Project-I Report

on

Image Processing and Computer Vision Implementation in Underwater Environment

Bachelor of Technology

in

Computer Engineering

Submitted by

Ishpreet Singh Luthra (2K20/CO/207)

Under the Supervision

of

Ms. Chingmuankim

Department of Computer Engineering
Delhi Technological University
Bawana Road. Delhi -110042
December-2022

DECLARATION

I (Ishpreet Singh Luthra), student of BTech. (Computer Engineering), hereby certify that the work, which is presented in the Project-I entitled “Image Processing and Computer Vision Implementation in Underwater Environment” which is submitted by me and submitted to the Department of Computer Engineering, Delhi Technological University, Delhi is an authentic record of my own, carried out under the supervision of Ms. Chingmuankim. The work presented in this report has not been submitted and not under consideration for the award for any other course/degree of this or any other Institute/University.

Ishpreet Singh Luthra

2K20/CO/207

B.Tech,
Computer Engineering

SUPERVISOR CERTIFICATE

To the best of my knowledge, the report comprises original work and has not been submitted in part or full for any Course/Degree to this university or elsewhere as per the candidate’s declaration.

Place: Delhi

Supervisor name and signature

Date

Introduction

An autonomous underwater vehicle (AUVs) is a robot that travels underwater without requiring input from an operator. They are used to perform a variety of tasks, including marine research, hydrologic survey, environmental monitoring, construction, facility maintenance, and rescue. It also includes underwater target identification, communication, and underwater docking.

For the development of applications and algorithms for unmanned underwater vehicles (UUVs), such as autonomous underwater vehicles (AUVs), an appropriate simulation platform for speedy prototyping and simulation in reproducible virtual environments is required. This is desirable in all robotics applications, but it is especially crucial in underwater ones.

Underwater vehicles usually rely on expensive hardware and their target domain can be difficult to access depending on the application. Whereas several good open-source simulation environments exist for aerial as well for ground based robots, the scenario for underwater robots is less appealing. This is mostly due to the difficulties of accurately modelling not only the robot's hydrodynamic forces, but also the complex environments in which the operations take place. Since, robotics frameworks are commonly used to manage complicated robots, a framework-simulator integration is a useful tool for designing and verifying algorithms directly in frameworks.

The Gazebo and UUV simulator is already built into the Robot Operating System (ROS), demonstrating its value to the robotics community. Gazebo is a general-purpose open source robotics simulation platform maintained and developed by the Open-Source Robotics Foundation (OSRF). It provides interfaces to four different physics engines for the simulation of rigid-body dynamics and a graphical user interface to visualise the scenario.

Literature Review

We studied various research papers related to different components and features that are required to establish an underwater simulation environment.

In one research paper, we studied about Autonomous Underwater Vehicles (AUVs), how to design and model it and how it functions and what are the tasks that an AUV can perform.

In another research paper, we studied about the features of an underwater simulation environment, how it is created, what are the features and difficulties in it.

In another research paper, we studied a software called Gazebo which is used to set up a simulation environment quite easily. The research paper was especially focussed on the UUV package which is used to set up an underwater environment in the Gazebo software.

There was also a paper focussed on image analysis and image processing algorithms that are required to be performed in the underwater environment to get a more clear image and a more accurate output so that an AUV can function accurately.

There was a research paper that explained the entire working of an AUV in an underwater environment with its practical feasibility and success ratios along with the benefits.

Designing Autonomous Underwater Vehicles (AUVs)

The mechanical and electrical systems of the AUV, as well as its sensor equipment, have to be planned and considered simultaneously when building a new AUV prototype.

1. Mechanical design : two-hull, four-thruster configuration has been chosen because of different advantages.
   The design provides the following advantages:
   - ➢ Ease in machining and construction due to its simple structure
   - ➢ It also provide relative ease in ensuring watertight integrity because of the lack of rotating mechanical devices such as bow planes and rudders
   - ➢ Substantial internal space owing to the existence of two hulls
   - ➢ High modularity due to the relative ease with which components can be attached to the skeletal frame
   - ➢ Cost-effectiveness because of the availability and use of common materials and components
   - ➢ Relative ease foreseen in software control implementation in using a four thruster configuration when compared to using a ballast tank and single thruster system
   - ➢ Acceptable range of motion provided by the four thruster configuration
   - ➢ Ease in submerging with two vertical thrusters
   - ➢ Static stability due to the separation of the centers of mass and buoyancy, and dynamic stability due to the simple alignment of thrusters with easily identifiable centres of drag

