25

Decentralized Control System Simulation for Autonomous Underwater Vehicles

Nanang Syahroni[1], Young Bong Seo[2] and Jae Weon Choi[2]
[1]*Electronics Engineering Politechnic Institute of Surabaya*
[2]*School of Mechanical Engineering, Pusan National University*
[1]*Indonesia*
[2]*Korea*

1. Introduction

The problem of power and communication limitation in underwater environment makes it more challenge to increase the degree of autonomy and intelligence for an autonomous underwater vehicle (AUV). An infrastructure of autonomous teleoperation platform for AUV is established and described, which allows control to be shared between the intelligent decision system in AUV system and operators throughout a mission.

In the (Paunicka J.L. et al., 2001, Samad T. et al., 2003, and Wills L. et al., 2000), the information-centric control and engineering have a remarkably successful history of enabling for designing, testing, and transitioning embedded software to unmanned air vehicle (UAV) platforms. A new software infrastructure called Open Control Platform (OCP) will accommodate in changing navigation information and control components, interoperate in heterogeneous environments, and maintain viability in unpredictable and changing environments. The OCP extends new advances in real-time middleware technology, which allows distributed heterogeneous components to communicate asynchronously in real-time via CORBA middleware. It uses event-based distributed communication and it capable of transmitting events at different priorities. This enables highly decoupled interaction between the different components of the system, which tends to localize architectural or configuration changes that promising to be implemented quickly and high reliability in the real system.

There are many examples of nice control algorithms for AUV which had done in several platforms (Valavanis K.P. et al., 1997), but in the implementation of those control systems in the sense of tightly coupling model in remote operation is widely open for sub-discipline of software engineering. We further investigate how the real-time control system performance could be reconfigured easily both in semi-automatically or manually interventions by remote station, and also develop a simulation platform to support a tuning mechanism of control parameters during runtime (i.e. feedback gains or trajectories) by using Matlab on separated machines connected via CORBA event-channel.

In this paper we organized as follows: Section 2 presents AUV dynamic model, physical values, and control algorithm. Section 3 gives the simulation systems design; include the

hardware of simulation workstation, tools and interfaces, and middleware infrastructure. Section 4 presents results from the simulations together with the assumptions of problems solution. The last section covers conclusions.

2. Equations Of Motion

2.1 AUV Dynamic Model

The AUV Model for depth control is depicted in figure 1.

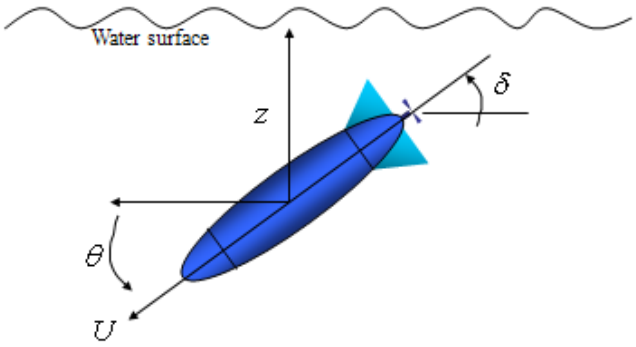


Fig. 1. AUV Model

The simple's form of equation of motion is obtained with body axes coincident with the principles axes of inertia, and the origin at the center of mass center of gravity (CG), for this case the equation in the dimensionless form as in (Sname 1950) are:

$$\begin{aligned} X &= m[\dot{u} + qw - \upsilon r] \\ Y &= m[\dot{\upsilon} + ru - pw] \\ Z &= m[\dot{w} + pv - qu] \\ K &= I_x \dot{p} + (I_z - I_y)qr \\ M &= I_y \dot{q} + (I_x - I_z)rp \\ N &= I_z \dot{r} + (I_y - I_x)pq \end{aligned} \tag{1}$$

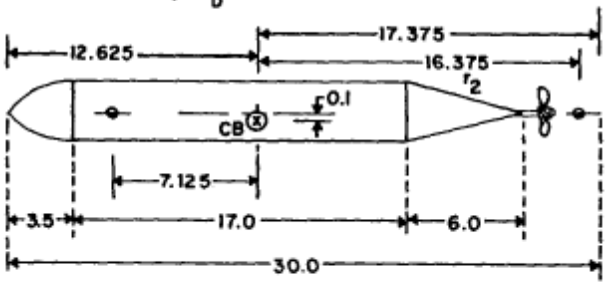
The 6DOF components of the rigid body dynamic equations of motion of the submerged vehicle are:

