Software Design Document for Robo Sub

Version 2.0 approved

Prepared by Victor Solis, David Camacho, Hector Mora-Silva, Roberto Hernandez, Bart Rando, Andrew Heusser, Bailey Canham, Milca Ucelo Paiz, Thomas Benson, Brandon Cao

Advisor: Richard Cross

Computer Science Department, California State University, Los Angeles

May 12, 2023

Version History

Date	Version	Notes
Dec 9, 2022	1.0	First release of document
May 12, 2023	2.0	Second release of document

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 Methods

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 system is designed as a ROS system and uses ROS conventions.

4.1 Overview of the System

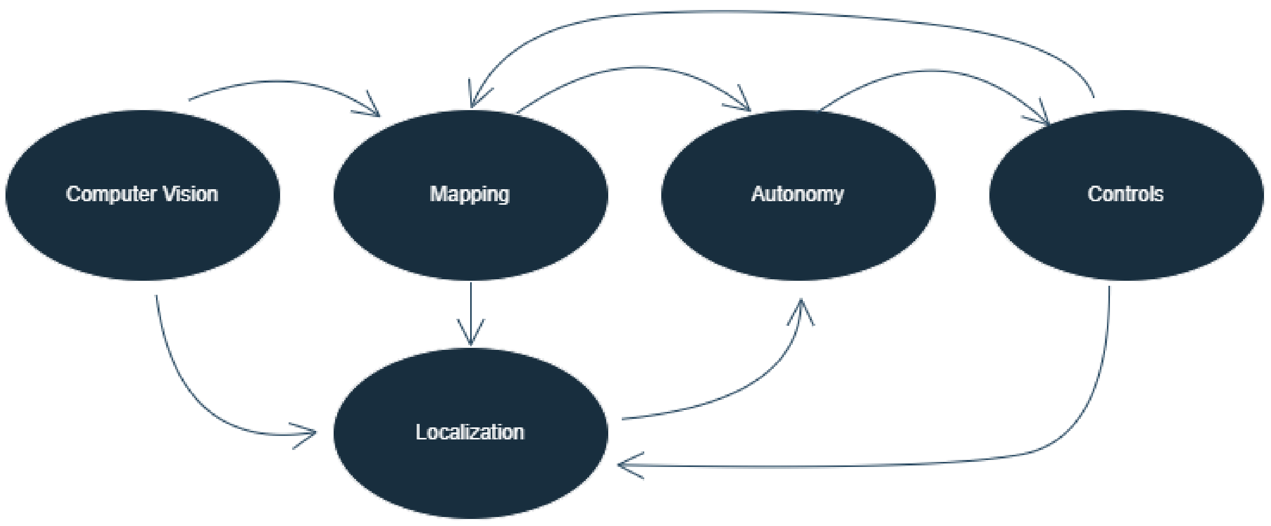


Figure 01: A high level overview of the software modules in the system

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, and Teledyne DVL. 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.

Autonomy

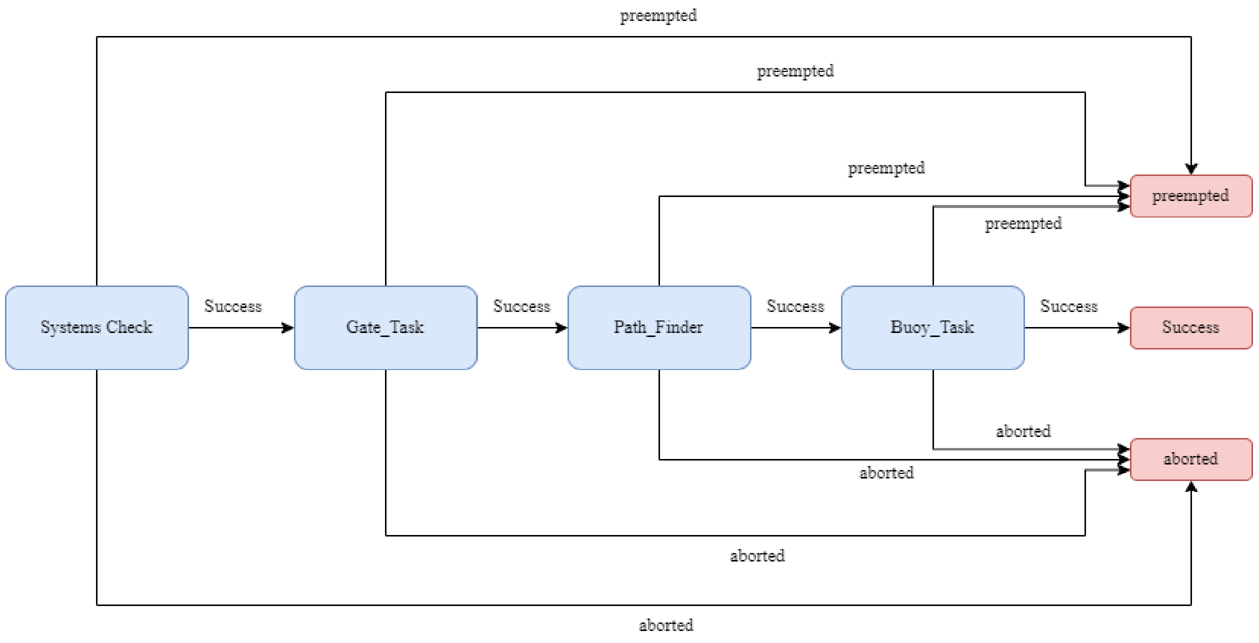


Figure 02: DFD LV1 Autonomy

The autonomy part of the software is responsible for completing the robosub goals. This is done using behavior trees 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. Autonomy class is subscribed to all hardware components to be able to determine the next goal, and to controls to reach the next goal or complete the task. Each task/goal is broken down into their own behavior trees in their own class.

