MATEC Web of Conferences **125**, 02017 (2017) DOI: 10.1051/matecconf/201712502017
CSCC 2017

High–Level Control System for Biomimetic Autonomous Underwater Vehicle

Tomasz Praczyk[1,⋆] and *Piotr* Szymak[2,⋆⋆]

[1] *Institute of Naval Weapon and Computer Science, Polish Naval Academy, Gdynia, Poland*
[2] *Institute of Electrical Engineering and Automatics, Polish Naval Academy, Gdynia, Poland*

Abstract. Usually, a rough software architecture designed for a robot can be can be shortly presented in the form of layers. The lowest layer is responsible for direct control of the hardware, i.e. engines, energy system, sensors, navigation devices, etc. A next layer is a low–level control which knows how to use the hardware in order to achieve a desired state of the robot, e.g. to stay on a desired course. And the last layer, the layer which is the nearest to the human–operator, is a high–level control which decides how to use the low–level control and sometimes also individual pieces of the hardware to achieve predefined objectives. The paper describes architecture, tasks and operation of the high–level control system (HLCS) designed for Biomimetic Autonomous Underwater Vehicle (BAUV).

1 Introduction

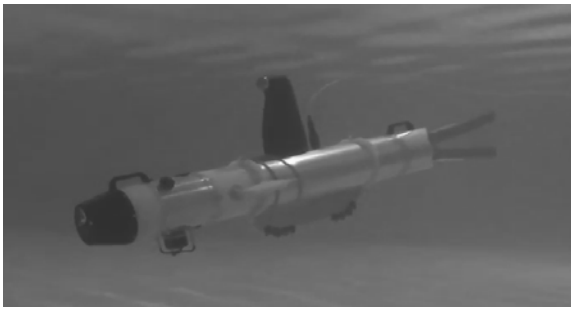
Robots are usually a combination of many different devices, sensors, engines, in other words, each robot consists of different sorts of the hardware. To control the hardware, the software is necessary, and it occurs at different levels. The lowest level is responsible for direct control of the hardware, each device, sensor, battery, engine has a dedicated software which provides the interface required to control it via a definite communication protocol. For example, Vector Nav VN200 inertial navigational system (INS), used by many robots to orientate in the environment, has the software integrated with the device which communicates via RS232 and allows all external users to configure it and to read all navigational data provided by it.

At the higher level, the low–level control typically occurs, and it is responsible for achievement of low–level goals like maintaining a desired speed, direction of motion, in the case of underwater vehicle also depth. In addition to the parameters above, also other ones can be subjected to regulation by means of low–level software components, e.g. speed of engine rotation, position of manipulator, motion of artificial fin. For the purpose of regulation, different techniques can be used, e.g. traditional PID controller or fuzzy controllers.

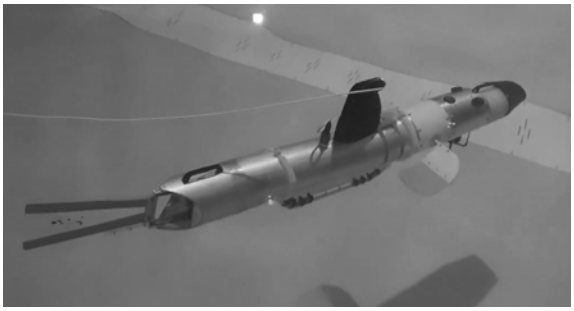
The highest level of the software is responsible for achievement of high–level goals defined by a human–operator. When we deal with remotely operated robot, the task of the high–level control system (HLCS) is only to integrate the hardware and the software, also low–level control software,

⋆e-mail: t.praczyk@amw.gdynia.pl
⋆⋆e-mail: p.szymak@amw.gdynia.pl

into one organism. In this case, usually, the HLCS is responsible for passing commands of the operator to appropriate robot components. The operator knows, in this case, how to use the robot and all its devices and mechanisms to achieve the goals. Other situation is when an autonomous robot is in use, in such a case, the HLCS has to play the role of the operator and has to know how to use the hardware and the software of the robot to perform the task defined by the operator. Moreover, it also has to suitably respond to emergency situations which have to be detected somehow, sometimes also by the HLCS which is the only robot system which is in possession of all available information, and in consequence, it is the only system that is able to properly analyze the situation and to make a suitable decision.



