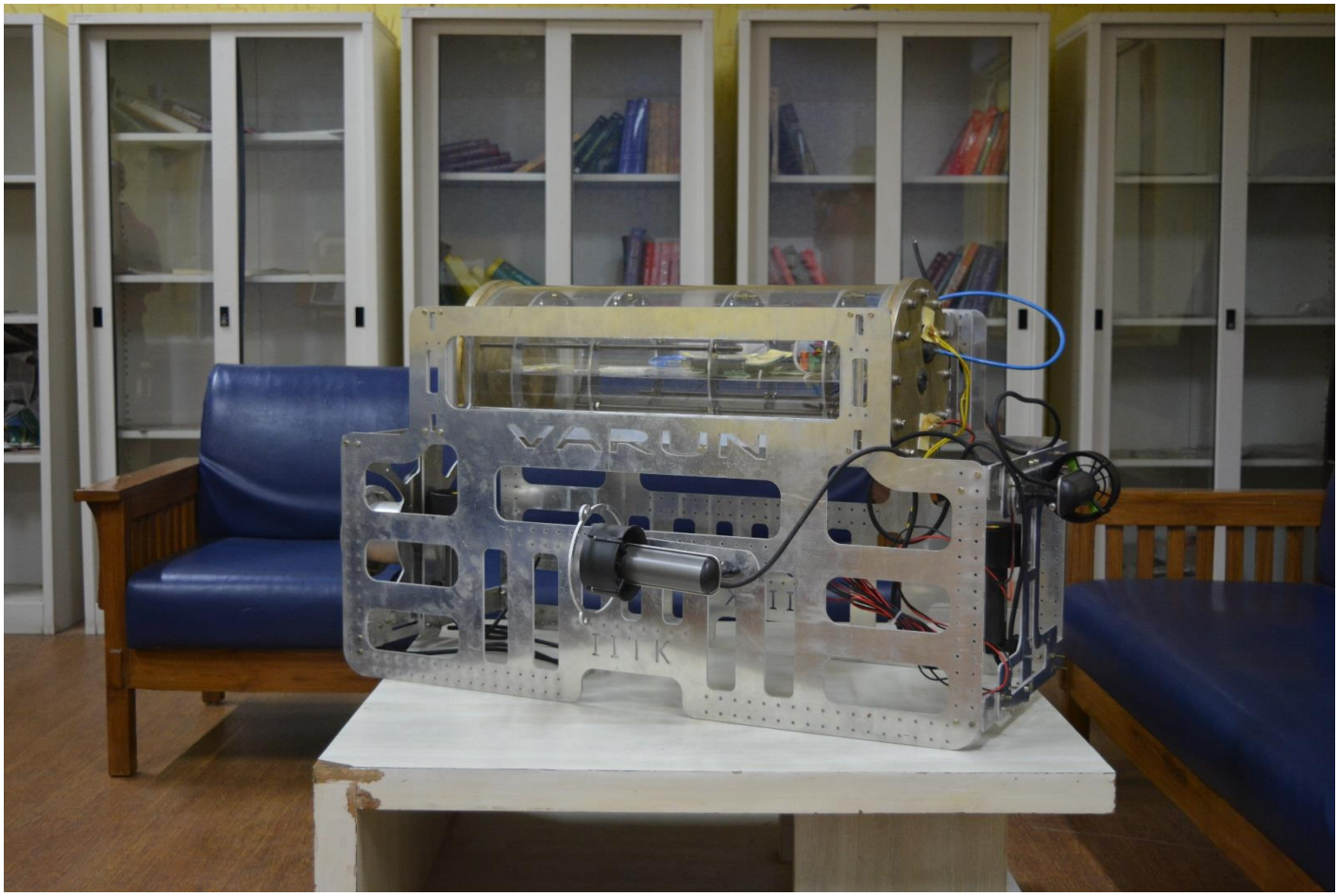
INDIAN INSTITUTE OF TECHNOLOGY KANPUR

Autonomous Underwater Vehicle

VARUN



Conceptual Design Report

Faculty Advisor: Professor K. S. Venketesh

Team Members

Divyanshu Patel, Mohit Pandey, Mayank Mittal, Satyam Diwedi, Shibhansh Dohare, Jayant Agarwal, Shikher Verma
Abhishek Attal, Abhishek Shastry, Sagnik Bhattacharya, Rithvik Patibandla, Kumar Anupriya, Prannay Khosla,
Raktim Mitra, Harsh Vardhan, Samarth Kapoor, Siddharth Saxena, Prabhav Kumar Gaur,
Prakhar Agarwal, Aditya Chopra, Aman Singhal

Abstract

'*Varun*' is the Autonomous Underwater Vehicle currently being developed by a team of undergraduate students from the Robotics Club in IIT Kanpur. Although Varun is our institute's first underwater vehicle, we aren't setting our expectations low. After a year of designing and running several experiment based prototypes, the vehicle is now on its verge of completion to participate in the 5th NIOT-SAVe organized by the National Institute of Ocean Technology, Chennai.

The vehicle will consist of a main pressure hull along with separate enclosures for the sensors, and pneumatic valves. Varun has been equipped with two cameras to provide it with vision, an inertial measurement units for navigation and orientation, and a depth sensor to enable it conquer greater depths. Varun shall also have the ability to drop markers and fire torpedoes underwater, as and when required.

In conclusion, the vehicle will have the autonomous capability of completing the several tasks by the time of the competition, more about which shall be commented upon its final testing in the summers of 2016.

Mechanical

Varun's mechanical system comprises of an aluminum structural frame, pressure casings, and electromagnetic actuators. The mechanical parts were first designed in Solidworks and Autodesk Inventor and then improved using Ansys Workbench, before its final fabrication using in-house available facilities at the institute. The weight of Varun is currently 27.510 kg.

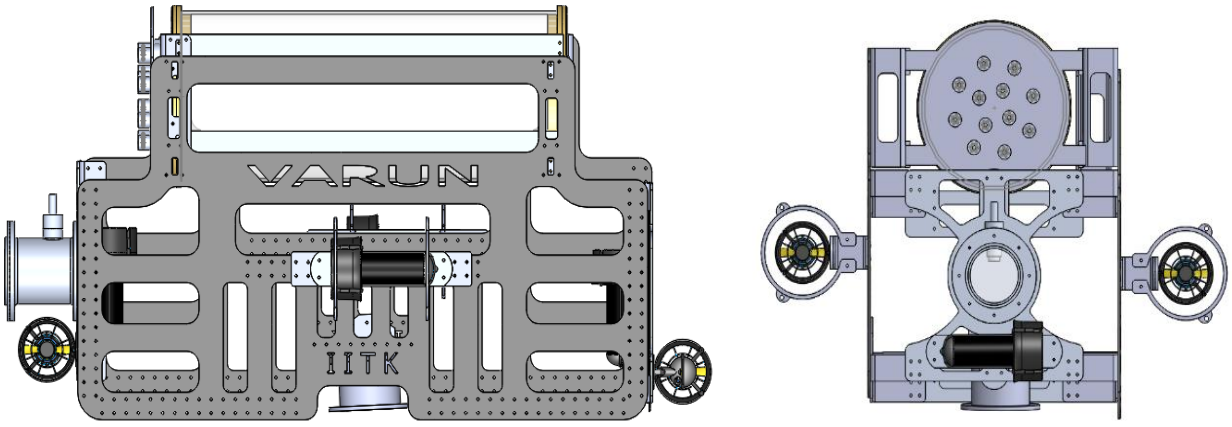


Figure 1: *Side and Front View of Varun*

Vehicle's Frame

Varun's frame mainly consists of two aluminum sheet side plates, to which several structural supports and parts are joined. The side plates have been optimized to reduce the weight of the side plates without creating any critical stress points. The distance between the two side plates is approximately 52 cm, ensuring a sturdy shape to the vehicle.

