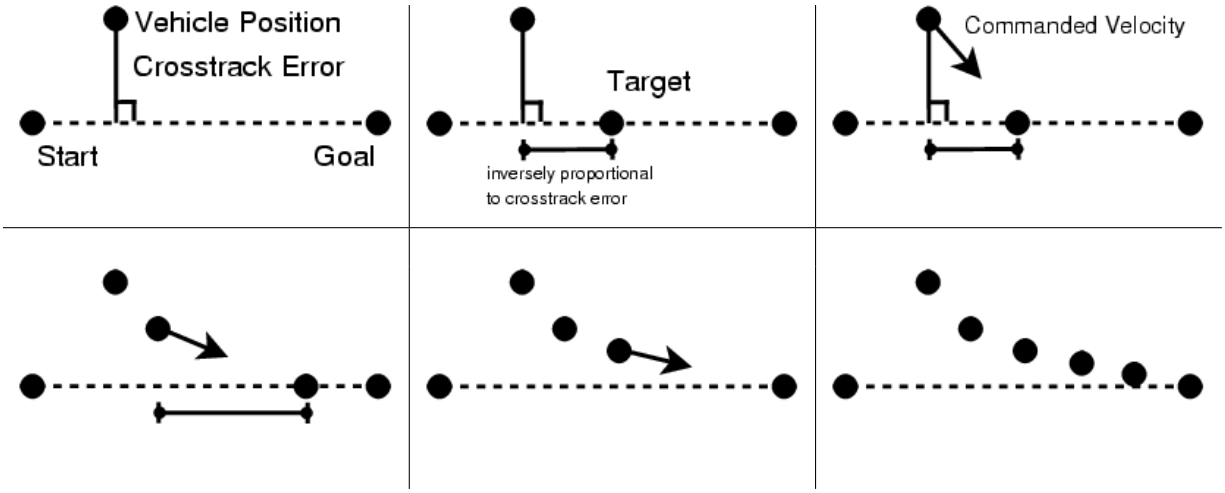


Figure 2.3: Diagram of the open water line tracking method, an example of pure pursuit.

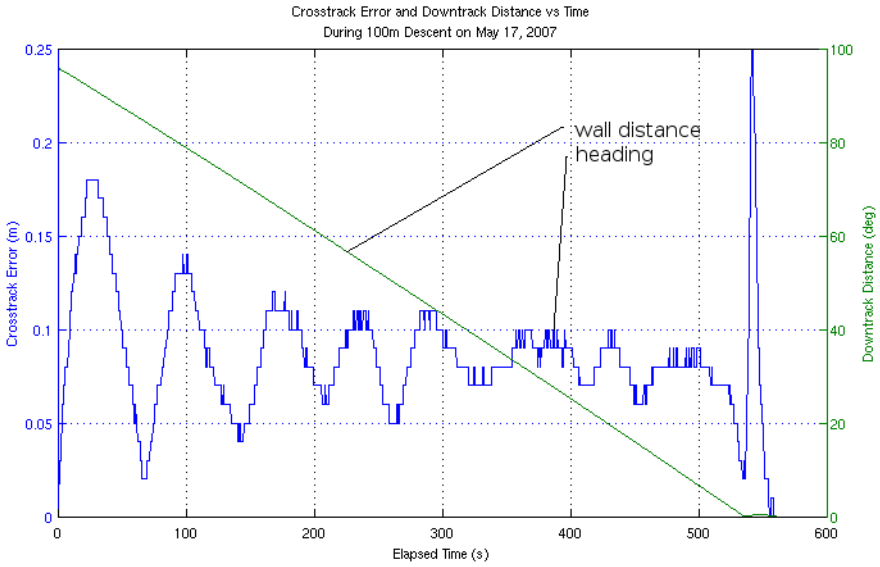


Figure 2.4: Cross track error and downtrack distance from a 100m descent (in Zacatón, May 17).

are also supported, but are omitted here). Thus, the kinematics are

$$\begin{bmatrix} \dot{\theta} \\ \dot{\ell} \\ \dot{d} \end{bmatrix} = \underbrace{\begin{bmatrix} 0 & 0 & 1 \\ 0 & \sec\theta & d\sec\theta \\ -1 & \tan\theta & d\tan\theta \end{bmatrix}}_{\triangleq B(\theta)} \begin{bmatrix} v_x \\ v_y \\ \omega_z \end{bmatrix}$$

The pose controller relies on the forward looking sonars, including the ones mounted on the DVL, to sense the wall. It maintains a point cloud by projecting recent sonar returns into global coordinates. By fitting a plane to the point cloud, the pose controller can compute how far to turn

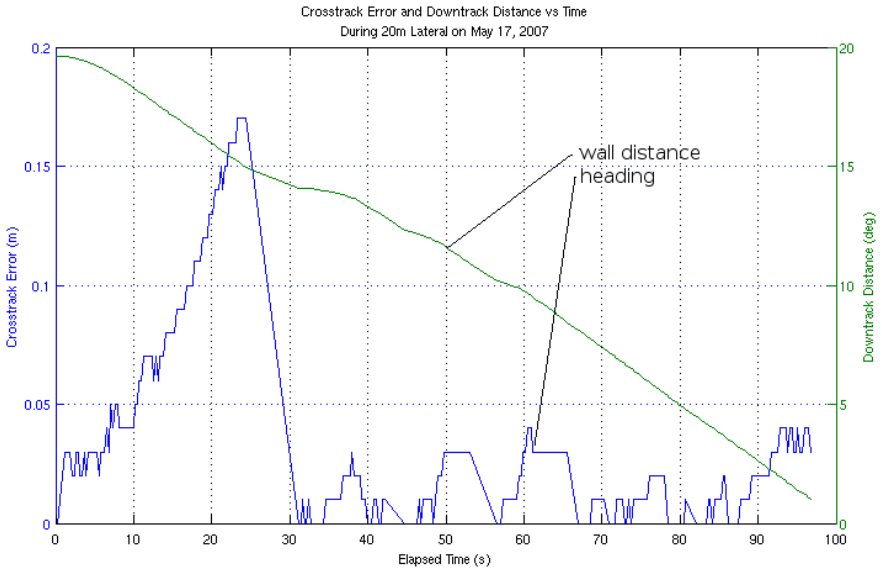


Figure 2.5: Cross track error and downtrack distance from a 20m lateral transit (in Zacatón, May 17).

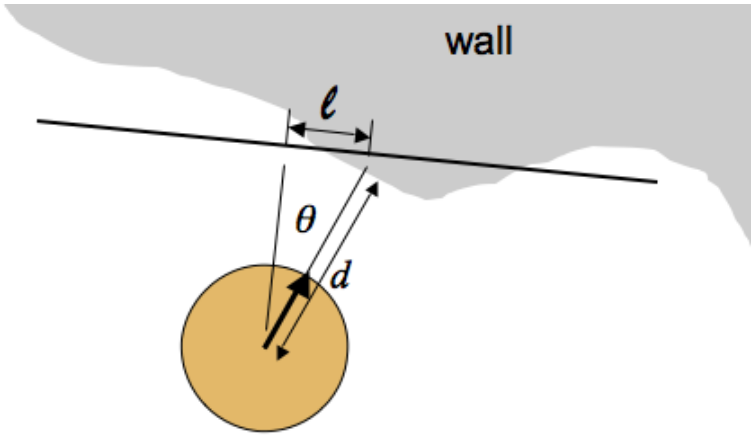


Figure 2.6: Diagram showing the proximity mode parameters: distance d from the wall, lateral motion ℓ along the wall, and heading θ from perpendicular to a plane fit to the wall.

to point perpendicular to the wall.

The resulting control law decouples the variables θ, ℓ and d, and exponentially drives each one to the desired values θ_d, ℓ_d and d_d.

$$\begin{bmatrix} v_x \\ v_y \\ \omega_z \end{bmatrix} = B(\theta)^{-1} \begin{bmatrix} \gamma_\theta\,(\theta_d - \theta) \\ \gamma_\ell\,(\ell_d - \ell) \\ \gamma_d\,(d_d - d) \end{bmatrix} \quad \Longrightarrow \quad \begin{aligned} \dot{\theta} &= \gamma_\theta\,(\theta_d - \theta) \\ \dot{\ell} &= \gamma_\ell\,(\ell_d - \ell) \\ \dot{d} &= \gamma_d\,(d_d - d) \end{aligned}$$

Figure 2.7 shows plots of wall distance and heading error (the angle from from perpendicular to the wall) during a wall approach maneuver. Note that the control algorithm does not attempt to

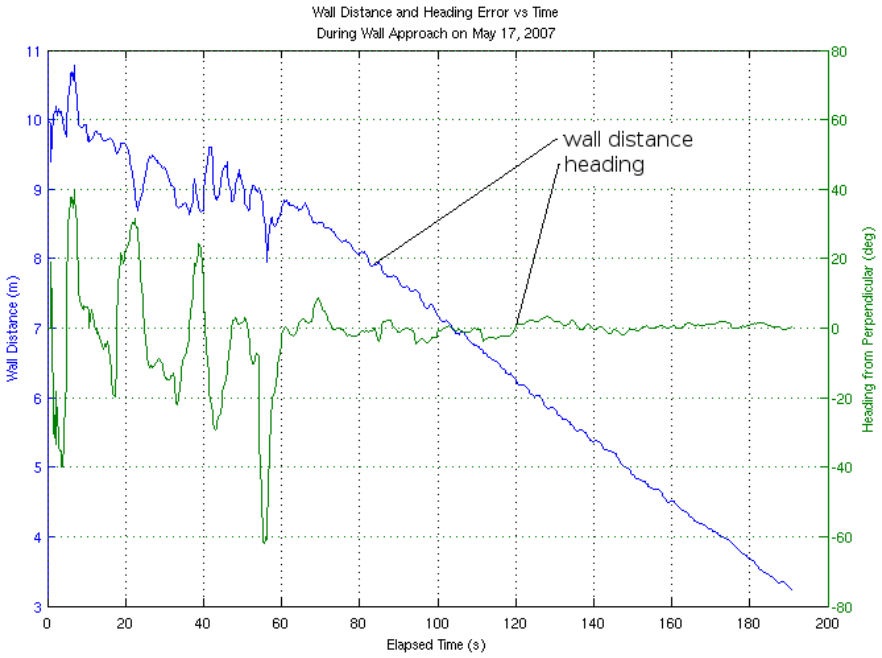


Figure 2.7: Vehicle control performance during a wall approach maneuver (in Zacatón, May 17).

minimize lateral travel. Also note that the wall distance does not reach 3m, the threshold is 0.3m so the vehicle typically stops about 3.2m from the wall. Finally note that a fixed set of sonars is used, so data collected further from the wall has lower fidelity.

Autonomous core sampling

A key DEPTHX objective was to have the vehicle determine a core sampling site using online environmental modeling and then to collect a core sample from the cenote wall, all completely autonomously. The control portion of this task was to develop a control sequence that would reliably yield a sample using the robotic coring arm.

From an objective viewpoint, the first requirement for collecting a sample from the wall was to find a smooth portion of the wall. The second was to bring the corer into contact with the wall, and to keep it lightly pressed flush to the wall during the sampling sequence. A final requirement was to minimize the amount of turbulence directed at the wall from the vehicle thrusters, so as to allow the vehicle to collect photographs and undisturbed water samples from the core sample area.

This sequence evolved over the field season. Initial attempts to react to sensors (a mechanical contact switch and an altimeter) failed because those sensors were not aligned with the corer and did not give useful feedback. This led to low success rate; the walls were not flat so the sensors would often be measuring a protrusion or depression. We also attempted to use the mapping sonars to estimate wall contact; but we got noisy readings due to the short range and wall debris. This led to unstable behavior, causing motion perturbations and endangering the coring arm. After extensive testing we found the open loop approach to be the most reliable.

The first phase of the sampling routine used Proximity mode to approach the wall. The forward

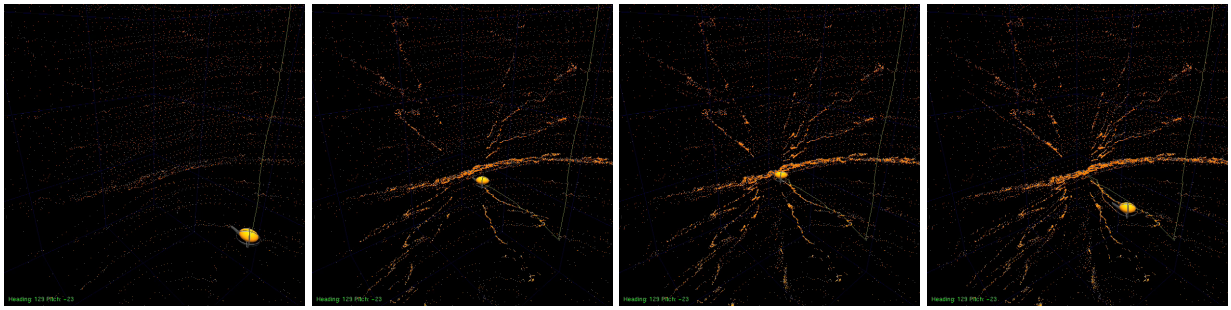


Figure 2.8: These images were generated from data collected during an autonomous wall sampling mission in Zacatón. Left to right, the vehicle achieves the desired depth, approaches the wall, takes the sample, and backs away. The star pattern in is due to the sonar beams sweeping the cenote wall as the vehicle approaches the wall.

looking sonars were monitored to estimate the distance and angle to the wall. The pose controller issued velocity commands to maintain a perpendicular stance and slowly move in to 3 m. We found that 3 m was a sufficient standoff distance to cause minimal disturbance to the wall.

The second phase gently thrust the vehicle forward into contact with the wall using a specialized controller that bypassed the velocity controller and issued thrust commands, as opposed to velocity commands. The vertical thrusters maintained depth, the forward thrusters were disabled and the rear thrusters maintained heading. The rear thrusters were carefully tuned such that one thruster was always providing the minimal thrust (as defined by the thruster deadband) while the other was slightly stronger to correct the heading. This motion persisted for 40 seconds, long enough to contact the wall and press against it during sampling.

The third phase used the same specialized controller to gently thrust backward. After 30 seconds the sampling arm was retracted, then another 30 seconds elapsed before the vehicle was stopped and the Executive reactivated the Open Water mode of the pose controller.

See Table 4.3 for an actual sampling sequence encoded as a mission plan, Figure 2.7 for control performance during an approach, and Figures 2.8 and 2.9 for a visualization of the sampling sequence during a wall approach.

Wall sample sequence resulted in a usable core sample about 75% of the time. Failures were due to problems during the approach sequence, bad contact with the surface, highly uneven surfaces, hard surfaces that resisted coring, and failure of the sample lid to close properly.

Reactive Abort

The Reactive mode is used during abort sequences. In this mode the pose controller commands the lateral component of velocity to move away from sonar returns. Each sonar return "pushes" the vehicle according to an inverse square law. All the sonar pushes are averaged to get an overall direction. The vertical velocity is not affected by the sonar ranges, and always pushes the vehicle toward the surface: this means that Reactive mode is susceptible to local depth minima, such as enclosed domes.

Figure 2.9: Images taken from the two vehicle cameras as the core sampler came into contact with the wall during a wall sample misson. Left: the probe camera, mounted on the sampling arm itself and showing the core sampling device in the left of the image. Right: the stage 1 camera, mounted on the body of the vehicle and showing the extended sampling arm.

2.5 Summary

In this section, we derived the full 6 DOF model of a hovering vehicle and showed how we could simplify vehicle dynamics such that the vehicle could be well modeled as a decoupled 4 DOF system with linear dynamics. We then described how we used this decoupled controller to perform the desired vehicle maneuvers, both in open water and in close proximity to walls.

Throughout the discussion of the control system, we have assumed that the vehicle pose was known. In the next section, we describe the pose estimation system and also the mapping system, which builds 3D models of the environment.

Chapter 3

Pose Estimation and Mapping

The vehicle uses three different pose estimation methods: dead reckoning, localization with a prior map, and simultaneous localization and mapping. Onboard the vehicle a simple switching process decides which of the three pose sources to use, falling back to more primitive methods in the case of failure.

3.1 Dead Reckoning

GPS does not work underwater, nor do other radio localization methods. There are a variety of techniques employed for determining the position of underwater vehicles and they can be divided into those that utilize emplaced infrastructure and those that do not. (See Leonard et al. [1998] for a comprehensive survey.)

When accurate position is needed underwater, most AUVs and many human-driven remotely operated vehicles (ROVs) rely on a surveyed array of acoustic beacons, known as a long baseline (LBL) array. Acoustic beacons provide a fixed frame of reference for positioning the vehicle Whitcomb [2000]. Yoerger et al. [2007] have shown detailed sea-floor mapping of subsea vents using an LBL array with the Autonomous Benthic Explorer. Kirkwood et al. [2001] have developed an AUV for multi-day under-ice arctic survey.

Over large distances, or in underwater caves and tunnels, the performance of LBL systems is unknown due to signal attenuation, reverberation and multipath. Without a fixed LBL infrastructure, an AUV uses a combination of depth sensors, inertial sensors, magnetic compasses, and acoustic Doppler velocity logs (DVL) to compute a dead reckoned estimate of its position while at depth. With high accuracy attitude and depth sensors the uncertainty in the AUV's 3D pose (roll, pitch, yaw, x, y, z) is primarily in x and y. Most underwater navigation systems are based on a Kalman Filter which combines Doppler velocity and inertial measurements Larsen [2000]. These systems report navigation errors as low as 0.015% of distance traveled, however error accrues drastically in situations where the DVL is unable to make accurate velocity measurements.

Dead reckoning is the process of integrating vehicle velocities over time in order to produce a position estimate. As described in Fairfield et al. [2007a], our implementation of dead-reckoning attempts to fall back to the IMU velocity measurement when DVL velocity measurements are not

available. Unlike the DVL velocities, the IMU velocities drift over time since they are the result of the integration of accelerometers, and this leads to a drift in the position estimate. When the DVL velocities are available, they are used to estimate and correct for the IMU drift using a standard Kalman filter. This allows the IMU to fill in for brief DVL dropouts due to noise, uneven surfaces, and loss of lock when too close to a surface. Unfortunately, as discussed below, the IMU velocity estimates were often completely wrong due to IMU faults and could not be trusted even after filtering, probably due to temperature issues.

