USCR AUVSI TECHNICAL REPORT, July 2011

University of Southern California Competition Robotics (USCR)

SeaBee III

Written By: John O'Hollaren, Edward Kaszubski, Noah Olsman, Michael Benzimra

University of Southern California iLab
uscrs.org

ABSTRACT

The new SeaBee III is an autonomous underwater vehicle (AUV) designed and built by undergraduate and graduate engineering students from the University of Southern California. This year's vehicle incorporates a variety of improvements over last year's vehicle including a new software platform, upgraded electronics, and new mechanical components. The SeaBee III will enter its second competition in the 13th Annual International AUV Competition hosted by AUVSI and ONR.

1. INTRODUCTION

USCR's consistent and active participation in the AUVSI competitions reflects dedication to furthering robotics research at USC and around the world. The team hopes to improve upon its 6th place finish in 2009, 6th place finish in 2006 and 2007, and 15th place finish in 2008. The improved SeaBee III includes many new features that will help us achieve our goals, such as a single hull layout and a liquid cooling system. The electronics and software architecture have also been redesigned to accomplish more of the competition tasks.

2. MECHANICAL DESIGN

The mechanical design of SeaBee III stems from a desire to make the team's AUV more compact and lightweight. The single-hull design with an internal rack system was developed to enable easy removal and access to electronics. In addition, connections are established by using waterproof connectors from the end cap to external components such as the compass, SONAR, marker droppers, cameras, thrusters and the bump sensor.

2.1. Hull

The hull for Seabee III is constructed out of 3/16" thick T6061 anodized aluminum. The enclosure employs a cylindrical design measuring 7.5" in diameter and 13" in length. It shields the internal electrical systems from water damage with a watertight design. In the hull cap design, the mechanical team members covered the cap with waterproof Fischer Connectors allowing access to the electrical systems from components outside of the hull. The new SeaBee III can be seen in Figure 1.

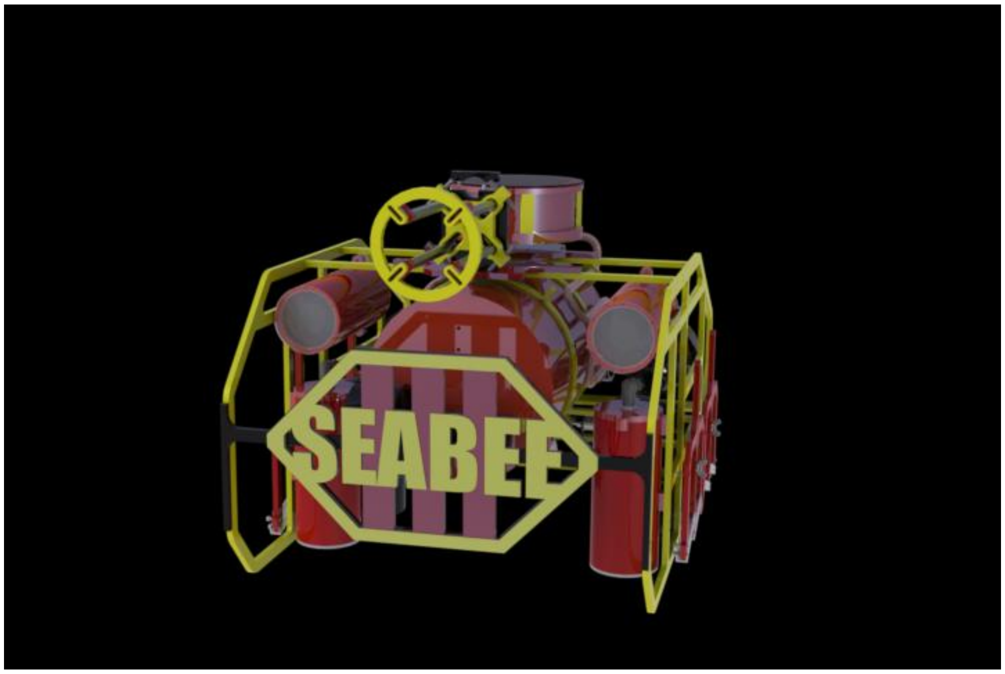

Figure 1. SeaBee III Assembly

2.2. External Frame

Using ¼" thick T6061 anodized aluminum, the mechanical team created a lightweight, hydrodynamic frame that could support the weight of the hull. The frame measures 24" x 12.5" x 16.5". There are two mount bars on top and four on the bottom for competition specific components. The side rails, can be used to mount our flotation system. We wanted to make the external frame flexible for future modular upgrades, so we placed auxiliary ports on the end cap.

