Software Design Document

for

Robosub

Version 1.0 approved

Prepared by Carl Solli, Christian Castillo, Ashkan Aledavoud, Robin Romero, Alan Chan, Leslie Araujo, Edwin Tran, Bryan Sanchez, Jeanie Jeon, Daniel Valadez

May 13th 2022

Table of Contents

Revision History

Name	Date	Reason For Changes	Version

1. Introduction

1.1 Purpose

The purpose of this document is to explain the software in development for the AUV project that will be competing in the 2022 RoboSub competition.

The mission planning purpose is to design and implement the overall software of the AUV by continuously receiving data from all the other sub-teams and making decisions based on the data and its current state.

Controls will be focusing on the AUV's movement by receiving instructions from the mission planning sub-team and determining how to accomplish that given instruction. The movement will be controlled by using eight thrusters, where four thrusters will be used for vertical movement and to balance the sub and the other four thrusters will be used for horizontal movement. The pitch axis will be used to move the sub in a forward and backwards rotation about the side to side axis. The roll axis will rotate the AUV about the axis running through the nose to the tail. Yaw is the rotation of the AUV about the vertical axis.

The computer vision portion is designed to recognize specific images and objects in the underwater environment that are relevant to the competition. It will be used to detect and send data about objects and images to the mission planning system so it can be used alongside the rest of the data that is being collected by the other sub-teams.

This document should guide any future developers and users to the product and give them the opportunity to modify and expand on the AUV.

1.2 Intended Audience and Reading Suggestions

This document is intended for the future developers and users that wish to continue the work done by the 2021-2022 Senior Design Team. It is recommended that future developers and users go through the first five sections to get a better understanding of the product. The most relevant section for future developers is section six since it goes further into detail on what each sub-team should focus on and accomplish.

1.3 System Overview

The software system is split up into five parts (Mission Planning, Computer Vision, Controls, Sensors, and Website) where each part has its own specific task and responsibility.

1.3.1 Mission Planning

Design and implement the overall software of the AUV using SMACH to complete each task provided by the robosub competition. Mission planning also gathers every sub-team's data to make it possible for the AUV to maneuver and complete its goals.

1.3.2 Computer Vision

Design and train a computer vision system that has the ability to recognize a desired image or object that is in an underwater environment. The desired image or objects are set by the robosub competition.

1.3.3 Controls

Design and test a control system, with the use of eight thrusters, to allow six degrees of movement with precision and accuracy.

1.3.4 Sensors

Implements sensor data acquisition and processing using arduino and serial connections where data will be retrieved from the IMU sensor, DVL sensor, Barometer, Sonar, and Hydrophones and sent to mission planning to be used to complete the desired task.

1.3.5 Website

Design a website to document the progress of this year's AUVs, as well as provide a resource for anyone interested in joining the competition this year as a member of the RoboSub Club Design team.

2. Design Considerations

2.1 Assumptions and Dependencies

We're using Linux 18.04 for our OS on the Jetson TX2. Together with ROS Melodic for breaking our code into different components. Python 2.7 is also used.

Technologies:

- Linux Ubuntu 18.04
- ROS Melodic
- Python 2.7

Factors that may affect the requirements in the document are:

- The use of other Ubuntu versions
- Another version of ROS
- A programming languages other than Python and C++

2.2 General Constraints

Due to the ROS version on the platform, ROS nodes are restricted to be written in one of the following languages:

- Python 2.7
- C++

Any code to be run on an Arduino must be written in the Arduino language.

Main safety requirements are to not damage any components of the AUV as well as any item involved in the execution of competition tasks. Additionally,we must not cause any harm to the divers observing the competition in the water.

The software must be able to run on a Jetson TX2 and an Arduino Mega.

2.3 Goals and Guidelines

The goal of the software is to operate the AUV to complete the tasks of the Robosub competition. We also have a goal to design the software in a way that allows another team to continue work on the AUV instead of starting over from square one.

Mission Planning - Users will be able to build upon existing state machines as well as adding new state machines if needed. These states are independent,sharing utility functions and states

Controls - Designing controls code to be tested when the hardware is completed in 2021

Navigations - Test sensors independently and create packages dedicated to each individual sensor

Computer Vision - Design the workflow for labeling and training a computer vision system for use on a yearly basis. Each of these goals will be focused on the documentation and commenting of code with the expectation of reducing the time to onboard a new senior design team.

2.4 Development Methods

We've been using an Agile Development philosophy where we've adapted some of the methodologies that Scrum uses. Scrum is a framework that helps teams work together.

