

UGSim: Autonomous Buoyancy-Driven Underwater Glider Simulator with LQR Control Strategy and Recursive Guidance System

Zhizun Xu, Yang Song, Jiabao Zhu, and Weichao Shi

Abstract

This paper presents the UGSim, a simulator for buoyancy-driven gliders, with a LQR control strategy, and a recursive guidance system. Building on the top of the DAVE and the UUVsim, it is designed to address unique challenges that come from the complex hydrodynamic and hydrostatic impacts on buoyancy-driven gliders, which conventional robotics simulators can't deal with. Since distinguishing features of the class of vehicles, general controllers and guidance systems developed for underwater robotics are infeasible. The simulator is provided to accelerate the development and the evaluation of algorithms that would otherwise require expensive and time-consuming operations at sea. It consists of a basic kinetic module, a LQR control module and a recursive guidance module, which allows the user to concentrate on the single problem rather than the whole robotics system and the software infrastructure. We demonstrate the usage of the simulator through an example, loading the configuration of the buoyancy-driven glider named Petrel-II, presenting its dynamics simulation, performances of the control strategy and the guidance system.

Index Terms

underwater simulation, marine simulation, underwater glider, glider control, glider guidance.

I. INTRODUCTION

In autonomous buoyancy-driven underwater gliders, electric or thermal energy is converted to pressure-volume work to change the vehicle volume to cycle vertically in the ocean. They use lift on wings to project the vertical velocity into forward motion. These gliders are steered by changing the centre of gravity with respect to the centre of the buoyancy, thus controlling both the pitch and the roll. Because of those special designs, there are four distinguishing characteristics for the class of vehicles are: 1)the use of buoyancy, 2)a sawtooth operating pattern, 3)long duration, 4)relatively slow operating speeds [1]. These features make the vehicles be ideal tools for sampling vertical data of the interior ocean. They have huge amount of potential in oceanographic sensing missions [2], military reconnaissance operations, and marine exploration applications [3].

Due to above merits, main oceanographic research institutions have put their efforts and resources to accelerate the development of this cutting-edging technology. Advent of material and microelectronic technical revolutions, the body of the vehicle are able to endure extreme conditions introduced by the uncertain ocean environment, the electronics are supposed to be reliable and able to drive the buoyancy pump, change the position of the battery, and collect data from onboard sensors. Some of them have been put into commercial productions [4] [5] [6]. However, correspondent researchers lack a high quality simulator for helping software developments.

In general robotics field, a high quality numerical simulator, which estimates robotic motions, interacts with the virtual environment and communicates with sensors in a realistic fashion, enables researchers and developers to develop the guidance, the navigation and the control software without the need for the advanced hardware to test their innovations [7]. This is expected to lower the cost, making it faster and easier to test new software and making it possible to study how robots interact with the environment without the risk of breaking equipment. Compared with ground robotics and drones, the scenarios for underwater glider operation have three Rs [8]: remote, risky, and recalcitrant. The gliders are supposed to be evaluated in deep ocean, receiving commands remotely via satellite signals. These field trials usually require oceanographic support vessels and large logistical requirements for the deployment and the recovery. That makes the physical testing particularly costly. Furthermore, for gliders are designed for long-endurance data collecting missions, they are expected to travel along the sawtooth pattern in ocean and maintain feeding back measurement data for weeks or months, even years. Such long term physical tests are conducted to guarantee their proper functioning. However, these operations involve significant risks because undiscovered software bugs might cause the losing of the robotic platform. Testing gliders in various environmental conditions is another challenge. The interior ocean environments are recalcitrant in the sense that the software in gliders must be designed for a wide range of oceanographic conditions, but it is impossible for operators to regulate environmental conditions where they are

Zhizun Xu is with School of Engineering, Newcastle University, Newcastle upon Tyne UK, and with School of Naval Architecture and Maritime, Guangdong Ocean University, China, Zhizun.Xu@newcastle.ac.uk

Yang Song is with School of Engineering, Newcastle University, Newcastle upon Tyne UK

Jiabao Zhu is with School of Naval Architecture and Maritime, Guangdong Ocean University, China

Weichao Shi(Corresponding author) is with School of Engineering, Newcastle University, Newcastle upon Tyne UK, Weichao.Shi@newcastle.ac.uk

launched. Hence, as for autonomous buoyancy-driven underwater gliders, simulator is especially valuable as a complement to perfect the software prior to field trials.

Unfortunately, existing underwater vehicle simulators are not designed to deal with the unique challenges coming with distinctive features of autonomous buoyancy-driven underwater gliders, including the special propulsion and steering mechanism, and complex hydrodynamic and hydrostatic effects. General physics engines intend to tackle of rigid-body dynamic simulations, while hydrodynamic and hydrostatic forces and moments must also be taken into account. The computation of the interaction of a fluid with submerged body is still infeasible for an off-the-shelf computer for complex geometries like underwater vehicles in real-time [9]. Conventional underwater robotics simulators [10] [9] [8] [7] mainly rely on the Fossen's model [11] to estimate the robotic motion impacted by the fluid, which has been proved successfully in ordinary underwater robotics. However, due to their unique features, the Fossen's model is hard to describe the motion of autonomous buoyancy-driven underwater gliders.

Attempting to bridge the gap and offer a high-quality tool in the software development, a novel simulator of autonomous buoyancy-driven underwater gliders, named UGSim, has been proposed. The simulator is built on top of the UUV simulator and the DAVE project, taking advantages of their merits, including the multi-robots collaboration, the stratified ocean currents simulation, the large scale environment generation, and the dynamic bathymetry spawning. The simulator employs the Gazebo [12] as a physic engine, tackling rigid-body dynamics and the collision detection. It also works as a 3D rendering pipeline, visualising subsea environments and offshore engineering scenarios. The simulator utilises the Gazebo model plugin API(Application Programming Interface) to impose hydrodynamic and hydrostatic effects on gliders. As mentioned before, behaviour patterns of buoyancy-driven gliders are huge different with ordinary ROVs and AUVs. That means most controllers and guidance systems for underwater vehicles are unsuitable. In order to reduce software development workload, and avoid reinventing wheels, the paper provides novel control and guidance modules. Integrating the dead-reckoning module offered by Zhang [8], the basic waypoints-tracking function can be achieved.

Hence, the contributions of the paper are:

- A novel motion simulation of buoyancy-driven gliders with environmental interactions, including collisions, and external disturbances(such as current);
- A LQR control strategy of the buoyancy-driven glider in consecutive work cycles;
- A novel recursive guidance system of the buoyancy-driven glider, which is designed for its features: the long-term operation, the low manoeuvrability, and the low accuracy underwater positioning.

II. LITERATURE REVIEW

The underwater simulator has been developed for decades. The UWsim is one of the first and most commonly known underwater simulator [13], it offers the simulation platform meeting needs of the underwater inspection and intervention missions with one or more robots. The simulator utilised OpenSceneGraph(OSG) and osgOcean libraries to render realistic underwater environments. The former one is an open source 3D graphics application programming interface used by application developers in fields such as the visual simulation. The latter one is another open source project that implements the realistic underwater rendering using the OSG. The UWSim uses the above mentioned libraries and adds the further functionality for easily adding underwater robotics to the scene, and sensors simulations. It has interfaces with external control programs through the Robot Operating System(ROS). However, it has been noted that while it is advantageous in visualization and sensors' simulation, it is lacking in dynamic simulation [9]. The kinematic and kinetic module based on Fossen's models are coded in MATLAB. The output variables(position and attitude of vehicles) from MATLAB codes are broadcasted through the ROS network and captured by the UWSim core for updating the visualisation.

In the mobile robotics community, the Gazebo is a very popular simulator, which is designed for general-purpose open-source robotics. The Gazebo was used in a humanoid robotic project called DexROV [14], coupled with a set of plugins. The Gazebo Classic utilises the Open Dynamic Engine(ODE) as its default physics engine with three alternative physics engines: Bullet [15], Simbody [16] and DART [17]. The 3D rendering pipeline in the Gazebo Classic is implemented by the Object-Oriented Graphics Rendering Engine(OGRE), which is a scene-oriented, real-time, open-source, 3D rendering engine. The controversy over the Gazebo is that it lacks native supports for hydrodynamics and its poor rendering quality which does not allow for the convincing simulation of the underwater vision. But the Gazebo prioritizes physics over rendering, offering the higher physical confidence.

Hence, in order to overcome inferior dynamic performances in the UWSim, freefloating-gazebo [18] was presented as an example of a bridge between the Gazebo and the UWSim for simulations of underwater vehicles. The package includes plugins for the Gazebo to allow the generation of hydrodynamic and hydrostatic forces to be applied on underwater vehicles. The estimated pose of the vehicle will be transmitted to the UWSim via the ROS, where the better rendering quality can be achieved.

A similar implementation involves the extension of the Gazebo to the Robot Construction Kit(ROCK), known as the ROCK-Gazebo [19]. In this case, the ROCK visualisation tool was extended with *OpenSceneGraph* [20] for rendering the underwater environment while the physic simulation runs in the Gazebo. At this moment, it does not yet support the simulation of multiple underwater vehicles.

The latest gazebo-based underwater simulator is the DAVE project [8], which is developed on the top of the UUV simulator [10]. Zhang et al. provided a simulation framework for the multiple-types marine craft, including surface ships, ROVs, AUVs,

and gliders. The highlights of the simulator includes: 1) the definition of stratified ocean currents by directions and depths either constant or periodic in time; 2) the dynamic bathymetry spawning, which means that grid tiles produced from the large, high-resolution bathymetry height map data are dynamically spawned and unloaded. It will increase the efficiency of the memory usage and contribute to accelerate the large scale world simulation.

