Gonzaga University Autonomous Underwater Vehicle Implementation

Tyler Willis, Liam Jones, Aaron Wong, Daniel Mobley, Navath Nhan, Aaron Weber, Brandon Takahashi, Kevin Mattappally, Kimbrell Larouche, Eric Av, Marissa Encarnacion, Jalen Tasciat, Philip Whitney

***Abstract*—Gonzaga's first RoboSub AUV entry features ideas that adapt to limited resources and experience, but are mostly in themselves unique. Due to limited access to experienced individuals, the design for the autonomous systems was completed by our memebers after years of creative brainstorming and trying new approaches.**

The most prominent acheivment of this design is having a fundamental structure that is easy to reproduce by others because all the code relies on open source frameworks and our hardware is not very costly to obtain. We have also aimed to make our Graphical User Interface as easy to use as possible. Our mechincal design is unlike our previous two generations of protoypes which were heavy, leaky, not easily maneauverable, and difficult to maintain from test to test. Our new design resolves all these issues and allows for adaptiblity in future competitions. Most Importantly, the overall design allows us to have something to build on and continue moving forward with as our project grows in the years to come.

I. Introduction

The Gonzaga RoboSub Team is in its very early stages and after a couple years of research and development has figured out a way to compete while still allowing the chance to learn practical approaches to problems presented by competition tasks. Our primary focus this year will be to learn as much as possible and score as many points as our current system will allow us. We inherited a large, bulky platform that served us well for learning about motors and sensors, but had far too many practical limitations to be worth bringing to competition. Beginning in the spring semester of this year, we built a lightweight chassis that is currently equipped only for the specific tasks we've chosen to complete, but with room to add other capabilities later on. With the very limited resources we have (i.e. money, faculty, prior experience) we've had to make critical decisions based on the competition's design to decide which competition goals we could attempt.

II. Approach to Competition Strategy

Our plan of approach to the complexity of the course is keeping things as simple as possible for the first time around to maximize success. Our final decision about the course was to be able to do the qualifying maneuver, obtain a chip from the dispenser, and lastly hit a six sided dice. The rest of the competition would require system solutions that are currently beyond the limit of our resources and skills. That being said, we left space open for additions and modifications going into next years competition.

III. Mechanical Components

We inherited a design that featured a large enclosed volume requiring a lot of ballast weight, a flat frame that didnt allow room to add actuators or cameras, and a finicky sealing mechanism that took a lot of force to close and required a time consuming seal check every time we closed it. That vehicle allowed us to develop and test our motion tracking system and our motion control system, but that was about all we could build into it. Starting when Daniel and Navath took over as mechanical leads in January, we knew we needed a new chassis to work with. All we really knew was what to avoid, and what other teams and companies designs looked like. We also had a tight timeframe, needing to create an original design that would be easily manufacturable, while addressing the clumsiness issues of the first generation and providing a base to develop on in future challenges. With these goals in mind, we created the sub you see today.

A. Frame

The old subs frame was a flat rectangle, good for mounting motors and holding the hull, but that was about it. Adding launchers or actuators would have been hard to balance, because they would be by necessity offset from the center of the sub. For the new design, we wanted to have it be self-righting, have space for actuators, grippers, and launchers, and be easily adjustable as future needs require. This lead to the open box design, with the electronics and battery tubes suspended from the top of the frame and space for future development below. The crossbars were moved inwards to maximize the cameras field of view and reduce unnecessary frame components. We chose the outside dimensions of 24x18x14 inches because it worked well with the electronics and battery packages we need to incorporate, while being small enough to be easily handled above water. Finally, we built the frame out of 80-20 T slot aluminum for its expandability and ease of modification.

B. Hull

The old subs hull was 40 inches long and about 16 inches outside diameter, requiring an absurd amount of ballast to compensate for the displacement. We had to go smaller. We based our main hull size on the Bluerobotics 4.5 inch series because it was the smallest platform that would accommodate the electronics we use, and because it had a proven thru-hull connector design that would work with the motor wires we use. Based on designs from bluerobotics, other Robosub teams, and our own volume calculations, we realized we could minimize displacement by moving the batteries to external tubes, which would also give us more freedom in locating them for maximum stability. This did add complexity in terms of more seals and more cable connectors, but we think that the benefits of a smaller main tube are worth it.

C. Electronics Rack

In the old sub, the electronics tray was a piece of yard sign mounted between two of the same pieces that were used for the subs frame. This excessively closed in design decimated the usable space in the otherwise too-large hull, making routine maintenance and upgrades a challenge. In the new sub, we took cues from the Bluerobotics design, but realized that we didnt need the external bracing system. This design allows us to maximize space by using both sides of the board to attach electronics and controls. We made a strong push towards using 3D printed brackets and mounts instead of hot glue and cardboard, as this encouraged more forward planning and a cleaner look, which is very necessary with a clear hull.

