Illinois Autonomous Underwater Vehicle Design and Implementation of Enigma

Shubhankar Agarwal, Adrian Brandemuehl, Saleh Ahmed, Raimi Shah, Mohammad Saad, Tanay Vardhan, Krishna Dusad, Akshay Mishra, Wale Adeyinka, Harshit Agarwal, Jay Patel, Volodmyr Kindratenko and David Forsyth.

Abstract - IllinoisAUV is a research oriented team comprising 11 driven undergraduate students drawn from diverse backgrounds and disciplines, who aim to explore the applications of cutting-edge technology and engineering in building an autonomous underwater vehicle. AUVs have immense uses for industry, government, and science. This year, IllinoisAUV has designed, programmed and built, Enigma. With the goal of bringing and adapting a lot of the cutting edge research in Machine Learning and related fields to AUVs. With Enigma, the focus was on setting the right foundations such as using GPU computing for future work using Neural Networks. The team has also focused on keeping the costs as low as possible and instead rely on software to do many of the tasks that have traditionally been performed using costly specialized hardware, such as using cameras for localization, mapping and odometry instead of using DVL.

1 Introduction

Vehicles capable of understanding their environment and navigating without explicit human input are a major value to humanity. Recent progress in Machine Learning, Computer Vision, and Robotics has led to big advancements in the field of autonomous cars. This, however has not been the case for the advancement of other types of autonomous vehicles such as those that travel in air and water. Autonomous underwater vehicles (AUVs) are a subset of these vehicles which have great value in industry with uses such as exploration of oceans, surveying sea floors for the oil and gas sector, mine exploration for the military, and in research for discovering and studying underwater life.

RoboSub, organized by the Association for Unmanned Vehicle Systems International (AUVSI) Foundation and Office of Naval Research, is a competition for students that simulates tasks and challenges that an AUV would face in the real world. Every year a course is set with challenges that an underwater vehicle must overcome autonomously.

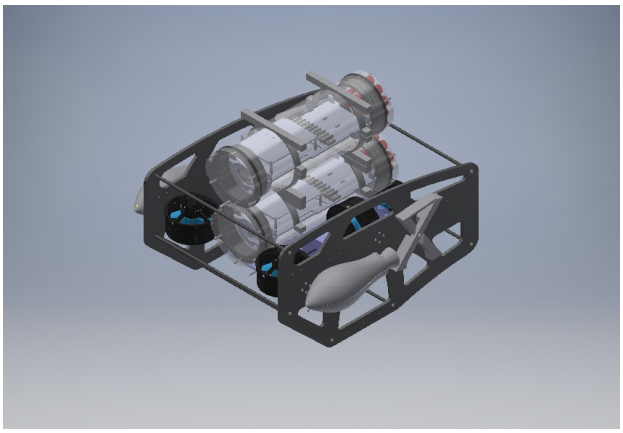
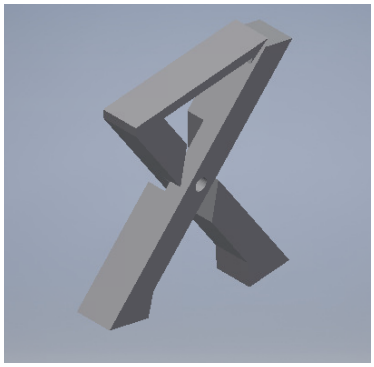


Figure 1: Enigma

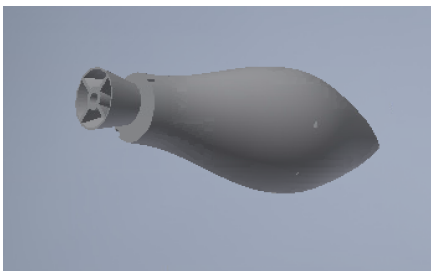
IllinoisAUV is a team of students from various departments at the University of Illinois at Urbana-Champaign. With the goal to adapt and incorporate both cutting edge research in various fields, and allowing interested students the opportunity to experiment, learn and research; IllinoisAUV has built Enigma, an AUV which uses a variety of new techniques to complete tasks involving vision, navigation, acoustic sensing, planning and object manipulation.