The parts of Scrum we've implemented so far is:
Backlog - We keep track of the tasks we need to do in a backlog. We've been using Github and Github Projects for this. Each sub team leader is in charge of managing the team's backlog.
Standup Meeting - We meet twice a week. In the meeting every team member goes over:

- What they have been working on
- What they are planning on working on going forward
- What issues they have been running into

3. Architectural Strategies

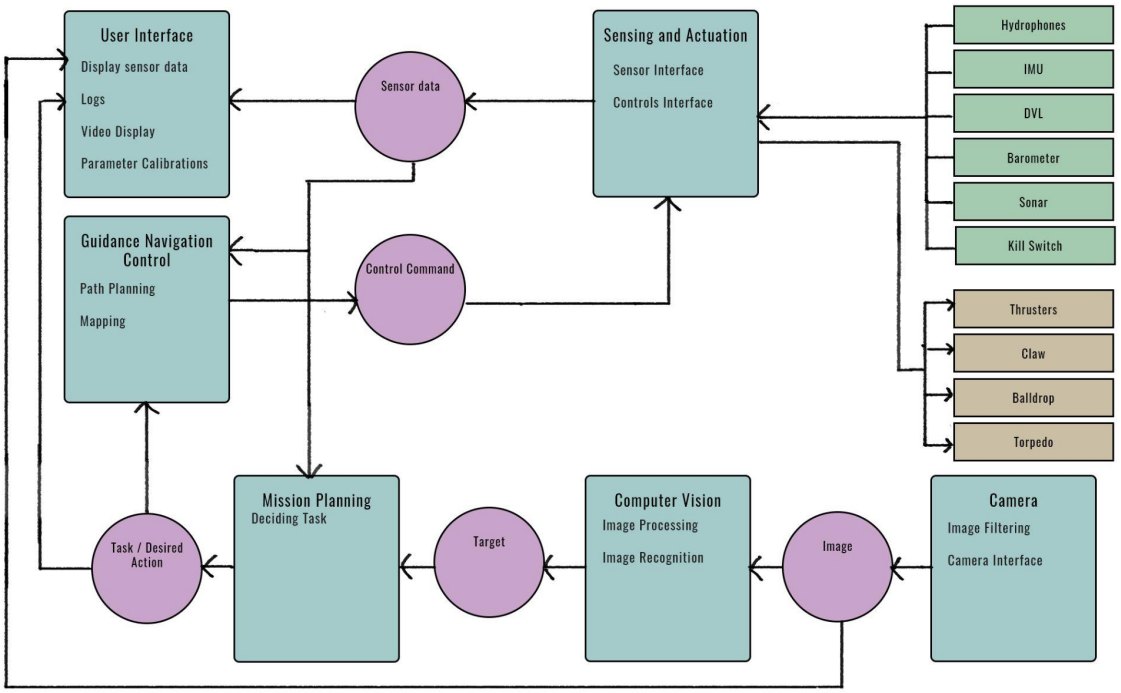
The software has been split into different components with the help of ROS. In ROS terminology these components are called packages. This has allowed us to separate the code of the different functions of the Robosub allowing for a well organized structure and that each package has responsibility for just one part of the overall system. The different packages we have is:

- Mission Planning
- Controls & Sensors
- Computer Vision
- Camera
- Guidance Navigation Control

Having the code in different components makes it easier to reuse the code, and makes it easier for each team to work on their part without affecting other parts of the system.

In addition the SMACH system was separated into various sub systems, each with the goal of completing a single task, allowing for faster development, debugging and testing times, as well as readability of code for similar tasks that reappear in the competition. An error package was also implemented to simplify monitoring of the various functionalities. This package can be expanded as functionality and requirements change.

4. System Architecture



This is how we've separated our ROS packages. We have a total of 6 different components working together. The blue squares are packages, the purple circles are ROS Topics. On the right, we have green squares which are sensors, and brown squares for controls.

User Interface - In the user interface we display sensor data, and logs. The user interface is just for testing and debugging, the AUV will operate without it in the competition. It takes in data from the Sensing and Actuation package as well as displaying the images from the Camera.

Sensing and Actuation - This package handles all the sensors and controls. The package will have nodes for the connecting to the different sensors and it will publish this data from the sensors, so the rest of the system can use that data. It will also send commands to the controls.

Guidance Navigation Control - When Mission Planning has decided on a task, it will communicate with this Guidance Navigation Control package, and this package will translate the commands into commands that will execute what the Robosub wants to do.

Mission Planning - Mission Planning uses SMACH to create state machines that allow the AUV to perform tasks given to it. The state machines work alongside other subsystems such as Computer Vision to achieve the various tasks outlined by the competition. SMACH states are subroutines that are developed to perform a specialized task. This will usually lead to an execution state that triggers a transition to a new state.

Computer Vision - This package will take in the data / images from the Camera and process these images. Its job is to find out what we’re looking at, and where in the frame this object is. Then publish this data for the Mission Planning to use.

Camera - This is for the camera, it will publish the images the camera takes.