(a)



(b)

Figure 1. BAUV used in experiments

Biomimetic Autonomous Underwater Vehicle (BAUV) [1, 6, 7] is a type of robot which is able to operate under the water, which is autonomous meaning that it is capable of performing some missions independently of a human–operator, and which is biomimetic, that is, it mimics appearance and behavior of living organisms, of course, to certain extend. An instance of the BAUV was constructed within the project entitled "Autonomous underwater vehicles with silent undulating propulsion for underwater ISR[1]", financed by Polish National Center of Research and Development – see Fig. 1. The autonomy of the BAUV entails the necessity of equipping the vehicle with the HLCS which is responsible for: executing orders of the operator in remotely operated mode, providing selected information to the operator, performing a mission defined by the operator, mapping, detecting emergency situations and responding to them. In addition to that, the HLCS is required to be easily scalable through adding new high–level behaviors.

[1] ISR – Intelligence, surveillance and reconnaissance

The paper describes process and implementation architecture of the HLCS mentioned above. Moreover, it also gives some algorithms of system operation. The remaining part of the paper is organized as follows: section 2 is an outline of the BAUV, section 3 is a description of the HLCS architecture and its operation in the form of algorithms, and the final section is a summary.

2 Biomimetic Autonomous Underwater Vehicle

The BAUV whose HLCS is the main subject to the paper consists of a number of hardware and software components [2]. When They all can be divided into seven functional parts, i.e. drive, energy system, sensors, collision avoidance, navigation, communication, and control.

As for the drive, the BAUV is equipped with two separate undulating propellers, i.e. two pectoral fins and one tail fin, with their engines. The pectoral fins are mainly used to control depth whereas the tail fin is applied to control course (heading). When the BAUV is on the surface the pectoral fins can be also used to change course, in that case, they work in the opposite direction, i.e. one fin pushes the vehicle forward whereas the second one backward. In addition to the fins and their engines, the drive part also includes the low–level control system (LLCS) which is the software component whose task is to implement decisions of the HLCS or the operator. In the remotely operated mode, the LLCS can force the vehicle to turn, to go ahead, and to change speed. Change of depth is not allowed in this mode. In turn, in the autonomous mode, the LLCS continuously works as a depth and course controller, desired values of course and depth it gets from the HLCS. The BAUV is not equipped with speedometer and that is why the LLCS cannot play the function of a speed controller.

The energy system consists of batteries and the software called Battery Control System whose task, as the name implies, is to ensure optimal use of the batteries, and to inform about critical and emergency situations requiring a response, e.g. low voltage on batteries, battery failure.

With regard to sensors, the BAUV has two cameras and one sonar. One camera is mounted in an external container above the hull and it is mainly used on the surface. The second camera is located in the "head" of the BAUV which is always below the surface and for that reason it is only used to observe underwater environment. The sonar is an acoustic "radar" which like the second camera is applied under the water. Currently, all the three sensors are only used to record vision data which can be then analyzed by the operator, information provided by them is not used by the BAUV in the process of making decisions. To handle the sensors two software systems are necessary, i.e. Sonar Control System (SOS) and Camera Control System (CCS), which are run on a separate auxiliary on–board PC104 computer.

The collision avoidance system consists of three echo–sounders and the software responsible for: (i) reading signals from the devices, (ii) interpretation of signals, and (iii) calculating coordinates of obstacles and sending them to the HLCS. The system is able to detect obstacles 40 meters away from the vehicle and located at the top, bottom and front of it.

Navigation system is responsible for locating the BAUV in a global Earth coordinate system. It is equipped with VN200 Inertial Navigational System (INS), Digital Compass OS5000, pressure sensor, GPS receiver, and USBL[2] transponder. As mentioned above, the BAUV does not have speedometer. All the devices mentioned above enable the BAUV to fix position in a different way. First, when the vehicle is on the surface, accurate GPS position can be recorded, second, when the vehicle works close to a mother ship or in a region where USBL system can be installed, USBL position can be used, otherwise, a dead reckoning method is applied to calculate the BAUV position based on INS, Compass and pressure sensor indications. In the latter case, speed is determined by means of odometry, that is, there are a number of propeller settings, each setting corresponds to a vehicle speed.