$$
\begin{aligned}
X &= m[\dot{u} - \upsilon r + wq - x_G(q^2 + r^2) + y_G(pq - \dot{r}) + z_G(pr + \dot{q})] \\
Y &= m[\dot{\upsilon} + ur - wq + x_G(pq + \dot{r}) - y_G(p^2 + r^2) + z_G(qr - \dot{p})] \\
Z &= m[\dot{w} - uq + \upsilon p + x_G(pr - \dot{q}) + y_G(qr + \dot{p}) - z_G(p^2 + q^2)] \\
K &= I_x\dot{p} + (I_z - I_y)qr + I_{xy}(pr - \dot{q}) - I_{yz}(q^2 - r^2) - I_{xz}(pq + \dot{r}) + m[y_G(\dot{w} - uq + \upsilon p) - z_G(\dot{\upsilon} + ur - wp)] \\
M &= I_y\dot{q} + (I_x - I_z)pr - I_{xy}(qr + \dot{p}) - I_{yz}(pq - \dot{r}) - I_{xz}(p^2 - r^2) + m[x_G(\dot{w} - uq + \upsilon p) - z_G(\dot{u} - \upsilon r + wq)] \\
N &= I_z\dot{r} + (I_y - I_x)pq - I_{xy}(p^2 - q^2) - I_{yz}(pr + \dot{q}) + I_{xz}(qr - \dot{p}) + m[x_G(\dot{\upsilon} + ur - wp) - y_G(\dot{u} - \upsilon r - wq)]
\end{aligned}
\tag{2}
$$

where, X, Y, and Z are surge, sway, and heave force; K, M, and N are roll, pitch, and yaw moment; p, q, and r are roll, pitch, and yaw rate; u, v, and w are surge, sway, and heave velocity; x, y, and z are body fixed axes in positive forward, positive starboard, and positive down; I_x, I_y, and I_z is vehicle mass moment of inertia around the x-axis, around the y-axis, and around the z-axis; x_G, y_G, and z_G are longitudinal position, athwart position, and vertical position of center of gravity; ϕ, θ, and ψ are roll, pitch, and yaw angle.

PRINCIPLE DIMENSIONS

MODEL : WEIGHT, W = 19.6 lbs
MASS, M = 0.609 slugs
LENGTH, L = 30. inches
BREADTH, B = 7.0 inches
HEIGHT, T = 3.5 inches
RESTORING ARM, BG = 0.1 inch
DISPLACED VOLUME, V_D = 543 in^3



DIVE PLANES : *FWD*

MOMENT ARM, r_1 = 7.6 inches
MEAN CHORD, $\bar{c}$ = 1.9 inches
MEAN SPAN, b = 2.5 inches
AREA (EACH), A = 4.75 in^2
GEOMETRIC ASPECT RATIO, a_g = 1.316
EFFECTIVE ASPECT RATIO, a = 2.63

AFT

MOMENT ARM, r_2 = 16.375 inches
MEAN CHORD, $\bar{c}$ = 1.6 inches
MEAN SPAN, b = 2.0 inches
AREA (EACH), A = 3.2 $inches^2$
GEOMETRIC ASPECT RATIO, a_g = 1.25
EFFECTIVE ASPECT RATIO, a = 2.5

Fig. 2. Physical Dimensions of Vehicle

We can further simply equations (2) by assuming that y_G is small compared to the other terms. After several steps of linearization as in (Cristi, R. et al., 1990, and Riedel J.S., 1993), vertical motion equations become:

$$\begin{aligned} \dot{\theta} &= q \\ (m - Z_{\dot{w}})\dot{w} - (mx_G + Z_{\dot{q}})\dot{q} &= Z_w Uw + (m + Z_q)Uq + U^2 Z_\delta \delta \\ (-M_{\dot{w}} - mx_G)\dot{w} + (I_y - M_{\dot{q}})\dot{q} &= -(z_G W - z_B)\theta + M_w Uw + (M_q - mx_G)Uq - M_\delta U^2 \delta \\ \dot{z} &= -U\theta + w \end{aligned} \tag{3}$$

It can rewritten in the matrix form

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & (m - Z_{\dot{w}}) & -(mx_G + Z_{\dot{q}}) & 0 \\ 0 & (-mx_G - M_{\dot{w}}) & (I_y - M_{\dot{q}}) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \dot{\theta} \\ \dot{w} \\ \dot{q} \\ \dot{z} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & Z_w U & (m + Z_q)U & 0 \\ -(z_G W - z_B B) & M_w U & (M_q - mx_G)U & 0 \\ -U & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} \theta \\ w \\ q \\ z \end{bmatrix} + \begin{bmatrix} 0 \\ Z_\delta U^2 \\ M_\delta U^2 \\ 0 \end{bmatrix} \delta \tag{4}$$

Then, an AUV dynamic equation typically represented using the notation: $\dot{x} = Ax + Bu$, with state vector $x = [\theta \quad w \quad q \quad z]^T$ and the control input $u = \delta$:

$$\begin{bmatrix} \dot{\theta} \\ \dot{w} \\ \dot{q} \\ \dot{z} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ a_{21} z_{GB} & a_{22} U & a_{23} U & 0 \\ a_{31} z_{GB} & a_{32} U & a_{33} U & 0 \\ -U & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} \theta \\ w \\ q \\ z \end{bmatrix} + \begin{bmatrix} 0 \\ b_1 U^2 \\ b_2 U^2 \\ 0 \end{bmatrix} \delta \tag{5}$$

where:

$$Dv = (m - Z_{\dot{w}})(I_y - M_{\dot{q}}) - (mx_G + Z_{\dot{q}})(mx_G + M_{\dot{w}})$$