Computer Vision

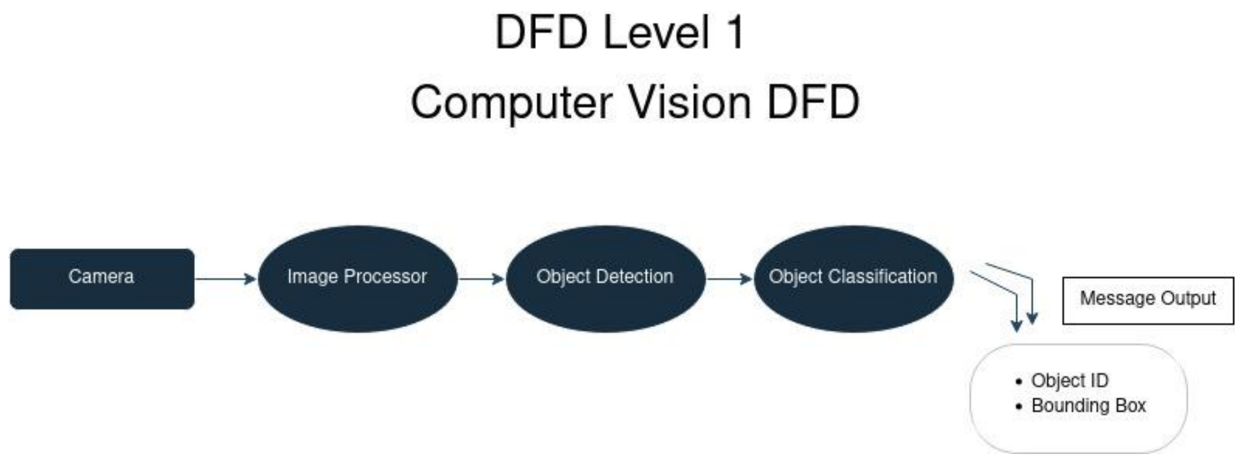


Figure 03: DFD LV1 Computer Vision

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.

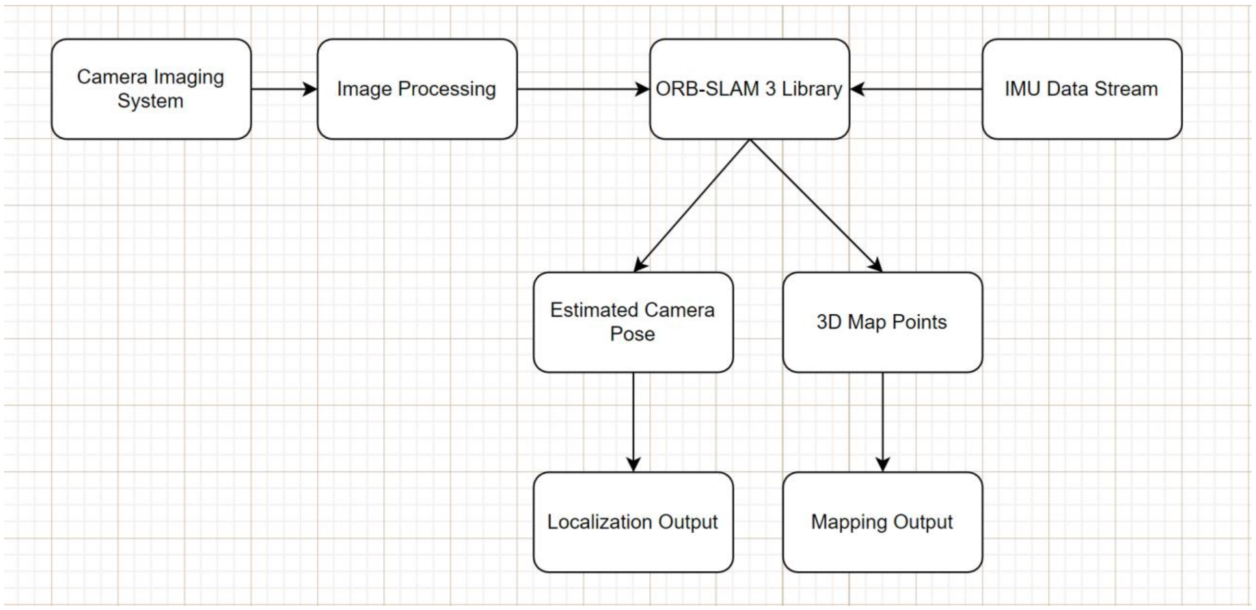
Mapping & Localization

High-level Description:

The module receives sensor data inputs, such as camera images and IMU data, and processes them using the ORB_SLAM3 library. The outputs of the module include the estimated camera pose, which represents the position and orientation of the camera in the environment, and a reconstructed 3D map in the form of a point cloud.

The module is built using a combination of the ORB_SLAM3 library, ROS 2 framework, and additional components for handling sensor data and publishing the results. The architecture employs the Observer and Publisher-Subscriber design patterns for efficient communication between different components.