In detail, the raw DVL measurement is a vector specifying the velocity of the DVL in the DVL's own coordinate frame. Using the notation of Table 2.1, we denote this vector by $\mathbf{v}_D^D$. We actually want to know the velocity of the origin of the vehicle body frame (which is located at the centroid of the vehicle) represented in the world frame, which we denote by $\mathbf{v}_B^W$. To get this, we first compute the velocity of the vehicle in the vehicle frame:

$$\mathbf{v}_B^B = R_D^V \mathbf{v}_D^D - \omega_B^B \times \mathbf{o}^D.$$

Here, R_D^B is the constant rotation matrix describing the orientation of the DVL relative to the vehicle frame; $\omega_B^B = [\omega_x\ \omega_y\ \omega_z]^T$ are the angular velocities about each axis of the vehicle frame as measured by the IMU; and $\mathbf{o}^D$ is the position of the DVL in the vehicle frame (or lever-arm.) It is then a simple matter to rotate the vehicle frame velocities into the world frame

$$\mathbf{v}_B^W = R_B^W \mathbf{v}_B^B,$$

where R_B^W is the rotation matrix describing the orientation of the vehicle relative to the world frame, as measured by the IMU. The x and y components of $\mathbf{v}_B^W$ are then numerically integrated to get the new dead reckoned x and y position estimates using Euler's method.

As mentioned above, most AUV's use a Kalman filter to fuse the dead-reckoned position estimate with some sort of position aiding, either periodic GPS on the surface or LBL. DEPTHX has no such position aides, but we do use a Kalman filter to ameliorate the effect of DVL dropouts, as described next.

3.1.1 Estimating Velocity

Under normal conditions, the DVL provides 3D velocity measurements with error below 1 mm/s. However, there are circumstances in which the DVL measurements "drop out" and we must use alternative methods to estimate the vehicle velocities.

The DVL is mounted to the front of the vehicle facing forward and tilted down 30 degrees from horizontal, a nonstandard configuration for this instrument intended to allow the DVL to lock onto vertical walls as well as the bottom. This compromise, together with uneven bottom conditions causes it to frequently lose bottom lock in open water situations. During these intervals, which may be several minutes long, no DVL velocity measurements are available, making it difficult to update the dead reckoned position estimate.

An obvious solution to the DVL dropout problem is to use the velocity estimates provided by the IMU when DVL measurements are unavailable. However, since the IMU velocity estimates result from integrating measurements from the IMU accelerometers, they are subject to significant

drift. To address this drift, we use a Kalman filter to estimate the drift of the IMU velocity estimates during the times when the DVL has a lock.

We use two independent and identical Kalman filters, one to estimate the IMU velocity and drift in the world frame x direction, the other to estimate the same parameters in the y direction.[1]. The state of each filter is $q = [v,\ d]^T$, where v is the world frame vehicle velocity in the relevant direction, and d is the associated IMU drift term. If we define q_k to be the value of the state at the time t and q_{k+1} to be the value of the state at the time $t + \Delta t$, then the state transition model that maps (q_k, a_k) to q_{k+1} is

$$q_{k+1} = \begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix} q_k + \begin{bmatrix} 1 \\ 0 \end{bmatrix} a_k + w_k \triangleq F_k q_k + G a_k + w_k,$$

where the process noise w_k is assumed to be white, zero-mean, and Gaussian with covariance matrix Q. The DVL provides a measurement of the velocity v, so the measurement model is

$$y_k = \begin{bmatrix} 1 & 0 \end{bmatrix} q_k + \nu_k \triangleq H_k q_k + \nu_k,$$

where the measurement noise ν_k is assumed to be white, zero-mean, and Gaussian with variance R. From here, the filter is implemented using the standard Kalman equations (see, e.g., Maybeck [1979]), with the small caveat that prediction steps are executed whenever an IMU measurement is received (about 50 Hz) and update steps are executed whenever a valid DVL measurement is received (about 4 Hz).

Another symptom of losing bottom lock is that the last few DVL measurements before and the first few measurements after the loss of lock are likely to be noisy. In order to filter out these measurements, a simple first difference filter discards DVL measurements which indicated a change in speed of more that 0.05 m/s, which is higher than the vehicle's maximum acceleration.

3.2 3D Maps – Evidence Grids

An evidence grid is a uniform discretization of space into cells in which the value indicates the probability or degree of belief in some property within that cell. In 3D, the cells are cubic blocks of volume, or voxels. The most common property is occupancy, so evidence grids are often also called occupancy grids [Martin and Moravec, 1996]. The primary operations on a map are inserting new evidence, querying to simulate measurements, and copying the entire map. We call the process of updating all of the voxels which are affected by a particular measurement an *insertion*, and likewise the process of casting a ray within a map until intersects with an occupied voxel we call a *query*. Often, the log-odds value for each voxel θ

$$l(\theta) = \log\left(\frac{p(\theta)}{1 - p(\theta)}\right)$$

[1]Note that the IMU running its own internal Kalman filter, but we have no way to inform that filter of the DVL velocity measurements

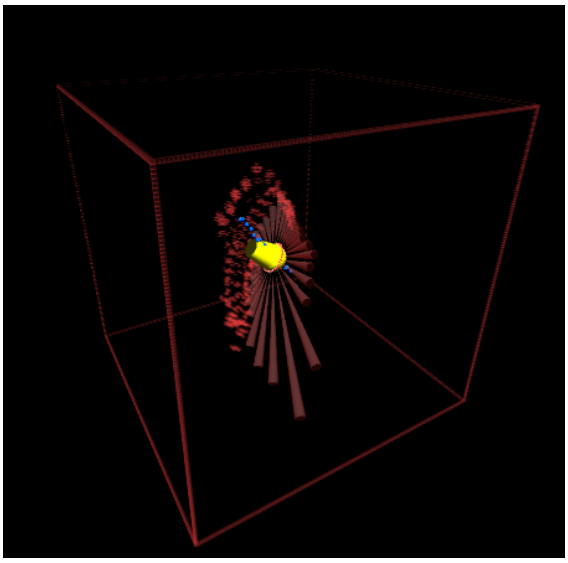


Figure 3.1: A cut-away view of sonar data being inserted into a 3D evidence grid. The vehicle is modeled as a yellow oblong, its sonar beams as red cones which leave traces of evidence behind in the grid.

is stored in the map rather than the raw probabilities because it behaves better numerically, and because the Bayesian update rule for a particular voxel according to the sensor model (like a conic beam-pattern) for some measurement z becomes a simple addition [Martin and Moravec, 1996]:

$$\log(\theta') = \overbrace{\log\left(\frac{p(\theta|z)}{1-p(\theta|z)}\right)}^{\text{beam model}} + \overbrace{\log\left(\frac{1-p(\theta)}{p(\theta)}\right)}^{\text{map prior}} + \log(\theta).$$

The first term on the right-hand-side is the sensor model, and the second is the map prior. If the prior $p(\theta) = 0.5$, the second term is zero and the initialization simply sets all voxels to zero. The update for each voxel can be reduced to simply summing the value of the sonar model with current voxel evidence. One important (and plainly false) assumption underlying the Bayesian insertion is that the cells are independent – that is that the occupancy of one cell is independent of the occupancy of any other cell. However, without this assumption evidence grids become intractable since the repercussions of updating a single cell could (hypothetically) propagate through the entire map. The drawback is that maps tend to be more noisy in response to ambiguity in the measurements.

Digression on Sonar Sensors

The DEPTHX vehicle had a total of 56 2 degree beamwidth pencil-beam sonars in two banks: 32 at 600 kHz and 24 at 300 kHz. Due to hardware constraints, gains and range gates could only be applied to an entire bank, and even worse, the entire bank had to be fired at once. This led to high levels of acoustic noise, particularly in the larger cenotes (Zacatón and Verde), and particularly near the water-air interface, which is an almost perfect acoustic reflector. In an attempt to mitigate

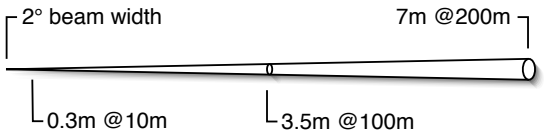


Figure 3.2: Illustration of the spreading of the 2 degree beamwidth sonar.

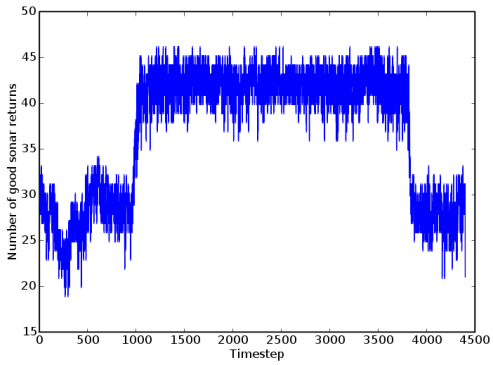


Figure 3.3: The number of 'good' sonar returns over the course of a typical mission in Caracol – the vehicle had a total of 56 sonars. Near the surface, many upward-looking sonars were prone to surface reflections.

this noise, we fired the sonars at around 1 Hz, which reduced the noisy ranges, but also reduced the available measurements.

Simplifying somewhat, the 2 degree beamwidth means that the range value returned by the sonar could have been caused by a reflection from anywhere within a 2 degree cone projecting from the sonar transducer (Figure 3.2). At large distances, the sonar data is coarse, meaning that fine detail cannot be resolved perpendicular to the direction of the beam.

The sonars were arranged in three perpendicular rings around the vehicle, at intervals of about 18 degrees. This geometry was designed to measure ranges in all directions, but with only 56 sonars the coverage was very sparse. To fill in the gaps in the coverage, the vehicle spun at about 10 degrees/s during descents and ascents.

Sound travels at $\tilde{1}500$ m/s in water, so the round trip time for a 150 m range takes 0.2 s. Since the robot was often rotating at 10 degrees/s, this means that the sonar would rotate by 2 degrees between the time the sonar pulse was emitted and the time it was received. In an attempt to resolve the ambiguity about exactly what is measured by such a ping, we based our timestamped the vehicle's position at the time that the sonars fired.

Sonar Model

Summarizing the description of the sonar sensors above, sonar measurements are sparse, noisy, low resolution, and low rate – but over time they do provide information about the environment around the vehicle. As measurements are collected, the evidence they provide about the occupancy of each

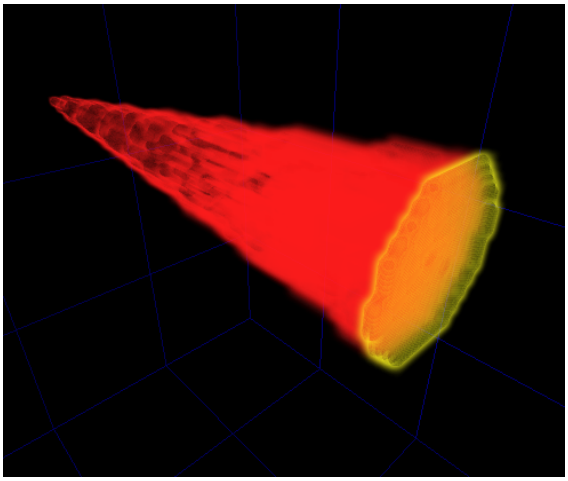


Figure 3.4: A single 6° sonar beam as represented in an evidence grid. Unknown regions are transparent, probably occupied regions are yellow, probably empty regions are red. In this rendering, very low and very high probability regions have been clipped out, leaving iso-probability shells.

voxel is entered into the evidence grid map. Given a particular range measurement, a sonar beam model specifies which voxels within the cone are probably empty and which voxels are probably occupied. There are several methods which can be used to construct a beam model, including deriving it from physical first principles [Urick, 1983] or learning it [Martin and Moravec, 1996]. We chose to use the simplest reasonable approximation – a cone with a cap that is loosely based on the beam-pattern of the sonar (Figure 3.4). The cone is drawn as a bundle of rays with constant negative value, with terminating voxels with constant positive values. These log-odds values were experimentally chosen to be -2 and 8. Likewise, the simplest method to query a sonar range from the 3D evidence grid is to trace a ray until some threshold (or other terminating condition) is met. Using matrix transformations for each voxel is too computationally expensive for operations such as filling in evidence cones or simulating ranges. These tasks can be decomposed into raster operations, which we performed with a 3D variant of the classic 2D Bresenham line drawing algorithm [Bresenham, 1965].

Octree Data Structure

The main difficulties with 3D maps arise from the cost of copy operations and the storage requirements that increase with map size and resolution. If we store the evidence log-odds as single bytes (with values between -128 and 127), then an evidence grid 1024 cells on a side requires a megabyte of memory in 2D and a gigabyte in 3D. A typical memory bus can handle transfer rates of around 400 Mb/s, and so would require over two seconds to copy such a map. In the case of the Particle Filter (Section 3.3), we need to store and copy hundreds of maps per second. This requires a more efficient data structure than a uniform array; the octree is one such structure.

An octree is a tree structure composed of a node, or *octnode*, which has eight children that equally subdivide the node's volume into *octants* (Figure 3.5). The children are octnodes in turn,

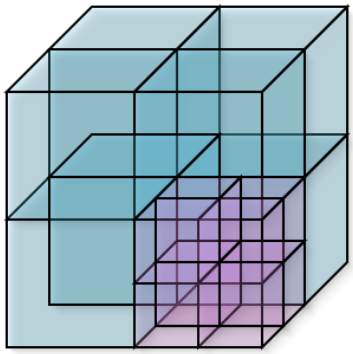


Figure 3.5: Each level of an octree divides the remaining volume into eight octants, but the tree does not have to be fully expanded.

which recursively divide the volume as far as necessary to represent the finest resolution required. The depth of the octree determines the resolution of the leaf nodes. The main advantage of an octree is that the tree does not need to be fully instantiated if pointers are used for the links between octnodes and their children. Large contiguous portions of an evidence grid are either empty, occupied, or unknown, and can be efficiently represented by a single octnode – truncating all the children which would have the same value. As evidence accumulates, the octree can mcompact homogeneous regions that emerge, such as the large empty volume inside a cavern. Note that even with compaction the octree supports the insert, query, and copy operations, and is a drop-in replacement for the uniform array: it is possible to convert losslessly between the two representations. Insert and query can be done with a tree-traversing ray-tracing algorithm (see [Havran, 1999] for an overview). We employ our own Bresenham 3D top-down approach, largely for its simplicity. Octrees have been widely used in ray-tracing as a way to sort polygons – not to store evidence, as we do here. Most applications also do not need to copy octrees, as we do for the particle filter (see below). The common approach to copying an octree is simply to traverse the entire tree copying every node (see the "Naïve" row of Figure 3.6).

To improve performance, we use our own custom memory management for the octnodes. Octnode memory is created as needed in large blocks (about 10k nodes) which are initialized as a FIFO linked list of free nodes. The first word of each node points to the next free node. As nodes (and entire octrees) are freed, nodes are pushed onto the front of this linked list. When a new node is needed, the first entry of the linked list is pulled off and initialized with default values.

Reference Counting Octree (RCO) The first real-time challenge when using octrees is not the ray-tracing operation, but rather copying the maps during the resampling step of the particle filter. Copying an octree is an expensive operation, but we can allow octrees to share subtrees by maintaining reference counts to the octnodes: each node keeps track of the number of references to itself. We refer to this number as refCount. This means that we replace naïve copying with copy-on-write (see the "Reference Counting" row of Figure 3.6). When an octree needs to be copied, we simply increment the reference count. Then when either copy is modified only subtrees which have a reference count greater than one need to be copied. And of course once a copy has been

Procedure 1 DRCO COPY(A to B)

Require: A and B are pointers to octnodes

```
1: A→defC++;
2: LAZYFREENODE(B); // See Procedure 2
3: B = A;
```

Procedure 2 DRCO LAZYFREENODE(N)

Require: N is a pointer to an octnode

```
1: if N→defC > 0 then
2:     N→defC--;
3:     return
4: end if
5: N→refC--;
6: for i = 1...8 do // Recursively free children
7:     LAZYFREENODE(N→child[i]);
8: end for
9: if N→refC == 0 then // Free the node
10:    N→nextNode = globalFreeNode;
11:    globalFreeNode = N;
12: end if
```

made, the reference count of the source is decremented. We call this "copy-on-write", and find it useful because in the particle filter application, queries to the map are much more common than insertions.

Deferred Reference Counting Octree (DRCO) However, maintaining the bookkeeping for reference counts requires a full traversal of the octree – or at least of the modified subtree – which is almost as expensive as copying. Our novel solution is to maintain deferred reference counts. Every octnode has a refCount and also a deferred reference count, which we will call the defCount (see the "Deferred Reference Counting" row of Figure 3.6). The defCount represents reference counts which have not yet been propagated to the children. This is similar to the work of [Baker, 1994]. The true reference count of the node is the sum of refCount and defCount – but changes in defCount do not usually trigger a recursive traversal through the subtree.

Copying a map now simply requires incrementing the defCount of the node pointed to by the source, freeing the node pointed to by the destination (if it is not new), and setting the destination pointer to the source node (see Procedure 1). If either of the two copies is modified then the copy-on-write code will automatically push the defCount down the tree, copy the portion of the tree which will be changed, and set the reference counts accordingly (see the "Deferred Reference Counting" row of Figure 3.6 and Procedure 3).

LazyFreeNode applies the same deferred (or lazy) principles of recursively freeing octnodes: if the octnode has defCount > 0, then it can just decrement the defCount (see Procedure 2): what the children don't know won't hurt them. If the defCount $= 0$, then LazyFreeNode must call itself recursively on the children (who may decide they don't need to tell their children, etc).