$$z_{GB} = z_G - z_B$$

$$a_{21} = -\frac{(mx_G + Z_{\dot{q}})W}{Dv}$$

$$a_{22} = \frac{(I_y - M_{\dot{q}})Z_w + (mx_G + Z_{\dot{q}})M_w}{Dv}$$

$$a_{23} = \frac{(I_y - M_{\dot{q}})(m + Z_q) + (mx_G + Z_{\dot{q}})(M_q - mx_G)}{Dv}$$

$$a_{31} = \frac{-(mx_G - Z_{\dot{q}})W}{Dv}$$

$$a_{32} = \frac{(m - Z_{\dot{w}})M_w + (mx_G + M_{\dot{w}})M_\delta}{Dv}$$

$$a_{33} = \frac{(m - Z_{\dot{w}})(Mq - mx_G) + (mx_G + M_{\dot{w}})(m - Z_q)}{Dv}$$

$$b_1 = \frac{(I_y - M_{\dot{q}})Z_\delta + (mx_G + Z_{\dot{q}})M_\delta}{Dv}$$

$$b_2 = \frac{(I_y - Z_{\dot{w}})M_\delta + (mx_G + M_{\dot{q}})Z_\delta}{Dv}$$

Finally, we refer to the physical parameter of NPS AUV1 in the (Healey A.J. et al., 1997) as depicted in figure 2, the state space equation became:

$$\begin{bmatrix} \dot{\theta} \\ \dot{w} \\ \dot{q} \\ \dot{z} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0.0175 & -1.273 & -3.559 & 0 \\ -0.052 & 1.273 & -2.661 & 0 \\ -5 & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} \theta \\ w \\ q \\ z \end{bmatrix} + \begin{bmatrix} 0 \\ 0.085 \\ 21.79 \\ 0 \end{bmatrix} \delta \tag{6}$$

2.2 LQR Formulation

The general problem of Linear quadratic (LQ) optimal control problem to find a control law $u(t) = -Kx(t)$ to minimize $J = \int_0^\infty \left[x^T(t)Qx(t) + u^T(t)Ru(t) \right] dt$, where $K = R^{-1}B^T P$ and P is solution of Riccati equation $A^T P + PA + PBR^{-1}B^T P + Q = 0$, and weighting matrices are: $Q \geq 0$ and $R > 0$.

Linear quadratic (LQ) servo is command following regarding to the reference input. An nth-order system having r inputs and m outputs: $x(t) = \left[x_r(t); y_p(t) \right]$, where $y_p(t) \in R^{m \times 1}$ is output and $x_r(t) \in R^{(n-m)\times 1}$ is rest of system state, $e(t) = r(t) - y(t) \in R^{m \times 1}$ is the error vector with $r(t) \in R^{m \times 1}$ is plan output and $y(t) \in R^{m \times 1}$ is reference. Consider state space model:

$$\dot{x}(t) = Ax(t) + Bu(t) \tag{7}$$

with $y(t) = C_p x(t)$, $x_r(t) = D_p x(t)$, and C_p is $\left[0_{m \times (n-m)} \quad I_{m \times m} \right]$, D_p is $\left[I_{(n-m)\times(n-m)} \quad 0_{(n-m)\times m} \right]$.

Consider control input $u(t) = -Gx(t)$, and control gain G consists of $G = \begin{bmatrix} G_y & G_r \end{bmatrix}$, where the gain vector is $G_y \in R^{m \times m}$ and $G_r \in R^{m \times (n-m)}$. The control law than became:

$$u(t) = -G_y y(t) - G_r x_r(t) + G_y r(t) \tag{8}$$

substitute (8) into (7), we have differential equation of close-loop system as follows:

$$\dot{x}(t) = \left[A_r - BG_y C_p - BG_y D_p \right] x(t) + BG_y r(t) \tag{9}$$

Solution of equation (9) could be solving by using Runga-Kutta 4th order approximation:

$$x(t+1) = x(t) + \frac{(k1 + 2k2 + 2k3 + k4)}{6} \tag{10}$$

where:

$$k1 = f(t, x(t))$$
$$k2 = f(t + \frac{h}{2}, x(t) + \frac{h}{2} k1)$$
$$k3 = f(t + \frac{h}{2}, x(t) + \frac{h}{2} k2)$$
$$k4 = f(t + h, x(t) + hk3)$$

2.3 Q, R Selection via GA

In order to choose the weighting matrices of LQR control algorithm, it is often by trial and error method. Another method to find an expected values of weighting matrices component is using guided random search, this is one of the easiest way and it could be handle by search function in Matlab simulation source. The most simple and well known technique for guided random search in machine learning application is using the Genetic Algorithm (GA), as shown in figure 3.

