AUVSI TECHNICAL REPORT, July 2015
University of Southern California Autonomous Underwater Vehicle Design Team

SeaBee IV

Written By: Michael Kukar, Jessica Freidin, and Ben Shiroma

Seabee IV is an autonomous underwater vehicle (AUV) developed by a team of students at the University of Southern California for the International Robosub Competition.

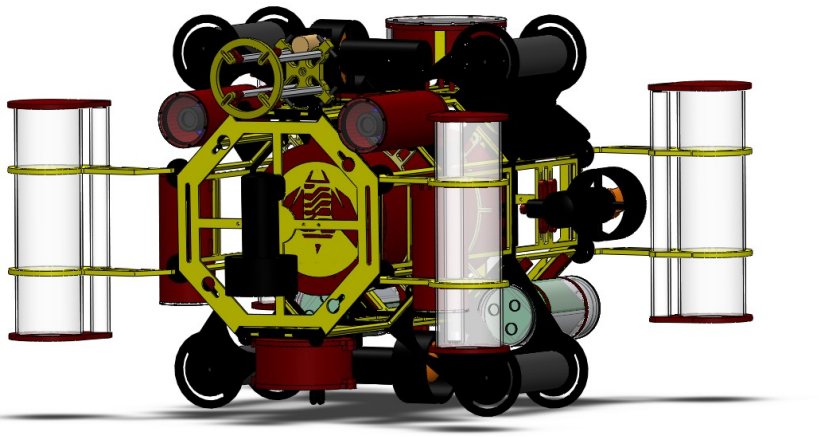
1.0 INTRODUCTION

The University of Southern California's Autonomous Underwater Vehicle Team (USC AUV) is dedicated to participating in and furthering robotics research not only at USC, but also around the world. USC AUV is a consistent participant in the Robosub Competition and strives to continually improve through exchange of ideas and involvement in the AUV community. USC AUV uses a full-year design cycle between iterations of the SeaBee AUV, allowing time for thorough testing and integration of new designs. From preliminary internal design reviews to a critical review attended by experts in the field, designs and changes to SeaBee undergo a thorough analysis culminating in vital improvements to the AUV without wasteful or unnecessary expense. This year USC AUV has undergone a complete rebuild of Seabee's internals and pod design, keeping only the frame and original hull from last year.

2.0 MECHANICAL

2.1 MECHANICAL OVERVIEW

The mechanical design philosophy for Seabee is intended to create a modular, compact, and lightweight AUV. The single-hull design with an internal rack system enables easy removal and centralized access to electronics while the external frame provides support for long term development by allowing individual component redesigns with little to no effect on the overall structure of the AUV. Electrical connections are established by using waterproof connectors, supplied from Fischer Connectors, from the hull end cap to external components such as the IMU, SONAR, marker dropper, shooter, cameras, and thrusters allowing simplified electrical reconfiguration. Similar wet-pluggable connectors are also in place to allow devices to be plugged into SeaBee without worrying about water creating a short, saving time during wet tests and competition runs.