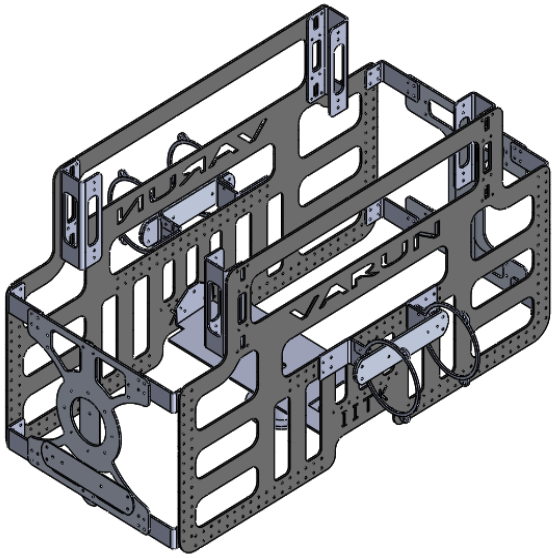


Figure 2: Varun's Frame

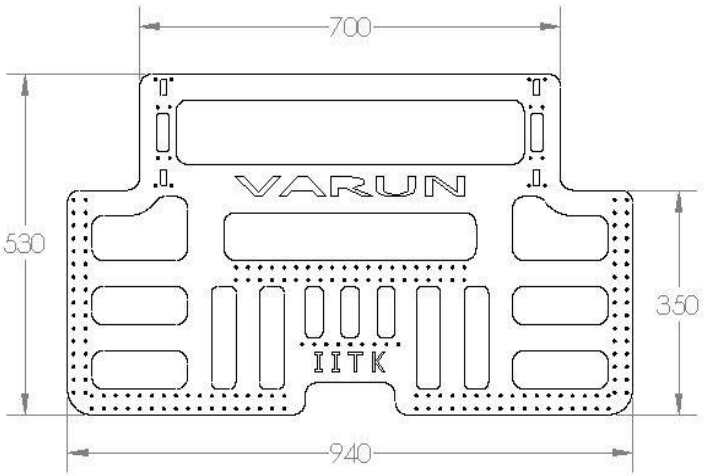


Figure 3: Side Plate Design

Main Pressure Hull

A crucial part of Varun, the pressure hull provides the housing for all the electronic components of the vehicle such as the micro-controllers, micro-processor and other circuitries. Fabricated at Concorde Engineering at Bangalore, India the hull is made out of an acrylic cylindrical tube so that the contents are visible from outside, with aluminum end flanges. The visibility from outside would help troubleshooting from mere inspection if mishaps occur while testing.

Figure 4: Main Pressure Hull

Customized Waterproof Casings

Independent casings have been designed for the front and lower cameras, as well as for the solenoid valves. Similar to the pressure hull, face seal using O-rings of appropriate size is being used to make the casings waterproof. The camera casings have an aluminum tube welded to the respective structure plate. The front end of the enclosures is covered by an acrylic lid for clear visibility.

Figure 5: Front Camera Casing

Figure 6: Lower Camera Casing

Waterproof Connector

In order to connect the electronic components inside the pressure hull with Varun's thrusters, pneumatic and sensor systems; the connectors provide a leak-free interface which fulfils our purpose.

Considering the limitation of resources available in Kanpur and to fulfill the need, custom designed waterproof connectors, a total of twelve, have been made which would be installed on the main pressure hull's front flange. The connectors use two bore seals for waterproof sealing this year, having removed a face seal from its last prototype.

Using the in-house available three-dimensional printing facility, the mounts for USB connectors have been fabricated while the electrical connectors for the supply cables are being brought from the local market.

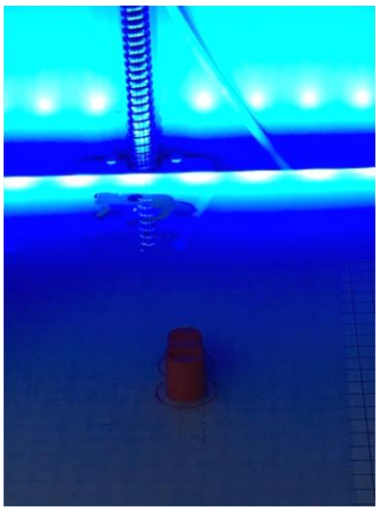


Figure 7: *3D Printing*

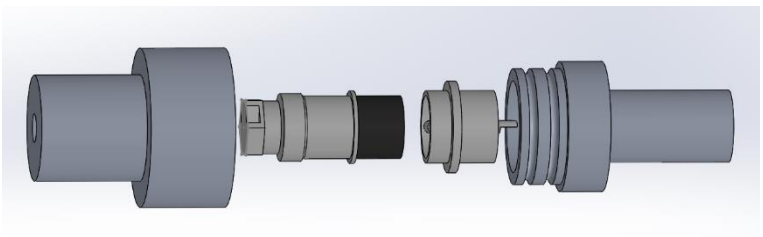


Figure 8: *Assembled Connector and its*

Hull Internal

The three rack, hull internal tray provides the area for placing all the electronic components inside the hull to be comfortably, ensuring sufficient space for cooling and running wires throughout the hull. It consists of four annular disks which are fixed inside the hull using tie rods. The circuit plates can slide smoothly into the grooves made in the disc for them.

The top-most shelf shall have an LCD display mounted on it which will provide the required data from the various electronic systems. The middle shelf contains the processing unit that is the micro-controller, micro-processor etc. while the bottom shelf carries the battery supplies for Varun. A separate shelf for battery ensures easy replacement at the time of testing without having to disturb the other two shelves.

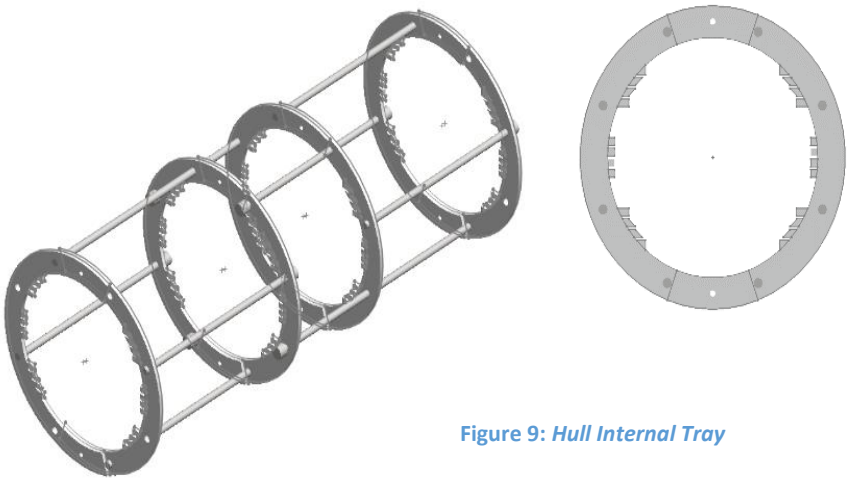


Figure 9: *Hull Internal Tray*

Thrusters

Varun uses six AUV/ROV BTD-150 thrusters from SeaBotix® Inc. The actuators have been positioned in a way which provides the vehicle a total of five degrees of freedom. The two thrusters mounted on the side plates ensure translation motion while the other four thrusters' assembly provide sway and vertical motion along with yaw and pitch. According to the data collected for the thruster calibration, a 100W thruster develops over two kgf forward thrust and up to two kgf reverse thrust.

Figure 10: *SeaBotix® BTD-150*

Figure 11: Tiberius Guerrilla Paintball Tank

Pneumatics System

The pneumatic system is responsible for shooting torpedoes, and dropping markers. Varun currently consists of assembly capable of firing up to two torpedoes and releasing two droppers. The pneumatic system for Varun consists of the following three basic components:

Storage: A 3000 psi Tiberius Guerilla paintball tank provides the storage unit for the compressed air which serves as the air supply for the entire system. This high pressurized gas being supplied to the various outlets at 100 psi, using a regulator valve on the tank.

Supply**:** Supply consists of feed lines which are being used only for transportation of air throughout the system along with the various valves, specific to each task. Solenoid valves are being used as they are easy to manipulate using signals sent from the micro-controller. Janatics 5/2 double acting solenoid valve is being used for the marker dropper while 2/2 double acting solenoid valve is being used for shooting torpedo.

Cylinders**:** Two Janatics double acting pneumatic cylinders have been used for the dropper mechanism.