   Both the mechanical and electrical systems are designed with simplicity and adaptability in mind. The mechanical designs are made symmetrical not just to make building it easier, but also to make software modelling easier. Also,the materials that are chosen should be common, cost-effective and easily machinable

2. Embedded Controller: The controllers primary purpose is controlling the vehicles movement and the AUVs sensors.
   Both speed and direction are controlled by motor controllers designed and constructed specifically for the thrusters. Two servo ports connect each motor controller to the Eyebot controller. Each motor controller generates a lot of heat due to the high current used by the thrusters. A heat sink attached to the motor controller circuit on the exterior hull

was created to keep the temperature inside the hull from rising too high and destroying electronic components. As a result, the water continuously cools the heat sink, keeping the temperature inside the hull at a comfortable level.

3. Sensor Integration: The goal of developing an active acoustic sensor system for the AUV is to construct an active acoustic sensor system that determines the distance between an AUV and an obstacle or landmark maximising cost effectiveness.
   To ensure the AUV's successful navigation of the surroundings using just sound, the following essential specifications for the sonar system were developed:
   - ➢ To be able to avoid an obstruction that may obstruct the AUV's path
   - ➢ To determine the distance between an obstacle and the AUV
   - ➢ Detecting landmarks that may aid in the orientation of the AUV
   - ➢ Mapping additional landmarks to aid in the AUV's localization

Software Used

The Gazebo and UUV simulator is already built into the Robot Operating System (ROS).The main requirements for this simulation environment is to make it compatible with ROS so that the applications developed for the vehicles can be tested seamlessly with it.

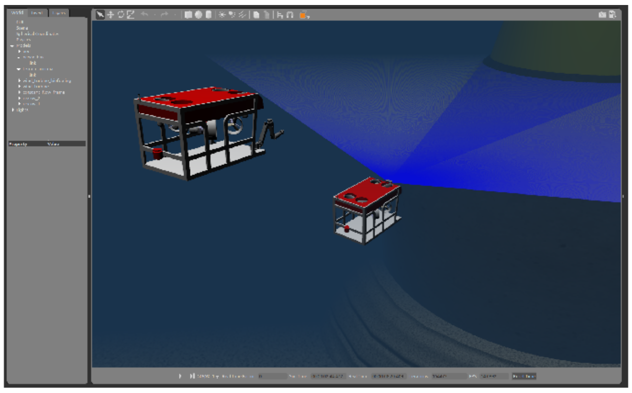
Vehicle system functions are implemented using ROS and currently provide simple dynamic positioning controllers. Thruster plugin model thruster allocation services, teleoperation modules (e.g. a joystick interface) and, in the case of robotic arms, Cartesian control and kinematics solver services. This set of modules provides a basic structure to interact with the simulated vehicles and the Gazebo simulation environment.

