FEFU RoboSub 2015
Autonomous underwater vehicle

Vladislav Goi, Andrei Gatsenko, Gleb Shestopalov, Maksim Sporyshev,
Anton Tolstonogov, Mark Guliaev, Sergey Kulik, Vitalii Storozhenko
Far Eastern Federal University
8, Sukhanova str., 690950, Vladivostok, Russia
Email: tolstonogov.anton@gmail.com

***Abstract*—The following paper describes the development of the AUV which our team has prepared for RoboSub 2015. The vehicle is intended for the competition, the research and development of vision-based navigation and control, and group control research (as a member of a fleet).**

***Keywords*—Autonomous underwater vehicle, RoboSub 2015.**

I. INTRODUCTION

Our team have been participating in RoboSub for last 3 years. This year we have got a new frame, payload of the vehicle and propulsion system, however we retained our classic housings and electronics from last years.

As a lesson from the last year, we use new Enthernet cameras. This type of connection is more reliable then USB which we had before.

As for software, we have significantly improved our propulsion system driver and PID controllers.

II. HARDWARE

A. General construction

Approximate dimensions of the vehicle are: $0.9 \times 0.5 \times 0.4$ meters, weight 30 kilos. The design of the AUV is demonstrated on fig. 1.

The frame of the vehicle is represented by two main vertical polypropylene plates, which hold all the equipment by means of special clamps.

The following housing layout is applied:

- electronics unit, front camera, and depth sensor are in front of the AUV;
- battery unit and side thruster in the center;
- DVL unit, bottom camera, air housing for pneumatic actuators and two remaining horizontal thrusters in the back.

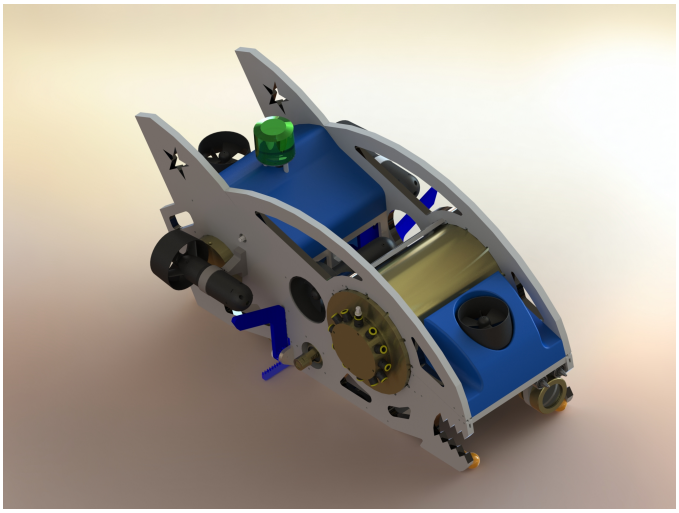

Fig. 1: Render of the AUV model.

There are two vertical thrusters located in front and back side of vehicle.

A compass, Wi-Fi access point and LED of Control-Emergency System are placed in a separate transparent plastic housing located in the upper section of the vehicle as far as possible from the thrusters.

Hydrophones are spaced at four corners and located at the bottom in the inner side.

Block scheme of the vehicle is demonstrated on fig. 2.

