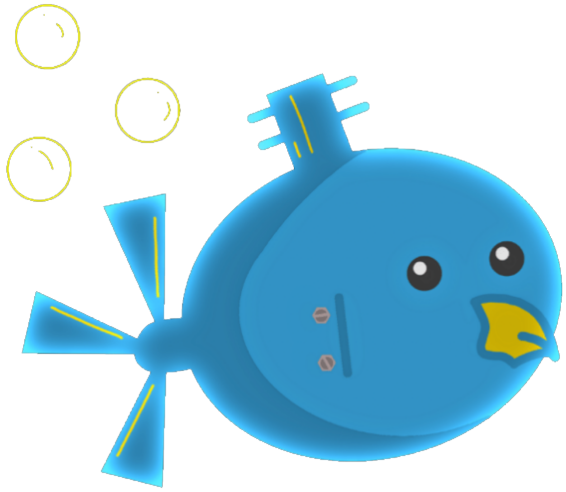
CALIFORNIA STATE UNIVERSITY, LOS ANGELES

Software Design Document



ROBOSUB

Members
Thomas BENSON, David CAMACHO, Bailey CANHAM, Brandon CAO,
Roberto HERNANDEZ, Andrew HEUSSER, Hector MORA-SILVA,
Bart RANDO, Victor SOLIS

Monday 6th March, 2023

Table of Contents

Revision History

Version	Description	Date
1.0	First release of Software Design Document.	9 December 2022

Table 1: Revision History

1 Introduction

1.1 Purpose

This document represents the software component of Lanturn, an Autonomous Underwater Vehicle made by the Robosub Senior Design Software and Hardware/Electrical teams. This will break down into further sub teams which put together make up the entire RoboSub Senior Design Software Team. These sub teams include Autonomy, Computer Vision, Controls, and Navigation. Some modules used for software include: the ROS module and the Arduino module.

1.2 Document Conventions

The standards/conventions that were used to write this document are very common formatting such as bold headings, highlighted words that are important, and smaller fonts for paragraphs.

1.3 Intended Audience and Reading Suggestions

The types of readers that this document is intended for are future developers of the RoboSub Senior Design years, developers who are interested in robotics, and hobbyists. This document will cover the four different sub teams that make up the RoboSub Senior Design Team working on Lanturn: Autonomy, Computer Vision, Controls, and Navigation.

Autonomy covers state machines, Computer Vision covers machine learning and image processing, Controls covers movement and IMU readings, Navigation covers sensor readings and mapping/localization.

1.4 System Overview

The Autonomy/Mission Planning sub team is to cover the SMACH model developed to control the AUV subsystems through each task at the RoboNation Competition.

The Computer Vision sub team is to recognize specific images, such as badges or dollar signs, in underwater environments and then process them through a machine learning model and send the relevant data to Autonomy/Mission Planning.

The Controls sub team is to control Lanturn, the sub, with its thrusters using PID controllers. This is done using three different axes: Pitch to tilt forward or backward, roll to move side to side, and Yaw to rotate left or right.

The Navigation sub team is to gather data from the sensors such as barometer and then also send relevant data to Autonomy/Mission Planning.

2 Design Considerations

This section describes many of the issues which need to be addressed or resolved before attempting to devise a complete design solution.

2.1 Assumptions and Dependencies

Lanturn will have a TX2 module, a small form-factor embedded computing device, for its main computer. The TX2 module comes flashed with Ubuntu 18.04 and comes packaged with Jetpack 4.6.1, a set of libraries designed for AI computations designed for the TX2 module.

Over the Operating System base and the Kernel overlay will be ROS2 Foxy, a piece of middleware that sets an environment for the different processes that will run on Lanturn.

The second computing device that will exist in the Lanturn system is a microcontroller, a teensy 4.1. The microcontroller will be used to interface with the onboard sensors, motors, and actuators.