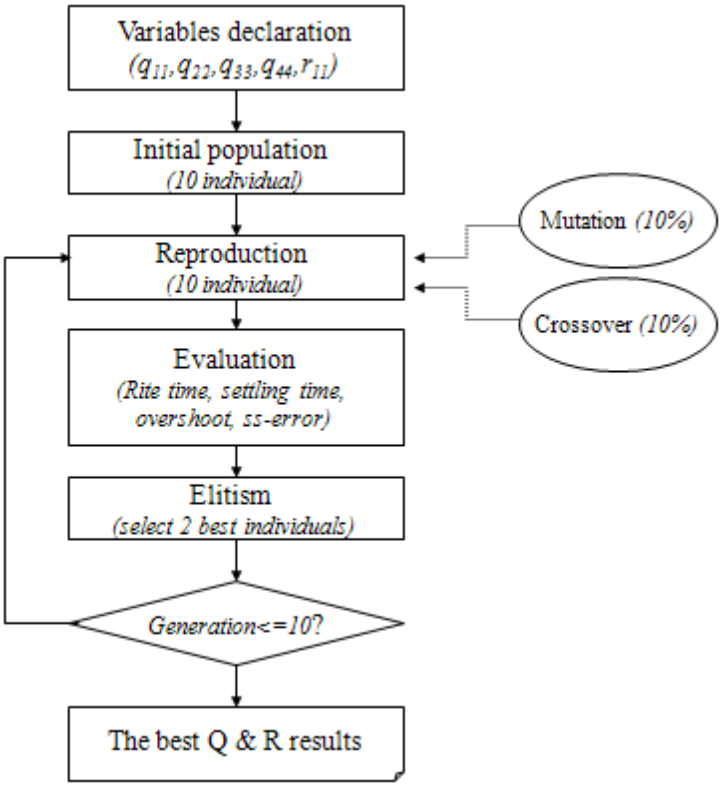


Fig. 3. GA Flowchart

Genetic algorithm performed chromosome operation and individual evaluation. In the first iteration, chromosome operation generates the initial population consists of 10 populations randomly with logarithmic distribution. Each population consists of 5 chromosomes, 4 chromosomes of Q vector matrix, and 1 chromosome of R matrix. Individual evaluation consists of selection procedure according to the fitness function variables of time domain characteristic; such as maximum overshoot tolerance is 10%, maximum rise time tolerance is 15%, maximum settling time tolerance is 10%, maximum steady-state error tolerance is 0.2%, also to check the location of the close-loop poles whether it near by imaginary axis. If there are several values are passed through those selection criterions than GA choose 2 best values, then it transfer to become a new candidate in next generation selection together with new 8 populations, where 6 from randomly generated and the 2 other new populations from both crossover and mutation from 2 best previous populations. Finally, in the end of iteration only one best value is selected.

An Implementation of GA into LQ Servo control algorithm as depicted in figure 4. The purpose of this combination is to obtain the global optimal feedback gain K, which could be

change anytime during runtime. The purpose of changing K is to suppress undesired controller performances.

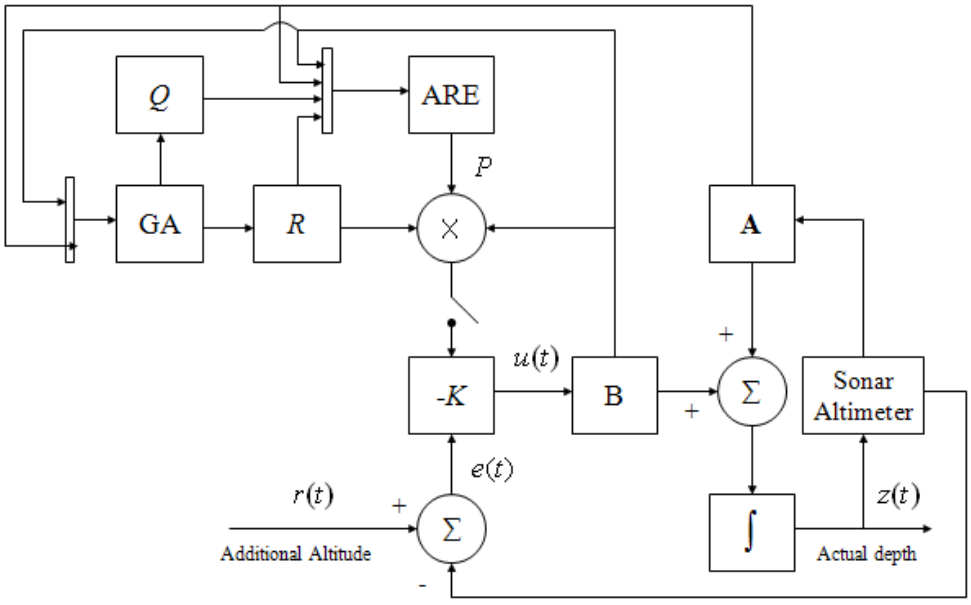


Fig. 4. LQ Servo with GA

3. System Environment

In this paper, an AUV decentralized control system approach is investigated, this will open a possibility to enable control system component to interact between various control components on the simulation network infrastructure. During the course of theoretical studies to simulation platform development, an OCP has emerges some promises to overcome any boundary for both in control system domain and network infrastructure domain.

3.1 Simulation Workstation

To implement the proposed AUV depth control system simulation, information may needs to be rerouted between AUV subsystems or control components. In this situation, sometimes a certain data may became temporarily very important and at other time not needed at all. In figure 5, a simulation system consist of two node connected with a general 10 Mb Ethernet, PC1 as a server and PC2 as client, every node will consists of two blocks, first block consist of vehicle model and control algorithm, and second block is consist real-time network components that support hard control reconfiguration.