5. Policies and Tactics

5.1 Choice of which specific products used

Arduino IDE

ROS - http://wiki.ros.org/

Arduino - https://www.arduino.cc/

SMACH - http://wiki.ros.org/smach/Documentation

Ubuntu 18.04.5 - https://releases.ubuntu.com/18.04.5/

Ubuntu 20.04.2.0 - https://releases.ubuntu.com/20.04/

VMware Workstation - https://my.vmware.com/en/web/vmware/downloads/details?downloadGroup=PLAYER1556&productId=800&rPId=47861

5.2 Plans for ensuring requirements traceability

Traceability is enforced by requiring appropriate readme documents that explain how to utilize the different programs the AUV uses as well as the tasks that the AUV has to perform using those programs.

5.3 Hierarchical organization

The Software shall follow the following organization.

Each package will be located in its own directory and contain the following:

- Readme File containing description of package
    - Software requirements
    - Interface specification
    - Hardware description (if applicable)
- Start script named start.py (or c++ file where appropriate)
- Required libraries (for hardware components)
- Scripts (or src) directory containing all software.
- Design Images of software (where appropriate)

5.4 Software Implementation Procedure

The software must be cloned from the repository in the catkin workspace folder created by ROS

Example structure:

Catkin_ws

-Software Directory

-src

The software directory folder must be renamed to src, removing the existing src directory.

To compile the following command shall be run in the terminal:

catkin_make

6. Detailed System Design

6.1 Mission Planning

6.1.1 Responsibilities

The main responsibility of mission planning is to design and implement the overall software of the AUV to complete each task of the competition using SMACH. In addition, mission planning has to publish data to the controls component based on the current state of the AUV to maneuver throughout the competition.

6.1.2 Constraints

- SMACH will fall short as the scheduling of the task becomes less structured
- SMACH is not meant to be used as a state machine for low-level systems that require high efficiency, SMACH is a task-level architecture.
- The state machines perform one task at a time, unless it is designed to run in concurrence

6.1.3 Composition

- ROS
- SMACH
- SMACH Viewer

6.1.4 Uses/Interactions

- Define the current state of the AUV
- Subscribe to receive data from other components of the AUV
- Publish data to controls component of the AUV
- Define transition between sub-states of a state machine
- Define transition between different state machines
- Pass user data between different state machines

6.1.5 Resources

- SMACH – A ROS-independent Python library to build hierarchical state machines (http://wiki.ros.org/smach/Documentation)
- SMACH Viewer – GUI that shows the state of hierarchical SMACH state machines. (http://wiki.ros.org/smach_viewer#Documentation)

6.1.6 Interface/Exports

- Mission planning only interfaces through other systems that directly interact with the hardware and does not directly interface with any hardware.
- Mission planning shall interact with the user interface to provide the current state of the AUV
- Mission planning shall interact with controls to provide instructions to maneuver the AUV

6.2 Controls & Sensors

6.2.1 Responsibilities

The controls and sensors component are in charge of interfacing with the controls and the sensors of the robosub. The sensors group will receive data from the sensors on the AUV and then send this data to the mission planning group to determine the AUV's next action. Commands will then be received from the mission planning node to execute the action needed.

6.2.2 Constraints

Communication between the software developers and hardware developers working on the same or different parts of the AUV.

6.2.3 Composition

ROS - Robot Operating System

Arduino IDE - Arduino Integral Development Editor

IMU - Inertial Measurement Unit

DVL - Doppler Velocity Log

Barometer - Measures depth and temperature

Sonar - For detecting objects

Hydrophones - For sound detection

6.2.4 Uses/Interactions

The functions of the software involved:

- Retrieving data from the IMU sensor
- Retrieving data from the DVL sensor
- Retrieving data from the Barometer
- Retrieving data from the Sonar
- Retrieving and processing the data from the Hydrophones
- Publishing all the data

6.2.5 Resources

IMU - https://www.arduino.cc/reference/en/libraries/mpu6050_tockn/

DVL - https://www.eol.ucar.edu/system/files/VN100manual.pdf

Barometer - https://github.com/bluerobotics/Bar30-Pressure-Sensor

Sonar - https://bluerobotics.com/store/sensors-sonars-cameras/sonar/ping-sonar-r2-rp/

Hydrophones - https://www.aquarianaudio.com/as-1-hydrophone.html

6.2.6 Interface/Exports

The sensors nodes will publish ROS topics with the following data for each sensor:

Barometer.msg

Field name	Type
depth	float32
temperature	float32

Hydrophones.msg

Field name	Type
direction	int32

Sonar.msg

Field name	Type
distance	float32
confidence	float32

IMU.msg

Field name	Type
roll	int32
pitch	int32
yaw	int32

DVL.msg

Field name	Type
roll	int32
pitch	int32
yaw	int32
x_translation	float32
y_translation	float32

6.3 Computer Vision

6.3.1 Responsibilities

Process video stream from the on board camera. Detect and classify objects relevant to the competition and give useful information to the AUV. Determine current distance from target object and direct AUV to move to object.