2.3. Internal Frame

In order to mount all of our electronics inside of the hull, we utilized a custom designed internal frame attached to our end cap, which can slide in and out of the hull on Teflon rails. This frame is machined from aluminum and is designed around our electronics, ensuring a secure mounting platform for every device that is installed in the SeaBee III. We used CAD software to model all of the electronic components and ensure that they would both fit into the hull when it was completed. We aimed to maximize and efficiently use space within the hull. The rendering in Figure 3 below demonstrates the utility of this design.

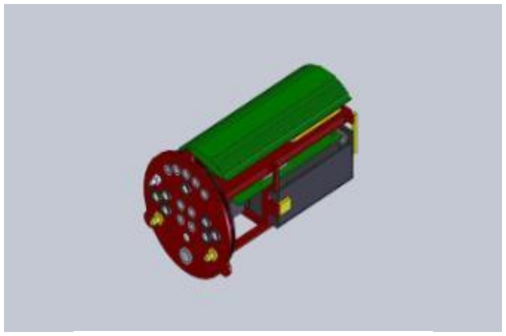
Figure 3. Internal Assembly

2.4. Cooling System
Because of the SeaBee III's dual core processor and a number of other highly thermally demanding elements, it is necessary to provide a cooling system to prevent damage to the electronics. To that end, we designed a unique water-cooling system consisting of two cooling blocks, an external radiator and a circulation pump. The primary cooling block is mounted to the standard XTX heat-spreader to cool our CPU while the second smaller cooling block pulls heat away from the motor-driver H-bridges. The radiator is designed to be both light and effective, consisting of a series of copper pipes attached securely to the end cap and extending just between the hull and the external frame.

2.5. Motors/Thrusters
The SeaBee III uses six (6) SeaBotix BTD150 thrusters – the same propulsion mechanisms used by our previous AUVs. These thrusters are arranged in three main groups, horizontal for forwards and backwards movement, vertical for depth, and two strafing to enable five degrees of control.

2.6. Marker Dropper
The marker dropper mechanism was constructed using linear actuators. This design features a compact frame and a robust releasing structure that is quick, accurate, and reliable. The dropper is connected to the SeaBee III internal electronics using a 4-pin connector on the end cap. When the control signals activate the relays, the two solenoids on the marker dropper release a marker which is held in place with two spring-loaded arms.

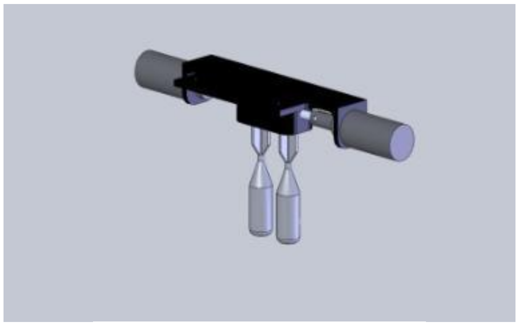
Figure 4. Marker Dropper

The markers are shaped like mini-torpedoes with angled fin tails to induce a spin during the descent. This shape ensures a highly hydrodynamic marker that will fall in a more streamlined way than that of a spherical marker.

2.7. Grabber

The grabber mechanism is the component of the SeaBee III that is used to pick up the counselor in the recovery phase. SeaBee III's grabber, shown below, is an attachment on the bottom made of aluminum. It is lined with several small acrylic teeth that push the PVC briefcase into one of the slots. With this revision of the grabber we have incorporated a retractable design. The grabber is able to slide within the constraints of the robot when it is not in the pool to maintain SeaBee III's compact design. When the SeaBee III sits on the counselor, the slots will open up allowing the PVC pipe to fall into place. The slot levers are controlled by small torsion springs that close the slots after the counselor is in place. The SeaBee III can then breach with the counselor secured by the grabber.