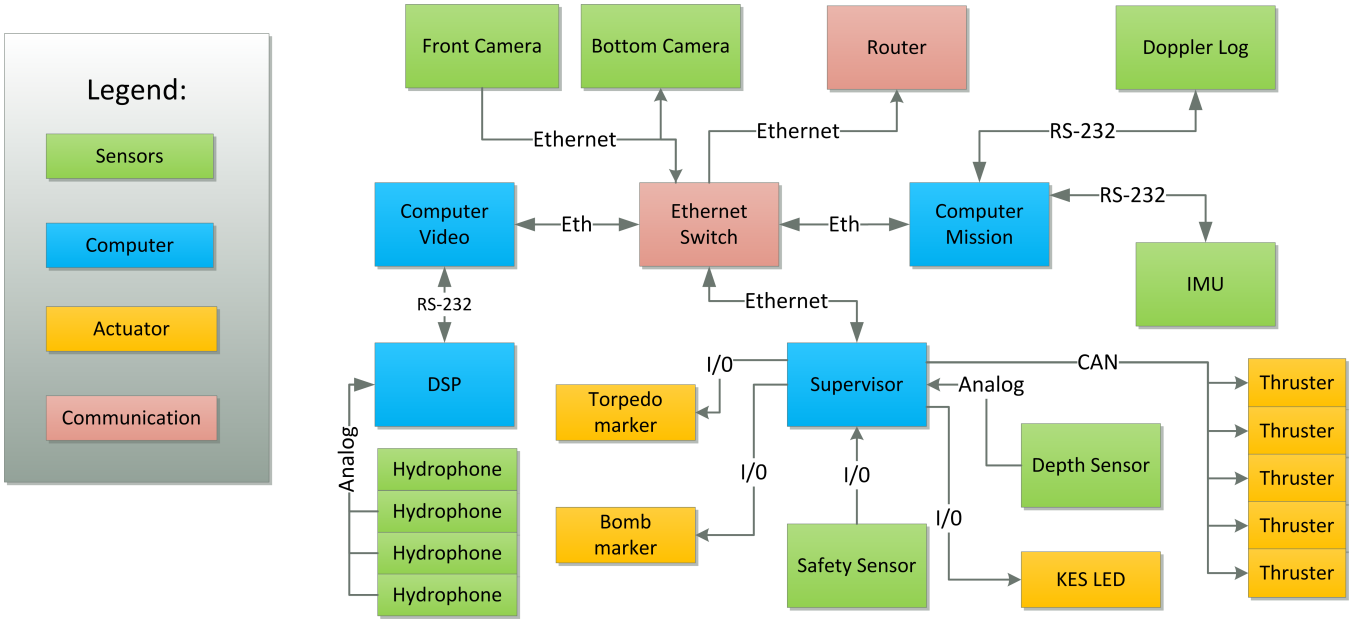

Fig. 2: Block scheme of the vehicle.

B. *Frame and housings*

The vehicle construction consists of a rigid frame made of polypropylene sheets. This material perfectly suits to our vehicle because it does not change its properties under water, it is easy to process, its density is less than water density but still it is strong enough. The whole hardware is attached by the clamps which are made of the same material as a frame. They hold the hardware hard enough and can be easily detached with the hardware for technical maintenance, repair, etc. Prepared sketches are forwarded to the water jet cutting where all the frame components are made from the plane polypropylene sheet.

Render of the vehicle frame with new propulsion system is demonstrated on fig. 3.

All electronic housings are custom-build. They are manufactured of aluminum and then oxidized for corrosion prevention. Our special focus is put on the waterproofing of all electronic housings. It is accomplished with the O-rings, that are sized according to each housing.

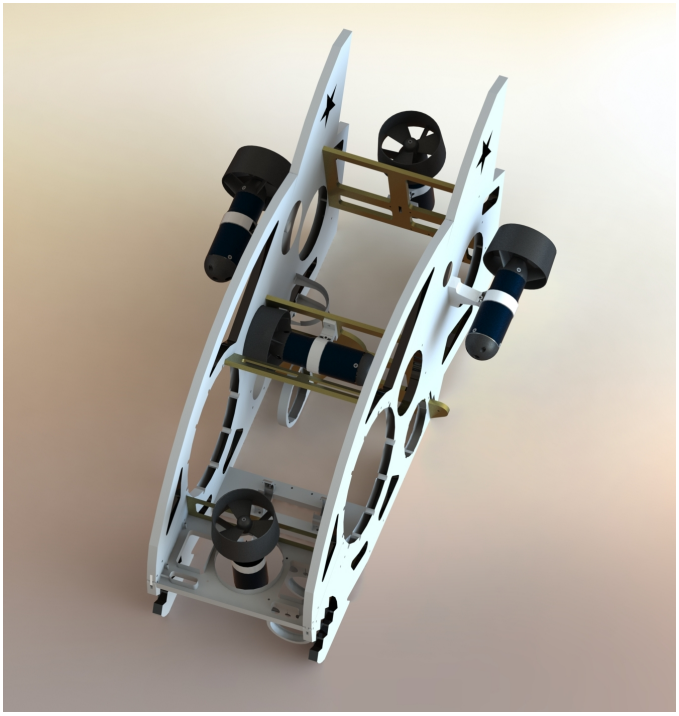

Fig. 3: Render of the vehicle frame with new propulsion system.