Prior to the DAVE project, the UUV simulator [10] is well-known for offering a set of newly implemented plugins that model underwater hydrostatic and hydrodynamic effects, thrusters, sensors, and external disturbances. It was designed originally to meet the need of the EU-Funded project SwarMs, and to support the development of new missions strategies and high-level algorithms for cooperative behaviours of underwater vehicles.

Additionally, the WHOI Deep Submergence Lab [21] released common tools and a multi-sensors package, named *ds_sim*. The package composes the Gazebo binary plugins for a number of useful sensors and some utilities(such as DVL, depth sensor, and USBL SMS device, and GPS). In this case, other Gazebo-based underwater vehicle simulators are able to make use of the multi-sensors plugin directly.

Because the Gazebo does not allow for a camera image with a degraded visibility which frequently happens in the subsea and results from the back refraction, Suzuki and Kawabta [22] reported two plugins named as *FluidDynamicsPlugin* and *ImageNoisePlugin* on a generic platform named Choreonoid [23], simulating the fluid dynamics, the buoyancy, the fluid resistance and the visual distortion caused by the camera lens.

Boosting the 3D rendering quality, there are graphics- and game-driven simulators which trade off the physical fidelity with the advanced rendering. It is a necessary balance for all simulators with finite computing resource. Sonefish [9] directly uses the OpenGL to provide the lightweight, high performance rendering pipeline. Its unique feature is being able to simulate complete dynamics or hydrodynamics of underwater vehicle-manipulator systems with contact and force sensing. Its dynamic simulation framework is a set of classes that wrap around the physics library(Bullet Physics) adding an abstraction layer which greatly simplifies creation of simulation scenarios and adds features related to robotics, especially marine robotics, e.g., the buoyancy and hydrodynamics.

Game engines known for the photo-realistic rendering have also been chosen as tools for the visualization of the underwater simulation. The Unreal Engine [24] has been used in vision-based applications. The Unity3D [25] has been used in simulators such the URSim [13]. Hydrodynamic forces achieved by using C# scripts that make use of Unity's APIs to emulate forces on the object's mesh to account for an external physical influence on the vehicle, but limited sensors supports. The MORSE [26] rendering with the Blender Game Engine, a generic simulator for academic robotics. The UW MORSE [7], a underwater vehicle simulator, is developed based on the MORSE, translating damping forces, the buoyancy, and thruster forces into vectors relative to world frame and apply them to the rigid body of the vehicle in the Bullet physic engine. It has multiple sensors support, with interfaces for the MOOS—IvP and the ROS.

Apart from visual tools, there is also the Marine System Simulator [27] which is a MATLAB/Simulink package. However, lacking of the visualisation, and no direct interfaces with the ROS make it unpopular in the community.

However, existing robotic simulators cannot estimate the motion of buoyancy-driven gliders. The paper proposes a simulator specified for the class of the vehicles with a control strategy and a guidance system. As the 3D rendering is not emphasised in glider simulations, the simulator is developed based on the DAVE project, the UUV simulator, and *ds_sim*, by adding the control module, the guidance module, and the kinetic module.

III. SIMULATOR FRAMEWORK

The Gazebo is considered as an accurate framework for simulation and visualization for robotics mechanisms, can be extended to new dynamics, sensors and world models through modular plugins [12]. It provides developers APIs(Application Programming Interfaces) for user-define plugins of models, sensors, and visuals. These plugins have access to the simulation objects and data, can transmit information via topics by using Protocol Buffer3 messages and apply torques and forces to objects in the scenario [10]. In order to make use of customised plugins, the robot description files in SDF format [12](an XML format designed for Gazebo) should declare the involved plugins. In order to integrate with ROS, the robot descriptions are represented in URDF or xacro [28].

The UUV simulator and the DAVE are open-source and Gazebo-based underwater vehicles simulators. They have implemented the fundamental functions, but lack the ability to estimate the manoeuvrability of the buoyancy-driven underwater glider. The proposed simulator is constructed at the top of the UUV simulator and the DAVE by adding a novel kinetic module for the special class of the vehicle. In order to be compatible with the collision detection and rigid body dynamics by the Gazebo build-in physics engine, the accelerations including rotation and translation, computed from the kinetic module, are fed into model status in the Gazebo. That means the equivalent force and torque, caused by hydrodynamics and hydrostatics effects, act on the rigid body. These are counted as initial conditions for rigid dynamic estimations by the build-in physic engine to update the status of the model.

Since manoeuvre mechanics are significantly different with common ROVs, AUVs, and surface ships. The proposed simulator also provides the basic control module and the guidance module to accelerate the software development. These two modules are written by Python, and wrapped as ROS nodes, which are capable of interaction with the Gazebo server. As for the kinetic

module, it is a Gazebo model plugin written in C++, using ROS libraries which allow the plugin to publish model states, and subscribe updates from the ROS node of the control module. In a word, the kinetic module can be regarded as a bridge between the Gazebo server and the ROS master, by using APIs of the Gazebo model and ROS libraries.

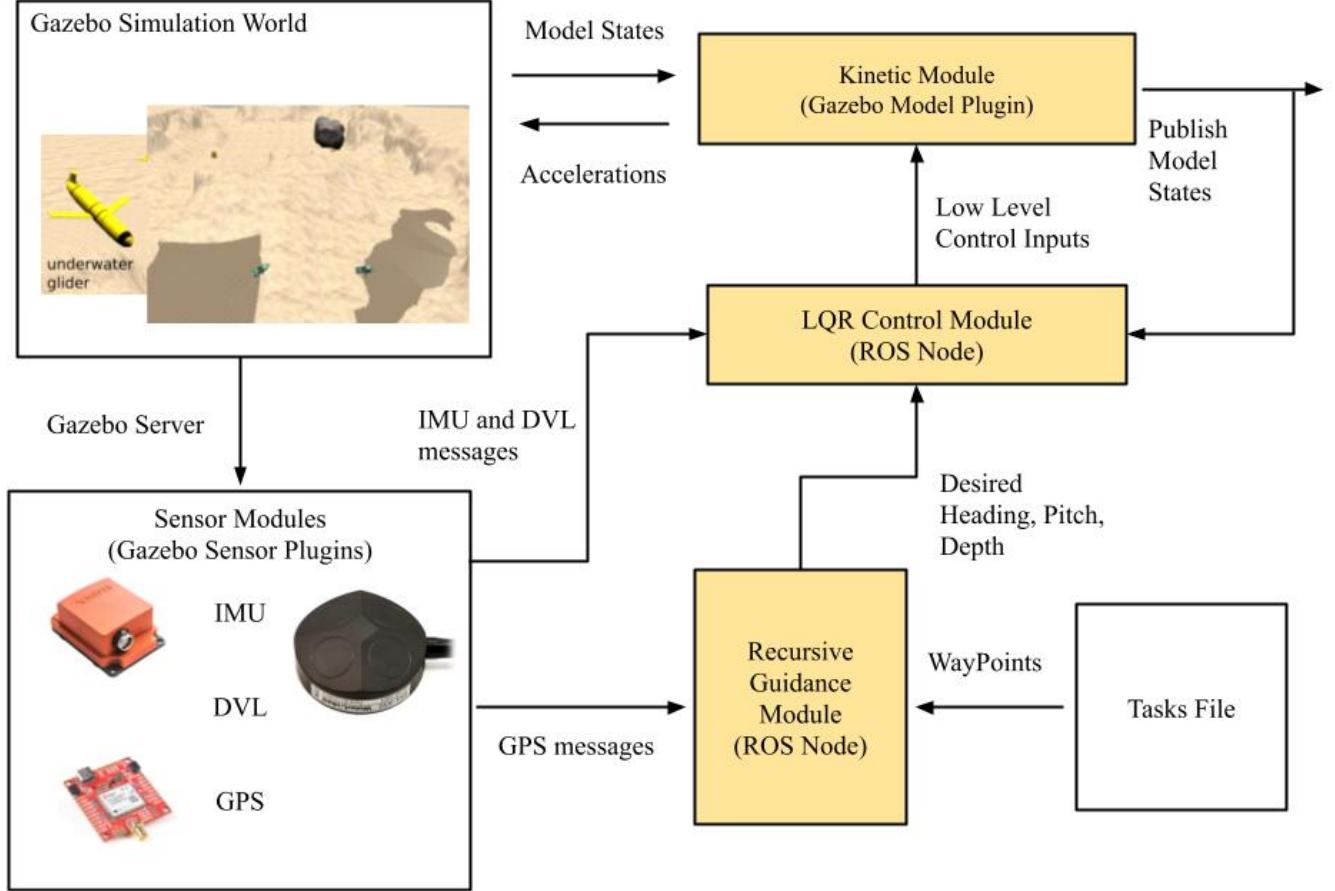


Fig. 1. Simulator Framework

In Fig. 1, the yellow coloured blocks indicate the modules proposed in the paper. The environment models and sensors plugins are provided by the DAVE project. The recursive guidance module is supposed to read waypoints, desired velocities and the target depth from a task file, and produce the desired heading angle and the desired pitch angle, which will be fed to the LQR control module. Note that in sake of the convenience of tests and evaluations, the control module can also receive information directly from the kinetic module. Once receiving sensor messages from sensor plugins, the controller will calculate the LQR control gains, then employ the feedback control law to send low-level control inputs to the kinetic module.

IV. KINETIC MODULE

The aim of the kinetic module is to bring hydrodynamic and hydrostatic effects into the rigid body dynamic simulation processed by the Gazebo build-in physics engine. In order to accomplish the goal, the kinetic equations of buoyancy-driven gliders are supposed to be identified firstly. Subsequently, since the kinetic equations are derived in the NED(North-East-Down) frame, rather than the ENU(East-North-Up) frame which is applied in Gazebo, the coordinate transformation matrix is required. Finally, the accelerations of linear and angular velocities computed through solving kinetic equations, are fed into model(the glider) status via Gazebo APIs before each rigid-body dynamic simulation iteration. The framework of the kinetic module is shown in Fig.2. The variables in kinetic equations are categorised into five groups: constant variables, preset variables, control variables, states of the glider, and iterative variables.

