Design and Implementation of Motion Control Autonomous Underwater Vehicle

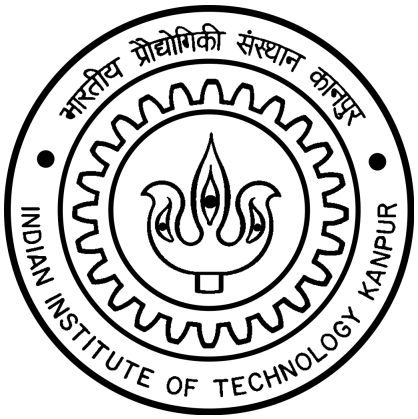
Prakhar Agarwal & Soumye Singhal

Indian Institute of Technology, Kanpur

Abstract

This report presents a concise description of the Motion Control Software employed for the functioning of the underwater vehicle *Varun* to perform various tasks. The main objective is to use the data from camera and sensors and then performing a manoeuvre using a robust algorithm and precise motion controls to successfully complete the required tasks. The software has been developed over the time not only to serve the bare minimum purpose but rather in a way to make it as easy as possible for use.

Keywords: AUV, IITK, ROS, OpenCV, Gazebo



Software Subsystem, AUV-IITK
INDIAN INSTITUTE OF TECHNOLOGY, KANPUR
Kanpur- 208016, INDIA
July 27^{th}, 2017

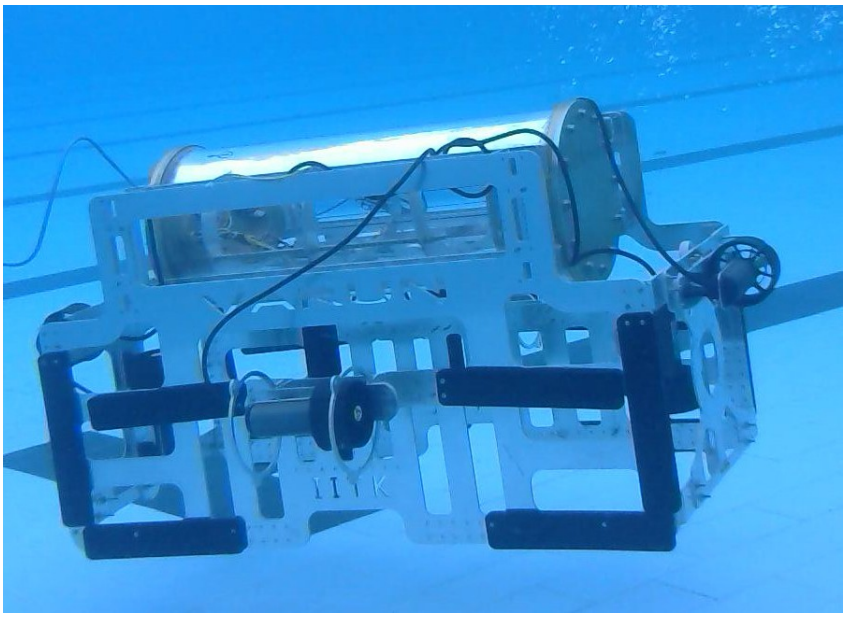
Contents

1 Introduction **1**
- 1.1 Line Detection . . . 1
- 1.2 Buoy Hitting . . . 1
- 1.3 Passing through a Gate . . . 2
- 1.4 Torpedo Shooting . . . 2

2 Implementation **2**
- 2.1 Sensors ie Hardware Layer . . . 2
- 2.2 Motion Library Layer . . . 3
- 2.3 Task Handler Layer . . . 3
- 2.4 Master Layer . . . 4
- 2.5 Debug Layer . . . 5
- 2.6 Software and Tools used . . . 5
  - 2.6.1 Robot Operating System a.k.a ROS . . . 5
  - 2.6.2 OpenCV . . . 5
  - 2.6.3 Gazebo . . . 6
  - 2.6.4 Other Tools and Software . . . 6

3 Challenges and Difficulties **6**

4 Future Work **7**



1. Introduction

Motion Control is an indispensable part of an autonomous vehicle as it is responsible for the movement of the robot underwater. The main objective is to take the input from the cameras after image processing and then performing the required tasks like detection, hitting, dropping by carefully analyzing the data from various sensors like arduino, imu and pressure sensor and then implementing the required controls to achieve the desired motion. To decrease the complexity of the task, we broke it down into levels of abstraction with the help of ROS nodes. We also have a master node which coordinates the task fo all other nodes so that the program works in a coherent manner.We explain the tasks at hand in the following sub-sections and then explain the how they lead to a similar solution.

1.1. Line Detection

This is the first of the task that needs to be handled, the robot needs to follow the lines to guide through the arena and for this it should be able to come in the center of the line present beneath it as well as align itself with the line. This requires precise motion controls to stabilize and align the robot with the center of the line.

