Entering The Realm

Nathan Henault, Lauren Strand, Michael Marquez and Luke Barbier

***Abstract*— In order for the Boulder RoboSub team to complete its first-ever tasks beyond the start gate, the team has focused on the fundamentals of building up a modular software stack, incorporating a computationally assistive GPU and expanding mechanical designs for more complex tasks. By doing so they have made themselves a competitive candidate for the 2019 finals and have built a strong platform for future development.**

A. Competition Strategy

- **We focused on the fundamentals**

We have steadily increased our rankings from 35th three years ago, to 19th two years ago, to 11th last year. It is our goal to get into finals this year, and if last year's scores are a model for the estimated number of points we would need to get into finals, we would need around 3000/4000 points to have a likely chance of getting into finals. We can achieve this by maxing out the start gate, path and buoys tasks. Our stretch goal to give our team an even higher chance of participating in the finals is dropping markers through the open slot in the droppers task.

If successful, this would be the first year in which our team has completed tasks "in the realm," tasks beyond the start gate. We had a strong controls and state estimation system that enabled us to score full points on the start gate, but our 2018 core mechanical, electrical and software systems were insufficient to perform the realm tasks. This is because of general lack of developed software systems, slow bounding box generation speed and lack of physical hardware to perform droppers.

Our team made progress on all of these issues this year and our vehicle is well capable of maxing out the points in start gate, following the path, and hitting its target buoy. Integration work remains on our torpedos, but our pneumatics enclosure and droppers have already been manufactured.

Besides not having a strong state machine and perception stack to complete more tasks than the start gate, we saw the largest barrier to performing the buoys as our bounding box generation speed. To resolve this issue we installed a GPU and redesigned mechanical and electrical systems as necessary to accommodate.

Similarly, in 2018 we did not have the hardware capability to attempt the droppers tasks. Our mechanical team went about constructing a new pneumatics enclosure to effectively fit a powerful pneumatics board.

To top off our competition strategy, this year we went about constructing and practicing the start gate, path and buoys at our campus' divewell before reaching the competition.

B. Vehicle Design

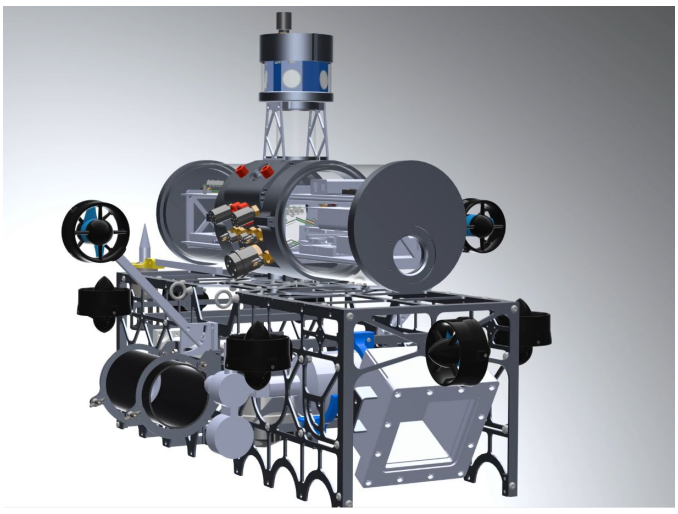

Fig 1: Leviathan 2019 CAD Model. Biggest mechanical difference from our 2018 vehicle is our upgraded frame.

1. Mechanical

This year, the team designed a new frame, iterated on last year's electronics rack, and fabricated a new

pneumatics enclosure. The frame was sized to comfortably accommodate our hydrophones, DVL, pneumatics, pressurized air canister, torpedos and dropper mounts. Our team does not plan to utilize our torpedo or hydrophones mounts, but space has been provided. It also ensures our DVL has a large enough safety clearance from contact beneath it. The frame also features integrated mounts for the battery pods, which are closed by a simple fastener and make the pods more accessible than last year's design. Pre-drilled holes allow for mounted parts to be moved around if needed, and for new parts to easily be added. A more rigid and custom electronics rack was made to better organize the electronics and wiring. The electronics rack was designed with the intent to optimize the space inside the hull. Lastly, the pneumatics enclosure was redesigned to fit the new PCB, which is smaller than the previous year's.
Last year, the team was extremely limited by the frame's size because they struggled with where to mount the different enclosures. The frame was also too short in height, and required temporary feet to be added so that the DVL did not touch the bottom. With the new design, there is more area for parts to be mounted and allows for more flexibility. However, the frame is slightly heavier which requires the motors to run more, and thus drains battery life quicker. The team's old electronics rack was made from acrylic, which broke under the weight of the boards and computer. A new rack was designed and fabricated from aluminum, providing more support for these components. Last year, the team used a pneumatics enclosure that was the same design and size as the hydrophone enclosure. With limited space on the old frame, it was a challenge to fit both. The enclosure was larger than it needed to be, and added a lot of unnecessary weight to the sub. The pneumatics enclosure has been redesigned this year to fulfill the same purpose but at a much lower cost space.

