TALLINN UNIVERSITY OF TECHNOLOGY

School of Information Technologies

Lishan Fernando 184586 IASM

DEVELOPMENT OF A DISTRIBUTED SOFTWARE FRAMEWORK FOR A SMALL-SCALE AUTONOMOUS UNDERWATER VEHICLE

Master's thesis

Supervisor: Christian Meurer
MSc.
Roza Gkliva
MSc.

Tallinn 2020

TALLINNA TEHNIKAÜLIKOOL

Infotehnoloogia teaduskond

Lishan Fernando 184586 IASM

HAJUSTARKVARA RAAMISTIKU ARENDAMINE VÄIKSELE AUTONOOMSELE ALLVEESÕIDUKILE

Magistritöö

Juhendaja: Christian Meurer
MSc.
Roza Gkliva
MSc.

Tallinn 2020

Author’s declaration of originality

I hereby certify that I am the sole author of this thesis. All the used materials, references to the literature and the work of others have been referred to. This thesis has not been presented for examination anywhere else.

Author: Lishan Fernando

[15-05-2020]

Abstract

Constructing, testing, and operating an autonomous underwater vehicle (AUV) is a costly operation. Therefore, the scientific community is looking to miniaturise AUVs using modern low-cost single board computers and microcontrollers. µ-CAT v2, developed by the Centre for Biorobotics in Tallinn University of Technology, is such a solution, that intends to provide an easy to use and cost-effective AUV for researchers and students.

This thesis focuses on the software development of the upgraded version of the µ-CAT AUV. The µ-CAT v2 is equipped with a single board computer and a microcontroller at its core. The distributed computing middleware called Robot Operating System (ROS) is utilised to develop the proposed software solution. The proposed software solution for µ-CAT v2 provides a testbed for researchers who are working with its bigger counterpart U-CAT.

Evaluating an underwater robot is a time-consuming and complex process. To address those problems, the hardware-in-the-loop concept is used for the evaluation of the proposed software solution, in simulation. Parallel to the hardware-in-the-loop simulation, web-based visual aids are developed to provide means and methods to collect information about the robot while it is executing. Achieving reliable communication among distributed computational units and preserving the responsiveness of the robot is another challenge addressed in this work. As a solution, checksum mechanisms and task allocation between real time and non-real time computational units are proposed. Accessing the microcontroller while it is connected to the single-board computer and programming makes the development process more challenging. Solutions are provided to improve the usability of the µ-CAT v2 in such scenarios.

This thesis is written in English and is 32 pages long, including 7 chapters, 19 figures and 6 tables.

List of abbreviations and terms

TUT	Tallinn University of Technology
UV	Underwater Vehicle
AUV	Autonomous Underwater Vehicle
ROV	Remotely Operated underwater Vehicle
ROS	Robot Operating System
LCM	Lightweight Communications and Marshalling
MOOS	Mission Oriented Operating Suite
UDP	User Datagram Protocol
TCP	Transmission Control Protocol
U-CAT	Underwater Curious Archaeology Robot
IMU	Inertial Measurement Unit
OS	Operating System
V1	Version 1
V2	Version 2
HiL	Hardware in the Loop

Table of contents

List of figures

List of tables

1 Introduction

Underwater environments such as oceans, rivers, and lakes cover 71% of the Earth's surface [1]. Those natural environments play a vital role in the Earth's ecosystem. Nevertheless, most of this resourceful environment is unexplored. However, explorations of this environment have become a necessity due to the high demand for energy, raw material, and food requirements. Maintenance of underwater technical equipment such as communication cables, tunnels, and oil pipelines is another area that requires the capability of underwater operations [2], [3]. The scientific curiosity of humans is another fact that drives the community to explore these environments rapidly [4] [5].

The extreme environmental conditions that humans cannot overcome, possible life threats, and cost factors make sending humans to explore the underwater environments impractical [6]. As a solution, manned submersibles have been used. These submersibles addressed some of the issues mentioned above and introduced some new problems such as cost, energy requirements, and limitations of operational time. Also, these manned submersibles did not eliminate the possible life threats. To overcome these challenges, Remotely Operated underwater Vehicles (ROVs) are used. The first recorded ROV was developed in 1953 by Dimitri Rebikoff, and the Cable-Controlled Underwater Research Vehicle (CURV) was created by the US Navy in 1961 [7], [8]. However, ROVs also had some limitations, such as requirements of a human operator, high cost due to necessary specialized equipment, and limited range of operation.

As a solution to the limitations of ROVs, Autonomous Underwater Vehicles (AUV) were introduced. Early stages of the development of the Underwater Vehicles (UVs) were driven by the military. Currently, the scientific community and engineers from fields such as the oil and gas industry are involved in this area [9], [10]. In recent years companies started to produce AUVs, and most of these robots have an immense cost. General areas of AUV application include commercial exploration such as seabed mapping, pipeline and cable inspection, scientific research, such as oceanographic studies, environmental

monitoring, and marine archaeology, as well as marine salvage and debris removal, and defence [11] [12] [13].

Most of the developed AUVs contain high power computational units and are equipped with cameras, sonar, lighting systems, inertial navigation systems, and other sensors depending on the application [14]. Most of these AUVs are larger and cannot be tested in a research lab. Therefore, dedicated space and human resources are required for their evaluation during development. As a result, AUV technology requires inexpensive novel solutions that can be realized in low budget projects. These robots are expected to be tested in a research lab.

The technological advancement of recent years has produced low cost, power-efficient single board computers, and microcontrollers [15] [16]. Compared to the conventional computational units, smaller size, less power requirement, ease of programming, and community support are some advantages of using these technologies. Because of these advantages, the scientific community started to use this technology on their projects.

The Centre for Biorobotics in Tallinn University of Technology developed a bio-inspired AUV using Raspberry Pi Zero W and an Arduino mini 05, which is an upgrade to their previous µ-CAT. Low cost, low power consumption, and small size give the advantage to the AUV to be more agile, lightweight, and easy to handle. Although this system is a definite upgrade over the previous version, it has computational limitations. Distributed computing provides the advantage of task dedication and extra computational power, yet it adds the challenge of achieving reliable communication between computing units. This thesis presents the development of a software framework for µ-CAT.

Main contributions of this thesis contain:

1. Analysis of the existing miniaturized AUVs and their software frameworks.
2. Development of a distributed software framework for an AUV using a low-cost single board computer and microcontroller.
3. Validate reliability and usability aspects of the proposed software framework.

This thesis consists of 7 chapters. Chapter 1 contains objective of the thesis, motivation behind the research and the scope. Chapter 2 of this thesis provides an insight of the state of the art in miniaturized AUVs and software frameworks used. Proposed software

designs are introduced in detail in chapter 3. Chapter 4 provides an overview of the methodology used to verify the work. The results of the work can be found in chapter 5, and chapter 6 includes the discussion of the results, future work and limitations. Chapter 7 includes the conclusion of the work.

2 Literature review

Autonomous underwater vehicles are increasingly used in a wide variety of marine sciences and commercial applications. Miniaturizing autonomous underwater vehicles is a trending subcategory in the underwater vehicle development domain. This chapter presents the state-of-the-art in small scale AUV technology and robot development frameworks. The first part of this chapter describes a selected number of AUVs in the field, which are smaller in size and have limited computational capabilities. The second part of this chapter discusses the software frameworks used for the development of AUVs. A comparison of the software architectures of the frameworks, communication mechanisms, supported programming languages, and platforms is provided.

2.1 AUV Technology

D. Richard Blidberg describes AUV as an underwater system that requires no communication from outside during the mission, yet it is powered and controlled while it achieves its given task [17]. There are different types of AUVs developed in academic and commercial fields, and [18] [19] describes some of them, such as Autosub6000 AUV, ABE AUV and Hugin AUV. Most of these AUVs are comparatively large and have powerful processing units.

Smaller AUVs are developed for various purposes, with the primary goal of reducing the cost and increasing usability. Osterloh et al [20], Amory et al [21], and Kalantar et al [22] propose using small size AUVs in a swarm for underwater monitoring. Wick et al [23] and Underwood et al [24] propose using smaller scale AUVs for military applications. Bonin-Font et al [25] propose using smaller AUVs for 3D reconstruction in shallow waters. Watson et al [26] suggest using AUVs for investigations in nuclear storage ponds. Chao et al [27] and Bennett et al [28] propose to employ AUVs for research purposes.

Pressure sensors, temperature sensors, and IMUs are some commonly used sensors in the AUVs discussed above. However, some AUVs carry specialized payloads. SEMBIO carries a WIFI module, an Xbee module, electronic speed controllers, and cameras [21].

Bluefin SandShark AUV focuses on military operations, and environmental monitoring and contains GPS sensors, Attitude Heading, and Reference System [24]. Riptide AUV consists of various sensors, such as cameras, hydrophone array, inertial navigation system, and pressure sensor [28].

AUVs discussed in this chapter have employed different computational units in their control systems. Riptide AUV uses a Core i7-4790S CPU processor with an MX87QD motherboard [28]. HippoCampus uses a Pixhawk as its control unit [29]. Mange et al [30] used a Raspberry pi with a Pixhawk for the control and navigation of the Proton AUV and upgraded to a Xu4 due to its higher framerate compared to the Raspberry Pi. SEMBIO AUV uses a Raspberry Pi at its core [21].

With the attempts taken to miniaturize the AUVs, space in the hull for the control unit becomes smaller. This prevents the use of large and powerful computational units in AUVs. As a solution to this, researchers use smaller computational units. However, this adds computational limitations to the AUV. As a result, small scale AUVs are equipped with multiple small computational units. Furthermore, for real-time computations, microcontrollers were introduced to AUV projects. μAUV2 uses Xilinx Virtex 4 FPGA with five custom made stackable PCBs [31]. AUVx is the upgraded version of μAUV2. It contains a Spartan3 XC3S1000 FPGA and a Raspberry Pi Zero in its computational stack [32]. In project MK VI a custom-built embedded system is used, which consists of three microcontrollers [33]. The Lancelet AUV uses an AVR32 microcontroller AT32UC3B0256 in its main control board along with a separate control unit for sensor control and power control [27]. The control system in USNA-l used the Rabbit 2000 microprocessor and microchip PIC computer [23].

AUVs discussed in this section differ in size, computational capability, actuators, and objective. However, multiple computational units are employed to serve the higher computational requirements, and microcontrollers are employed for the real-time operations. Design of the software architecture is segregated to logical layers and subcomponents, which provides the ability to place the software components in the appropriate computational unit. Table 1 provides a comparison of different AUV hardware discussed above.

Table 1. Summary of small-scale AUVs. '-'denotes no information was provided in literature.