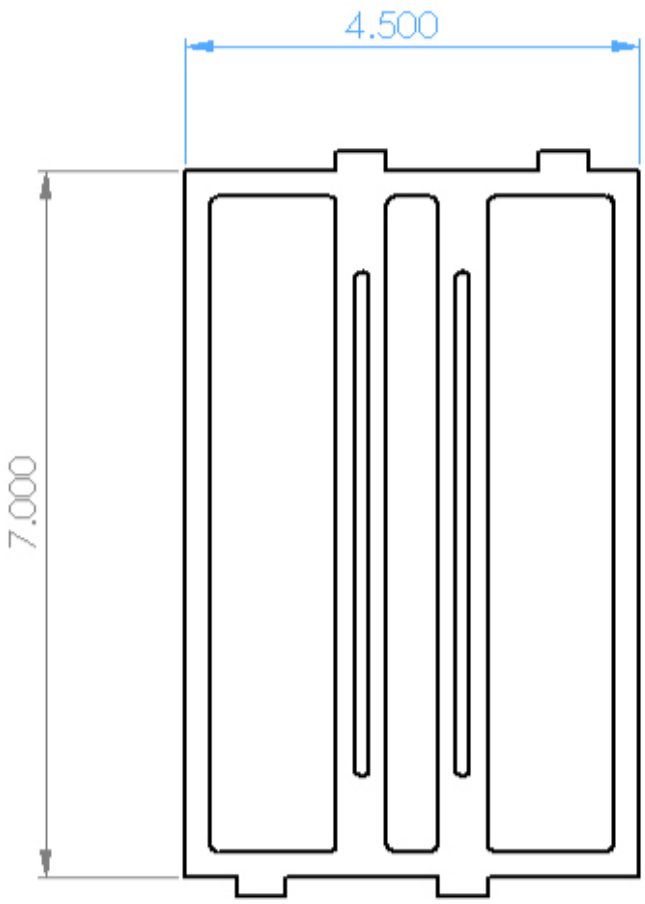
2.2 HULL

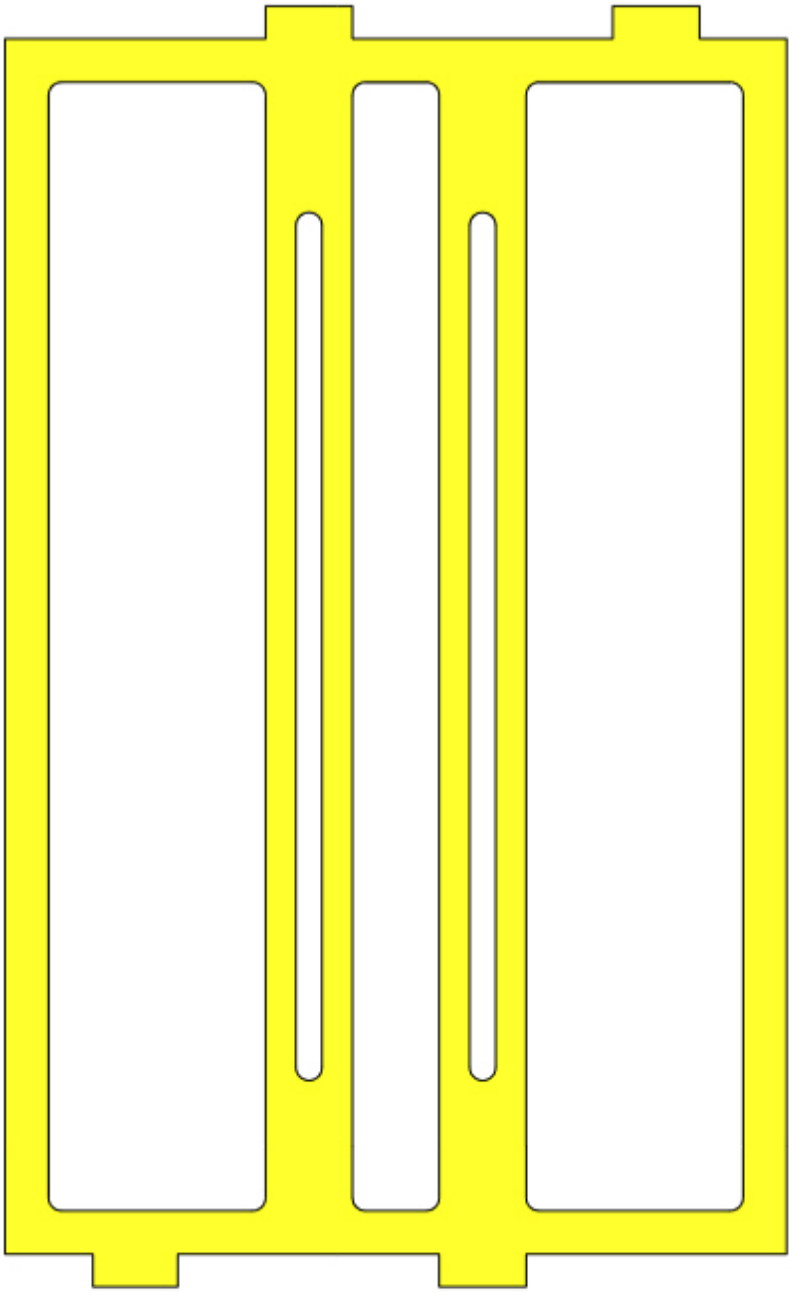
The hull for SeaBee is constructed out of 3/16" thick T6061 anodized aluminum. The enclosure employs a cylindrical design measuring 7.5" in diameter and 13" in length. It shields the internal electrical systems from water damage with a watertight design. In the hull cap design, the cap is covered with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

2.3 EXTERNAL FRAME

The new external frame is part of the design for the next iteration of the SeaBee AUV and consists of an octagonal cage surrounding the main hull. Each Octagon measures 10.836" around an inscribed circle. Four Octagons create three sections of the frame. Panels constructed of T6061 anodized aluminum measuring 7" x 4.5" x 3/16" make up the walls of the octagonal cage resulting in a total of twenty-four panels. Each panel is associated with a certain location of the frame and a component. In addition, panels can move around the frame easily if design changes are necessary. The front and back surfaces of the octagonal cage serve as mounts for the depth thrusters and the flotation structure.

These panels are designed around a common panel template in SolidWorks and can be quickly machined through a laser-cutting process and then anodized. The panel redesigns dramatically reduce the design and manufacturing time of any mechanical changes to the SeaBee AUV or it components and are compatible with either the Seabee or upcoming SeaBee IV designs.