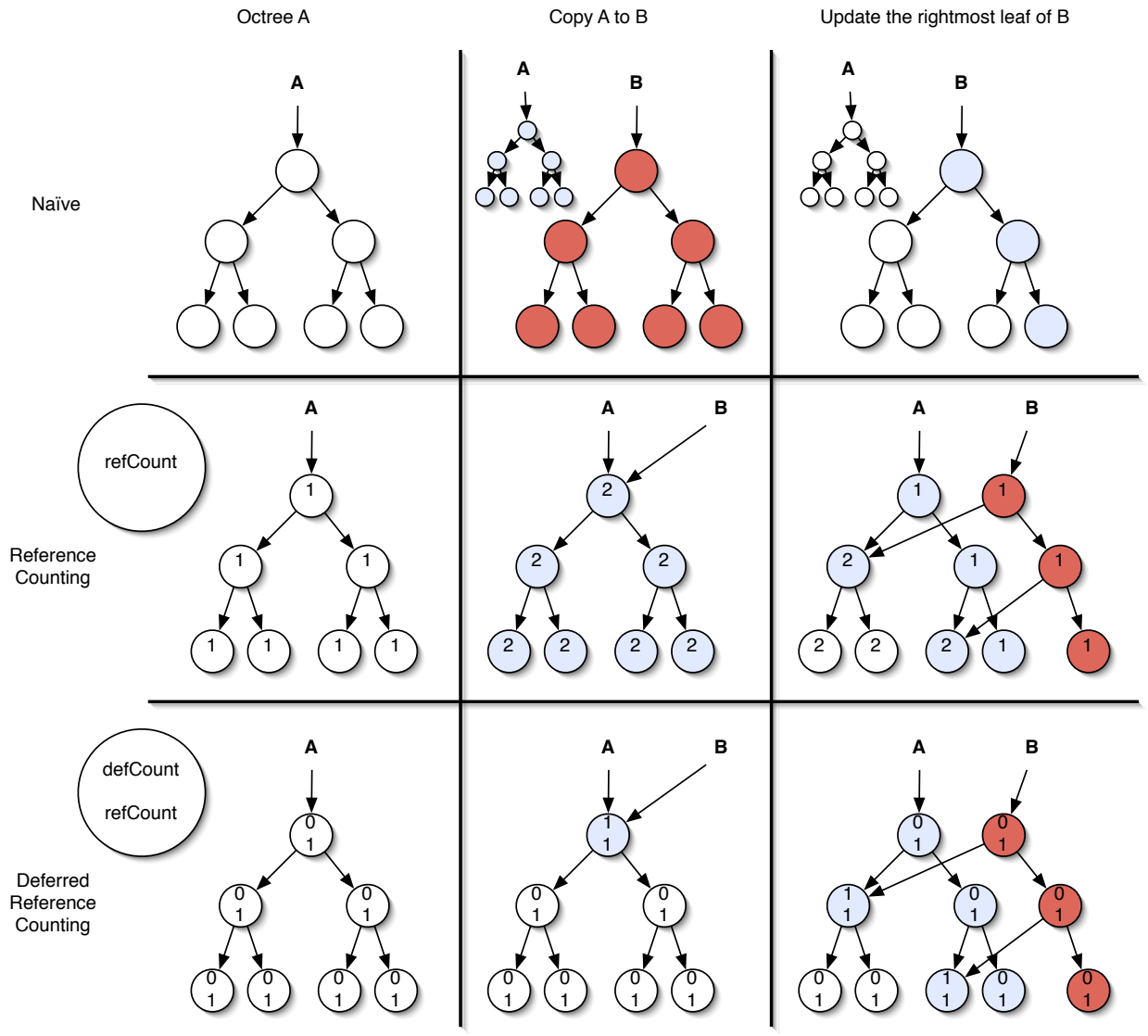


Figure 3.6: Octree diagram demonstrating the copy and write operations for three octree implementations: Naïve, Reference Counting, and Deferred Reference Counting. Blue (light gray) nodes are notes that must be accessed, and red (dark gray) nodes are new nodes.

The ray-trace write, or "insert", operation in the DRCO must deal with properly updating all these counts – the ray-trace read, or "query", operation is unchanged. When we want to insert updates into a map, copy-on-write propagates the node's defCount down to its children, sets the refCount = refCount + defCount, and sets defCount = 0 (since defCount represents the reference counts that the node hasn't told its children about) (see Procedure 3). If the new refCount is > 1, the node must be copied (maintaining the old connections to its children). The deferred updating works because the insert procedure always starts at the top of the octree – this top-down property is part of our ray-trace implementation. DRCO's yield a significant performance boost, and allow us

Procedure 3 DRCO SETNODE(N, value)

Require: N is a pointer to an octnode, value is some constant

```
1: if N→defC + N→refC > 1 then
2:     if N→defC > 0 then
3:         for i = 1...8 do // Propagate defC to children
4:             if N→child[i] != 0 then
5:                 N→child[i]→defC += N→defC
6:             end if
7:         end for
8:     end if
9:     newN= NEWNODE();
10:    COPYNODE(newN, N);
11:    newN→refC = 1;
12:    newN→defC = 0;
13:    newN→value = value;
14:    N→refC = N→defC + N→refC - 1;
15:    N→defC = 0;
16: else
17:    N→value = value;
18: end if
19: N = NEXTNODEINTOPDOWNTRAVERSE()
20: if N != 0 then
21:    SETNODE(N, value)
22: end if
```

to represent maps that would not even fit into memory as a uniform array (see the Results Section).

Compaction Compaction is the process of recursively traversing the tree, replacing groups of homogeneous nodes with a single parent node with the same value. As regions of the map become well explored, fuzzy compaction can simplify regions with small inhomogeneities, caused by noise. Periodically compacting the octrees can yield significant space savings, especially when the map will only be used for querying – in which case the values can be lossily thresholded to {empty, unknown, occupied} (see Table 3.1). Ray-tracing is also accelerated if the ray-tracing algorithm takes advantage of these compacted regions.

Now that we have developed a data structure to maintain and duplicate large 3D evidence grids efficiently, we can consider how to build maps and localize using a particle filter.

3.3 Particle Filtering Localization

The goal of SLAM is to estimate the probability distribution at time t over all possible vehicle states s and world maps Θ using all previous sensor measurements Z_t and control commands U_t

Original 512^3 octree map	71 MB	100%
Lossless compaction	55 MB	77%
{Empty, unknown, occupied} compaction	13 MB	18%
{Empty, unknown, occupied} fuzzy compaction	8 MB	11%
Standard 512^3 1-byte/voxel evidence grid	128 MB	180%

Table 3.1: Example compaction results on a map of Caracol, 512^3 map = 128^3 m at 0.25 m resolution.

$\#_{par}$	number of particles
$\#_{son}$	number of sonars
$s_t^{(m)}$	vehicle pose of the m-th particle at time t $= (roll, pitch, yaw, x, y, z)^T$
$S_t^{(m)}$	trajectory of m-th particle from time 0 to t $= \{s_0^{(m)}, s_1^{(m)}, s_2^{(m)}, \ldots, s_t^{(m)}\}$
z_t	sonar measurements at time t
Z_t	history of measurements from time 0 to t $= \{z_0, z_1, z_2, \ldots, z_t\}$
${}^n z_t$	n-th sonar measurement at time t
u_t	vehicle dead-reckoned innovation at time t
U_t	history of dead-reckoning from time 0 to t $= \{u_0, u_1, u_2, \ldots, u_t\}$
$\Theta^{(m)}$	map of m-th particle
θ	a particular voxel $= {}^{ijk}\Theta$
$w_t^{(m)}$	m-th particle weight at time t

Table 3.2: Particle filter notation.

(for a complete list of notation, see Table 3.2):

$$p(s, \Theta | Z_t, U_t)$$

This distribution is called the *SLAM posterior*. The recursive Bayesian filter formulation of the SLAM problem is straightforward (see Montemerlo et al. [2002] for a derivation) but the integral is usually computationally intractable to solve in closed form:

$$\underbrace{p(s_t, \Theta | Z_t, U_t)}_{\text{new posterior}} = \eta \ \underbrace{p(z_t | s_t, \Theta)}_{\text{measurement model}} \int \underbrace{p(s_t | s_{t-1}, u_t)}_{\text{motion model}} \underbrace{p(s_{t-1}, \Theta | Z_{t-1}, U_{t-1})}_{\text{old posterior}} ds_{t-1},$$

where η is a constant scale factor from Bayes' rule.

The key insight of [Murphy, 1999] is that the SLAM posterior distribution can be factored into two parts, or marginals: the path distribution and the map distribution. Furthermore, knowing the vehicle's trajectory S_t makes the observations U_t conditionally independent, so that the map sample Θ can be computed in a closed form. The process of factoring a distribution such that one

$N(\mu, \sigma)$	normal distribution with mean μ and std dev σ
$h(s_{t-1}, u_t, N_h)$	vehicle motion model with noise model N_h $= p(s_t \mid u_t, s_{t-1})$
$g(s_t, \Theta, N(0, \sigma_z))$	sonar measurement model $= p(z_t \mid s_t, \Theta)$

Table 3.3: Model notation.

part can be computed analytically is known as Rao-Blackwell factorization [Doucet et al., 2000]. As a result, following [Montemerlo et al., 2002] we compute the posterior over *trajectories*, we can factor the distribution as

$$p(S_t, \Theta | Z_t, U_t) = p(S_t | Z_t, U_t) p(\Theta | S_t, Z_t).$$

Particle filters are a Monte Carlo approximation to the Bayesian filter. The particle filter maintains a discrete approximation of the SLAM posterior using a (large) set of samples, or *particles*. The m-th instance of the $\#_{par}$ particles represents both a sample pose $S_t^{(m)}$ from the distribution of vehicle trajectories, and the sample map $\Theta^{(m)}$ which results from that trajectory combined with the sensor measurements Z_t. Since we update the particle maps at every time-step, they represent the combination of sensor measurements and vehicle trajectory – so each particle only needs to store the current map $\Theta^{(m)}$ and pose $s_t^{(m)}$ (rather than the whole trajectory $S_t^{(m)}$).

For practical purposes, when SLAM is being used to provide a pose for the rest of the vehicle control software, we usually want to turn the set particles into a single point estimate. If the posterior distribution is Gaussian, then the mean is a good estimator, but other estimators may be better if the distribution becomes non-Gaussian.

The particle filter algorithm has the following steps:

Initialize The particles start with their poses s_0 initialized according to some initial distribution and their maps Θ (possibly) containing some prior information about the world. This is called the *prior distribution*.

Predict The dead-reckoned position innovation u_t is computed using the navigation sensors (IMU, DVL and depth sensor). A new position s_t is predicted for each particle using the vehicle dead reckoning-based motion model (see Table 3.3):

$$s_t = h(s_{t-1}, u_t, N).$$

This new distribution of the particles is called the *proposal distribution*.

Weight The weight w for each particle is computed using the measurement model and the sonar range measurements (from the $\#_{son}$ different sonars):

$$w = \eta \prod_{n=1}^{\#_{son}} p({}^n z_t | s_t, \Theta),$$

where η is some constant normalizing factor (different than the one used in the expression for the Bayesian filter). In our implementation, the real range measurements z are compared to ray-traced ranges $\hat{z}$ using the particle pose and map. We compare the simulated and real ranges using the measurement model

$$z = g(s_t, u_t, N(0, \sigma_z)),$$

which is assumed to have a normal noise model, so

$$p(z|s, \Theta) = \frac{1}{\sqrt{2\pi\sigma_z^2}} e^{\frac{-(\hat{z}-z)^2}{2\sigma_z^2}}.$$

Substituting into the expression for particle weight and taking the logarithm of both sides shows that maximizing this weight metric is very close to minimizing the intuitive sum squared error metric:

$$\log w = C - \frac{1}{2\sigma^2} \sum_{i=1}^{\#_{son}} ({}^i\hat{z} - {}^iz)^2,$$

where $C = \#_{son} \times \log\left(\sqrt{2\pi\sigma^2}\right)$. An alternative weighting method based only on the comparison of the voxel values of the end-points of the sonar beams, called "point correlation" was found to be slightly less informative [Fairfield et al., 2005].

Resample The $O(n)$ algorithm described in [Arulampalam et al., 2002] is used to resample the set of particles according to the weights w such that particles with low weights are likely to be discarded and particles with high weights are likely to be duplicated. The set of particles is now our new estimate of the new SLAM posterior.

Update The measurements z are inserted into the particle maps $\Theta^{(m)}$ (as described in Section 3.2) to update the evidence of all the voxels θ which lie in the conic sonar beam model of each measurement relative to the particle position. This is when maps must be copied and updated. We save duplicate insertions by inserting *before* copying successfully resampled particles.

Estimate Generate a position estimate from the particles.

Repeat from Predict

3.3.1 Extensions

Sensor-based prediction The particle filter algorithm uses the current sensor models during the prediction phase. For example, the IMU provides excellent heading ($\pm 0.1^\circ$), while the DVL provides much worse heading ($\pm 2^\circ$). If the IMU is available, the predict step takes the heading value, adds Gaussian noise $N(0, 0.1^\circ)$, and uses the new value to dead-reckon the particle's new position. If the IMU fails, then the filter will fall back to the DVL data, and generate more noisy predictions.

This is vital to the robustness of the SLAM method. In the case where there are redundant sensors, it would make sense to use a Kalman filter to combine the information and perform the prediction step using the covariance estimates as the prediction noise model.

An important observation is that when all DEPTHX sensors are functioning, the particle filter only really estimates x and y as the particles are distributed in the xy plane. However when sensors fail or degrade the particle filter distributes particles over the newly uncertain dimensions. This will dramatically increase the risk of under-sampling, but since the failure or degradation of a sensor will also cause the vehicle to terminate the mission and return to the surface, SLAM can switch to a localization-only mode (without map updates) in which it can support many more particles, which may ameliorate the situation.

Adaptive particle count The processing time of a single iteration of the SLAM algorithm varies significantly depending on local world geometry. This is because of the range-dependent ray-tracing time, but the implication is that the particle filter could be using more particles and still maintain real-time performance. Furthermore, the Weighting and Resampling steps take approximately the same amount of time given a fixed number of particles. How to best spend the available computational resources: weighting more particles in the hopes of finding good ones, or resampling more particles in order to maintain particle diversity? Our approach is to weight as many particles as possible, but then to fully resample according to those weights. With DRCO's, resampling particles is fast. We modify the particle filter algorithm as follows:

1. Start with a huge number of particles (in our case 5000: more than could ever be supported computationally)

2. During the Weighting step, set all particle weights to zero and then randomly pick particles and weight them until the time limit is reached

3. Fully resample the particle set – particles which were not weighted will always be replaced with weighted particles

Note that it is important to randomly pick particles during the weighting phase to avoid sorting effects, such that the distribution is accurately sub-sampled.

Measurement minimum for resampling As seen in Figure 3.3, the number of good sonar returns varied significantly. In order to avoid resampling the particles when only a small number of sonar returns had been observed, we imposed a condition requiring that at least, for example, 200 sonar ranges had been integrated into the particle weights before allowing resampling. The vehicle usually collected around 30 good ranges a second, meaning that resampling occurred every 5-10 seconds. In some cases, such as when the vehicle was near the surface, or near a wall, or both, the resampling period was longer since there were fewer good ranges.

Tolerant sonar model We also implemented a more tolerant sonar model, in which nearer ranges (which are generally more accurate) were weighted more heavily than more distant ranges, such that the error from a 100 m range was weighted half as much as the error in a 10 m range. To keep a single outlier from unduly affecting the weight, we also thresholded the maximum error between the measured and simulated ranges to 10 m.

3.4 SLAM

Simultaneous Localization and Mapping (SLAM) is the task of building a map of the environment from sensor data and simultaneously using that map to localize, or recover the robot's actual trajectory. In most cases the robot uses various sensors to measure its own motion and sense its local surroundings. This sensor data is inevitably noisy, and must be appropriately filtered as part of SLAM.

SLAM methods usually depend on the detection of features from sensor data, and combine observations of these features with an Extended Kalman Filter (EKF) [Smith et al., 1990]. In the underwater domain, sonar sensors are not capable of providing the resolution necessary to resolve and recognize features. There has been work on off-line SLAM methods using tunnel cross-sections, or slide images, which can be derived from sparse sonar ranges as long as the environment is tunnel-shaped [Bradley et al., 2004]. In the case where there are free floating artificial features, scanning sonars have been shown to have high enough resolution to support feature-based SLAM [Williams et al., 2000]. Alternatively, in clear water with good lighting, SLAM has been demonstrated via video mosaicing [Eustice et al., 2005] and also a combination of vision-based feature detection and sonar [Williams and Mahon, 2004]. Roman [2005] uses multibeam sonar maps to do SLAM over varied topography.

Underwater localization has also been demonstrated in cases where there is variation in the sea-floor and the vehicle has a prior map of the bathymetry [Williams, 2003, Leonard et al., 1998]. Many underwater environments are characterized by large monotonous regions where there has been promising work with Synthetic Aperture Sonar (SAS) to support range-and-bearing SLAM [Newman et al., 2003].

As an alternative SLAM method to EKFs, particle filters provide a proven implementation of Bayesian filtering for systems whose belief state, process noise, and sensor noise are modeled by non-parametric probability density functions (for a good summary of particle filters, see [Arulampalam et al., 2002]). Particle filters are often used when certain requirements of the Kalman Filter formulation (unimodal position distributions, feature detection) cannot be satisfied.

In the most general sense, particle filters sample over the entire state space of the vehicle. As long as this state space is fairly low dimensional, as in the case of estimating the 3D position of a vehicle, a few hundred samples will adequately represent the true distribution. But for SLAM, the state space of the vehicle is both the vehicle position *and* the map. This is a very high dimensional space, so we can't apply the basic particle filter: each particle would be a sample from all possible maps and all possible positions within that map. Rao-Blackwellized Particle Filters [Doucet et al., 2000] address this problem by maintaining the map portion of the state in a computationally efficient closed for (in our case, the evidence grid). Thus, our particles are composed of both a pose and the map which has been constructed according to the particle's trajectory and the range sensor data.

The Rao-Blackwellized particle filter SLAM algorithm maintains a map for each particle (the maps are initialized either blank or with a partial map of the world), and adds following steps to the localization algorithm:

Update The measurements z are inserted into the particle maps $\Theta^{(m)}$ (as described in Section

3.2) to update the evidence of all the voxels θ which lie in the conic sonar beam model of each measurement relative to the particle position. This is when maps must be copied and updated. We save duplicate insertions by inserting *before* copying successfully resampled particles.

If the vehicle already has a map of the environment then it can use that map, together with its range sensor measurements, to localize itself. In this case, all the particles share the prior map, which is not updated.

Discussion

[Fox, 2003] describes Kullback-Leibler distance (KLD) sampling, which estimates the number of samples needed at each iteration such that the error between the true density distribution and the discrete particle filter approximation is below some bound. This approach is applied to the bathymetry-map localization mentioned above by [Bachmann and Williams, 2003]. In short, they conclude that too few particles will poorly approximate the true posterior and too many particles take too long to process (so that measurements have to be thrown away).

The idea of the adaptive particle count is to use as many particles as possible in real-time *without* discarding any data, which is the dual of the KLD sampling technique. The adaptive particle count method appears to provide an improvement over an equivalent (from a real-time performance perspective) fixed particle count, particularly while in enclosed areas . The most important contribution of adaptive particle count is the reliable real-time performance in the face of unknown world geometry. However, it does not provide any guarantees about avoiding under-sampling. We believe that a better system would incorporate both KLD sampling and temporal constraints to determine the particle count. In the case where KLD sampling cannot be satisfied within the given time constraints the vehicle should probably terminate the mission and resurface, but it still must provide a SLAM solution.