AUV	**Application**	**Computational Units**	**Sensors**	**Software Framework**
Bluefin SandShark [24]	Military, Monitoring	-	GPS, Attitude Heading and Reference System (AHRS), Altimeter, Pressure sensor, Temperature sensor	ROS, Huxley
Riptide [28]	Research	Core i7-4790S CPU processor, MX87QD motherboard	Cameras, Hydrophone array, Inertial navigation system, Pressure sensor	ROS
Proton [30]	Monitoring	Raspberry Pi 3, Odroid Xu4, Pixhawk	-	-
MONSUN II [20]	Monitoring	Blackfin processor	Camera, IMU, Attitude and Heading Reference System (AHRS), Infrared distance sensors, Pressure sensor, Temperature sensor	ROS
SEMBIO [21]	Monitoring	Raspberry PI, Xbee module, WIFI module	Pressure sensor, Compass, Temperature sensor, IMU, Electronic speed controllers, Pixy camera, Raspberry pi camera	ROS
Lancelet [27]	Research	AVR32 microcontroller AT32UC3B0256	IMU, Pressure sensor	-
Fugu-C [25]	Research	PC104 board based on an Atom N450 CPU	Stereo Cameras, IMU, Water leak detectors, Pressure sensor	ROS
USNA-l [23]	Military	Rabbit 2000 microprocessor, Microchip PIC computer	Electronic Compass, Gyroscope, Accelerometers, Pressure sensor	-
Serafina [22]	Monitoring	Motorola MPC 555, Dedicated real-time µcontrollers	Acceleration, Compass, Pressure sensor, Sonar	-
HippoCampus [29]	Monitoring	Pixhawk	Gyroscope, Accelerometer, Magnetometer, Pressure sensor	NuttX, PX4
µAUV2 [31]	Research	Xilinx Virtex 4 FPGA, 5 Stackable PCB	Camera, IMU, Pressure sensor	-

AUV	Application	Computational Units	Sensors	Software Framework
AUVx [32]	Research	Spartan3 XC3S1000 FPGA, Pi Zero,	Camera, IMU, Pressure sensor, Thruster and Pump Speed Sensors, Thruster Attitude Sensor	ROS
MK VI [33]	Research	Custom-built embedded system including three microcontrollers	Pressure sensor, Gyroscope, Digital compass, Acoustic positioning	-

2.2 Robot development frameworks

Robots are complex systems that consist of many components, that need to communicate with each other to work as a system to achieve its goal. Achieving this, while developing a sophisticated robot, is challenging. To minimize the effort on developing a framework from scratch and to focus on the goal of the AUV projects, development frameworks are used. Hentout et. al [34] discussed some of the key factors to consider when selecting a development framework for robotics. These frameworks provide modularity, communication mechanisms, and predeveloped software algorithms for faster development. Some of the features to consider while selecting a development framework are ease of installation, supported operating systems, supported programming languages, reliability of the communication protocol, clear and complete documentation, licensing, and community support.

LCM, MOOS, Huxley, CARMEN, ROS use a publish-subscribe model for inter-process communication [35], [36] [37] [38], [39] [40], [41]. In the Player/Stage project, a socket programming for inter-component communication is used [42]. As a result of this socket interface of Player/Stage, developers can use any programming language which supports socket programming to develop the external client program. The DUNE framework uses message passing for its inter-process communication [43]. LCM supports languages such as C, C++, C#, Java, Lua, MATLAB, and Python. MOOS is a C++ based cross-platform middleware and provides interfaces in Java and Python. CARMEN is written in C language and supports Java. CARMEN runs in Linux systems [38], [39]. DUNE is a C++ based framework that provides operating-system and architecture independence. Huxley is another framework written in C++. Huxley is a proprietary software product by BlueFin Robotics and runs on Linux systems. LCM, MOOS, CARMEN, DUNE, Player/Stage and ROS run on Linux systems and DUNE, Player/Stage and LCM support windows. Table 2 provides a summary of the frameworks discussed in this chapter.

In the present, the currently available software frameworks are focused towards different robotic development problems, and hence their main focus differs. Communication mechanisms, inter-process dependability, predeveloped libraries, language support, are factors to consider before selecting a framework. Robotic development frameworks provide leverage over developing software from the beginning.

Table 2. Summary of robotic software frameworks. '-'denotes no information was provided in literature. .

Framework	**License**	**Supported OS**	**Programming Languages**	**Communication Method**
CARMEN [39] [38]	GPL	Linux, Windows	C, C++, Java, Python	pub/sub
MOOS [37]	GPL	Linux	C++, Matlab	pub/sub
ROS [40], [41]	BSD	Linux, Windows	C++, Python, Octave, Lisp	pub/sub
LCM [35], [36]	-	Linux,Windows,OS X	C, C++, C#, Go, Java, Lua, MATLAB, Python	pub/sub
Player/Stage [42], [44]	GPL	Linux,Windows	C, C++, Tcl, Python, Java, LISP	Socket
Huxley [45]	Proprietary	Linux	C++	pub/sub
DUNE [43]	Commercial / EUPL v.1.1	Linux,Windows,OS X	C++	Message passing

3 Introduction to µ-CAT

µ-CAT has been developed in the Centre for Biorobotics in TUT, which focuses on the development of bio-inspired sensing and locomotion technologies. This chapter describes the µ-CAT project, its history, and its current development. Researchers in the Centre for Biorobotics have developed two robots, called U-CAT and µ-CAT. Their locomotion is inspired by the fin movement of a turtle, and they contain a small array of sensors to collect data from their environment. These types of robots can be used to observe underwater archaeological sites and marine ecosystems. The fin-based actuation enables those robots to explore the environment with less disturbances to their surroundings and the local sea life [46], compared to the classical propeller-based underwater robots such as ECA Group – AUV A18 [47] and Iver3 - L3Harris [48]. U-CAT was developed in the ARROWS project to be used for archaeological explorations [46]. The robot has a Linux based computer at its core and different sensors, including a camera and pressure sensor [49].

Figure 1. µ-CAT v1 (Centre for Biorobotics TUT).

Programming a robot is a complex and time-consuming task. Additionally, testing an autonomous mobile robot adds extra complications. Developers faced similar challenges while developing software for U-CAT. In order to evaluate novel developments, U-CAT needs to be taken to the sea or a lake, which requires logistic support and a dedicated crew. This operational requirement will add extra costs to the project. One solution for the identified problem will be to use a small scaled version of U-CAT in lab environment.

This will allow developers to test their software algorithms in a similar, yet easy to use platform in the lab environment.

µ-CAT v1 (Figure 1) is the miniature version of the U-CAT firstly developed for a science exhibition. It has been used, as an educational tool in the past and is planned to evolve into a scientific tool (µ-CAT v2) to test research hypotheses and algorithms for U-CAT. The initial design of µ-CAT features an Arduino mini 05 as the control unit, as well as a custom-designed motherboard for the power supply, motor drivers, and interfaces for communication buses. It has a pressure sensor for depth estimation, two light sensors that detect a specific frequency of light via tone detectors, an IMU for orientation feedback, and an LED to indicate the status of the robot. Limitations of this AUV are identified as minimal number of sensors and low computational power. µ-CAT v1 reached its limits regarding computational capabilities and room for peripheral components. The absence of a camera limits its environmental perception capabilities as well.

µ-CAT v2 will address these limitations while preserving µ-CAT v1 qualities such as the smaller size and low manufacturing cost. This thesis discusses the software design and development of the µ-CAT v2 and the next section of the chapter discusses the proposed software architecture for the µ-CAT v2.

3.1 Software Architecture

This section outlines the software architecture of the µ-CAT v2 AUV. µ-CAT v2 operates in two modes. The first mode is the default mode (Figure 2), where µ-CAT v2 operates autonomously in an underwater environment. In the second mode, HiL is used to develop and test novel algorithms without the necessity to put the robot in the water (Figure 3).

The new configuration enables distributed computing, which provides the robot with more processing power and grants the ability of task dedication when it comes to the software design. Another advantage is that the software framework of U-CAT can be replicated to some extent with µ-CAT v2. This provides the ability to test some software components of U-CAT in a smaller scale robot in the real environment. The software architecture of the µ-CAT v2 robot is divided into a higher and a lower execution level.

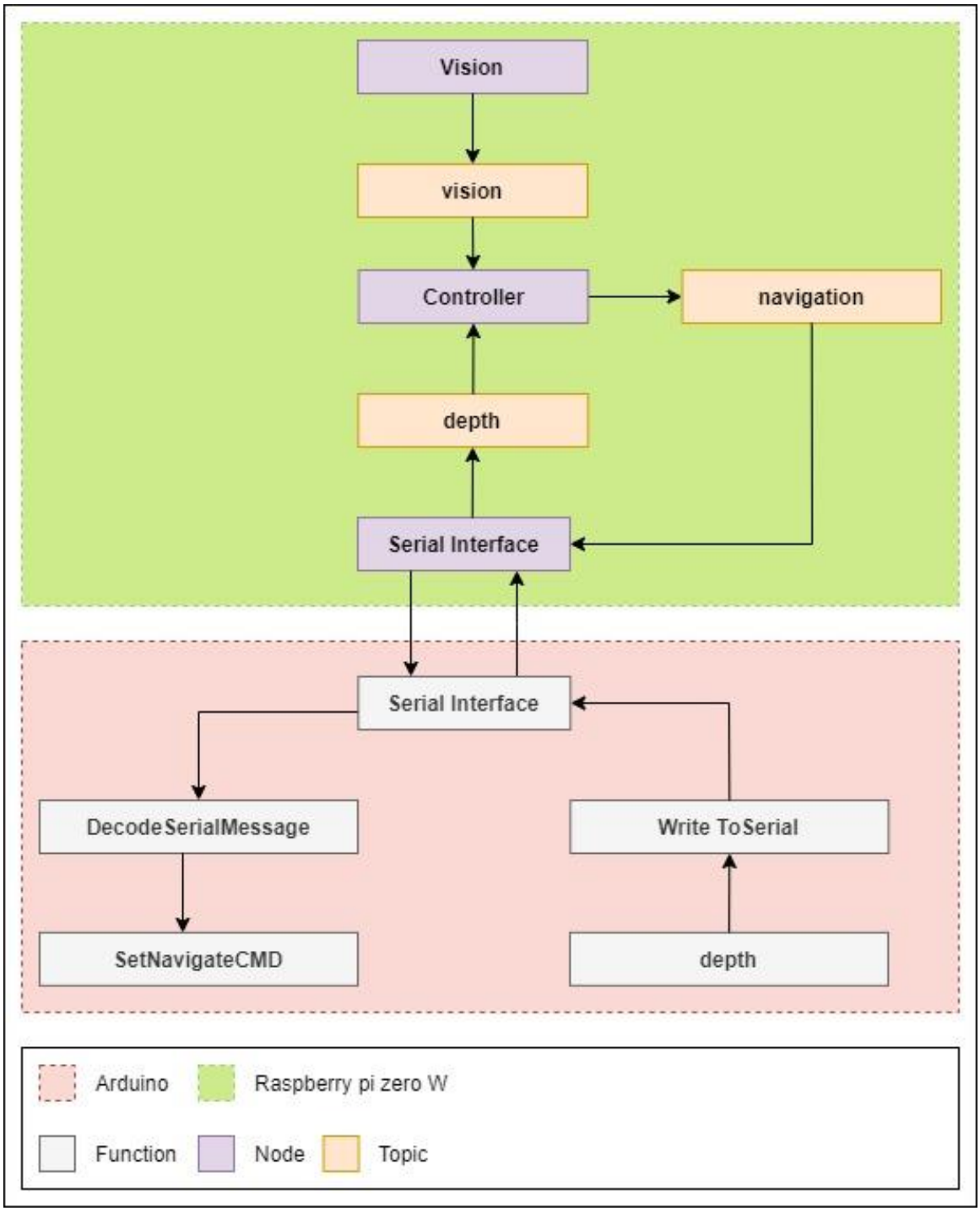


Figure 2. Overview of the proposed software framework for depth control of μ-CAT v2 in default mode.