4.500
7.000



The flotation is also built into the external frame. Due to the higher density of aluminum, flotation is a necessary addition to the sub. Four 3" diameter acrylic air canisters are placed at the corners of the AUV. Mounted alongside them are smaller acrylic tubes that can be loaded with weights. The

weight tubes can be manually adjusted to achieve neutral buoyancy. With the mounted flotation system, SeaBee occupies a space of 28" x 26" x 20".

2.4 INTERNAL FRAME

SeaBee utilizes a custom designed internal frame attached to the main hull end cap, which can slide in and out of the hull on Teflon rails. This frame is machined from aluminum and is designed around the central electronics, including the carrier board, power board, and batteries ensuring a secure mounting platform for every device that is installed in SeaBee. 3D CAD drawings are used to guarantee that the electrical boards are compatible with the Internal Frame. The frame maximizes and efficiently uses space within the hull and provides a cooling infrastructure for the main computer.

2.5 LIQUID COOLING

SeaBee's quad core processor and other highly thermally demanding elements necessitate a cooling system to prevent damage to the electronics. SeaBee takes advantage of a unique liquid-cooling system consisting of custom aluminum cooling block, an external radiator and a circulation pump. The cooling block is mounted to the standard XTX heat-spreader to cool the CPU while simultaneously pulling heat away from the motor-driver H-bridges. The radiator is an off the shelf 120mm radiator used in PC liquid cooling applications. Using the lower temperature of the water as a cooling source, SeaBee is able to passively cool the radiator in the surrounding environment.

2.6 MOTORS/THRUSTERS

SeaBee uses six BlueRobotics T100 thrusters arranged in three main groups: horizontal for forwards and backwards movement, vertical for depth, and strafing to enable five degrees of control.

2.7 MARKER DROPPER

The marker dropper mechanism is 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 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. 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 that is used to pick up the object in the recovery phase. SeaBee's grabber is an attachment on the bottom made of aluminum (See Figure). It is lined with several small acrylic teeth that push the PVC briefcase into one of the slots. This revision of the grabber incorporates 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's compact design. When the SeaBee AUV is positioned above the object, the slots open allowing the PVC pipe to fall into place. The slot levers are controlled by small torsion springs that close the slots after the object is acquired.

2.8 BATTERY PODS

Two aluminum battery enclosures are mounted to the underside of the vehicle's frame. This frees up space for the electrical systems inside the primary hull. Battery pods can be hot swapped, allowing for extended runtime. The battery enclosures feature a rectangular design in order to compactly fit the rectangular batteries. They are sealed via an O-ring in a face-seal configuration. They are also fitted with pressure relief valves as a safety measure to prevent the buildup of hazardous pressure levels.

3. ELECTRICAL

OVERVIEW

The Seabee electrical system provides the robot and its systems with power and provides the appropriate interface between all of the electrical devices. The main electrical system is made up of two backplanes, and three daughter cards each featuring its own Atmel 1280 microprocessor. This eliminates the need for excessive wires inside the submersible hull, giving the final product a cleaner look, and increasing the overall reliability by alleviating the risk of loose wires and poor connections

3.1 POWER DISTRIBUTION

The Seabee Power Board is developed as a versatile platform on which to develop a wide array of electrical systems. In addition to supporting the vehicle's power management and regulation needs, the Power Board serves as an interface for any sensors not capable of directly interfacing with the XTX Carrier Board.

At the heart of the power board is the Parallax Propeller P8X32A microcontroller. The P8X32A has eight 32-bit processors, making it well-suited to the robot's high-precision, low- latency requirements. Nicknamed BeeSTEM, the microcontroller is responsible for maintaining communication with these sensors. As the BeeSTEM receives data from the sensor suite, it updates its control loops and forwards the data to the XTX Carrier Board. The BeeSTEM also produces the appropriate commands to initialize the sensors and place them in desired operating modes.

The Power Board produces regulated power at the common voltages of +3.3V, +5V, and +12V. To ease the process of swapping or adding sensors, the Power Board has auxiliary connectors. The Power Board incorporates nine motor drivers, which are controlled by BeeSTEM. Six of these provide power to the thrusters, and the other three support additional actuators, such as the Marker Dropper or Torpedo Launcher.

3.2 COMPUTING

Computer design for the Seabee is governed by the vehicle's need to perform complex computer vision and machine learning algorithms in real time in spite of restrictive space requirements. An XTX computer-on-module (COM) was selected for its balance between size and performance. The Seabee XTX Carrier Board was designed to break out important signals such as power, USB, VGA, and Ethernet, SATA, and USART.

3.3 SENSORS

The Seabee sensor suite was designed to provide as close to a 6-DOF state-space solution as is possible within team budget. The Seabee is capable of actively monitoring the status of its own

systems through the use of internal temperature sensors, internal pressure transducers, current monitoring in each motor driver channel (nine total), and coulomb counting in each battery pod. Four high-performance Internet-Protocol (IP) cameras equipped with wide-angle lenses are used to complete specific vision-related tasks and to provide an additional position estimate via visual odometry.