Level 1 Data Flow Diagram (DFD):



Level 1 Control Flow Diagram:

1. Receive camera images and IMU data from sensor streams.
2. Convert sensor data into a format suitable for ORB_SLAM3 processing
3. Feed the sensor data into the ORB_SLAM3 library for simultaneous localization and mapping
4. Retrieve the estimated camera pose and 3D map points from the ORB_SLAM3 library
5. Convert the camera pose and map points into the desired output format
6. Publish the localization and mapping outputs for use by other components of the system

Design Patterns:

The architecture of the mapping and localization module employs the following design patterns:

1. Observer: The module observes the sensor data streams (camera images and IMU data) and processes them as they become available. This pattern ensures that the module stays up to date with the latest sensor data

2. Publisher-Subscriber: The module uses the ROS 2 framework's publisher-subscriber mechanism for communication between components. The pattern enables efficient and decoupled communication between the module and other parts of the system that rely on the localization and mapping outputs.

Controls

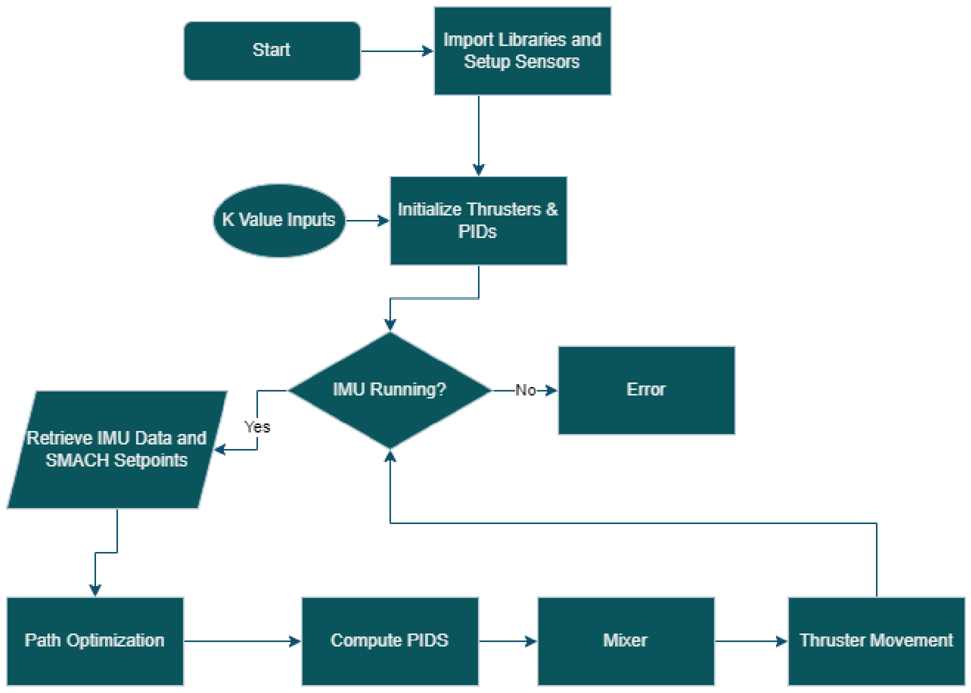

Start
Import Libraries and Setup Sensors
K Value Inputs
Initialize Thrusters & PIDs
IMU Running?
No
Error
Yes
Retrieve IMU Data and SMACH Setpoints
Path Optimization
Compute PIDS
Mixer
Thruster Movement

Figure 06: DFD LV1 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.

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 Choice of which specific products used

System

- Operating System
    - Ubuntu 20.04
- Build System
    - Colcon
    - Ament_cmake

Autonomy

- Visual Studio Code
- Groot
- BehaviorTree.CPP

Controls

- Visual Studio Code
- Platform.I0

Computer Vision

- LabelIMG
- Google CoLab
- YOLOv4
- Darknet
- OpenCV

Mapping & Localization

- Visual Studio Code
- ORB-SLAM3
- Pangolin

- OpenCV
- Eigen
- dBow
- g2o
- ROS 2

5.2 Plans for ensuring 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 Plans for testing the software

Navigation

- Simulation testing: Test the algorithm using a simulated environment. This will help identify and fix potential issues in a controlled setting. Simulation tools like Gazebo can help to produce realistic underwater scenarios.
- Unit testing: Develop unit tests for the individual components and functions of the ORB-SLAM3 implementation to ensure they perform as expected. This will help catch any errors early in the development process and improve the overall systems reliability.
- Integration testing: Once individual components have been tested, perform integration testing to ensure the ORB-SLAM3 implementation works correctly with other subsystems.
- Test dataset evaluation: Use publicly available underwater datasets to evaluate the ORB-SLAM3 implementations performance.
- Performance metrics: Define performance metrics, such as localization accuracy, map quality, and computational efficiency, to quantify the ORB-SLAM3 implementation's performance. These metrics will help assess the system's effectiveness and identify areas for improvement.
- Test scenarios: Create a diverse set of test scenarios to evaluate the performance of the ORB-SLAM3 implementation under different conditions.

Autonomy

- Create unit test for Behavior Trees.
- Create unit test for publishers
- Create unit test for subscribers.
- Simulate test environment for Behavior Trees
- Simulate test environment for publishers
- Simulate test environment for subscribers

- Use testing AUV (Blastoise)

5.1 **Engineering trade-offs**

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

5.2 **Coding guidelines**

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

5.3 **Protocol**

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

5.4 **Software Choice**

Ros supports both state machine and behavioral trees, currently using state machines, but moving to behavior trees will ensure that we are more efficient and easier to get to function and complete goals.

5.5 **Software Maintenace**

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.

5.6 **User Interface**

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

5.7 **Code Hierarchy**

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

-Readme File containing description of package

- Software requirements

- Interface specification

- Required libraries (for hardware components)

- Scripts (or src) directory containing all software.

- Design Images of software (where appropriate)

5.8 **Dependency and Building**

The installation process for each component has their instructions on their website, detail installation will be provided in the readme file for each component, any software that needs compiling will be using Cmake.