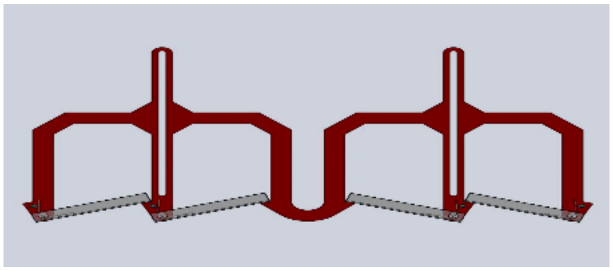
Figure 5. Grabber Mechanism

This design provides the SeaBee III with room for errors when picking up the PVC counselor due to its size and placement on the robot. No matter what the orientation of the AUV is, if one part of the counselor is on any of the 8 slots of the grabber, the counselor will be secured.

2.8. Manual Control Case (Flo-Case)

In past competitions, team members realized the impracticality of each person carrying different testing equipment to the practice side of the pool. Sometimes, cables are left in the booth, wasting practice run time. In addition, a team member would have to hold up umbrellas to block sunlight glare onto laptop screens.

To combat these difficulties, the team decided to create an all-inclusive testing platform for the SeaBee III. This manual control case, called the Flo-Case, is shown on Figure 6 below. The case allows for direct data monitoring and video feed from the submarine and contains a laptop, monitor, keyboard, router, USB ports, power supplies, gauges, and networking equipment. This device will provide convenient on-site testing and debugging of the software. This case will also serve as storage for spare parts.

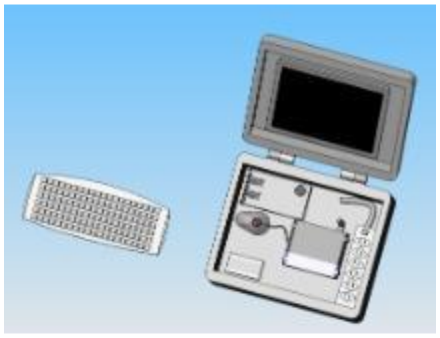
Figure 6. Manual Control Case

2.9. Shooter

The SeaBee III shooter incorporates a slingshot-powered design to fire a pool toy torpedo (Toypedo) through the window. For accuracy and space constraints this model only fires one toypedo when triggered. Due to the projectile's shape and buoyancy, the toypedo is capable of firing straight for 8ft. We chose the Toypedo itself as the projectile for a number of reasons. The Toypedo developed by Swim Ways has a hydrodynamic design combining a streamlined fin-stabilized shape with a rubber material. It measures 1" in diameter and 5" in length. The Toypedo is especially useful for us because it was designed to be neutrally buoyant, with a specific gravity between .95 and 1.05 relative to water. It also required no modifications for the competition and is cheap and dispensable.

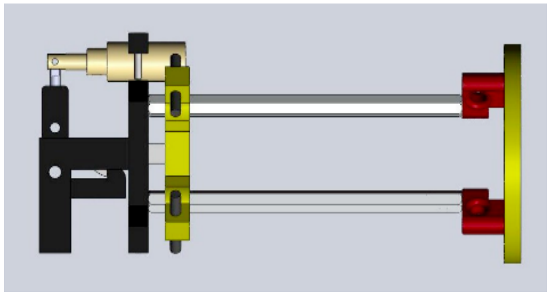
Figure 7. Shooter

3. ELECTRICAL DESIGN

The goal of the electrical team for SeaBee III was to design and integrate all electronics into a cohesive structure that interacts with software. To this end, electrical team members worked on different projects, each requiring the development of a printed circuit boards (PCB).

Eagle X11 software was used to create circuit schematics, and to place and route components on a board for fabrication. After ordering all the necessary integrated circuits and electronic parts from Digi-Key, we soldered the connections onto the PCBs fabricated by our sponsor, Advanced Circuits. The following sections below discuss each electrical subsystem of SeaBee III:

3.1. Computing

The computing and processing systems of the SeaBee III resemble the cerebral cortex of the brain. SeaBee III runs a custom processor with two XTX Computer on Modules (COMs) for all major path planning and intelligent decisions. The carrier board developed by our electrical team this year is shown in Figure 8. The two processing modules are networked together using an Ethernet switch. Together, the processor and network make up a small Beowulf Class I cluster to evenly share computational loads.