2 Design Overview

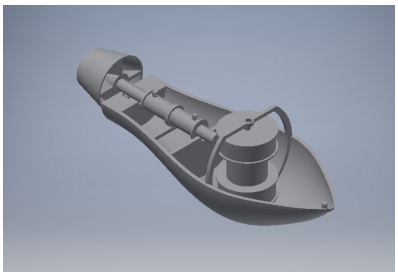
Enigma is Illinois AUV's first ever autonomous underwater vehicle(Figure 1). The main structure was purchased from Bluerobotics and has been modified for Robosub Competition. Enigma has six degrees of freedom and been gone through rigorous testing to get precise movement. Enigma was developed with a strong focus towards software which involves: eliminating the need of DVL, improving mapping(using SLAM [1] [3]) and localizing software frameworks, reinforcement learning [4] for battery optimized movement and using generative neural networks for developing better simulation softwares. Being a first year



(a) Bomb Dropper



(b) Torpedo



(c) Inside view of the torpedo

Figure 2: Bomb dropper and Torpedo

team, a lot of time was spent in developing the mechanical and electrical systems. Initial investment in mechanics and electronics helped us build a solid base for reliable and maintainable software.

Bluerobotics Bluerov 1 was chosen as the main structure for two main reasons: Our team was mostly experienced with software and bluerov 1 helped as accelerate our development cycle by 7 to 8 months. Although we bought the frame, additional mechanical changes were made to alter the bluerov1 for the Robosub competition. One way we did this was through the use of 3D printing which helped us expedite mechanical development and to experiment and implement various mechanical subparts.

One of the main considerations in the initial designs of Engima was to have a GPU onboard for faster image processing and running real-time machine learning algorithms. With 256 Cuda cores and the size of a Raspberry Pi2, the NVIDIA Jetson TX1 was a clear choice. The computing capabilties provided by Jetson TX1 opened many doors for us and the small size of Jetson TX1 also meant that we would not have to make major mechanical changes.

With major improvements in Artificial Intelligence in the past decade, this project has never been more interesting. Major investment was put into researching the robustness and application of modern Artificial Intelligence systems in underwater robotics.

Enigma's sensor suite consists of 3x SJ4000 Action Sports Cameras(2x front and 1 downward), Bluerobotics Bar30 High-Resolution Pressure Sensors and 3x Teledyne Reason TC 4013 Hydrophones.

3 Mechanical

3.1 Mechanical Structure

Enigma's mechanics were mainly developed on top of the Bluerov 1 frame. A second 4" wide pressure hull was added on top of the main pressure hull to hold the Enigma's compute. 3D printed custom brackets were designed and manufactured to hold the pressure hull on top of the first pressure hull. Due the constraints imposed by Bluerov 1 structure we decided to develop completely mechanical bomb droppers and torpedos. This drastically decreased the space requirements for torpedos and bomb droppers. Current implementations of bomb droppers and toredos operate without any external energy source.

3.2 Bomb Dropper

The bomb dropper is two parts, assembled like scissors(Figure 2a). On the top is a locking mechanism which opens and closes using the elasticity of the 3D print filament to our advantage. To lock, simply close the dropper until two pieces are latched together. To unlock, the latching piece must be pulled up before the two pieces can be pulled apart. The bottom of the dropper is where the bomb is held, using a spring-loaded nail to hold the bomb in place. The spring-loaded nail is also what pushes the two pieces apart when the latching piece is pulled up.

3.3 Torpedos

The torpedo is composed of many parts. The main parts are the shell of the torpedo, the input gear, the output gear, and the propeller fan(all three can be seen in Figure 2c). The shell of the torpedo is the case for the torpedo, inside of which the "motor" is held. The torpedo is set up like a wind-up toy. A rubber band is used to wind up the input gear, which, when unwinding, causes the output gear to turn. The output gear is attached to the same axle as the propeller fan, causing the whole torpedo to move through the water. The torpedo is wound up beforehand, and until it is ready to be fired it is suspended by locking the propeller fan in place. When ready to fire, the torpedo is pushed off the lock, allowing the propeller fan to rotate and move the torpedo forward. A complete implementation of Torpedo can be seen in Figure 2b.