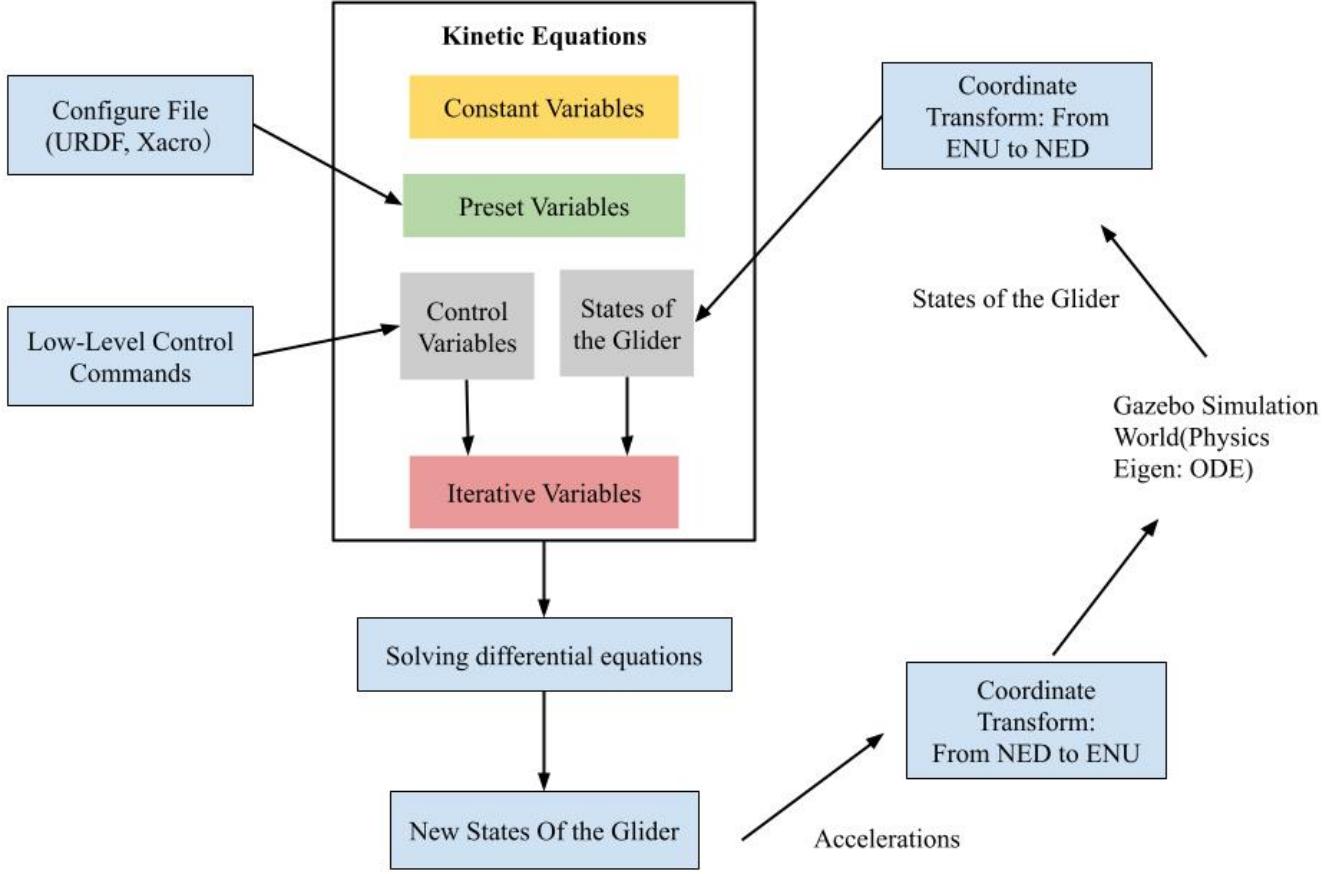


Fig. 2. Kinetic Module

A. Kinetic Equations

The kinetic equations were derived in Leonard and Graver's work [29], using Newton law. While the kinetic model is intended to design the SLOCUM glider control algorithm, it is valid and is true by inspections [30]. These kinetic equations are represented below.

$$\begin{bmatrix} m_s \mathbf{I} + m_p \mathbf{I} & -m_s \hat{\mathbf{r}}_s - m_p \hat{\mathbf{r}}_p \\ m_s \hat{\mathbf{r}}_s + m_p \hat{\mathbf{r}}_p & (\mathbf{J}_s + \mathbf{J}_p) - m_p \hat{\mathbf{r}}_p \hat{\mathbf{r}}_p \end{bmatrix} \begin{bmatrix} \dot{\mathbf{v}} \\ \dot{\boldsymbol{\omega}} \end{bmatrix} = m_s \begin{bmatrix} \mathbf{v}_s \times \boldsymbol{\omega} \\ \mathbf{r}_s \times \mathbf{v} \times \boldsymbol{\omega} + \boldsymbol{\omega} \times \mathbf{r}_s \times \mathbf{v} \end{bmatrix} + \begin{bmatrix} \mathbf{0} \\ (\mathbf{J}_s \boldsymbol{\omega}) \times \boldsymbol{\omega} \end{bmatrix} + m_p \begin{bmatrix} \mathbf{v}_p \times \boldsymbol{\omega} + \dot{\mathbf{r}}_p \times \boldsymbol{\omega} \\ \mathbf{r}_p \times (\mathbf{v}_p \times \boldsymbol{\omega} + \dot{\mathbf{r}}_p \times \boldsymbol{\omega}) \end{bmatrix} - \begin{bmatrix} m_p \ddot{\mathbf{r}}_p \\ \mathbf{J}_p \ddot{\mathbf{c}}_1 + m_p \mathbf{r}_p \times \ddot{\mathbf{r}}_p \end{bmatrix} + \begin{bmatrix} \mathbf{0} \\ (\mathbf{J}_p \boldsymbol{\omega}_p) \times \boldsymbol{\omega}_p \end{bmatrix} + \begin{bmatrix} \mathbf{0} \\ \mathbf{J}_p (\dot{\mathbf{c}}_1 \times \boldsymbol{\omega}) \end{bmatrix} + \begin{bmatrix} \mathbf{F}_{ext} \\ \mathbf{T}_{ext} \end{bmatrix} \quad (1)$$

The definitions of variables are described in Table I. Mass definitions for buoyancy-driven gliders are shown in Fig.3