C. *Sensors*

Navigation system represented by three sensors:

1) Depth sensor (model PD100). Accuracy of this sensor is about a couple of centimeters, which is sufficient to stabilize a given depth

during missions.

2) Inertial Module Xsens Mti IMU. It integrates three-axis gyroscope, accelerometer and magnetometer, as well as the Kalman filter is implemented, which provides sufficient accuracy for stabilization of roll, pitch and heading.
3) Doppler Velocity Log from RD Instruments. We integrate velocities in order to get current coordinates. This coordinate system is further used to return to the locality of points of interest. It is very useful when implementing a sequence of missions.

These sensors allow us to control the vehicle in 5 out of 6 degrees of freedom.

Video system. This system includes two professional cameras (model: Prosilica GC 1380). One of them is directed forward and the other one is directed down. The cameras work in 400×300 px mode (maximum resolution: 1360×1024) and allow us to adjust exposure, color correction and frame resolution in a fairly wide range, both in manual and automatic mode. This greatly facilitates of computer vision detection, allowing to deal with the adverse conditions of the open water. The camera operates at 30 frames per second at maximum resolution, but we are working with a frequency of 5-7 frames per second.

Finder sonar. There are digital signal processor based on STM32F4Discovery with integrated DSP unit and 4 hydrophones used in the AUV to perform the pinger task.

Safety system. Safety sensors are mounted in every waterproof housing. A warning message is sent immediately to a GUI operator program if one of them is activated. Control emergency system of the vehicle signals for a dangerous situation, if the vehicle is under the water and is not connected with operator.

D. Actuators

For successful tasks execution the AUV is equipped with pneumatic system for torpedo shooting. It contains 2 tubes with aluminum torpedoes inside. Tubes are linked to the air bottle which becomes opened right after the supervisor signal.

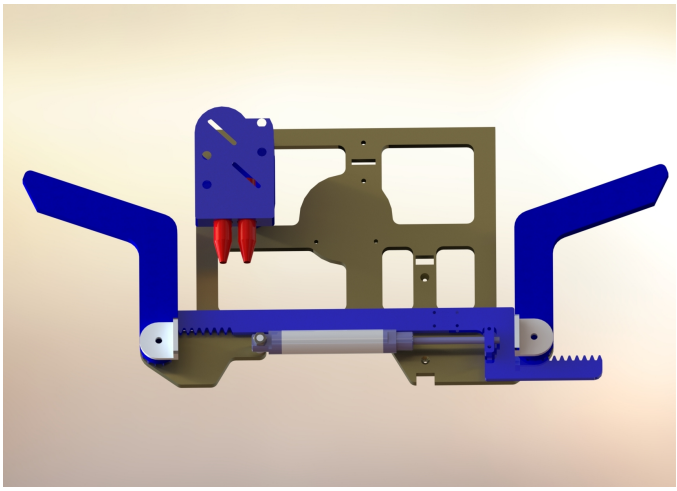

Fig. 4: Render of pneumatic actuator.

AUV is equipped with a system for dropping cargoes based on electromagnets.

We have designed a pneumatic grabber to deal with new buckets (fig. 4).

All actuators are spaced near the corresponding cameras to work efficiently with computer vision.

There are 5 thrusters on the AUV: 2 vertical ones and 3 horizontal. Thrusters use 24 volts Faulhaber motors and polyurethane propellers.

E. Computers

Two uniform computers are used for calculations — single-board computer PC/104-Plus with processor ool RoadRunner 945GSE with processor Intel Atom N270 (1.6 GHz, 1 GB SDDR2, 533 MHz). Most of our software modules (including a mission module) and sensor data processing is executed on the mission computer. The second computer processes computer vision and acoustics.

F. Low-level devices

Supervisor. There is a supervisor board of our team design placed in the electronic hull. The board is based on a STM32F207 micro controller and has an Ethernet interface for the communication purpose. This board digitizes the data from the depth sensor, stores the data from all water sensors and the voltage and amperage from the battery. It ensures the