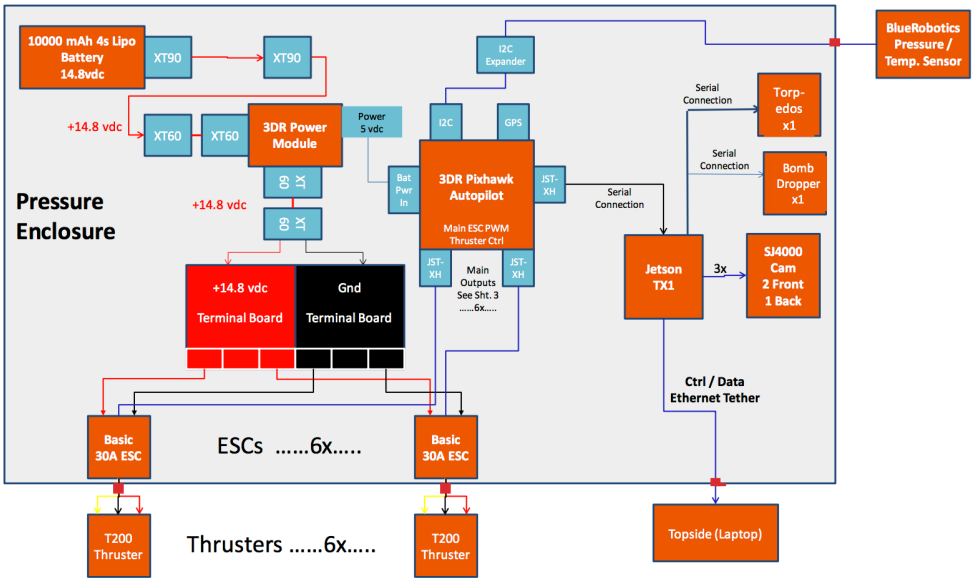


Figure 3: Electrical Block Diagram

4 Electrical Design

The design of the 2016-2017 AUV was created with ease of use and consistency in mind. The design is based on Bluerobotics BlueRov. The abundance of open source software and support brought us to conclude that the BlueROV suited our needs. It was further modified to make it suitable for our application. Block daigram of the design can be seen in Figure 3

4.1 BlueROV

The BlueROV kit consists of a frame, 6 BlueRobotics underwater T200 thrusters, and a 12" long, 4" wide watertight enclosure. The kit is easily assembled, and the simple electronics setup means that the BlueROV is fast to get from the box into the water.

4.2 Electronics

All electronics systems were COTS. From the Jetson TX1 to the motor speed controllers, we selected COTS parts for simplicity of implementation. Designing our own electronics would have been expensive and time consuming, since low volume manufacturing has low turnaround and high price. These limitations led us to use off the shelf hardware to meet our needs.

4.2.1 Power

The AUV has two 4S LiPo batteries, one for actuation and one for computing. The actuation battery first passes through a 3D Robotics power module for current and voltage measurement, as well as 5V regulation for the PIXHAWK autopilot. The Jetson TX1 is 12-19V tolerant, so no extra power conversion is needed for it.

4.2.2 Actuation

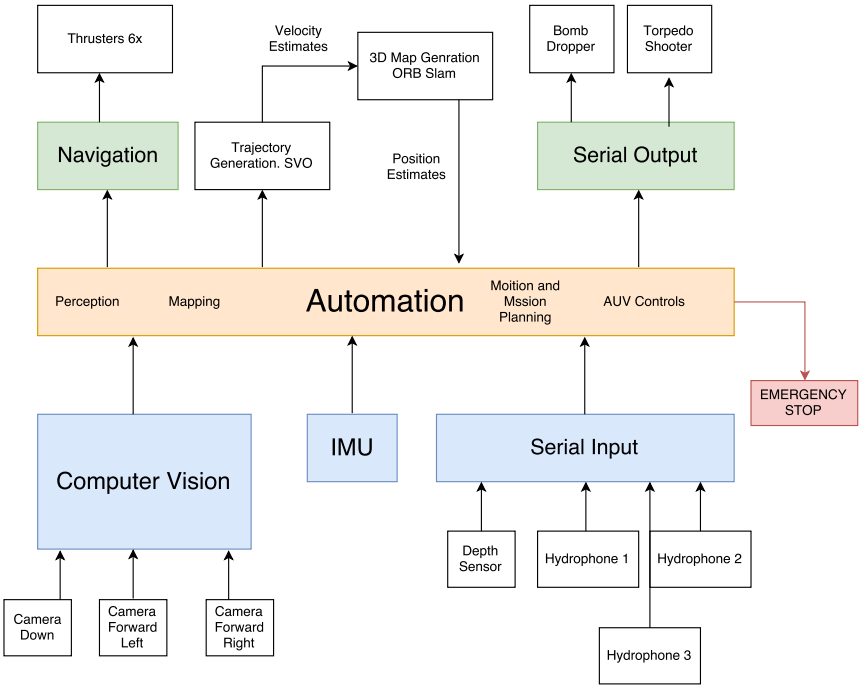
Six BlueRobotics T200 underwater thrusters provided motion control for our AUV. The T200 thrusters are brushless motors that are specifically designed for use underwater. Their inbuilt waterproofing and low pricepoint make them ideal for use on a small AUV such as our own. T200 thrusters are controlled through BlueESC's, which are the BlueRobotics rebranding of the Afro Electronic Speed Controllers. The motor outputs are controlled through a Pulse Width Modulated (PWM) signal generated from the PIXHAWK autopilot.