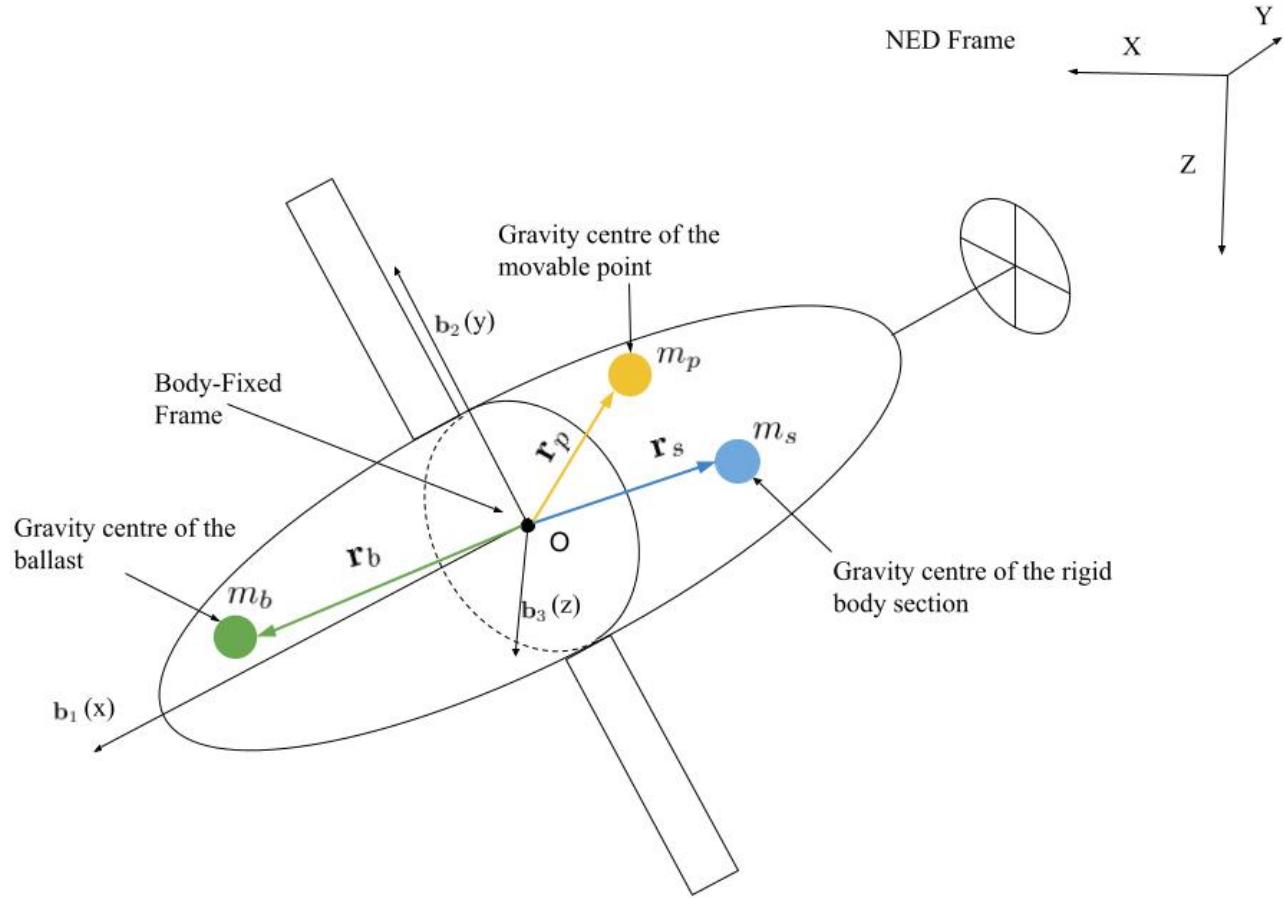


Fig. 3. Mass Definitions for Buoyancy-Driven Gliders

TABLE I
DEFINITION OF VARIABLES

| Name | Description |
|-------------------------|--|
| \mathbf{v} | linear velocity of the body-fixed frame with respect to the North-East-Down(NED) frame observed in the body-fixed frame |
| $\boldsymbol{\omega}$ | rotation velocity of the body-fixed frame with respect to the NED frame observed in the body-fixed frame |
| m_s | mass of the rigid body section(excluding the movable point) |
| m_p | mass of the movable point(the controllable moving part inside the glider body. It usually indicates the battery package.) |
| m_b | mass of the ballast(It is equivalent to the buoyancy resulted from the variance of volume.) |
| \mathbf{r}_b | position of the gravity centre of the ballast($\mathbf{r}_b = [r_{b1}, 0, 0]^T$) |
| \mathbf{b} | unite basis vectors of body-fixed frame($\mathbf{b}_1 = [1, 0, 0]^T$, $\mathbf{b}_2 = [0, 1, 0]^T$, $\mathbf{b}_3 = [0, 0, 1]^T$) |
| ζ | rotation angle of the movable point about the x-axis of the body-fix frame |
| \mathbf{r}_s | position of the gravity centre of the rigid body in body-fixed frame |
| \mathbf{v}_s | linear velocity of the rigid body section in the body-fixed frame relative to the North-East frame observed in the body-fixed frame ($\mathbf{v}_s = \mathbf{v} + \boldsymbol{\omega} \times \mathbf{r}_s$). |
| $\boldsymbol{\omega}_s$ | angular velocity of the rigid body section in the body-fixed frame relative to the North-East frame observed in the body-fixed frame($\boldsymbol{\omega}_s = \boldsymbol{\omega}$) |
| \mathbf{r}_p | position of the movable point in the body-fixed frame($\mathbf{r}_p = [r_p, -R_p \sin \zeta, R_p \cos \zeta]$, where R_p is deviation distance between the gravity centre of the movable point and x-axis of the body-fix frame.) |
| \mathbf{v}_p | linear velocity of the movable point with respect to the NED frame observed in the body-fixed frame($\mathbf{v}_p = \mathbf{v} + \boldsymbol{\omega} \times \mathbf{r}_p + \dot{\mathbf{r}}_p$) |
| $\boldsymbol{\omega}_p$ | angular Velocity of the movable point with respect to the NED frame in the body-fixed frame($\boldsymbol{\omega}_p = \boldsymbol{\omega} + \dot{\zeta} \mathbf{b}_1$) |
| \mathbf{J}_s | inertial matrix of the rigid body |
| \mathbf{J}_p | inertial matrix of the movable point with respect to the body-fixed frame($\mathbf{J}_p = {}^b \mathbf{R}_p \mathbf{J}_p^0$, where ${}^b \mathbf{R}_p$ and \mathbf{J}_p^0 is an initial inertial matrix, at this moment, $\zeta = 0$) |

$${}^b \mathbf{R}_p = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos \zeta & -\sin \zeta \\ 0 & \sin \zeta & \cos \zeta \end{bmatrix} \quad (2)$$

1) *Extension of External Force and Moments Terms:* The last term in right side of (1) can be expanded as,

$$\begin{bmatrix} \mathbf{F}_{ext} \\ \mathbf{T}_{ext} \end{bmatrix} = \begin{bmatrix} \Delta m(z)g^B \mathbf{R}_I \mathbf{i}_3 \\ (m_s \mathbf{r}_s + m_p \mathbf{r}_p + m_b \mathbf{r}_b)g \times {}^B \mathbf{R}_I \mathbf{i}_3 \end{bmatrix} + \begin{bmatrix} \mathbf{F}_h \\ \mathbf{T}_h \end{bmatrix} + \begin{bmatrix} \mathbf{F}_m \\ \mathbf{T}_m \end{bmatrix}. \quad (3)$$

In the above equation, $g = 9.81$ is gravitational constant, ${}^I \mathbf{R}_B$ is a transformation matrix mapping vectors in the body-fixed frame into the inertial frame. The symbol \mathbf{i} is the standard basis, which is $\mathbf{i}_1 = [1, 0, 0]^T$, $\mathbf{i}_2 = [0, 1, 0]^T$, $\mathbf{i}_3 = [0, 0, 1]^T$. For the Earth's rotation is ignored, the NED frame and the inertial frame can be considered to be coincided. That means ${}^N \mathbf{R}_B = {}^I \mathbf{R}_B$, where ${}^N \mathbf{R}_B$ is a rotation matrix presenting the pose of the body-fixed frame in the inertial frame, and also transforms the orientation of a vector with respect to the body-fixed frame into the NED frame.

The first term on right side of (3) is the buoyancy force and the buoyancy moment, which are,

$$\Delta m(z)g = (m_b + m_s + m_p - \rho(z)\left(\frac{m_s + m_p}{p_5} - K_{vh}z\right))g, \quad (4)$$

where z is the diving depth of the glider, $\rho(z) \approx \text{density of deepseawater}$, p_5 is the density of sea water. K_{vh} is a coefficient of the volume variance according to different diving depths. For the sake of simplicity, the moment can be divided into three components: the rigid body section, the movable point, and the ballast, which are,

$$\begin{aligned} T_s &= m_s g \mathbf{r}_s \times ({}^B \mathbf{R}_N \mathbf{i}_3) \\ T_p &= m_p g \mathbf{r}_p \times ({}^B \mathbf{R}_N \mathbf{i}_3) \\ T_b &= m_b g \mathbf{r}_b \times ({}^B \mathbf{R}_N \mathbf{i}_3). \end{aligned}$$

The second term on the right side of (3) is referred to as the viscous hydrodynamic force and moment. It can be extended as,

$$\mathbf{F}_n = {}^B \mathbf{R}_v \begin{bmatrix} -D \\ SF \\ -L \end{bmatrix} \quad (5)$$

$$\mathbf{T}_h = {}^B \mathbf{R}_v \begin{bmatrix} T_{DL1} \\ T_{DL2} \\ T_{DL3} \end{bmatrix} \quad (6)$$

where D , SF , and L are the resistance, the side force and the lift force. The T_{DL} is the hydrodynamic moment. The ${}^B \mathbf{R}_v$ is a transformation matrix mapping vectors from the velocity frame to the glider body-fixed frame. These are derived from hydrodynamic and hydrostatic coefficients, such as K_{D0} , K_β , K_{L0} , K_L , etc, through below equations,

$$D = (K_{D0} + K_D \beta^2) V_r^2 \quad (7)$$

$$SF = (K_\beta \beta) V_r^2 \quad (8)$$

$$L = (K_{L0} + K_L \beta) V_r^2 \quad (9)$$

$$T_{DL1} = (K_{MR} \beta + K_p p_r) V_r^2 \quad (10)$$

$$T_{DL2} = (K_{M0} + K_M \beta K_q q_r) V_r^2 \quad (11)$$

$$T_{DL3} = (K_{MY} \beta + K_r r_r) V_r^2 \quad (12)$$

where p_r , q_r , r_r are elements in the angular velocity related to the fluid, hence the velocity vector is $\boldsymbol{\omega}_r = [p_r \ q_r \ r_r]^T$. The V_r is the norm of linear velocity $\mathbf{v}_r = [\mu_r, \nu_r, \omega_r]$ related to the fluid, and the ${}^B \mathbf{R}_v$ is a transformation matrix mapping vectors from the velocity frame to the glider body-fixed frame, which are presented as,

$$V_r = \sqrt{\mu_r^2 + \nu_r^2 + \omega_r^2} \quad (13)$$

$$\alpha = \tan^{-1} \frac{\omega_r}{\mu_r} \quad (14)$$

$$\beta = \sin^{-1} \frac{\nu_r}{V_r} \quad (15)$$

$${}^B \mathbf{R}_v = \begin{bmatrix} \cos \alpha \cos \beta & -\cos \alpha \sin \beta & -\sin \alpha \\ \sin \beta & \cos \beta & 0 \\ \sin \alpha \cos \beta & -\sin \alpha \sin \beta & \cos \alpha \end{bmatrix} \quad (16)$$

The relative velocities $\mathbf{v}_r, \boldsymbol{\omega}_r$ are calculated from absolute velocities of the glider($\mathbf{v}, \boldsymbol{\omega}$) and velocities of fluid currents ($\mathbf{v}_f, \boldsymbol{\omega}_f$).

$$\mathbf{v}_r = \mathbf{v} - {}^B \mathbf{R}_I \mathbf{v}_f \quad (17)$$

$$\boldsymbol{\omega}_r = \boldsymbol{\omega} - {}^B \mathbf{R}_I \boldsymbol{\omega}_f \quad (18)$$

The third term on the right side of (3) is referred to as the inertial hydrodynamic force and moment, which is

$$\begin{bmatrix} \mathbf{F}_m \\ \mathbf{T}_m \end{bmatrix} = -\mathbf{M}_f \begin{bmatrix} \dot{\mathbf{v}}_r \\ \dot{\boldsymbol{\omega}}_r \end{bmatrix} \quad (19)$$

$$\mathbf{M}_f = \begin{bmatrix} \mathbf{M}_A & \mathbf{C}_A \\ \mathbf{C}_A^T & \mathbf{J}_A \end{bmatrix} = \begin{bmatrix} \lambda_{11} & 0 & 0 & 0 & 0 & 0 \\ 0 & \lambda_{22} & 0 & 0 & 0 & \lambda_{26} \\ 0 & 0 & \lambda_{33} & 0 & \lambda_{35} & 0 \\ 0 & 0 & 0 & \lambda_{44} & 0 & 0 \\ 0 & 0 & \lambda_{53} & 0 & \lambda_{55} & 0 \\ 0 & \lambda_{62} & 0 & 0 & 0 & \lambda_{66} \end{bmatrix} \quad (20)$$

where, \mathbf{M}_A , \mathbf{J}_A and \mathbf{C}_A are the added mass matrix, the added inertial matrix, and cross terms λ_{ii} are inertial hydrodynamic coefficients.