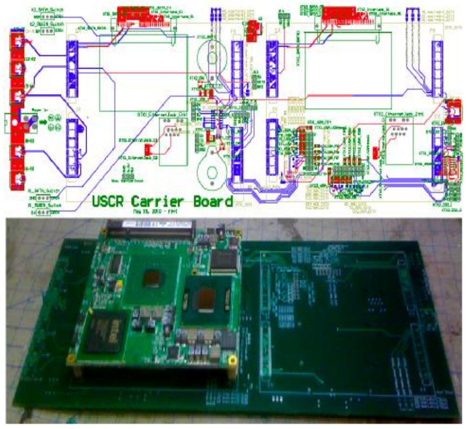

Figure 8. Custom Carrier Board

Each processing module is connected to a 250 GB SATA hard drive and interfaces to VGA, USB, and Ethernet. These large capacity drives supports large amounts of on-board computing power and allow us to use the submarine as our main development workstation. Each computer contains four (4) USB connectors, providing interface for the cameras and USART buses to establish communication with our I/O chip and SONAR system.

3.2. Battery Management

SeaBee III runs on 12 Lithium Polymer cells from MaxAmps that provide 10,000 mAh each. The electrical team members put together the 3.7-volt cells into two (2) battery packs of 6 cells each. The two battery packs supply 22.2 volts to power the submarine. As an interface with the lithium polymer batteries on the SeaBee III, the battery board will monitor battery life and charging cycles using an ATMEGA 406 microcontroller. A built-in Coulomb counting system in the microcontroller and extra differential Operational Amplifiers enable the team to identify the amount of battery power remaining. This information, visually outputted on the LEDs, is crucial in that it ensures consistent and reliable battery power during testing and the competition run.

3.3. Power Regulation and Management

The Power Board controls and interfaces all power, control, and sensor interfaces for the submarine other than the cameras, which are controlled by the Carrier board. This board ensures proper interfacing to two USCR custom-built lithium-polymer battery packs. In addition, the SeaBee III's power board manages power conditioning needed to convert voltage to acceptable levels for the computers, integrated circuit sensors, pump, and cameras.

Besides power regulation, nine motor drivers on the board control six thrusters, two marker droppers and an auxiliary motor for future expansion. These motor drivers are then controlled by an eight-core Parallax Propeller microcontroller. The Propeller not only controls the motor commands but also talks serially to numerous sensors including battery management, accelerometer, gyroscope, analog to digital converters, pressure sensors, and the compass.

3.4. Sensors
The SeaBee III has an impressive array of sensors to provide input to the main computers including internal and external pressure sensors, an HMC6343 digital compass with pitch and roll detection, a high speed accelerometer, three (3) internal temperature sensors, an active SONAR system, current monitoring for each of the five (5) motors and four (4) Internet Protocol (IP) cameras.
The team decided to use IP cameras to provide a high resolution and more responsive vision system. Two cameras are facing forward, while the other two face downward. The IP cameras are directly connected to the computing cluster via a USB network hub.

Our newest addition to SeaBee’s sensor array is the Xsens MTi Inertial Measurement Unit (IMU). This device is composed of multiple accelerometers and gyroscopes which allow it to measure acceleration, both angular and linear, and magnetic fields. These measurements allow for the capability to very accurately get an idea of the physical orientation of the robot in terms of relative spatial position and against absolute measurements like compass heading. Previously we had only used a compass which was sufficient for rough estimates but proved unreliable and lead to technical issues such as magnetic interference from thrusters. The IMU interfaces much more simply and gives more varied and accurate data. It is much less affected by noisy magnetic fields than our previous compass which makes localization a much easier and more reliable task.

The SeaBee III’s sensor array is monitored by a Propeller microcontroller, named the BeeSTEM. In addition to reading and forwarding sensor data to the main computer module via USB, the BeeSTEM is responsible for executing the SeaBee III’s main PID control loops and maintaining the AUV’s depth, heading, and speed. This offloading of our PID loops to the BeeSTEM ensures that our main computer is free for more complex tasks such as vision processing and mission planning. The BeeSTEM responds to commands given by the main computer module, manipulating the SeaBee III’s movement as our Mission Planning module sees fit.

3.6 Actuation Control
Our team utilizes a PID (partial integral derivative) controller to determine the thrust needed to bring the system from it's actual position to a desired position. The desired position is sent from the COM modules and the vision algorithms running within. After the system receives a new command, a separate control loop reads the sensors to find the current position and does it's best to achieve the ideal orientation. This approach takes a significant amount of tuning but has proven highly reliable in the past.