The particle filter/octree combination is stable with regard to the fraction of particles that are resampled at each time step (which can vary between 0 and $N_{part} - 1$). Although discarding a particle is costly because of the potential for a recursive freeNode, the node does not have to be updated – which saves many ray insertions. What does seem to be a good indicator of the duration of an iteration is the average sonar beam length, which makes sense.

The performance of the DRCO depends heavily on the circumstances. It will be most efficient when the environment and particle filter are amenable to the exploitation of spatial locality (particles share most of the map in common when the vehicle is only modifying small regions) and volumetric sparsity (octrees compactly represent a map that is mostly empty or full). Most large-scale outdoor environments seem to be well suited for this type of exploitation.

The Distributed Particle (DP) SLAM method of Eliazar and Parr [2006] is a similar approach to ours, in that it uses evidence grids, a RBPF, and a sophisticated data structure that exploits the similarity between particles of common ancestry to reduce the cost of copying and storing particle maps. However, in order to get linear ray-tracing performance, they must repeatedly process their data structure to create a cache of uniform "local maps", a complex and memory intensive process, even for 2D maps. The Deferred Reference Count Octree (DCRO) we introduce below avoids

this caching step and yields the additional advantages of full 3D, sparse spatial representation, and overlap between particles with common ancestry – all inherent properties of the relatively simple DRCO data structure.

3.5 Summary

In this section, we have described the onboard pose estimation system, which includes three related algorithms: dead reckoning, localization with a prior map, and SLAM. We also described how the Deferred Reference Count Octree allows us to build sparse high resolution maps of large-scale 3D environments – and also how the it can be integrated with a Rao-Blackwellized particle filter to exploit spatial locality such that hundreds of particle maps can be supported in real time.

In the next section we describe the Exective, which uses the control and pose estimation systems to guide the vehicle along a scripted mission plan.

Chapter 4

Executive

The System Executive is the software module responsible for high-level command and control of the vehicle. It is given a mission specification that includes vehicle motion, science activities and a contingency plans. The mission is executed serially unless a critical system fault is detected.

The Executive metes out steps in a dive specification based on the state of the system and its location from the Pose Estimator. In the DEPTHX system there is no onboard planning – rather the Executive chooses actions from a primary or cascading set of contingency plans. Dive specifications are composed into mission plan files that encode, in an interpreted command language, the sequence of behaviors to execute. These behaviors may initiate sensing, navigate to locations, trigger on conditions, perform maneuvers, or acquire samples.

4.1 Safety System

Keeping the vehicle safe is perhaps the most important function of the Executive. At any point in the plan, the Executive may elect to execute a contingency plan in response to a system fault or timeout. Fault monitoring is split into two roles; detecting faults and reacting to them.

4.1.1 Detecting Faults

The Health Monitor is a companion module responsible for detecting system-wide faults. It monitors data from all the sensors as well as internal software state (Table 4.1). When it detects loss of data or values beyond acceptable levels, a “faultNotification” is generated.

4.1.2 Fault handling

The System Executive itself decides how to react to a fault or timeout scenario. Most faults are handled in one of two ways; either switching to a contingency plan or using a hard coded abort sequence (Figure 4.1). In some very specific cases it may elect to ignore the fault, for example a proximity warning is expected when taking a core sample. Example abort plan: Table 4.2.

Safety conditions
maximum depth limit
sonar-based proximity
battery voltage
localization confidence
excessive roll and pitch
high humidity (leaks)
high temperature
Sensor faults
variance between depth sensors
sensor communication dropouts
thruster faults
Housekeeping
disk space
heartbeat from hardware drivers

Table 4.1: The list of faults that were monitored by the vehicle safety system.

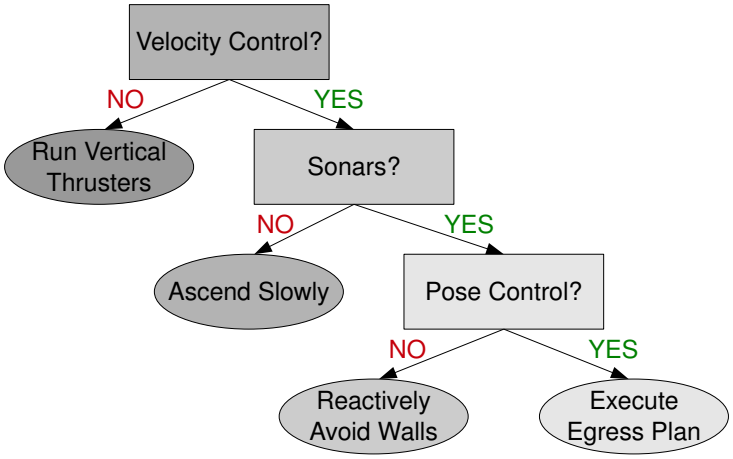


Figure 4.1: The vehicle's response to faults depends on its currently available capabilities: as more serious failures occur, the vehicle falls back to simpler egress methods.

The contingency plan defined in the mission specification is the preferred mechanism. It is an operator-specified plan just like the original mission, in fact it may contain contingency plans of its own. Specifying this plan is crucial when overhanging obstacles could trap the vehicle if it tried to return directly to the surface.

If a contingency plan is not specified, not enabled or not feasible, then a hard coded sequence is used. The particular sequence depends on how much functionality is still available. If all functionality is still available then the vehicle will attempt to ascend along the central ascent line. If the ascent line can not be used (for example, because of localization failure), then the vehicle ascends

```
#stop any wall sampling
kill thrustForward
kill thrustBackward
send armPosition -1

#go to the center and ascend
columncenter
send flyUp
```

Table 4.2: The abort plan from March 13, 2007 that stops any science activity and returns to the ascent line. Note that the "columncenter" command relies on the ascent line being inherited from the original mission.

while using the upward-looking sonars to reactively avoid obstacles. If neither localization nor the sonars are available, the only recourse is to fire the vertical thrusters and ascend blindly.

4.2 Science Activities

The Executive communicates with the science payload in two different modes; either handing off control to a peer "Science Executive" or directly commanding the components of the payload. Due to integration issues, we usually operate in the second mode: the Executive communicates with the payload components directly and all the individual commands needed to collect science data were incorporated into the mission specification.

Commanding the payload directly results in verbose and detailed plans. These plans start with some initialization and motion to a target sampling location. Then the majority of the mission specification deals with sampling protocol. With sampling complete, the final step in the mission is to return to the surface. A typical science sampling plan uses the fixed motion sequence: Table 4.3, however the whole sampling sequence can be triggered by environmental parameters to yield a reactive science sampling plan: Table 4.4.

4.3 Vehicle Motion

The third major responsibility of the Executive is coordinating vehicle motion. Closed loop vehicle motion is achieved through the pose controller, though the Executive can command the thrusters directly. Motion commands are specified in the plan, either with exact coordinates/distances, with behaviors and durations, or by an interval to be filled in randomly upon execution.

4.3.1 Interaction with the Pose Controller

The most common way of moving the vehicle is to send the pose controller an Open Water target and wait for completion. The two other common mechanisms are to invoke Station Keeping while taking a liquid sampling and Approach Wall before a solid sample.

In addition to those three common modes, the Executive can also use additional pose controller functionality when necessary. The Reactive behaviors are used in an abort situation to push away

```
#now we are somewhere near the wall
#start the samping procedure

#lock onto the wall
set nearWall 1
forwardDegrees 0 0
send approachWall 999 3
send stationKeeping
pause 10
run video_capture

#start extending the arm
send armPosition 0
pause 50

#start a liquid sample, this will
#be priming while the arm is extending
send liquidSample 2

#wait to finish extending the arm
#and do the purge
pause 40
hold

#thrust toward the wall and send
#a wallSample
run thrustForward
pause 35
#fire the sampler
send wallSample
pause 5
kill thrustForward

#now get away from the wall and
#retract the arm
run thrustBackward
pause 30
send armPosition -1
pause 30
kill thrustBackward

#all done with the wall
set nearWall 0
forwardDegrees 45 0
```

Table 4.3: The sample sequence of a typical plan (from May 25, 2007).

from nearby walls. The Wall Follow behavior can be used to move along a wall laterally or vertically. The Open Water Homing allows naive 4D (x,y,z,yaw) motion to face a particular direction or move laterally and vertically at the same time.

Most of the interaction with the pose controller also includes two key values, the "virtual forward" and the watchdog. The virtual forward is the nominal direction of travel and is typically set to 45 degrees for optimal use of the thrusters. During sampling however, this should be set to 0 degrees so that the sampling arm is considered "forward." Meanwhile the watchdog is a timeout on vehicle motion. Commands with a definite goal have a time limit to achieve that goal, while others (like Station Keeping) are considered to be instantaneous and perpetual.

```
#do scienceChecks at 10m increments
set sciencePlan sampleSequence.pln

t 10 -49  50
send stationKeeping
scienceCheck 10
hold

t 10 -49  60
send stationKeeping
scienceCheck 10
hold

t 10 -49 70
send stationKeeping
scienceCheck 10
hold

# never got a positive signal, go home
t 10 -49   1
```

Table 4.4: Part of mission plan that descends until favorable scientific conditions are met, at which point the sampleSequence plan is executed (from May 26, 2007).

4.3.2 Direct Thruster Control

The Executive can also opt to disable the pose controller by issuing a Hold command. This suppresses the pose controller and allows the Executive to command the thrusters directly (in the worst case abort scenario). It also allows the Executive to use other modules to move the vehicle, for example to gently thrust forward or backward while sampling.

4.3.3 Random Walks

It is possible to use the Executive to perform a random walk. The mission specification must include a loop that contains only commands with relative motions and random intervals. Commands like "randomDir" which invokes the 4D Homing routine to turn a random amount make this technique possible. See Table 4.5.

4.4 Summary

In this section, we described how the Executive uses the capabilities of the control system, the pose estimation system, and the science payload, in order to yield an autonomous robotic vehicle which is capable of safely carrying out the mapping and scientific exploration of unknown underwater environments. We described the simple mission specification language in which simple commands can be combined into more sophisticated behaviors, such as autonomous wall sampling.

In the next section, we describe the series of tests and field experiments that we used to develop and demonstrate the capabilities of the DEPTHX robot, culminating in the exploration and scientific characterization of four flooded sinkholes.

```
#now we enter an infinite loop:

pause 1
dive 5
randomDir 115 135
run thrustForward
pause 30
kill thrustForward
randomWall 9 0 1

#now back away a bit
run thrustBackward
pause 30
kill thrustBackward

#just keep repeating until we run out of time
set missionSpecLine 11
```

Table 4.5: Part of a random walk plan (from May 27, 2007).

Chapter 5

Experiments

The DEPTHX vehicle was assembled at the Stone Aerospace facility near Austin, TX. Our first integrated tests of control and localization were performed in an acoustic test tank at the Austin Research Laboratory in the fall of 2006. During the winter of 2006-2007, we began our first long range tests in a flooded quarry, also near Austin, and began integration with the science payload. Our first cautious forays into a cenote were in La Pilita during two field expeditions in February and March of 2007. La Pilita is only about 100 m deep and is very accesible for launching and recovering the vehicle, so it made an excellent testing site. Finally, in May 2007 we mounted the final expedition to explore Zacatón – at the very tail end of the expedition we were able to explore both Caracol and Verde in less than two days.

5.1 Exploration of the Cenotes

Over the course of three field expeditions to Mexico, the DEPTHX vehicle explored four of the cenotes of Sistema Zacatón. Here we present some context about the operational sites, as well as the final geometric maps of the cenotes. In the next section we present the technical performance of the localization and mapping system.

5.1.1 La Pilita

La Pilita is the most easily accessible cenote, with an existing roadway and a flat, cleared space around the entrance for deploying the launch and recovery crane. In addition, the water level is just five feet below the lip, meaning that we could set up relatively permanent operations tents within direct sight of the vehicle, while it was near the surface.

The map of La Pilita shows that it was 118 m deep with a narrow (30 m) neck, and a distinct dome on the north-west side.

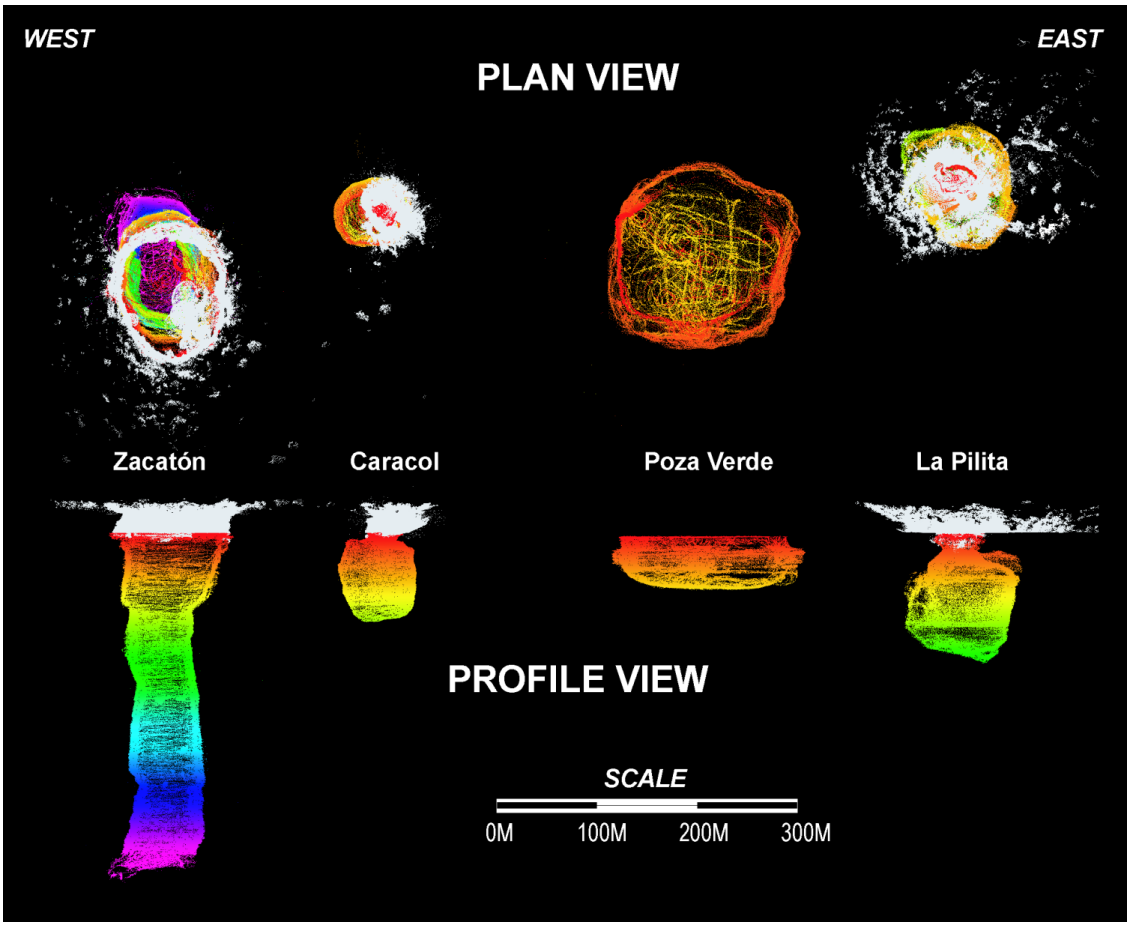


Figure 5.1: Plan and profile views of Sistema Zacatón, all distances are to scale. These point clouds include lidar data (in white) and sonar data (depth cued color).

5.1.2 Zacatón

Zacatón is the deepest cenote, and it is also the most difficult to access. Despite a good road up to the lip of the cenote, there is a 20 m cliff down to the water line. This meant that launching and recovering the vehicle was a more laborious and dangerous task. We based operations off of a small barge that could be tied off above the vehicle's dive point.

5.1.3 Verde

Verde is much more like a regular lake, its water has a completely different signature than the other cenotes. In fact it is believed to be a cenote that has been capped off, as if it had a narrow neck like La Pilita and Caracol (below) that has completely closed off Gary [2002]. Like Zacatón, Verde is ringed with cliffs.

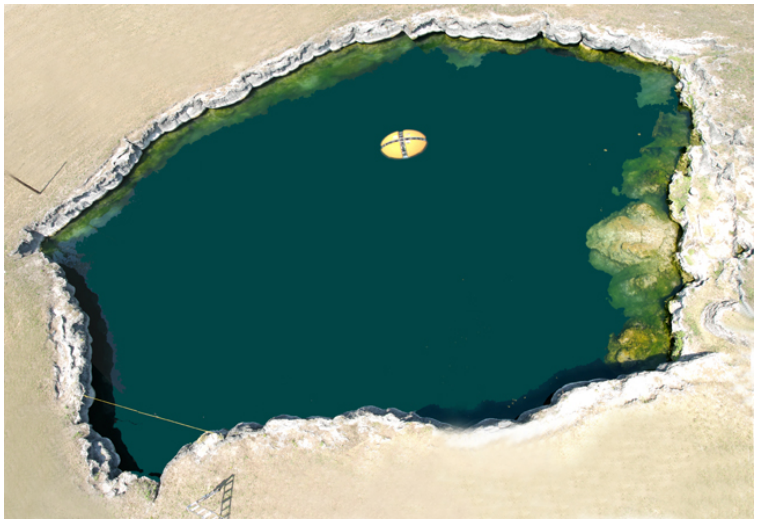


Figure 5.2: Aerial view of La Pilita, with the DEPTHX vehicle at the surface. The mouth of the cenote is only about 30 m across although it opens up to near 100 m diameter at depth. Courtesy Robin Gary and Jason Sahl.

Figure 5.3: Zacatón, deployment and the ops barge, showing 20 m cliff to the water level.

5.1.4 Caracol

Caracol is the smallest cenote, and it has the same type of narrow neck as La Pilita. It has tall cliffs on one side.

Verde and Caracol were explored in a single day each at the end of the field season.

Figure 5.4: Left: Verde Ops, Right: Caracol

5.2 Localization Performance

Here we describe the performance and results of the various localization and mapping methods used onboard the DEPTHX vehicle. We follow the natural progression from dead reckoning to mapping to localization with a prior map to SLAM. To illustrate each stage, we present results from our test locations, starting in a quarry in Texas before moving to the cenotes La Pilita, Zacatón, Caracol, and Verde.

5.2.1 Dead Reckoning

Dead reckoning is the basis of localization. While Localization and SLAM are intended to correct for gradual drift in the dead reckoning system, it is vital to make dead reckoning as good as possible.