[2]Ultra–Short BaseLine – underwater acoustic navigational system

Navigation system is not only the hardware, it also needs software components. First, a number of device drivers is necessary with the responsibility for reading navigational data from the devices and for providing them to other software components via shared memory of the main on–board computer. Second, the system is also necessary which, depending on the situation, selects sources of navigational information (GPS, USBL, or INS, Compas, pressure sensor along with dead reckoning) and relying on them fixes or calculates position of the BAUV.

In order to communicate with the operator and with the external system which provides USBL positions to the BAUV, the vehicle is equipped with the communication system. It works in three communication channels, i.e. acoustic, radio and WiFi channel. To this end, it uses three communication devices, i.e. hydro–modem, radio–modem, and WiFi station. Effective communication requires also the software which encodes/decodes messages that the BAUV sends/receives.

Finally, the BAUV needs also the system which, on the one hand, integrates all other systems, devices, and sensors mentioned above in one super–system, but on the other hand, it also makes decisions being the consequence of mission performed by the vehicle in autonomous control mode. The role outlined above is played by the HLCS which is the main subject of this paper and which is detailed in the following section. The HLCS along with other software but except those operating on sensors, is run on the main on–board PC104 computer.

3 High Level Control System

As already mentioned the HLCS is responsible for the following tasks:

1. executing orders of the operator in remotely operated mode,
2. providing selected information to the operator,
3. performing a mission defined by the operator,
4. mapping,
5. detecting emergency situations and responding to them

In order to perform the above tasks, two–threaded implementation of the HLCS is applied [3]. One thread handles all messages and signals coming from the operator or other software components of the BAUV. There are a number of messages, i.e. configuration messages, a message with an operation to run in remotely operated mode, a message with USBL position, a message with mission definition (main or emergency mission), and messages running/stopping the mission. Signals inform the HLCS about important events, there are also a number of signals, i.e.: state signal from other BAUV systems, timer signal, obstacle detection signal. At the highest level, implementation of that HLCS thread can be presented in the form of pseudo–code shown in Fig. 2, 3, and 4.

As for the initialization, it establishes communication between the HLCS and the remaining components of the entire system, sets timers (timers are outlined further), and default emergency mission which, if not modified by the operator, is run when emergency situation is detected. Moreover, the initialization also configures all the HLCS sub–components through reading default parameters of the components from the configuration file.

When a message is received, first, it is tested whether handling messages is allowed. They cannot be handled when an emergency situation is detected which makes it impossible to use the BAUV. In

[3] A solution with only two threads is mainly due to application of four–core PC104 computers on the BAUV board. A small number of cores forced an architecture of the HLCS which is not intensely concurrent. The same applies to other BAUV software components

```
HLCS::main(parameters)
begin
initialization(parameters);
loop forever
    messageOrSignal = receiveMessage();
    if messageOrSignal is message
        handleMessage(messageOrSignal);
    else
        handleSignal(messageOrSignal);
    end if
end loop
end
```

Figure 2. Main thread of HLCS

```
HLCS::handleMessage(message)
begin
if messages are allowed
    switch(message type)
        case configuration:
                configureHLCS(message);
        case remote operation:
                runOperation(message);
        case USBL position:
                setBAUVPosition(message);
        case mission definition:
                setMission(message);
        case mission run/stop:
                runStopMission(message);
    end switch
end if
end
```

Figure 3. Handling messages

that situation, the only thing the vehicle is allowed to do is to periodically send a message with its position.

Configuration message sets different parameters of the HLCS which by default are determined in the configuration file. The message can include a single parameter to update or it can also force the system to return to default values.

In the HLCS, the assumption is made that a desired behavior of the BAUV can be achieved in two ways, i.e. either by running a single operation or running a mission which consists of the operations. An operation is a simple behavior of the vehicle, e.g. turning right, submerging, turning on the camera, moving to a way–point, whereas the mission is an organized execution of many operations, some operations are run sequentially, i.e. one after another, and the other ones are run quasi–concurrently (the mission is detailed further). In the remotely operated mode, the operator is allowed to run only a single operation by sending an appropriate message to the BAUV and the HLCS. If the operation in the message is "not blocking", that is, it does not require a longer activity from the HLCS, then, it is run in the main HLCS thread, otherwise, another thread is started with a "blocking" operation. An example of "not blocking" operation is turning right or left, which is immediately sent to the LLCS, the HLCS is not involved in its execution, whereas, moving to a way–point is an example of "blocking" operation, the HLCS has, in this case, to supervise its execution.