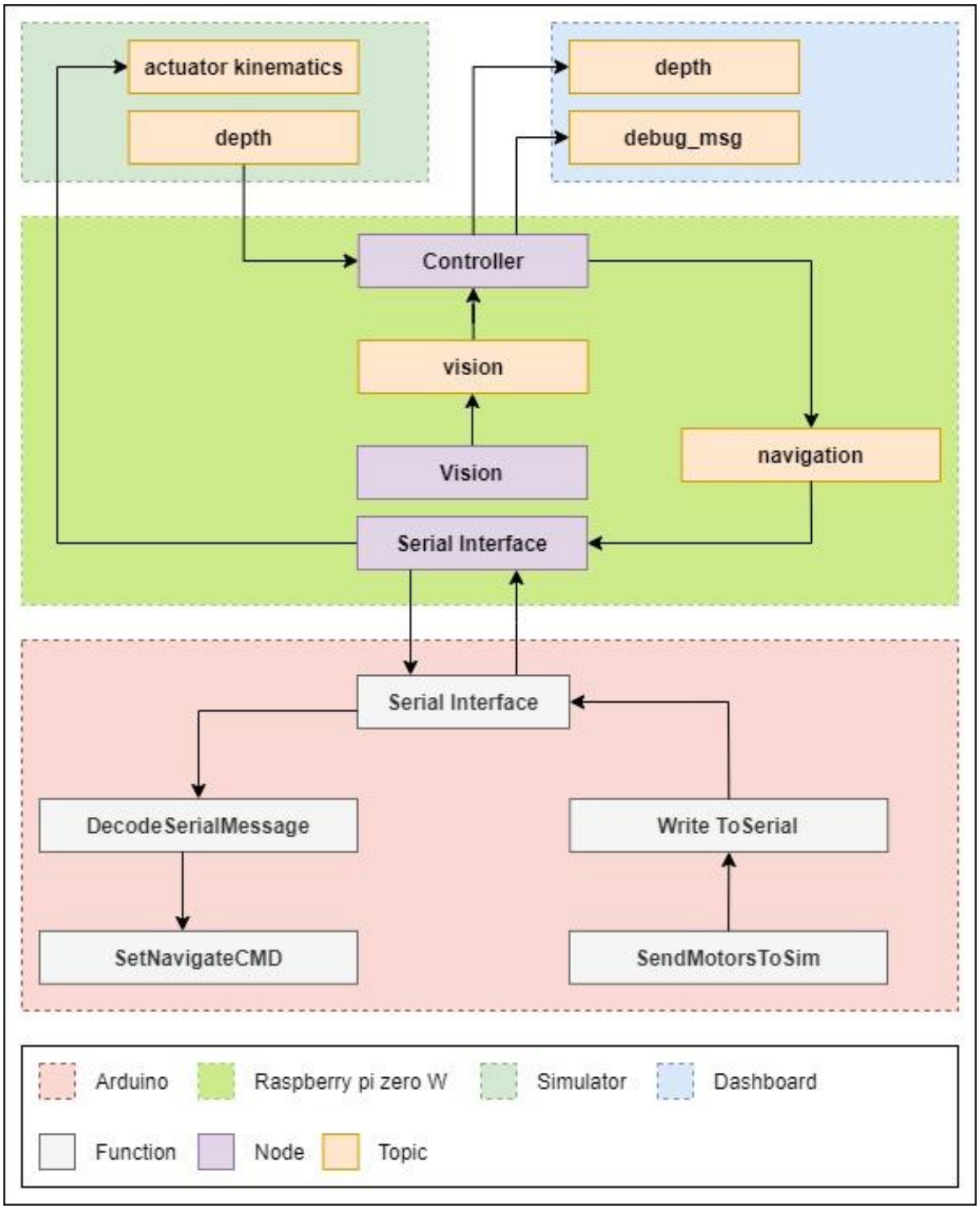


Figure 3. Overview of the proposed software framework for depth control of μ-CAT v2 in simulation mode. Simulator and Dashboard are connected over WIFI to Raspberry Pi Zero W.

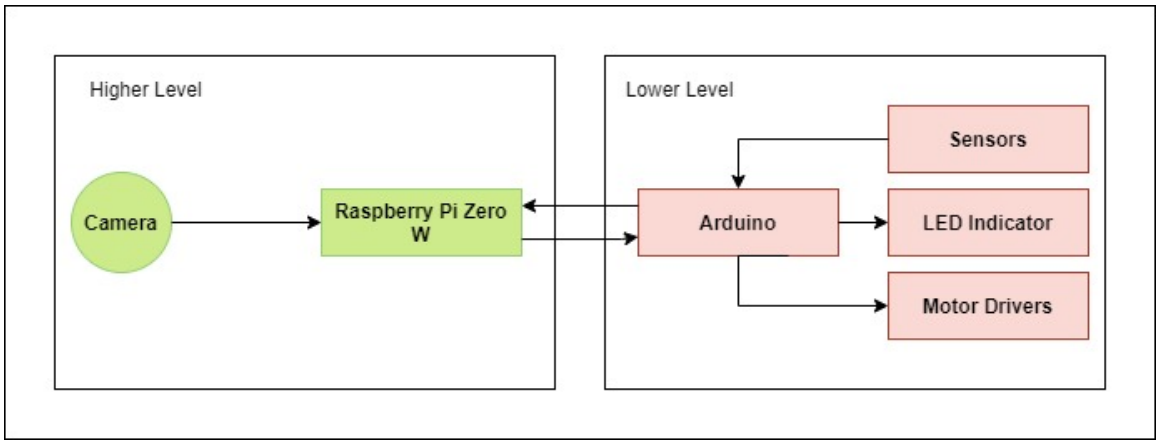


Figure 4. Simplified hardware overview of µ-CAT v2 in default mode. The distribution of the software execution layers is denoted by different colors.

3.1.1 Higher Level Execution and Software Architecture

Higher level execution of the robot is responsible for performing tasks that are not time-critical and that require more computational power. These tasks can be considered as the managerial task in the robot's operation, such as deciding the next move of the robot based on the sensor readings, analysing the images from the camera and maintaining the communication with lower level. This layer is executed in the Raspberry Pi Zero W and developed using the ROS framework (Figure 4).

Due to the compatibility with U-CAT AUV, higher level of modularity, pub-sub communication architecture and the wide variety of learning resources, ROS is used as the development framework of the µ-CAT v2 AUV in this work.

As described in Figure 2 the higher-level execution layer can be decomposed into three ROS nodes: a control node, a serial communication interface, and a vision node. The control node is considered as the main node of the system. The vision and serial communication interface nodes are considered as supporting nodes. The control node is subscribed to the topics published by supporting nodes, which will be used in the process of generating the control outputs. The control outputs include the directional information that AUV needs to perform movements as seen in Figure 5. The vision node analyses the images captured from Raspberry Pi camera and publishes visual features to the vision topic. The serial communication interface handles the data transmission between Arduino and the control nodes.

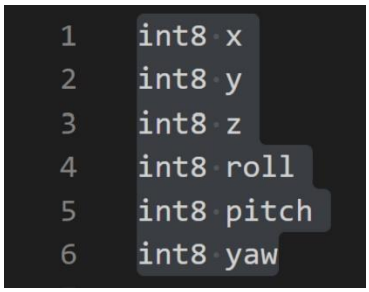


Figure 5. Definition of the control outputs.

3.1.2 Lower Level Execution and Software Architecture

In this configuration, the Arduino mini 05 acts as the lower-level execution layer and performs time-critical tasks as presented in Figure 4. Some of its tasks are to read and process the sensor values such as pressure sensors and light sensors. Afterwards, these values are sent to the higher-level controller.

The Arduino main loop runs at a frequency of 100Hz and is responsible for executing navigation commands. The microcontroller receives this data via serial communication. This data contains control outputs from higher execution level. The received control output is converted to actuator kinematics, which are forwarded to the motor drivers by the lower execution layer as seen in Figure 2. Low-level software, such as device drivers, are reused from the previous version of µ-CAT.