Figure 5.5 shows the performance of the dead reckoning system described in Section 3.1 during a long raster mission to map a flooded quarry near Austin, TX [Fairfield et al., 2007a]. This mission had a total path length of over a kilometer, at the end of which the positioning error was 3.16 meters. This gives a respectable dead reckoning accuracy of less than half a percent of distance traveled. It should be noted that GPS fixes used as ground truth in this experiment were taken using a hand held non-differential GPS receiver from a moving boat, so it is likely that the dead reckoning estimate is actually more accurate than the "ground truth".

The red dots in Figure 5.5 denote locations where the DVL failed to achieve bottom lock. During this 6300 second mission, the vehicle was without DVL measurements for a total of about 780 seconds. Some of these dropout periods were as long as sixty seconds. Given this DVL performance, it would have been impossible to achieve any sort of dead reckoning estimate without patching in the IMU velocities as estimated by the Kalman filter, as described above.

We did our initial testing of the dead reckoning system in the quarry near Austin, TX. However,

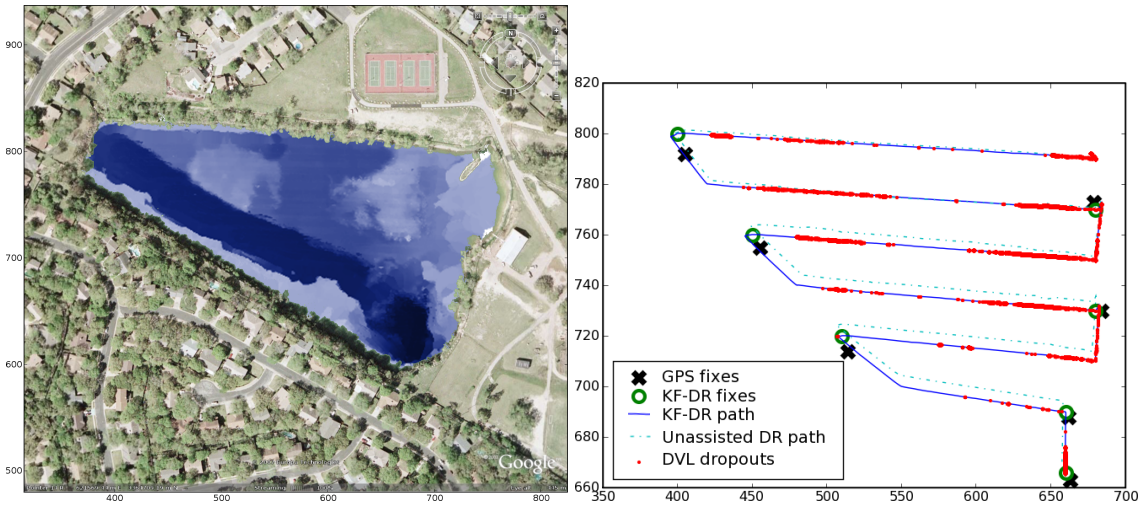


Figure 5.5: Left: A bathymetry map of the quarry, created by interpolating the sonar data from several raster survey missions, darker blue is deeper, max depth in the south corner was about 15m (Background satellite imagery courtesy Google, ©Europa Technologies, 2007). Right: a comparison of dead reckoning localization performance with and without the IMU velocity Kalman filter over a raster scan mission of the quarry. Coordinates are in truncated UTM northings and eastings.

due to the differences in the operating modes in a "shallow" quarry versus the deep cenotes, we did not discover several issues until we began testing in La Pilita. First, a bug in the IMU driver was causing intermittent latency issues. Second, a coordinate transformation issue caused bias in the global frame solution during high-rate rotations – such as our standard spinning descent for sonar coverage.

Barring a few instances when the DVL lost lock for extended periods of time (usually due to unusual maneuvers very near the walls), the dead reckoning error was barely detectable using our crude ground-truth method, which involved manually driving the vehicle underneath a plumb-bob at the beginning and end of each mission.

Over the open water missions with known start and end positions, which varied in length from 400 m to 3400 m and totaled over 9000 m, the dead-reckoning system drifted on average 0.07% of distance traveled. This is the equivalent of drifting by 0.7 m after traveling 1000 m. Note that this is loop-closure error – the robot's ability to return to its starting location – which is usually significantly better (lower) than the true position error due to symmetries in several of the common sources of error, such as heading bias and velocity scaling.

5.2.2 Mapping

The quality of the dead-reckoning was usually very good, but the range data from the sonar array was sparse, noisy, coarse, and low rate. Furthermore, the noise signatures of the sonars varied wildly between cenotes, being best in Caracol and worst in Verde, but generally poor (Figure 5.6). As discussed above, the evidence-grid approach is well suited to this type of noisy data because it can combine multiple readings to reject the noise (Figure 5.7). Using the dead-reckoning and

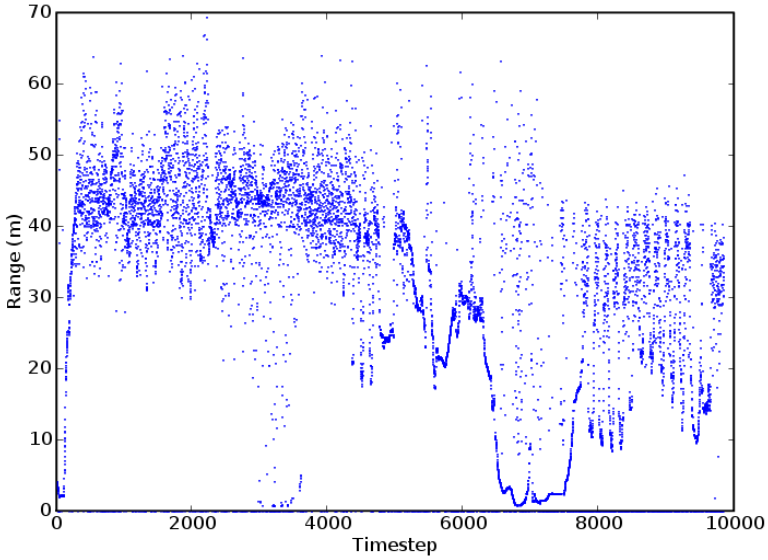


Figure 5.6: This plot of ranges from a single sonar transducer over time attempts to convey the level of noise in the sonar ranges. This is a forward-facing sonar, and between timesteps 6000 and 8000 the vehicle was fairly close to a wall, so the sonar ranges are reasonably good. Much of the rest of the time it is hard to see any signal in the noise, though the sinusoid after timestep 8000 is due to vehicle rotation.

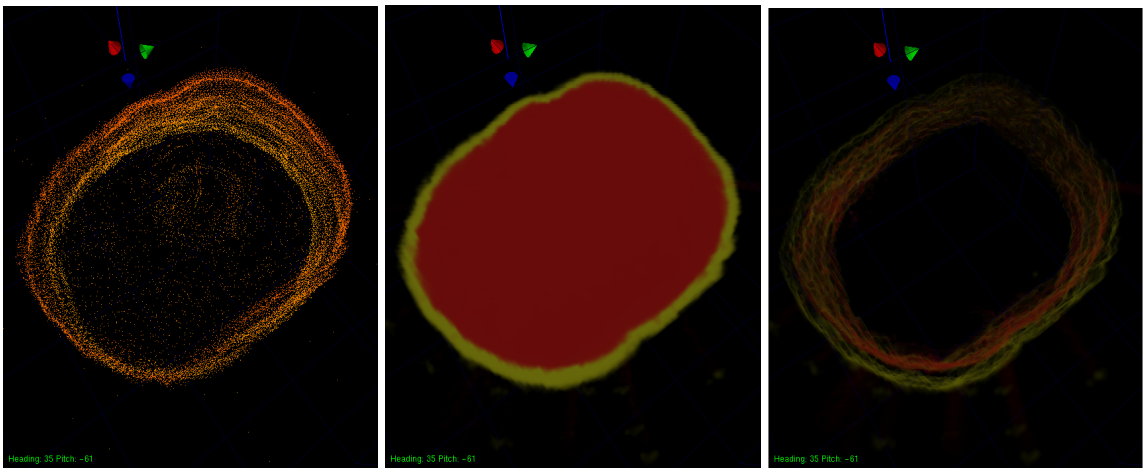


Figure 5.7: An example of how evidence grids cope with spurious sonar range values. Left, a slice from a noisy point cloud. Middle, a representation of the evidence grid where red shows low probability of occupancy, and yellow high. Right, the same evidence grid, but showing low and high occupancy isosurfaces – note that the noisy ranges inside the cenote have been filtered out.

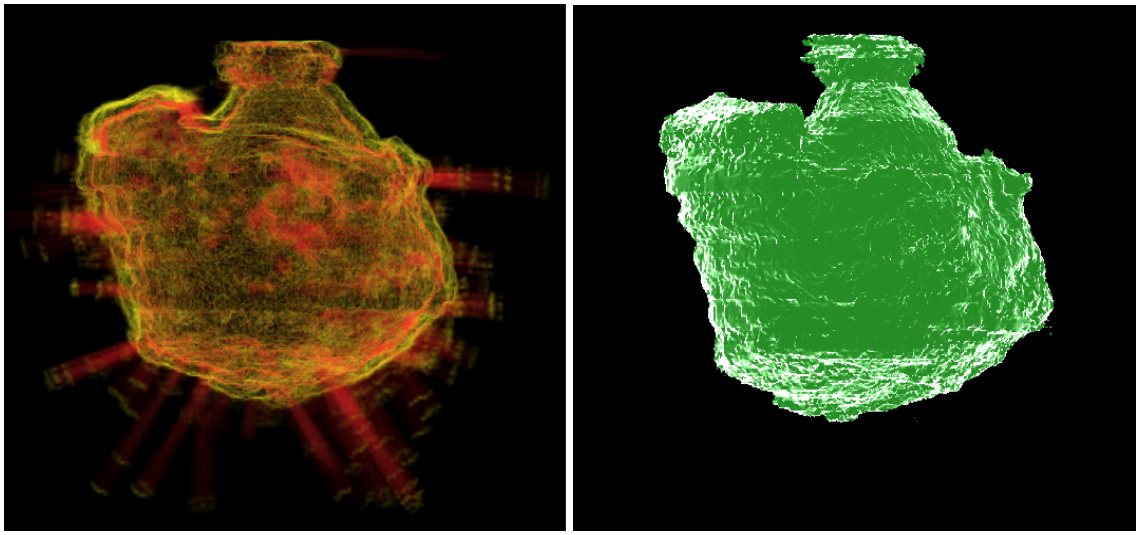


Figure 5.8: Left: An east-facing orthogonal view of an evidence grid of La Pilita. The maximum depth of La Pilita is 117 m, and its width is about 100 m, while the narrow neck that opens to the surface is only about 30 m wide. In this figure, yellow indicates an occupancy isosurface, and red indicates a vacancy isosurface. Right: The same evidence grid, converted to a mesh by extracting an occupancy likelihood isosurface, and shaded with a fresnel shader to bring out surface details.

sonar data collected during long missions into La Pilita, we constructed an evidence-grip map, which could also be converted to a mesh by extracting an occupancy isosurface, for ease of human interpretation (Figures 5.8 and 5.9). In the evidence grid representations, individual sonar beams caused by spurious sonar measurements can be seen projecting through the surface of the cenote (Figure 5.8:Left). From an algorithmic standpoint, these spurious measurements do not usually degrade the quality of the map, since the holes they bore in the cenote walls are patched by other sonar measurements, as can be seen when the evidence grids are converted to meshes by isosurface extraction (Figure 5.8:Right). Since there is no ground truth for the cenote geometry, we can't make firm statements about the map quality – however by comparing maps from multiple missions, we are convinced that the maps are at least consistent.

5.2.3 Localization

Particle filter localization with a prior map, usually a map built using an optimized dead reckoned trajectory, is faster and less memory intensive than full SLAM because there is only one map which is not updated in real time. Localization is also more robust than SLAM, partly because the reduced computation requirements mean that more particles can be supported, but primarily because the static map reduces local minima in the particle weighting function.

Generally, dead reckoning was just too good over the 4 hour mission periods to detect significant errors, at least with the very crude start and end point ground truth that we used. However, during our longest mission to the bottom of Zacatón, the IMU had an internal pitch/roll fault which caused it to report $\pm 2^{\circ}$ pitch and roll excursions. Although the vehicle managed to return to the

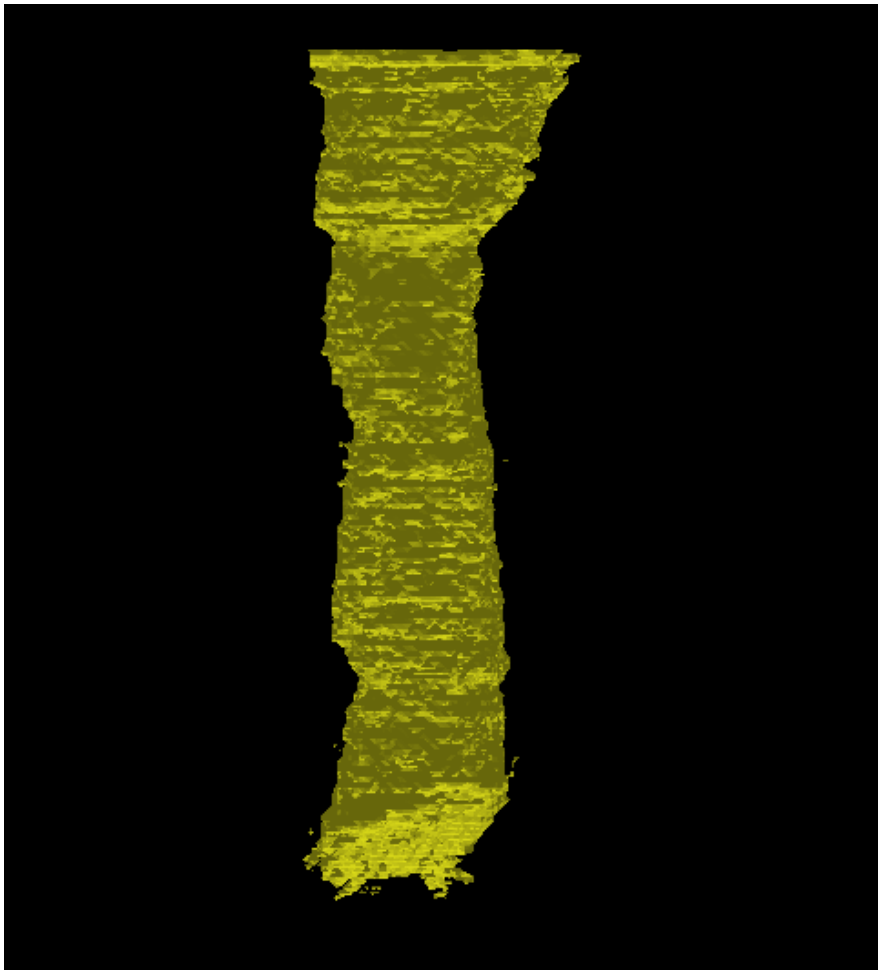


Figure 5.9: Side view of full depth (318 m) mesh map of Zacatón. In this rendered image, the evidence map of Zacatón has been converted to a mesh by extracting an occupancy likelihood isosurface.

surface, dead reckoning error was 12.5 m (0.76 % of DT) due to the incorrect attitude data.

On the other hand, a second mission to the bottom of Zacatón exhibited excellent dead reckoning performance, returning to the origin with 0.36 m error after traveling 1760 m (0.02% of distance traveled), with very few DVL dropouts (and no IMU fault). It would be very difficult to demonstrate any improvement from localization or SLAM on this second mission without more ground truth information.

To test our method's ability to localize in the face of erroneous sensor data, we built a map with data from the second mission (using the excellent dead reckoning), and then attempted to perform particle filter localization with the data from the mission with the IMU fault. With a modest 300 particles, localization yielded a 0.2 m drift after traveling 1644 m (0.01 % of DT), compared to the dead-reckoning drift of 12.5 m (0.76 % of DT) (Figure 5.10).

A simple test of the localization system is to take a map (constructed by using optimized dead reckoning), initialize the particle cloud over a broad area, and then see if the filter can use the map

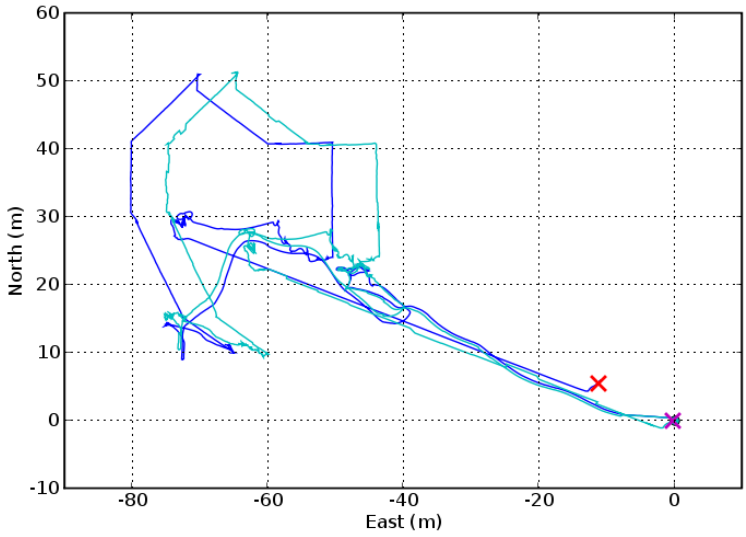


Figure 5.10: This example of localization in Zacatón shows how the particle filter is able to use the prior map to reject corrupt attitude data: dead reckoning in blue and localization (with the bad data but with a good map) in cyan. The 'return to origin' error for dead reckoning is 12.4 m and for localization is 0.2 m.

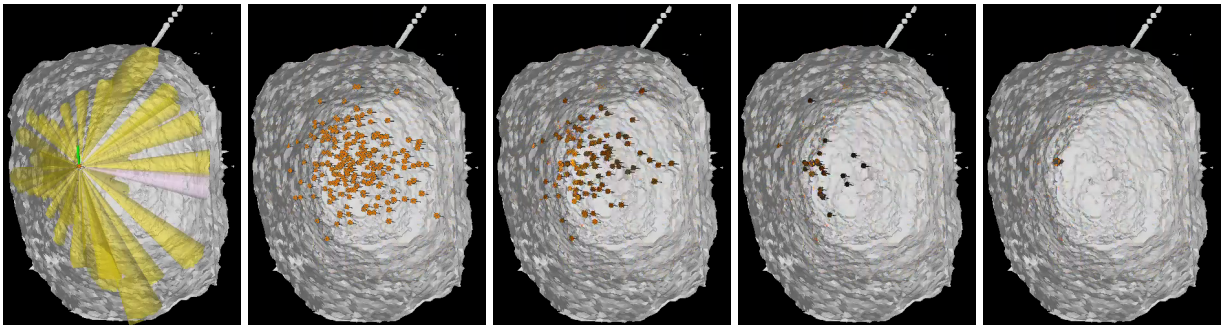


Figure 5.11: Kidnapped robot example in Caracol.

and the new range data to quickly and reliably converge to near the known true position. This test was easy to run, and was quite successful in the simple geometry of the cenotes (Figure 5.11).