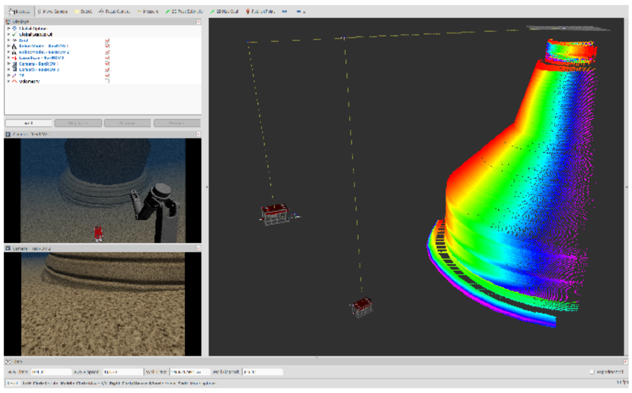
Gazebo Simulation Package

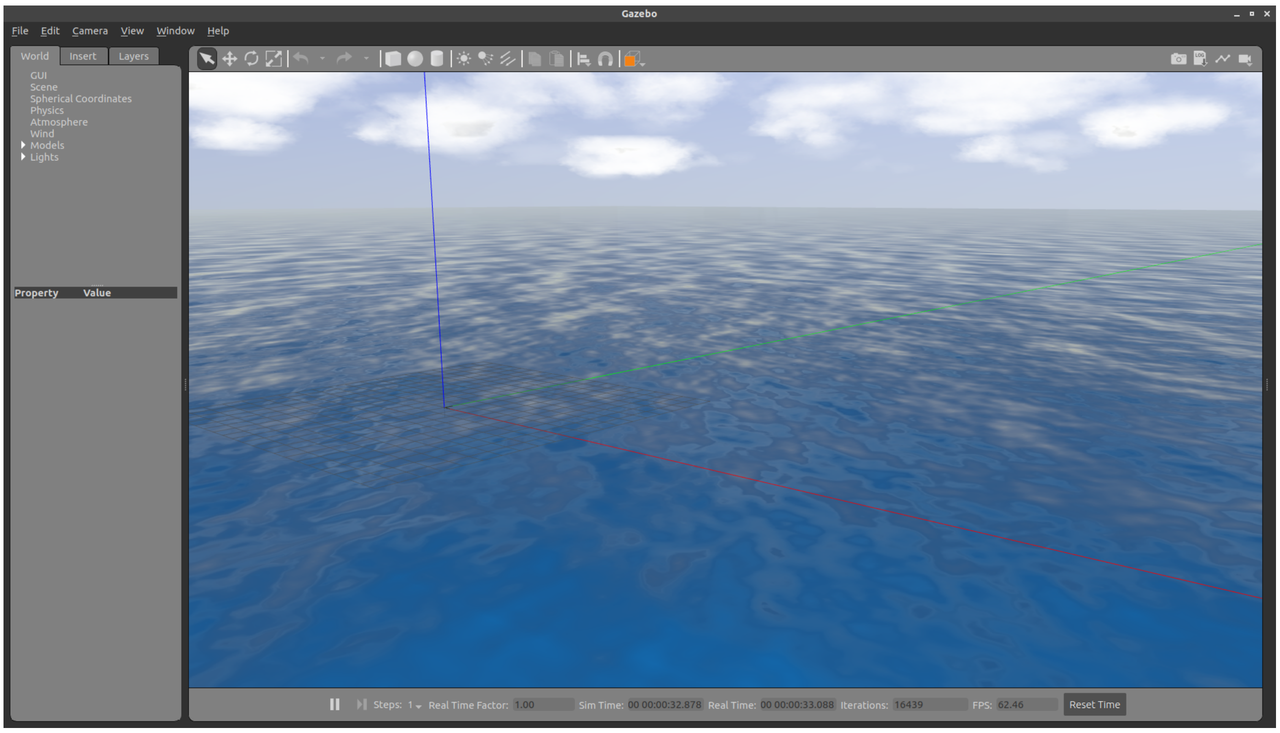
The Gazebo simulator gives the user the possibility to extend it with user-defined plugins. By using its API, the plugin has access to the simulation objects and data, can transmit information via topics by using Protocol Buffer3 messages and apply torques and forces to objects in the scenario.The plugins must be initialised with a robot, sensor or world models as described in their respective SDF4 file, an XML format designed for Gazebo. For applications using ROS, the robot descriptions are specified in URDF5, a similar file format.

Because of its integration with four physics engines, Gazebo can already simulate rigid-body dynamics, one of which must be initialised and specified within the world model description.

Hydrodynamic and hydrostatic forces and moments, as well as appropriate actuator and sensor models, must be considered in underwater situations. This is accomplished through the use of additional plugins included in the UUV Simulator package.







Steps: 1 Real Time Factor: 1.00 Sim Time: 00 00:00:32.878 Real Time: 00 00:00:33.088 Iterations: 16439 FPS: 62.46

A Simulation Environment for Unmanned Underwater Vehicles Development

The physical environment in which the robot must operate poses serious operational problems in the development of unmanned underwater drones. Using, operating, and recovering underwater vehicles is a time-consuming and costly task that requires highly qualified teams and complex logistics. Moreover, due to unpredictable and time-varying environmental conditions, performing repeatable tests at sea can be very difficult, if not impossible. Some tests can be performed in closed areas such as harbours and pools, but they still involve complex logistics and often time-consuming paperwork.

For these reasons, an accurate and comprehensive simulation environment to quickly design, develop, laboratory test and evaluate the functionality of underwater vehicles and minimise the risk of losing expensive equipment during dangerous field testing is required. This led to the development of a virtual underwater world. In this world, you can replace physical robots and operating environments with those simulated within the framework of hardware-in-the-loop simulation, completely transparent to robot control and sensor modules. The goal is to develop a single version of the robot software and use it in both real and virtual worlds. Thus, the real-time manipulation of the hardware inside the loop is combined with the execution of the simulation model. This goes beyond the concept of simulation as a preliminary proof of the concept. Of course, it is essential to use real robots and sensors to test each model.