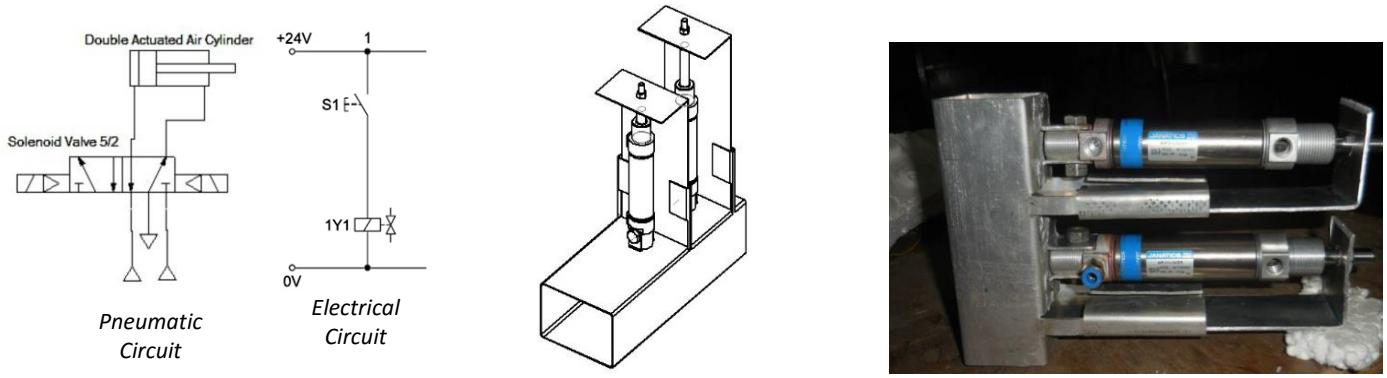


Figure 12: *Pneumatic Marker Dropper Assembly and circuit diagram*

Torpedo: The torpedoes are channelized through acrylic tube passage using pressurized air. Each torpedo is manufactured using the in-house available 3-D printer. The material used in manufacturing is polylactic acid (PLA), a biodegradable thermoplastic. These torpedoes have been ensured to have a streamlined design using Ansys Fluent analysis, providing it a range of around 7-8 feet when fired underwater.

Figure 13: Torpedo Prototype

Design Optimization

Ansys was used for all simulation and optimization purposes after creating the initial basic models using SolidWorks. Several iterations of designing and analysis cycle has led to Varun's current design. Fluent and Static Structure were the two main tools which were used from the Ansys Workbench.

Structural analysis studies the effects of loads on physical structures and their components. To optimize the parts using this tool, the curves and shapes in the part's geometry were first parametrized followed by the optimization paradigm for given parameters in Ansys. A minimum factor of safety of 10 has been ensured during the process to account for the stresses that weren't feasible to account for and safety reasons.

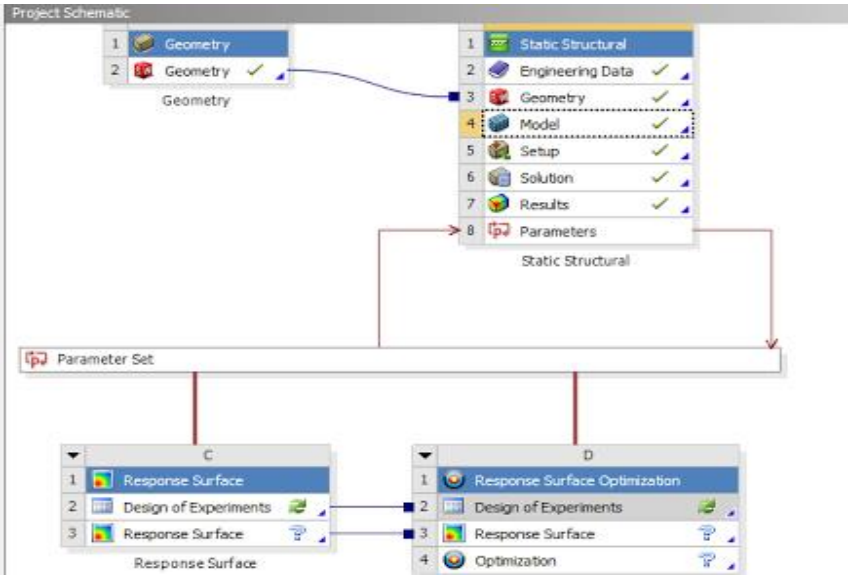


Figure 14: *Project Schematic Layout in Ansys*

Ansys Fluent solves the Navier–Stokes equations for fluids using finite element method (FEM) with a cell-centered formulation. Although SolidWorks CFX could have been used for the analysis as well, Ansys provided more flexibility. From the data collected, the present coefficient of drag of the vehicle in forward direction is 0.42.