2.2 General Constraints

Ubuntu is the only operating system that can run on the TX2 module. This requires any drivers or other implemented interfaces to be designed for, or to be compatible with Ubuntu versions.

The TX2 module and its accompanying software is designed for Ubuntu 18.04. Even though it is designed for Ubuntu version 18.04, it is possible to upgrade to a later Ubuntu version; however, the only Ubuntu version which is officially supported is Ubuntu 18.04. The Ubuntu version the TX2 will be running on is Ubuntu 20.04 Focal; all constraints that come with running on an unsupported version of Ubuntu on the TX2 module will exist in this project.

The middleware running on top of Ubuntu is a ROS2 version, ROS2 Foxy, which runs on Ubuntu 20.04. ROS2 Foxy comes with restrictions with the programming languages that can be used. The API that is used for development are rclcpp (C++ library) and rclpy (Python library).

The second computing device, Teensy 4.1, must be programmed in C++ using one of: teensyduino with Arduino IDE or platformio for Visual Studio Code.

2.3 Goals and Guidelines

This document will have the following goals and guidelines in mind:

- The Lanturn project must have a functioning autonomous system that can execute competition tasks by July 2023.
- All code written for this project should be documented.
- Programs written for this project should be refactored periodically to improve efficiency.

2.4 Development Methodology

The software team will be broken down into 5 sub-teams, each responsible for one module of the software system. Each week, the sub-teams show progress made and communicate any difficulties that have come across.

As sub-teams gain deeper understanding of their module, they will communicate to the other modules what data they can provide and what data they cannot.

This system will ensure that none of the modules depend on each other and can be switched out, if need be, in future versions of Lanturn.

3 Architectural Strategies

Each module will be programmed and packaged as a ROS2 package, except for the controls module which will exist as firmware on the microcontroller. Even though the microcontroller won't be a ROS package, it will still have all the characteristics of a ROS2 package; it will be entirely modular, using the same communication system provided by ROS2.

The controls module will interface with actuators, sensors and thrusters. The computer vision module will interface with cameras. All other modules will exist as software and will go through the controls and computer vision modules to interact with the environment.

Additionally, there will be a watchdog service that will monitor the liveliness and diagnostics of the system.

4 System Architecture

The software on Lanturn is organized into a number of modules, each of which is responsible for a specific task. These modules are build on the ROS framework and communicate with each other using ROS messages.

4.1 Overview of the System

Computer Vision shall output to: Mapping, Localization. Mapping shall output to: Autonomy, Localization. Localization shall output to: Autonomy Autonomy shall output to: Controls. Controls shall output to: Mapping, Localization.

Autonomy will manage time, manage state machines, read and interpret mapping/localization data, read and filter computer vision data, control claw, shoot torpedoes, release dropper, autonomously navigate map, and position/orient/center to desired orientations.

Comp Vision will publish raw images from front and bottom cams, detect and classify all task objects, provide distance from objects, calculate angle of incidence of objects.

Controls will read and publish data from Bar30 barometer, VN-100 IMU, Teledyne DVL, Sonar. Controls will also implement a PID library, generate PWM values that will move the sub to desired position, output PWM values to thrusters. Controls will also control a mechanical claw, shoot torpedoes, and release a ball from dropper all on command.

Mapping will subscribe to all computer vision data and Sonar data. Mapping will also implement a Kalman Filter, generate a map of the environment, position all task objects in map, and publish map.

Localization will subscribe to IMU data topic, Barometer data topic, both camera topics, DVL data topic, and map topic. Localization will position and orient the sub inside the map, and publish localization data.

4.2 Autonomy

The autonomy part of the software is responsible for completing the robosub goals. This is done using a state machine and using the ROS software to navigate the data to move from one state to another until each goal is completed, battery runs out, or time limit of 20 minutes is exceeded. The state machines are subscribed to all hardware components to be able to determine ubliches the next goal, and to controls to reach the next goal or complete the task. Each task/goal is broken down into their own states in their own class. Future implementation will move from state machine to behavior tree which allows for system to be goal oriented instead.