1.2. Buoy Hitting

The task is to hit a circular buoy present at a definite height below the surface of water. The challenges involved are to localize the robot with respect to the center of the buoy using camera data and to move the robot steadily with rapidly fluctuating camera data. To make the system more robust we need to avoid the absurd motion due to noisy and sometimes false data obtained from camera due to improper detection and also need to handle the movement of the robot when the buoy is out of the camera frame.

1.3. Passing through a Gate

The task is to pass the robot through an L-shaped gate without hitting its' ends within a certain region as defined by the competition rules. The task was to locate the orientation(distance and angle) and the center of the gate from the robot's position and then performing the required manoeuvre.

1.4. Torpedo Shooting

The task at hand is to shoot the torpedo through a launcher when the robot is at a certain distance from 2 separately colored cupids such that it passes through the center of the targets.

2. Implementation

The task at hand is gargantuan and needs to be abstracted out into different modules. ROS is perfect for helping out in this abstraction. It divides each task into a node with specific purpose and a fixed structure. The input of data to sensors and output to arduino is handled by the Sensor nodes in hardware layer. The tasks of handling individual calibrated motion is carried out by nodes in the Motion Library layer. And finally in the final layer of abstration, we have the Task Handler Layer with nodes to implement individual tasks with the help of nodes that are lower in the hierarchy.

2.1. Sensors ie Hardware Layer

- Arduino - This node takes the PWM from the motion library and sends it to arduino to apply variable voltages on thrusters accordingly.
  - Subscribe data on four different topics for four different motions. And publish pressure sensor data on a topic.
  - When a PWM is subscribed on a topic, it's callback function is called. It first apply calibration on the PWM to get the individual PWM of each thruster, then send it to the thrusters.
  - In case of turn motion, it uses side thursters if robot is moving sideward also, else it uses thursters at end.
- Camera and Image Processing - This node takes the raw data from the two cameras (front and bottom) and publish them on two different topics for processing on task nodes. It is done for each frame at a certain rate.
- Imu - Publish the angle from the North axis.
  - Takes data from the navstik port.
  - Runs calculations to compute the angle from North axis.
  - Publishes the angle on a topic to motion library and task handlers.

2.2. Motion Library Layer

- Forward
- Sideward
- Turn
- Upward

  All the four motions use similar algorithm.

  - Wait until goal is received from the task handlers.
  - On receiving the goal, this procedure is followed until the completion or failure of the task.
  - Kill if server is not active or goal is cancelled.
  - Subscribe current position from the sensors and compute difference from final position.
  - Send difference as feedback.
  - Apply PID on the difference to get the output value.
  - Scale output value in the PWM range (0-255).
  - If PWM value is between band range for a certain number of times specified by goal, stop the motion, else publish the PWM to a topic.

2.3. Task Handler Layer

- Task Buoy - This node is to make the robot hit the buoy. It will take the goal from master layer.

  - Wait until goal is received from the master layer.
  - Wait for motion library server to start.
  - Start stabilization of turning motion by sending a goal of 0 angle to turn.
  - Send a goal to upward motion library to make the buoy at vertical center of the camera of robot. The data for current position is sent by Image Processing node of object detection.
  - Once robot is in line of buoy, it starts upward stabilization in similar manner and sends a goal to sideward motion library to make it in the horizontal center of the robot. The present position data is sent by Image Processing node to the motion library.
  - At this point, the robot and the buoy are in the same line. It sends a goal to forward motion library to move the robot forward until it hits the buoy.
  - Stop stabilization and forward motion.

- Task Gate - This node is to make the robot pass the gate. It will take the goal from master layer.
  - Wait until goal is received from the master layer.
  - Wait for motion library server to start.
  - Send a goal to upward motion library to make the gate at vertical center of the camera of robot. The data for current position is sent by Image Processing node of gate detection.
  - Start stabilization of turning motion by sending a goal of 0 angle to turn.
  - Send a goal to sideward motion library to make the gate at horizontal center of the camera of robot. The data for current position is sent by Image Processing node of gate detection.
  - When the robot is in the line of the center of the gate, goal is send to forward motion library to pass the robot from the gate.
  - Stop stabilization and forward motion.
- Task Line - This node is to detect and follow the lines. It will take the goal from master layer.
  - Wait until goal is received from the master layer.
  - Wait for motion library server to start.
  - Start stabilization of turning motion by sending a goal of 0 angle to turn.
  - Send a goal to forward motion library to move the robot forward to search line.
  - Cancel the forward goal when line is found.
  - Send sideward and forward goal to place the robot at the center of the line. The data for current position is sent by Image Processing node of line detection.
  - Stop turning stabilization.
  - Send a goal to the turn motion library to align the robot with line. The data for current position is sent by Image Processing node of line angle.

2.4. Master Layer

- Master Node - This is the main node which invoke all the tasks according to the arena. This node automates all the tasks.
  - Wait for all the task handler servers to start.
  - The procedure is followed until success or failure.
  - Send a goal to the task line to find and follow the line.
  - After finding line, it sends a goal to task buoy to hit the desired buoy.
  - It again search for line to follow, and hence sends another goal to task line.
  - Kill if line not found.
  - Sends a goal to task gate to pass the gate.