A next message that can be received by the HLCS is the message with USBL position. The position is given to the navigational system which decides what to do next with it.

In order to define, start or stop mission, appropriate messages have to be sent to the HLCS. There are two types of mission, i.e. main and emergency missions, the former has to be defined by the operator, whereas the latter is present in the system in a default form, however, with the possibility to change it. The message that defines the mission prepares the HLCS to run it, that is, internal representation of the mission is created in the form of appropriately organized operations ready to run. After defining the mission, it can be run by sending a proper message. It takes place by creating and running a separate mission thread which can be stopped in two situations solely, i.e. after completion of the mission or when a message is received which means intention of the operator to interrupt it.

```
HLCS::handleSignal(signal)
begin
switch(signal type)
    case timer message to operator:
                sendMessage();
    case timer emergency:
                testIfEmergency();
    case timer clear obstacles:
                clearObstacles();
    case obstacle:
                addObstacle(signal);
    case depth:
                setDistanceToBottom(signal);
    case state:
        updateStateAndTesIftEmergency(signal);
end switch
end
```

Figure 4. Handling signals

Whereas the ordinary messages are mainly used in communication between operator and the HLCS and their role is to send informations or commands requiring a greater amount of data, the signals are short, quick messages and they are used in internal communication between software components of the BAUV.

A separate class of signals are signals from a timer which is a mechanism that makes it possible to control the system by means of time. In the HLCS, the timers generate periodically, according to their setting, three types of signals. One signal activates procedure of sending message to the operator. The message includes informations about state of the BAUV, which informations are sent depends on HLCS setting. A next signal runs procedure which tests whether emergency mission should be run, and if yes, it is started. An example reason of emergency situation is missing course information for a time, or detection of water in the hull. The last signal from the timer is signal which results in browsing the set of obstacles memorized in the system and removing outdated obstacles.

The obstacles are added to a environment map managed by the HLCS by other signal generated by the collision avoidance system. During navigation, all the obstacles contained in the map are considered when maneuvers for the BAUV are calculated. The more obstacles is on the map, the slower the process of maneuver fixing is. Since positions of the obstacles in the global coordinate system can be inaccurate – it is only important for the obstacles to be accurately positioned with reference to the vehicle, outdated obstacles which are far away from the vehicle can be removed.

Apart from the signal with the obstacle, the collision avoidance system generates also a signal with current distance to the sea–bottom. As in the previous case, the information carried by the signals is necessary for save navigation, the vehicle simply should always move above the bottom, each hitting the bottom can be dangerous for the vehicle.

In order to monitor state of the BAUV, the HLCS is periodically informed by all vehicle components about their state. To this end, each component sends to the HLCS a signal with encoded state.

Lack of signal for a time or a signal with the information about an error are often the basis for starting the emergency mission.

In contrast to the main thread of the HLCS which deals with the majority of tasks assigned to the system, the second thread is used is used to run the "blocking" operations and missions. The mission consists of three lists of operations. One list includes operations which are run one after another, and they are called "A" operations. An example of an "A" operation is "go to way–point". A sequence of such operations means a path of the BAUV passing through some way–points.

```
operation::run()
begin
if condition to activate operation
    not satisfied
    reset();
    return true;
end if
switch(state)
    case state1:
        take action in state1;
        if condition in state1 satisfied
            state := state2;
        end if
        return false;
    case state2:
        take action state2;
        return false;
end switch
end
```

Figure 5. Example operation

The second list contains the so–called "B" operations which are executed quasi–concurrently. It is assumed that, one "A" operation and all "B" operations can work simultaneously. In fact, only one operation is active in a given moment, however, the effect and impression on the outside is that all mentioned operations work together. To avoid the situation in which both an "A" operation and some "B" operations at the same time try to control the BAUV propellers, with unpredictable effect, "B" operations cannot affect vehicle drive, it is only allowed to "A" and "C" operations (see further). The task of a "B" operation can be for example to turn on the camera in specified way–points and for a definite period. To this end, the operation has to continuously, for all the mission, monitor vehicle position and start the camera when the vehicle is close to any way–point.

