Design and Implementation of the Hydro Bison Autonomous Underwater Vehicle

Andrew Jones
North Dakota State University
Fargo, ND, USA
andrew.jones.4@ndsu.edu

Daniel Anderson
North Dakota State University
Fargo, ND, USA
daniel.anderson.3@ndsu.edu

Isaac Burton
North Dakota State University
Fargo, ND, USA
isaac.burton@ndsu.edu

Nicholas Snell
North Dakota State University
Fargo, ND, USA
nicholas.snell@ndsu.edu

Jaron Pollman
North Dakota State University
Fargo, ND, USA
jaron.r.pollman.2@ndsu.edu

Benjamin Kading
North Dakota State University
Fargo, ND, USA
benjamin.kading@ndsu.edu

Dr. Jeremy Straub
North Dakota State University
Fargo, ND, USA
jeremy.straub@ndsu.edu
[Faculty Advisor]

***Abstract*—In this paper, the Hydro Bison's RoboSub 2018 competition AUV is discussed. An overview of the vehicle and its software is presented. The planned strategy of the AUV is given for each task and for the overall competition.**

I. Introduction

The Hydro Bison team participating in this year's RoboSub Competition is comprised of students from North Dakota State University (NDSU). This marks the first year of participation in RoboSub for all team members and advisors. The design process alone has been a great learning experience for the team and we are excited to participate in this year's competition.

Our team is primarily software oriented, with over half the team being computer science majors. As such, our focus has been on simulations and developing the task specific software for the competition. With this being our first year competing, it has been a particularly challenging experience as well as an excellent learning opportunity.

On the mechanical side, our focus has been less on design optimization, and more on fulfilling mission requirements. This has led to a more modular design, with more flexibility in regards to potential component arrangements and configurations.

II. Vehicle Overview

Our AUV has a frame that consists of both aluminum and HDPE. The HDPE has the benefit of being near neutrally buoyant [1], while the aluminum provides additional strength and its slight negative buoyancy helps offset the positive buoyancy of the acrylic enclosures.

The vehicle has an eight-thruster setup, with four for vertical movement and four for horizontal movement. The utilized setup is also known as 'vectored 6dof' and is supported by the Pixhawk ArduSub firmware.

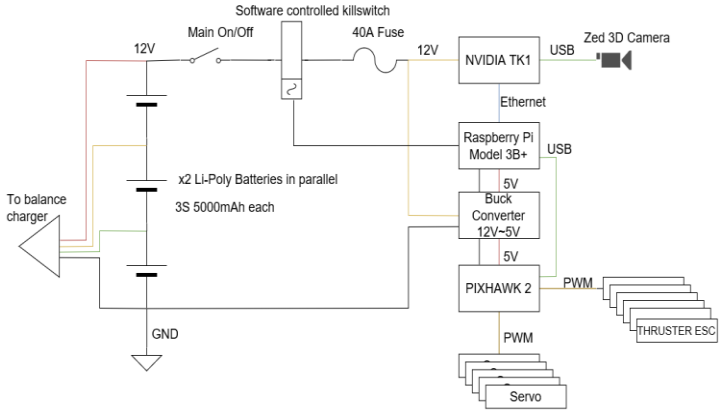

Fig. 1. Electrical Diagram

The Pixhawk acts as the motor controller for the eight thrusters, as shown in Fig. 1, and is controlled via the Nvidia TK1 (using a raspberry pi 3 as an intermediary). The TK1 is used in order to process the point cloud data from the ZED camera, which is the primary sensor on the AUV.

III. Software Overview

The software architecture for our AUV, depicted in Fig. 2, consists of original Python code, ROS,

OpenCV and the ZED SDK. ROS is currently only used for localization, although further usage of it may be added in the future (for subsequent competitions), particularly when ROS 2 becomes available.

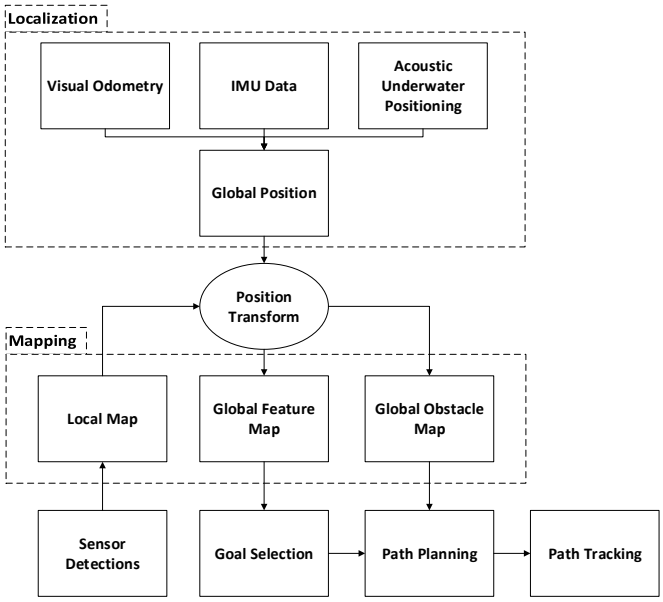

Fig. 2. Software Overview Diagram

In the following subsections, an overview of each of the main software modules is presented.