The two PCs as server and client are connected via middleware communication using TAO-CORBA Event channel, which is described more details in (Schmidt D. C. et al., 2000). A PC1 running a Matlab simulation of vehicle model and control algorithm, while PC2 running a

Matlab simulation as sensor source and mission control station to allows a user to dynamically modify any parameters during a runtime simulation.

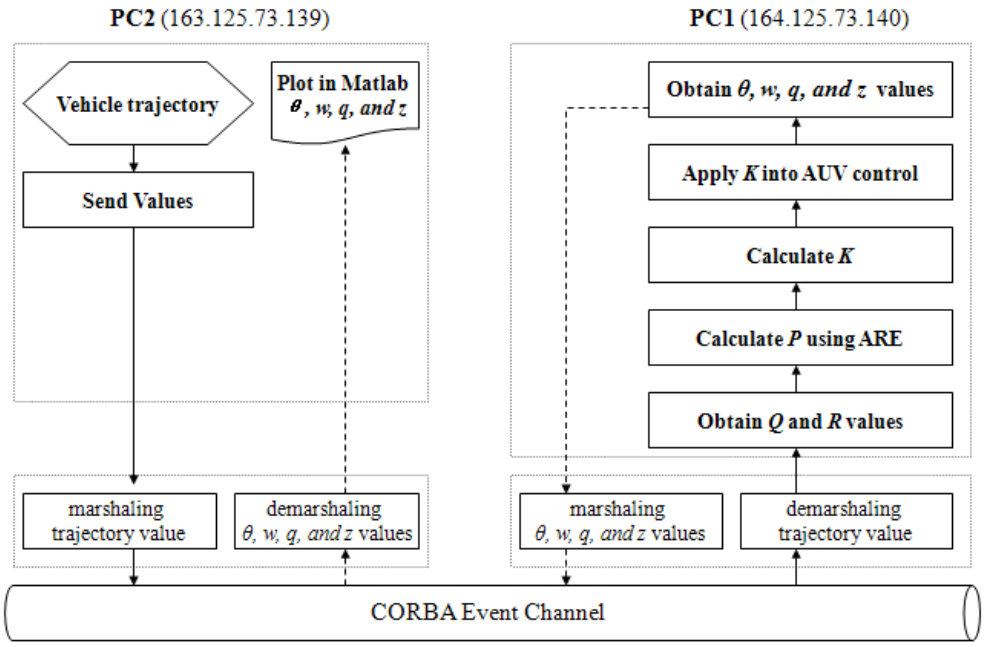


Fig. 5. AUV Simulation Block Diagram

3.2 Simulation Tools and Interfaces

Generally, in the development step, mostly control engineer test the new control algorithm in Matlab environment. Matlab is a convenient tool for graphical plotting; it is relatively difficult to use C++ to plot system responses in multiple dimensions. However, C++ is widely used in real-time data acquisition and control in industrial applications.

The interface between C++ and Matlab offers a significant improvement in data acquisition and control system analysis. This makes the analysis for complicated systems possible in the real world. Using the interface method, it is much more convenient to perform matrix operations with real-time controllability.

Another point is, for real-time systems, especially for the multiple variables control system, a state variable matrix has to be used to make the real-time analysis based on the state feedback from the system outputs. By using the interface between C++ and Matlab, a lot of data analysis and real-time control tasks for actual systems are possible. Another advantage to using the interface between C++ and Matlab is to handle multi dimension matrix operations and continuous plotting of system responses. Almost all data acquisition and control processes need time response of trajectory in real-time.

In figure 6, illustrates the block diagram of operation principle C++ and Matlab interface. Via this interface, C++ program should collect the data from PC2 through CORBA and

create the data variables in ASCII format. Matlab first picks up the data from the data variables have stored by C++ and then performs the matrix operations based on the data. The results can be sent back to C++ by Matlab in the ASCII variables, while C++ program continue to executing the communication task to send a Matlab results.