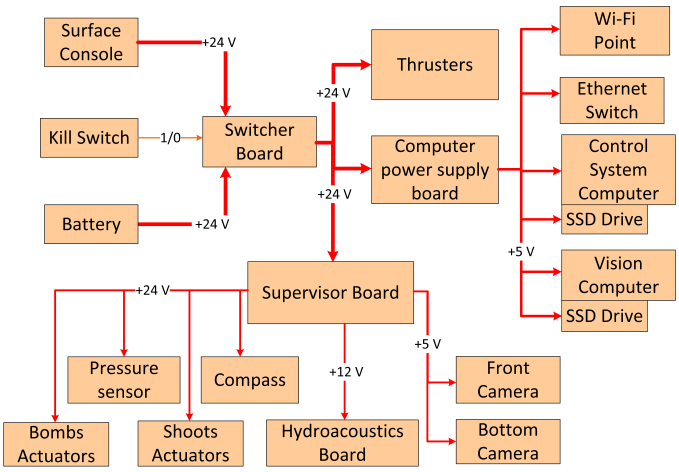

Fig. 5: Power scheme.

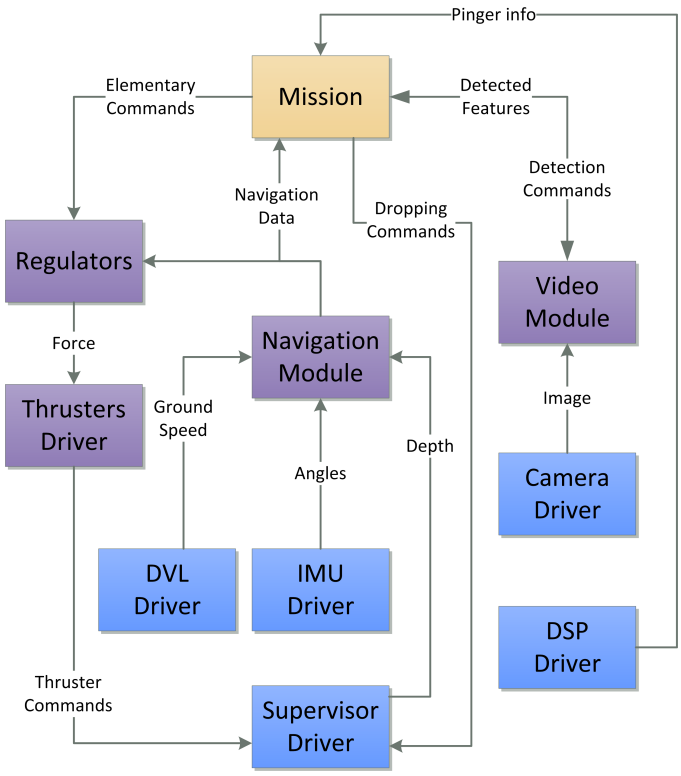

Fig. 6: Control system structure.

thrusters control signal transmission and controls the actuators power. Also the low level of the control-emergency system is implemented on supervisor.

Digital signal processor. For the acoustic pinger signals processing we use a separate signal processor board. First step is signal amplification with 4 hydrophones, then a passive filtration and a voltage level conditioning for the further signal digitizing. After that, the signal digitizing and further processing of the signals from four channels is ensured by the micro controller STM32F407VGT6, which contains DSP instructions. The controller does the digital filtration with the narrow bandpass IIR filter and if the signal was received in all four channels, the relative time of signal delay is calculated. Then, the calculated time is sent to the computer, that controls the acoustic data processing, via USB protocol.

G. Power supply

The power to run our vehicle is provided by a lithium-polymer battery 5 Ah, which is located in a separate housing with a detachable connector. If necessary, we simply remove the discharged battery housing and replace it with the charged one. Such an approach saves a lot of time and is safe enough. Battery power is supplied to the main electronics unit, where it has already been distributed between all devices.

Our power system scheme is presented at fig. 5.

III. SOFTWARE

A. General structure

We have used hierarchic structure of control system for our vehicle. Each node of this structure is implemented as a separate executable module. Information transfer between modules is organized by messaging.

Our control system structure is presented at fig. 6.

System consists of 9 modules. Mission module defines the vehicle behavior is placed on top of the hierarchy. Regulators module is responsible for motion control.Navigation module processes the information from navigation sensors and passes it to motion and mission modules. Other modules are designed to interact with hardware devices. It sends commands to actuators and processes sensor data.

B. Operator software

Our operator software allows us to monitor the vehicle status. It connects to the control system

modules via local network (connection is established between router on the vehicle board and operators laptop via Ethernet or Wi-Fi). During development of the operator software we paid special attention to the information from safety sensors such as water alarm sensors and battery voltage.