For the aforementioned 6-DOF localization the Seabee incorporates a Xsens MTi Attitude and Hearing Reference System (AHRS), a Honeywell HMC6343 3-axis compass, and a pressure transducer used to calculate depth. The primary advantage of using an AHRS like the MTi over a traditional inertial measurement unit (IMU) comes in the form of the unit's on-board signal processor, which allows it to be calibrated for the electromagnetic fields present on the Seabee and thus achieve virtually no drift.

Some of the sensors on the Seabee such as the IP cameras, are capable of directly interfacing with the XTI carrier board via USB. Most of the sensors on the robot, however, utilize serial protocols such as RS232, I^2C, and SPI. The BeeSTEM is responsible for meeting these bus requirements.

3.4 BATTERIES

The Seabee battery system consists of two custom +22.2V lithium polymer battery packs in parallel for a total of 20,000 mAh. Each pack contains a Seabee Battery Board based on the ATMEGA 406 microcontroller. In addition to regulating the LiPo cells to ensure even discharge, the Battery Boards actively monitor state of charge (SOC) through use of current integration, or "coulomb counting". Each Battery Board incorporates an LED display to provide visual feedback to the operator.

3.5 KILLSWITCH

The killswitch was designed with reliability as the primary focus. The design is robust and minimal, functional in the most demanding situations. A single reed switch is mounted inside the primary hull, actuated by a magnet tethered to the outer hull. A tiny logic-gate- based circuit ascertains the state of the reed switch and produces a 5V TTY "kill" signal to the microcontroller on the power board. The microcontroller is then responsible for placing the motor and actuator drivers into a low-power state.
One can place a high degree of confidence in the killswitch as its implementation, which takes place at the lowest level possible short of physically disconnecting the batteries, ensures that it will function properly even if other systems on the sub malfunction. Additionally, the robot's software monitors the state of the killswitch via the power board microcontroller. This means that if the robot enters the "kill state", processing can be halted until the state is exited and then proceed as normal, a helpful tool during development.

3.6 PASSIVE SONAR ARRAY

A passive sonar system is essential to completion of the recovery task, and can provide an estimate of relative location to be incorporated into the SLAM implementation. F Four Reson TC4013 hydrophones are placed in a tetrahedral configuration in a pressure case attached to the primary mechanical frame. The hydrophones are spaced to allow for non- ambiguous determination of phase difference between incoming waves in the 20-30 kHz range. Using a plane-wave approximation, a high-confidence estimation of the relative

location of the pinger in three-dimensional space is obtained.
or the sake of minimizing complexity and integration time, a largely DSP-oriented approach was chosen. The Electrical Team's role in the sonar system, then, is to ensure that the hydrophone signal makes its way to the computer with minimal loss. Due to size limitations, placing a full ADC solution in the same pressure case as the hydrophones is not possible. Instead, a single-channel preamplifier stage is connected to the immediate output of each hydrophone, inside the pressure case, and the ADCs is placed inside the main hull. Because of the small distance between the hydrophones and preamplifiers, very little noise is amplified. Since any noise picked up between the preamplifiers and the ADCs will have a significantly lower amplitude than the hydrophone signal, filtering is trivial. The ADCs interface with the on-board computer using Cheetah SPI Host Adapters. Each ADC- Cheetah pair (one per hydrophone) can achieve a sampling rate of 40 MHz. As such, the data that reaches the software portion of the sonar solution is as close to the "pure" signal as possible.

3.7 ACTUATION

The actuation daughtercard features twelve H-bridge motor controllers with PWM. Six controllers directly drive the robot's thrusters, and are able to run them forwards and backwards, as well as to throttle them effectively via a PWM signal. The other six controllers can be used for manipulators on the robot, including the dropper and torpedo firing system. The current draw of each motor controller is monitored by a current sense resistor, providing the computer with vital information as to the power being drawn by any motor. The computer is also able to shut off any or all of the motors in the event of a malfunction. The vehicle employs six SeaBotix BTD-150 thrusters. They are positioned to allow for control over all six degrees of freedom.

4 SOFTWARE

4.1 UBUNTU

Seabee3 uses Ubuntu Linux 10.10 "Maverick Meerkat" as its operating system. This selection was made based on the open-source nature of Ubuntu, its portability, and its good support for our intended software architecture.

4.2 ROS

Seabee has used ROS, an open-source toolkit developed by Willow Garage, since the 2010 RoboSub competition. ROS, or the "Robot Operating System", provides a language-generic, modular paradigm for the development of software systems.

System components are discretised into units called "nodes", each of which has at least one dedicated thread and the ability to control parts of its life cycle.