3.2 Simulator

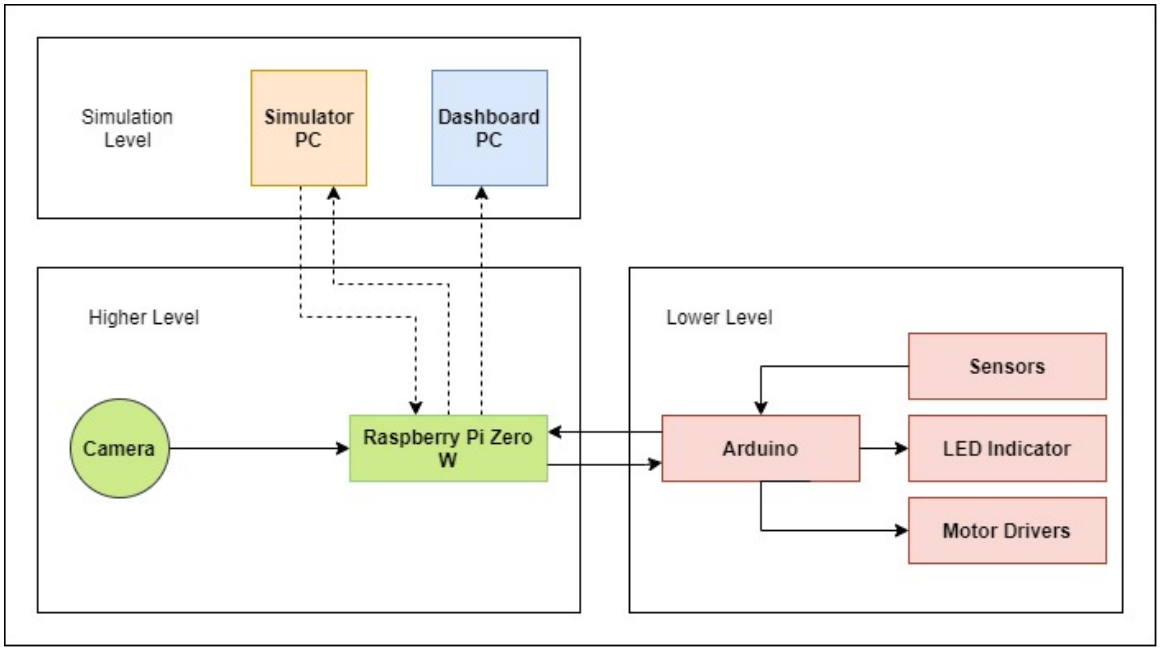


Figure 6. Simplified hardware overview of µ-CAT v2 in simulation mode.Solid lined arrows show the hardware connection between different electrical components, and dashed lines show the connection between computers over WIFI.

Developing an underwater robot and testing the robot in its actual environment is a complex process. Due to these complexities, robotic projects use simulators for evaluations. According to Shannon (1975) [50] simulation is "*the process of designing a model of a real system and conducting experiments with this model for the purpose either of understanding the behaviour of the system or of evaluating various strategies (within limits imposed by a criterion or set of criteria) for the operation of the system*." Therefore, simulations can add benefit to development of the project to determine the behaviour of the system without performing in actual environment. Underwater Simulator (UWSim) is such a solution used in µ-CAT version 2 development.

UWSim is an open-source underwater simulator, written in C++ and OpenGL. This program provides the capability to add robots to underwater scenes conveniently and implements sensor simulations. UWSim provides interfaces to external programs through ROS for the control purposes of the simulated robot [48].

The computational power of Raspberry Pi Zero W was inadequate to run the UWSim. Hence, a distributed architecture of ROS is used to run UWSim of µ-CAT v2 in a separate computer as presented in Figure 8. As seen in Figure 6, this includes running the simulation along with the µ-CAT v2 hardware.

Lower cost, flexibility, repeatability, and efficiency are the key factors to use the HiL and this concept was used in flight simulations, missile guiding systems, and in the automotive industry [51] [52] [53]. In this approach, hardware components are used with real-time simulated components and data in a control loop. The need for testing the overall system, reproducibility of test scenarios, and the ability to test the µ-CAT v2 without deploying to the real environment are the major reasons that HiL is used in the µ-CAT v2 project.

In the following, the necessary configuration steps to run UWSim are listed:

1. Launch ROS master in the Raspberry Pi Zero W.
2. Export the ROS Master URI in the computer which has the UWSim Program.
3. Export the URI of the computer which has the UWSim program as ROS IP.
4. Launch the UWSim program.

This configuration information can be placed in ~/.bashrc file (Figure 7).

```
export ROS_MASTER_URI=http://raspi:11311
export ROS_IP=192.168.0.105
```

Figure 7. ROS master URI export configuration in ~/.bashrc file of the simulation computer.

raspi is the host address of the Raspberry Pi 0 W which runs the ROS master. 192.168.0.105 is the IP of the simulator.

This configuration allows to access the simulated data via ROS topics in UWSim. In this thesis, the depth topic and vision topics are used by the control node in the Raspberry Pi Zero W. The depth topic provides the current depth of the simulated µ-CAT v2. Control node uses this depth data in its bang-bang controller to determine the next move of the µ-CAT v2. Once the next movement is calculated, this data is sent to the Arduino mini 05. Lower execution layer in Arduino mini 05 calculates the motor kinematic values. In the simulation mode, this motor kinematic data is sent back to the higher execution level via serial interface. Afterwards, the serial interface publishes motor commands to the actuator kinematic topic. UWSim uses these values to update its internal motor kinematics to simulate the behaviour of the robot.

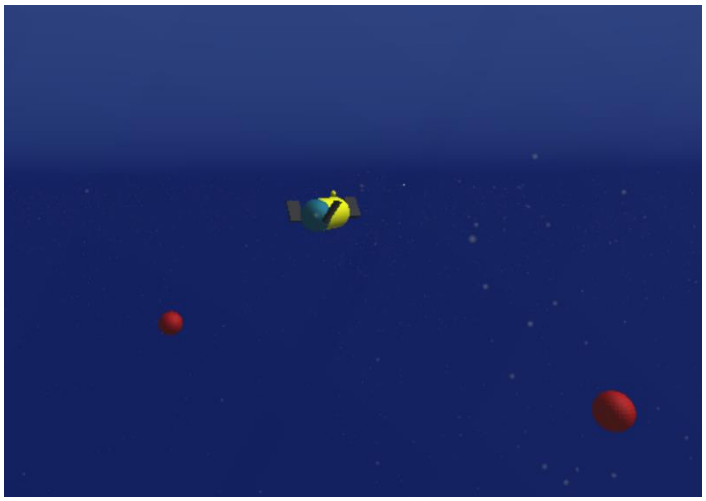


Figure 8. The µ-CAT during simulation with UWSim.

3.3 Dashboard

The serial console is the most common tool used in the community for debugging the Arduino solutions, yet it has its own drawbacks. The current µ-CAT v2 hardware configuration adds additional difficulty to debug the Arduino by preventing the usage of the serial console, since the serial communication channel of the Arduino is already utilized for the communication between higher and lower execution levels.

As a solution, this work suggests a method to pass the debug messages over the serial communication to Raspberry Pi Zero W, and from the Raspberry Pi Zero W, the debug messages can be published to a ROS topic. The next chapter (chapter 4) will explain the data structure used in detail.

When a debug message arrives along with a data packet, the Raspberry Pi Zero W will identify the debug message using the packet type identifier. Debug message will be extracted from the serial data packet and will be published to the debug message topic. As presented in Figure 3, a computer that runs rosbridge package will provide access to the ROS topics. The Standard ROS JavaScript Library (roslibjs) is used to create an informative dashboard solution which is subscribed to the ROS topics. The Dashboard developed in this thesis visualizes the IMU data, pressure data, debug messages, test results, and depth changes. This dashboard is used in simulation mode only. With this

dashboard component, developers get extra visibility inside the Arduino, and it is possible to customize the functionality of the dashboard as per the needs of the developers.

Non-ROS programs can access the ROS functionalities in a ROS based system via the json API provided by rosbridge [54]. The roslibjs is a JavaScript library which allows the development of web solutions for ROS [55]. In this work, rosbridge and roslibjs are employed with developing the dashboard solution.

4 Methodology

This thesis aims to achieve a distributed software framework for the AUV named µ-CAT v2 using ROS framework. A number of tests were carried out to understand the communication characteristics of the hardware. Possible limitations are identified as per the results of the experiments and the test results, as well as, suggested solutions are discussed in the communication section of this chapter. The vision and control sections describe the development done to for the framework and the final product is evaluated using the HiL method. The evaluation section of this chapter provides further details about this topic.

4.1 Communication

In the proposed configuration, Raspberry Pi Zero W and Arduino mini 05 are exchanging data via the serial channel. Raspberry Pi Zero W sends control outputs to the Arduino mini 05, and Arduino mini 05 sends sensor data and debug information back. Corrupted data from either party can affect the execution of the robot. Hence, it is vital to prevent the execution of the wrong function or parameter. As a safeguard, an error detection mechanism is proposed for the communication systems of the µ-CAT v2.

A data structure as described in Figure 9 is suggested for the information transmission between Raspberry Pi Zero W and Arduino. This structure is called a data packet in this work. Raspberry Pi Zero W and Arduino mini 05 are equipped with software components to validate the data packets based on the introduced structure. For each data packet type, as presented in Table 3, there is a corresponding function that can identify the packet type, extract related data and execute corresponding functions.

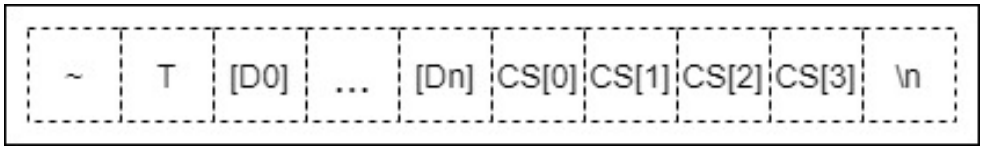


Figure 9. Data packet structure of the proposed software framework for µ-CAT.“~” denotes the start of the packet. “T” denotes the type of the data, and [D0] to [Dn] contain the data in byte format. CS [0] to CS [3] contain the checksum, and “\n” denotes the end of the data packet.

Reliability of the communications is an essential part of µ-CAT v2. Tests are carried out to figure out the behaviour of the serial communication between Raspberry Pi zero W and Arduino mini 05. Another aspect of the tests is to find the optimal parameters for the communication between Arduino mini 05 and Raspberry Pi Zero W.

Table 3. Data packet types used in the communication system.

Command	**Value**	**Description**
PKT_START	~	Denote the start of the data packet
PKT_END	\n	Denoted the end of the data packet
STOP_EXE	0x00	Start lower execution layer
START_EXE	0x01	Stop lower execution layer
NAVIGATION_PKT	0x02	Data packet with navigation instructions
IMU_PKT	0x03	Data packet with IMU data
DEPTH_PKT	0x04	Data packet with depth data
BEACON_PKT	0x05	Data packet with light beacons data
MOTOR_PKT	0x06	Data packet with motor data
DEBUG_PKT	0x09	Data packet with debug details

4.1.1 Testing the communication: a simple test

A simple test was performed to find the influence of the time delay between serial writes of Arduino mini 05 and Raspberry Pi Zero W. The experiment was intended to find the relation of the time gap between serial write operations, and the correctness of the communication.