2.5. Debug Layer

- Gazebo Simulation - Simulate all the tasks in it.
- Remote GUI - Provide buttons and key bindings for motion.
- Image Calibration - Used to find the HSV values of the line and the buoy before the task starts.

2.6. Software and Tools used

The software developed at AUV-IITK is Open Sourced under BSD License and available on github at `https://github.com/AUV-IITK`. Implementing these techniques on their own from scratch can be a really difficult problem but we use the right set of tools to tackle the problem. The following software were used in it's development.

2.6.1. Robot Operating System a.k.a ROS

We use Robot Operating Systems (ROS) developed by Stanford A.I. Labs with open Licence to form the underlying framework of the Robot. It is a Collection of tools, libraries, and conventions that aim to simplify the task of creating complex and robust robot behavior. Has built-in messaging system that manages details of communication between distributed nodes via publish/subscribe mechanism. Easier to capture data published by some task (node) to a file, and then republish that data from the file at a later time.The list of main Ros packages used is :

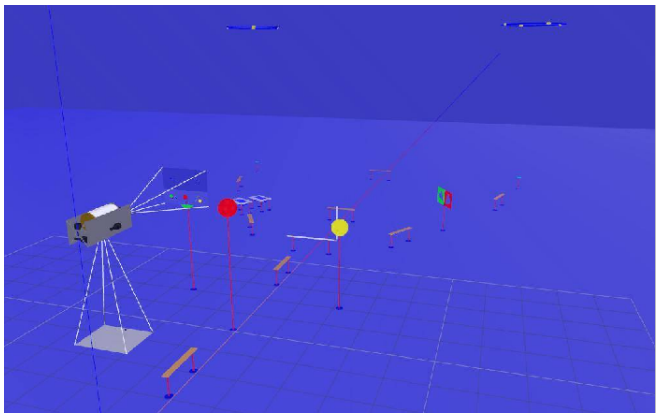
- Rosserial package to send output data to arduino.
- image_transport package to handle the input of data from the cameras.
- action_lib for communication between different layers and action (of nodes) handling.
- Dynamic Reconfigure to change the parameters used by various nodes during run time.
- Rosparam to change parameters before start like camera ports.
- Roslaunch file to automate the tasks of individual processes, saving a lot of time.
- rqt_reconfigure package for interactive graphical user interface (GUI) to operate robot and different parameters.
- Rosbag to store data of all sensors and topics during testing for later simulations.
- Roslint to maintain code formatting.

2.6.2. OpenCV

Further for the task of implementing the Computer Vision Code we use OpenCV which is an open source library for C++ that has various inbuilt algorithms for object detection. It can be easily be used along with ROS framework, essential for our purpose.

2.6.3. Gazebo

- Gazebo is a robot simulator that we used for the project.
- We designed the arena in the simulator and applied some thrust and drag forces to simulate the environment in which the bot will operate.
- This simulator will approximate real life situations which helps in debugging the code before going for actual testing in the pool.



2.6.4. Other Tools and Software

- Ubuntu (Linux) - Operating System for the processor(Intel NUC).
- Git - Version Control System more than one people can work on same code.
- Arduino IDE - To upload and run code on arduino.

3. Challenges and Difficulties

There is often difference between theory and practice. Working on the project, made us realize this fact time and again. We face many difficulties in implementing the algorithms so we had to do following things to make them work efficiently.

- Calibrate PID constants for motion library to make the motion smooth and fast.
- The thrust of thrusters were different hence the robot didn't move in the straight line. So we calibrated thrusters and plotted their data to get relation between there forces and PWM and then applied the PWM according to it.
- Added stabilization for steady motion.
- As the testing of algorithms can't always be done underwater, we also made a ground bot and tested our code on that.

4. Future Work

There is a lot of scope for improvement in the existing motion control system. The initial aim of the project of the project was to get ready a rudimentary motion control system that can get the work done. Not much focus was given to the automation part and efficiency of algorithms.

To make the robot truly autonomous we can apply various concepts of Machine Learning. For example one is improving detection tasks where one can apply machine learning to form a data-set and then using Convolutional Neural Networks and window scanning to get the position and orientation of the object to be detected from the required image. Although this might be tough due to the resources required to form a dataset.

Another idea is Inverse Kinematics using Machine Learning. Path planning gives us the trajectory of the most efficient path but to move on that path we have to fire up our actuators and accelerate in the right direction. This is done by Inverse Kinematics i.e. mapping displacement to actuator voltages. The task involves implementation of automatic calibration tools using machine learning algorithms like SVM for automatically calibrating these mappings using simulator for acquiring test data.

Apart from that we can also add filters to our sensor data to reject the noises in them. We can also improve our thruster calibration to make motion more smooth and accurate.

We can add some more sensors for more data like Doppler Velocity Log to get the velocity of each direction by which we can localize the objects with respect to the robot.