FEFU/IMTP RoboSub 2017 Autonomous underwater vehicle

Vladislav Goi, Maksim Sporyshev, Sergey Kulik,
Ivan Chemezov, Andrei Makhliarchuk, Grigorii Eliseenko
Aleksandra Zhikhareva, Aleksander Pavin, Vladislav Bobrov
Far Eastern Federal University, Institute for Marine Technology Problems
8, Sukhanova str., 690950, Vladivostok, Russia
Email: msporyshev@yandex.com

***Abstract*—The following paper describes the development of the AUV which our team has prepared for RoboSub 2017. The major goals of the vehicle are the competition, the research and development of vision-based navigation and control methods, and the research of group control methods.**

Keywords—Autonomous underwater vehicle, RoboSub 2017.

I. INTRODUCTION

Our team has been participating in Robosub for last 4 years and each year we consistently reach the final stage of the competition. Each year give us lessons of robotics development. We started to develop the vehicle from scratch twice, given the experience of previous years during four years of participation. All components of the vehicle: both electronics design and programming have been upgraded, and sometimes re-designed from scratch.

We prefer the evolutionary path of development of the vehicle, wherein the only one part of the AUV is significantly changing at a time. For example, we substantially redesigned the frame of the vehicle, almost not touching other parts from last year. Now we have kept the design of the last year's vehicle and partially renovated electronics. But our main achievement of the year is radically modified software architecture. This year we have completely migrated to ROS.

You can see at figure 1 how the vehicle has been implemented.

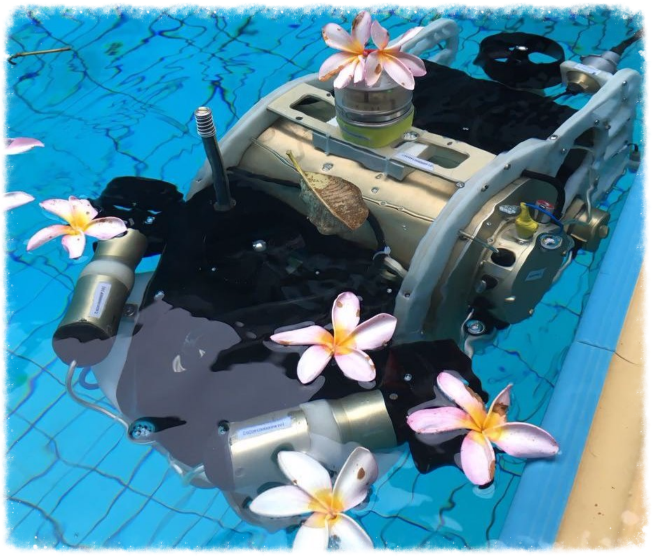


Fig. 1: Implemented of the vehicle.

II. HARDWARE

A. General construction

Approximate dimensions of the vehicle are: $0.9 \times 0.5 \times 0.4$ meters, weight 30 kilos.

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;