At this stage the main technical issue is in guaranteeing the pragmatism of the simulation, which is essentially due to the precision of the mathematical models used and exact reproduction of sensor and sensorial information. Comparison between the simulation results and the real world measurements is an obvious way to see how well a simulated AUV mission matches a real-world mission.

Underwater Image Analysis

Image Processing

The process in which a digital image is processed by using a digital computer is called image processing. Computer or programming algorithms can be used to process images to alter or enhance images or to extract some useful information.

Steps involved in Image Processing

1. Using image acquisition tools to import the image.
2. Analysing and manipulating the image.
3. Producing an output in the form of some information or data or an altered image.

What is an Image?

F(x,y) is the 2-dimensional function used to define an image, where x and y are spatial coordinates. Intensity of an image at a point is given by the amplitude F at any given coordinates (x,y). For an image to be digital, the values of x, y and F need to be finite.

In technical terms, an image is basically a two-dimensional array specifically arranged in rows and columns.

A digital image is made up of a finite number of elements and each element has a particular value at a particular location. These elements are referred to as picture elements, image elements and pixels. Digital images are mostly measured in terms of pixels.

Types of an Image

1. Binary Image: In this type of image, there are only two pixel elements, i.e., 0 and 1 which refer to black and white colours respectively. Binary images are also called Monochromes.
2. Black and White Image: This type of image consists of only two colours, i.e., black and white.
3. 8-bit Colour Format: Commonly known as grayscale image, it consists of 256 different shades of colours where 0 refers to Black, 127 refers to Gray and 255 refers to white.
4. 16-bit Colour Format: Also known as High Colour Format, it consists of 65,536 different shades of colours. A 16-bit format is actually

divided into 3 further formats which are Red, Green and Blue which makes it the famous RGB format.

Image as a Matrix

As discussed earlier, matrices can be represented in the form of matrices. The following syntax is used to represent images as matrices.

$$f(x,y) = \begin{bmatrix} f(0,0) & f(0,1) & f(0,2) & \dots & f(0,N-1) \\ f(1,0) & f(1,1) & f(1,2) & \dots & f(1,N-1) \\ \vdots & \vdots & \vdots & & \vdots \\ f(M-1,0) & f(M-1,1) & f(M-1,2) & \dots & f(M-1,N-1) \end{bmatrix}$$

Every element of this matrix is called image element, picture element or pixel.

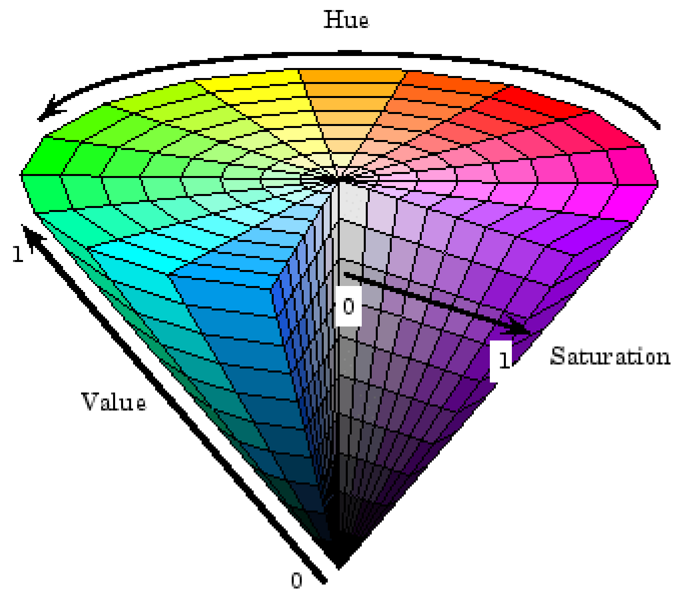
Colour Thresholding

It is a process used to extract a particular shade of a colour from a coloured image (like a photograph taken using a camera). Thresholding of an image converts a coloured image into a binary image in which the target colour becomes the foreground and the rest of the shades go into the background. Thresholding is done using the hue, saturation and value (HSV) of each pixel.

The HSV Model

In the RGB model, only colours can be filtered but in the HSV model, not only colours can be filtered but one can adjust the intensity of the required colour.