3.7 Passive SONAR Array

After various attempts at developing sonar capabilities we have decided on an passive, hydrophone array sonar system. Rather than trying to passively filter and process sonar as we did in the past, we decided to do all of our signal processing using the Cheetah SPI Host Adapter. This device has a sampling rate over 40MHz and easily interfaces with our XTX computers to process incoming digitized signals. Now rather than trying to do stripped-down geometric phase angle calculations, we implement a Fast Fourier Transform (FFT) to determine the phase difference of two incoming hydrophone signals using three Reson TC4013 Hydrophones. The new system has the added benefit of requiring a much smaller PCB since the only components we need are input pins for the hydrophones, Analog-to-Digital converters and a low-noise Variable Gain Amplifiers (VGAs) for the pre-processing of hydrophone signals, and output pins from the Cheetah. An additional input pin is used by an external Arduino microcontroller to tune the gain of the VGA.

This new sonar layout is very simulation friendly since the only real device that needs to be approximated is the input signal. In the past we had to try to simulate a large amount of circuitry which did not always behave as we expected. This ease of testing has allowed us to develop more effective signal processing algorithms this year.

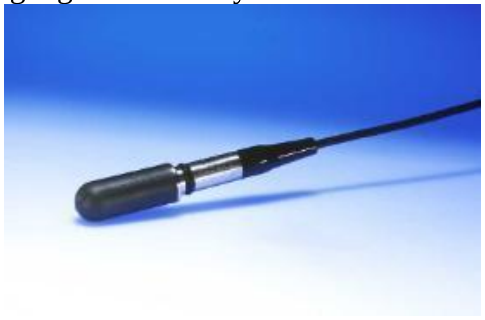

Figure 9. Reson TC4013 Hydrophone

3.10 Active SONAR Module

A new addition to SeaBee III is the Tritech Micron Digital Sonar Technology (DST) Scanning Sonar module. The sonar device rotates a transducer into a series of pinging positions, emitting a narrow fan-shaped acoustic beam at each position and waits for the reflection from obstacles in the pulses path. The process is repeated until the device has produced a complete, 360 degree scan. The Micron, however, exhibits an enhanced detection resolution over conventional sonar by sweeping the frequency of its signals from 300 to 350 MHz during the pulse duration rather than emitting an acoustic pulse of a singular frequency. Mounted on the bottom of Seabee, the Micron will provide our localization and mapping algorithms real-time updates on the submarines physical environment.

Figure 10. Tritech Micron DST Scanning SONAR

3.9. Killswitch

The new SeaBee III uses the same magnetic Killswitch that our previous submarine used - a magnetic sensor and a cylindrical enclosure on top of the submarine to start or kill the SeaBee III.. However, we have made mechanical improvements to ensure it works even more reliably. The Killswitch, illustrated in Figure 10 below, looks like a handle and can be rotated to turn the submarine on or off using static magnets. In addition, the on or off status will be displayed to the outside world using green and red LEDs and an LCD screen for custom output determined by the main computers.

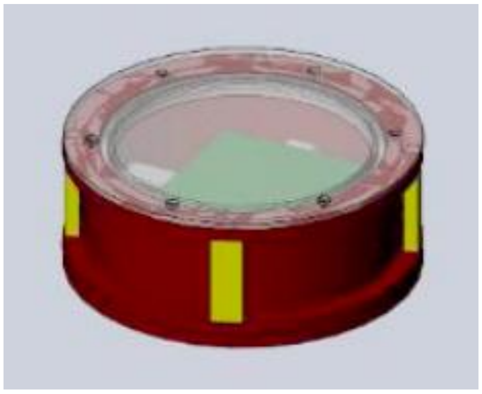

Figure 11. Magnetic Killswitch

4. SOFTWARE DESIGN

4.1 Architecture

4.1.1 Nodes

When working with large and/or complicated software architectures such as those found on autonomous robots, it is often beneficial to divide the system into smaller functional units, or nodes. In particular, a node-based architecture allows for the creation of a dynamic pipeline from raw data to high-level features while both exposing the transformations taking place at each step in the pipeline and allowing for the pipeline to be distributed across multiple computational units (as required by the hardware platorm used). In designing Seabee's software archicecture, we sought to gain the above benefits and, therefore, chose to follow a node-based design pattern.

4.1.2 Image Pipeline

While most of the nodes in our architecture process relatively small amounts of data and, therefore, require low bandwidth, some components, most notably those in the low-level vision pipeline, require high-bandwidth connections and extensive optimization to remain feasible. This low-level vision pipeline is effectively an interface between high-bandwidth, low-level data (in the form of images) and the low-bandwidth, high-level features extracted from those data. Thus, we sought an implementation of inter-node data transport specialized for images for use in our pipeline.