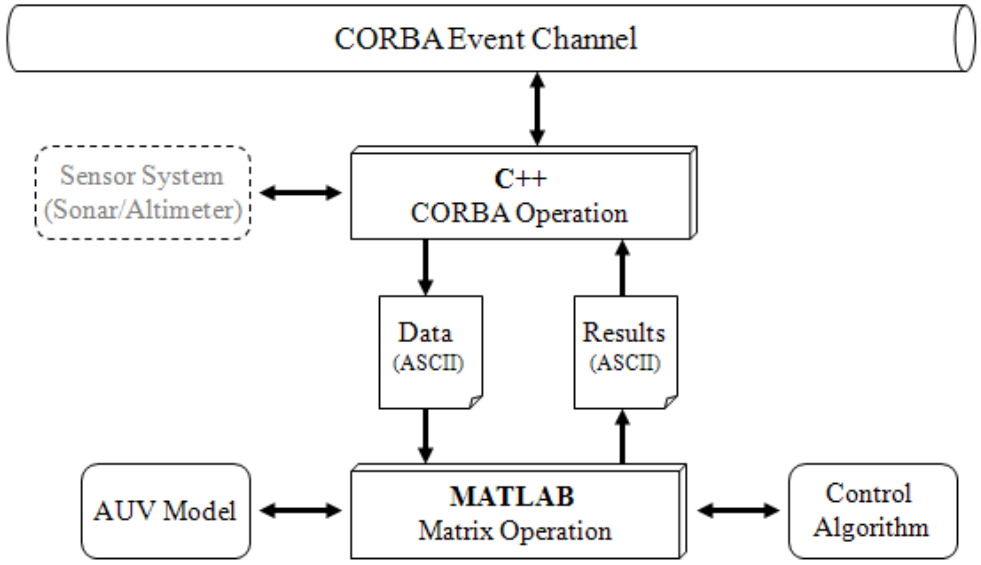


Fig. 6. Matlab and C++ Interfaces

The interface between Matlab and C++ in order to transfer a data through CORBA event channel is not so complicated, although for control engineers, this method offers a significant improvement in data acquisition and control system analysis; this makes the analysis for complicated systems possible in the real world.

3.3 Middleware Infrastructure

The CORBA middleware is an application framework that provides interoperability between objects, built-in different languages, running on different machines in heterogeneous distributed environments. Using a CORBA, a client can transparently invoke a method on a server object, which can be on the same machine or across a network. The ORB intercepts the call and is responsible for finding an object that can implement the request, pass it the parameters, invoke its method, and return the results.
The CORBA event service provides support for decoupled communications between objects. It allows suppliers to send messages to one or more consumers with a single call. The event service acts as a mediator that decouples suppliers from consumers.
In figure 7, a CORBA event service provides a flexible model for asynchronous and group communication among distributed and collocated objects. Consumers are the ultimate targets of events generated by suppliers. Suppliers and consumers can both play active and

passive roles. An active push supplier pushes an event to a passive push consumer. Likewise, a passive pull supplier waits for an active pull consumer to pull an event from it. Suppliers use event channels to push data to consumers. Likewise, consumers can explicitly pull data from suppliers. The push and pull semantics of event propagation help to free consumers and suppliers from the overly restrictive synchronous semantics of the standard CORBA two way communication model.

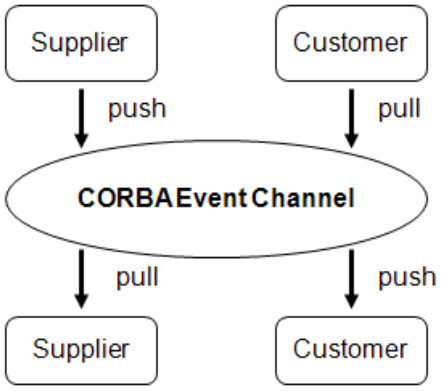


Fig. 7. Participants in the Event Channel Architecture

In this paper, a CORBA event service implementation focuses on real-time enhancements to the push model, which allows suppliers of events to initiate the transfer of event data to consumers. Suppliers push events to the event channel, which in turn pushes the events to consumers.

4. Simulation Results

4.1 Simulation Condition 1

Let consider a first example as depicted in figure 8; in this simulation we want to control θ near zero and z near -2 meter with 30 times counter duration. We use a reasonable amount of dive planes to do the job. Assumption: 4^o dive planes when pitch angle deviates to 5^o from zero, the AUV reaches a depth of -2 meter with 0.32 meter deviation. Therefore, we assume all terms in $Q \rightarrow 0$ and $R \rightarrow 0$, except: $q_{11} = (4/57.2958)^{-2}=205.21$, $q_{44} = (5/57.2958)^{-2}=131.31$, and $r_{11}= (0.32)^{-2}=9.76$, simulation result as illustrated in figure 8 using solid line. To overcome undershoot and overshoot in the runtime simulation, after duration of $t = 4$ seconds PC2 send new weighting matrices to the PC1 to change a pitch angle deviates to 5^o with 0.02 meter deviation at -2 meter depth: $q_{11}=(4/57.2958)^{-2}= 205.21$, $q_{44} = (5/57.2958)^{-2}=131.31$, $r_{11}= (0.02)^{-2}=2500$, simulation result as illustrated in figure 8 using dash line.

The time response of both controller are equal for $t \leq 4$ seconds because of all parameters are same, by intervention from PC2 to PC1 when $t \geq 4$ seconds, the new control parameters are apply during runtime, then it could be seen that time response is improved significantly, especially to suppress undershoot and overshoot.