4.3 Computer Vision

The DFD Level 1 will start with the Camera sending information to the Image Processor, after processing the image it will send the information to Object Detection. Object Detection will process that information and once it's able to detect the object it will send the data to Object Classification, and it will define the location of the object. Once Object Classification processes the data of the object detected, it will send a Message Output. The Message Output will give out the data of the Object ID and the Bounding Box.

4.4 Controls

The controls code has the same form as any other microcontroller code. It starts with a setup() function then starts executing a loop() function until told to otherwise.

In the setup() function, sensors, actuators and the thruster motors are initialized and general setup like importing libraries and configuring the PID controller is done here. This function is executed once then control is turned over to the loop() function.

The loop() function contains the code for the PID controller, sensor reading and actuator control that will run independently. The flow of the data is this: grab sensor data, fix setpoints to account for circular rollover error, compute PWM values, combine the PWM values, output them to the ESCs and loop back to the beginning.

4.5 Localization

The localization node takes input data from the various environmental sensors (IMU, Sonar, Barometer, DVL, hydrophones). Localization will process this data and then publish the processed data to the entire system. Any node that subscribes will have access to the data.

4.6 Mapping

TBD.

5 Policies and Tactics

The main influences on Lanturn's Software design has already been established by previous years.

Some adjustments have been made by individuals to suit their ideas and programming style, but most of the system is inherited from previous years.

5.1 Specific Product Used

- System
  - Operating System
  - Ubuntu 20.04
  - Build System
  - Colcon
  - Ament_cmake
- Autonomy
  - Visual Studio Code
  - Groot
  - SMACH
  - BehaviorTree.CPP
- Controls
  - Visual Studio Code
  - Platform.I0
- Computer Vision
  - LabelIMG
  - Google CoLab

  - YOLOv4
  - Darknet
  - OpenCV
- Mapping and Localization
  - Gazebo
  - Visual Studio Code
  - Cubik Studio

5.2 Requirements Traceability

Each module of the system will have its own git repository. The sub-team assigned to the module will be responsible for keeping detailed documentation for the process of setting up, installing and using the software module.

5.3 Testing the Software

- Navigation
  - Create Gazebo Simulation for localization testing
  - Unit test for processing sensor data
  - Unit test for noise filter on sensor data
  - Unit test for publishing position data
  - Unit test for subscribing to position data
- Autonomy
  - Create unit test for states.
  - Create unit test for publishers
  - Create unit test for subscribers.
  - Simulate test environment for states
  - Simulate test environment for publishers
  - Simulate test environment for subscribers
  - Use testing AUV (Blastoise)

5.4 Engineering Trade-Offs

ROS is the leading opensource robotics software in use today. There are no engineering trade-offs with hardware and software support being mutigenerational.

5.5 Guidelines and Convetions

Using the selected IDE and choice of CPP instead of python ensures that the languages general convention is enforced.

5.6 Protocols

By using ROS the built in API of the standardized subscriber and publisher model is always enforced.

5.7 Maintianing the Software

No plans to maintain software, only fix bugs. The software is specially attached to the hardware, the only way to maintain software or update is to upgrade hardware. If we do that then we need to refactor or use a different ROS version.

5.8 Interfaces

There is no interface for users, fully autonomous, web GUI subscribes to publishers for user visualization.

7 Detailed Lower Level Component Design

7.1 Name of Class or File

7.1.1 Classifications

7.1.2 Processing Narrative(PSPEC)

7.1.3 Interface Description

7.1.4 Processing Details

7.1.5 Design Class Heirarchy

7.1.6 Restrictions/Limitations

7.1.7 Performance Issues

7.1.8 Design Contraints

7.1.9 Processing Detail For Each Operation

8 User Interface

8.1 Overview of User Interface

8.2 Screen Frameworks or Images

8.3 User Interface Flow Diagrams

9 Database Design

10 Requirements Validation and Verification

11 Glossary

12 References