- Install Ubuntu 20 (link in resources)
- Install ROS (link in resources)
- Install SMACH (link in resources)
- Install BehaviorTree.cpp (link in resources)
- Install Visual Studio. (link in resources)

5.9 **Database**

No database is used in autonomy.

6. Detailed System Design

6.1 Autonomy

The main responsibility of autonomy is to design and implement the logic software of the AUV to complete each task of the competition, using BehaviorTrees. In addition, autonomy is responsible for receiving and publishing data. Autonomy receives data from computer vision, localization, and mapping components. Based on the current state of the AUV, to maneuver throughout the competition, the data collected is then published to the control's component.

6.1.2 Constraints

- Autonomy will be responsible for system checks prior to starting goal search, and aborting mission if critical component failure occurs.
- Autonomy will run until either all goals at met, battery capacity runs out or time limit exceeds 20 minutes.
- Autonomy will assume all data published is correct, data is formatted and validated prior to being published.
- Autonomy is responsible for completing tasks that lead to completing each goal successfully, based on its current state and retry if it fails the goal state and moving on to the next goal/state.
- Failure to complete tasks in a timely matter will result in Behavior Tree failing.
- Behaviors Trees are flexible, rescuable and can be multi embedded, performing one task at a time, but will parallel goals.

6.1.3 Composition

- ROS2: Robot Operating System V2
- Behavior Tree: BehaviorTreeCpp/PyTrees

Future Implementation:

- Behavior Tree: Complete all Goals Logic
- Choose CPP or Python Implementation, initial state currently written in both languages for testing.

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-behavior Trees
- Define the transition between different behavior trees.
- Pass user data between different trees using blackboard

6.1.5 Resources

- BehaviorTree.CPP - Implemented using C++, assembled using a scripting language based on XML. Behavior Trees are composable. You can build complex behaviors by reusing simpler ones. (https://www.behaviortree.dev/)
- PyTrees – Behavior Trees implemented in Python. (https://py-trees-ros-tutorials.readthedocs.io/en/stable/)
-

6.1.6 Interface/Exports

- Autonomy interfaces only through other systems that directly interact with the hardware and doesn't directly interface with any hardware.
- Autonomy shall interact with the user interface to provide the current state of the AUV
- Autonomy interacts with all other components to receive data to determine the current state of the AUV
- Autonomy interacts with controls to provide instructions to maneuver the AUV

6.2 Computer Vision

6.2.1 Responsibilities

The primary responsibility of the computer vision is to utilize the cameras attached to the RoboSub in order to identify various competition items. The computer vision program will then output a bounding box around the identified object and send that information to the navigation controls. The computer vision model should have a high enough accuracy to allow us to be confident in the result. Additionally, the program should output the distance and angle the robot is from the identified object and send this data to the navigation controls.

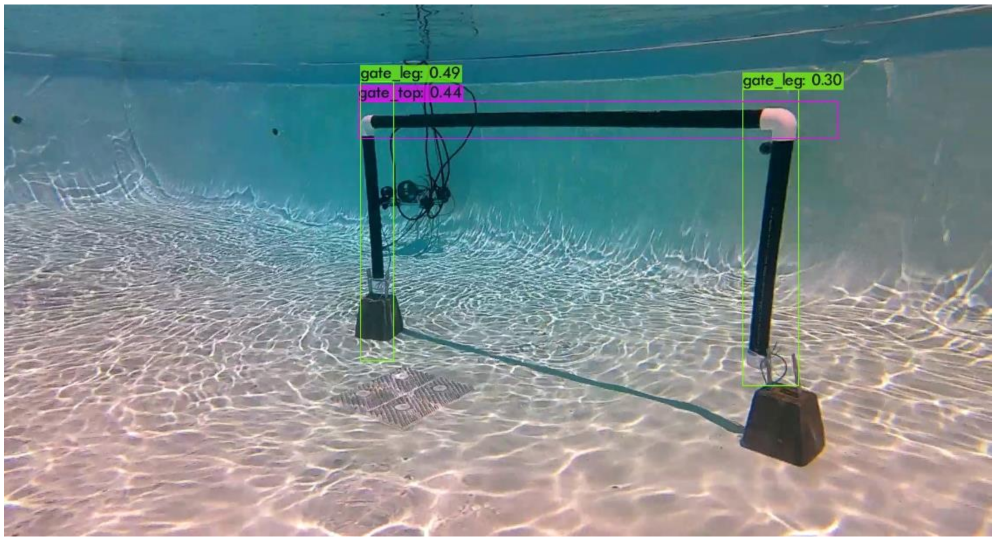
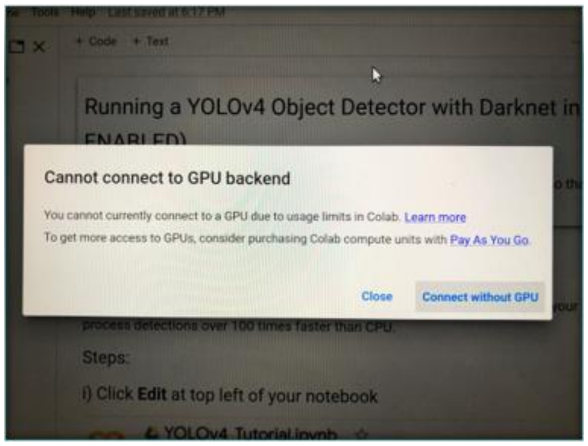


Figure 07: Example Task Object Detection

6.2.2 Constraints

The main constraint on the computer vision model is time and storage. We must train the computer vision model on pictures of each object we want it to be able to identify. However, in order to do this, we must have hundreds of pictures available and saved for when we train the model. Therefore, we must consider how much storage we have available on the computer. Additionally, training the computer vision model takes quite a bit of time. In our first training session, we only got through around 500 iterations of training, and it took over 5 hours. Due to this, we must consider how much time we have and carefully plan it out in order to ensure we have enough time for the model to train enough to be reliable.