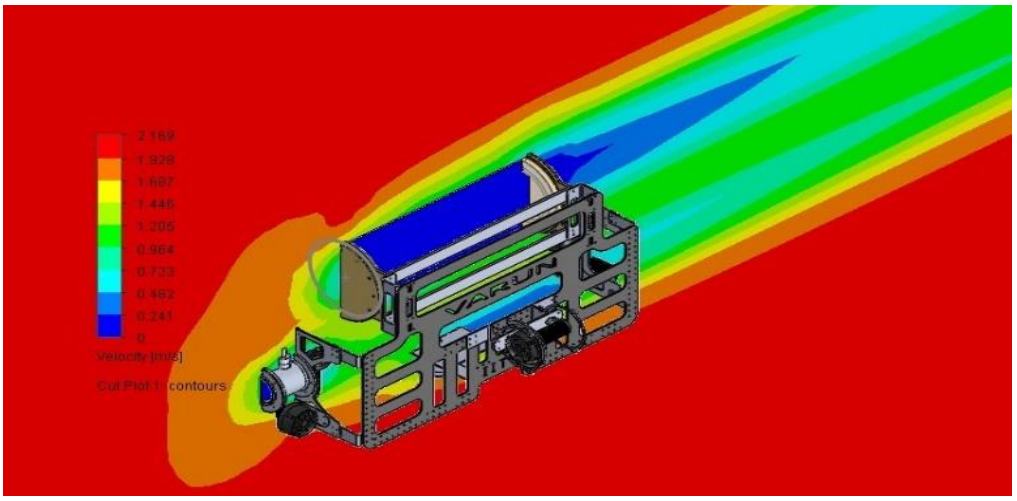


Figure 15: *Distribution of Velocity on a vertical plane bisecting the vehicle*

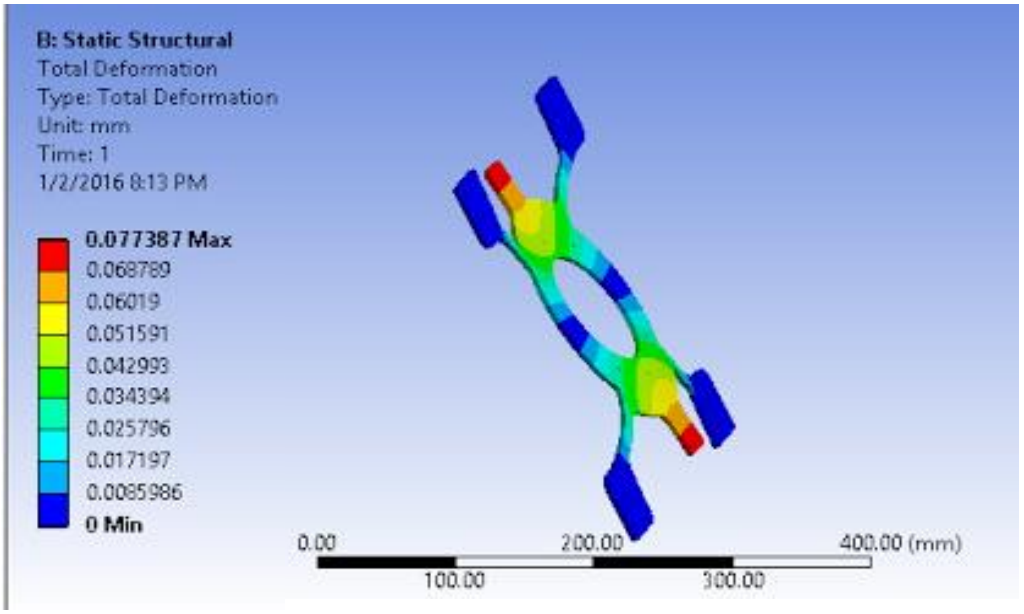


Figure 12: *Stress analysis on Front Camera assembly*

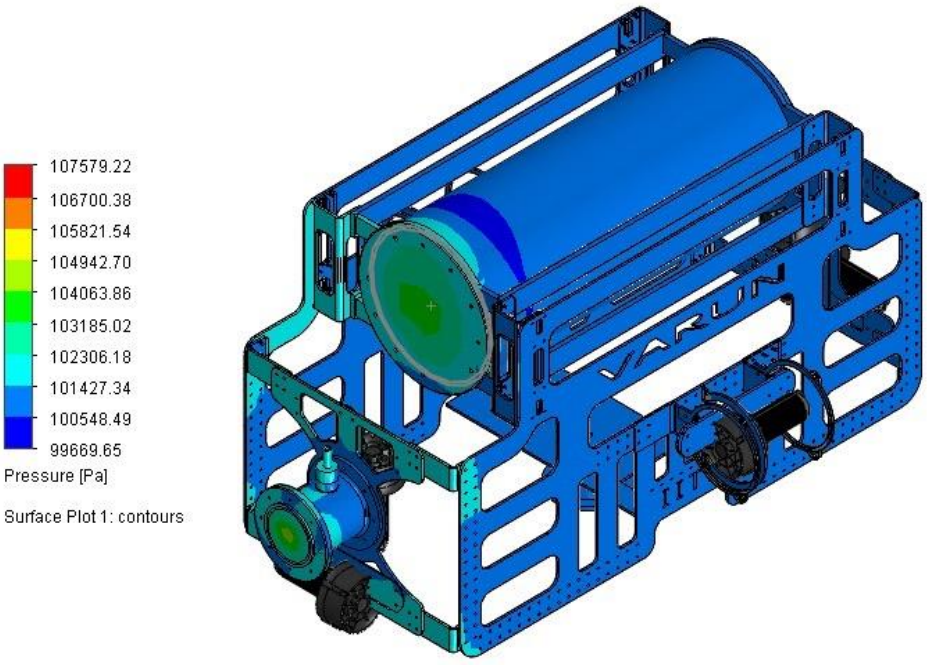


Figure 17: *Pressure variation in Varun underwater*

Electrical

Varun's Electrical systems provide the vehicle power and an interface between processor and other electronic devices. We will be designing our own PCBs for some of the boards. There will be 3 layers of stacks inside the hull which will be used for mounting different electronic devices.

Arduino

It is based on ATmega 1280 microcontroller and has an operating voltage of 5V. We are using Arduino Mega that has 16 Analog input pins and 54 digital I/O pins (of which 15 provide PWM output). The open source Arduino platform is used for processing the input signals from main processor and converting it into desired signals to be given to the actuators like motor drivers and pneumatic system.

Figure 18: Arduino Mega

Battery Management System

Varun uses a 14.2V (4 cell - 5200mAh) Lithium polymer battery to power its components. Power in Varun is distributed in two lines of 5V and 12V, capable of supplying 3A and 40A at maximum load. DC-DC converters are used to supply and regulate these lines and a monitoring circuit is implemented that monitors the voltage and current drawn from the battery and cut-off the lines if current drawn from the battery is too large or if the battery voltage falls below a certain level. Current measurement is performed using a Hall Effect based current sensing module. An LCD is used to show all the battery related information and alerts to the end user. All this is controlled using an eight bit microcontroller that is dedicated solely for battery management tasks.

We are also planning to implement a cell balancing system that ensures that each cell of the battery discharges at a rate similar to other cells to ensure longer battery life and an on-board battery charging system so that we can charge the battery without having to take it out each time.

Processor

Odroid is the main processor that will host our Operating System (Ubuntu 14.04) and will act as the brain of the Robot. It will act as an interface for all the sensors and after processing the input, send the output to the micro-controller which will control the actuators. With the use of ROS (Robot Operating System), image processing and a lot of real time computing, a strong processor is required. Odroid U3, manufactured by Hardkernel, is an excellent choice for this purpose. It is an extremely compact processor, the dimensions being 83 x 48 mm. Its Key features include:
1.7 GHz Quad core ARM cortex-A-9 MP Core Samsung Exynos 4412 Processor, 2GB RAM, 3 x High speed USB 2.0 ports 10/100 Mbps Ethernet with RJ-45 LAN jack.

We use L- Ubuntu 14.04 LTS as our operating system. On top of the L-Ubuntu runs the Robot Operating System, a Meta operating system designed specifically to link different hardware components of a robot. ROS provides the system a proper architecture and is an excellent tool to coordinate between different tasks. Odroid also come with an UART port that operates on 1.8V and grants direct access to the system control monitoring for development.

Figure 19: Odroid X2

Inertial Measurement Units

Any vehicle running underwater faces different fluid forces which cause a change in the state of the body. These changes are usually very small but always have a risk off adding up together and cause major problems, especially in autonomous systems. IMUs are devices which keep track of the orientation of the body. They usually consist of accelerometers and gyroscopes and sometimes even magnetometers. NavStik is a generic navigation and control board with a wide range of navigation sensors (accelerometers, gyroscopes and magnetometer) and a powerful processor. There are a number of interface-boards available for NavStik, each optimized for a different set of parameters. For our vehicle we will be using IvyPro board. In addition to this, it features a Mali-400 Quad core 440 MHz 3D Accelerator which reduces the processing time of the camera outputs.

Figure 20: NavStick IMU

The sensors in the Navstik base board are include 3-axis Magnetometer, 3-axis Accelerometer, 3-axis Gyroscope and a barometer. All these sensors have user selectable ranges and the barometer also consists of a temperature sensor allowing for temperature compensation algorithms.The Base board also has an ARM Cortex-M4 168Mhz processor, mainly used for data filtering. It also has GPIO pins for directly connecting actuators to the board. It runs on an open source Real Time Operating System and has an excellent compatibility with Linux. The Navstik Board comes with an on-board Micro-SD card support which comes very handy for data-logging in test phase. The IvyPro board has 6 PWM I/O pins, Telemetry port, a USB port, UART port, IvyGS port and various in-built protections against short-circuiting and other power anomalies. The size of this board is just 41mm x 35mm and with the Navstik main board it weighs just 13 grams. Navstik runs on Pandapilot or ChibiOS. We have chosen Pandapilot because of its simplicity and well developed community support system. All the call on Pandapilot to Navstik is made using the NuttX plugin. This plugin allows us to access the console of the Navstik directly from our main processor.

Pressure Sensor

We will be using Measurement Specialties US381-000005-030PG analog pressure sensor which works in range of 0-30psig (Pound per Square Inch Gage) and gives current output between 4mA-20mA. Maximum Depth at which our vehicle has to maneuver is 39ft (approximately 12m) so maximum gage pressure which we need to measure is 17psig. However the sensor available with Measurement Specialties either gives output from 0-15psig or 0-30psig. So for more accuracy we are using 30psig pressure sensor. The analog value transmitted by pressure sensor is read by depth measuring board which converts this pressure into depth and communicates this depth to control unit via serial communication.

Figure 21: US381 Pressure Sensor

Motor Drivers

We will be using Nex-Robotics NR-MDR-012 motor driver. It is a dual motor driver whose operating voltage is 8-28V and it can deliver 5A peak current to each motor. The motor driver has inbuilt current sense mechanism for each motor, that is if output current exceeds 6.5A the motor driver will automatically shut down. Also if temperature exceeds 150 degree Celsius the peak current is restricted to 4A and if temperature exceeds 170degree Celsius the motor driver will automatically shut down.

Figure 22: Nex Robotics Motor Drivers for Thrusters

Software

The Software Architecture of Varun that we are planning to work with is based on the Robot Operating System (ROS) Software Framework from Willow Garage which encompasses the underlying messaging infrastructure for inter-process communications in our distributed system.