5.2.4 SLAM

We ran SLAM onboard the vehicle in La Pilita, Zacatón, and Caracol. We also ran Localizaton with a prior map as a preliminary step. Generally the onboard CPU ran with between 150-250 particles (manually adjusted to accommodate CPU load). During our onboard tests, SLAM was configured to be very conservative and to fall back to DR if the solutions diverged significantly. This was largely due to a compressed testing schedule.

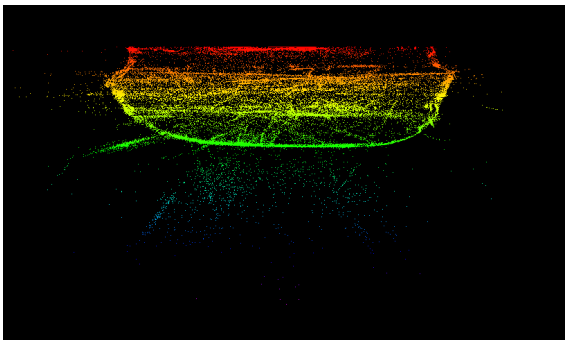


Figure 5.12: This thin slice of the sonar pointcloud from Verde illustrates the high level of sonar noise present in Verde – the interior of the cenote should be empty! The denser center band of noise is due to the fact that the vehicle spent most of its time operating within that depth band.

La Pilita

Although it exhibits much of the same morphology and biology as Zacatón, La Pilita is only about 100 m deep. Because it was smaller and more accesible, we did most of our initial testing in La Pilita (Figure 5.2).

We ran SLAM onboard the vehicle in La Pilita with 300 particles and used it to provide the real-time pose to control the vehicle (Figure 5.8). The lack of ground truth makes it complex to make any strong assertions about the accuracy of the SLAM solution, except that it was reliable and comparable to dead reckoning without any DVL dropouts: both solutions usually differed by less than a meter upon returning to the vehicle's starting location (our only ground truth point).

Verde

Verde has a different morphology than the other cenotes, 150 m across and only 30 m deep, an example of a more typical lake environment where the mapping task could be better accomplished by a different robot design. Furthermore, the temperature profile in Verde shows distinct layers, more like a typical lake. Perhaps due to these conditions, the acoustic environment in Verde was poor, resulting in noisy sonar readings (Figure 5.12) and frequent DVL dropouts.

Caracol

As we have seen, the dead reckoning performance is generally very good. However, when the DVL loses bottom lock and the vehicle loses its accurate velocity data, dead reckoning degrades significantly. The vehicle often lost bottom lock during two (common) situations: while it was on the surface waiting for the next mission, and during near-wall operations. These intermittent DVL dropouts caused enormous dead reckoning errors, despite our attempts to substitute the IMU's velocity estimate.

We have several metrics for evaluating the vehicle's navigation performance. As previously described, we manually drove the vehicle under a plumb-bob at the start and end of the mission, which yielded the loop closure error. Another metric is based on the map: intuitively, if the vehicle

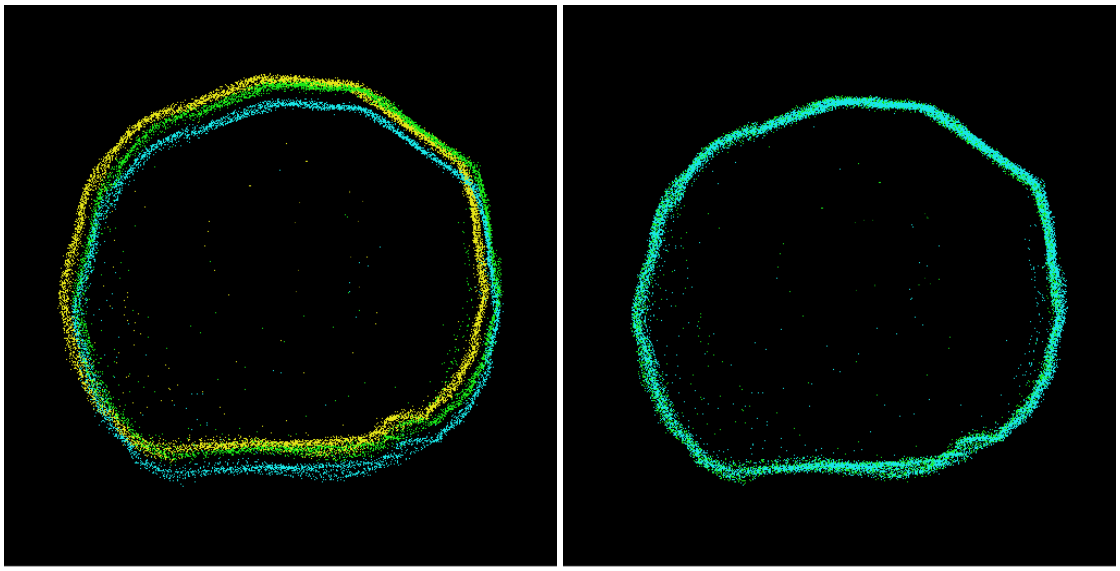


Figure 5.13: A top-down view of a slice of the dead reckoning (left) and SLAM (right) pointclouds constructed over the course of several missions (color coded). Dead reckoning shows significant drift, while SLAM keeps the map properly aligned.

has accurate navigation then a map produced with that navigation will be high quality, with none of the offset surfaces that indicate position drift (Figure 5.13). While this "map quality" is easy to observe, a more objective metric is that of map entropy. The information-theory entropy H of the evidence grid map Θ is the sum of the entropies of each voxel θ_i, where $p_i = p(\theta_i)$ is the probability that voxel θ_i is occupied:

$$H(\Theta) = -\sum_i p_i \log(p_i) - (1 - p_i) \log(1 - p_i)$$

This is because evidence grid maps represent space with a grid of independent binary random variables indicating occupancy. By constructing evidence grid maps with various navigation solutions, we can compare the resulting entropy of the maps. Maps with low entropy are very "certain", that is they are composed of sharply delimited regions of very low or very high probability of occupancy. High entropy maps are blurrier, with regions closer to 50% probability of occupancy.

The vehicle explored Caracol over the course of two hours, running several missions and returning to the surface between missions. Dead reckoning loop-closure error was 2.7 m due to frequent DVL dropouts. SLAM loop-closure error was only 0.3 m. Similarly, a the map drift (compared visually in Figure 5.13) and the more rigorous SLAM map entropy were both significantly lower: Table 5.1.

5.2.5 Discussion

The onboard SLAM tests in La Pilita indicate that our method to works, despite the low resolution, noisy, sparse, and low rate range data afforded by the pencil beam sonars. Our results in Caracol

Algorithm	Entropy	Loop error
Dead reckoning	38436.4	2.68
20p SLAM	33317.7	1.33
50p SLAM	32045.2	0.41
150p SLAM	32312.7	0.29
300p SLAM	31917.0	0.26
600p SLAM	31952.3	0.25

Table 5.1: SLAM results in Caracol

demonstrated the ability to resist bad velocity data (due to DVL dropouts), resulting in a significant improvement compared with dead reckoning.

We have shown the localization and mapping results of exploring four flooded cenotes. Dead reckoning was very good, except during DVL dropouts or IMU faults. Localization with a prior map was able to provide robust position information, even in the face of bad sensor data. SLAM (with no prior map) was more sensitive to sensor noise, but was able to improve on the dead reckoning during DVL dropouts. Importantly, our method worked on noisy, coarse, sparse, and low rate sonar data.

Chapter 6

Conclusions

In this report we have described the design and operation of the DEPTHX autonomous underwater vehicle. Specifically, our contributions include

- Iterative exploration – the operational procedures for safely exploring an unknown environment
- Autonomous sampling – including both discriminative site selection and reliable core and water sample collection
- 3D SLAM – localization and mapping in large natural environments with unknown structure
- Sparse SLAM – using sparse, noisy, low resolution and low rate sonar data to perform SLAM
- Executive – a layered safety contingency system and flexible mission
- Maps – the first geometric, environmental, and scientific maps of Sistema Zacatón specifications for mixed pre-scripted and autonomous missions

We have demonstrated autonomous control, localization, and mapping with the DEPTHX hovering autonomous underwater vehicle, which explored four flooded cenotes in Mexico. The vehicle dove hundreds of meters, used sonar to build accurate 3D maps of cenote geometry, collected scientifically valuable environmental data, images, water samples, and core samples, all with no human interaction. The DEPTHX project has broken new ground in underwater scientific exploration with autonomous robots.

Acknowledgments The DEPTHX project was managed by Stone Aerospace, William Stone Principal Investigator, and supported by the NASA ASTEP program, grant NNG04GC09G, Carl Pilcher, Program Scientist, and David Lavery, Program Executive.

We would like to thank Bill Stone and John Kerr at Stone Aerospace, as well as Marcus Gary at the University of Texas.

Appendix A

Operations

DEPTHX is a research vehicle with an unpolished operator interface. For example, mission plans and configuration files are updated using a text editor and logs are parsed with command-line utilities. We use three more user-friendly programs to monitor and control the robot during testing: a process manager, a 3D visualizer, and a camera image viewer.

We use the Microraptor process manager to manage the software modules running onboard the robot's two primary computers. Microraptor displays a summary of all vehicle subsystems, and can be used start and stop processes A.1. Microraptor is also used to switch the vehicle into autonomous mode at the beginning of autonomous missions.

The Drake 3D visualization program provides an intuitive representation of of the vehicle pose and live sonar data, as well as a short-term point-cloud map (Figure A.2). We initially developed Drake to debug the SLAM system, and its map representations, but Drake evolved into a flexible tool for 3D visualization (see `http://drake-3d.sourceforge.net`).

The image viewer, developed by Southwest Research Institute, controls the payload's onboard video capturing system. It displays images from the two main cameras: one on the end of the probe arm, and one back on the main body of the robot (Figure A.3).

We also use a Bluetooth gamepad for controlling the vehicle manually. This joystick is connected to an operator's computer, which relays the commands to the robot over the fiber-optic tether. The joystick is used on the surface to move to an extraction location or a reference point. When tethered, we also use it for driving at depth. This requires careful coordination with Microraptor, Drake, and Image Viewer for operator feedback.

A Day in the Life of the Vehicle Each day began with vehicle insertion and checkout. This involved land-side preparation including removing charging cables, replacing sampling containers and powering all the devices. Next the vehicle was lifted by a crane and stabilized for the IMU calibration, or alignment, procedure. Once all the software modules and sensors were ready, the vehicle was lowered into the water. At this point the performance of each device was checked (humidity levels, sonar returns, thruster performance, etc). Once satisfied with the vehicle behavior, the buoyancy was fine-tuned to be slightly positive and the vehicle was moved (using the joystick) to a fixed reference point.

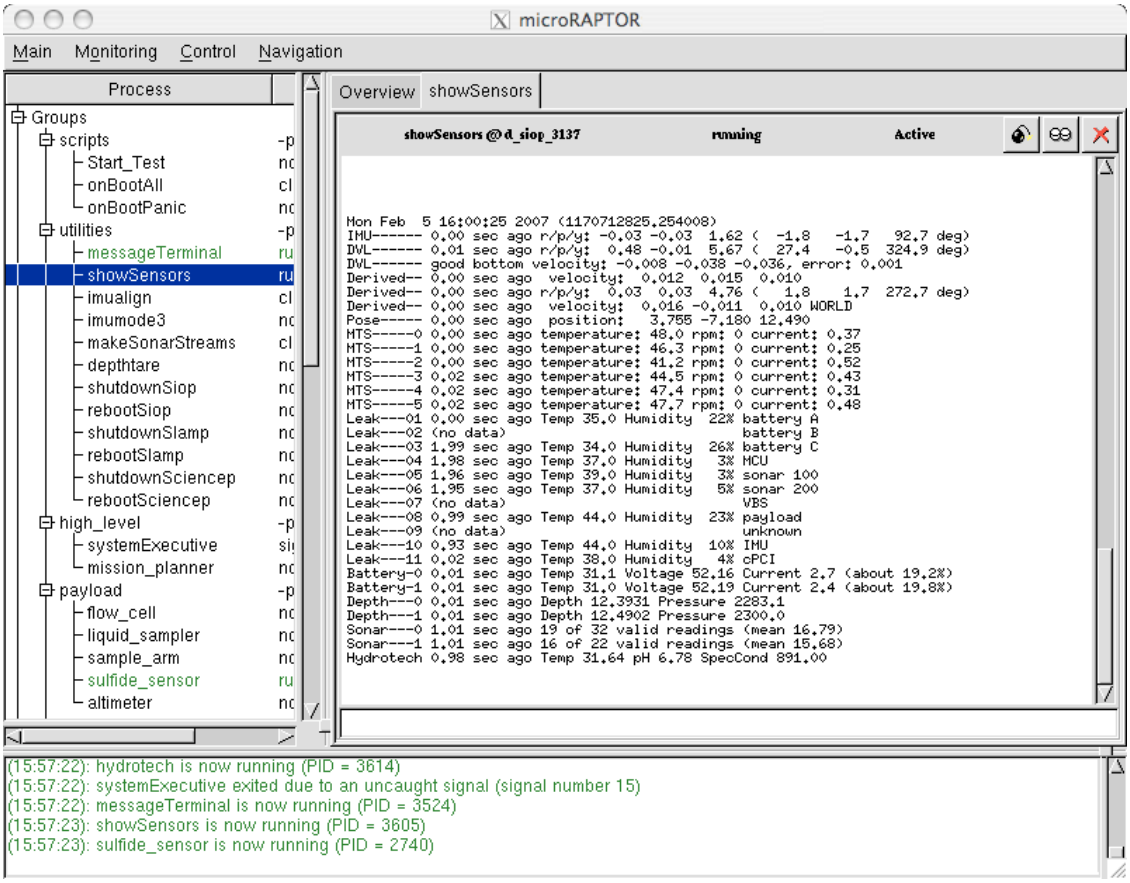


Figure A.1: Microraptor displayed textual information from each of the vehicle's onboard software modules, as well as a summary of vehicle status.

Early operations in each cenote emphasized short interactive operations near the surface. This required a minimum of three people. One person was needed for the careful and laborious tether management. A second person handled the joystick and watched the vehicle using Drake and Image Viewer. The third person used a laptop to monitor vehicle state using Microraptor and broker control between the joystick and the autonomy system.

Although a dead-man switch was on the joystick, transitioning to or from joystick control involved additional steps. When changing to joystick control, autonomous navigation had to be disabled by issuing a hold command to the pose controller. To resume autonomy, we first enabled station-keeping to safely "park" the vehicle and stopped joysticking. Then manual commands could be issued to the pose controller, or a mission could be started.

After thoroughly exploring the surface and probing some deeper areas, we continued our iterative exploration strategy. With an initial map of the upper area we generated mission plans to dive deeper and visit waypoints near the edges of the map. As the vehicle dove further and further we improved our map of the cenote.

These probing missions were autonomous, though not always untethered. We often left the

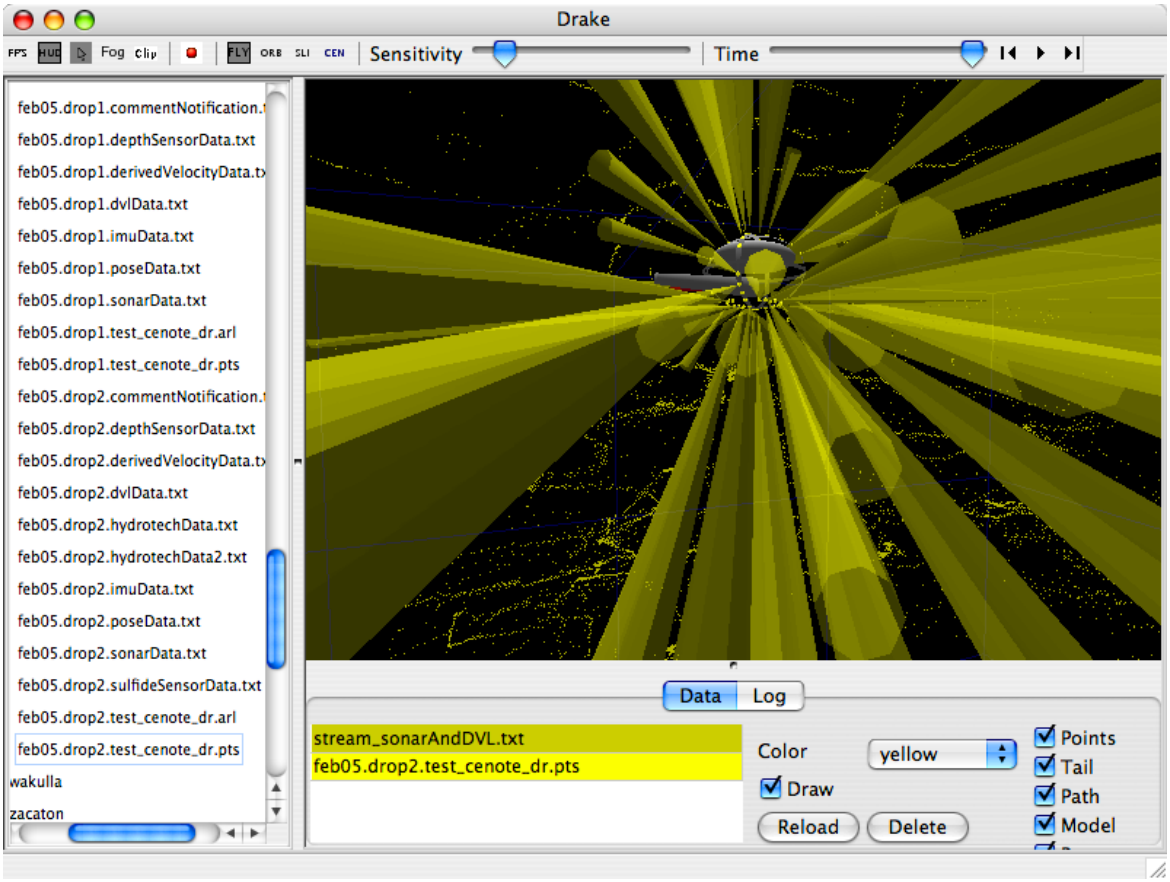


Figure A.2: Drake provided an interactive 3D visualization of the vehicle pose, as well as the sonar range data (shown as yellow cones). Drake also displayed a local map (shown as yellow dots), which gave the operators a sense of the local environment.

communications tether on so that we could see new data live. This drastically increased our productivity because we could start missions at depth and avoid the time-consuming process of ascending and transferring data before the next mission could be generated. The big drawback was a limited area of operations: we had to avoid areas where the tether could snag. Those areas were considered the most dangerous and deferred until safer regions were examined first.