IV. Electronics Systems

The Submarine is powered by six 5000 mAh Lithium Polymer batteries and is connected in-line with a relay to act as our killswitch, which is activated when a magnet is removed from a Hall effect sensor.

V. Mission Computer

It was determined that it would be best to have a Mission Computer running Linux on board to handle all autonomy related decisions, interface with the image recognition camera and communicate with the Microcontroller with easy-to-use functional control commands via a custom UART serial protocol (described in more detail in the Microcontroller Unit section). All Mission Computer code is written in Java, and all communications with the Microcontroller and camera are abstracted away for ease of use. The Computer follows sequential step by step procedures to complete each task, while making real-time decisions that allow the Computer to decide when and how to perform each tasks. There is also a graphical user interface that it is used for testing and debugging and was made with the idea of keeping debugging time to a minimum. OpenCV for Java was used for interfacing with a Logitech Camera.

VI. Embedded Systems

A. Microcontroller Unit

The microcontroller used is the TM4C123GH6PM. This unit was chosen for its widespread support as well as flexibility in functionality. These qualities made it possible to prototype and develop functionalities in a timely manner. The microcontrollers functionalities were accessed through the widely supported TivaWare drivers. The drivers made it easy to use the various peripherals provided without extensive knowledge in the microcontrollers architecture. The flexibility of the Nested Vectored Interrupt Controller allowed for a responsive system. The use of interrupts provide an illusion of concurrency which is a key component of the embedded system. The interrupt service routines provide 3 main control loops. The main function loop, the UART receiving interrupt service routine, and the real time interrupt service routine. The UART receiving interrupt service routine is triggered when a character is received on the UART channel. The main function loop controls prototyping and specific function testing, the real time interrupt service routine runs the PID loop that alters motor values to achieve the given set point. The UART interrupt service routine appends the received characters onto a global string. When the interrupt service routine has received our predetermined ”end of transmission” character, it will then proceed to process the string it has received. Each of these strings will be 6 bytes long, 1 byte as an identifier character, 4 bytes as a standard IEEE 752 floating-point number and the final byte as our predetermined ”end of transmission” character, the ” ”. The identifier character appended to the start of the string will signify what the subsequent floating value represents. Different identifiers have been selected to represent different values such as desired depth, desired heading or desired forward thrust. A special identifier character ”*” signifies that the subsequent characters will represent a debugging string that the main function loop will handle. Upon receiving the special identifier character, a flag will be raised to signify for the main function loop to execute its debugging scripts. Upon receiving a general identifier with a floating point value, the UART interrupt service routine will store the received float in its appropriate variables and change the appropriate flags to signal a new setpoint has been received. The main function loop waits on the ”foundEOT” flag. This flag is raised when a special identifier character * has been received. The main function will then parse through the received string which often contains a debugging request like ”pssr” which requests for the microcontroller to test the pressure sensor, or ”mtr” which requests a motor test. This main function loop is used primarily for debugging and prototyping purposes. The real time interrupt service routine waits its timer to expire before triggering. It is set to trigger once every 100ms. The real time interrupt service routine will run through a PID calculation, taking in the current sensor data and comparing it to the desired sensor data. It will then compute the error between the two values and the 3 corresponding proportional, integral and derivative values. These values are recombined to give a motor output value that will be used to bring the submarine closer to the set point. This loop will run continuously to bring the submarine to its set point and hold its set point. Due to the nature of the PID algorithm, the computation has to be computed at very specific intervals for the output value to be meaningful. The use of a real time interrupt service routine is crucial to maintaining a consistent sample time.

B. Sensors

The TM4C123GH6PM also interfaces with a large portion of the sensors on the submarine. These include the depth sensor, accelerometer, gyroscope and magnetometer. The MPU9150 is used to provide acceleration, gyroscopic and magnetic heading data while the MS5837 is used to provide pressure data. The MPU9150 is housed as a sensor boosterpack for the TM4C123GH6PM while the MS5837 is housed as the Bar30 as provided by BlueRobotics. The microcontroller interfaces with the MPU9150 through I2C and performs a complementary filter and a discrete cosine matrix calculation to compute the submarines current magnetic heading in degrees. Currently, the depth sensor is being interfaced with through an Arduino Nano. The Arduino Nano reads the data from the depth sensor through I2C, applies the appropriate conversions, and sends the depth to the TM4C123GH6PM through UART.

VII. Design Creativity

In writing the software for the submarine, the team was faced with the challenge of designing a protocol that was easy to use, efficient and descriptive. The protocol had to communicate both the Raspberry Pi as well as the Texas Instruments TM4C123GH6PM. In addition to the hardware restrictions, the software team had chosen to use Java as their programming language for the mission computer. In light of these restrictions, the software team designed a protocol around the rudimentary UART serial communication. UART serial communication is widely used in both microcontrollers like the TM4C123GH6PM as well as Raspberry Pis. The team was tasked with configuring a Java application to communicate through UART, designing a communication protocol that was efficient, and implementing the protocol on all systems.