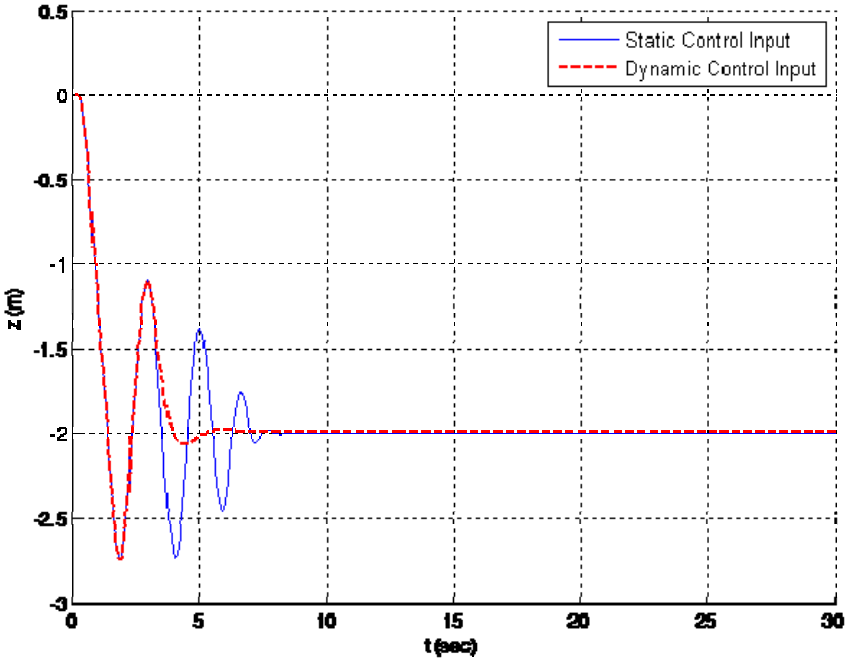


Fig. 8. Online Overshoot &Undershoot Suppression after 4 seconds

4.2 Simulation Condition 2

Similar to the previous simulation, PC1 running the controller gain and system matrix with assuming all terms in $Q \rightarrow 0$ and $R \rightarrow 0$, except: q_{11} = 205.21, q_{44} = 131.31, and r_{11} = 400, as illustrated in figure 9 using solid line. To reduce the rise time duration in the runtime simulation, when $t = 2$ seconds PC2 send a new weighting matrices to the PC1 to change a pitch angle deviates to 10^{o} from zero with 0.02 meter deviation at -5 meter depth: q_{11} = 205.21, q_{44}= 23.83, and r_{11} = 2500, as illustrated in figure 9 using dash line. In this case, a settling time response will be change during runtime when $t \geq 2$ seconds. It could be seen that time response is improved significantly.

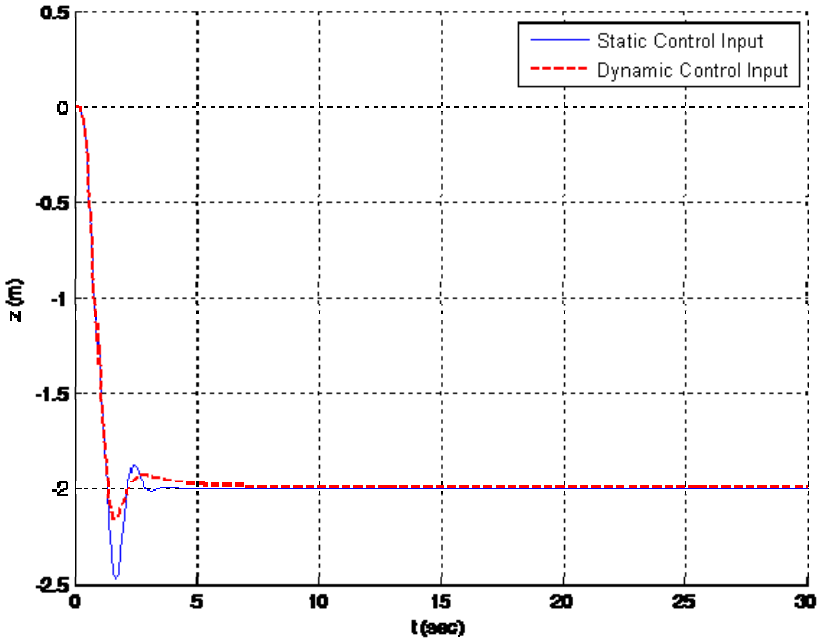


Fig. 9. Online Settling Time Suppression after 2 second.

4.3 Simulation Condition 3

In the first simulation, we perform a static simulation without OCP infrastructure using single weighting matrices, we assume all terms in $Q \rightarrow 0$ and $R \rightarrow 0$, except: q_{11} = 205.21, q_{44}= 23.83, and r_{11} = 2500, as illustrated in figure 10 using dot line. In the second simulation under OCP infrastructure, PC2 send any weighting matrices value to the PC1 (i.e. Q=diag[2,10,20,90], and R=[0.1]). Then GA will find a new weighting matrices, a global optimal value in this runtime all terms in $Q \rightarrow 0$ and $R \rightarrow 0$, except: q_{11} =397.33, q_{44} =98.67 and r_{11} = 1272.8, as illustrated in figure 10 using solid line.