4.1.3 ROS

We found the solution to our low-level architecture requirements in ROS, a popular open-source toolkit developed by Willow Garage. ROS provides the core libraries necessary to create a software architecture as set of separate processes (nodes) linked by single-directional and multi-directional communications in the form of generic, customizable messages and services, respectively. The contents of these messages, which exist in namespaces called topics, are serialized into TCP streams and sent between nodes, which can reside almost anywhere. ROS also provides excellent debugging tools, allowing for communications between nodes to be monitored, logged, simulated, and visualized. Furthermore, ROS provides the means for algorithms to be implemented as nodelets, which are compiled as shared objects and can be dynamically loaded at runtime into one multi-threaded nodelet manager process. Most notably, nodelets allow for fast intra-process data transport via shared pointers and provide an ideal scenario for sending images and other otherwise high-bandwidth data data between nodes with minimal overhead.

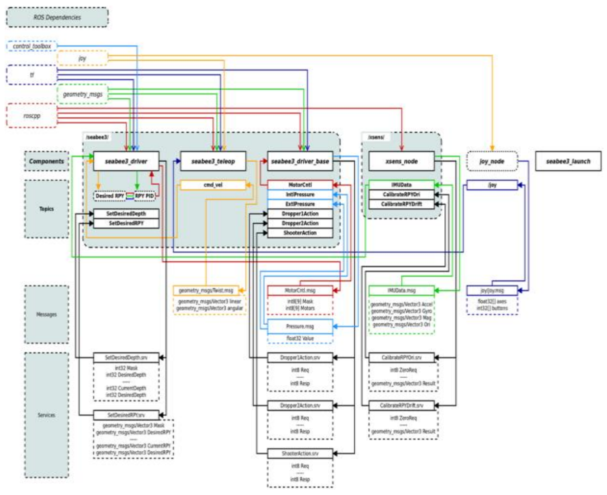

Figure 12. ROS Software Hierarchy

4.2 Implementation

4.2.1 Tools

While ROS provided the core functionality we needed for our software architecture, we found it helpful to utilize additional tools for including those for automation, testing, and code re-use. In some testing and debugging scenarios, it is desirable to distribute the computational load of the software components across both the robot and some other client machine. To simplify this process, and with the help of the roslaunch utility, a server and client environment variable can be set; the launch file for a given scenario can specify which nodes should be started on the client and which on the server. Then, by simply changing an environment variable, the same launch file can be used to start up a single-machine scenario (client and server variables point to the same machine) or a multi-machine scenario (environment variables point to difierent machines). We found gtest to be an acceptable testing framework; its existing integration into the ROS build system also made it an appealing choice. We found that we were frequently copying ROS-related skeleton code from existing files into newly created nodes to save time. As such, this commonly-used skeleton code was factored out and compiled into a comprehensive library of ROS-related, generic base classes. Among these are BaseNode, for core ROS functionality including the optional initialization of a dynamic_reconfigure server (allows for intuitive parameter tuning during runtime through a generic GUI), BaseImageProc for nodes utilizing an image pipeline, and BaseNodelet, which wraps the existing Nodelet class and provides a more useful yet still generic interface including interruptable threading. Other skeleton code didn't make sense in the form of base classes; instead, for maximum re-usability and post-compilation performance, these resources were optimized and made generic using templates and meta-programming and then moved into a common_utils package for easy inclusion in other packages. These utility classes include useful operators, tools for working with the ROS tf library, and helper functions for use with OpenCV.

4.2.2 Low-Level Vision Pipeline

As mentioned earlier, the purpose of the low-level vision pipeline is to extract higher-level features from high-bandwidth, low-level image data. As such, the algorithms in this pipeline deal with color filtering and segmentation, shape recognition, camera intrinsics, and rough object position estimation. We accomplish color segmentation through a multi-step process. First, we convert the original image color space to HSV and define a 3-dimensional color vector $C_i = \{h, s, v\}$ for each desired output color i. Next, we define a 3-dimensional Gaussian $G(\mu, \sigma, w)$ with mean $\mu = C_i$ and variance $\sigma = \{\sigma_h , \sigma_s , \sigma_v \}$. Then, for each pixel $Q_{x,y} = \{h_{x,y} , s_{x,y} , v_{x,y} \}$ at row y and column x in the HSV color space of the original image, we define that pixel's color vector $C = Q_{x,y}$. Finally, we compute the probability that $C = C_i$ by evaluating $G(\mu, \sigma, w)$ with $w = C$. This results in a probability image P_i for each desired output color i. Classification of a given pixel $Q_{x,y}$ is then acheived by iterating through the set probability images and finding $P_{i,x,y}$, the largest $P_{i,x,y}$ (the pixel at the location (x, y) with the largest value accross all probability images); then i is the output color classification for $Q_{x,y}$. Optionally, if P is below some threshold, $Q_{x,y}$ can be classified as unknown. The classifier can then, given a an image S_i and a mask M_i for each output color i, be trained by simply calculating the mean and variance μ_i , σ_i , respectively, of the set of unmasked pixels in S_i .