- Hue: It measures the colour of each pixel.
- Saturation: It measures the intensity of each pixel.
- Value: It measures the brightness of each pixel.



A normal coloured image is in RGB format which can be converted to HSV format in opencv using the following command:

```
cvtColor(img, imgHSV, COLOR_BGR2HSV);
```

HSV colour space in image is a three dimensional matrix in which each matrix represents each hue, saturation and value.

Example of conversion of RGB matrix to HSV matrix:

RGB Matrix

1.0000	0	0
1.0000	0.5000	0
1.0000	1.0000	0
0	1.0000	0
0	0	1.0000
0.6667	0	1.0000

HSV Matrix

0	1.0000	1.0000
0.0833	1.0000	1.0000
0.1667	1.0000	1.0000

0.3333	1.0000	1.0000
0.6667	1.0000	1.0000
0.7778	1.0000	1.0000

In colour thresholding, one colour needs to be extracted. Each shade of colour lies in specific ranges of hue, saturation and value. Using trackbars (sliders), the range of values for the required colour is determined.

```
Scalar lower (hmin, smin, vmin);

Scalar upper (hmax, smax, vmax);

inRange(imgHSV, lower, upper, mask);
```

Using the above commands, HSV image is converted to an image called mask image. Mask image is a binary image in which the required colour is in foreground as white and remaining colours as black.

The inRange function converts all the values between lower and upper to white and the remaining values to black to create the mask image.

Contour Detection

After obtaining the mask (binary) image, its edges are detected and sharpened to make the process of drawing its shape easy.

To define forms and shapes present in an image, curved and straight lines are drawn. These lines are called contours.

The mathematical principle behind the algorithm is based on vectors. As all the edges are detected, vectors are drawn from 1 point to its adjacent point until the shape is completed.

Simulation

(Code used for the simulation)

```
#include <iostream>

#include "opencv2/highgui/highgui.hpp"

#include "opencv2/imgproc/imgproc.hpp"
```

```
#include "opencv2/imgcodecs/imgcodecs.hpp"

using namespace cv;

using namespace std;

Mat imgHSV, mask, imgGray, imgBlur, imgCanny, imgDil, imgErode;

int hmin=0, smin=57, vmin=0;

int hmax=9, smax=255, vmax=101;

double left_length, right_length;


void getContours(Mat imgDil, Mat img)

{

   vector<vector<Point>> contours;

   vector<Vec4i> hierarchy;

   int a,b;

   double l1, l2;


   findContours(imgDil, contours, hierarchy, RETR_EXTERNAL, CHAIN_APPROX_SIMPLE);

   int j=0;


   for (int i=0; i<contours.size(); i++)

   {

       int area= contourArea(contours[i]);

       //cout<< area<< endl;


       vector<vector<Point>> conPoly(contours.size());

       vector<Rect> boundRect(contours.size());


           double peri= arcLength(contours[i], true);

           approxPolyDP(contours[i], conPoly[i], 0.02*peri, false);

           drawContours(img,conPoly, i, Scalar (0, 255, 0), 2);

           boundRect[i] = boundingRect(conPoly[i]);

           rectangle(img, boundRect[i].tl(), boundRect[i].br(), Scalar(0,255,0), 5);
```

```
        //Point P= boundRect[i].x;
        //circle ( img, boundRect[i].tl(), 5.0, Scalar(0,0,255), 2, 8 );


        if (j==0)
        {a=boundRect[i].tl().x;
        l1 = peri/2;}
        if (j==1)
        {b=boundRect[i].tl().x;
        l2 = peri/2;}
        j++;
  }
  if (a<b)
  {
    left_length=l1;
    right_length=l2;
  }
  else
  {
    left_length=l1;
    right_length=l2;
  }
}


int main()
{
  string path = "/home/ishpreet/opencv_codes/Gate_Robosub/Gate.png";
  Mat img = imread (path);
  //cout<<"hello: "<<img.empty()<< endl;
  imshow("Original", img);


  cvtColor(img, imgHSV, COLOR_BGR2HSV);
```

```
Scalar lower (hmin, smin, vmin);

Scalar upper (hmax, smax, vmax);

inRange(imgHSV, lower, upper, mask);


//imshow("Image", img);

//imshow("Image HSV", imgHSV);

imshow("Image Mask", mask);

imwrite ("mask.png", mask);

imwrite ("threshold.png", imgHSV);


string path2 = "/home/ishpreet/opencv_codes/Gate_Robosub/mask.png";

Mat img_mask = imread (path2);

cvtColor(img_mask, imgGray, COLOR_BGR2GRAY);

GaussianBlur(imgGray, imgBlur, Size(3,3), 3, 0);

Canny(imgBlur, imgCanny, 25, 75);

Mat kernel= getStructuringElement(MORPH_ELLIPSE, Size(6, 6));

dilate(imgCanny, imgDil, kernel);

Mat kernel2= getStructuringElement(MORPH_CROSS, Size(5, 5));

erode(imgDil, imgErode, kernel2);


//imshow("Image Mask", img_mask);

//imshow("Image Gray", imgGray);

//imshow("Image Blur", imgBlur);

//imshow("Image Canny", imgCanny);

//imshow("Image Dil", imgDil);

//imshow("Image HSV", imgHSV);

//imshow("Image Mask", mask);

//imshow("Image Erosion", imgErode);


getContours(imgDil, imgHSV);
```