During the autonomous missions we relied on the health monitor and various abort strategies to bring the vehicle to the surface. These were carefully defined regardless of whether or not the tether was attached. Operators were not solely relied upon due to the sheer volume of data and sporadic communication failures (due to loose cables and loss of power to the repeater). Though some false positives were triggered, it was possible to easily clear faults and resume the mission if the tether was connected.

Figure A.3: Image Viewer displayed images captured by the science payload's onboard cameras.

Appendix B

Daily Logs

Summary of the daily logs of our field tests from August 2006 till May 2007:

August 2006: In this first dive test of the DEPTHX vehicle our goal is to measure and calibrate our thruster models and perform coordinated control of the vehicle. At the same time we intend to confirm sensor models and accurately dead reckon the vehicle's path in the test tank.

December 2006: The first set of experiments in a natural environment: a flooded quarry.

January 2007: This will be the second set of experiments in a natural environment. We would like to completely shake-out the system here before taking it to a real cenote. Ideally, this will involve a demonstration of each critical component followed by a science-driven mission.

March 2007: Our objective this week is to complete testing of the vehicle and payload and to conduct a scientific survey of la Pilita. The investigation of la Pilita also serves to prepare and validate the methods we will use to explore cenote el Zacatón.

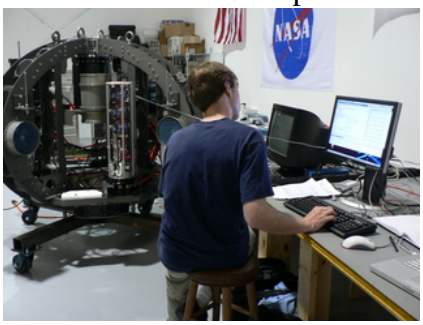
February 2007: This trip commences field operations at cenote la Pilita. This 100m deep sink hole boasts 90 degree water, overhanging rock and complex biology. We will run long autonomous missions to map the area and collect novel scientific data.

May 2007: This was the final trip for the project and the culmination of our endeavour. We explored Zacatón, the deepest flooded sinkhole in the world, and made the first complete map. We also made brief dives into Verde and Caracol, thus mapping the entire system.

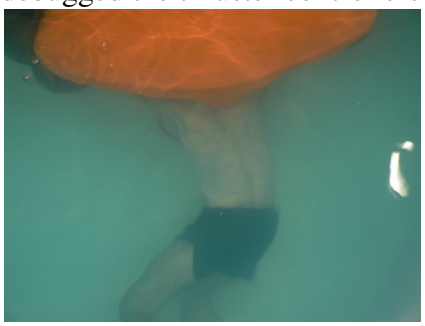
B.1 August 2006: SAS and ARL

```
Monday, July 31 - Friday, August 4, 2006
Stone Aerospace (SAS) and Applied Research Lab (ARL), Austin, Texas
```

In this first dive test of the DEPTHX vehicle our goal is to measure and calibrate our thruster models and perform coordinated control of the vehicle. At the same time we intend to confirm sensor models and accurately dead reckon the vehicle's path in the test tank.

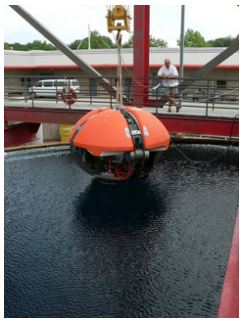



Monday and Tuesday were mostly spent debugging. The entire system had not yet been run from the batteries. Electrical problems and low level communications had to be addressed. The team split into three groups; one examined the batteries, another looked at the Doppler Velocity logger (DVL) and a third debugged the thruster controllers (MTS).

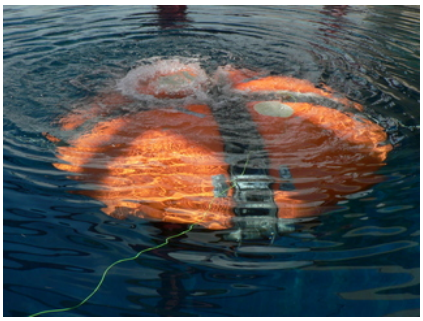




By the end of day on Tuesday, many of the electronic and communication issues had been addressed. The vehicle was sealed up and put in the small tank. Once submerged, we distributed weights to balance the robot and neutralize buoyancy. We moved on to a trailer for transport to the Applied Research Labs (ARL) test tank.

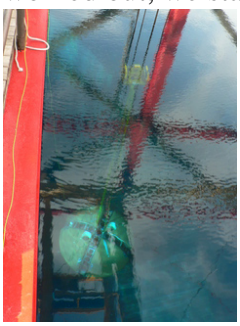
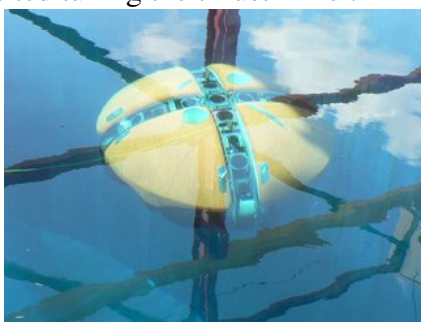
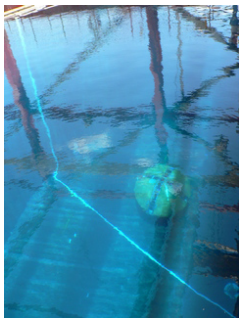




On Wednesday morning the robot arrived at ARL. The DVL's power problem was resolved with a slow-blow fuse courtesy of ARL. We dunked it into the tank. The good news: nothing leaked, the fibre optic worked great. The bad news: thrusters were unuseable, IMU didn't align, wireless didn't work. Such problems are routine during initial checkout of a new robot. After some debugging we were able to fire thrusters individually and IMU alignment was worked around.

We conducted our initial system identification tests; firing each thruster and observing the vehicle's response. Thruster debugging continued into Thursday, and by the end of the day we could reliably command individual thrusters in torque mode greatly speeding up the system identification process. We also got an unexpected surprise; 2 of the thrusters had been swapped. Once that had been worked out, we started tuning the thrust mixer.

On Friday we used the crane to drag the vehicle and collect sonar and inertial sensor data. We felt confident in the thruster mixing table and ran the vehicle untethered, collecting more accurate thrust-response data. Finally we pushed the vehicle down to the bottom of the tank, reaching a depth of 9 meters.

B.2 December 2006: SAS and the Quarries

`Wednesday, December ?? - Friday, December ??, 2006`
`Stone Aerospace (SAS) and The Quarries at Hyde Park Baptist Church, Austin,`

Over the course of this week we will learn how the vehicle operates in a natural environment.



Tuesday the 12th was spent familiarizing ourselves with the quarry and setting up operations. The major activity was transporting the vehicle and lowering it into the water. We also had tents to

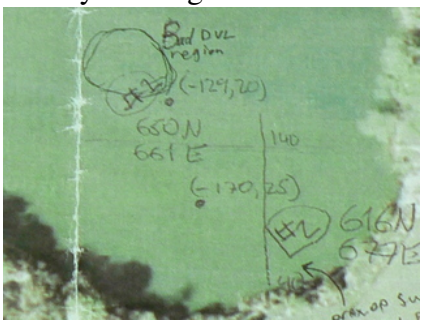
pitch, a satelite dish to mount, computers to set up and myriad of other activities. By the end of the evening we were ready to go.
INCOMPLETE

B.3 January 2007: SAS and The Quarries

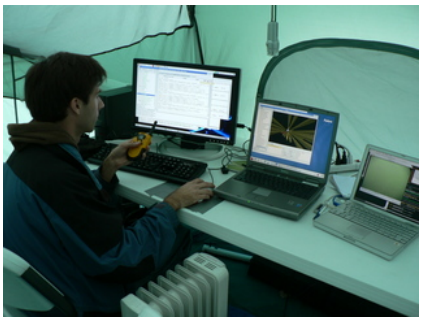
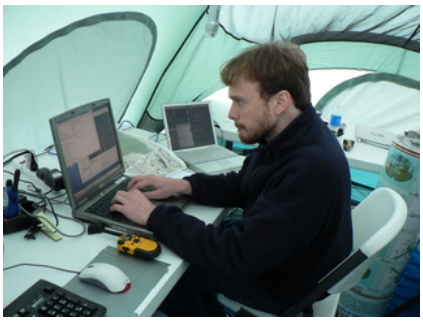
Wednesday, January 3 - Friday, January 12, 2007
Stone Aerospace (SAS) and The Quarries at Hyde Park Baptist Church, Austin,

This will be the second set of experiments in a natural environment. We would like to completely shake-out the system here before taking it to a real cenote. Ideally, this will involve a demonstration of each critical component followed by a science-driven mission.

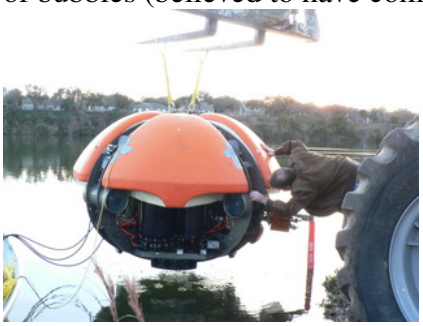

Wednesday and Thursday was spent in the lab. There was a problem with the payload computer which prevented us from using the payload until Friday. In the meantime we sealed the vehicle, did a lab checkout, synced the system clocks, and improved the sonar and hydrotech software. We finally arrived at the quarry on Thursday evening.

On Friday, January 5th we cleared the launch area and put the vehicle in the water. We immediately noticed spurious problems with the dead reckoning, though it was working well otherwise (it brought us back to within a meter of the starting point on a 200m run). Initially we believed the problem was with the IMU and tried several different alignment procedures over the next few days. We then determined that the problem was with the DVL, it would occasionally dropout, particularly in one region. By Monday we had a solution; a Kalman filter now helps smooth data streams from the IMU and DVL, it can handle short dropouts very well but long dropouts cause us to abort.

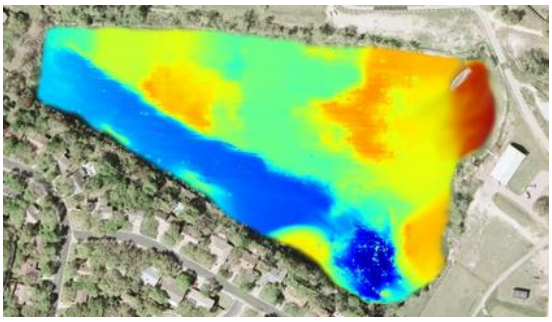



While debugging dead reckoning, we simultaneously continued work in other areas. Friday through Monday we tested new timeouts in a navigation control loop, collected sonar data (and noted a failed bank), improved wall approaches and exercised other functionality. We did hit a few problems however; on Saturday we found that we were negatively buoyant and observed a steady stream of bubbles (believed to have come from the VBS).




On Tuesday the 9th SWRI arrived and noted significant problems with the sample arm and flow cell. The following day a film crew was present getting interviews throughout the day. On these days we collected high-quality data sets on the north shore and in the southeastern regions. Also we operated after sunset, this was made possible by afternoon charging cycles (since the vehicle had to be on shore).


Finally Thursday the 11th was our last day. SWRI returned and John Spear arrived, so the morning was spent looking at the various sensors. We then ran two missions covering the entire quarry, the second one was completely untethered and mostly successful (the run aborted autonomously after detecting significant dvl dropouts). Based on those results, we believe we can safely operate the vehicle and generate a map of the environment.



This is a preliminary map of the quarry where depth is represented by color; red is at the surface and blue is about 12 meters (40 feet) deep. This data was pasted on top of a screenshot from Google Earth to give some context.

B.4 February 2007: Cenote la Pilita

`Sunday, February 4 - Sunday, February 11, 2007`
`Cenote la Pilita, Rancho la Azufrosa, Tamaulipas, Mexico`

This trip commences field operations at cenote la Pilita. This 100 m deep sink hole boasts 90 degree water, overhanging rock and complex biology. We will run long autonomous missions to map the area and collect novel scientific data.


February 4th, we finishing assembling the vehicle. We also did a preliminary software checkout that revealed some minor wiring mistakes. The rest of the day was spent transporting Clementine to cenote la Pilita. Rains made the trip a little tricky, but by the end of the day we were prepared to start operations.



With the robot in the water we started testing the basic functionality. Each thruster and sensor was checked in the new environment. We then did a static drop to test the pressure housings and collect

data for an initial map. The final activity was to “trim” the vehicle by adjusting weights to make it level and slightly buoyant.



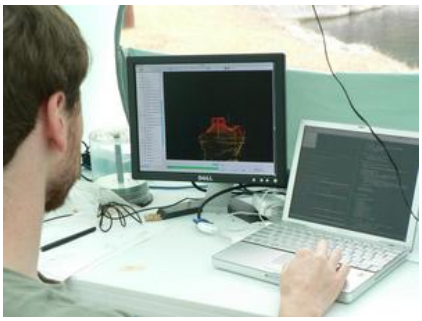
The 7th and 8th were spent debugging. We started driving autonomously and found some problems with the wall following behaviors related to the complex terrain. We also identified hardware and software issues with the science payload and started to characterize its performance. One of those issues was lighting, so we bolted on some dive lights and got good illumination.



The morning of the 8th was spent fixing some cabling issues and getting a group photo. After lunch we tested abort behaviors at a depth of 50 meters. The day culminated with a core sample at depth, we manually fired the instrument and collected some goo.



February 9th was an important day. The day involved more payload debugging, tuning the proximity faults and running short, untethered missions. At one point the tether got snagged and Marcus had to dive. We charged the batteries all afternoon, dialed in a long untethered mission then enjoyed dinner and a movie, followed by some nervous waiting. Around 1:30am Clementine returned; our mission took longer than expected and the executive aborted after 3 hours had elapsed.



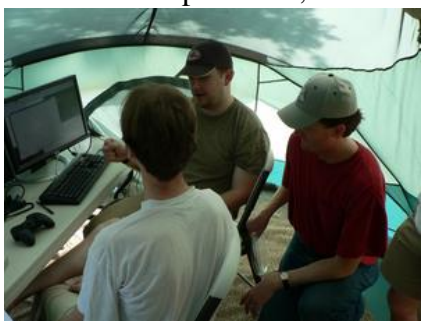
Our last day of testing. We continued trying science activities and ran another 3 hour autonomous mission. With yesterday's long run we generated a good map of the cenote. There were a few interesting bulges however, and with the remaining battery power we sent the vehicle to investigate. Around 6 we finished striking camp and drove the vehicle back to the ranch, to await our return.

B.5 March 2007: Cenote la Pilita

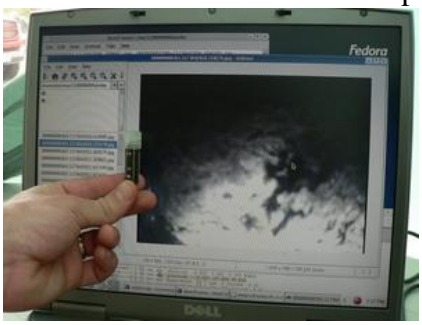
`Friday, March 9 - Thursday, March 15, 2007`
`Cenote la Pilita, Rancho la Azufrosa, Tamaulipas, Mexico`

Our objective this week is to complete testing of the vehicle and payload and to conduct a scientific survey of la Pilita. The investigation of la Pilita also serves to prepare and validate the methods we will use to explore cenote el Zacatón.


When we arrived the vehicle was waiting by the water and we immediately went to work. The morning was spent testing certain theories and code modifications. After charging the batteries, we resumed operations, this time testing the payload and core sampling strategy.

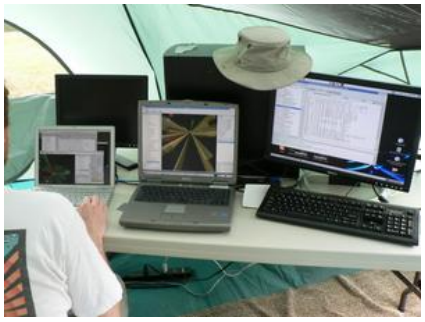
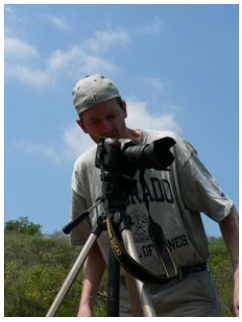



On March 10th we collected our first samples. The morning was spent finalizing sample protocol; doing dry runs and consulting with John Spear. Then in the evening we ran several untethered missions to collect science samples.



March 11th was a memorable day. In the morning we successfully collected a core sample at 80m depth, completely untethered. During the afternoon charge we analyzed data and made last minute modifications. The evening plan was a long untethered mission... from which the vehicle did not return.


Fortunately Marcus quickly located the vehicle at 17m depth and salvaged it! Last night's problem turned out to be a configuration error that resulted in the vehicle ramming the northern wall on its final ascent. After the batteries charged, we decided to sample the cenote floor. A pair of pooper-scoopers were attached and we retrieved mud from the bottom.

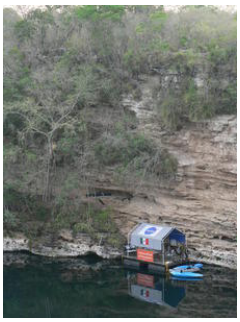
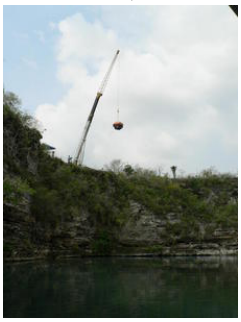



On the 13th we ran 2 missions into a bulge, this area was incompletely mapped and inaccessible with the tether attached. The final day included some remotely-operated activities as well as running SLAM. We ran down the batteries, took a group photo, and started packing.