6.3.2 Constraints

The processing, object detection, and classification should be performed as close to real time as possible.

Storage of the video may take a large amount of memory if not compressed.

Transmitting video to a GUI may take a large amount of bandwidth

Processing the large amount of video data may also cause the computer to generate heat which can not be easily cooled off.

The object classification algorithm must be well trained.

Object detection and/or classification may be affected by the image processed when underwater.

6.3.3 Composition

OpenCV - Computer Vision Library

YOLOv4 - Object detection algorithm

Darknet - Neural Network framework

CUDNN - GPU accelerated library

6.3.4 Uses/Interactions

Computer Vision uses YOLO for real time object detection and classification. YOLO is trained using Darknet by giving it labeled images of the objects to be found in the competition. CUDNN accelerates the whole process by allowing us to use more of the hardware to process the video stream. OpenLabeling allows us to label images for training.

6.3.5 Resources

DARKNET https://github.com/AlexeyAB/darknet

OPENCV https://opencv.org/

YOLO https://pjreddie.com/darknet/yolo/

CUDNN https://developer.nvidia.com/cudnn

OpenLabeling https://github.com/Cartucho/OpenLabeling

6.3.6 Interface/Exports

Computer Vision communicates to the Mission planning

The output is an object named data with the following attributes:

object - String, label for the object that was detected

confidence - double, a double from 0.0 to 1.0 of how confident the algorithm is. confidence levels under our set threshold will not be shown

vertical - int, number of pixels that the center of the object is away from the center of the video. Negative is to the up, positive is to the down.

horizontal - int, number of pixels that the center of the object is away from the center of the video. Negative is to the left, positive is to the right.

6.4 User Interface

6.4.1 Responsibilities

The user interface is to be used only for testing purposes and not during the competition itself. It will display the data that is received from the AUV's multiple sensors as well as video, control status, and the current state of the AUV. It will not be used when it is the AUV's time to compete.

6.4.2 Constraints

Limitations for the user interface is no communication is allowed when the AUV is competing in the competition, this includes receiving data from the AUV to use for the user interface. When it is the AUV's time to compete, all communication with the sub must stop; this includes receiving data to be displayed during the competition.

6.4.3 Composition

The user interface is divided into 5 sections that are the general, video, sensor, control, and state. The general section will display options to start creating a log for the data received and calibrating. The video section will display what the video camera on the AUV is seeing. The sensor section will display all data that is received from the sensors on the AUV. The control section will display the status of the different controls on the AUV. The state section will show the current state of the AUV.

6.4.4 Uses/Interactions

The user interface will receive data from the sensor programs, camera, control programs, and state machine programs which will allow the user to oversee the general state of all the mission planning, sensors, computer vision, and control data. The user interface should not have any effect on any part of the system that it will be receiving data from.

6.4.5 Resources

Resources that the user interface will require and use is Tkinter. Tkinter is a graphical user interface library that will allow us to effectively create our user interface application to display the data we receive during the testing of the AUV.

6.4.6 Interface/Exports

The user interface will directly communicate and receive data with the following modules.

- Mission Planning
    - State of the AUV
- Sensors
    - IMU
        - Pitch, yaw, roll
    - Barometer
        - Depth

    - Sonar
        - Distance
        - Confidence level
    - DVL
        - Depth
        - Velocity
    - Hydrophone
        - Direction of sound
- Computer Vision
    - Video Camera feed
- Controls
    - Thruster status
    - Torpedo status
    - Claw status
    - Ball drop status

6.5 Website

6.5.1 Responsibilities

The website is a requirement for the RoboSub competition. In the competition, there are a few requirements that the website must satisfy in order to get a higher score. Therefore, the website is mainly used to satisfy the requirements for the RoboSub competition, but it can also serve as a resource for all the members of the RoboSub Senior Design team, the RoboSub Club team, as well as any other people who might be interested in RoboSub. The website will therefore provide information and resources about RoboSub. This information will be as general as possible, so as to allow everyone, whether they are familiar with RoboSub or not, to understand the general idea of what RoboSub is and does.

6.5.2 Constraints

The website will not interact with the rest of AUV software. Therefore, the software will be limited to what it can do separate from the rest of the software, and that is first to satisfy the RoboSub competition requirements, and second to serve as a resource for anyone that is involved or interested in RoboSub, providing information about this year's project, as well as any previous RoboSub projects and designs. The updates to the website will therefore be connected with the development of the AUV. As progress is made in the development of the AUV(s) and their software, the website is updated with information on the state of the AUV(s) in question. Thus some of the website's information depends on the overall development of the AUV and its software.