The above–mentioned "C" operations are system operations, i.e. operations which are added to the mission not by the operator but by the system. Their responsibility is to perform tasks common to all missions, e.g. to emerge the vehicle every n meters to get GPS position, and to solve problems which can appear during the mission and which cannot be solved by "A" and "B" operations, e.g. to control the vehicle when collision or when the vehicle cannot move even though its drive is working. Since "C" operations can influence on vehicle propellers, they cannot work together with "A" operations. In consequence, once any "C" operation is started, current "A" operation is immediately interrupted.

The "C" operations are arranged according to the importance criterion. In one moment, only one "C" operation can be active. All them test, one after another, conditions necessary to activate them. Once the conditions are satisfied, the operation is started and current "A" operation is stopped. While one of "C" operations is active, the remaining "C" operations continue to test their conditions. If conditions of a "C" operation with a greater priority are also met, it takes control over the vehicle, whereas, "C" operation active so far has to be stopped. After completion of all "C" operations, an interrupted "A" operation is reseted and restarted.

```
HLCS::runMission(mission)
begin
currentAOperation := 0;
currentCOperationActive := -1;
reset := false;
loop until (currentAOperation < mission.listOfAOperations.size()
        and mission not interrupted)
    currentCOperation := -1;
    cOperationNotActive := true;
    loop until (currentCOperation < mission.listOfCOperations.size()
            and cOperationNotActive = true)
        currentCOperation++;
        cOperationNotActive := mission.listOfCOperations[currentCOperation].run();
    end loop
    if cOperationNotActive = true
        mission.listOfBOperations.runAll();
        currentCOperationActive := -1;
        reset := false;
        if mission.listOfAOperations[currentAOperation].run() is end
            currentAOperation++;
    else
        if (currentCOperationActive >=0
            and currentCOperationActive != currentCOperation)
            mission.listOfCOperations[currentCOperationActive].reset();
        end if
        currentCOperationActive := currentCOperation;
        if reset = false
            mission.listOfAOperations[currentAOperation].reset();
            reset := true;
        end if
    end if
end loop
end
```

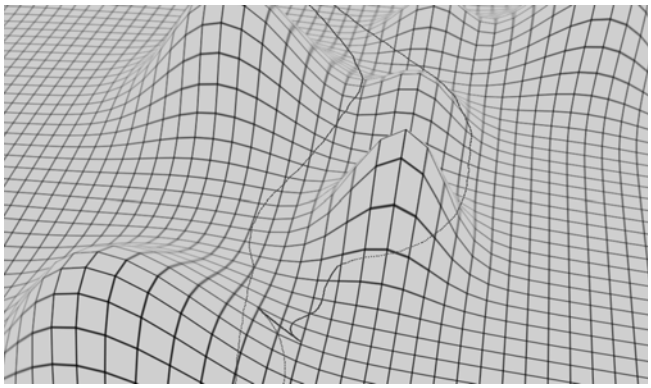
Figure 6. Algorithm of mission

Ultimately, algorithm of mission can be presented in the form of pseudo–cod depicted in Fig. 6.

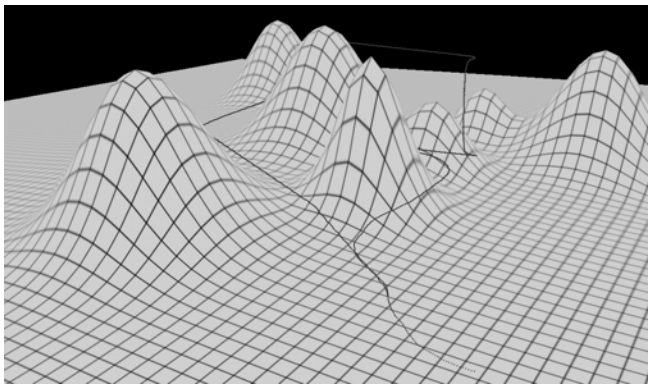
As already mentioned, operations contained in the mission can work quasi–concurrently. To achieve such an effect, the main mission loop runs all operations, which are supposed to be run concurrently, one after another. However, execution of an operation in the loop does not mean that the operation runs from the beginning to the end. If so, each operation would block other operations. Each operation is a state machine which in a single run takes only actions that correspond to a state in which the operation currently is. Example operation can be presented in the algorithm shown in Fig. 5.