Contour segmentation and matching is accomplished by first running cvFindContours on an input image i and each element t of a list of template images, then finding the shortest distance between any two contours (i, t) using cvMatchContours(i, t). This contour matching is used to determine how similar two arbitrary shapes are and can be run on the grayscale transformation of a raw input image (for color invariance) or on a pre-filtered image such as a probability image calculated by the color classifier (if a shape with a particular color, say a yellow buoy, needs to be found). When attempting to recognize shapes that may appear distorted from different perspectives, that is, shapes without radial symmetry, it is necessary to undistort the shapes prior to further processing. This is especially true for competition the objects known as the bins, since the image inside the bin wil also be distorted. We accomplish this undistortion by finding quadrilaterals, assuming those shapes are bins, using a perspective transform from OpenCV, and then attempting to find bins in those undistorted images.

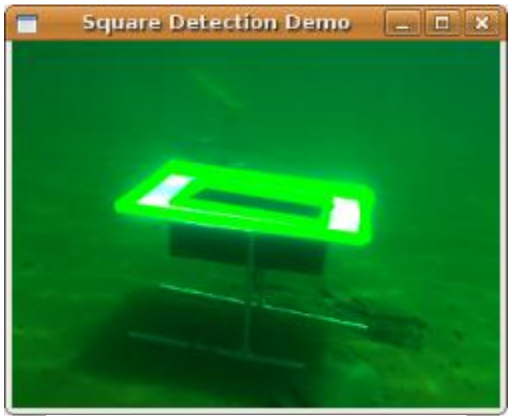

Figure 14. Actual Camera Image

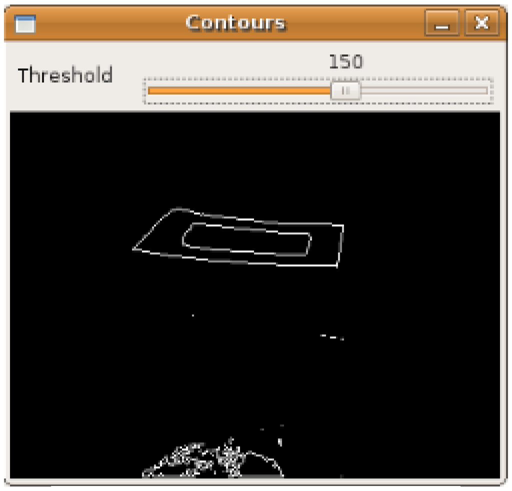

Figure 15. Contour Recognition

When a camera is submerged under water, it's field of view is narrowed and everything appears magnified. Since many of our object detection algorithms rely on accurate camera intrinsics, it is necessary to know what these parameters will be when the camera is submerged. However, we have found calibrating camera intrinsics under water to be both dificult and undesirable; instead, we calculate these values above water (or more specifically, in any medium with a known index of refraction) and can then extrapolate from that calibration to calculate the camera intrinsics in any other medium since the distortion matrix for the camera remains constant (only the focal length changes). This both speeds up the calibration process, since all calibration can be performed above water, and allows for automatic switching between camera intrinsic parameters depending on whether or not the sub is submerged (which we detect using a pressure sensor).

4.2.3 High-Level Vision Pipeline

The high-level vision pipeline is designed primarily to filter through the features extracted by the low-level vision pipeline and identify high-level landmarks, or competition objects, including the gate, buoys, pipes, hedges, windows, and bins. It is also responsible for 2D to 3D reprojection of these landmarks for localization. The node responsible for landmark detection is the landmark finder. This node accepts high-level queries, consults the appropriate nodes from the low-level vision pipeline and returns all known information about any matching landmarks.