Besides, the GUI provides the capabilities for vehicle remote control, video module control and receiving images from cameras (images are compressed in JPEG format for transmitting over the network).

One of the tabs of the operator GUI interface is given in fig. 7.

C. Module communication framework

We use IPC library [1] for communication between modules in our control system. There were several problems with IPC related to necessity of accurate description for each message, neat memory management, the usage of C interface for the message processing, which inevitably led to errors.

Data transfer between modules is a common topic and there are some means in robotics software frameworks to solve this problem. It could be solved via both shared memory (e.g. MOOS control system [2], Cornell University Team control system [3]), and via messaging. We decided to use the second option since our current software is based on messaging. Among robotics platforms which use messaging we have chosen ROS [4] as our future platform - its usage is growing intensively, there are examples of its use in an AUV specially designed for RoboSub (e.g. Bumblebee team [5]).

Full transition to ROS is a labour-intensive process. So, we decided to implement a step-by-step transition, starting with the declarative description of messages in ROS format.

We have developed a program for this, which uses ROS format text description to generate messages in IPC format and C++ code that translates parts of the messages into C++ object-oriented structures and automatically manages the resources.

We have also developed a library, which encapsulates all IPC facilities and simplifies programmers work in modules communication.

D. Mission module

The mission module contains the motion and actions logic of the vehicle.

This module consists of separate tasks, which are executed in a linear sequence. The order of execution is described in a set of configuration files. There is a global map that stores information about recently found orange stripes. This information is used for vehicle positioning before starting the next scheduled task.

Our previous task execution scheme was pretty similar to the state machine. The set of states for each task is explicitly formulated for a state machine. We have strictly structured our code in accordance with that and developed a corresponding object oriented structure. A typical action in one of the states is stabilization of the vehicle in front of an object on the video frame.

The mission module is implemented with several C++ classes, their interaction is shown at fig. 8.

The Environment class receives and processes navigation messages and stores the physical vehicle parameters (e.g. it stores camera models) and information about environment objects used by tasks. Tasks can communicate with each other via this class. The Environment is the interlayer for the interaction between mission and lower layers.

The mission class is designed to create, configure and execute each of the tasks. The configuration of each task is formed here. Each task has a corresponding C++ class. Each of those classes analyses the data from the Environment and uses a state machine implementation to process its states consequently.

YAML is used for mission configuration files. Each task reads parameters from three sources in the following order:

1) task.yml
2) <task_name>.yml
3) mission.yml

The task.yml file describes default parameters which are common for all tasks. A separate file for each kind of task gives parameters common to all instances of that kind. The mission.yml file contains