6.5.3 Composition

As with most websites, the composition of the RoboSub website will consist of multiple JS pages, all of which will be connected with each other. The website will also be styled and decorated with CSS files for each of the pages. Finally, the website

and its components will be updated on GitHub, where commits to the website's repository will also update the website itself with the updated version on GitHub.

6.5.4 Uses/Interactions

The website is for the most part separate from the rest of the AUV software, as it is used for satisfying the competition requirements/rules and for providing information and resources rather than for the functionality of the AUV itself. Thus, the website does not interact with any other components of the software. However, this website will be interacted with by various people, including the people who are hosting the RoboSub competition, and many who may not be members of the senior design team or competition, because the website is a resource for anyone who is interested in RoboSub. Thus, the interactions of the RoboSub website are separate from the rest of the AUV software, being a resource that is made for anyone, whether a member of the senior design team, a RoboSub competition judge, or anyone else interested in RoboSub.

6.5.5 Resources

As mentioned previously, the RoboSub website will not be managing anything that is essential to the function of the AUV. The website is separate from the rest of the RoboSub software, but it will have its own libraries which it will use, separate from the rest of the AUV's software. One of these is React, which is a JavaScript library used for building UIs for websites. And, as mentioned previously, the website can be modified and updated from its GitHub repository, which will make changes to the website as new commits are made to the website's GitHub repository.

6.5.6 Interface/Exports

The website will not provide any services for any other part of the AUV, whether code or hardware, and it will consist mainly of text and buttons.

7. Detailed Lower level Component Design

7.1 state_zero.py

7.1.1 Classification

This file contains the state machine for state zero.

7.1.2 Processing Narrative (PSPEC)

It is a python file that is loaded onto the AUV and makes sure that all state zero implementations contain the proper methods needed for runtime.

7.1.3 Interface Description

The state_zero.py interface utilizes ROS

7.1.4 Processing Detail

To be determined at a later date.

7.1.4.1 Design Class Hierarchy

Not applicable.

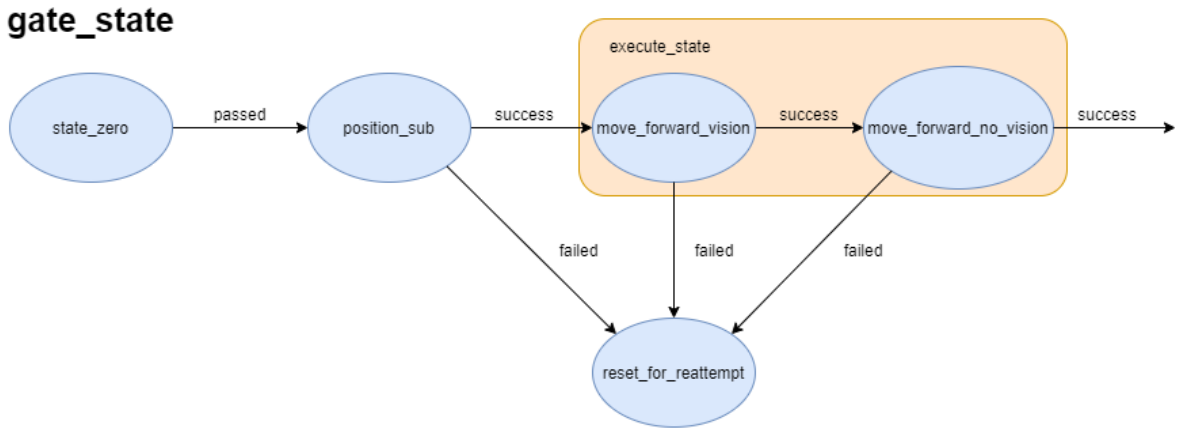
7.1.4.2 Restrictions/Limitations

Must use as little CPU/GPU resources as possible to complete the task.

7.1.4.3 Performance Issues

Not issues at the moment.

7.2 execute_gate.py



7.2.1 Classification

This file contains the state machine for gate state.

7.2.2 Processing Narrative (PSPEC)

It is a python file that is loaded onto the AUV and makes sure that all gate state implementations contain the proper methods needed for runtime.

7.2.3 Interface Description

The execute_gate.py interface utilizes ROS

7.2.4 Processing Detail

To be determined at a later date.

7.2.4.1 Design Class Hierarchy

Not applicable.