Running a YOLOv4 Object Detector with Darknet in

ENABLED)

Cannot connect to GPU backend

You cannot currently connect to a GPU due to usage limits in Colab. Learn more

To get more access to GPUs, consider purchasing Colab compute units with Pay As You Go.

process detections over 100 times faster than CPU.

Steps:

i) Click **Edit** at top left of your notebook

YOLOv4_Tutorial.ipynb

Figure 08: Google CoLab contraint

6.2.3 Composition

The first subcomponent is Google CoLab, which is a collaborative python programing space that provides access to cloud computing. We use Google CoLab for its GPU to train and test our computer vision model. The second subcomponent is YOLOv4, which is an object detection algorithm. It works by dividing images into a grid system with each cell in the grid responsible for detecting objects within itself. The third subcomponent is Darknet, which is a neural network framework written in C and CUDA. We use this in conjunction with YOLOv4 to complete object detection. The fourth subcomponent is OpenCV, which contains an optimized computer vision library, tools, and hardware all aimed at real-time object recognition. We are testing this as a second option to compare the results with YOLOv4 and Darknet. The fifth and final subcomponent is LabelIMG, which is a graphical image annotation tool. We use this to label our images prior to using them to train our computer vision model.

6.2.4 Uses/Interactions

The other component that will be receiving the data is Navigation, either by sending them the data of the output of the bounding box as well as the angle and distance that the robot is from the identified object, which will be whatever object they present us with during the competition. The only side-effects that we could experience in this component would be if we were to mislabel an input from the camera.

6.2.5 Resources

The main resource needed by the computer vision model is the cameras on the RoboSub. Since we need to identify the objects around the robot, we need to receive data from the cameras to run through our object detection model. We would also need significant processing power to run our computer vision model. However, by knowing this and communicating with other teams, we can plan for this and create a system that works with our program.

6.2.6 Interface/Exports

- Export bounding box to Navigation Controls
- Export distance from the robosub to the object to Navigation Controls
- Export angle between robosub and object to Navigation Controls

6.3 Controls

6.3.1 Responsibilities

Controls acts as the primary interface between the hardware and the software for communications to the main motherboard and ROS operating system. It is responsible for allowing the various software components to communicate back and forth with all the sensors, motors, and actuators attached to the robot as necessary.

6.3.2 Constraints

- Requiring permission from manufacturers in order to gain access to libraries and documentation of components.

- Communication between software developers and hardware developers.

- IMU data is sent in a binary bit format and must be parsed and converted into floats to be usable.

6.3.3 Composition

Vector Nav VN-100	Inertial Measurement Unit measure orientation, velocity, and changes in acceleration
Bar30 Barometer	Pressure sensor that measures the depth of the robot
Teledyne Pathfinder DVL (Doppler Velocity Log)	Responsible for measuring the distance, the robot has traveled using sonic waves.
T-200 Motors	Responsible for moving and leveling the robot through the water

6.3.4 Uses/Interactions

Retrieving data from the IMU sensor

Retrieving data from the DVL sensor

Retrieving data from the Barometer

Sending commands to motor controller

Publishing data to ROS

6.3.5 Resources

IMU - https://www.vectornav.com/resources/user-manuals/vn-100-user-manual

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

DVL - http://www.teledynemarine.com/Pathfinder_DVL?ProductLineID=34

6.3.6 Interface/Exports

- Publish IMU data to ROS
- Publish Barometer data to ROS
- Publish DVL data to ROS
- Publish Sonar data to ROS

6.4 Mapping & Localization

6.4.1 Responsibilities

The ORBSLAM3NODE is a ROS 2 node responsible for interfacing with the ORB_SLAM3 system. Its primary responsibilities include:

1. Subscribing to the camera topic
2. Process the incoming images and IMU messages
3. Passing the image and IMU data to the ORB-SLAM3 system for monocular tracking with IMU
4. Publishing the estimated camera pose and reconstructed 3D map

6.4.2 Constraints

1. Assumes the correct path to the ORB-SLAM3 settings file is provided and the settings file contains the proper camera and IMU intrinsics
2. Assumes the incoming image messages are greyscale and single channel.
3. Assumes the incoming IMU messages have valid linear acceleration and angular velocity data
4. ORB-SLAM3 system should be properly initialized and configured

6.4.3 Composition

1. slam_system: A shared pointer to the ORB-SLAM3 system instance
2. Image_sub: A subscription to the image topic for receiving image messages
3. imu_sub: A subscription to the IMU topic for receiving IMU messages
4. pose_pub: A publisher for broadcasting the estimated pose
5. map_pub: A publisher for broadcasting the reconstructed 3D map
6. map_pub_timer: A timer for periodically publishing the reconstructed 3D map

6.4.4 Uses/Interactions

1. ORB-SLAM3: The ORB-SLAM3 system is used for processing image and IMU data and performing monocular SLAM.
2. ROS 2: The node is built using the ROS 2 framework and interacts with other nodes through subscriptions, publishers, and timers.