To effectively control the BAUV, the operations often need information about state of different vehicle components, including the HLCS itself. For example, "go to way–point" operation has to know whether it can lead the vehicle directly to the way–point or it has to perform a collision avoidance maneuver. Providing the information to the operations is the task of the HLCS. Each component of the BAUV stores its state in the shared memory from where it can be downloaded by the HLCS which, in turn, can provide it to any operation.

The HLCS with the architecture described above confirmed its effectiveness during many tests in simulation [3–5] and with real vehicle. In fact, the tests in simulation were even more valuable because they could be carried out in any conditions. During simulation, all elements that could have the influence on the operation of HLCS were tested. The system was verified for the vehicle with different dynamics, for various underwater environments, for different vehicle and HLCS parameters, for different operations and so forth. Example paths of BAUV during tests in simulation are depicted in Fig. 7.



(a)



(b)

Figure 7. Example paths of BAUV during tests in simulation

Moreover, the tests in simulation made it also possible to eliminate implementation errors which appeared in the software. Appearance of the errors during tests with the real vehicle could result in damaging or losing it.

The tests on the real vehicle were also useful, first they were carried out on a pool and next in a real conditions, that is, at sea. As it appeared, the HLCS as well as the remaining components of the system passed also this exam. In the tests, the vehicle, generally behaved as we expected.

4 Summary

The paper presents the High–Level Control System for Biomimetic Autonomous Underwater Vehicle designed within the project "Autonomous underwater vehicles with silent undulating propulsion for underwater ISR", financed by Polish National Center of Research and Development. The tasks of the system are itemized and its the most crucial elements are outlined, including: handling messages and signals, cooperation with other systems, construction and modus operandi of mission, and operations. In order to allow for better understanding of the presented material, the paper includes some algorithms.

Acknowledgment

The research described in this paper were achieved in the project No. DOBR-BIO4/033/13015/2013, financed by Polish National Center of Research and Development in the years 2013-2016.

References

[1] M. Malec, M. Morawski, and J. Zajc, "Fish-like swimming prototype of mobile underwater robot", Journal of Automation, Mobile Robotics and Intelligent Systems, Vol. 4, No 3, pp. 25-30, (2010)

[2] T. Praczyk, P. Szymak, Software architecture of biomimetic underwater vehicle, Proc. SPIE 9831, Ground/Air Multisensor Interoperability, Integration, and Networking for Persistent ISR VII, doi:10.1117/12.2225871, (2016)

[3] T. Praczyk, A quick algorithm for planning a path for biomimetic autonomous underwater vehicle, Scientific Journals of Maritime University of Szczecin, no. 45 (117), pp. 23-28, (2016)

[4] T. Praczyk, Using Genetic Algorithms for Optimizing Algorithmic Control System of Biomimetic Underwater Vehicle, Computational Methods in Science and Technology (CMST), vol. 21 (4), pp. 251-260, (2015)

[5] T. Praczyk, The influence of parameters of biomimetic underwater vehicle control system on the ability of the vehicle to avoid obstacles, Scientific Journal of Polish Naval Academy, 2(205), pp. 75-91, (2016)

[6] P. Szymak, T. Praczyk, K. Naus, M. Malec, M. Morawski, Research on biomimetic underwater vehicles for underwater ISR, Ground/Air Multisensor Interoperability, Integration, and Networking for Persistent ISR VII, edited by Michael A. Kolodny, Tien Pham, Proc. of SPIE Vol. 9831, 98310K, (2016)

[7] P. Szymak, M. Malec, M. Morawski, "Directions of development of underwater vehicle with undulating propulsion", Polish Journal of Environmental Studies, Hard Publishing Company, Vol.19, No 3, pp. 107-110, (2010)