In this test, a sequence of numbers was written to the serial channel with different time delays. Time delay was achieved using the python time module. A data packet was written to the serial channel and a time delay was added using the time module. Afterwards, Arduino mini 05 retrieved numbers and it observed for the correct sequence. Once when the test ended, the results were returned to the Raspberry Pi Zero W and collected results are presented in Table 5. Numbers were sent in two bytes format.

```
[
    {
        "id": 1,
        "name": "9600 baud rate - simple count",
        "ino_path": "/micro_cat_test/arduino/test_6/test_6.ino",
        "delay": [0.0, 0.5, 1.0],
        "baud_rate": [1200, 2400, 4800, 9600,115200],
        "from_val": 1,
        "to_val": 101,
        "expected": 100,
        "arduinodelay": [0.0, 0.1, 0.5, 0.75, 1.0]
    }
]
```

Figure 10. Example of a test configuration used in the advanced test.

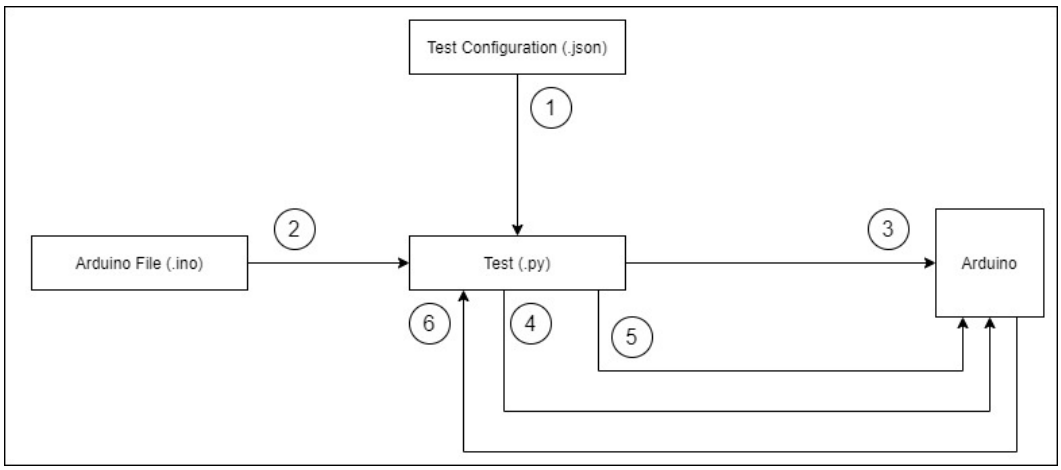


Figure 11. General overview of the communication test. Step 1 – Read the test configuration file. Step 2 – Find the corresponding Arduino sketch. Step 3 – Modify the parameters from the test configuration file in the Arduino sketch and compile and upload the to the Arduino. Step 4 – Perform the test. Step 5 – Request test results. Step 6 – Receive test results.

This thesis introduced a separate ROS package for the communication testing. This package contained a python script, template Arduino sketch and json files for test configuration as shown in Figure 10. This Python file contained a ROS node and the json file was read to find the configurations for the test. Arduino sketch was opened by the ROS node and modified the baud rate and the delay parameters. Afterwards, ROS node compiled the Arduino sketch and uploaded to the Arduino mini 05. Test data was sent via serial to Arduino, once when the ROS node requested for the results. Figure 11 presents a summary of the test procedure in µ-CAT v2 and 4 presents the data packet types used for the test mode.

4.1.2 Testing the communication: an advanced test

A data packet structure as seen in Figure 9 is used in this test. A sequence of numbers was converted as data and sent to the Arduino mini 05. From Arduino side data packet was evaluated to verify if there are any corruptions in data packet. Then the data from the packet is retrieved and evaluated for the correctness of the sequence. Test steps were repeated as described in the chapter 4.1.1. Users can observe the results using the dashboard presented in Figure 18.

4.1.3 Fault detection (reliability)

To achieve the reliability aspect of the communication, the proposed data structure in Figure 9 is used. An error detection mechanism was implemented in both higher and lower execution layers. Serial communication can populate erroneous data packets due to reasons such as power surge, bad software implementations, and buffer overflows. Fault detection mechanism in the proposed solution used a XOR checksum to detect errors in such instances.

Table 4. Data packet types used in the communication system when the robot is in testing mode.

Command	**Value**	**Description**
MSG_START	~	Denote the start of the data packet
MSG_END	\n	Denoted the end of the data packet
TEST_MSG	0x01	Denote a test data packet
REQUEST_RESULT	0x02	Request for the test results

4.2 Vision

A Raspberry Pi Camera v2 is used [56] as the hardware of the vision system of µ-CAT v2. The camera was planned to be placed in the front section of the µ-CAT v2. The Picamera library [57] was used as the driver software for accessing the camera module.

OpenCV is a library optimized for real-time computer vision applications. OpenCV version 2.4 was used as the image processing library for this module. This library provided a framework for computer vision applications which consisted of 2500 optimized algorithms [58].

For image processing, a separate node was created, and frames were analysed from the images taken from the camera. First, an image was acquired from the camera. Afterwards, HSV data was acquired for the colour separation. A colour mask was applied to check if the required colour was presented. The result was published to vision topic. In this work, a simple vision node was implemented as a proof of concept. Figure 12 describes the flow of the camera node and how it identifies a given color and publishes the name of the detected color to the topic. In the future, developers can implement more complex vision algorithms.

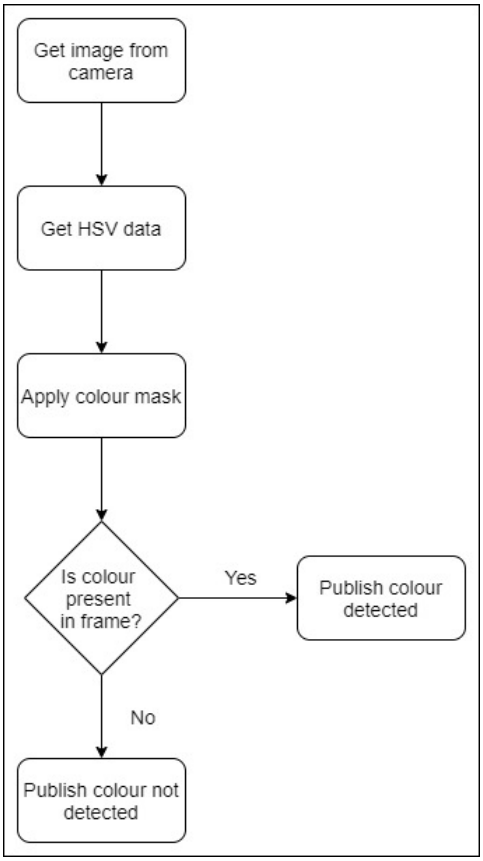


Figure 12. Steps of the color detection algorithm. Image processing includes converting a frame to HSV, applying a colour mask and checking if the requested colour is present in the frame. The result is published on a topic.

4.3 Control

This chapter discusses the decision-making process behind the proposed robot. The control node is responsible for generating control commands for the actuators in the robot. To generate control commands, camera, pressure, and light sensor data was collected. As

shown in Figure 13 the control node extracted data from the vision topic and the depth topic to determine the next move of the robot. The vision node detected a given colour and published the result. Based on the data from the vision node the desired depth of the robot is defined. The depth node provided the current depth of the robot. Based on the depth, the depth error was calculated, and then the control node formulated the navigation command to minimize the depth error.

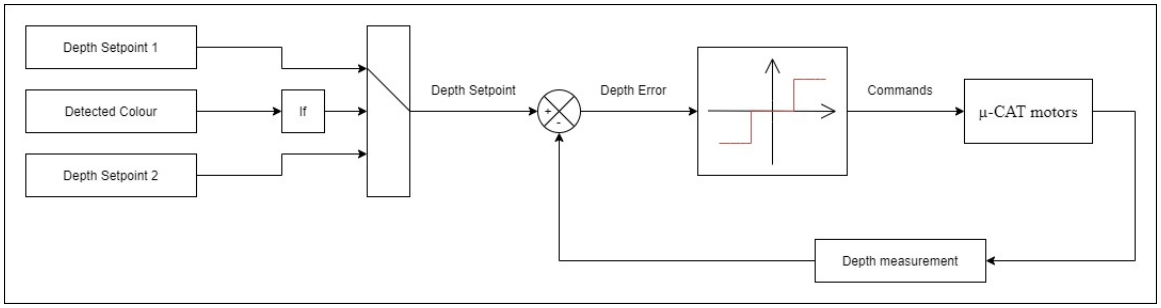


Figure 13. Bang-Bang depth control of the proposed robot.

4.4 Evaluation (Simulation)

In the evaluation phase, UWSim [48] was running in a separate Linux based computer. The simulator program simulated sensor data such as depth and light. These sensor data were published to ROS topics. The vision node was configured to run at a rate of 4HZ to detect colours in the RGB range of (120, 0, 70) - (255, 255, 10). Images were captured using the Raspberry Pi camera module. Vision node published the results to the vision topic as a simple string value. In the controller, there were two set points available as 0.5 m as the default set point and 1.0 m as the set point for the colour detected scenario. Controller node had a simple bang-bang controller without dead zones. The control node output the navigation command and, in the bang-bang controller scenario it manipulated the z value of the navigation command. The controller was configured to run at a rate of 10Hz.

5 Results

Communication between the Arduino mini 05 and Raspberry Pi Zero W is tested as described in section 4.1.1 and section 4.1.2. Table 5 presents the results of the simple communication test done by sending a sequence of numbers to the Arduino mini 05. Correlation between baud rate and success rate was tested and found that faster baud rates provide higher success rate in data transfer. Test is carried out with providing a different time delays between two serial write operations in Raspberry Pi Zero W. Results indicate having time delay between each serial write can improve the success rate.

Table 5. Results of sending a number as simple message using different Baud rates and time delays (ms). Delay (ms) denotes the delay in Raspberry Pi.

Baud rate	Delay (ms)	Success Rate
1200	0	0
1200	0.5	98
1200	1.0	98
2400	0	0
2400	0.5	0.98
2400	1.0	0.98
4800	0	0.94
4800	0.5	0.98
4800	1.0	0.98
9600	0	0.94
9600	0.5	0.98
9600	1.0	0.98
115200	1.0	0.94
115200	1.0	0.98
115200	1.0	0.98

Table 6 presents the collected results from the advance communication test. Compared to the simple communication test, this test adds delays to both Arduino mini 05 and Raspberry Pi Zero W sides. Test results conclude higher baud rate will deliver the data fast yet receiving end of the serial communication channel require some time to receive the data.

Table 6. Results of sending a structured data packet using different Baud rates and time delays (ms). Delay (A) denotes delay in the Arduino and Delay (R) denotes the delay in Raspberry Pi.