6.4.5 Resources

1. ORBSLAM3NODE(): Constructor for the ORBSLAM3NODE class
2. ~ORBSLAM3NODE(): Destructor for the ORBSLAM3NODE class
3. publish_map(): Function to publish the map points generated by ORB-SLAM3
4. image_callback(): Callback function for handling the incoming image messages
5. imu_callback(): Callback function for handling the incoming IMU messages
6. pose_to_msg(): Static function to convert an OpenCV pose matrix to a geometry_msgs::msg::Pose ROS message
7. map_to_msg(): Static function to convert a vector of ORB_SLAM3::MapPoint pointers to a sensor_msgs::msg::PointCloud2 ROS message

6.4.6 Interface/Exports

TBD

7. Detailed Lower-level Component Design

7.1 IMU Class

7.1.1 Classification

Class responsible for initializing and acquiring data from the Vector Nav VN-100 IMU

7.1.2 Processing Narrative (PSPEC)

No previously written library for the VN100 was publicly available capable of communicating over the 3v UART serial interface. Implementation of such a library was necessary to acquire data from the IMU, from the Teensy microcontroller.

7.1.3 Interface Description

The VN100.cpp interface utilizes serial communication to get system register information directly from the VN100 in the form of binary outputs at the byte level.

7.1.4 Processing Detail

The VN100 shall be implemented within a non-blocking loop, running at the minimum of 100hz

7.1.4.1 Design Class Hierarchy

This class does not inherit from any previous class structures.

7.1.4.2 Restrictions/Limitations

The VN100 must read serial data at high speeds, any code that slows down looping, will prevent the sensor from communicating. Non-blocking code is mandatory.

7.1.4.3 Performance Issues

No issues now.

7.2 ORBSLAM3NODE Class

7.2.1 Classification

7.2.2 Processing Narrative (PSPEC)

This class serves as the primary interface between the ROS 2 framework and the ORB_SLAM3 library. It subscribes to image and IMU topics, processes incoming data using ORB_SLAM3, and publishes the estimated camera pose and the reconstructed 3D map.

7.2.3 Interface Description

- Subscribes to '/camera/image_raw' topic for incoming messages

- Subscribes to ‘/IMU/data’ topic for IMU messages
- Publishes camera pose to ‘/orbslam3/pose’ topic as ‘geometry_msgs::msg::PoseStamped’ messages
- Publishes reconstructed 3D map to ‘/orbslam3/map’ topic as ‘sensor_msgs::msg::PointCloud2’ messages

7.2.4 Processing Detail

7.2.4.1 Design Class Hierarchy

- Inherits from ‘rclcpp::Node’

7.2.4.2 Restrictions/Limitations

- Assumes the messages are in “mono8” encoding
- Assumes the settings file for ORB_SLAM3 is located at a specific path

7.2.4.3 Performance Issues

- The processing time for image and IMU data depends on the complexity of the scene and the computational resources available
- The map publishing rate is limited by the timers interval

7.2.4.4 Design Constraints

- Requires the ORB-SLAM3 library and ROS 2 framework to be installed and properly configured

7.2.4.4 Processing Detail for Each Operation

- Constructor: Initializes the ORB_SLAM3 system, sets up subscriptions, publishers, and timers
- Destructor: Shuts down the ORB_SLAM3 system
- ‘publish_map()’: Retrieves map points from ORB_SLAM3, converts them to a ‘PointCloud2’ message, and publishes the message
- ‘image_callback()’: Processes incoming messages, passes them to ORB_SLAM3, retrieves the current pose, and publishes it if available
- ‘imu_callback()’: Processes incoming IMU messages and passes them to ORB_SLAM3
- ‘pose_to_msg()’: Converts an OpenCV pose matrix to a ‘geometry_msgs::msg::Pose’ message
- ‘map_to_msg()’: Converts a vector of ‘ORB_SLAM3::MapPoint’ pointers to a ‘sensor_msgs::msg::PointCloud2’ message
- ‘main()’: initiliizes the ROS 2 framework, creates an instance of the ‘ORBSLAM3Node’ class, and starts the event loop

8. Database Design

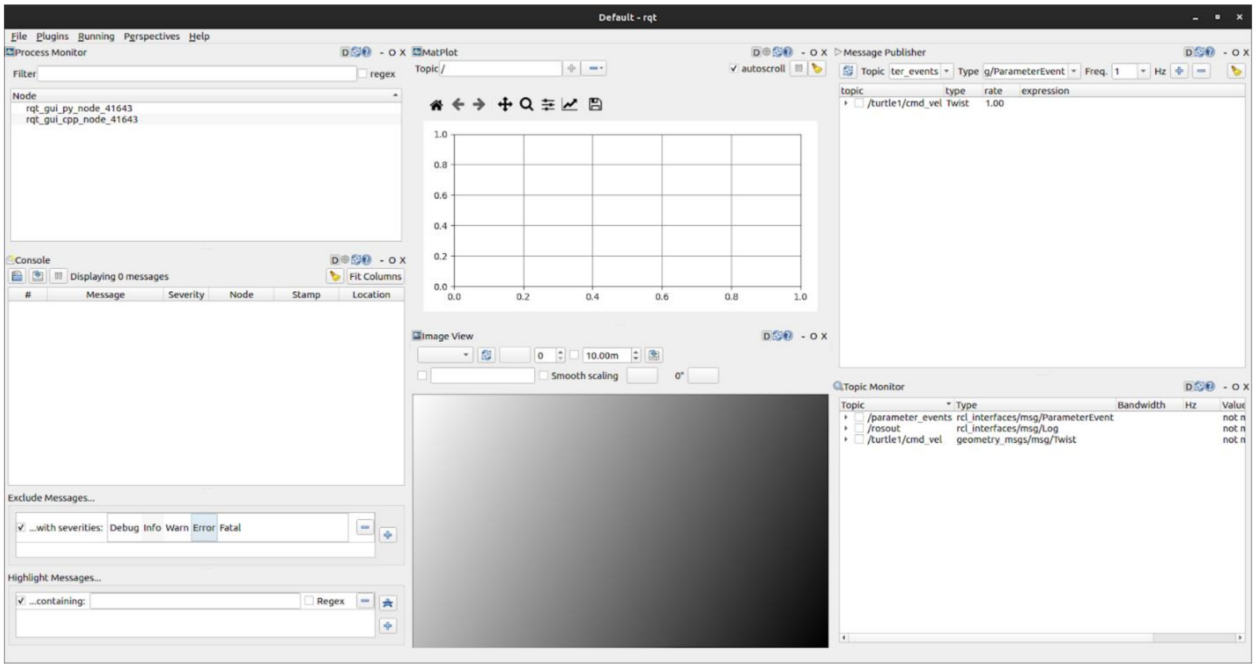
Not applicable.