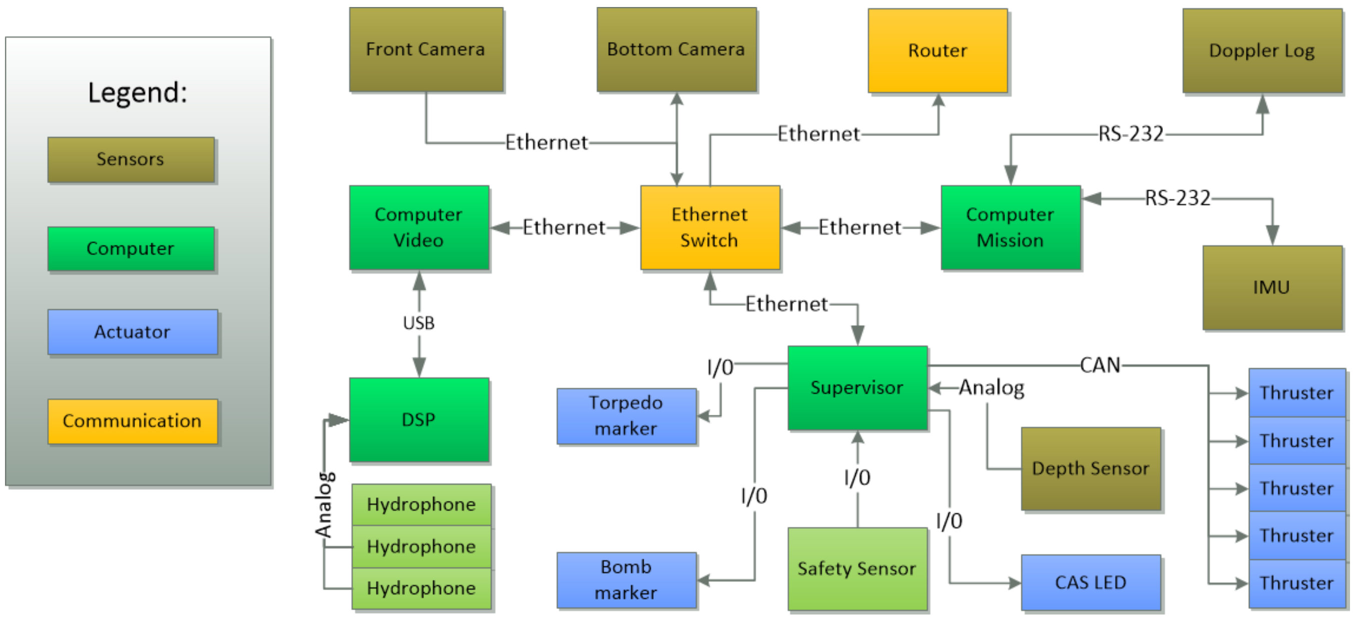


Fig. 2: Block scheme of the vehicle.

- DVL unit, bottom camera, air housing for pneumatic actuators and two remaining horizontal thrusters in the back.

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.

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.

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.

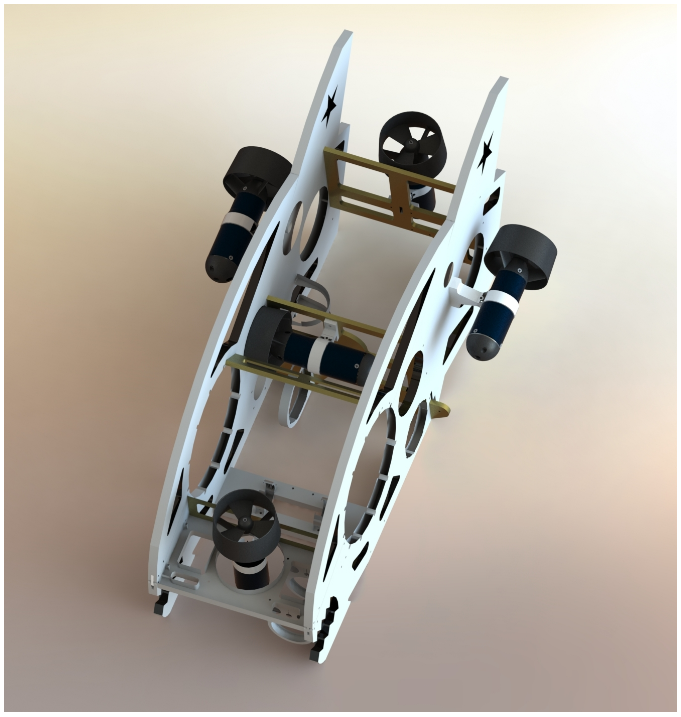


Fig. 3: Render of the vehicle frame with new propulsion system.

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.