A. Localization

For localization, the data from the IMU on the pixhawk and the visual odometry from the Zed camera are published to ROS. Then, the 'robot_pose_ekf' ROS node fuses the data and uses it in an extended Kalman filter to estimate the current position of the vehicle. Hydroacoustic positioning would be an additional valuable positioning utility and may be added in the future.

B. Mapping

The Nvidia TK1 utilizes the ZED camera along with the ZED SDK to generate a point cloud in each frame such that each pixel has (x, y, z) spatial position values. The raw images from the ZED camera map one to one with the point cloud, allowing analysis and filtering in the image to be mapped to the corresponding point cloud position values. This allows features of interest in an image to be mapped to spatial locations.

The detected features of interest and obstacles are first placed in a local map. Then, a position transform is applied to local map so that it can be merged to the global map. A separate global map is kept for features of interest and detected obstacles / arena boundaries. The global obstacle map is used for path generation and traversing, while the global feature map is used for goal determination and task related activities.

C. Pathing

Due to the relatively small amount of obstacles (compared to the amount of open space), our AUV's path generation is done using A* and updated if a newly detected obstacle obstructs the current route. The grid used for the search is the global obstacle grid, with the goal position determined by a separate goal planning thread. Furthermore, instead of calculating the path in 3-dimensions, the horizontal plane is planned out first with A* using a Euclidean distance heuristic. This is because grid search methods can be computationally expensive when considering more than two dimensions [2]. Instead, the depth descent/ascent path is planned out after the horizontal route is calculated.

The generated path is then given to the path tracking thread, which utilizes a modified pure pursuit path tracking algorithm. For certain precision-oriented task activities, the tracking is changed to a set of predetermined motions that are specific for the current task.

IV. Competition Strategy

In this section, the strategy for our AUV to accomplish each individual task is discussed. Then, our overall strategy for which tasks are prioritized over others is presented.

A. Task Strategies

1) "Enter Casino Gate"

For the first task of passing through the validation gate, our strategy involves processing images from the zed stereo camera in order to detect the distinctive orange color of the vertical posts. Once located, the detections in the image can be mapped to the point cloud so that the position of the posts relative to the vehicle can be determined. The red and black horizontal posts are then color filtered for, such that they can be used to ensure that the AUV goes between the orange posts (in case of potential misdetection of one of the posts). In the case of an accurate reading of the locations of both orange

vertical posts and the distinction of the red side/black side of the horizontal post, the AUV passes through either the red side or black side (depending on predetermined input) and remembers which side was chosen. If the distinction of which side is red and which is black cannot be accurately determined, the AUV simply plots a course to pass through the gate.

2) "Follow the Path"

To follow the orange path leading to the 'shoot craps' and 'play slots' tasks, our AUV filters images for the orange color range and localizes those detected image portions (if any are detected). Due to the low positioning of the path markers, the AUV would reorient to face downward if it is currently at higher depths. Descending is an alternative option, although a higher vantage point angled downwards may provide a more advantageous field of view to find the path markers. Once located, the offset of the detected path from the AUV is determined by the corresponding point cloud data, which is placed in the global feature map. This allows our AUV to position itself over the path and traverse it. The heading at the end of the path is then followed in order to find the task the path leads to.

In a scenario where the vehicle overshoots the path when searching for it, the position of the preceding task in the global feature map is used to mark which end of the path is the beginning (such that the AUV wouldn't traverse the path backwards).

3) "Shoot Craps"

For this task, the AUV needs to touch the dice buoys - with preference to having the last two dice touched add up to seven or eleven. The four available dice for the task comprises one, two, five, and six dot variants. Adding the last two to eleven would mean touching the five and six last, while adding to seven yields the options of using five and two, or six and one.

The strategy for our AUV to accomplish this task is as follows: First, image data is processed and light gray rectangles of a certain scale are identified. This is done using OpenCV to color filter for high luminosity values, and then applying contour detection to find objects of a certain scale. Second, these areas of interest are analyzed further by detecting dark circles or blobs in their interior. This is done using OpenCV blob detection.