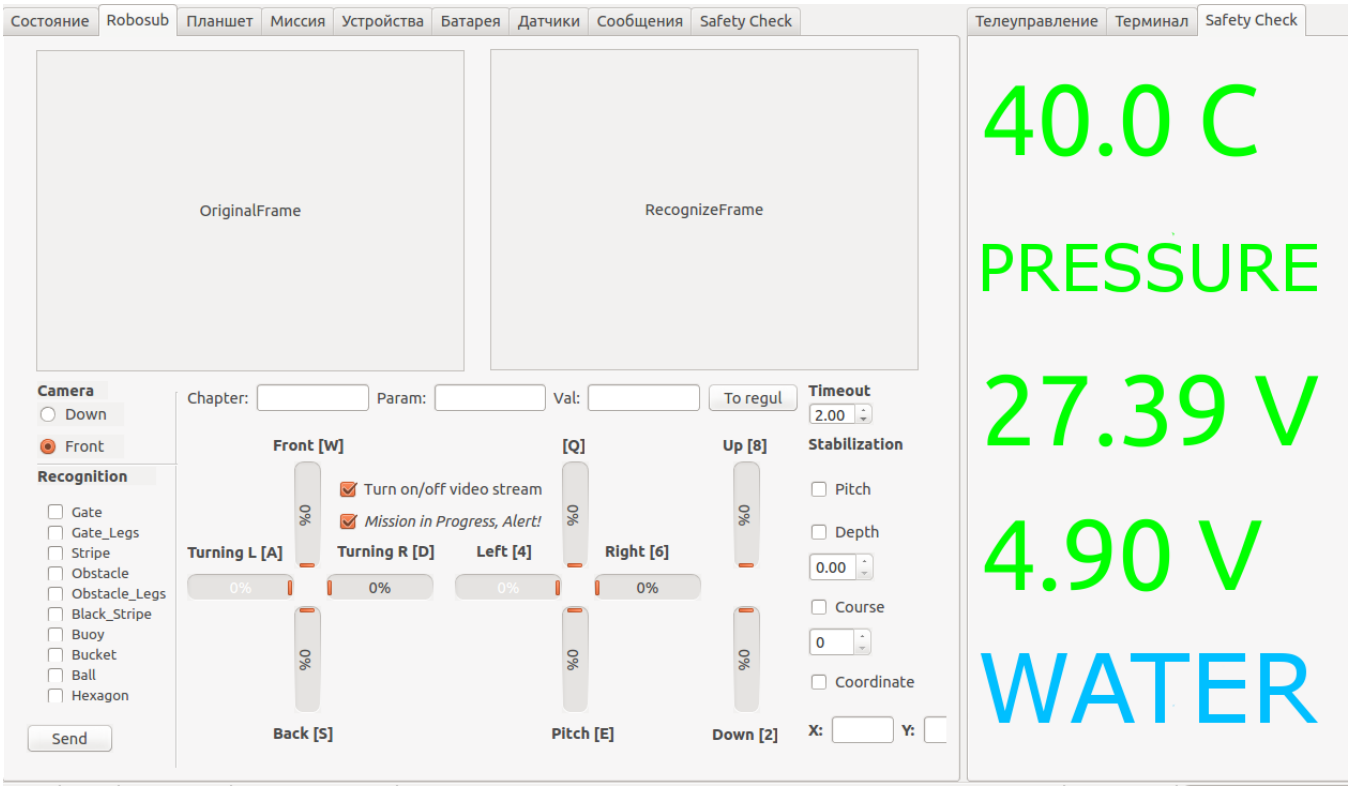

Fig. 7: Interface of one of operator GUI tabs.

the whole mission parameters. A task may have about 50 parameters, and this approach allows to decrease the size of the main configuration file and to make preparation for the run easier.

The mission execution time can be essential this year. So, it is important to stabilize faster at objects of interest. We have added a differential component to a vision regulator to deal with that. We have adjusted coefficients using our post processing tools.

E. Low-level controls

Motion module. The vehicle motion control is performed in the event loop. The data from navigation module is processed at the each iteration of this loop. The message containing thrusts in all six degrees of freedom is formed on the basis of this data. The vehicle is able to control only five degrees of freedom, so the roll thrust is always equal to zero.

One of the aims during module development was the extensibility and opportunity of code reusing for different vehicles with different configurations. Hence, we use the following structure: a separate regulator implemented as a module is created for each control command (heading stabilization, position stabilization, etc.). Such a module subscribes to a message, that describes the module command and activates the regulator. Regulators lifetime is defined by the command execution time. After its creation regulator reserves necessary control degrees (e.g. position regulator reserves longitudinal, transverse and heading axes). Capability control channels conflicts are solved in command creation order: new command, which uses these axes, replaces the old one.

Each regulator is the means for solving the physical control problem. PID controller is used on this vehicle for solving mathematical control problem.

Propulsion system module. Propulsion system module task is to receive thrusts and torques from motion module and calculates control signals for each thruster on the basis of received values. In accordance to each thruster orientation its direction can be described with 3-dimensional vector in ve-