4.2.3 PIXHAWK Autopilot

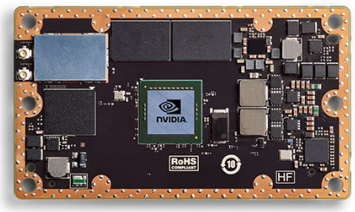
The PIXHAWK Autopilot is a flight controller initially designed for fixed wing Micro Air Vehicles (MAV) and multirotors. Though it is not specifically designed for underwater motion, open source projects have added the required functionality for AUV and ROV control. The PIXHAWK features a 3 axis accelerometer, 3 axis gyroscope, and 3 axis magnetometer. The PIXHAWK communicates with a companion computer over a UART connection via the MAVLink communication protocol.

4.2.4 Jetson TX1

The Jetson TX1(Figure 4b) is a compute module from Nvidia that features a powerful and low power CPU/GPU combo. Featuring 256 CUDA cores and quad-core 64-bit ARM processor, the compactness



(a) Software Design



(b) Jetson TX1

Figure 4: Software Design and Jetson TX1

of the Jetson makes it ideal for our vision processing application. Typically, Jetson TX1 modules are placed on large (6.7"x6.75") development board that is meant to take advantage of every possible application of the Jetson. Rather than attempt to use the development board, we opted to use one of the many carrier boards available, the ConnectTech Orbitty carrier board. The Orbitty board (3.425"x1.968") is small enough to fit into one of our watertight enclosures (4" diameter).

5 Software

Our software stack(Figure 4a) runs aboard the Jetson TX1, communicating with the Pixhawk PX4 stack via MavLink serial packets. Our software stack utilizes three main components: Robot Operating System (or ROS), our computer vision frameworks, and our navigation system.

5.1 Robot Operating System

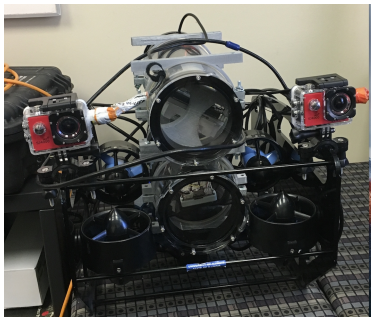
Robot Operating System (ROS) is a centralized communication system for robot software. It utilizes a publisher / subscriber model to transmit data and commands between different programs. For example, one program (say, an image processing framework) can publish whether it has seen a buoy in the water, and different navigation programs can pull from that and move our submarine in such a way to avoid or pick up the buoys.

We utilize several ROS packages in order to communicate and process the large amount of data being generated by our inputs. These ROS packages include MAVROS, usb_cam, image_proc, stereo_image_proc, as well as our own custom ROS packages, including a buoy detector and an image publisher.

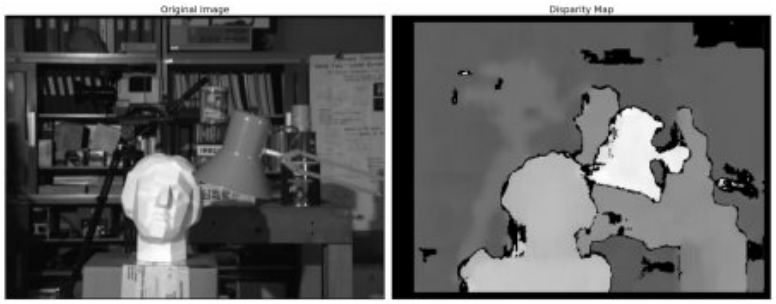
5.1.1 MAVROS

To interface with each of the submarine's engines, we utilize MAVROS onboard our TX1, an open-source ROS package which can send and receive MAVLink messages over serial communication to a PixHawk device. The PixHawk, as described in the Hardware section, has multiple sensors onboard, including an IMU and temperature sensor. It also contains a flight management unit which we can utilize to move our submarine. It becomes essential to interface with it at a higher level, and send commands / receive data to autonomously navigate our submarine.