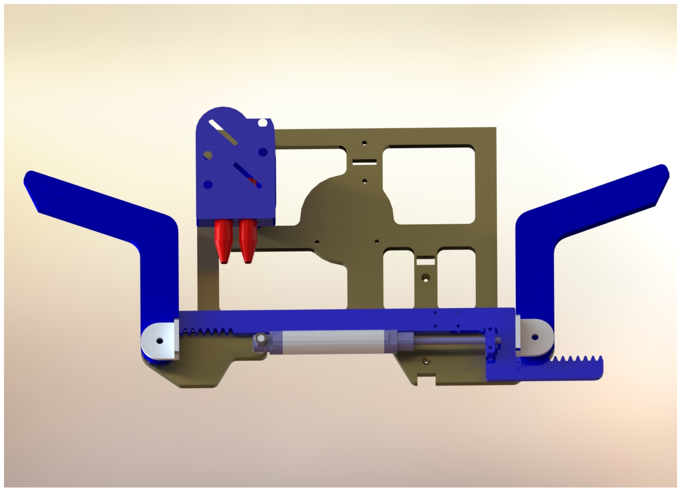


Fig. 4: Render of pneumatic actuator.

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.

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 Cool 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 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.

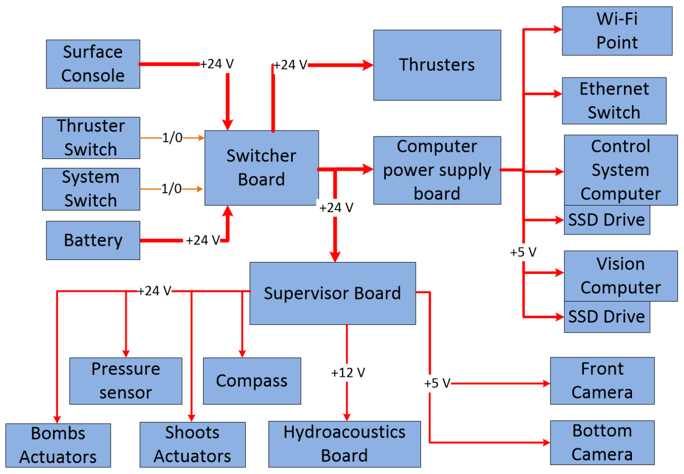


Fig. 5: Power scheme.

Our power system scheme is presented at fig. 5.

III. SOFTWARE

A. Interprocess communication

This year we moved from ROS system to our own message transport. The reason was leaving complicating build system, big amount of libraries for more scalable library for our goals. Also the new created system is distributed, which is more reliable and comfortable to use.

Communication between processes is carried out via special messages that include user data and service data, and are sent asynchronously using UDP (User datagram protocol). Thus, the data provider is not blocked after the publication, and acts on the principle ?published and forgotten?. All messages have unique identifiers (message names), which are a symbolic representation of the C++ data type. Such an approach guarantees the absence of message name conflicts.

Each message name is associated with a UDP-port, which is calculated as a hash function of the message name. The Fig. 3. Example of a simple dead-reckoning system on C++ language. maximum hash value can be more than the UDP-port range. Because of this, the remainder of dividing the hash-value to the port-range is used as the final UDP-port. Thus, all messages are sent to the associated port and are available for listening to all other system