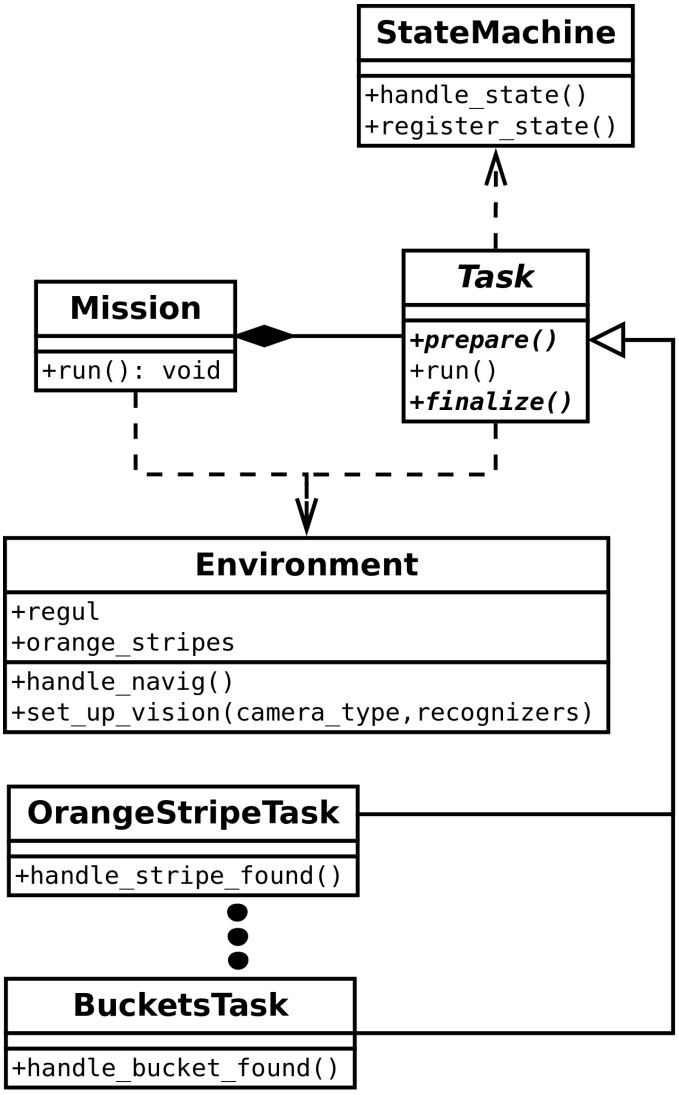

Fig. 8: UML class diagram used in the mission.

hicle coordinate system. Wherein thrusters having a part in calculating thrust at different axes have the contribution to the corresponding axis proportionally the angle between the axis and the thruster vector.

Given information above and each thruster shoulder we get 5×5 matrix which shows the thruster (moment) allocation at the certain axis among thrusters.

Thrusters firmware receives control signals (codes) in $[-128, 127]$ interval. So the module have to convert these variables from $[-1, 1]$ interval to the proper interval. The vehicle uses symmetrical propellers. Its performances were received during

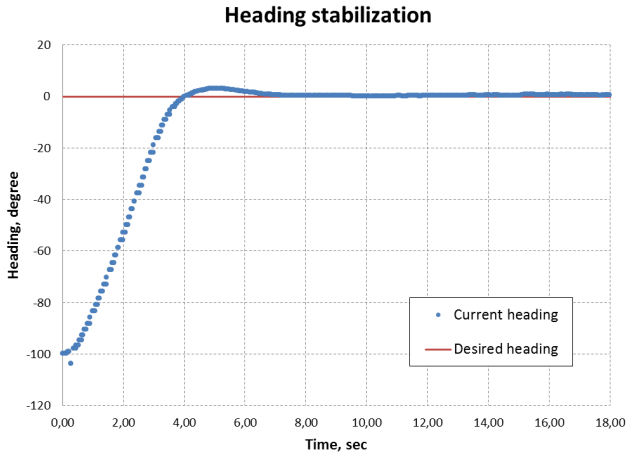

Fig. 9: Example of heading control with the new motion module.

bench tests. We are able to calculate control signal for any thrust sign linear interpolation if we know a number of conformance points between thrusters thrust and control signal.

F. Video module

The main goal of video module is analyzing images taken from cameras and passing information about detected objects to mission module. Mission sends to video module a list of required objects and the number of camera to take images from.

OpenCV library is used for image processing. The version 2.4.8 is installed on the vehicle.

Each detection algorithm usually consists of the following parts:

1) Preprocessing. It's done by applying filters or clustering algorithms to correct underwater colors, remove noise and small useless details. Color correction is done by increasing red channel in RGB image. Then three ways of image preprocessing are used: median blur, gaussian blur, SEEDS superpixel segmentation. Median and gaussian filters are implemented in OpenCV. We have developed a custom SEEDS implementation following [6].
2) Color binarization. The source image is translated to HSV color space and the bi-

narization is executed by hue and saturation threshold.

3) Contour analysis. It is based on OpenCV implementations of Suzuki-Abe algorithm and polygon approximation, a custom Hough implementation is also used.