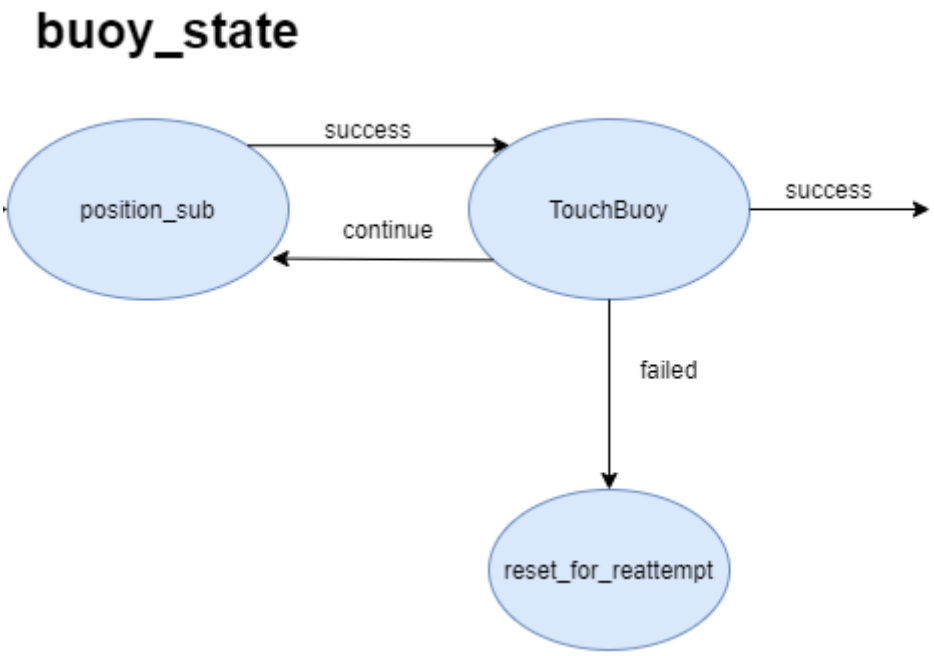
7.2.4.2 Restrictions/Limitations

Must use as little CPU/GPU resources as possible to complete the task.

7.2.4.3 Performance Issues

No issues at the moment.

7.3 execute_buoy.py



7.3.1 Classification

This file contains the state machine for buoy state.

7.3.2 Processing Narrative (PSPEC)

It is a python file that is loaded onto the AUV and makes sure that all buoy state implementations contain the proper methods needed for runtime.

7.3.3 Interface Description

The execute_buoy.py interface utilizes ROS

7.3.4 Processing Detail

To be determined at a later date.

7.3.4.1 Design Class Hierarchy

Not applicable.

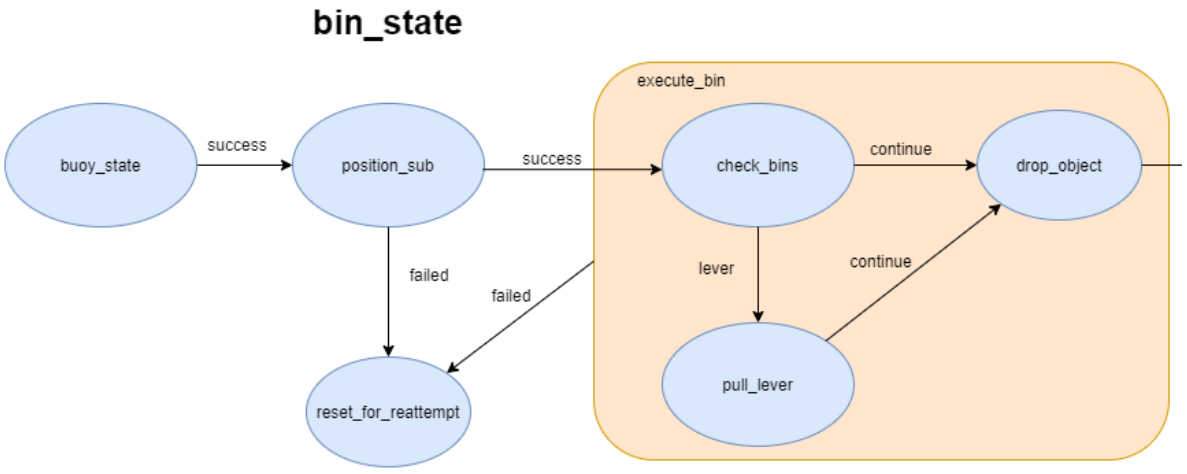
7.3.4.2 Restrictions/Limitations

Must use as little CPU/GPU resources as possible to complete the task.

7.3.4.3 Performance Issues

Parts of the file has to be modified in oder to complete the task.

7.4 execute_bin.py



7.4.1 Classification

This file contains the state machine for bin state.

7.4.2 Processing Narrative (PSPEC)

It is a python file that is loaded onto the AUV and makes sure that all bin state implementations contain the proper methods needed for runtime.

7.4.3 Interface Description

The execute_bin.py interface utilizes ROS

7.4.4 Processing Detail

To be determined at a later date.

7.4.4.1 Design Class Hierarchy

Not applicable.