Baud rate 1200			**Baud rate 2400**			**Baud rate 4800**			**Baud rate 9600**			**Baud rate 115200**		
Delay (A)	**Delay (R)**	**Success Rate**	**Delay (A)**	**Delay (R)**	**Success Rate**	**Delay (A)**	**Delay (R)**	**Success Rate**	**Delay (A)**	**Delay (R)**	**Success Rate**	**Delay (A)**	**Delay (R)**	**Success Rate**
0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.88	0.00	0.00	0.35	0.00	0.00	0.13
0.00	500	0.00	0.00	500	0.00	0.00	500	0.00	0.00	0.50	0.59	0.00	500	0.92
0.00	1000	0.00	0.00	1000	0.00	0.00	1000	0.00	0.00	1.00	0.53	0.00	1000	0.96
0.10	0.00	0.00	0.10	0.00	0.00	0.10	0.00	0.88	0.10	0.00	0.35	0.10	0.00	0.14
0.10	500	0.00	0.10	500	0.00	0.10	500	0.16	0.10	500	0.59	0.10	500	0.96
0.10	1000	0.00	0.10	1000	0.00	0.10	1000	0.19	0.10	1000	0.55	0.10	1000	0.94
0.50	0.00	0.00	0.50	0.00	0.00	0.50	0.00	0.88	0.50	0.00	0.35	0.50	0.00	0.12
0.50	500	0.00	0.50	500	0.00	0.50	500	0.00	0.50	500	0.56	0.50	500	0.93
0.50	1000	0.00	0.50	1000	0.00	0.50	1000	0.16	0.50	1000	0.60	0.50	1000	0.96
0.75	0.00	0.00	0.75	0.00	0.00	0.75	0.00	0.88	0.75	0.00	0.35	0.75	0.00	0.16
0.75	500	0.00	0.75	500	0.00	0.75	500	0.18	0.75	500	0.60	0.75	500	0.94
0.75	1000	0.00	0.75	1000	0.00	0.75	1000	0.16	0.75	1000	0.59	0.75	1000	0.97
1.00	0.00	0.00	1.00	0.00	0.00	1.00	0.00	0.88	1.00	0.00	0.34	1.00	0.00	0.13
1.00	500	0.00	1.00	500	0.00	1.00	500	0.57	1.00	500	0.94	1.00	500	0.98
1.00	1000	0.00	1.00	1000	0.00	1.00	1000	0.56	1.00	1000	0.96	1.00	1000	0.98

Figure 14 presents the depth graph over the time using the bang-bang controller. Initial setpoint is set to 0.5 m and set point is change when the given colour is detected. Oscillation of the current depth and the depth error in the Figure 14 proves that the bang-bang controller is working expectedly. Also, Figure 14 proves lower execution layer receiving controller outputs from the higher execution layer. Step change of the controller outputs are present in the Figure 15. Calculated motor kinematics of the lower execution layer is transferred to the higher execution layer and then passed to the UWSim program. Figure 14 proves this as well by showing current depth oscillation.

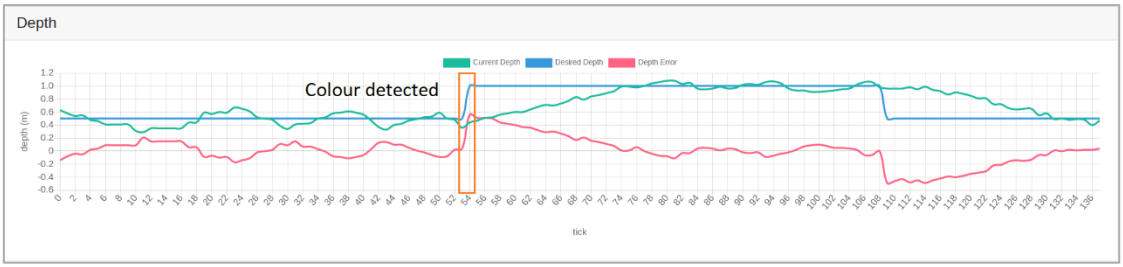


Figure 14. Performance of the Bang-Bang depth controller. Setpoints are selected based on the detected colour by the camera.

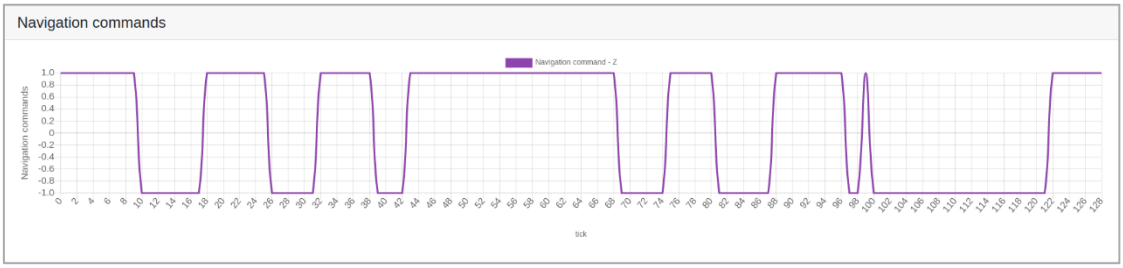


Figure 15. Bang-Bang controller output.

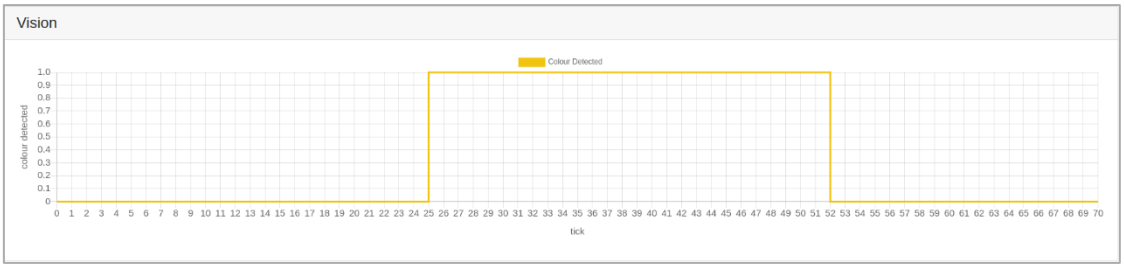
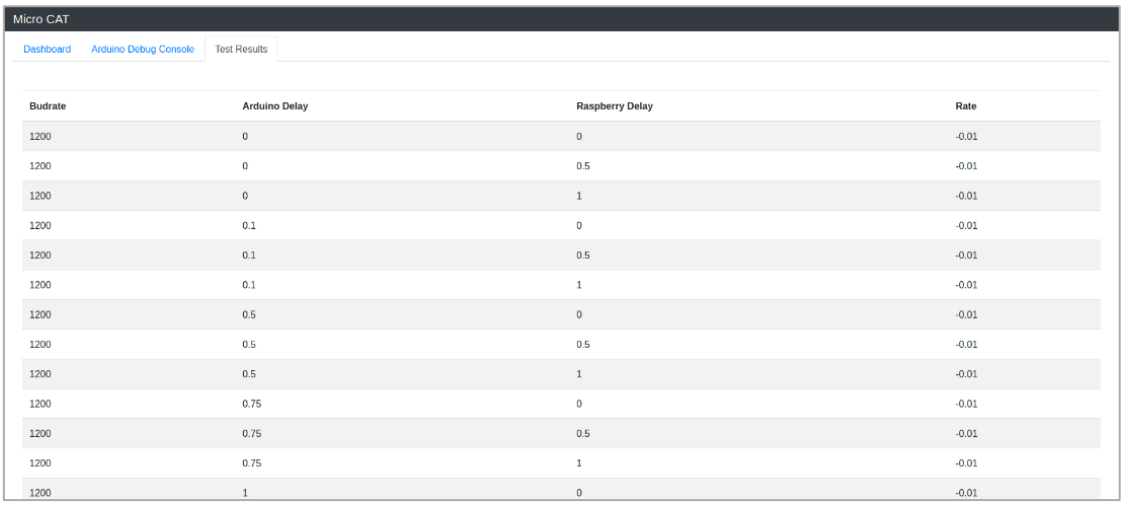


Figure 16. Colour detection of the raspberry pi camera. Node runs at a rate of 4Hz.

Figure 16 presents the step change of the colour detection. This figure proves that the vision node detected the given colour. Also, it provides evidence that control node is subscribed to the vision node and it is changing its set point according to the value of vision node. Depth error is also calculated according to this new set point.

Figure 17, Figure 18 and Figure 19 presents the dashboard solution developed for the µ-CAT v2. Graph in the Figure 17 provides a real time feedback about the depth values to the users. Figure 18 visualize the test result of the communication test performed in µ-CAT v2. Figure 19 presents the debug console that can be used to get insight of the lower level execution. This dashboard solution is highly customizable, and usage of the ROS bridge provide the access to the all ROS topics available. Compared to µ-CAT v1, version 2 provides considerable amount information in debug mode.

Figure 17. Dashboard visualizing the depth data, navigation commands and vision data of µ-CAT v2 in simulation.



Budrate	Arduino Delay	Raspberry Delay	Rate
1200	0	0	-0.01
1200	0	0.5	-0.01
1200	0	1	-0.01
1200	0.1	0	-0.01
1200	0.1	0.5	-0.01
1200	0.1	1	-0.01
1200	0.5	0	-0.01
1200	0.5	0.5	-0.01
1200	0.5	1	-0.01
1200	0.75	0	-0.01
1200	0.75	0.5	-0.01
1200	0.75	1	-0.01
1200	1	0	-0.01

Figure 18. Test results of the communication test observed through the dashboard.