The next step has two cases. The first case involves the second step being successful in determining the dot count of each buoy. In this scenario, the AUV would plot a path to touch the five dot buoy and then the six dot buoy (or vice versa). In contrast, the second case is concerned with having detected buoys but not being able to determine their dot count. In this scenario, the AUV would attempt to reposition itself and attempt to repeat the process. If this is unsuccessful, then the AUV would plot a path to touch the two buoys with higher interior dark to light ratios, as this would correspond with five and six. This is assuming that the AUV is viewing the buoys from a head-on angle of one of the sides and not viewing the corner of the dice.

4) "Buy Gold Chip"

This task, while providing a high amount of points in and of itself, also has the potential to give additional points to subsequent tasks. To accomplish this task, our strategy is to use image feature matching to detect the distinctive '$' icon on the push plate. Once detected, the corresponding point cloud data is used to map the offset from the vehicle and a course to the plate is generated. The gold golf ball is intended to be caught by a net which is extended and positioned such that it is aligned with the dispenser for the in-line version. The plate is pushed with the lower front bar of the AUV after the net is extended.

5) "Play Slots"

Our vehicle doesn't currently have operational torpedoes and thus our strategy for this particular task will be limited to pulling the lever or skipping this task entirely. If we are able to add functional torpedoes (which meet the torpedo specifications), then our strategy for this task would involve detecting the outline of the rectangular slots with OpenCV color filtering and canny edge detection. Once identified, our AUV would align itself (more specifically, the torpedo pod) with the target and launch a torpedo.

6) "Play Roulette"

For this task, the AUV must find the roulette wheel and place chips into any of its six bins. Placing it into the bin of the called color or the green bin scores more points, although our strategy is focused on ensuring that the chip is successfully placed in a bin rather than risking an unsuccessful chip deployment. To this end, our AUV would position itself directly above the roulette wheel and release the

chip from its holder (controlled via a servo motor). Furthermore, determining when the vehicle is positioned directly above the roulette wheel is done through analyzing the visual offset of the detected portions of the wheel, and adjusting the vehicle's position to centralize the image. This imagery is obtained from the AUV's downward facing low light camera.

Both this task and the next task ('cash in') involve dispensing chips. Thus, if both tasks are intended to be done in one run, the blue chips may be allocated such that one blue chip is used in roulette and the other used in the 'cash in' task. Each initial blue chip has a latch which is opened by a servo, so the two blue chips can be deployed independently. Rationing of the initial chips and any obtained chips is further discussed in the overall strategy section.

7) "Cash in"

For this task, there are four red and four green golf balls in bins on the bottom of the arena that need to be collected. The second portion of this task is to place chips into either the registers at the surface or into the registers two feet below the surface (subsurface registers).

First, placing any collected gold chips into the subsurface gold register is a priority. This is because it would gain maximum points and have the same associated difficulty of placing chips into any other register. Furthermore, the net used to collect additional chips would at this point still have the gold chip (if successfully collected) and not doing this step first may introduce a risk of losing the collected gold chip when attempting to collect red/green chips. To place the gold chip in the register, our AUV would color filter for yellow/gold and select the point cloud data for the positional offset of the bin. Once in range, the AUV would turn the net upside-down over the register and let the golf ball sink into it.

Next, the chips in the red bin on the bottom would be targeted, as the red chips have a subsurface register and the net of our AUV is less effective at the surface. Our strategy for collecting them is to use the net and have the vehicle move forward at a slight downward angle and then tilt upwards in order to perform a scooping motion. If successful, the chips would be placed in the red register using the same strategy as with the gold chips.

B. Overall Strategy

Our current overall strategy is pass through the gate, follow the first path, and do the 'shoot craps' task. Then, an attempt at acquiring the first gold chip would be made. If unsuccessful for a preset timeout period, or the gold chip is acquired, our AUV would move on and follow the second orange path. However, our AUV would not go to the 'play slots' task, pending the status of our torpedo development.

The next stage would be to find the next task using visual feature matching. If both the 'roulette wheel' and the 'cash in' tasks are found, and adequate time is left, both would be attempted. Our blue chip rationing strategy is currently in favor of using them both for roulette, as the wheel is a larger target than the cash in registers. In the case that only one of them is found, both the blue chips would be used on that task. In either case, when the time is sufficiently low or if neither of the latter two tasks are found, our AUV would plot a course to the breaching area and surface there, ending that run.