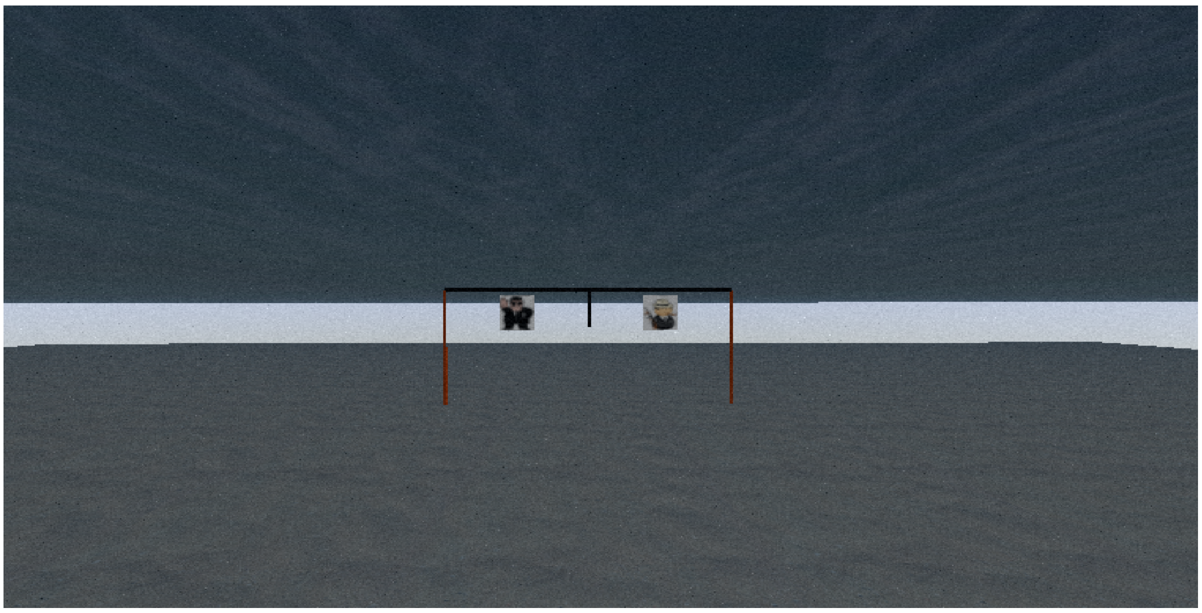
```
    imshow("Image with Contours", imgHSV);

    imwrite("final.png", imgHSV);

    imwrite("erosion.png", imgErode);



    waitKey(0);

    return 0;

}
```

Algorithm

Firstly a standard Mat image is inputted as ‘img’. Then colour thresholding is applied to obtain the ‘Mask Image’ named as ‘mask’ using the algorithm as explained above. Then some processes are applied to remove unwanted detected items and sharpen the edges to make contour detection as precise as possible. Firstly the mask image is converted to a grayscale image and then it is blurred a bit. Then Canny Edge Detector is used to identify and detect the edges of the shape. The image is dilated to make its features more prominent and erosion does the opposite. The values of both of these operations need to be set correctly to get a clear image in which contours can be detected. Then the contour detection is done using the algorithm explained above. It gives us the final output.