To this end, MAVROS supports all of these functions and more. We can easily pull data by subscribing to any of the /mavros/ ROS topics, and control the individual thrusters by publishing to the /mavros/rc/override topic. To control the submarine, we first utilized an Xbox controller to see how the submarine reacted to different values on each of the RC channels. The RC channels are expressed as an array of values corresponding to a direction, with thresholds determining button presses, and ranges corresponding to how much we want to push the thrusters in a specific direction. Utilizing that, we can send the



(a) Stereo Camera Implementation



(b) An example of Disparity Image

Figure 5: Stereo Camera and Software Implementation

Figure 6: Stereo camera calibration

approximate speed and direction we want to go to, a task more specifically covered in section 1.4: Navigation.

One important thing to note is that MAVROS here is specially configured for a submarine operation. Originally, MAVROS is utilized on quadcopters/UAVs, and so has different topics optimized for that, including /global_position/gps_fix. On our submarine, we disable these topics as to not utilize any extra memory. We keep MAVROS as submarine operation is relatively similar to quadcopter operation, just in different modes (water versus air).

5.1.2 usb_cam, image_proc and stereo_image_proc

Our three cameras are connected to the Jetson via USB, and so require a USB driver to send images to the board. This driver takes shape in the form of usb_cam, which publishes to a /camera/image_raw topic. These images are raw sensor values - they have not been transformed into real image values yet. This is where the image_proc and stereo_image_proc packages come in. image_proc takes in a raw image and transforms it into a color image, while also removing any distortions that the camera may add to the image. (See the section on calibration for more details about the distortion removal.) image_proc also outputs greyscale and compressed images for faster transport and use.

stereo_image_proc, on the other hand, does the work of image_proc, but with two cameras instead of one. stereo_image_proc takes in raw camera feeds, along with a calibration file, and undistorts/rectifies the camera streams, along with calculating a disparity map. This disparity map acts as a depth map of sorts, and allows us to build up a small point cloud of the view right in front of us. For more details about stereo calibration, see the section under Computer Vision.

5.2 Computer Vision

Cameras are cheap and powerful ways to obtain a representation of the environment around us. As computer vision algorithms have progressed in the last decade, the ability to put it on a robot and run in real-time have become achievable. We use OpenCV for our vision, mainly to provide an estimation of our position and velocity, as well as an object detector to determine where to go next.

5.2.1 Calibration

Calibration here refers to undistorting an image such that straight lines in the real world are straight lines in the image. An example of this can be found in the Figure 6. We see that the checkerboard's straight lines are not straight in an image from the camera. Knowing the intrinsic and extrinsic parameters of the camera, however, we can undistort and rectify the image such that straight lines are straight. Most of the undistortion takes place in image_proc and stereo_image_proc.

To obtain the intrinsic and extrinsic calibration, we utilize the ROS camera_calibrator node. This takes in the raw image streams with a checkerboard pattern in the feed. We move the checkerboard around at different angles to get an estimate of the focal length, camera center, skew (otherwise called intrinsic parameters) and the distortion coefficients (extrinsic param-

eters). The calibration routines picks up the squares on the checkerboard and computes the parameters to undistort the actual image such that the lines are straight. It is essential to therefore move the checkerboard at as many angles and places in the image to obtain a good estimate of the parameters.

5.2.2 Stereo

We utilize two SJ4000 action cameras(red cameras in Figure 5a) in order to provide a stereo estimate using stereo_image_proc. Stereo refers to estimating a depth map based on two cameras. Similarly to human vision, using 2 cameras allows us to estimate how far away things are, or at least help us associate a depth value to objects. This is useful for the submarine for multiple reasons, as we can build up a point cloud for the environment around us, as well as do object detection and collision avoidance.

Our cameras are mounted with a 29cm baseline between the cameras, allowing for a 83.5 degree horizontal field of view and an error of 12.3 cm at a 5 meter distance, according to [1]. For underwater operation, where we typically cannot see more than a few meters anyways, this is a very good result. An example of the output of the depth estimation algorithm can be seen in Figure 5b.