2) *Variables Classification:* Since (1) involves dozens variables, these variables are categorized into different groups: constant variables, preset variables, states of the glider, control variables, iterative variables. Constant variables mean the variables in (1) are supposed to remain same all time. The constant values involve the standard basis and the gravity coefficient, i.e. \mathbf{b} , $\mathbf{b}_1 = [1, 0, 0]^T$, $\mathbf{b}_2 = [0, 1, 0]^T$, $\mathbf{b}_3 = [0, 0, 1]^T$, \mathbf{i} , $\mathbf{i}_1 = [1, 0, 0]^T$, $\mathbf{i}_2 = [0, 1, 0]^T$, $\mathbf{i}_3 = [0, 0, 1]^T$, and $g = 9.81$.

Preset variables are assigned during the simulation bootstrap and would remain same during the whole simulation. These variables and their descriptions are presented in Table II. Generally, they are loaded through the URDF or Xacro robot description file.

TABLE II
PRESET VALUES

| Name | Description |
|--|--|
| m_s, \mathbf{r}_s | mass and position vector of the gravity centre of rigid body section |
| m_p | mass of the movable point(battery package) |
| m_b, \mathbf{r}_b | mass and position vector of the gravity centre of the ballast |
| \mathbf{J}_0^p | initial inertial matrix for the movable point(battery package) |
| \mathbf{J}_S | initial matrix of the rigid-body section |
| R_p | deviation distance between the gravity centre of the movable point(battery package) and the axis-x of the body-fixed frame |
| p_5, K_{vh}, ρ | deep density, surface density, and deformation coefficients |
| $K_{D0}, K_D, K_\beta, K_{L0}, K_L, K_{MR}, K_p, K_{M0}, K_M, K_q, K_{MY}, K_r, M_f$ | hydrostatic coefficients |
| $\Delta m_b, \Delta \zeta, \Delta r_{p1}$ | incremental values of actuators |
| $\text{Max } r_{p1}, \text{Min } r_{p1}, \text{Max } Mb$ | ranges for control inputs |

States of a glider consist of the velocity, the orientation and position, as shown in Table III. The states of the glider, solved from kinetic equations, are supposed to be with respect to the inertial frame which takes the Earth's rotation into account. Here, we ignore the Earth's rotation and assume that the states are with respect to the NED frame.

TABLE III
STATE VARIABLES

| Name | Description |
|----------|---|
| v | linear velocity of the body-fixed frame with respect to the NED frame observed in the body-fixed frame |
| ω | angular velocity of the body-fixed frame with respect to the NED frame observed in the body-fixed frame |
| q | quaternion presenting the orientation of the body-fixed frame in the NED frame |
| p | position vector of the body-fixed frame the in the NED frame |

Control variables indicate the variables which are directly impacted by low-level control inputs or commands. Commonly, they are the rotation angle of the battery ζ , the translation of the battery r_{p1} , and the ballast mass m_b .

Iterative variables, depending on the states and the control variables, are ones which keep volatile among simulation iterations. In other words, due to variances of states and control variables, some associated variables in (1) would change correspondingly, which are referred to as iterative variables and shown in Table IV.

TABLE IV
ITERATIVE VALUES

| Name | Dependencies |
|-----------------|--|
| v_s, ω_s | depending on state variables v and ω , calculated by $\omega_s = \omega$, $v_s = v + \omega \times r_s$ |
| v_p, ω_p | depending on the state variables (v and ω) and the gravity centre of the movable point(r_p), calculated by $v_p = v + \omega \times r_p + \dot{r}_p$, $\omega_p = \omega + \dot{\zeta}b_1$ |
| v_r, ω_r | depending on environmental current measurement and state values |
| α, β | depending on the relative speed of fluid current ω_r and v_r |
| R_v^B | depending on α and β |
| ${}^B R_N$ | depending on the orientation q |
| J_p | depending on R_p^b |
| r_p | depending on the translation of the battery package r_{p1} and the rotation angle of the battery package ζ |
| R_p^b | depending on the rotation angle of the battery package ζ |

In order to reduce redundant computations in simulations, approximations for some variables are conducted, as shown in Table V. These approximations are mainly for some trivial derivatives and second-order derivatives.

TABLE V
APPROXIMATION

| Name | Description |
|------------------|---|
| $\dot{\omega}_r$ | $\dot{\omega}_r = 0$ |
| \dot{v}_r | $\dot{v}_r = 0$ |
| $\dot{\zeta}$ | $\dot{\zeta} = \Delta\zeta$ or 0 if the battery package rotated to the desired angle |
| $\ddot{\zeta}$ | $\ddot{\zeta} = 0$ |
| \dot{r}_p | $\dot{r}_p = [\Delta r_{p1}, -R_p \cos \zeta \dot{\zeta}, -R_p \sin \zeta \dot{\zeta}]^T$ |
| \ddot{r}_p | 0 |

3) *Control Variables Variation:* Since restrictions of electronic or mechanical structures, the control variables can't reach the given control inputs immediately. In realistic situations, the control variables descend or ascend gradually to approach the control inputs given from an operator or a high-level control algorithm, through relevant electronic or mechanical structures regulating the pose of the battery and the ballast mass. In this case, the control input can be regarded as the desired control values $Du(D\zeta, Dr_{p1}, Dm_b)$, and incremental value(incremental value per unit time) for the control variables are Δu , where u can be substituted with ζ , r_{p1} , and m_b . To mimic the work pattern of electronic or mechanical structures, variation of control variables will obey Algorithm 1.

Algorithm 1 Control Variable Variation

D_u Desired Control value

u Control variable

Δu Incremental value per unit time for the control variables

Δt Endurance of one simulation iteration

if $u_d - u > \Delta u \Delta t$ **then**

$u = u + \Delta u \Delta t$

else if $u_d - u < \Delta u \Delta t$ **then**

$u = u - \Delta u \Delta t$

else

$u = u_d$

end if

B. Coordinates Transformation

The kinetic equations (1) are derived in the NED frame. Hence, the states(poase, velocities, and accelerations) of a glider solved from these differential equations are presented in the NED frame as well. However, the states of the glider model in the Gazebo simulation are described in the ENU frame [31]. That means the state variables are supposed to be transformed into the ENU frame before updating the accelerations of a glider in the Gazebo world via model plugin APIs. These two frames are referred to as world frames or global frames, which are depicted in Fig.4.

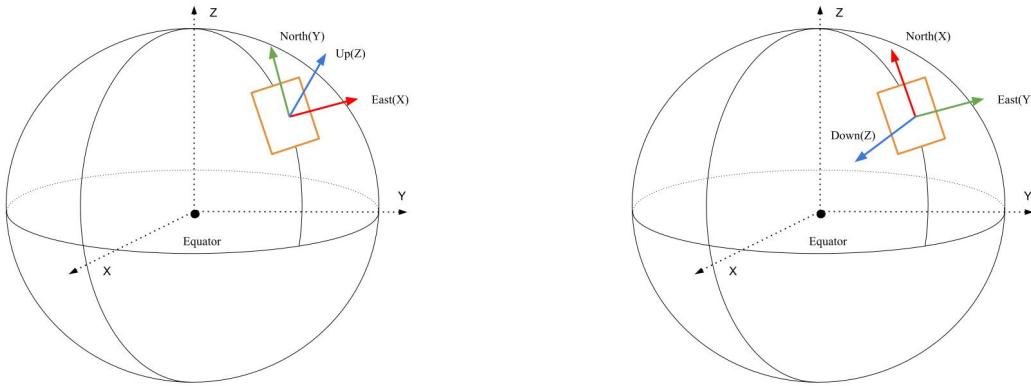


Fig. 4. Comparison of ENU and NED Frames: Left figure presents an ENU Frame; Right figure presents a NED frame.

Furthermore, local frames or body-fixed frames of the Gazebo model(visual model) and the kinetic model are different as well. Because the local frame usually is parallel to its world frame. The local frame of the Gazebo model(visual model) comply the ENU convention, while that of kinetic model obeys the NED convention. Similar with world frames, they share the same original point. The relations among global frames and local frames are illustrated in Fig.5.