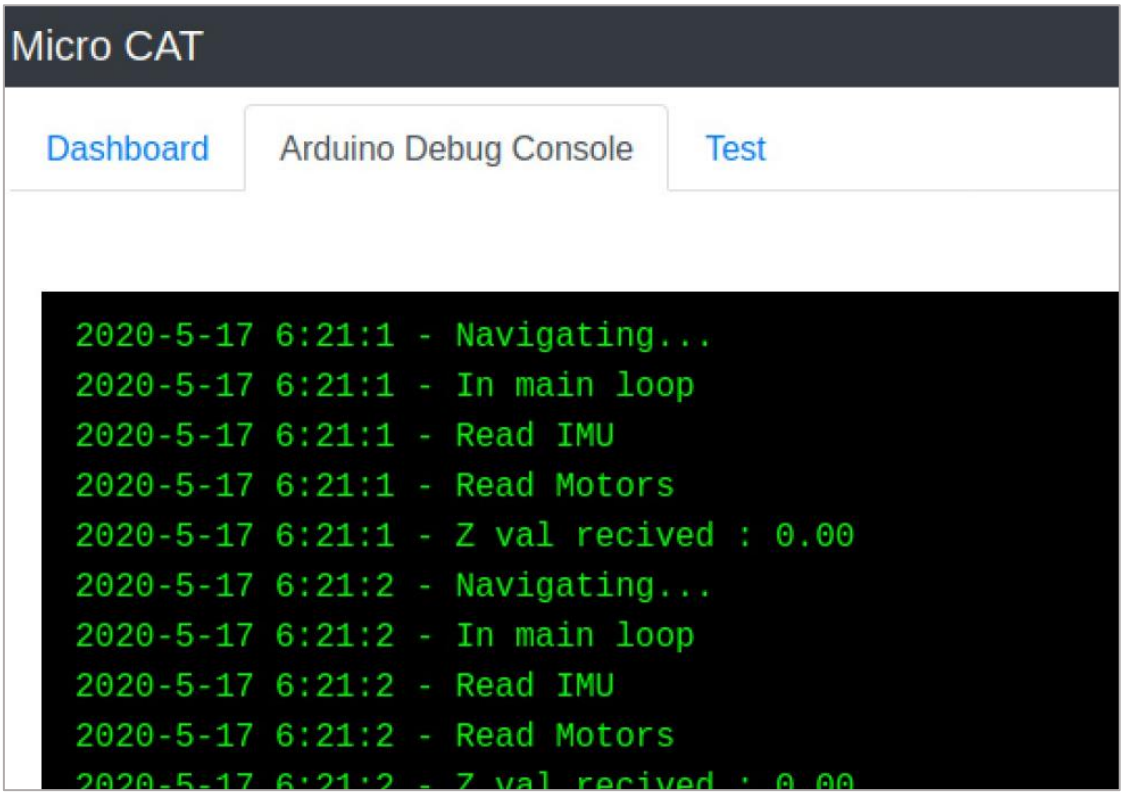


Figure 19. Debug console of the lower execution level of µ-CAT v2.

6 Discussion

This chapter discusses the results of the tests performed, methods and approaches used in the proposed solution, limitations and future work that can be carried by researchers as a continuation of this work.

Serial communication was tested using multiple test parameters, and results denotes that the success rate of the communication depends on the baud rate and time delay between two serial messages. In this work, communication was tested from Raspberry Pi Zero W to Arduino only. As future work, researchers can further test the success rate of the communication from Arduino mini 05 to Raspberry Pi Zero. Another possible scenario to test is the behaviours of the serial channel when data is sent from both higher and lower execution layers. These tests and results can be beneficial in terms of optimization of the communication between two execution layers.

Vison module was examined by presenting colour objects to the raspberry pi camera and observing the output on vision node. Test results for the vison node as seen in Figure 16, proves that the vision module was successful. However, there is a latency between the initial point where image was presented to the camera and extracting the output from the vision node. The possible explanation for the cause of the latency could be the lower rate used in the vision node. Since this is not tested in the current scope of the study it can be treated as a future work. Current vision node performs only a simple colour detection and sophisticated vision algorithms can be developed as an extension of the study.

Figure 15 presents the output of the bang-bang controller that indicates the controller is successful receiving depth data from UWSim and vison data from vision node. Evaluation of the controller is performed using HiL concept. Figure 14 presents the output of HiL test and, proves that the developed software components are working successfully together with the controller. However, experiments are done to evaluate the responsiveness of the controller by changing coloured objects rapidly infront of the robot's camera and results indicated that the robot is not highly responsive. Robot spent considerable amount of time to dive to a given certain depth. As future work researchers

can optimize the behaviour of the robot or add more complex control algorithms to the control node. Programming µ-CAT v2 with a state machine such as smach [59] and performing a mission also a possible future topic for researchers.

The results indicate that building an AUV for educational, research, and testing purposes is achievable using low-cost single board computers, microcontrollers, and distributed computing middleware. Results of the work align with previous efforts taken to miniaturize AUVs [32] [31] [33] yet used a comparatively low powered hardware. The software system was distributed among Raspberry Pi Zero W and Arduino mini 05 and the system was divided as a higher execution level and lower execution. Essential parts of the system were identified as control, vision and communication. Raspberry Pi Zero W is used for the higher execution layer because of its computational capability compared to the Arduino mini 05.

Goldberg et al [45] discussed the two main principles followed in the development of their software framework Huxley, as maintain appropriate interaction with the world and to perform activities at the appropriate timescales. When developing the µ-CAT v2 these principals were taken into consideration. Data from camera module and pressure sensor are used to monitor the environment and evaluate robot's status in the environment frequently. Vision node provides information as what's around the robot's environment, and this information is used to make decisions. Depth node provides the information where is robot in a specific time. This is also a vital information for the robot to perform its task. Any deficiency of these sensing will lead the robot to a catastrophic failure. Interactions of the robot and its reactions to its environment also should happen in a timely manner. Failure to perform an action after a successful identification will also lead to a catastrophic failure. With this knowledge time critical tasks of the robot such as actuator control are delegated to Arduino mini 05. Task that require higher computational power, but not time critical are placed in Raspberry Pi Zero W. Camera and sensor values are collected periodically.

µ-CAT v2 differs from the miniaturized AUVs considered in this work because of the reliability aspect considered in communication when developing the solution. Developer friendly features provided along with software solution such as passing debug messages from the micro controller to dashboard is also a feature introduce in this work.

7 Conclusion

In this thesis, a software solution, based on ROS, was developed for the AUV called µ-CAT v2. It is vital to understand the time-critical and non-time critical tasks in the development of robot software. The time-critical tasks of the robotic system need to take the least possible time to execute. Communication errors are very much possible in a distributed system. To prevent catastrophic failures due to communication errors checksum can be used. When using a checksum, the developer needs to consider the computational capacity and the memory usage of the checksum algorithm. HiL simulation is another useful technique when it is required to test the hardware related solution. HiL can save time, cost, and logistical overhead in such projects.

Selecting a framework for the development of a robot can provide leverage over the necessary time and effort spent in the development process. In the µ-CAT v2 ROS was selected as the framework. ROS provides a communication mechanism between processors, readily developed algorithms, and supporting libraries such as rosbridge package and roslibjs. Accessing the microcontroller can be difficult, when it is connected to a low powered computational unit like a Raspberry Pi Zero W. By using the proposed method in this thesis to occupy serial communication, rosbridge, and roslibjs can obtain efficiency in development. The usage of the tools developed in the thesis, such as a dashboard based on rosbridge and roslibjs, can make research and troubleshooting effective.

The current development of the software solution provides a framework for future developers. However, further research is needed to implement mission capabilities to the AUV and more sophisticated image processing algorithms. Researchers and students can work on implementing control algorithms on µ-CAT v2 as future work, and usage of modern software concepts such as containers (Docker) for AUV development will be another interesting research topic.

References

[1] ‘How Much Water is There on Earth?’ https://www.usgs.gov/special-topic/water-science-school/science/how-much-water-there-earth?qt-science_center_objects=0#qt-science_center_objects (accessed Apr. 18, 2020).

[2] Z. Zhu, Z. Wu, Z. Deng, H. Qin, and X. Wang, ‘An Ocean Bottom Flying Node AUV for Seismic Observations’, in *2018 IEEE/OES Autonomous Underwater Vehicle Workshop (AUV)*, Nov. 2018, pp. 1–5, doi: 10.1109/AUV.2018.8729726.

[3] B. M. M. Anwar, M. A. Ajim, and S. Alam, ‘Remotely operated underwater vehicle with surveillance system’, in *2015 International Conference on Advances in Electrical Engineering (ICAEE)*, Dec. 2015, pp. 255–258, doi: 10.1109/ICAEE.2015.7506844.

[4] M. R. Patterson, J. Elliott, and D. H. Niebuhr, ‘A STEM experiment in informal science education: ROVs and AUVs survey shipwrecks from the American Revolution’, in *2012 Oceans*, Oct. 2012, pp. 1–8, doi: 10.1109/OCEANS.2012.6404865.

[5] J. E. Manley, ‘Autonomous underwater vehicles for Ocean Exploration’, in *Oceans 2003. Celebrating the Past ... Teaming Toward the Future (IEEE Cat. No.03CH37492)*, Sep. 2003, vol. 1, pp. 327-331 Vol.1, doi: 10.1109/OCEANS.2003.178578.

[6] E. Vidal, J. D. Hernández, N. Palomeras, and M. Carreras, ‘Online Robotic Exploration for Autonomous Underwater Vehicles in Unstructured Environments’, in *2018 OCEANS - MTS/IEEE Kobe Techno-Oceans (OTO)*, May 2018, pp. 1–4, doi: 10.1109/OCEANSKOBE.2018.8559224.

[7] A. Foubert and J.-P. Henriet, *Nature and Significance of the Recent Carbonate Mound Record: The Mound Challenger Code*. Springer Science & Business Media, 2009.

[8] ‘ROV - Ages of Exploration’. https://exploration.marinersmuseum.org/object/rov/ (accessed Apr. 15, 2020).

[9] R. L. Wernli, ‘AUVs-a technology whose time has come’, in *Proceedings of the 2002 Interntional Symposium on Underwater Technology (Cat. No.02EX556)*, Apr. 2002, pp. 309–314, doi: 10.1109/UT.2002.1002444.

[10] W. Wang, R. C. Engelaar, X. Chen, and J. Chase, ‘The State-of-Art of Underwater Vehicles - Theories and Applications’, May 2009, doi: 10.5772/6992.

[11] J. W. Nicholson and A. J. Healey, ‘The Present State of Autonomous Underwater Vehicle (AUV) Applications and Technologies’, *Mar. Technol. Soc. J.*, Mar. 2008, doi: 10.4031/002533208786861272.

[12] T. Hiller, A. Steingrimsson, and R. Melvin, ‘Expanding the small AUV mission envelope; longer, deeper more accurate’, in *2012 IEEE/OES Autonomous Underwater Vehicles (AUV)*, Sep. 2012, pp. 1–4, doi: 10.1109/AUV.2012.6380725.

[13] R. Marthiniussen, K. Vestgard, R. A. Klepaker, and N. Storkersen, ‘HUGIN-AUV concept and operational experiences to date’, in *Oceans ’04 MTS/IEEE Techno-Ocean ’04 (IEEE Cat. No.04CH37600)*, Nov. 2004, vol. 2, pp. 846-850 Vol.2, doi: 10.1109/OCEANS.2004.1405571.