The actual estimation itself is done in stereo_image_proc, This ROS package combines two image streams, takes in a calibration file, and outputs a disparity map like in Figure 2. It also outputs a point cloud estimate relative to the current pose (position + orientation) of the robot. Utilizing these two pieces of information, we can infer distances, as well as associate pixels in our cameras with a depth value for collision avoidance.

5.2.3 Localization and Odometry

Odometry refers to the estimation of speed values based on sensor inputs. For example, a car odometer estimates the number of miles traveled on a car based on the number of times the wheels have spun. Similarly to that, we use an odometry system called Semi-Direct Visual Odometry (SVO) [2] to provide velocity and speed estimates. Unlike most VO/SLAM systems, SVO utilizes a downward-facing camera in order to estimate the position and velocity. SVO is a semi-direct method, meaning it minimizes photometric error of several already extracted features between frames to obtain a more precise depth estimate. Since it relies on both feature-based and direct methods, SVO is able to handle high-frequency textures and fast movements. In addition, it runs much faster than real-time, allowing us to continually re-estimate the pose with the smallest change. Finally, the open-source version can be integrated with ROS, allowing us to quickly and easily bring together the estimation into our system.

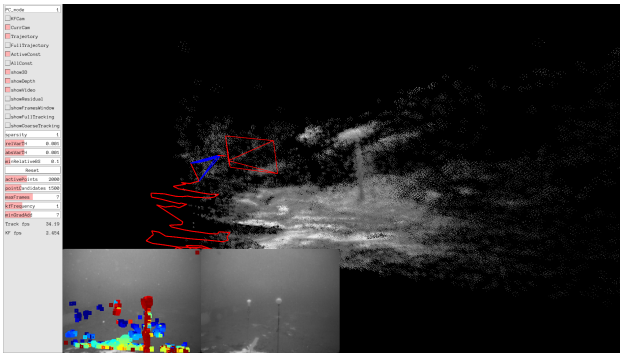


Figure 7: Trajectory generation from SVO

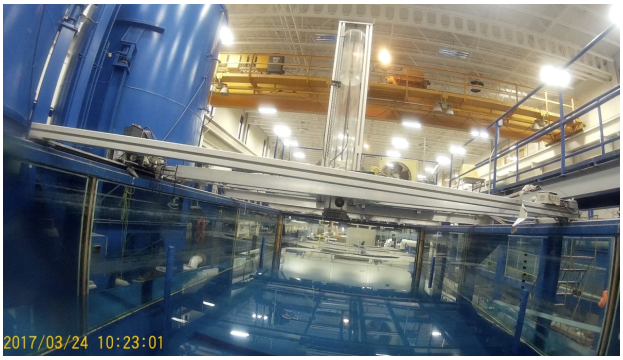


Figure 8: Hydrosystems Lab Testing Facility

SVO [2] was originally designed (and tested with) quadcopters, and we soon realized that we can adapt it to submarine operation as well. Based on previous videos and experiences, we saw that the floor of the pool we were testing was advantageous to a downward-facing camera, as we could extract many features off of it. As a result, we put a third SJ4000 action camera on the bottom of the submarine, allowing us to extract pose estimates using SVO. Since SVO is fairly light computationally (running well above 30 frames per second) adding it to our system does not put too much of a burden on our hardware. An example of trajectory generation:7

6 Acknowledgement

Illinois AUV would like to specially thank our faculty advisors and mentors, Professor David Forsyth and Professor Volodmyr Kindratenko, who has given us direction and financial support through our many hours of development and testing. Additional thanks goes out to Van Te Chow Hydrosystems Lab at University of Illinois for providing pool and other testing facility(8). Finally, we would like to thank our sponsors: Microsoft, Nvidia, UIUC SORF, ACM UIUC and Wittenstein SE.

References

[1] J. Engel, T. Schöps, and D. Cremers. LSD-SLAM: Large-scale direct monocular SLAM. In *European Conference on Computer Vision (ECCV)*, September 2014.

[2] Christian Forster, Matia Pizzoli, and Davide Scaramuzza. Svo: Fast semi-direct monocular visual odometry. In *IEEE International Conference on Robotics and Automation*. IEEE, 2014.

[3] Raul Mur-Artal, Jose Maria Martinez Montiel, and Juan D Tardos. Orb-slam: a versatile and accurate monocular slam system. *IEEE Transactions on Robotics*, 2015.

[4] Junku Yuh. Learning control for underwater robotic vehicles. *IEEE Control Systems*, 14(2), 1994.