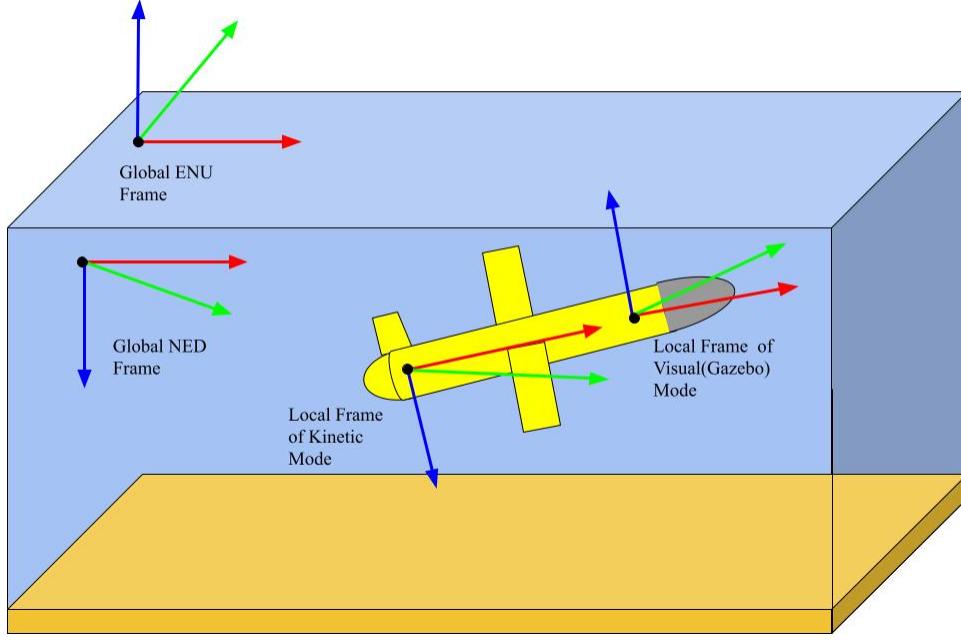


Fig. 5. Frames in the simulator. Note that the two global frames and two local frames share the same original points, which means the distances of between the two local frames and the two global frames don't exist actually. We separate these frames so that they can be recognized visually.

Transformations from the NED frame to the ENU frame are categorised into following conditions. In this part, the symbol k indicates the kinetic model's local frame, and v represents the Gazebo(visual) model's local frame. The NED frame is abbreviated by N , and E means the ENU frame.

- Orientation Transformation(${}^N\mathbf{R}_k \rightarrow {}^E\mathbf{R}_v$)

Given the orientation of the kinetic model in the NED frame ${}^N\mathbf{R}_k$, obtain the orientation of Gazebo model(visual model) in the ENU frame ${}^E\mathbf{R}_v$:

$${}^E\mathbf{R}_v = {}^E\mathbf{R}_N {}^N\mathbf{R}_k {}^k\mathbf{R}_v \quad (21)$$

- Position Transformation($\mathbf{p}_k^N \rightarrow \mathbf{p}_k^E$)

Given a vector of position of the kinetic model in NED frame \mathbf{p}_k^N , obtain the position vector of the Gazebo model(visual model) in the ENU frame \mathbf{p}_k^E :

$$\mathbf{p}_k^E = {}^E\mathbf{R}_N \mathbf{p}_k^N \quad (22)$$

We can conclude $|\mathbf{p}|_k^N = |\mathbf{p}|_v^E$, since two local coordinates' origins are coincided.

- Linear Velocity, Acceleration and Rotation Rates Transformation($\mathbf{v}_{Nk}^k \rightarrow \mathbf{v}_{Ev}^E$)

Given linear or rotation velocities of kinetic model \mathbf{v}_{Nk}^k in NED frame, obtain linear or rotation velocities \mathbf{v}_{Ev}^E of the Gazebo model(visual model) in the ENU frame:

$$\mathbf{v}_{Ev}^E = {}^E\mathbf{R}_k (\mathbf{v}_{EN}^k + \mathbf{v}_{Nk}^k + \mathbf{v}_{kv}^k) \quad (23)$$

Since the world frame and the NED frame are static, and local frames k and v have zeros relative linear velocities, $\mathbf{v}_{kv}^k = 0$ and $\mathbf{v}_{EN}^k = 0$. Therefore, $\mathbf{v}_{Ev}^E = {}^E\mathbf{R}_N {}^N\mathbf{R}_k (\mathbf{v}_{Nk}^k)$. Similar deductions can be made for accelerations and rotation rates ω (the \mathbf{v}_{kv}^k can further be decomposed into addition of velocities in inertial frame).

In the implementation, ${}^w\mathbf{R}_N$ and ${}^k\mathbf{R}_v$ are constants and can be calculated through the Euler angles,

$${}^w\mathbf{R}_N = \mathbf{R}_Z\left(\frac{\pi}{2}\right)\mathbf{R}_Y(\pi)\mathbf{R}_X(0) \quad (24)$$

$${}^k\mathbf{R}_v = \mathbf{R}_Z(0)\mathbf{R}_Y(0)\mathbf{R}_X(\pi). \quad (25)$$

The rotation matrix ${}^N\mathbf{R}_k$ is obtained by solving kinematics equations. The reverse transformations can be realised through the matrix reverse operation.

V. LQR CONTROL MODULE

The control of buoyancy-driven glider is implemented by the LQR method. The controller receiving the desired gliding velocity, pitch and heading angles from the guidance system or an operator, and then generates control commands for low-level actuators to tune the craft to reach these desired states.

There are various control strategies for buoyancy-driven gliders have been reported in recent years [32] [33]. Among the various controllers, PID (proportional–integral–derivative controller) and LQR(Linear Quadratic Regulator) are two most commonly implemented methods. Compared to the PID controller, the LQR is a standard linear optimal control design method which produces a stabilizing control law that minimizes a cost function that is a weighted sum of the squares of the states and input variables [29]. It is capable of finding optimal control inputs that minimise overall system performance criteria [30]. In many scenarios, researchers have presented that the LQR outperforms the PID controller in glider systems [34] [29].

A. LQR Control Strategy

When the glider floats up, the guidance system received the signal from GNSS(Global Navigation Satellite System), correct its location. Then it would send instructions to the LQR Controller. Once LQR gains are solved, the controller would send low-level commands to the actuators. Note that LQR controllers are decoupled into the vertical plane and the horizontal plane. The state-space representations linearized from (1) at equilibrium points are different in ascending and descending processes. Hence, there are totally four different LQR gains are supposed to be obtained, prior to start a work circle. They individually match four different circumstances: Ascending in the horizontal plane; Descending in the horizontal plane; Ascending in the vertical plane; Descending in the horizontal plane.

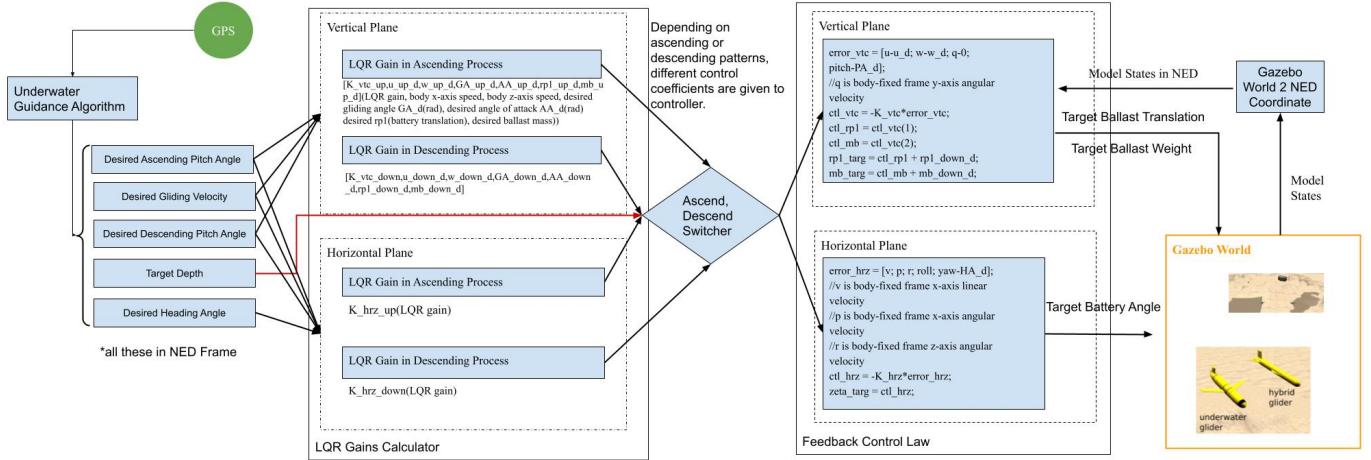


Fig. 6. LQR Control Strategy

The control strategy is shown in Fig.6. The guidance system provides the desired pitch angles in ascending and descending processes, the desired heading angle, and the desired velocity for the LQR controller. Before the glider executes next work circle shown in Fig.8, the four LQR control gains are computed. Once computations complete, the feedback control laws in horizontal and vertical planes are implemented. States of the glider, including orientations and velocities are transferred into the NED frame, are fed into the control laws. The outputs of the controller are commands of the battery linear translate, the ballast mass, and the battery angle, provided for the low-level actuators. The glider's orientations and velocities can be obtained through a DVL sensor or the Gazebo model states publisher. However, sometimes, in order to minimize power consumptions, a DVL might be not equipped. In this case, we can set the velocity error zero in the feedback control laws for detaching the speed control.

B. LQR Controller Derivation

The LQR controller has been reported in [29] [30] [35]. Here, a short summary of LQR controller derivation is presented. The LQR control problem can be defined to minimize the cost function,

$$J = \int_0^{\infty} \Delta x^T Q \Delta x + \Delta u^T R \Delta u \, dt \quad (26)$$

where \mathbf{Q} and \mathbf{R} are state and control penalty matrices. The matrices \mathbf{Q} and \mathbf{R} were chosen to ensure well-behaved dynamics and to prevent large motions in the movable mass position and variable mass that would exceed physical limitations. The

weight selections are given by [29] [30]. Let $\Delta\mathbf{x}$ and $\Delta\mathbf{u}$ be the small disturbances of state variables and control inputs related to the glider equilibrium, which are represented as,

$$\begin{aligned}\Delta\mathbf{x} &= \mathbf{x} - \mathbf{x}_{eq} \\ \Delta\mathbf{u} &= \mathbf{u} - \mathbf{u}_{eq}\end{aligned}$$

where \mathbf{x} and \mathbf{u} are states variables and control inputs respectively. We denote with subscript eq variables at the glider equilibrium point.