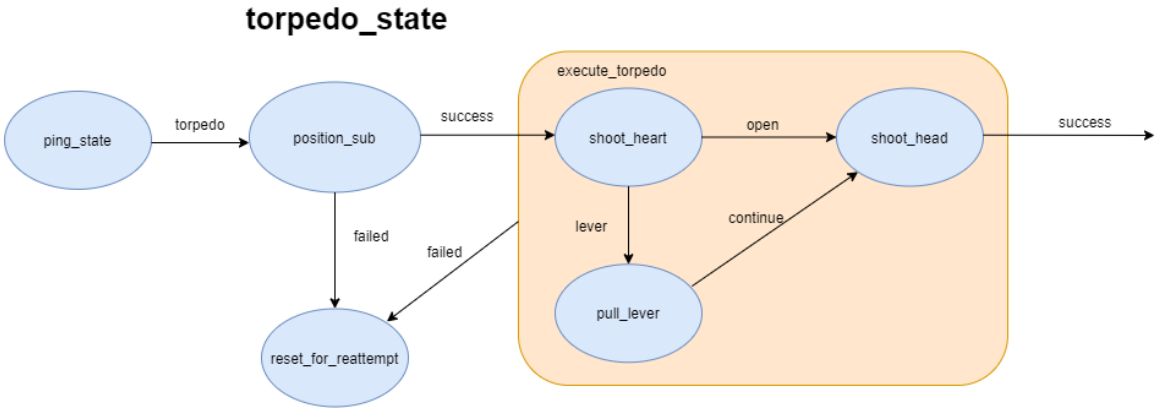
7.4.4.2 Restrictions/Limitations

Must use as little CPU/GPU resources as possible to complete the task.

7.4.4.3 Performance Issues

Parts of the file has to be modified in oder to complete the task.

7.5 execute_torpedo.py



7.5.1 Classification

This file contains the state machine for torpedo state.

7.5.2 Processing Narrative (PSPEC)

It is a python file that is loaded onto the AUV and makes sure that all torpedo state implementations contain the proper methods needed for runtime.

7.5.3 Interface Description

The execute_torpedo.py interface utilizes ROS

7.5.4 Processing Detail

To be determined at a later date.

7.5.4.1 Design Class Hierarchy

Not applicable.

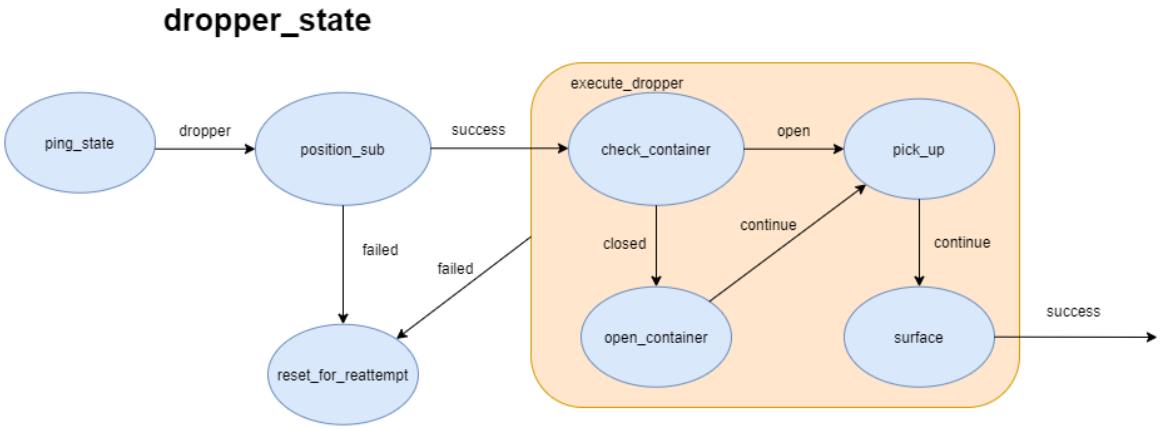
7.5.4.2 Restrictions/Limitations

Must use as little CPU/GPU resources as possible to complete the task.

7.5.4.3 Performance Issues

Parts of the file have to be modified in order to complete the task.

7.6 execute_dropper.py



7.6.1 Classification

This file contains the state machine for octagon state.

7.6.2 Processing Narrative (PSPEC)

It is a python file that is loaded onto the AUV and makes sure that all octagon state implementations contain the proper methods needed for runtime.

7.6.3 Interface Description

The execute_dropper.py interface utilizes ROS

7.6.4 Processing Detail

To be determined at a later date.

7.6.4.1 Design Class Hierarchy

Not applicable.

7.6.4.2 Restrictions/Limitations

Must use as little CPU/GPU resources as possible to complete the task..

7.6.4.3 Performance Issues

Parts of the file have to be modified in order to complete the task.