When necessary, these nodes are able to communicate via explicitly defined, language- generic messages sent over a named, simplex channel, or "topic". The direction of these topics is determined at compile time; a node can "advertise" an outgoing topic via a "publisher" object or "subscribe" to an incoming topic via a "subscriber" object. However, topics can be redirected or "remapped" at runtime, allowing for the creation of more loosely-defined distributed systems.

Arbitrary levels of complexity can be achieved through the creation of multiple publishers and subscribers within a node. More complex paradigms, such as “actions” or “preemptable tasks”, have been implemented to take advantage of this fact.

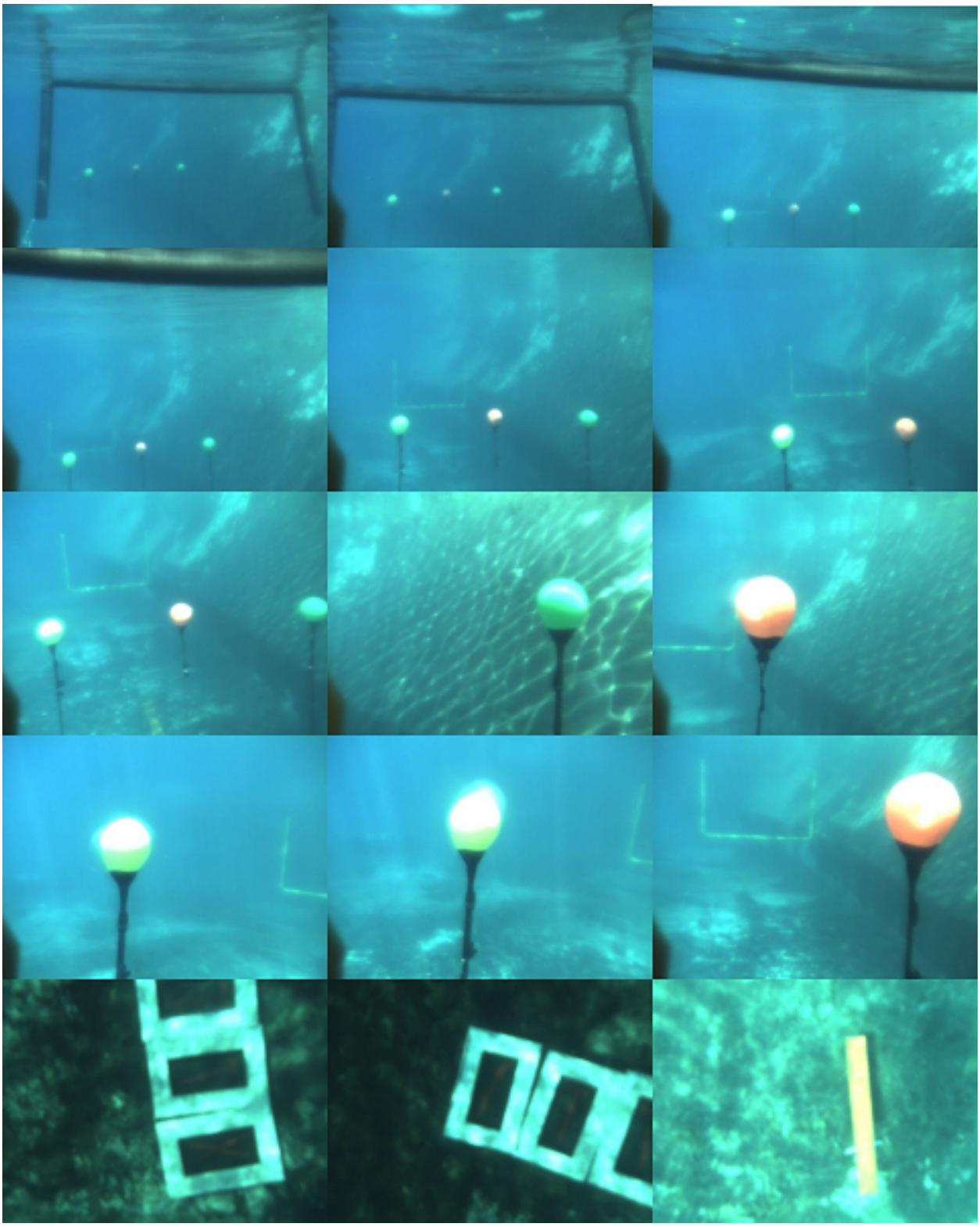
At a low level, inter-node communication is accomplished by message serialization within publisher objects, transmission of serialized data, and message deserialization within subscriber objects. However, if two nodes can be run on the same machine, and furthermore within the same process, this communication can instead be performed without serialization via shared pointers, thus allowing for high-performance, low-overhead communication.[ROS]

Perhaps more importantly, ROS provides fairly generic implementations of many common algorithms known to the field of robotics, including such categories as sensing, navigation, planning, and visualization.