In order to solve (26), the kinetic equations (1) should be linearized to the form of,

$$\Delta\dot{\mathbf{x}} = \mathbf{A}\Delta\mathbf{x} + \mathbf{B}\Delta\mathbf{u}, \quad (27)$$

where \mathbf{A} is the state matrix and \mathbf{B} is the input matrix.

The corresponding control law is $\mathbf{u} = -\mathbf{K}\Delta\mathbf{x}$ where \mathbf{K} is computed using the Python Control Systems Library [36] and the SciPy [37] from the Riccati equations formed by \mathbf{A} , \mathbf{B} , \mathbf{Q} , and \mathbf{R} . In this case, control inputs are $\mathbf{u} = [r_{p1}, m_b, \xi]$, r_{p1} the battery package translation, m_b the ballast mass, and ξ the rotation angle of battery package. The state variables are $\mathbf{x} = [u, w, q, \theta]$, where u is the forward velocity, w is the heave velocity, q is the pitch angle, and θ is the yaw angle or the heading angle.

VI. RECURSIVE GUIDANCE MOULE

The aim of the recursive guidance strategy is to suit the long-term operation, the low manoeuvrability, and the unsatisfactory accuracy of underwater localisation technologies. The former two features come from the special structure of this class of vehicles, as mentioned in Section I. The latter feature results from the fact that this kind of gliders usually do not equip with the DVLs(Doppler Velocity Logs) or other acoustic positioning devices in order to minimise the power cost. Because of their sawtooth travel pattern, the GNSS is a common means of relocation as soon as they float up the sea surface.

Based on common realistic scenarios, a guidance strategy is developed based on the waypoint system. The selected waypoints(latitudes and longitudes) and target depths are preloaded via a task file. They are stored in a database and used for generation of a trajectory or a path for the moving craft to follow [11]. With these information, the guidance would calculate the desired heading, the pitch angles when the vehicle is on the sea surface. Every floating up is regarded as an iteration for the proposed guidance algorithm, with updating latest position information via GNSS(or GPS), the desired heading and pitch angles are corrected. In this case, pitch angles are changing, and become steeper and steeper as the glider approaching target positions. Alternatively, the pitch angles during a whole mission can be assigned from the task file. In this case, the pitch angles are independent of the guidance system.

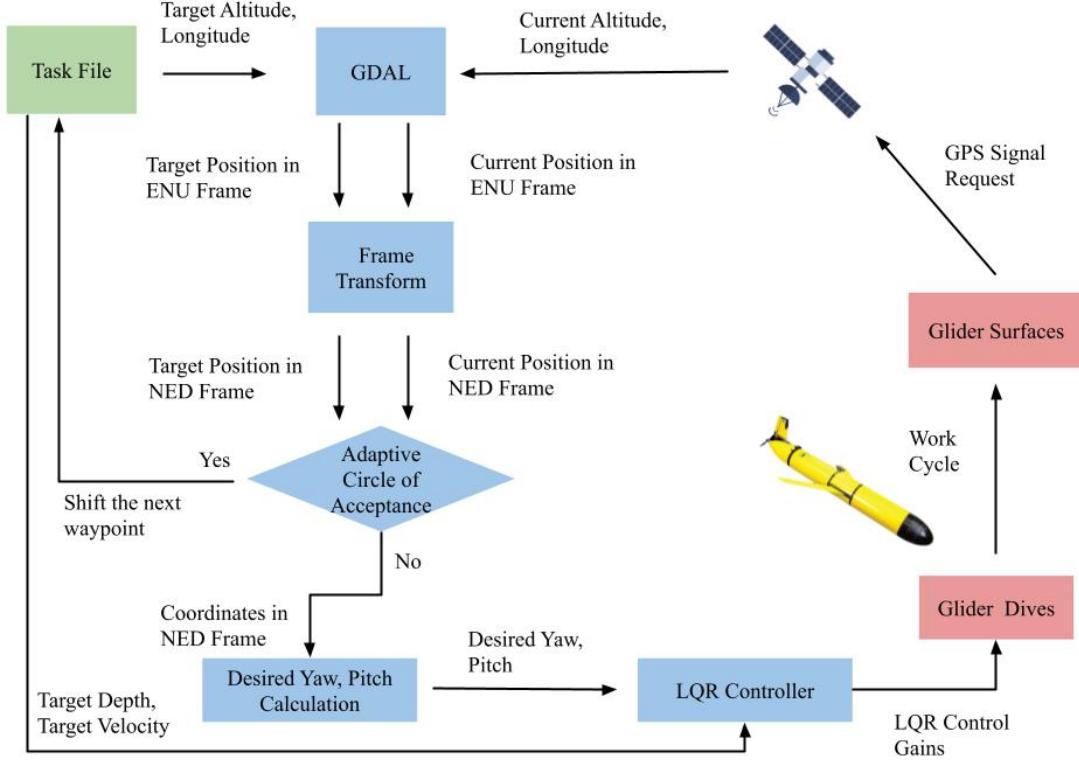


Fig. 7. Recursive Guidance System

Fig.7 presents the whole guidance strategy which can be divided into two subprocesses. The GPS sensor produces the latitude and longitude coordinates, which are supposed to be converted into the NED frame. According to given latitude and longitude from a task file, the desired heading and pitch angles are computed, or pitch angles are pre-assigned.

A. Conversion from Geographic Coordinate System to NED Frame

The simulator fully inherits the sensor plugins from the DAVE project. The GPS module developed in the DAVE project transforms ENU coordinates to the WGS84 geodetic system(latitude & longitude system) [8] [31], rather than directly acquiring the latitude & longitude through the Gazebo server. It is implemented by specifying certain EPSG(European Petroleum Survey Group) codes through the GDAL(Geospatial Data Abstraction Library) Warp API, which provides services for the high performance image warping using the application provided geometric transformation functions [38]. In the simulator, we inverse the transformation by using the GDAL Warp API to obtain the coordinates in the ENU frame. Then, these position vectors are transformed into the NED frame.

B. Desired Heading Angle Identification

The desired heading angle(Euler angles) is determined by Line-Of-Sight(LOS) steering law [11]. The LOS vector is defined by the glider's position and the target(the waypoint). Both of them are in the NED frame.

Firstly, the LOS vector between the current position \mathbf{p}_t and the target position \mathbf{p}_k is defined as,

$$\mathbf{p}_d = \mathbf{p}_t - \mathbf{p}_k \quad (28)$$

where \mathbf{p}_d is a LOS vector between the current position to target position. The \mathbf{e}_d is the normalised vector of \mathbf{p}_d ,

$$\mathbf{e}_d = \frac{\mathbf{p}_d}{\text{norm}(\mathbf{p}_d)}. \quad (29)$$

The cosine of the desired heading angle is the inner product of \mathbf{e}_d and the x-axis of the NED coordinate,

$$\cos(\theta) = \mathbf{e}_d^T \mathbf{e}_x \quad (30)$$

Since $\mathbf{e}_x = [1, 0, 0]^T$,

$$\cos(\theta) = e_d^1 \quad (31)$$

where e_d^1 is the first component of the vector e_d .

The heading angle θ is in the range of $[-\pi, \pi]$, and is computed by $\theta = \cos^{-1}(e_d^1)$. However, the return value of $\cos^{-1}()$ function lies between 0 and π . The sign of θ can be determined by the second component of the distant vector, which is e_d^2 . When $e_d^2 > 0$, the sign is -1 . Oppositely, the sign is 1 , when $e_d^2 < 0$. (If $e_d^2 = 0$, the $e_d^1 = 1$, and θ is zero.)

C. Desired Pitch Angle Identification

Once pitch angles are determined by the guidance system, the magnitude of pitch angles(ϕ) in ascending and descending processes are assumed to be identical. It is determined by the expected travelling distance(L) of one work cycle and the target depth(D), as shown in Fig.8,

$$\phi = \tan^{-1}\left(\frac{L_d}{2D}\right) \quad (32)$$

The travelling distance(L_d) is the quotient of the distance(L_t) from the current position to the target position divided by the minimum loops number, i.e. $L_d = \frac{L_t}{\text{minimum loops number}}$. As the glider tends to approach the target, the pitch angle will become steeper and steeper, and finally hits the boundary of the range of pitch angle. By this way, a glider is expected to get the target as closely as possible.