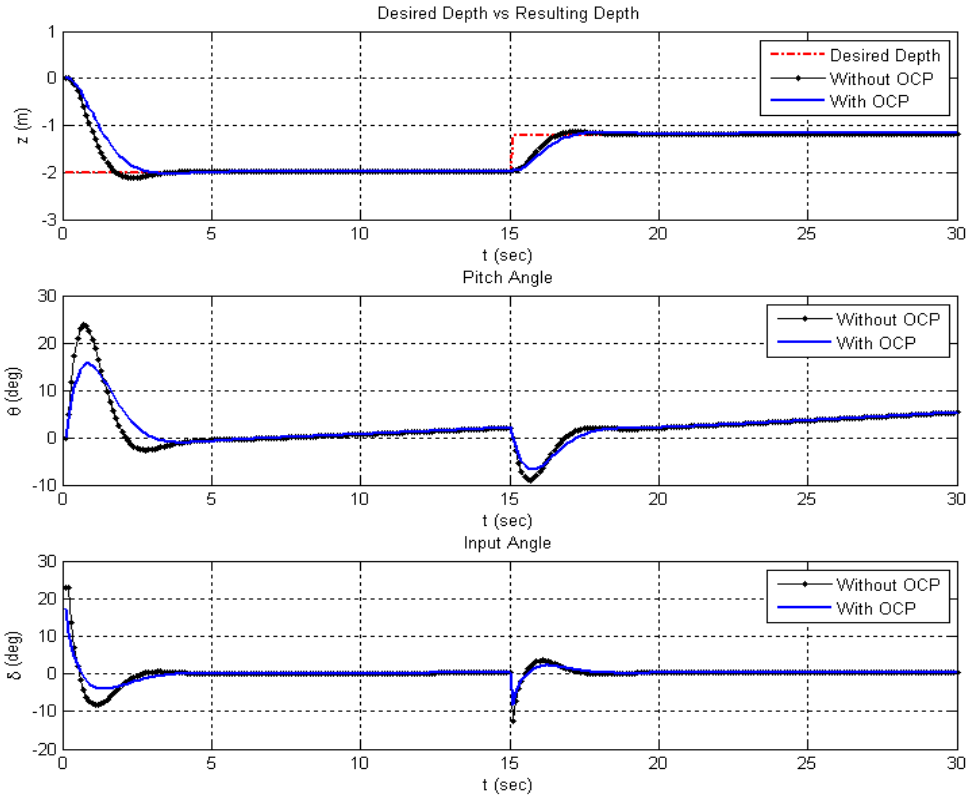


Fig. 10. Depth Control Simulation Result

We have 2 simulation results, with OCP and without OCP, but the performance is similar as depicted in figure 10 because of given system is simple. If the system is more complex using 6DOF with more sensors and actuators we can see the difference performance that OCP is more effective in complicated cases.

5. Concluding Remarks

In this paper, we propose a new approach of decentralized system environment for AUV simulation using Matlab and CORBA event channel coexistence on several machines, we believe it will emerge more investigation how the real-time control system performance could be reconfigured easily both in semi-automatically or manually interventions by a remote station.

In the future research, we expect to uncover the effective CORBA programming to support Matlab and CORBA event channel coexistence that will be affected to increase the degree of real-time reconfigurable control significantly.

6. References

Cristi, R., Papoulias, F.A., and Healey, A.J. (1990), Adaptive Sliding Mode Control of Autonomous Underwater Vehicles in the Dive Plane, In: *IEEE Journal of Oceanic Engineering*, Vol. 15, Issue: 3, page(s): 152-160, Piscataway, NJ, USA.

Healey A.J., Papoulias P.A., and Cristi R. (1989), Design and Experimental Verification of a Model Based Compensator for Rapid AUV Depth Control, In: Proceeding of the Int'l Symposium on Unmanned Untethered Submersible Technology, page(s): 458-474, Washington DC, USA.

Paunicka J.L., Corman W.E.,and Mendel B.R. (2001), A CORBA-Based Middleware Solution for UAVs, In: Proceeding of Fourth International Symposium on Object-Oriented Real-Time Distributed Computing, page(s): 261-267, Magdeburg, Germany.

Riedel, J.S., (1993), Pitchfork Bifurcations and Dive Plane Reversal of Submarines at Low Speeds, Engineer's Thesis, Naval Postgraduate School, Monterey, California, USA.

Samad T. and Balas G. (2003), *Software-Enabled Control: Information Technology for Dynamical Systems*, John Wiley & Sons/IEEE Press, Hoboken, NJ, USA.

Schmidt D. C., and Kuhns K. (2000), An Overview of the Real-time CORBA Specification, In: *IEEE Computer special issue on Object-Oriented Real-time Distributed Computing*. Vol. 33, Issue: 6, page(s): 56-63, Los Alamitos, CA, USA.

Sname, The Society of Naval Architects and Marine Engineers (1950), Nomenclature for Treating the Motion of a Submerged Body Through a Fluid. In: *Technical Research Bulletin*, No. 1-5, NY, USA.

Valavanis K.P., Gracanin D., Matijasevic M., Kolluru R., and Demetriou G.A. (1997), Control architectures for autonomous underwater vehicles, In: *IEEE Control Systems Magazine*, Vol.17, Issue: 6, page(s): 48-64. Ann Arbor, MI, USA.

Wills L., Kannan S.K. (2000), Heck B.S.,Vachtsevanos G., Restrepo C., Sander S., Schrage D.P., and Prasad J.V.R. An Open Software Infrastructure for Reconfigurable Control Systems, In: Proceeding of *American Control Conference,* Vol. 4, page(s): 2799-2803, Chicago, IL, USA.