2. *Electrical*

Leviathan features two cameras, one IMU, one depth sensor and a DVL. Our first camera is the Occam Omni 60 camera, located in the periscope on top of our sub. This series of 5 cameras gives our autonomy 360 degree vision. Our second camera is our downcam, a BlackFly S GigE camera with a 141 degree angle Theia Lense. The primary role of all of these cameras is task localization, or map creation, through our perception pipeline. We have transform functions set up with the tf package in ROS to each one of these cameras. This role for these cameras suffices for the tasks we are aiming to accomplish because our solution for these tasks relies purely on localizing a task. Our task logic is often along these lines of once we find the task's pose, we can figure out what poses the AUV needs to go to to complete the task.

The reason we adopted this task solving strategy is completely derived from the amount of visibility our camera setup grants us and our good quality state estimation from our sensors. If we were fairly restricted on how much we could see, we might opt for task solutions, where, once found, we restricted our controls such that we did not lose sight of the task because we might have trouble finding it again. But this is not a problem for us because we can see 360 degrees horizontally and have a very wide FOV beneath us, so we will typically always see the task if it is spatially near us. This grants reliability to our task solving approaches because we have a high probability of seeing a task and localizing it.

The Sparton AHRS-8 IMU gives us absolute heading and orientation. We rely on this to solve the random start attribute of the start gate task. We do not use the acceleration data at all in our state estimation, but we do monitor it for a jump in the buoy task, indicating a successful hit.

Our depth sensor and DVL are the reason we can solve tasks by simply going to poses without visual feedback because these sensors are so accurate. Our depth sensor is a BlueRobotics Bar30 High-Resolution depth sensor, giving us depth accuracy to 2mm. Our Nortek DVL gives us velocity accuracy of ±0.1 cm/s. Combined with the magnetometer onboard the AHRS, these sensors allow us to reach the poses we request with high accuracy and reliability.

We also made many changes to our custom electronics board this year, because we needed to

reduce the space claim to make room for a GPU. Last year we had four boards, which we have reduced to two boards this year. Our merge/ESC power board has two functions, which were split into two boards last year and were very easy to condense into one board this year. The first task is to assure that the batteries are being used evenly, which is accomplished by limiting the amount of current drawn from each battery based on each of their voltages. The output of the merge circuit is a single battery voltage of about 14V-17V. The second half of this board is the ESC power outputs. This half has power outputs to our eight ESCs, each with a high side kill MOSFET to cut power to the ESCs when the kill switch is flipped.

Our second board has our power conversion circuits and our embedded system, and it connects to the merge board to receive a battery voltage input. The conversion side of this board contains five circuits to output 48V, 19V, 12V, and 9V, all of which are required by the off the shelf electronics within the sub. The embedded system is a new addition to the sub, which was added to remove many of the USB peripherals from the inside of the sub. For this year, the goal of the embedded system was to remove the need for the Pololu PWM controller and multiple Arduinos which controlled the depth sensor and pneumatics actuators. Removing these USB peripherals will free up more space for additional electronics within the sub by removing excess boards and the USB hub which we currently need to connect to everything.

3. *Software*

Our software team's approach this year was to focus on the fundamentals of using our hardware effectively for maximizing our points in the three tasks: start gate, path and buoys. This meant effectively using all of our sensors and building up our software stack with the following things: modular state machine, fast perception pipeline, and an accurate simulator. If we achieved all of these, we would be in a fantastic position for the 2020 competition in attempting the more complex tasks such as hydrophones, droppers, torpedos and fine object manipulation.

This approach built on our controls and state estimation tuning that we had worked on considerably last year. We were happy with what we had so no novel changes were made to our control system or state estimation this year.

Our software team continued solving the 2018 tasks after the competition in order to debug, validate and most importantly practice writing software for the 2019 tasks. We had never built and validated a state machine and perception stack before. These were simply not a priority if our controls and state estimation were not tuned in well. This was the first year our team was going to attempt tasks beyond the start gate, so we were in fairly uncharted territory and, in terms of our software stacks, we were starting from scratch.