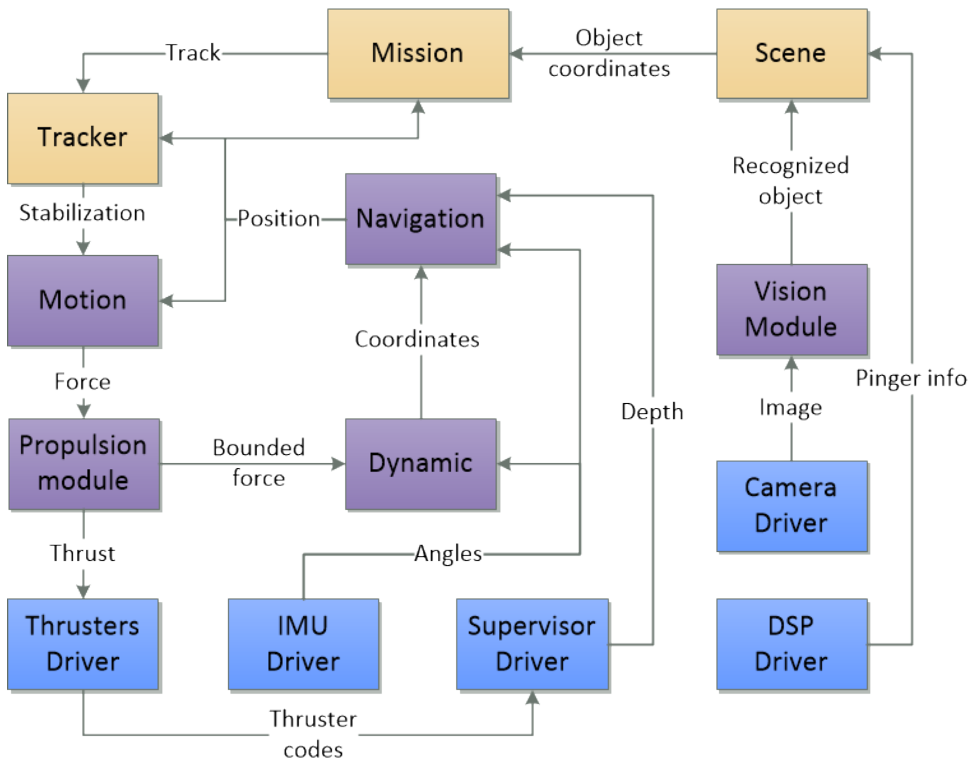


Fig. 6: Software modules.

components without knowledge of each other. A program-listener reads the data (via the library) from the corresponding port, checks the message name (to avoid port conflicts) and puts a new message in the memory or internal queue. All incoming messages are stored in a separate thread that allow using OS-interrupt functions, like Sleep() or delay().

On the following figure (fig. 6) you can see our software modules.

B. *WebGUI*

A web-based interface was used for operator interaction with the UUV software. We applied Jquery [17] and JqueryUiMobile [18] techniques to achieve compatibility with mobile devices and ROS-system. Such an approach allows remote UUV control from any computer or mobile device without installing additional software.

The main problem of implementation of graphical user interface (GUI) is the reconfiguration of control elements in a browser window for different systems (Fig. 6). It should be noted that the structure of information, the number of elements and indicators may change when adding/editing software modules. The following approach is proposed to solve this problem:

- The user interaction with any programs is performed through the message-passing library, and web-socket technology uses for sending and receiving data.
- A simple reconfiguration of the system is carried out using the JSON and JSON schema formats for data exchange between UUV modules and GUI. All you need to add a new module ? just describe your new data in the form of JSON schema. After first browser-request to the web-server all described data will automatically added to the GUI in accordance to the data description.
- All messages are divided into input and output data in relation to the UUV programs. There are two types of input data: commands which are sent once by request (for example, the command of ”switch on” or ”switch off”

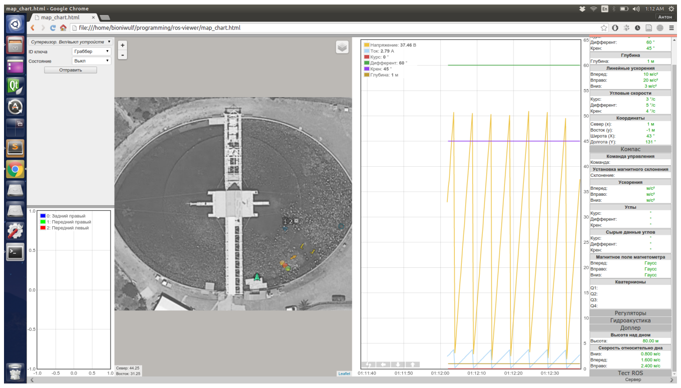


Fig. 7: Example of our webGUI.

to some device) and control values which are sent periodically (e.g., velocity correction of the robot). Outgoing data are called messages and can be published by any modules at any given time.

Web GUI console is one of the most useful tools for us. On the figure 7 you may see example of the web interface.