ROS (Robot Operating System)

ROS is an open-source, meta-operating system for our robot. It provides the services we would expect from an operating system, including hardware abstraction, low-level device control, implementation of commonly-used functionality, message-passing between processes, and package management. It also provides tools and libraries for obtaining, building, writing, and running code across multiple computers. For simulation and testing our algorithms quickly we will use Gazebo which is a 3D indoor and outdoor multi-robot simulator, complete with dynamic and kinematic physics, and a pluggable physics engine. Integration between ROS and Gazebo is provided by a set of Gazebo plugins that support many existing robots and sensors. Because the plugins present the same message interface as the rest of the ROS ecosystem, we can write ROS nodes that are compatible with simulation, logged data, and hardware. We can develop our application in simulation and then deploy to the physical robot with little or no changes in our code.
Given below is the high level architecture of our ros package:

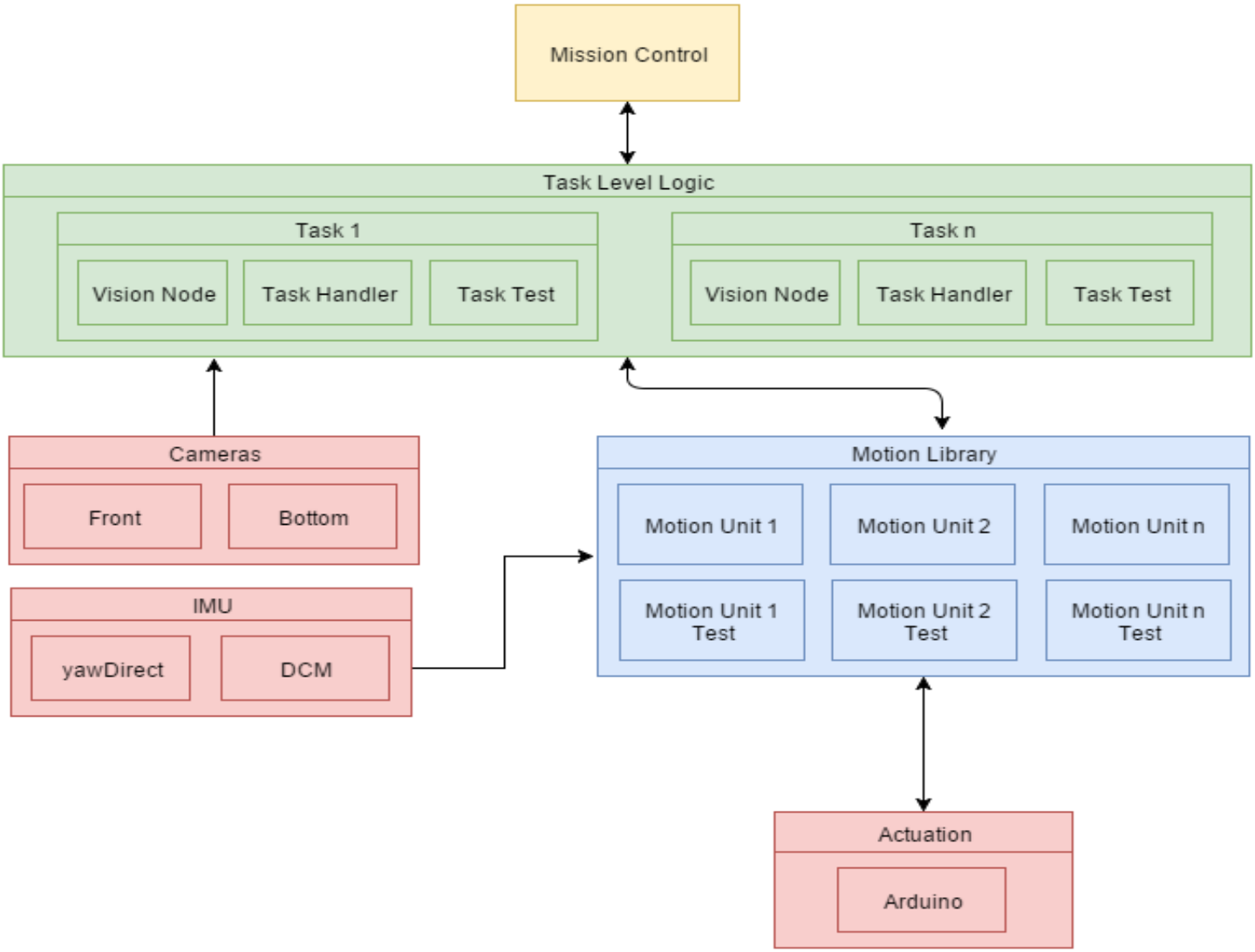


Figure 23: ROS Architecture

Sensor input nodes - The nodes will have the hardware specific code for data capture. They will abstract sensor details from the rest of the system. Core - The nodes that do the general functions of the robot. Path planning and mission control being some of the primary functions. Image Processing - Nodes for recognition of predefined shapes for the specific tasks. Output - nodes that control the actuation of the robot and converting high level movement instructions to low level pwm instructions for each thruster.

Actual hierarchy of packages in ROS Our system currently consists of **4 levels of logic**.

At the highest level we have the **motion control** package which manages the logic deciding the order to do tasks. It also approximates the position of robot in the environment

At the second level we have **task dependent logics**. We have a separate package dedicated for each task. Each package has different ROS nodes. Each package has a "*task_handler*" node that communicates with rest of the system, this node also contains the logic for the motion of the robot with respect to the current position. All the packages have a "*vision*" node that read the data published on one of the "*image*" topics. It is this node where all of the image processing in done. This node gives the output that will be required by the "*task_handler*" to decide the further motion of the robot.

At the third level we have a **motion library** that contains ROS nodes which implements unit motions in different directions, for example a node that moves the robot a given angle in X-Y plane. The "*task_handler*"s of different packages of the 2nd level logic gives the order to these nodes. These nodes then gives the final output to the arduino.

On the lowest level we have the **hardware level implementation** for the actuation. Here we have the arduino that takes the input from "*motion_library*" and finally outputs the required voltages on the output pins.

Other than this we have 2 packages. One is **imu** this take raw data from our imu, then implements different algorithms and gives the values of accelerations in all directions and the values of yaw, pitch and roll. The second package "*camera*" has two nodes that takes the input from the camera and publishes on ROS topics **image**. This allows multiple image processing nodes to be run simultaneously.

ROS-Accessories

"rqt_graph" provides a GUI plugin for visualizing the ROS computation graph. It shows the nodes running in the system and the topics that are being used for communication.

For example, In the ROS Graph shown below all the ellipses represents the nodes and each arrow is a topic.

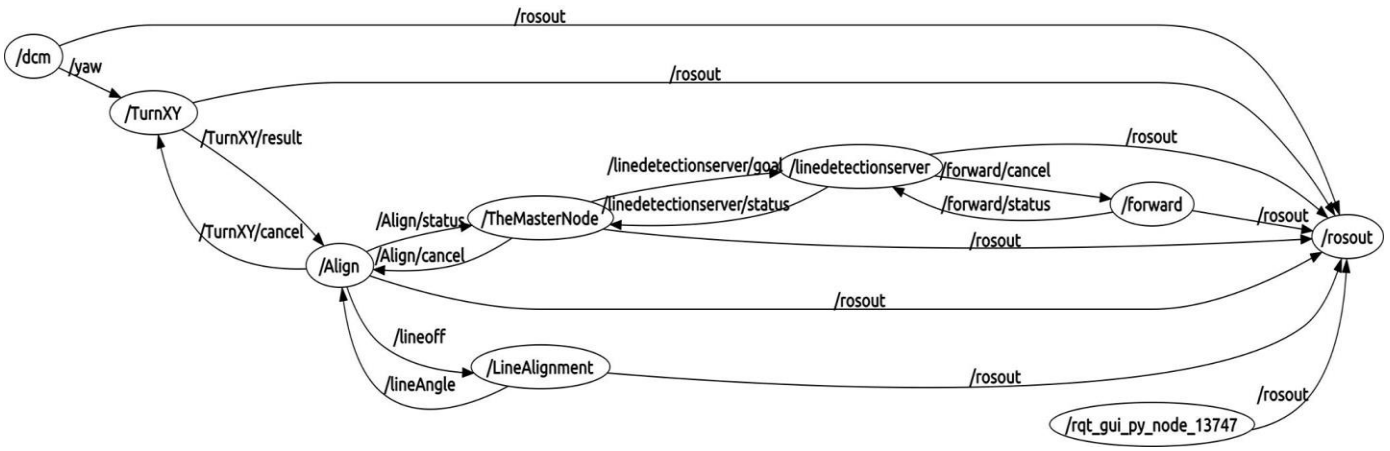


Figure 24

ROS BAG

This is a set of tools for recording from and playing back to ROS topics.