4.3 QUICKDEV

While ROS provides a solid foundation on which to build, working within the toolkit often involves creating unnecessary redundancies across implementations. Furthermore, the serialization-free communication described above is traditionally time consuming to set up, as modules utilizing it must follow the “nodelet” paradigm rather than the more common “node” paradigm. We solve this issue by maintaining a set of scripts and generic wrappers around the more commonly-used ROS components in an open-source package called “quickdev” available in USC's “usc-ros-pkg”.

4.4 VISION PIPELINE



Seabee's view of the competition pool

In its current state, Seabee views the world through two PointGrey Firefly USB cameras: one facing forward and one facing downward. Both cameras are configured to stream bayered 320x240 images

at 30 hz. While the hardware interface to these cameras is USB, they support the IIDC 1394-based Digital Camera Specification over USB, allowing for the use of "firewire" camera drivers. Conveniently, ROS provides an acceptable firewire camera driver node, which we use to read images from our cameras. Figure 1 shows an example of some of the competition objects as seen by SeaBee's cameras.

Images streamed from our cameras are bayered and distorted by various optical effects, including those from the camera lenses and the results of different physical mediums surrounding the sensor tubes containing the cameras. To compensate for these effects, we use another useful community-developed ROS node called "image_proc", which utilizes several common OpenCV functions to de-bayer and un-distort (given a camera calibration) the incoming raw images.

Following basic low-level image filtering, we perform a color-space conversion from BGR to HSL. In order to optimize our the vision algorithms farther down the pipeline, we seek to mimic neural adaptation and avoid unnecessarily processing pixels that are not changing by a sufficient amount. This value is determined by calculating a weighted sum of the differences between each pixel in each channel in the current image and the value of that pixel at the point in time that it was last determined to have changed. Pixels found to have changed by a minimum amount are recorded in a binary mask, which is published along with the

corresponding HSL image. We call this mask-image pair an "adaptation image." Figure 2 shows an adaptation image fed to the color classifier, reducing the number of pixels fed through the classification algorithm by over 90%. The bands seen in the middle image (probability image) are the result of pixels that have not yet changed enough to necessitate re-classification.

Newly-generated adaptation images are fed through a color classification node which utilizes a native Bayes classifier trained on actual camera footage. The classifier identifies newly- changing pixels, calculates the likelihood that a given pixel should be classified under each of a set of colors, and then publishes these probabilities as an array of images, with each image representing the classification for the corresponding color. Subsequent color-sensitive feature extraction algorithms in the pipeline can utilize these probability images in their calculations. Figure 3 shows an example of a probability image for the orange color produced by the color classifier next to the input image as well as the image on which the orange color model was trained.

4.6 RECOGNITION PIPELINE

Recognition and subsequent localization relative to landmarks, or unique competition objects, is critical to success in the RoboSub competition when the robotic platform used is not aided by a Doppler Velocity Logger; however, these landmarks are subject to change each year. From a software standpoint, it is desirable to interact with a generic landmark recognition interface rather than directly calling upon multiple specialized interfaces. In order accomplish this, Seabee utilizes an extensible landmark recognizer that accepts a landmark filter and calls on child modules to perform specialized landmark recognition. This ensures that, with the exception of drastic changes to the competition, landmark-dependent algorithms can remain mostly unchanged, while specialized recognition algorithms can be easily developed, tested, and deployed through a standard, familiar interface.

Where possible, it is desirable to be able to recognize landmarks via an adaptive, generic system, thereby avoiding as much specialization-related overhead as possible. SeaBee accomplishes this by calling on a scale- and rotation-invariant feature recognition system which utilizes OpenCV 2D

feature extraction, or contour extraction, performed on color- classified images produced by the vision pipeline. A set of template contour features, in the form of normalized, rotation-aligned histograms, is trained for any landmarks or landmark components and passed along with any candidate contour features located within incoming images to a generic recognition algorithm, which calculates and returns match qualities for each template-candidate pair. In this way, specialized landmark recognition algorithms can offload a significant amount of specialization to a central, generic system, yet still ensure the use of alternate, arbitrarily specialized recognition methods.

In its current state, SeaBee uses a specialized recognition algorithm for each unique landmark, excepting landmarks differing only in color. Furthermore, recognition is specialized based on the expected sensor source of landmarks relative to the sub; for example, we assume that pipelines and bins will only enter the field of view of the downward-facing camera, while buoys, hedges, and windows are expected to be found only in the field of view of the forward-facing camera. Given this assumption, we only search for the former set of landmarks in the images streamed from the corresponding source, and so on.