C. Navigation node description

Navigation is developed to solve two main problems:

1) Filtration and broadcasting data from IMU, DVL and other sensors.
2) Path‘s calculation based on information of velocity and heading.

The first problem is related to the abstraction consumer of navigation data from the sensor data providers. For example, motion node uses a velocity calculated by navig node during motion stabilization, without worrying about sensor delivers this speed actually. In turn, the navigation node receive messages from DVL and publishes speed on its own behalf if it just received. If the data become obsolete navigation node corrects old DVL data using acceleration’s data provides by compass.

The problem of path’s calculation is solved by classic method (integration of the vehicle’s speed, taking into account the orientation data).

D. Mission node description

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

This node 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.

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

- *task.yml*
- *task_name.yml*
- *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 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 ROS tools.

E. Motion control

Motion control divided in two functional part: motion node and TCU (thruster control unit or propulsion system) node.

Motion solve high level motion problem including stabilization. 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. Regulator's 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 node 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 vehicle 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 x 5 matrix which shows the thruster (moment) allocation at the certain axis among thrusters.

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

F. *Video module*

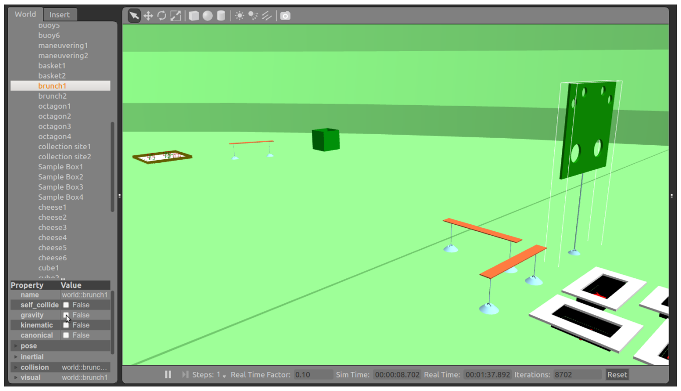
The main goal of the video module is analyzing images taken from cameras and passing information about detected objects to mission module. Mission sends to video module a list of required objects and the number of camera to take images from.

OpenCV library is used for image processing. We use version 2.4.8, because this is the most stable and reliability version at the current time.

Each detection algorithm usually consists of the following parts:

- 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. Median and Gaussian filters are implemented in OpenCV.
- Color binarization. The source image is translated to HSV color space and the binarization is executed by hue and saturation threshold.
- Contour analysis. It is based on OpenCV implementations of Suzuki-Abe algorithm and polygon approximation, a custom Hough implementation is also used.

We have calibrated our cameras using OpenCV calibration tools. A chessboard of 1 x 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 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.



Property	Value
name	world::brunch1
self_collide	False
gravity	False
kinematic	False
canonical	False
collision	world::brunc...
visual	world::brunch1

Steps: 1 Real Time Factor: 0.10 Sim Time: 00:00:08.702 Real Time: 00:01:37.892 Iterations: 8702

Fig. 8: Gazebo GUI.

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. Simulation system GUI shown in fig. 8.

B. Debugging

The developed data storage mechanism uses a distributed (decentralized) model too, where each module stores its published data to a local folder in the JSON (JavaScript Object Notation) format. This approach allows increasing the number of modules without overloading the network and storage server (not needed in this case). Also each module stores the JSON-schema description of its data, which allows the user to interpret data correctly during post processing.

The object schema stores description of the message in accordance with the JSON-schema standard: type, title, properties and some others. Also some additional keys added to the schema.

Such an approach allows:

- Store all data not faster than the data appears in the system (absence of duplicates).
- Always save data structure and its description at the time of publication. It is necessary for correct data interpretation after system reconfiguration.
- Selected data are available for online and offline uploading.

C. Pool tests

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

We used a new way to monitor our vehicle state using a real time webGUI and thin debugging wire. That gave us the ability to find out bugs and failures in real time and not spend to much time watching bagfiles after mission execution. Furthermore, the Gazebo simulation reduced necessity of vehicle testing in vivo to a minimum.

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

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

[3] 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.

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

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