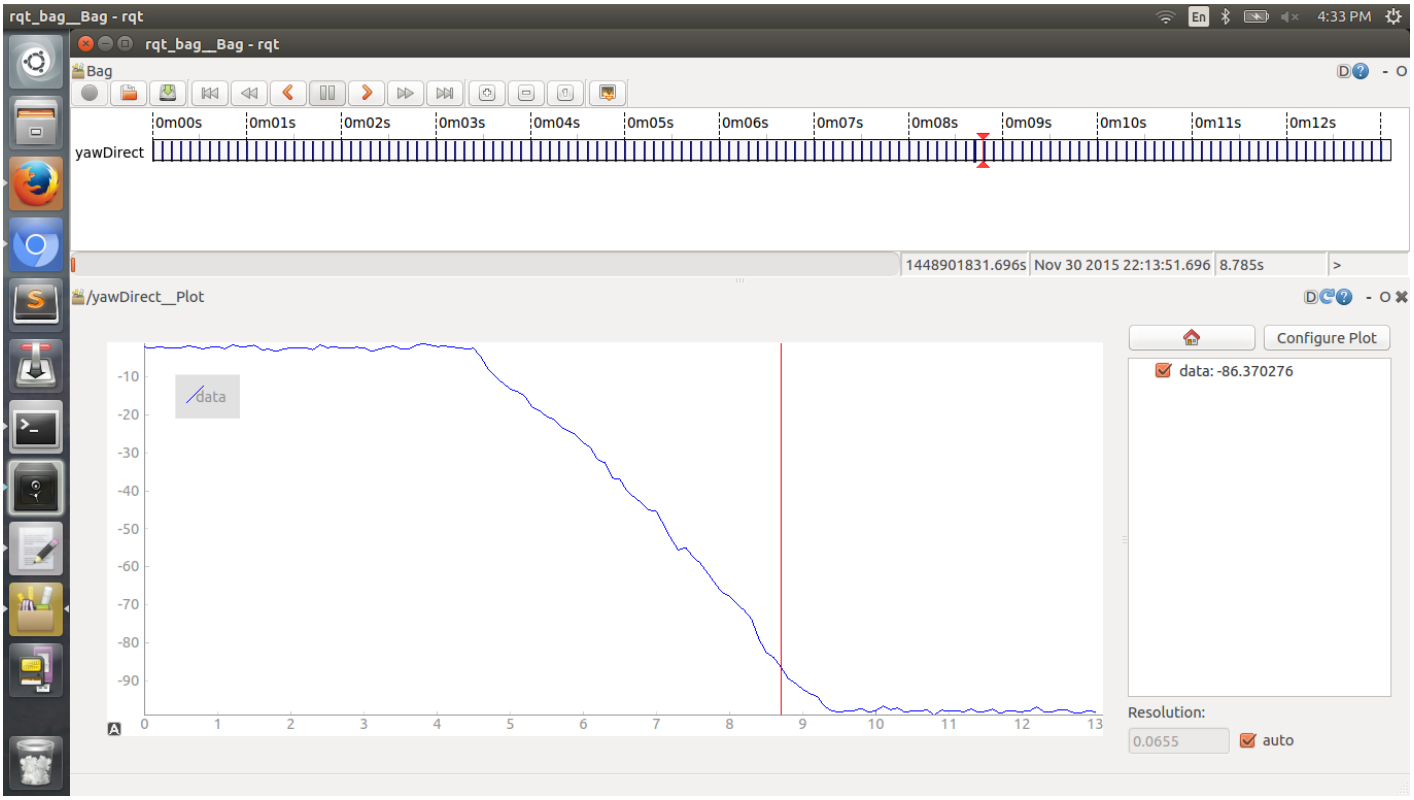


Figure 25

GUI

This is a rqt plugin. It provides the way to view and edit the parameters that are accessible via *dynamic_reconfigure*.

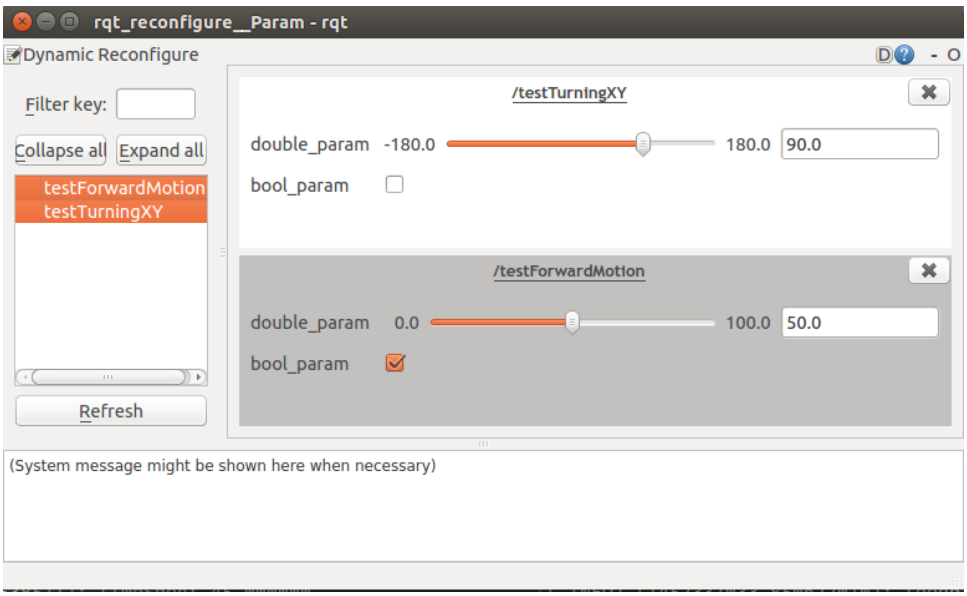


Figure 26

Vision

CAMERA

There are two cameras installed in the vehicle. Their position along with the tasks for which they will provide live video feed are as follows:

1. **Front Facing Camera** :
    a. Buoy Detection
    b. 'L' shaped Bar Crossing
    c. Torpedo Firing
2. **Bottom Facing Camera** :
    a. Line Detection and Following
    b. Marker Dropping

Figure 27: The camera being used is Logitech C270.

SOFTWARE INTERFACE

There are separate ROS nodes for both the cameras. The nodes will be responsible for collecting the video feeds and publishing them to topics. These will then be subscribed by the task-specific nodes where the vision algorithm is implemented. The relevant outputs after processing will be send to the task specific processing nodes which after processing data from other sensors will send the actuation signals to the control nodes.