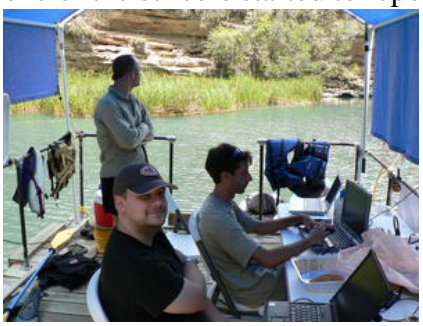
B.6 May 2007: Cenote Zacatón

```
Tuesday, May 15 - Monday, May 28, 2007
Cenote Zacat\'on, Rancho la Azufrosa, Tamaulipas, Mexico
```

This was the final trip for the project and the culmination of our endeavour. We explored Zacatón, the deepest flooded sinkhole in the world, and made the first complete map. We also made brief dives into Verde and Caracol, thus mapping the entire system.

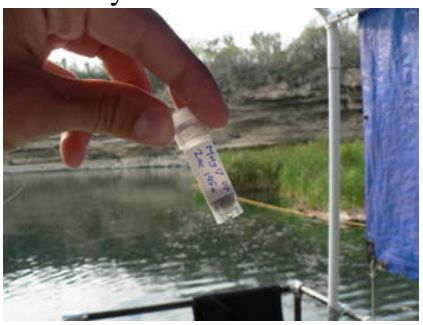




Tuesday, May 15th was our first day at Zacatón. We quickly familiarized ourselves with the barge and lowered the vehicle into the cenote. After a checkout period we manually dove to 281m (stopping periodically to check all the sensors) and found a sloping floor around 300m. Unfortunately one of the sensors started to report wildly noisy values and we had to ascend.

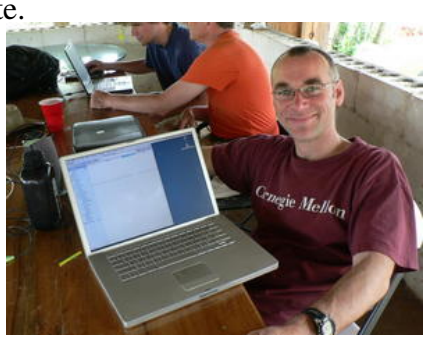



We put the vehicle back in the water on the 16th and did a checkout of the payload and autonomy software. With this additional functionality in place, we again dove deep and collected water samples along the way. This time we reached 290m and pushed under an overhang into the deepest section. The sonar data collected on this dive allowed us to create our initial map, leaving only a small segment of the very bottom unknown. The sinkhole appears to be a deep shaft with a sloping floor around 300m. According to our environmental sensors, the water column is quite uniform throughout.


The morning of the 17th was spent giving interviews to the press and shooting underwater video. Many reporters were present including crews from local stations, NASA and various magazines. By noon the fervor died down and we returned to our scientific investigation by starting a systematic study of the cenote walls.



After lunch we returned to the water and made a big push for science samples, collecting 5 water samples and 3 core samples (114m, 198m and 272m). The batteries for the lights died after 2 missions, so the final dive (200m) was autonomous, relying only on sonars. As we continue to analyze the samples from this mission, we expect they will provide a wealth of information about the environment within the cenote.

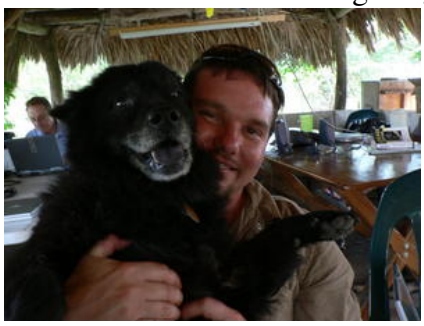



There were no operations between the 18th and the evening of the 25th due to unforeseen circumstances. We spent the down-time doing vehicle maintenance, analyzing telemetry, archiving data and generally catching up on work.

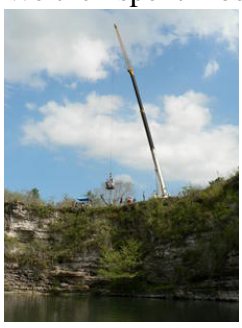
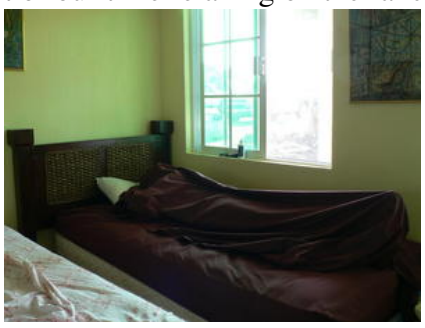




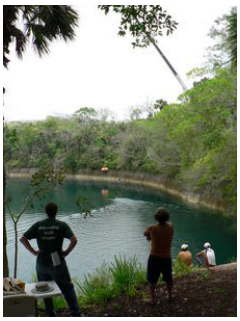

As the week wore on we spent more of our time exploring the area and learning about the region. We encountered incredible geologic formations, fossilized mammoth bones and beautiful wildlife.



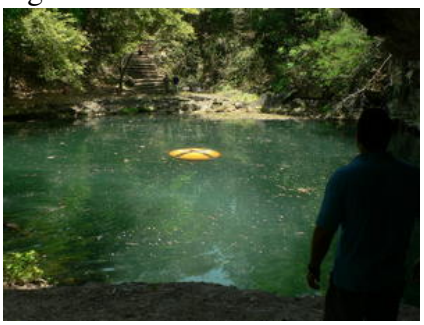
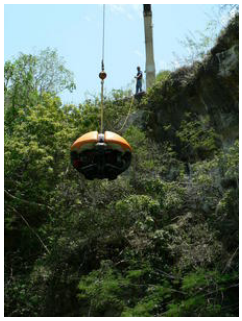
By the end of the week we had exhausted our "to do" lists and seen much of the local environment. We then spent most of our time relaxing on the ranch; playing music, reading and hanging out.


The evening of Friday, May 25th we redeployed into Zacatón. We ran three times in 36 hours, stopping only to charge batteries (and ourselves). During this whirlwind of activity we probed deeper than ever and collected several samples completely untethered. Operating the vehicle untethered represents the definitive test of the robot's capability to collect scientific and handle unexpected conditions without human guidance or monitoring.



On the 27th we deployed in Verde. We autonomously explored this large, shallow cenote using a randomized wandering planner. Throughout the day we collected environmental data confirming major chemical changes below 10m.

On the 28th we had a short deployment in Caracol. The small opening quickly opens into an 80m deep oval void strongly resembling La Pilita, which we explored thoroughly in February and March. The mapping sensors worked very well in this environment and we successfully ran an untethered mission using our simultaneous localization and mapping (SLAM) system.


On the afternoon of the 28th we packed and cleaned. Cleaning up proved to be a lot of fun; we had to release the floating zacate and find a new home for some puppies. Around sunset we left Rancho Azufrosa, concluding the DEPTHX field operations.

Appendix C

Lessons Learned

Use a fiber-optic tether for testing and debugging The fiber-optic tether was invaluable for testing and debugging the vehicle underwater. While we had to dedicate one person to handling the tether, the tether gave us an enormous amount of insight into the vehicle's condition, the flexibility to adapt our tests on the fly, and the confidence to push the vehicle systems beyond the comfort zone.

Find a way to establish ground-truth position It was a major oversight not to include a GPS unit, which we did on the grounds that it would only be useful on the surface. We underestimated the amount of time that we would spend testing the control and pose estimation algorithms near the surface, and as a result we were hampered by our lack of good ground truth.

Illumination is more important than image resolution Our ability to use the cameras was constrained by the illumination more than by the resolution of the images. We ended up strapping more lights on the robot, but these last minute alterations were not rated for use at depth.

Simplify vehicle dynamics By changing the physical design of the robot and its actuators and by restricting the set of maneuvers, we were able to dramatically simplify the control system without compromising any operational functionality. A more specific lesson for a hovering vehicle is that while it is natural to want to tune the control system to yield a snappy response, this really isn't necessary for a slow-moving vehicle. We were able to dramatically improve the stability and power consumption of the control system by adjusting the dead-bands and gains for a more leisurely response.

Use control electronics that allow you to control thrust The DriveBlok control electronics were overly sophisticated: it was actually a struggle to get them to perform simple torque (or current) control, and while the thrusters were entirely capable of turning very slowly, we had to impose very large deadbands or the DriveBlok would throw internal limit errors and shut down.

Monitor temperatures We had several instances of IMU instability and faults – ultimately, the faults mostly went away. Almost all logs show strange IMU velocity anomalies, which may be symptomatic of yaw problems (that we can't detect directly). These anomalies were probably related to the age of the IMU and high operating temperatures (75 degC), which were in excess of the recommendation of 50 degC.

Test the DVL in all conceivable operating modes The most salient aspect of operating a DVL was the loss of "bottom lock", usually due to very rough surface geometry or very short ranges as the vehicle approached the wall. In Fairfield et al. [2007b], we described scaling the vehicle motion error model depending on the data quality from various sensors, such as the IMU and the DVL. We found that the quality of the DVL velocity estimates varied significantly, especially just before it lost bottom lock. Scaling the vehicle velocity motion model when the DVL error values indicated loss of lock allowed us to generally trust the very good dead reckoning (which was important for building up the map from the sparse sonar readings), while then providing appropriate particle diversity to handle the high amount of uncertainty during DVL dropouts.

Check water model assumptions Standard pressure to depth calculations for fresh water assume a constant coefficient, which is usually accurate enough for the shallow depths usually associated with rivers and fresh water lakes, but not for the cenotes. On the other hand, ocean depth calculations are often based on a fourth order polynomial least squares fit to the true density integral, using the Standard Mean Ocean Water model (35 practical Salinity, 0 degrees Celsius) based on the international equation of state of seawater (EOS-80) Fofonoff and Jr. [1983]. Using the standard conductivity to salinity conversion of the practical salinity scale (PSS-78, which ignores the possibility that the exotic ion composition of the cenote might skew the salinity, and thus the density), the cenotes had near zero salinity (0.45 S), but temperatures around 30 degrees Celsius. Using this new water model, we found that the depth calculations were about 2 m deeper than the basic freshwater calculation at 300 m depth.

Bibliography

S. Arulampalam, S. Maskell, N. Gordon, and T. Clapp. A tutorial on particle filters for on-line non-linear/non-gaussian bayesian tracking. *IEEE Transactions on Signal Processing*, 50(2): 174–188, February 2002.

A. Bachmann and S. Williams. Terrain aided underwater navigation - a deeper insight into generic monte carlo localization. In *Proc. of the Australasian Conf. on Robotics and Automation*, 2003.

H. Baker. Minimizing reference count updating with deferred and anchored pointers for functional data structures. *ACM SIGPLAN Notices*, 29(9):38–43, 1994. ISSN 0362-1340.

D. Bradley, D. Silver, and S. Thayer. A regional point descriptor for global localization in subterranean environments. In *IEEE Conf. on Robotics Automation and Mechatronics*, December 2004.

J. Bresenham. Algorithm for computer control of a digital plotter. *IBM Systems Journal*, 4(1): 25–30, 1965.

R. R. Burridge, A. A. Rizzi, and D. E. Koditschek. Sequential composition of dynamically dexterous robot behaviors. *Int. J. Rob. Res.*, 18(6):534–555, Jun 1999.

C.F. Chyba and C.B. Phillips. Possible ecosystems and the search for life on europa. *Proc. Natl. Acad. Sci. USA*, 98(3):801–4, 2001.

C.F. Chyba and C.B. Phillips. Europa as an abode of life. *Origins of Life and Evolution of Biospheres*, 32:47–67, 2002.

R. Craig Coulter. Implementation of the pure pursuit path tracking algorithm. Technical Report CMU-RI-TR-92-01, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, January 1992.

A. Doucet, N. de Freitas, K. Murphy, and S. Russell. Rao-blackwellised particle filtering for dynamic bayesian networks. In *Proc. of the Sixteenth Conf. on Uncertainty in AI*, pages 176–183, 2000.

A. Eliazar and R. Parr. Hierarchical linear/constant time slam using particle filters for dense maps. In Y. Weiss, B. Schölkopf, and J. Platt, editors, *Advances in Neural Information Processing Systems 18*, pages 339–346. MIT Press, 2006.

R. Eustice, H. Singh, J. Leonard, M. Walter, and R. Ballard. Visually navigating the RMS Titanic with SLAM information filters. In *Proc. of Robotics Science and Systems*, June 2005.

N. Fairfield, G. Kantor, and D. Wettergreen. Three dimensional evidence grids for SLAM in complex underwater environments. In *Proc. of the 14th Intl. Symposium of Unmanned Untethered Submersible Technology*, August 2005.

N. Fairfield, D. Jonak, G. Kantor, and D. Wettergreen. Field results of the control, navigation, and mapping systems of a hovering auv. In *Intl. Symp. on Unmanned Untethered Submersible Technology*, 2007a.

N. Fairfield, G. Kantor, and D. Wettergreen. Real-time slam with octree evidence grids for exploration in underwater tunnels. *Journal of Field Robotics*, 2007b.

N. Fofonoff and R. Millard Jr. Algorithms for computation of fundamental properties of seawater. Technical Report 44, Unesco, Paris, France, 1983.

T. I. Fossen. *Marine Control Systems: Guidance, Navigation and Control of Ships, Rigs and Underwater Vehicles*. Marine Cybernetics AS, 2002.

D. Fox. Adapting the sample size in particle filters through KLD-sampling. *I. J. Robotic Res.*, 22 (12):985–1004, 2003.

M. Gary. Understanding Zacatón: Exploration and initial interpretation of the world's deepest known phreatic sinkhole and related karst features, southern Tamaulipas, Mexico. *Karst Frontiers, Karst Waters Institute Special Publication*, 7:141–145, 2002.

R. H. Gary. Anthropogenic activities and karst landscapes: A case study of the deep, thermal, sulfuric karst system in tamaulipas, mexico. Master's thesis, The University of Texas at Austin, 2005.

R. Greenberg. *Europa, the Ocean Moon: Search for an Alien Biosphere*. Springer-Praxis, Berlin, 2005.

V. Havran. A summary of octree ray traversal algorithms. *Ray Tracing News*, 12(2), 1999.

H. Hussmann, F. Sohl, and T. Spohn. Subsurface oceans and deep interiors of medium-sized outer planet satellites and large trans-neptunian objects. *Icarus*, 185:258–273, 2006.

W.J. Kirkwood, D. Gashler, H. Thomas, T.C. O'Reilly, R. McEwen, N. Tervalon, F. Shane, D. Au, M. Sibenac, T. Konvalina, A. Bahlavouni, and J.G. Bellingham. Development of a long endurance autonomous underwater vehicle for ocean science exploration. In *IEEE Oceans 2001*, pages 1504–1512, November 2001.

M. B. Larsen. High performance doppler-inertial navigation experimental results. In *Proc. of IEEE/MTS OCEANS*, pages 1449–1456, 2000.

J. Leonard, A. Bennett, C. Smith, and H. Feder. Autonomous underwater vehicle navigation. Technical report, MIT Marine Robotics Laboratory, 1998.

M. Martin and H. Moravec. Robot evidence grids. Technical Report CMU-RI-TR-96-06, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, March 1996.

Peter S. Maybeck. *Stochastic models, estimation, and control*, volume 141 of *Mathematics in Science and Engineering*. Academic Press, 1979.

M. Montemerlo, S. Thrun, D. Koller, and B. Wegbreit. FastSLAM: A factored solution to the simultaneous localization and mapping problem. In *Proc. of the AAAI National Conference on Artificial Intelligence*, pages 593–598, 2002.

K. Murphy. Bayesian map learning in dynamic environments. In *Neural Information Processing Systems*, pages 1015–1021, 1999.

Richard M. Murray, S. Shankar Sastry, and Li Zexiang. *A Mathematical Introduction to Robotic Manipulation*. CRC Press, Inc., Boca Raton, FL, USA, 1994. ISBN 0849379814.

P. Newman, J. Leonard, and R. Rikoski. Towards constant-time slam on an autonomous underwater vehicle using synthetic aperture sonar. In *Eleventh Intl. Symposium of Robotics Research*, pages 409–420, 2003.

C. Roman. *Self Consistent Bathymetric Mapping from Robotic Vehicles in the Deep Ocean*. PhD thesis, Massachusetts Institute of Technology & Woods Hole Oceanographic Institution, May 2005.

R. Smith, M. Self, and P. Cheeseman. Estimating uncertain spatial relationships in robotics. *Autonomous Robot Vehicles*, pages 167–193, 1990.

W. Stone, D. Wettergreen, G. Kantor, M. Stevens, E. Hui, E. Franke, and B. Hogan. DEPTHX (deep phreatic thermal explorer). In *Proc. of the 14th Intl. Symposium on Unmanned Untethered Submersible Technology*, 2005.

R. Urick. *Principles of Underwater Sound*. McGraw-Hill, New York, NY, 3rd edition, 1983.

David Wettergreen, Nathalie Cabrol, James Teza, Paul Tompkins, Christopher Urmson, Vandi Verma, Michael D Wagner, and William Red L. Whittaker. First experiments in the robotic investigation of life in the atacama desert of chile. In *International Conference on Robotics and Automation*. IEEE, April 2005a.

David Wettergreen, Paul Tompkins, Christopher Urmson, Michael D Wagner, and William Red L. Whittaker. Sun-synchronous robotic exploration: Technical description and field experimentation. *The International Journal of Robotics Research*, 24(1):3–30, January 2005b.

L. Whitcomb. Underwater robotics: Out of the research laboratory and into the field. In *IEEE 2000 International Conference on Robotics and Automation*, May 2000.

S. Williams. A terrain aided tracking algorithm for marine systems. In *Intl. Conf. on Field And Service Robotics*, pages 55–60, July 14-16 2003.

S. Williams and I. Mahon. Simultaneous localisation and mapping on the great barrier reef. In *Proc. of IEEE Intl. Conf. on Robotics and Automation*, volume 2, pages 1771–1776, April 26-May 1 2004.

S. Williams, P. Newman, G. Dissanayake, and H. Durrant-Whyte. Autonomous underwater simultaneous localisation and map building. In *Proc. of IEEE Intl. Conf. on Robotics and Automation*, pages 22–28, April 2000.

D. Yoerger, M. Jakuba, A. Bradley, and B. Bingham. Techniques for deep sea near-bottom survey using an autonomous underwater vehicle. *International Journal of Robotics Research*, 26(1): 41–54, January 2007.