We've calibrated our cameras using OpenCV calibration tools. A chessboard of 1×1 meter size was made. The chessboard is positioned so that it lies entirely in the field of vehicle camera view, the camera takes images. Then the images are given to the calibration program which detects angles on the chessboard and calculates camera parameters, such as focal distance and radial, tangential distortion factors. All this parameters are considered in our camera model which helps us to calculate the heading to the detected object and determine distance to this object more accurately.

G. Implementation

C++ programming language is used in our vehicle control system software implementation. Some modules significantly depend on C++11 standard features so we were to update GNU compiler to 4.8 version. Boost libraries provides us with some useful extensions that we apply to, for example, our thread pool implementation. We also need it in command line interface implementation for our tools. Cmake intelligent build system is used to build our system. Bash scripts are used to launch it. We have 6 software engineers in our team. Git version control system and central private repository on bitbucket.org are chosen to organize their team development process.

IV. TESTS AND TRIALS

A. Simulator

Our simulation system was implemented using Gazebo simulation tools and API. Gazebo uses Protobufs for interprocess communication, so we use adapter tool which converts messages from our internal format to protobufs. This approach allows us to use the same software with both simulator and real devices.

Simulation system GUI shown in fig. 10.

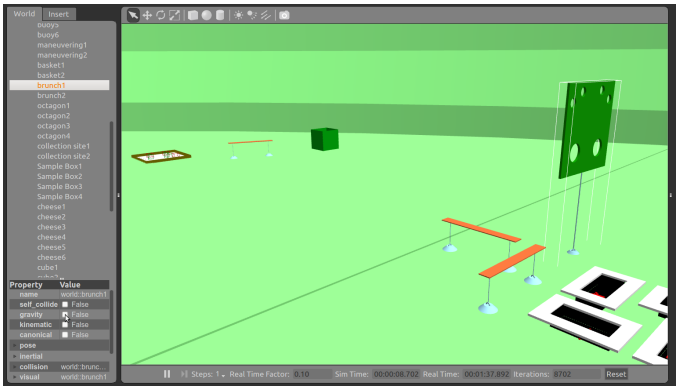

Fig. 10: Gazebo GUI.

B. Pool tests

One of the most essential parts in vehicle preparation is testing in pool. This process is required to detect hardware failures, produce some components setup and find out unexpected bugs, that have not been appeared earlier. We started to test our vehicle in pool at the beginning of June in our campus swimming pool.

C. Debugging

We use text logs to debug our software. It is human-readable could be fluently analyzed without special software. We have an agreement of a "basic form" of the log file. It is a CSV-like format with series of numbers. We have developed log_lot a Python utility using Matplotlib to show charts for series in a basic form log. Log_plot has more then 10 parameters allowing to point out a given period of time, axis names etc. It receives the data from the standard input.

We also have a log_merge utility for merging to logs form two separate modules.

The mission log is not in the basic form since in contains text messages for human analysis. A simple run on the file with regular expressions can extract the specified info and represent in is the basic form.

Since all of our utilities use standard input and output, Unix pipes can be used for rapid log analysis.

ACKNOWLEDGMENT

The development of FEFU AUV is supported by Far Eastern Federal University and by Institute for Marine Technology Problems of Far Eastern Branch of Russian Academy of Sciences. We would like to thank our mentor Dr. Alexander Scherbatyuk. We also would like to thank those from FEFU and IMTP FEB RAS who have supported us in this work.

REFERENCES

[1] R. Simmons, *Inter-Process Communication: a reference manual*, http://www.cs.cmu.edu.

[2] P. Newman, *MOOS - mission orientated operating suite*, Massachusetts Institute of Technology, Tech. Rep. 2299/08, 2008.

[3] M. Burkardt, L. Barron, T. Brook et al, *Cornell University Autonomous Underwater Vehicle: Design and Implementation of the Ragnarok*, http://cuauv.ece.cornell.edu.

[4] M. Quigley, B. Gerkey, K. Conley et al, *ROS: an open-source robot operating system*, Open-source software workshop of the Int. Conf. on Robotics and Automation, Kobe, Japan, 2009.

[5] O.T. Chang, G.E. Wei, J. Ong et al, *BBAUV: Autonomous Underwater Vehicle, software overview*, www.bbauv.com.

[6] M.V. den Bergh, X. Boix G. Roig et al, *SEEDS: Superpixels Extracted via Energy-Driven Sampling*, ECCV 2012.