3.1 UUV Simulator

In fitting with most software design paradigms, the first thing we built this year was a testing environment. We used the UUV simulator plugin for Gazebo which is designed for simulating hydrodynamics. Gazebo has a plugin for interfacing with ROS which made it fit very well with our existing state estimation and controls code. We put the CAD model of Leviathan, our divewell at CU, Transdec, the start gate, the dice, the path and the roulette table in the simulation and began development. Once the 2019 tasks came out, we added the vampire buoys as well.

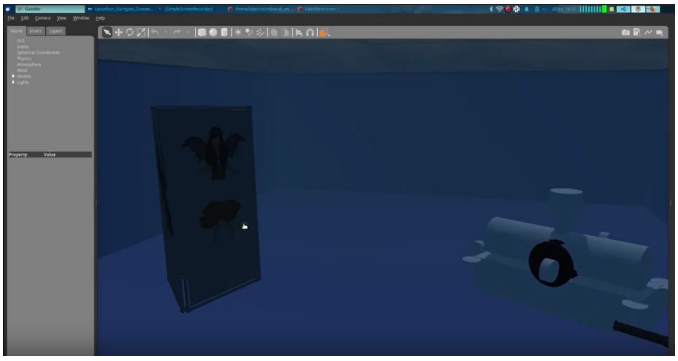

Fig 2: Leviathan about to slay a vampire on the triangular buoy inside our UUV simulator.

3.2 SMACH State Machine

The state machine architecture we chose was SMACH. We had attempted Smach development last year, but never made significant progress. Smach is natively python and because of python's development speed and its ROS support we chose to write all of our state machine code in python.

Our state machine consists of one outer smach state machine containing several other inner smach state machines. The inner state machines are the tasks, whose states are the steps to complete the task. The outer state machine manages the transitions between tasks.. At a high level, this architecture has been effective in structuring the methodology of how we solve tasks and can chain together tasks in a mission.

3.3 Nodeleted Perception Stack

We at least knew of Smach last year, but in terms of computer vision we were starting from scratch. Knowing that our state machine code was going to rely heavily on our ability to solve task poses, and that no matter how well we wrote the classical cv pose generation algorithms there were always going to be rare cases of lighting or some other form of noise that was going to give us a bad pose solve. So, we needed to generate poses very quickly so that we could effectively filter out these bad pose solves through averaging and outlier rejection.

To speed up our perception stack, we went with a software paradigm that every piece of code that touched an image was written in C++ and nodeleted. Nodelets are a feature of ROS that allow for skipping the serialization of interprocess messaging, allowing for much higher messaging speeds between processes. From our camera drivers to our ML algorithms to our pose generation code, images are passed by reference instead of by value.

The process we always follow in our pose generation code is: receive bounding box, employ some classical technique to extract key points, solvepnp the pose of the task. All of the difficulty in pose generation is cleverly utilizing classical techniques to find the 4 points in an image that can be put through an opencv solvepnp function call to localize the pose. The basic idea behind solvepnp is that if we know where 4 points of an object in an image are, we know how far away those points should be from each other in reality and we know the distortion coefficients of the camera, we can reverse solve for the pose of the object. We currently employ this pipeline with localizing the gate, path and buoys and look forward to experimenting with it on droppers and torpedos.

3.4 YOLO Neural Network Platform

Our ML platform for class recognition and bounding box generation is YOLO. Our primary use for YOLO in our perception pipeline is to lessen the problem for our classical cv algorithms. When our classical cv algorithms receive an image with the YOLO bounding box, they crop out the bounding box and reduce the problem space to just the bounding box. This effectively brings our problem to a known state where we can generally find the shape and color we are looking for.

The other practical use we have had for YOLO is recognizing our target vampire in the Slay Vampires task. When we are orbiting the triangular buoy we keep orbiting until we get indication from YOLO that the only vampire we are seeing is our target vampire. Upon which, we slay the vampire.

C. Experimental Results

Software testing was done both in our UUV simulator and at our divewell located on the CU Boulder campus. The UUV simulator showed its greatest value in debugging mission level code in our state machine and controls code for specialized movements. As expected of most simulators, it was not as useful for debugging computer vision code because it could not reproduce the real world problem to the precision required by classical cv techniques. YOLO did however, when trained with simulated and real world data, successfully generalize to both real world and simulated data, making the UUV simulator a valuable tool for debugging controls algorithms utilizing bounding box recognition.