We assume that certain landmarks are only located within certain parts of the competition pool, and we further assume that given an arbitrary set of goals, only some subset of these landmarks need to be recognized. Given these assumptions, we conclude that it is possible to search for only some subset of landmarks dependent on our current location and/or goal. Therefore, it is desirable to utilize some simple means of applying a landmark filter, with either narrowing or widening constraints, through our recognition algorithms, in order to improve performance. We accomplish this via a custom color- and shape-based filtering API that accepts a list of filter items, each specifying either a narrowing or widening constraint to be applied to the color or type of a landmark. For example, when we are attempting to locate a buoy, we look for orange pipelines and buoys of any color on approach, then look for buoys of a single color on each buoy-touching attempt, then look for only orange pipelines and yellow hedges as we attempt to locate the first hedge, etc.

4.7 SENSING AND LOCALIZATION

Currently, our most advanced sensor is an XSens MTi IMU. This inertial measurement unit provides us with “drift-free” 3D heading and acceleration data calculated by an on-board EKF fed by the device's accelerometers, gyroscopes, and magnetometers in realtime at 100 Hz. Our team used this device effectively at the 2011 RoboSub competition to maintain a surprisingly accurate heading while navigating un-assisted within the competition pool. This year, we plan to expand our use of this device to pitch and roll stabilization, both of which proved to be significant issues for SeaBee while moving at high speeds during last year's competition.

SeaBee's hull is also fitted with an external pressure sensor, which is used to estimate the absolute depth of the AUV below the surface of the water via an experimentally-derived conversion from arbitrary pressure units to a distance in meters. With our current electronics, this measurement is taken at about 10 Hz.

Many other teams utilize a Doppler velocity logger (DVL), a very precise sonar device which can provide the linear components of velocity and pose of the sensor with respect to its environment. This sensor is currently out of our price range, so we must rely on alternate,

often noisy sources of odometry. Furthermore, both the sensor and thruster configuration of our platform is almost always subject to change; depending on the progress of the electrical and

mechanical teams, we may or may not have at our disposal a variety of sensors, each with varying capabilities in terms of both functionality and measurement noise. For this reason, we have found it necessary to develop both a generic realtime simulation of our vehicle's dynamics, as well as a generic Bayesian measurement fusion system capable of combining all components of all observables, whether simulated or actual, into corresponding "filtered" measurements.

Given our current sensing capabilities, the linear components of pose and velocity (those that would be trivially provided by a DVL) are the most difficult for us to obtain. In order to compensate, we run a realtime simulation of the sub's dynamics using a software library called BulletPhysics. Given our vehicle's mass, per-axis linear drag coefficients, the current thruster configuration (per-thruster capabilities and relative pose), and the motors value being set on each thruster, we are able to calculate all components of pose and velocity with moderate accuracy. We then fuse this output and any other odometry measurements together on a per-axis basis into a complete odometry estimate.

When combined, our filtering and simulation modules allow for maximum functionality and modularity within the constantly changing constraints of our platform; as sensors are added and removed, the accuracy of the corresponding measurements will vary accordingly. For example, if we were to integrate a DVL into our platform, we would expect to see the accuracy of the linear components of our odometry increase significantly. As an added bonus, these systems also allow for the advanced testing of other software modules via arbitrary levels of simulation of sensor values and other information, in circumstances when a desirable real-world testing environment is not practical or is entirely unavailable, or when the effects of a theoretical change to the system need to be studied.

4.8 NAVIGATION PIPELINE

As with most of our software, we sought to build our navigation system out of modules with varying levels of specialization, connected by generic interfaces. At a high level, our current implementation accepts a series of navigation constraints in the form of "waypoints," generates a trajectory with an arbitrarily high "temporal resolution", and then attempts to follow the trajectory within an additional set of constraints. This functionality is distributed over several modules including a trajectory planner, a high-level trajectory follower, a low- level velocity-based controller, and a platform-specific serial interface. Any module wanting to accomplish trajectory-based control of our vehicle must provide a trajectory to be followed. Within our system, a trajectory is composed of discrete intervals, each containing a constant acceleration over that interval and the desired state of the vehicle at the beginning of that interval, in the form of a waypoint (pose and velocity). Low-level velocity-based control is also possible; indeed, it is utilized by our trajectory following module, which is discussed in a later section.

A trajectory-planning paradigm was developed to simplify the creation of these trajectories while following waypoint-based constraints. In general, a trajectory planner accepts a list of two or more waypoints to be traversed in order, including the initial state of the vehicle, any intermediate states, and the final state of the vehicle, and returns a trajectory that meets the given constraints as closely as possible. The trajectory planner also accepts a temporal resolution parameter, which determines the maximum length of intervals over changing accelerations, as well as constraints on the maximum velocity and acceleration of the resulting trajectory (used to specialize the trajectory for the capabilities of a given platform). We currently use a linear trajectory planner, which iteratively generates a trajectory between each waypoint, ignoring intermediate velocities for simplicity. Starting with the initial vehicle state, the planner attempts to accelerate at the maximum given acceleration, up to the maximum given velocity, and generates a new trajectory interval for each change in

acceleration, following a simple set of rules: if the error in position to the current waypoint is below some threshold, the planner returns an empty trajectory; otherwise, if the error in position is below some threshold, the planner attempts to strafe to the desired position, then attempts to face the desired heading; otherwise, if the error in position is above some threshold, the planner attempts to rotate the vehicle to face the location of the desired waypoint, then attempts to translate the vehicle to the desired position along its forward axis, and finally attempts to face the desired heading. After the final waypoint is reached, the planner returns the complete trajectory.