9. User Interface

9.1 Overview of User Interface

9.2 Screen Frameworks or Images

RQT



Process Monitor — Filter regex

Node
rqt_gui_py_node_41643
rqt_gui_cpp_node_41643

Console — Displaying 0 messages

#	Message	Severity	Node	Stamp	Location

Exclude Messages... ✓ ...with severities: Debug Info Warn Error Fatal

Highlight Messages... ✓ ...containing: Regex

MatPlot — Topic: /, ✓ autoscroll

Image View — 0, 10.00m, Smooth scaling, 0°

Message Publisher — Topic: ter_events, Type: g/ParameterEvent, Freq.: 1 Hz

topic	type	rate	expression
/turtle1/cmd_vel	Twist	1.00	

Topic Monitor

Topic	Type	Bandwidth	Hz	Value
/parameter_events	rcl_interfaces/msg/ParameterEvent			not n
/rosout	rcl_interfaces/msg/Log			not n
/turtle1/cmd_vel	geometry_msgs/msg/Twist			not n

Figure 09: RQT Example Plugin Layout

RQT is a user interface that comes packages with ROS and other visualization tools. Ir can be used to read, send and interpret data. It allows a user to interact with the ROS system the GUI is connected to. It uses the concepts of plugins to allow the user to view different forms of data and interpret them in different ways.

In this image, there are six plugins open.

1. Top Left. The Process Monitor plugin helps the user see all processes interacting with or through the ROS system.
2. Top Center. The MatPlot plugin can interpret data in a grid form and, in the context of Lanturn, it can be used to interpret stability, velocity and other odometry data.
3. Top Right. The Message Publisher allows the user to send different messages to nodes in the ROS system.
4. Bottom Right. The Console plugin displays log messages the nodes in the ROS system are saving.

5. Bottom Center. The Image View plugin displays images that are being streamed through a topic.
6. Bottom Right. The Topic Monitor shows the topics in the current ROS system and allows users to see what messages are going through the topic.

9.3 User Interface Flow Model

The RQT user interface uses plugins to display information giving the user full control as to what is displayed and what is not. This means the user decides on the flow of the user interface, meaning, any flow diagram displayed here would miss the point of the modularity of this user interface design.

10. Requirements Validation and Verification

Method of Testing:
- Testing using additional ad-hoc created software including a correlation testing unit.
- Demonstration of the specified capability
- Inspection of the software code possibly using additional inspection techniques
- Analysis of the specific code operation/algorithm to prove functionality.

Requirements related to 1. Autonomy		
Requirement Number	Requirement Task	Method for Testing
1.1	The autonomy module shall manage time in a run through the course	Unit Testing
1.2	The autonomy module shall send current state to system logs	Unit Testing
1.3	The autonomy module shall read and interpret mapping data	Unit Testing
1.4	The autonomy module shall read and interpret localization data	Unit Testing
1.5	The autonomy module shall read and filter computer vision data	Unit Testing
1.6	The autonomy module shall send command to control claw	Unit Testing
1.7	The autonomy module shall send command to shoot torpedoes	Unit Testing
1.8	The autonomy module shall send command to release dropper	Unit Testing
1.9	The autonomy module shall send a heartbeat to the watchdog service	Unit Testing
1.10	The autonomy module shall know the task being executed	Unit Testing
1.11	The autonomy module shall autonomously navigate map	Unit Testing
1.12	The autonomy module shall position submarine in a desired position	Unit Testing
1.13	The autonomy module shall orient submarine in a desired orientation	Unit Testing
1.14	The autonomy module shall be able to center with objects	Unit Testing

Requirements related to 2. Computer Vision		
Requirement Number	Requirement Task	Method for Testing

2.1	The Computer Vision shall send a heartbeat to the watchdog service	Functional Testing
2.2	The Computer Vision shall publish raw images from front camera	Functional Testing
2.3	The Computer Vision shall publish raw images from bottom camera	Functional Testing
2.4	The Computer Vision shall receive images from front camera	Functional Testing
2.5	The Computer Vision shall receive images from bottom camera	Functional Testing
2.6	The Computer Vision shall detect and classify all task objects	Functional Testing
2.7	The Computer Vision shall publish bounding boxes of objects	Functional Testing
2.8	The Computer Vision shall provide distance from objects	Functional Testing
2.9	The Computer Vision shall calculate angle of incidence of objects	Functional Testing

Requirements related to 3. Controls		
Requirement Number	Requirement Task	Method for Testing
3.1	The Controls shall send a heartbeat to the watchdog service	Unit Testing
3.2	The Controls shall read and publish data from Bar30 barometer	Unit Testing
3.3	The Controls shall read and publish data from VN-100 IMU	Unit Testing
3.4	The Controls shall read and publish data from Teledyne DVL	Unit Testing
3.5	The Controls shall implement a PID Library	Unit Testing
3.6	The Controls shall generate PWM values that will move submarine to a desired position and/or orientation	Unit Testing
3.7	The Controls shall output PWM values to thrusters	Unit Testing
3.8	The Controls shall clench and release a mechanical claw on command	Unit Testing
3.9	The Controls shall shoot torpedoes on command	Unit Testing
3.10	The Controls shall release ball from dropper on command	Unit Testing