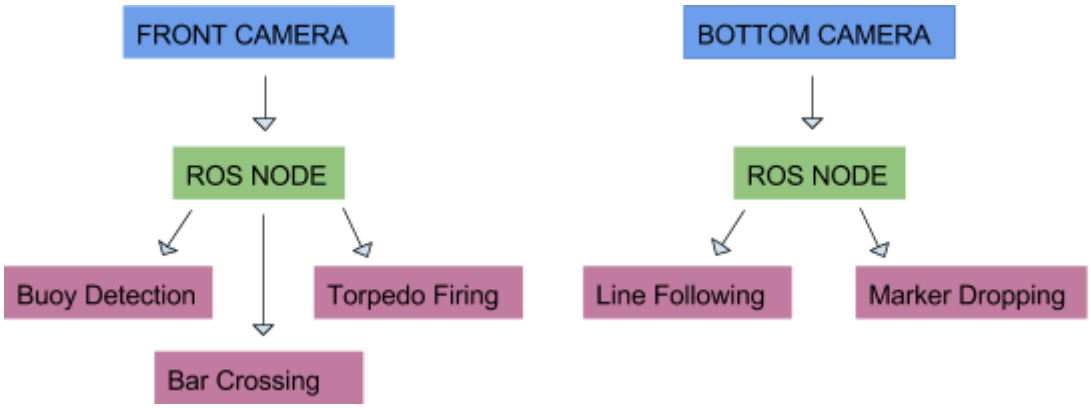


Figure 28

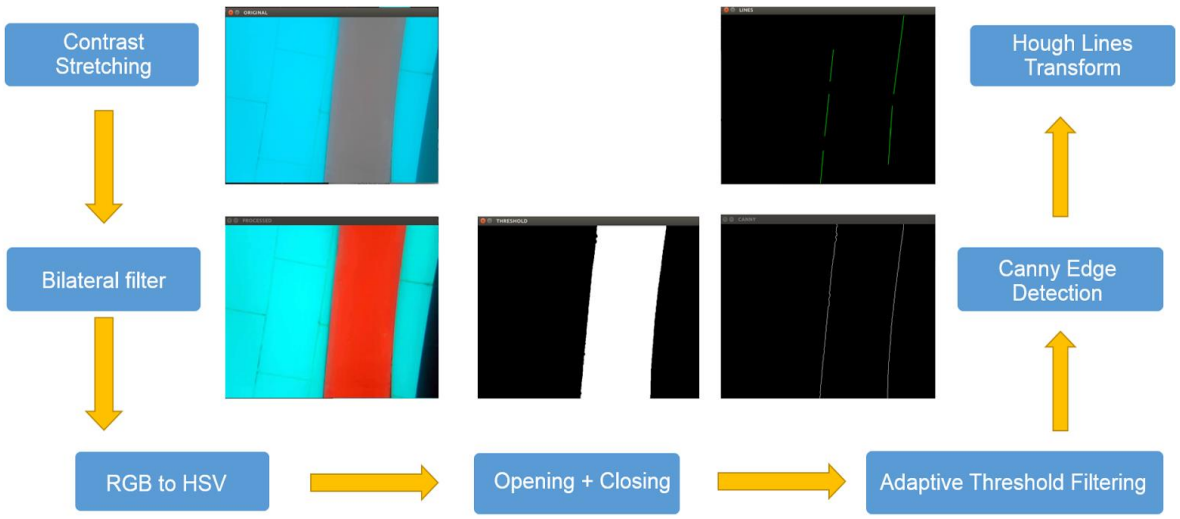
VISION ALGORITHM

The implementation of the algorithm for various tasks is implemented in C++ using *OpenCV-2.4.9*. OpenCV libraries offer inbuilt implementations for image pre-processing and object-detection algorithms.

The entire process is divided as:

1. **Pre-processing**: This process filters out any noise that might occur due to changes in water and lighting conditions. It essentially includes all the image enhancement techniques such as contrast stretching, Image sharpening, white balancing etc.
2. **Foreground Detection**: This process is used to separate the foreground and the background and to identify the region of interest (roi). Every part of the image that is important w.r.t to the task is brought to the foreground and the rest goes into the background. This is done with simple HSV thresholding where a range is given to determine if a pixel is positive or negative. After this process we get a binary image with the foreground.
3. **Task Specific Code**: During this stage, the task specific algorithm is implemented. This is an important stage as the actuation of the vehicle depends on the output obtained from these algorithms. All the Feature Detection and Object Detection Algorithms are implemented here.

The current algorithm being used for Line Following Task is as follows:



The image is first normalized across three RGB channels to enhance contrast lost in underwater environment and to make it easier to differentiate the foreground from the background. To smoothen out the noise that may be there in the image, it is fed to a filter. Filters usually do not only dissolve the noise but smooth away the edges. To avoid this, a bilateral filter is being used, which also takes into account the difference in intensities of the neighboring pixel and the evaluated one.

The *bilateral filter* (1) is defined as:

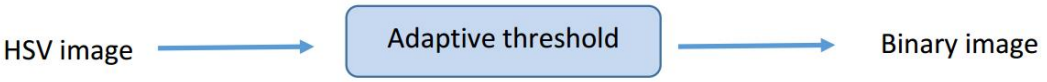
$$I^{\text{filtered}}(x) = \frac{1}{W_p} \sum_{x_i \in \Omega} I(x_i) f_r(\|I(x_i) - I(x)\|) g_s(\|x_i - x\|),$$

where, the normalization term:

$$W_p = \sum_{x_i \in \Omega} f_r(\|I(x_i) - I(x)\|) g_s(\|x_i - x\|)$$

ensures that the filter preserves image energy and I^{filtered} is the filtered image; I is the original input image to be filtered; x are the coordinates of the current pixel to be filtered; Ω is the window centered in x; f_r is the range kernel for smoothing differences in intensities. This function can be a Gaussian function; g_s is the spatial kernel for smoothing differences in coordinates. This function can be a Gaussian function.

The image is then converted into an HSV image and after applying Opening (erosion followed by dilation) and Closing (dilation followed by erosion), the image is thresholder by using Adaptive Thresholding (2).



$$T_{local}(x,y) = \mu_{global}(x,y) + k \times \sigma_{local}(x,y)$$

$$\sigma^2_{local}(x,y) = \frac{1}{w^2}\left[\sum_{j=y-w/2}^{y+w/2}\sum_{i=x-w/2}^{x+w/2}(\sigma_{local}(x,y) - f(i,j))^2\right]$$

$$\mu_{global} = \frac{1}{M \times N}\left[\sum_{j=0}^{N}\sum_{i=0}^{M} f(i,j)\right]$$

where, $T_{local}(x,y)$ is the threshold value of the pixel located at (x,y) coordinate, $\sigma_{local}(x,y)$ is the local standard deviation and is obtained from local variance. μ_{global} is the global mean, k is taken to be 5 and w is the window size.

After this, the image is passed through a feature detector (canny edge detector) to identify the edges in the image. Hough Line transform is then used to detect the lines and extract information such as the angle of the lines from the vertical. This data is then sent to the processor nodes for further process.

Similarly, the object detection for the other tasks is implemented using contour detection and hue moment analysis. The library offers several functions for such tasks like HuMoment, connected Components, findContours, convex Hull etc.

Control System

Vehicle dynamics

The dynamic model is derived from Newton's –Euler equation of motion.

$$MV + C(V)\,V + D(V)\,V + G = T$$

M is a mass and inertia matrix, C (V) is a Coriolis and centripetal terms matrix, D (V) is a hydrodynamic damping matrix, G is the gravitational and buoyancy vector, T is the external force and torque input vector, and V is the velocity state vector. This equation does not take into account other environmental forces which is a good assumption for our AUV as it will remain within an otherwise stagnant water-body such as a swimming pool. The following assertions were made for the dynamics of the VARUN in order to simplify the modeling:

- The vehicle travels at low speeds, that is, less than 2m/s.
- Roll and pitch movement is passively controlled and therefore, considered to be negligible.
- The vehicle is considered to be symmetrical about its three planes.
- During all maneuvers the vehicle is always maintained in a horizontal posture.
- Disturbances from the water environment on the vehicle such as currents and waves are negligible.
- Sway, that is, movement along the vehicle's y axis, is negligible.
- The vehicle's degrees of freedom are decoupled.

These assumptions simplify the model as:

$$MV + D(V)\,V + G = T$$

Here M is the mass matrix and is represented as sum of rigid body mass matrix (MRB) and hydrodynamic added mass matrix (MA). For VARUN,

$$MRB = di\{m\ 0\ m\ 0\ 0\ 0\}$$
$$MA = dia\{X\dot{u}\ 0\ Z\dot{w}\ 0\ 0\ N\dot{r}\}$$

Where m is the mass and Izz is the moment of inertia about z-axis. The components of MA are constants and are to be estimated from underwater experiments. The hydrodynamic damping matrix (V) contains linear and quadratic damping terms, and is represented as

$$(V) = \{Xu + Xu|u||u|\ 0\ Zw|w||w|\ 0\ 0\ Nr|r||r|\}$$

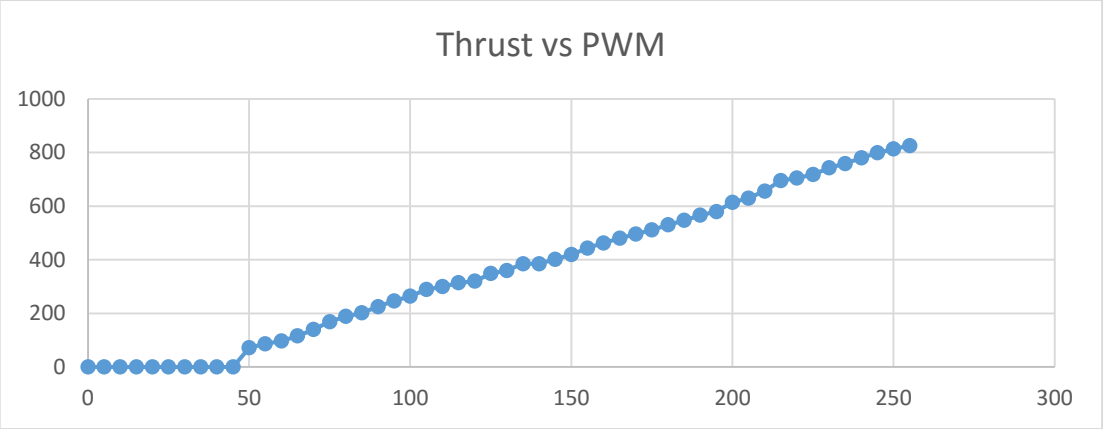
G is the gravity vector and is represented as

$$G = [0\ 0\ mg - FB\ 0\ 0\ 0]$$

Where, FB is the buoyant force acting on VARUN. The external forces and torques vector are obtained from the forces of the six thrusters as the product of a mapping matrix L and thruster forces U. Thruster forces U are controller through PWM signals.