[14] B. Sahu and B. Subudhi, ‘The state of art of Autonomous Underwater Vehicles in current and future decades’, presented at the 1st International Conference on Automation, Control, Energy and Systems - 2014, ACES 2014, Feb. 2014, pp. 1–6, doi: 10.1109/ACES.2014.6808014.
[15] A. Raza, A. A. Ikram, A. Amin, and A. J. Ikram, ‘A review of low cost and power efficient development boards for IoT applications’, in *2016 Future Technologies Conference (FTC)*, Dec. 2016, pp. 786–790, doi: 10.1109/FTC.2016.7821693.
[16] A. Nayyar and V. Puri, ‘A review of Arduino board’s, Lilypad’s Arduino shields’, in *2016 3rd International Conference on Computing for Sustainable Global Development (INDIACom)*, Mar. 2016, pp. 1485–1492.
[17] D. R. Blidberg, ‘The Development of Autonomous Underwater Vehicles (AUV); A Brief Summary’, 2001.
[18] R. B. Wynn *et al.*, ‘Autonomous Underwater Vehicles (AUVs): Their past, present and future contributions to the advancement of marine geoscience’, *Mar. Geol.*, vol. 352, pp. 451–468, Jun. 2014, doi: 10.1016/j.margeo.2014.03.012.
[19] T. B. Curtin, D. M. Crimmins, J. Curcio, M. Benjamin, and C. Roper, ‘Autonomous Underwater Vehicles: Trends and Transformations’, *Mar. Technol. Soc. J.*, vol. 39, no. 3, pp. 65–75, Sep. 2005, doi: 10.4031/002533205787442521.
[20] C. Osterloh, T. Pionteck, and E. Maehle, ‘MONSUN II: A small and inexpensive AUV for underwater swarms’, in *ROBOTIK 2012; 7th German Conference on Robotics*, May 2012, pp. 1–6.
[21] A. Amory and E. Maehle, ‘SEMBIO - a small energy-efficient swarm AUV’, in *OCEANS 2016 MTS/IEEE Monterey*, Sep. 2016, pp. 1–7, doi: 10.1109/OCEANS.2016.7761458.
[22] S. Kalantar and U. Zimmer, ‘Control of Open Contour Formations of Autonomous Underwater Vehicles’, *Int. J. Adv. Robot. Syst.*, vol. 2, no. 4, p. 33, Dec. 2005, doi: 10.5772/5776.
[23] C. E. Wick and D. J. Stilwell, ‘A miniature low-cost autonomous underwater vehicle’, in *MTS/IEEE Oceans 2001. An Ocean Odyssey. Conference Proceedings (IEEE Cat. No.01CH37295)*, Nov. 2001, vol. 1, pp. 423–428 vol.1, doi: 10.1109/OCEANS.2001.968762.
[24] A. Underwood and C. Murphy, ‘Design of a micro-AUV for autonomy development and multi-vehicle systems’, in *OCEANS 2017 - Aberdeen*, Jun. 2017, pp. 1–6, doi: 10.1109/OCEANSE.2017.8084807.
[25] F. Bonin-Font, M. Massot-Campos, P. L. Negre-Carrasco, G. Oliver-Codina, and J. P. Beltran, ‘Inertial Sensor Self-Calibration in a Visually-Aided Navigation Approach for a Micro-AUV’, *Sensors*, vol. 15, no. 1, pp. 1825–1860, Jan. 2015, doi: 10.3390/s150101825.
[26] S. A. Watson and P. N. Green, ‘Design considerations for Micro-Autonomous Underwater Vehicles (μAUVs)’, in *2010 IEEE Conference on Robotics, Automation and Mechatronics*, Jun. 2010, pp. 429–434, doi: 10.1109/RAMECH.2010.5513154.
[27] S. Chao, G. Guan, and G.-S. Hong, ‘Design of a finless torpedo shaped micro AUV with high maneuverability’, *OCEANS 2017 – Anchorage*, 2017.
[28] C. Bennett *et al.*, ‘Ohio State University Underwater Robotics : Riptide AUV Design and Implementation’, 2016.
[29] A. Hackbarth, E. Kreuzer, and E. Solowjow, ‘HippoCampus: A micro underwater vehicle for swarm applications’, in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Sep. 2015, pp. 2258–2263, doi: 10.1109/IROS.2015.7353680.

[30] V. Mange, P. Shah, and V. Kothari, ‘Autonomous Underwater Vehicle: Electronics and Software Implementation of the Proton AUV’, p. 6.
[31] H. Hanff, K. Schmid, P. Kloss, and S. Kroffke, ‘µAUV2 - Development of a Minuscule Autonomous Underwater Vehicle’, Aug. 2016, doi: 10.5220/0005982201850196.
[32] H. Hanff *et al.*, ‘AUVx — A novel miniaturized autonomous underwater vehicle’, in *OCEANS 2017 - Aberdeen*, Jun. 2017, pp. 1–10, doi: 10.1109/OCEANSE.2017.8084946.
[33] S. A. Watson and P. N. Green, ‘Depth Control for Micro-Autonomous Underwater Vehicles (µAUVs): Simulation and Experimentation’, *Int. J. Adv. Robot. Syst.*, vol. 11, no. 3, p. 31, Mar. 2014, doi: 10.5772/57334.
[34] A. Hentout, A. Maoudj, and B. Bouzouia, ‘A survey of development frameworks for robotics’, in *2016 8th International Conference on Modelling, Identification and Control (ICMIC)*, Nov. 2016, pp. 67–72, doi: 10.1109/ICMIC.2016.7804217.
[35] A. S. Huang, E. Olson, and D. C. Moore, ‘LCM: Lightweight Communications and Marshalling’, in *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Oct. 2010, pp. 4057–4062, doi: 10.1109/IROS.2010.5649358.
[36] ‘LCM: Lightweight Communications and Marshalling (LCM)’. https://lcm-proj.github.io/ (accessed May 08, 2020).
[37] P. M. Newman, ‘MOOS - Mission Orientated Operating Suite’, 2008, Accessed: Apr. 29, 2020. [Online]. Available: https://ora.ox.ac.uk/objects/uuid:e7948d5b-efc3-481a-ad55-f0d39c28c74e.
[38] M. Montemerlo, N. Roy, and S. Thrun, ‘Perspectives on standardization in mobile robot programming: the Carnegie Mellon Navigation (CARMEN) Toolkit’, in *Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003) (Cat. No.03CH37453)*, Oct. 2003, vol. 3, pp. 2436–2441 vol.3, doi: 10.1109/IROS.2003.1249235.
[39] ‘CARMEN’. http://carmen.sourceforge.net/core_functionality.html (accessed May 08, 2020).
[40] M. Quigley *et al.*, ‘ROS: an open-source Robot Operating System’, in *ICRA 2009*, 2009.
[41] ‘Documentation - ROS Wiki’. http://wiki.ros.org/ (accessed May 08, 2020).
[42] B. P. Gerkey, R. T. Vaughan, and A. Howard, ‘The Player/Stage Project: Tools for Multi-Robot and Distributed Sensor Systems’, in *In Proceedings of the 11th International Conference on Advanced Robotics*, 2003, pp. 317–323.
[43] J. Pinto, P. S. Dias, R. Martins, J. Fortuna, E. Marques, and J. Sousa, ‘The LSTS toolchain for networked vehicle systems’, in *2013 MTS/IEEE OCEANS - Bergen*, Jun. 2013, pp. 1–9, doi: 10.1109/OCEANS-Bergen.2013.6608148.
[44] R. T. Vaughan and B. P. Gerkey, ‘Reusable Robot Software and the Player/Stage Project’, in *Software Engineering for Experimental Robotics*, D. Brugali, Ed. Berlin, Heidelberg: Springer, 2007, pp. 267–289.
[45] D. Goldberg, ‘Huxley: A flexible robot control architecture for autonomous underwater vehicles’, in *OCEANS 2011 IEEE - Spain*, Jun. 2011, pp. 1–10, doi: 10.1109/Oceans-Spain.2011.6003512.
[46] B. Allotta *et al.*, ‘The ARROWS project: adapting and developing robotics technologies for underwater archaeology’, *IFAC-Pap.*, vol. 48, no. 2, pp. 194–199, Jan. 2015, doi: 10.1016/j.ifacol.2015.06.032.

[47] T. Salumäe *et al.*, ‘Design principle of a biomimetic underwater robot U-CAT’, in *2014 Oceans - St. John’s*, Sep. 2014, pp. 1–5, doi: 10.1109/OCEANS.2014.7003126.
[48] M. Prats, J. Perez, J. J. Fernandez, and P. J. Sanz, ‘An open source tool for simulation and supervision of underwater intervention missions’, in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Vilamoura-Algarve, Portugal, Oct. 2012, pp. 2577–2582, doi: 10.1109/IROS.2012.6385788.
[49] ‘Iver3 Standard’, *OceanServer - Autonomous Underwater Vehicles*. https://ocean-server.com/iver3/ (accessed Apr. 25, 2020).
[50] R. E. Shannon, *Systems simulation: the art and science*. Prentice-Hall, 1975.
[51] R. Isermann, J. Schaffnit, and S. Sinsel, ‘Hardware-in-the-Loop Simulation for the Design and Testing of Engine-Control Systems’, *IFAC Proc. Vol.*, vol. 31, no. 4, pp. 1–10, Apr. 1998, doi: 10.1016/S1474-6670(17)42125-2.
[52] J. S. C. Jr and A. C. Jolly, ‘Hardware-in-the-loop simulation at the U.S. Army Missile Command’, in *Technologies for Synthetic Environments: Hardware-in-the-Loop Testing*, May 1996, vol. 2741, pp. 14–19, doi: 10.1117/12.241103.
[53] R. Isermann, S. Sinsel, and S. Schaffnit, ‘Hardware-in-the-Loop Simulation of Diesel Engines for the Development of Engine Control Systems’, *IFAC Proc. Vol.*, vol. 30, no. 3, pp. 91–93, Apr. 1997, doi: 10.1016/S1474-6670(17)44470-3.
[54] ‘rosbridge_suite - ROS Wiki’. http://wiki.ros.org/rosbridge_suite (accessed May 17, 2020).
[55] ‘roslibjs - ROS Wiki’. http://wiki.ros.org/roslibjs (accessed May 17, 2020).
[56] ‘Camera Module - Raspberry Pi Documentation’. https://www.raspberrypi.org/documentation/hardware/camera/ (accessed Apr. 25, 2020).
[57] ‘picamera — Picamera 1.13 Documentation’. https://picamera.readthedocs.io/en/release-1.13/ (accessed Apr. 25, 2020).
[58] ‘About’. https://opencv.org/about/ (accessed Apr. 26, 2020).
[59] ‘smach - ROS Wiki’. http://wiki.ros.org/smach (accessed May 17, 2020).