8. Database Design

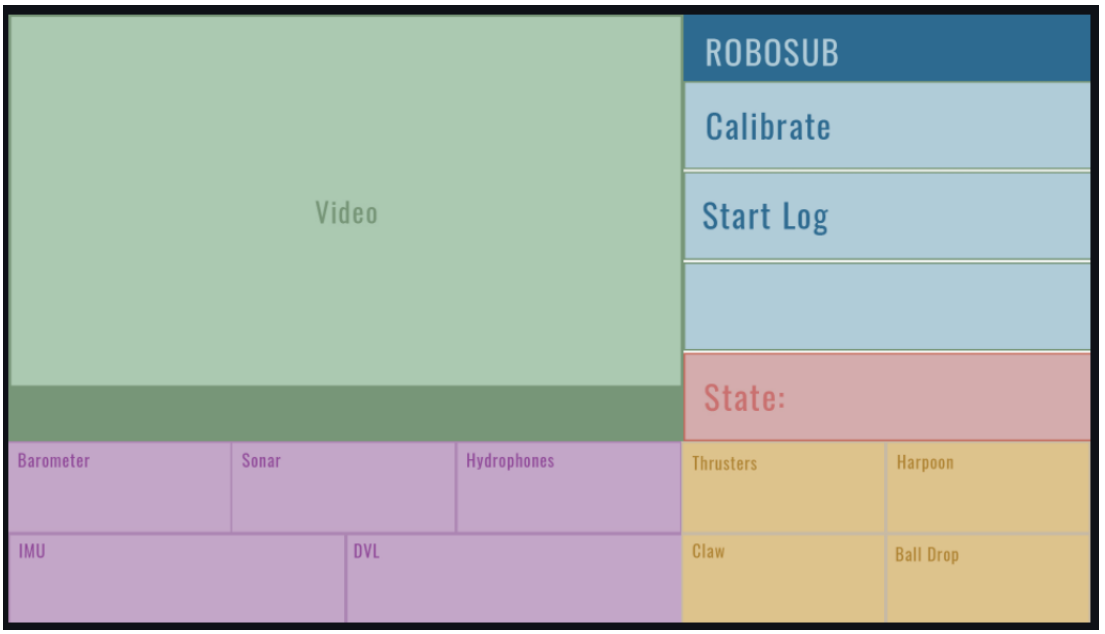
This project does not require a database.

9. User Interface

9.1 Overview of User Interface

The user interface will display all the data received from our AUV's sensors, video camera, controls, and state of the AUV. The user interface is to not be used during the competition and will only be used when testing the AUV.

9.2 Screen Frameworks or Images



9.3 User Interface Flow Model

The user interface will be divided into five sections. The general section will give the option of calibrating and starting the log that is to be used to save data. The video section of the user interface will display what the AUV is seeing through its video camera in real time. The sensors section will display all the information that is received from the sensors. The state section will display the current state that the AUV is currently in. The controls section will display the status of the AUV's controls.

10. Requirements Validation and Verification

10.1 Mission Planning

Requirement No.	Requirement Description
10.1.1	The system shall provide data to other systems of the AUV to perform specific instructions.
10.1.2	The system shall receive data from other systems to perform current state.
10.1.3	The system shall follow a plan of tasks the AUV needs to complete.

10.2 Navigation

Requirement No.	Requirement Description
10.2.1	The system shall communicate pinger information received from the Hydrophones to the AUV.
10.2.2	The system shall communicate object detection data from the Sonar sensor to the AUV.
10.2.3	The system shall communicate velocity measurements and translational motion data received from the DVL to the AUV.
10.2.4	The system shall communicate pitch, roll, and yaw movements received from VectorNav IMU sensor to the AUV.
10.2.5	The system shall communicate depth measurements received from the Barometer to the AUV.

10.3 Controls

Requirement No.	Requirement Description
10.3.1	The system shall use data retrieved from sensors to send directional commands to controls.
10.3.2	The system shall communicate appropriate speed to thrusters for balancing the AUV.
10.3.3	The system shall use ROS publishers and subscribers to communicate data.

11. Glossary

- Amplifier - a device that increases the amplitude of the hydrophone signals
- AUV – Autonomous Underwater Vehicle
- GUI - Graphical User Interface
- Hydrophone - a microphone designed for underwater use
- Pitch - the turn about an axis from the left to right side of the RoboSub
- React – A JavaScript library for building UIs for websites.
- Roll - the turn about an axis from the nose to the tail of the RoboSub
- SMACH – State Machine
- Sonar - A device that can tell us distance to an object
- Yaw - the turn about an axis from the top to the bottom of the RoboSub

12. References

RoboSub 2022 Handbook:
https://robonation.org/app/uploads/sites/4/2022/05/2022-RoboSub_Team-Handbook_v2.0.pdf

ROS: http://wiki.ros.org/Documentation
SMACH: https://wiki.ros.org/smach?distro=melodic
Smach Viewer: http://wiki.ros.org/smach_viewer