After a landmark has been positively identified, it needs to be projected into 3D space. However, Seabee uses only monocular cameras; to perform the reprojection, we must first calibrate certain parameters for each landmark object. This is done by measuring the observed pixel width of each landmark at a known distance from the camera along with the physical width (in meters) of the object. We can then extrapolate from those measurements the distance from the camera to the object given any other observed pixel width. Furthermore, using our camera intrinsics extrapolation method, we can perform this calibration above water and remain confident that, once submerged, the calculated distance to the object will be identical.

4.2.4 Localization and Controls

Seabee does not have a DVL, or doppler velocity logger, so we have had to find other means of localization. Our approach utilizes a physics model of the sub for rough velocity estimation, sparse 2D SLAM on landmark data, dense 2D SLAM on scanning sonar data, and a 6-degree-of-freedom PID loop for maintaining both position and orientation. Calculating the state of motion of a UAV without a DVL is non-trivial; since the relative velocity of the vehicle can only be occasionally measured by analyzing the observed position of landmarks or by taking sonar scans, we created a physics model of the sub that allows for rough, short-term estimation of the full 6-degree-of-freedom pose and velocity of the vehicle given the actual thruster commands being sent to the sub's thrusters. However, some axes can be measured absolutely. Specifically, an onboard IMU can measure absolute roll and pitch, and a pressure sensor can measure absolute depth; the remaining axes must be simulated with only hints from the IMU's accelerometers and gyros.

Given the estimated pose and velocity from the physics model, along with sparse landmark data and dense sonar data, it should be possible to accurately estimate the sub's current position using a particle filter against a pre-made map, or even generate the map in real-time. The final product of our localization pipeline is a 6-degree-of-freedom pose and velocity of the robot. In order to actually navigate, Seabee uses a PID controller for each available control axis. This controller simply tries to minimize the error between some desired position and the current estimated position. The vehicle is always trying to maintain this desired position; whenever we want to navigate to some position, we calculate a safe path in the form of several waypoints and then set the desired position of the sub to each of these waypoints. Similarly, we can manually drive the sub around by using a joystick to modify this virtual target pose. If we want to maintain some velocity, we must simply move the target pose at that velocity and the vehicle will match that velocity in the most optimal way possible.

4.2.5 Drivers

Seabee uses three primary devices which required custom drivers to function in Linux and/or in ROS; these devices include Seabee's power board (designed and assembled in-house), an XSens MTi high-precision solid-state MEMS IMU, and a Tritech Micron DST scanning sonar. Seabee's power board uses a simple serial connection to accept and set thruster values. This driver was written several years ago and was tightly integrated with the USC iLab's Saliency Neuromorphic Vision Toolkit; it needed to be ported partially re-written. The XSens MTi came with full linux source code and example code; we simply had to port the code to ROS. The Tritech Micron DST came with no source code and software only compatible with Windows; hence, it was necessary to inspect the data sheets for the device and implement a new linux-compatible driver from the ground up.

5. FUTURE WORK

The last year has been predominantly dedicated to completely overhauling the software architecture to be compatible with ROS as mentioned above. While there was a great deal of overhead involved in this process, we are now developing software at an incredible rate and plan to have by far the most sophisticated architecture we have ever had for SeaBee. To support these developments we are developing a new carrier board to handle the additionally computational oiwer which we will undoubtedly need.

As a mark of the team's expansion and progress, USCR has begun development of an Unmanned Aerial Vehicle (UAV). This helicopter-like apparatus will include a plethora of inertia-sensing equipment along with GPS, range-finders and vision-based navigation systems.

Most importantly, the key to the organizations future success lies in the team members. As a result, USCR has established training programs for its new members. These training sessions are held during the first month of the semester and focus on developing fundamental skills frequently used for electrical, software and mechanical projects.

6. ACKNOWLEDGMENTS

Generous support from USC, iLab, industry, and academia allow USCR to continue being an integral part of student research and involvement in the Viterbi School of Engineering.

As a student run organization, this endeavor was made possible by: Fischer, Digi-Key, Advanced Circuits, Max Amps, ORE Offshore and the USC Viterbi School of Engieering.

Special thanks goes to both the USC Viterbi School of Engineering and the USC College of Letters Arts and Sciences Machine Shop, without whom we would not have our SeaBee AUVs at all. The Viterbi School of Engineering provided us with research funding. Don, our machinist, and his team in the shop have proven themselves to be one of our most valuable resources.