V. Design Creativity

Throughout the design process, our team has had to find creative solutions to problems/tasks that all felt very new – with this being our first year competing. Our decision to use a stereo camera (ZED) was based on our successful experience using it for ground vehicles and researching their usability for underwater robotic applications. The design of the vehicle was influenced by this as it made it necessary to use a larger enclosure in order to fit the camera and the accompanying TK1. In this regard, utilizing the ZED with a TK1 and Python proved challenging in and of itself because the 32bit TK1 wasn't compatible with the opensource python-wrapper that Stereolabs provided. Thus, an older version of the ZED SDK was used with C++ and bridged with Python by making a custom Cython bridge.

With the majority of the team being software oriented, the bulk of the customization was on the programming side. The code consists primarily of custom Python code, with a particular focus on developing it to be loosely coupled such that changes could easily be made throughout our development process. On the hardware side, the focus was on achieving the buoyancy and mobility requirements and less so on novelties. Due to this being our first

year competing, this scheme may completely reverse next year with the software largely being reusable and the mechanical side being the primary customization element.

VI. Experimental Results

A. Simulations

For testing our AUV, we designed a simulator that takes preset scenario inputs and runs them through the main program. The simulator also generates (and updates) a visualization of the current global obstacle and feature maps. In addition, the current path the vehicle intends to traverse is also added and updated on the visualization.

Furthermore, the ZED SDK can record and save data in SVO format, which contains not only the RGB video but also 3D point clouds for each frame and the camera's position relative to the starting point for movement tracking. These SVO files were recorded during trial runs of the vehicle in a pool, allowing the feature matching and other image processing techniques to be refined and tested without needing the vehicle to be constantly in a pool (which could introduce some logistical issues).

The location information in each frame of the SVO file allowed us to simulate the path taken by the robot on any recorded trial. This also aided in further refining the image processing as it provided feedback on how the obstacle and feature maps would be populated based on the image selection.

B. Vehicle Testing

The vehicle was tested in a pool using both manual drive testing and autonomous feature testing. In particular, the pool testing provided the means to adjust the buoyancy and mobility of the vehicle as even refined paper calculations aren't always accurate in determining these attributes.

Certain task components and props were assembled and used for testing the AUV's ability to accomplish them. Some of the more difficult and complicated task assemblies (roulette wheel/slots) are still being evaluated for a means to physically represent them. These will likely be far more simplistic versions of the real task assemblies and will consist of only the portions of the task that the vehicle would need to be tested for.

Acknowledgments

The Hydro Bison team would like to thank the NASA North Dakota Space Grant Consortium, and the NDSU Foundation & Alumni Association for providing materials used in this project.

References

[1] S. W. Moore, H. Bohm, and V. Jensen, *Underwater Robotics: Science, Design & Fabrication*. Marine Advanced Technology Education (MATE) Center, 2010.

[2] Z. Zeng, K. Sammut, L. Lian, F. He, A. Lammas, and Y. Tang, "A comparison of optimization techniques for AUV path planning in environments with ocean currents," *Rob. Auton. Syst.*, vol. 82, pp. 61–72, Aug. 2016.

APPENDIX A: COMPONENT SPECIFICATIONS

Component	Vendor	Model/Type	Specs	Cost
Buoyancy Control	Blue Robotics	Static Ballast/Buoyancy Foam	-	~$50
Frame	-	HDPE/Aluminum	-	~$200
Waterproof Housing	Blue Robotics	8” and 4” Acrylic Enclosures	1/2” thick	$580
Waterproof Connectors	Blue Robotics	Cable Penetrators	6mm and 8mm	~$50
Thrusters	Blue Robotics	T200	Quantity: 8	$1300
Motor Control	ArduPilot	Pixhawk 2.4.8	ArduSub	$60
Actuators	Power HD	LW-20MG Digital Servo	Quantity: 5	$150
Propellers	Blue Robotics	T200 Propellers	4 CW, 4 CCW	-
Battery	Venom	3s Li-Po	5000mAH	$50
Converter	eboot	LM2596 DC-DC	12v to 5v	$5
CPU	Nvidia	TK1	192 Cuda Cores	$220
Compass	-	Pixhawk (internal)	LSM303D	-
IMU	-	Pixhawk (internal)	MPU6000	-
Camera 1	Stereolabs	ZED	Stereo Camera	$450
Camera 2	Blue Robotics	Low Light HD USB Camera	(2MP, 1080p)	$90
Programming Language 1	-	Python (2.7)	-	-
Programming Language 2	-	C++	-	-
Open source software 1	-	ROS	EKF, tf2	-
Open source software 2	-	OpenCV	-	-
Team size	-	6	-	-
HW/SW expertise ratio	-	HW: 2, SW: 4	-	-
Testing time: simulation	-	~50 hours	-	-
Testing time: in-water	-	~20 hours	-	-