In order to access the serial communication port, the software team wrote a driver to wrap the JSerialComms Library. The driver configured the serial port to have no parity bits, one stop bit with flow control disabled. This driver allowed us to easily write byte or char arrays to the serial port. This was used to send characters that could then be read as commands by a receiving microcontroller. These commands were used for debugging and prototyping. A protocol was written around UART communication to send data values between the Raspberry Pi and the TM4C123GH6PM.

The communication protocol designed uses 6 bytes for a single communication unit. The communication unit can describe an identifier and a floating point value attributed to that identifier. The identifier can describe a characteristics like a desired depth that the mission computer would like to achieve or a current magnetic heading that the submarine is facing towards. The first of the six bytes would contain the identifier character followed by 4 bytes that describe the standard IEEE 752 floating-point number and the final byte as our predetermined ”end of transmission” character, the ” ”. Using a single byte as an identifier allows us 255 unique messages which was sufficient to communicate with. This protocol proved to be lightweight, easy to implement and sufficient for communicating between the microcontroller and the mission computer.

VIII. Experimental Results

The software team went through several control boards before settling on the current Texas Instruments TM4C123GH6PM microcontroller. The team initially used an Arduino Mega as a microcontroller. The Arduino Mega provided the team with quick prototyping, essential low level features as well as access to a wide range of open source libraries. The Arduino Mega was essential in allowing the team to develop the mission computer and other peripheral components. The Arduino Mega was unfortunately limited in the interrupt capabilities available. The software team found it essential to have a consistently timed sample rate for the PID control loop. The Arduino Mega had limited support for its interrupt service routines and could not provide the functionalities needed. The software team then transitioned into prototyping for a flight controller as the submarines control board. The flight controller selected was the PixHawk V2. This flight controller along with the ArduSub framework allowed the team to experiment with a robust control system. While the PixHawk provided the software team with numerous functionalities, the software team found it difficult to interface with PixHawk through their mission computer. The software team finally landed on the TM4C123GH6PM. This unit was chosen for its widespread support as well as flexibility in functionality. The flexibility of the Nested Vectored Interrupt Controller allowed for multiple interrupt service routines which is heavily used in the submarines control system.

The mechanical team did a number of experiments to make sure the new tubes were sealed, involving different types of grease and different designs for the O-ring grooves. 3D printing allowed us to create, test, observe, and change designs much more rapidly than if we had to send off drawings to a machine shop, as well as adding more ports to the caps as the electrical team needed them. We also experimented with a number of different sealing greases for the o-rings. We started with vaseline since its what we had in our cabinet, but we werent impressed by its sealing ability or its durability. We tried a generic silicone grease from Ace hardware, and it was better at sealing, but it still would wash away after a few tests. Finally, we got some silicone grease from our local pool supply store called Magic Lube, and its significantly greater viscosity

Acknowledgments

We’d like to thank Janeane Schmidt who has been a big help in shedding some publicity light on our project, and our advisor Patrick Nowacki who always listens to our crazy ideas and helps us move forward.

APPENDIX: COMPONENT SPECIFICATIONS

Component	**Vendor**	**Model/Type**	**Specs**	**Cost (If New)**
Buyancy Control	NA	NA	NA	NA
Frame	8020	1x1 in. T Slot	20 feet, cut to length	$60
Waterproof Housing	Bluerobotics, Home Depot	Acrylic PVC pipe, 3D printed endcaps	4.5 in dia 0.5 in. thick	$80
Thrusters	Blue Robotics	T200		$169x8
Motor Control	Texas Instruments	TM4C123GH6PM		$16
High Level Control	Raspberry Pi	Raspberry Pi 3B+		$36
Actuators				
Propellers	Blue Robotics	Propellor Set		$5
Battery	Turnigy	4S1P 14.8V 20C Hardcase Pack		$32.80x6
Converter				
Regulator				
CPU	Raspberry Pi	BCM2837B0	Quad-core A53 64-bit @ 1.4GHz	
Internal Comm Network				
External Comm Network				
Programming Language 1	Java			
Programming Language 2	C++			
Compass	Texas Instruments	MPU9150 (SensorHub BoosterPack)		$40
Inertial Measurment Unit	Texas Instruments	MPU9150 (SensorHub BoosterPack)		$40
Doppler Velocity Log				
Camera(s)	Logitech	C310	720p Webcam	
Hydrophones				
Manipulator				
Algorithms: Vision	Open-Source	OpenCV		Free
Algorithms: Acoustics				
Algorithm: localization and mapping				
Algorithms: Autonomy	Texas Instruments	TivaWare		
Open Source Software	OpenCV, Raspbian Jessie OS			
Team Size	16			
HW/SW Expertise Ratio	1:3			
Testing Time: Simulation				
TEsting Time: In Water	15 Hours			