Requirements related to 4. Mapping		
Requirement Number	Requirement Task	Method for Testing
4.1	The mapping module shall send a heartbeat to the watchdog service	Unit Testing
4.2	The mapping module shall subscribe to all computer vision data	Unit Testing
4.3	The mapping module shall subscribe to Sonar data	Unit Testing
4.4	The mapping module shall implement a Kalman Filter	Unit Testing
4.5	The mapping module shall generate a map of environment	Unit Testing
4.6	The mapping module shall position all task objects in map	Unit Testing
4.7	The mapping module shall publish map	Unit Testing

Requirements related to 5. Localization		
Requirement Number	Requirement Task	Method for Testing
5.1	The localization module shall send heartbeat to the watchdog service	Unit Testing
5.2	The localization module shall subscribe to IMU data topic	Unit Testing
5.3	The localization module shall subscribe to Barometer data topic	Unit Testing
5.4	The localization module shall subscribe to both camera topics	Unit Testing
5.5	The localization module shall subscribe to DVL data topic	Unit Testing
5.6	The localization module shall subscribe to map topic	Unit Testing
5.7	The localization module shall position and orient submarine inside the map	Unit Testing
5.8	The localization module shall publish localization data	Unit Testing

Requirements related to 6. Watchdog		
Requirement Number	Requirement Task	Method for Testing
6.1	The watchdog module shall subscribe to all heartbeats from all modules	Unit Testing
6.2	The watchdog module shall shutdown submarine if major components fail	Unit Testing
6.3	The watchdog module may have fallback module configurations	Unit Testing

11. Glossary

Term	**Description**
S.L.A.M	Simultaneous localization and mapping
IMU	Inertial measurement unit
ROS	Robotic Operating System. An open-source framework that helps researchers and developers build and reuse code between robotics applications.
Localization	The process of determining where a mobile robot is located with respect its environment
DVL	Dopler Velocity Log
SMACH	State Machine
YOLOv4	YOLOv4 is a SOTA (state-of-the-art) real-time Object Detection model.
Darknet	Darknet is an overlay network within the Internet that can only be accessed with specific software, configurations, or authorization, and often uses a unique customized communication protocol.
OpenCV	OpenCV (Open-Source Computer Vision Library) is an open-source computer vision and machine learning software library. OpenCV was built to provide a common infrastructure for computer vision applications and to accelerate the use of machine perception in commercial products.
Google CoLab	Colab allows anybody to write and execute arbitrary python code through the browser and is especially well suited to machine learning, data analysis, and education.
LabelIMG	LabelImg is a graphical image annotation tool which allows you to draw visual boxes around your objects in each image, it also automatically saves the XML files of your labelled images.
PID	Proportional Integral Derivative
PWM	Pulse Width Modulation
ESC	Electric Speed Controller

12. References

(1) ROS: “Wiki.” *Ros.org*, http://wiki.ros.org/

(2) SMACH: FelixKolbe. “Wiki.” *Ros.org*, http://wiki.ros.org/smach/Documentation.

(3) SMACH Viewer: PlayFish. “Wiki.” *Ros.org*, http://wiki.ros.org/smach_viewer#Documentation

(4) BehaviorTree.CPP: *BehaviorTree.CPP*. www.behaviortree.dev.

(5) Groot: *Groot* | *BehaviorTree.CPP*. www.behaviortree.dev/groot.

(6) YOLOv4: “Yolov4 Tiny Object Detection Model.” *YOLOv4 Tiny Object Detection Model*, Roboflow Inc. , https://roboflow.com/model/yolov4-tiny.

(7) Darknet: Bochkovskiy, Alexey. “Home · Alexeyab/Darknet Wiki.” *GitHub*, GitHub, Inc., https://github.com/AlexeyAB/darknet/wiki.

(8) OpenCV Download: Linuxize. “How to Install Opencv on Ubuntu 20.04.” *Linuxize*, Linuxize, 5 July 2020, https://linuxize.com/post/how-to-install-opencv-on-ubuntu-20-04/.

(9) OpenCV: doxygen. “OpenCV Modules.” *OpenCV*, OpenCV, https://docs.opencv.org/4.x/.

(10) Google CoLab: Google. “Colaboratory.” *Google Colab*, Google, https://research.google.com/colaboratory/faq.html.

(11) LabelIMG: Heartexlabs. (n.d.). *Heartexlabs/labelimg: LabelImg is now part of the label Studio Community. the popular image annotation tool created by Tzutalin is no longer actively being developed, but you can check out label studio, the open source data labeling tool for images, text, hypertext, audio, video and time-series data.* GitHub. Retrieved December 9, 2022, from https://github.com/heartexlabs/labelImg

(12) VN-100: VectorNav “VN-100 User Manual.” V*ectorNav*, *https://www.vectornav.com/resources/user-manuals/vn-100-user-manual*

(13) Mur-Artal, R., & Tardós, J. D. (2021). ORB_SLAM3 [Source code]. GitHub. https://github.com/UZ-SLAMLab/ORB_SLAM3

(14) Mur-Artal, R., & Tardós, J. D. (2017). ORB-SLAM2: An Open-Source SLAM System for Monocular, Stereo, and RGB-D Cameras. IEEE Transactions on Robotics, 33(5), 1255-1262