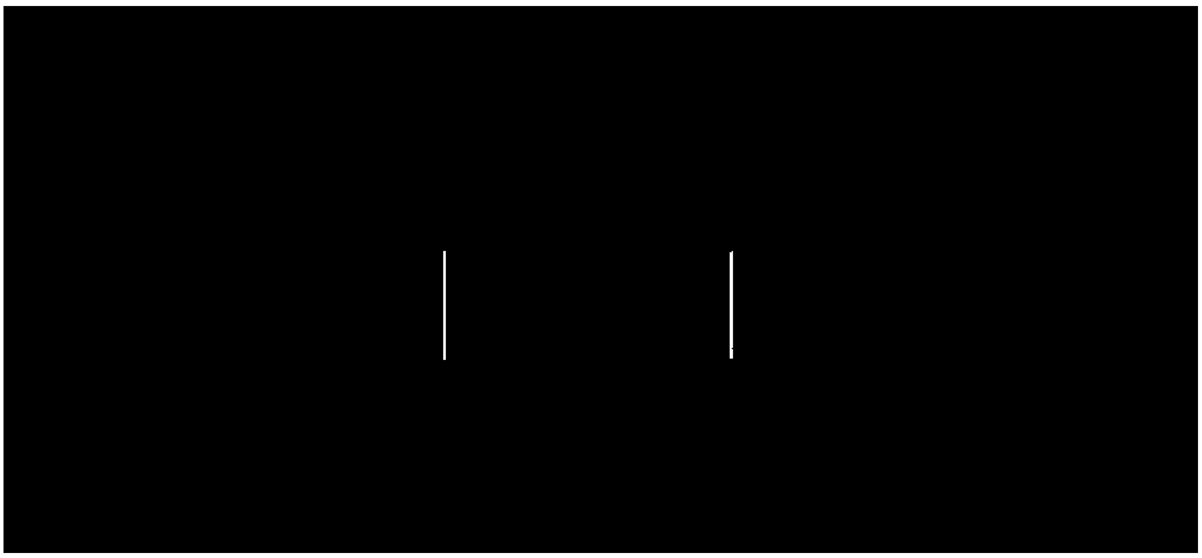
Input:

Original Image

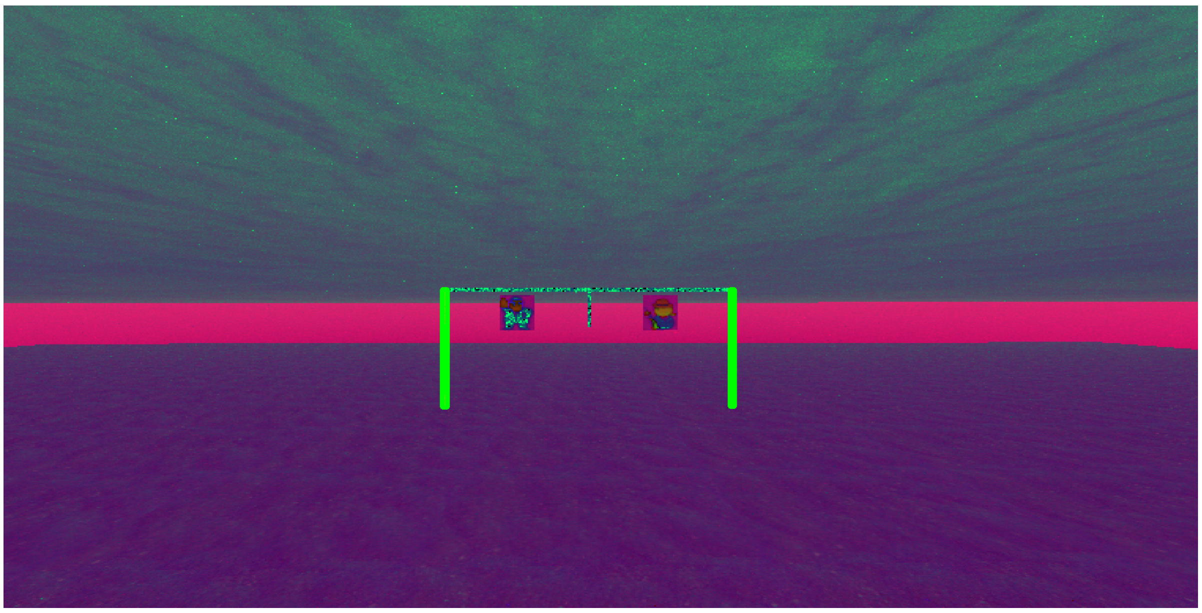


Output:

Mask Image



Final image with contours (green)



The above code and algorithm was performed on Gazebo simulation software in Underwater environment (as given in the problem statement of a Robotics Competition). In the simulation, an Autonomous Underwater Vehicle is searching for the gate as shown in the 'Original Image' by rotating 360 degrees about its z-axis (yaw controls). When the gate appears in the field of view of the camera, the vehicle aligns with the gate translationally as well as rotationally by detecting the orange coloured ends of the gate using colour thresholding. The length of the edges is detected by calculating the perimeter of contours and halving them. Relative difference between the lengths of 2-sides depends on the angle of the view. This relative difference is used to align the vehicle rotationally and translationally so that the vehicle moves towards the centre of the gate.

Link to Simulation video: Video of Simulation of above code - Click here

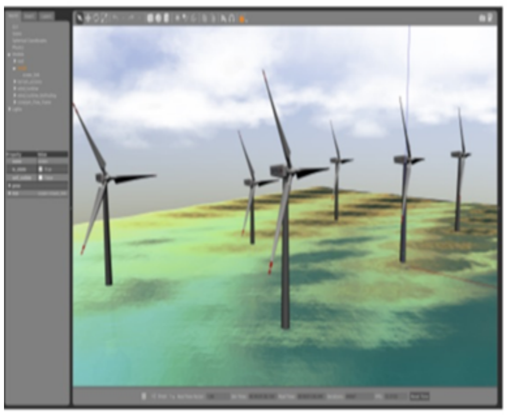
CONCLUSION

A package for simulating autonomous underwater vehicles is presented in this project. It adapts Gazebo, a general-purpose robotics simulator, for use in underwater situations and offers a ROS interface for controlling the simulated vehicles.

The Gazebo API implements vehicle, actuator, and sensor models, which can be extended to other robotics middlewares if needed.This package enables for the dynamic initialization of environment and robot models in runtime, reducing the time and effort required to test, build, and develop new applications and mission strategies.

In the research, we found out how AUV's are built and designed, how an underwater environment can be set-up for simulations, how the image analysis works under the water, how the AUV navigates underwater, how to set-up underwater simulation environment using the UUV package of Gazebo software, etc.

FUTURE WORK



The lack of characteristic visual effects such as floating particles and adequate light dampening as a function of water depth, as well as the formation of waves on the sea surface, is one of the most noticeable features missing in this underwater robotics simulator for the Gazebo environment.

Implementing such effects with the help of Gazebo's graphics engine, OGRE, is now in the works, as it is critical in the simulation of visual servoing applications and the control of teleoperated vehicles with visual input.

This figure shows an example of the current stage of wave integration in Gazebo.

The addition of an umbilical plugin to better ROV simulation is also a desired feature. Although umbilical forces can be simulated, contact forces on the tether will necessitate the geometrical modelling of cables in Gazebo, which is currently unavailable in this platform.

REFERENCES

[1] M. M. M. Manh˜aes and S. A. Scherer and M. Voss and L. R. Douat and T. Rauschenbach, “UUV Simulator: A Gazebo-based package for underwater intervention and multi-robot simulation”, 2016, Link

[2] Thomas Bräunl, Adrian Boeing, Louis Gonzalez, Andreas Koestler, Minh Nguyen, “Design, Modeling and Simulation of an Autonomous Underwater Vehicle”, International Journal of Vehicle Autonomous Systems (IJVAS), 2006, Link

[3] Bruzzone Ga., Bono R., Caccia M., Veruggio G., “A Simulation Environment for Unmanned Underwater Vehicles Development”, MTS/IEEE Oceans 2001. An Ocean Odyssey. Conference Proceedings, Link

[4] G.R. Fournier and M. Jonasz, “Computer based underwater imaging analysis”, SPIE Conference on Airborne and In-Water Underwater Imaging, Denver, Colorado, July 1999, Link

[5] J. Antich, A. Ortiz, “An Underwater Simulation Environment for Testing Autonomous Robot Control Architectures”, IFAC Conference on Computer Applications in Marine Systems - CAMS 2004, Ancona, Italy, 7-9 July 2004, Link

[6] N. Carlevaris-Bianco, A. Mohan, and R. M. Eustice, “Initial results in underwater single image dehazing”, In Proc. MTS/IEEE OCEANS, 2010, Link