Testing at the CU divewell occured weekly for two hours. This was the first year in which we constructed tasks to practice with. This ended up being instrumental in our ability to debug perception code and train YOLO models. Not only were we able to get example images to test classical code, but also, we were able to test the entire pipeline from camera driver to perception to autonomy code to motor control. We started off the 2018 fall by constructing the start gate, dice and the roulette table. In the spring we constructed the 3-legged start gate, path marker and both buoys.

D. *Acknowledgments*

The team would like to thank the CU Computer Science Dept for providing the space in which the club resides. In addition, the team would like to thank the EEF Committee and the COHRINT lab for the funding they have provided. The team would also like to thank the ITLL machine shop for the machining work they have done for the club.

Appendix A: Expectations

Below is the scoring table showing the points associated with each task. Enter the points you **expect to score** with the vehicle(s) that you have designed and engineered. At the end of the competition, enter the points you **actually scored** in the last column.

Subjective Measures			
	Maximum Points	Expected Points	Points Scored
Utility of team website	50	25	
Technical Merit (from journal paper)	150	135	
Written Style (from journal paper)	50	45	
Capability for Autonomous Behavior (static judging)	100	90	
Creativity in System Design (static judging)	100	90	
Team Uniform (static judging)	10	10	
Team Video	50	50	
Pre-Qualifying Video	100	100	
Discretionary points (static judging)	40		
Total	650	545	
Performance Measures			
	Maximum Points		
Weight	See Table 1 / Vehicle		
Marker/Torpedo over weight or size by <10%	minus 500 / marker		
Gate: Pass through	100	100	
Gate: Maintain fixed heading	150	150	
Gate: Coin Flip	300	300	
Gate: Pass through 60% section	200		
Gate: Pass through 40% section	400	400	
Gate: Style	+100 (8x max)	400	
Collect Pickup: Crucifix, Garlic	400 / object		
Follow the "Path" (2 total)	100 / segment	200	
Slay Vampires: Any, Called	300, 600	900	
Drop Garlic: Open, Closed	700, 1000 / marker (2 + pickup)	1400	
Drop Garlic: Move Arm	400		
Stake through Heart: Open Oval, Cover Oval, Sm Heart	800, 1000, 1200 / torpedo (max 2)		
Stake through Heart: Move lever	400		
Stake through Heart: Bonus - Cover Oval, Sm Heart	500		
Expose to Sunlight: Surface in Area	1000		
Expose to Sunlight: Surface with object	400 / object		
Expose to Sunlight: Open coffin	400		
Expose to Sunlight: Drop Pickup	200 / object (Crucifix only)		
Random Pinger first task	500		
Random Pinger second task	1500		
Inter-vehicle Communication	1000		
Finish the mission with T minutes (whole + factional)	Tx100		
		4250	

Appendix B: Component Specifications

In the past, a detailed list of components constituted the bulk of many paper submissions. This practice is discouraged as it distracts from the underlying strategic thinking, system engineering decisions, or novel contributions. For the record, teams should list the components actually used in the vehicle in the table below.

Component	Vendor	Model /Type	Specs	Cost (if new)
Buoyancy Control				
Frame		Custom		1200
Waterproof Housing		Custom		
Waterproof Connectors	~~Amphenol~~	Subconn		
Thrusters	BlueRobotics	T200		150 x 8
Motor Control		PID		
High Level Control		SMACH		
Actuators				
Propellers				
Battery	Multistar	10,000mAh 4S		90
Converter		Custom		400
Regulator		Custom		400
CPU	Intel	NUC		250
Internal Comm Network	Ethernet	Switch		50
External Comm Innterface	BlueRobotics	Fathom		160
Programming Language 1	Python			
Programming Language 2	c++			
Compass	Sparton	AHRS-8		1400
Inertial Measurment Unit (IMU)				
Doppler Velocity Log (DVL)	NorteK	DVL 1MHz		~~10K~~
Camera(s)	Occam	Omni 60		
Hydrophones				
Manipulator				
Algorithms: vision	Yolo	Darknet	OpenCV	
Algorithms: acoustics				
Algorithms: localization and mapping	SolvePnP	Watershed	Hough lines	
Algorithms: autonomy				
Open source software	ROS			
Team size (number of people)	14 comp	30 home		
HW/SW expertise ratio	50:50			
Testing time: simulation	200+h			
Testing time: in-water	80h			