After a trajectory is generated by a trajectory planner, it is passed on to a trajectory follower for realization. In this case, the trajectory follower is generic due to our generic trajectory design; regardless of the implementation of the planner that generated the trajectory, the trajectory follower can utilize the same algorithm to traverse the given trajectory. Along with a trajectory, this module also accepts constraints related to the accuracy of the realization of the given trajectory, including the maximum deviation from the trajectory, both in pose and time, as well as the planner to use, if any, to attempt to recover in the event that the given constraints are not able to be met. If this failure occurs, the trajectory follower notifies the module that initiated the current goal, then attempts to re-plan from the vehicle's current pose to the beginning of the given trajectory, if a recovery planner was specified. In this way, the trajectory follower can be passed a trajectory that does not necessarily start at the vehicle's current state, and it will automatically prepend a path that brings the vehicle to the beginning of the original trajectory, if possible. For a given interval in the trajectory, the follower interpolates between the starting velocity and the calculated ending velocity according to the acceleration specified by the interval, for the duration specified by the interval, publishing a desired velocity at each step; this interpolation occurs at a user- specified rate, and can therefore be specialized for a given platform. Realization of a trajectory, then, is as simple as iterating over all intervals in the trajectory, interpolating, and performing any recovery applicable recovery behaviors.

The output format of the trajectory follower (velocities) was selected to ensure generic means of controlling a given platform. However, generic conversion from desired velocity to platform-specific motor values was not handled, though it is likely feasible. Instead, we opted for a PID-based controller over error between current and desired position; specifically, for each movement axis, we employ both an independent PID controller and a specialized conversion function. In the current implementation, these axes include all individual linear and angular axes. However, we only seek to stabilize pitch and roll, while we seek to both stabilize and manipulate all other axes. That is, we actively attempt to keep the actual value for each axis near the desired value for each axis, but we do not allow the desired value of pitch and roll to vary from zero. Therefore, in high-level code, we assume that the pitch and roll of the vehicle are near zero. In the future, we may lift this constraint on pitch in order to assist in diving; the current implementation was chosen in order to simplify the vehicle's behavior.

The final actuation is handled by a custom serial driver written specifically for our vehicle's current hardware implementation. This driver is able to read sensor values, including internal pressure, external pressure, and kill switch state, as well as set the voltage of all motor drivers on the vehicle's power board.

In order reduce the significant overhead required to interact with the vehicle's complex control systems, we have implemented a set of motion primitives that allow the vehicle to be commanded through a short list of simple yet powerful functions. These functions will automatically invoke the functionality of the appropriate navigation components mentioned earlier, and include the ability to move to the position and/or orientation encoded in a pose, align to a position, orbit a given

position, and activate the torpedo launchers and marker droppers. Furthermore, the pose of a named object (whether static or dynamic), such as a buoy or any other landmark, can be trivially looked up and passed to these functions, enabling the creation of short, easily-readable high-level navigation code.

4.9 COMPETITION AI

The final, highest-level controller in our software architecture is our competition AI. This component is responsible for directing our vehicle to perform the most effective possible action at any given time, given all known goals and constraints. It is also responsible for enabling/disabling lower-level, situational modules and filters, such as those related to movement, object recognition, and behavior production. We fulfill these complex requirements by discretizing all actions into subtasks, each containing any actions to perform as well as the cost (in terms of distance and estimated completion time) and reward (in terms of points earned) associated with the task. We then build a task tree of arbitrary height, thereby allowing for an arbitrary level of task specificity, and feed this structure into a custom hierarchical cost-based decision algorithm, which attempts to maximize the overall reward earned in the allotted time. Any distance-based costs are converted into estimated completion time using the vehicle's current pose and any distance that would be accumulated while traversing the tree down to that task.

ACKNOWLEDGEMENT

Support from The University of Southern California, iLab, the USC Dornsife College of Letters, Arts, and Sciences Machine Shop, and the Viterbi School of Engineering allows USC AUV to continue to be an integral part of student research at the USC Viterbi School of Engineering.

Thank you to our industry sponsors: the Boeing Company, Northrop Grumman, Digi-Key Corporation, Lockheed Martin, and ADL Embedded Solutions.

REFERENCES

M. Quigley, K. Conley, B. Gerkey, J. Faust, T. B. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, “ROS: an open-source Robot Operating System,” in International Conference on Robotics and Automation, ser. Open-Source Software workshop, 2009.