Thruster calibration

Since the input to the thrusters is a PWM signal and the output is the thrust, for controller development we need to know the relation between the two variables (PWM and thrust) for each thruster. This is also necessary because for the same PWM, any two thrusters may not produce the same output thrust. Hence an experiment was performed in Hydraulics Lab of IIT Kanpur to determine the relation. The result for one of the thruster is shown in the plot below



Controller Design

The simplest possible controller is PID which needs no or very little parameter estimation. At present, a PID controller for following a coloured strip using the data from camera and also for following a direction vector using the data from IMU has been designed and successfully tested. Since the motion of the vehicle in X-Y plane can be performed by using either one or a combination of both, we are confident that the PID controller will be sufficient for all the tasks involved. For heave motion(i.e. motion along Z-axis) another PID controller will be designed using the data from pressure sensor. However if at any stage we feel that the PID controller is incapable, we have planned and designed a non-linear controller based on dynamic inversion for controlling the vehicle.

Parameter Estimation

Dynamic inversion technique requires a detail knowledge of vehicle's dynamic coefficients. Some of these can estimated to a good degree of accuracy using ANSYS. But for most of these, we will require to perfom some underwater experiments. Several static and dynamic experiments have been devised for the determination of coefficients. Since we have assumed each degree of freedom to be decoupled (which is a reasonable approximation in this case), we can use the Least Squares Method for the estimation of the parameters. The decoupled equation for each degree of freedom looks like

$$m\xi\, \dot{\xi} + d\xi\, \xi + d\xi|\xi|\xi|\xi| + g\xi = \tau\xi$$

$m\xi$ represents the inertial parameters, $d\xi$ and $d\xi|\xi|$ the linear and quadratic damping parameters respectively, $g\xi$ the gravitational/buoyancy force, $\tau\xi$ the input force/torque and ξ the velocity component for a particular degree of freedom. $m\xi$ consists of both a rigid body parameter, m_{RB},ξ , and an added mass parameter, m_{A},ξ. The notation for these two parameters will be used later to differentiate between the two inertial parameter values.

In order to determine the parameters for each degree of freedom, least squares estimation is used to estimate these parameters given a set of experimental data. Equation can be written in the regression form where λ represents a vector of known values, H is a matrix containing experimentally obtained data, and Λ is a vector containing the unknown parameters.

$$\lambda = H\Lambda$$

The method of Least Squares for estimation of parameters is illustrated below:

$$\hat{\sigma}_\Lambda = \sqrt{diag\left(H^T H\right)^{-1} \hat{\sigma}^2}$$

The estimate of a parameter is found from the following equations

$$\hat{\Lambda} = \left(H^T H\right)^{-1} H^T \lambda$$

$$\hat{\sigma}^2 = \frac{\left(\lambda - H\hat{\Lambda}\right)^T \left(\lambda - H\hat{\Lambda}\right)}{\dim \lambda - \dim \Lambda}$$

The standard deviation and estimate of the parameter can be used to determine the parameter error which is defined by,

$$\varsigma = \frac{\hat{\sigma}_\Lambda}{\left|\hat{\Lambda}\right|}$$

The H matrix is obtained using Static and Dynamic experiments:

The Static Experiment:

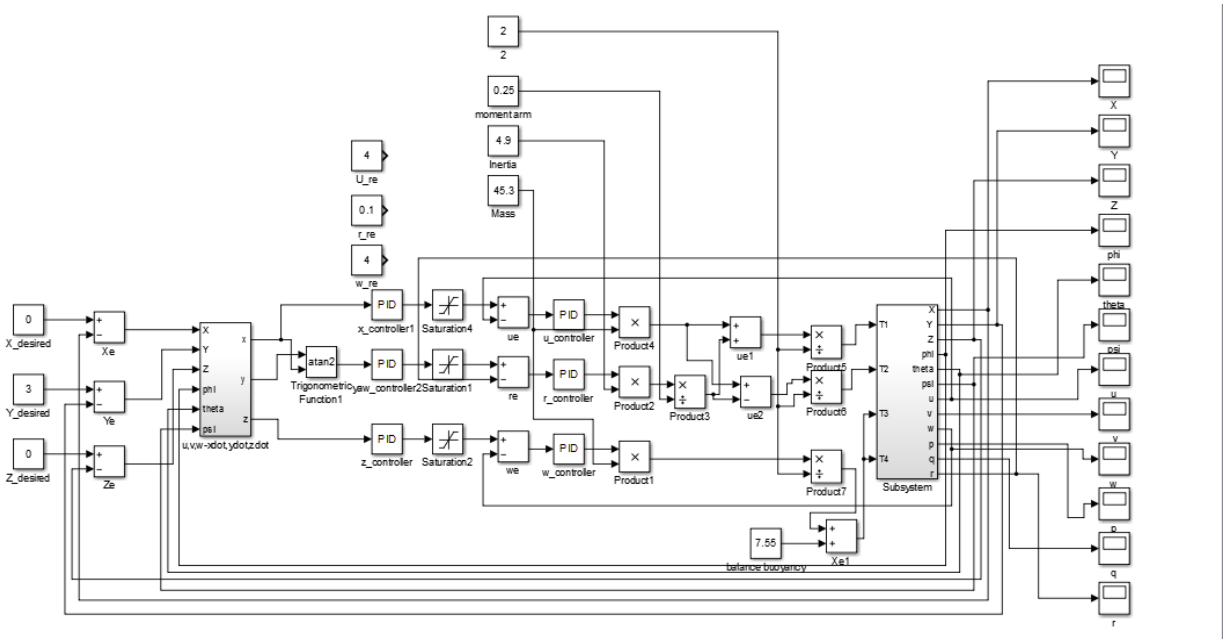
The static experiment involves measuring the velocity of the vehicle under steady-state conditions for a particular degree of freedom. The vehicle has a particular force/torque applied to it and when a constant velocity is reached, this velocity is recorded. The least squares approach described previously are then used to estimate the parameters for the corresponding degree of freedom. Under steady-state conditions, equation for each degree of freedom simplifies to,

$$d\xi\,\xi + d\xi|\xi|\xi|\xi| + g\xi = \tau\xi$$

as acceleration becomes zero under constant velocity. From equation it can be seen that the only unknown parameters are the linear and quadratic damping terms.

The Dynamic Experiment:

The dynamic experiment involves applying a sinusoidal input force/torque to the vehicle about a particular degree of freedom. Since the damping parameters have been estimated from the static experiment, these are now assumed to be correct and can now be used in determining the remaining parameters, that is, the inertial parameters. Hence the inertial mass parameter can be obtained using the dynamic experiments. Therefore, we'll follow this strategy for finding the coefficients for each degree of freedom. A simple GUI has also been developed for the effective testing of the vehicle underwater which could manipulate each or a combination of motors with a required PWM.



Simulink Model implementing simple PID

REFERENCES:

1. Bilateral Filter : https://en.wikipedia.org/wiki/Bilateral_filter
2. Adaptive Thresholding: http://homepages.inf.ed.ac.uk/rbf/HIPR2/adpthrsh.htm