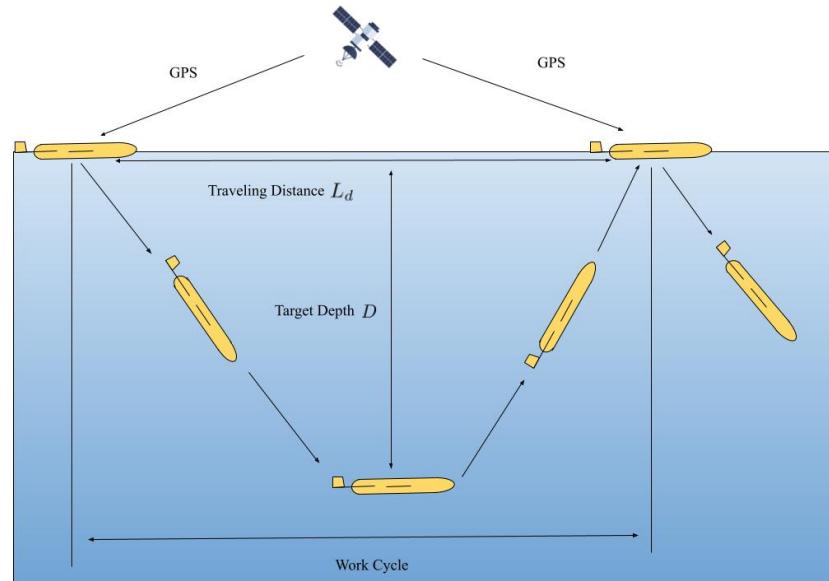


Fig. 8. Pitch Angle Identity

D. Adaptive Circle of Acceptance

When moving along a piece wise linear path made up of n straight-line segments connected by $n+1$ waypoints, a switching mechanism for selecting the next waypoint is needed. This waypoint($p_{k+1} = (x_{k+1}, y_{k+1})$) can be selected on the basis of whether or not the craft lies within a circle of acceptance with the radius R around the current waypoint($p_k = (x_k, y_k)$). Moreover, if craft positions p_t at time t satisfy,

$$|p_t - p_k|^2 \leq R^2 \quad (33)$$

or,

$$|p_d| \leq R \quad (34)$$

the next waypoint (p_{k+1}) should be selected.

For normal surface ships and submarines, the circle of acceptance R is constant. However, the buoyancy-driven gliders cannot be located precisely when it is submerged. Hence, it might pass by the waypoint or the target when it is dividing in underwater. An adaptive circle of acceptance for buoyancy-driven gliders is defined as,

$$R_a = \sqrt{|p_d|^2 - |p_d \cos \theta|^2} \quad (35)$$

where \mathbf{p}_d is the LOS vector ($\mathbf{p}_d = |\mathbf{p}_t - \mathbf{p}_k|$) and,

$$\cos \theta = \frac{\mathbf{p}_d(\mathbf{p}_t - \mathbf{p}_{t-1})}{|\mathbf{p}_d||\mathbf{p}_t - \mathbf{p}_{t-1}|} \quad (36)$$

If $\cos \theta \geq 0$, the current position \mathbf{p}_t and the previous position \mathbf{p}_{t-1} are on the either side of the waypoint. It implies the vehicle past the waypoint \mathbf{p}_k and the adaptive circle of acceptance should be derived and checked. Oppositely, we will not process further steps and let the vehicle continue moving. when $\cos \theta = 1$, the adaptive circle of acceptance can be simplified to,

$$R_a = L - |\mathbf{p}_d| \quad (37)$$

where L is the horizontal distance of the previous work cycle, which $L = \frac{2D}{\tan \phi_d}$, ϕ_d is the desired pitch angle for the previous work cycle, D is the depth of the previous work cycle. In realistic situations, the simplified adaptive circle of acceptance is preferred, for $\theta \rightarrow 0$ as the glider approaches the waypoint.

When the glider finishes the next work cycle and floats up, the guidance system detects if the adaptive circle of acceptance derived by the current position \mathbf{p}_t satisfies,

$$R_a^2 \leq R^2 \quad (38)$$

that means the glider has past by the target within the constant acceptable circle R , and the next waypoint should be shifted.

VII. MANOEUVRABILITY CHECK

Due to the low manoeuvrability of buoyancy-driven gliders, a tool is designed to predict if the buoyancy-driven glider is capable to reach the target or the next waypoint in a single turning. The tool is developed for helping to generate waypoints for the guidance system. The initial heading angle, the forward velocity, the initial position, and the target position are identified before the prediction. Then the trajectory of the glider with least turning radius is estimated. Subsequently, the intersection point of the least turning radius trajectory and a line defined by the initial position and target position(waypoint) in x-y plane, is located.

1) *Trajectory Prediction of Least Turning Radius:* When the glider is making a turning, its positions can be estimated by,

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} \int_0^T \cos(\theta_i + rt)v dt \\ \int_0^T \sin(\theta_i + rt)v dt \end{bmatrix} \quad (39)$$

where r is the rotation rate, v is the forward velocity, and θ_i is the initial heading angle.

After integrating, the positions at the time T is,

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} \frac{1}{r} \sin(\theta_i + rT)v - \frac{1}{r} \sin(\theta_i)v \\ -\frac{1}{r} \cos(\theta_i + rT)v + \frac{1}{r} \cos(\theta_i)v \end{bmatrix}. \quad (40)$$

If the lower bound of the forward velocity substitutes the forward velocity and the upper bound of rotation rate substitutes the rotation rate in (40), the estimated positions will satisfy the least turning radius, which is proved by,

$$r_{upper}, v_{lower} = \operatorname{argmin}_{r,v} \|f(r,v)\| = \operatorname{argmin}_{r,v} \left| \begin{bmatrix} \frac{1}{r} \sin(\theta_i + rT)v - \frac{1}{r} \sin(\theta_i)v \\ -\frac{1}{r} \cos(\theta_i + rT)v + \frac{1}{r} \cos(\theta_i)v \end{bmatrix} \right| \quad (41)$$

for an arbitrary time T .

The sign of rotation rate is able to let the glider reach the desired heading angle along the shortest path. To implement it, we need two intermediate variables: the angle deviation(θ_e) and the optimal angle deviation(θ_o). The θ_e is defined as,

$$\theta_e = \theta_d - \theta_c, \quad (42)$$

where θ_c is the current heading angle, and θ_d is the direction angle of a vector between the current waypoint(\mathbf{p}_k) and the next waypoint(\mathbf{p}_{k+1}), which can be solved by,

$$\mathbf{p}_{k,k+1} = \mathbf{p}_{k+1} - \mathbf{p}_k; \quad (43)$$

$$\theta_d = \operatorname{atan2}(\mathbf{p}_{k,k+1}) \quad (44)$$

where $\operatorname{atan2}(y, x)$ is the four-quadrant version of $\arctan(y/x) \in [-\pi/2, \pi/2]$.

The value of the optimal angle deviation (θ_o) is dependent on θ_e ,

- If $\theta_e < \pi$, then $\theta_o = \theta_e$.
- If $\theta_e > \pi$, then $\theta_o = \theta_e + \theta_d - 2\pi$.

When $\theta_o > 0$, the sign of the rotation rate r is positive, otherwise the sign of the rotation rate r is negative.

2) *Intersection Location:* A intersection point between the vector p_k defined in (43) and the estimated trajectory of least turning radius (40) can be found by solving linear equations. If the intersected point locates within the interval between initial position and target position, it means without any disturbance, the glider could reach the target point from the initial position, as the black solid line shown in Fig.9; Otherwise, it is impossible for the glider to arrive at the target position, as the black dotted line presented in Fig.9.

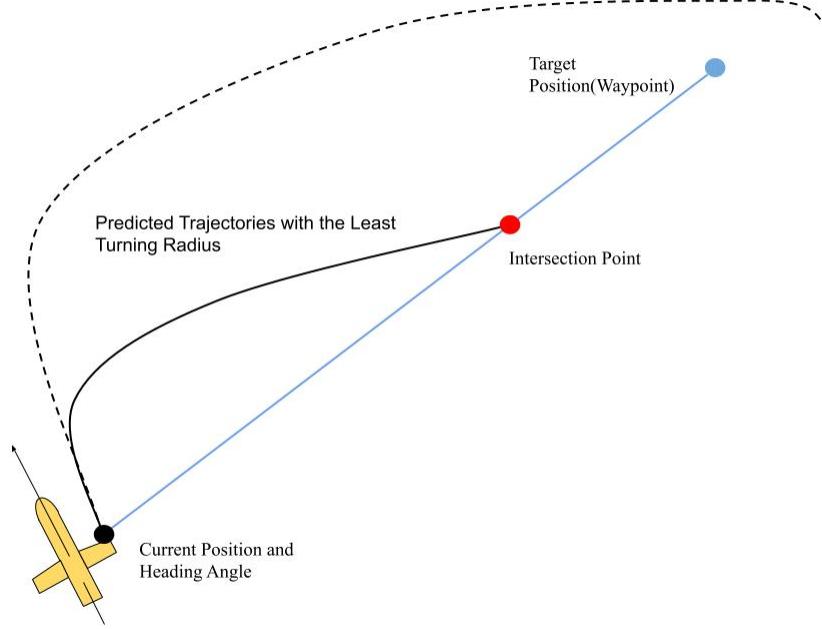


Fig. 9. Intersection Location: The black dotted line indicates the glider cannot reach the target within a single turning theoretically; The black solid line shows the glider could arrive at the target without external disturbances

VIII. EXAMPLE: PETREL-II GLIDER

Petrel II glider is a 1.8m-long torpedo-shaped buoyancy driven glider, developed by Tianjin University independently. It has passed a 1500m-deep water test in the northern part of the South China Sea [39]. The configuration information and hydro-parameters of the gilder have been published [40].

In the example, the configuration of Petrel II is loaded to the simulator via an URDF file. With the Gazebo 3D rendering engine, the floating, the ascending and the descending of Petrel II are presented in Fig.10 to Fig.12. A 3D profile of it's one work cycle is depicted in Fig.13. In the process, the glider is manipulated by the proposed controller, starting from the surface and dividing into 30 meter depth along the desired descending pitch angle 0.6 radians, then going up along the desired ascending pitch angle 0.7 radians. Red arrows refer to direction vectors of velocities of the glider. The coordinates has been transformed into a special NED frame, of which the z-axis is reversed(depth coordinates are negative, instead of being positive), in the sake of clear presentation. The LQR control strategy and the recursive guidance system are implemented on Petrel II as well. In the section, the control module would subscribe model states from the Gazebo server directly. The guidance module would receive data from the GPS sensor plugin with the free noise.

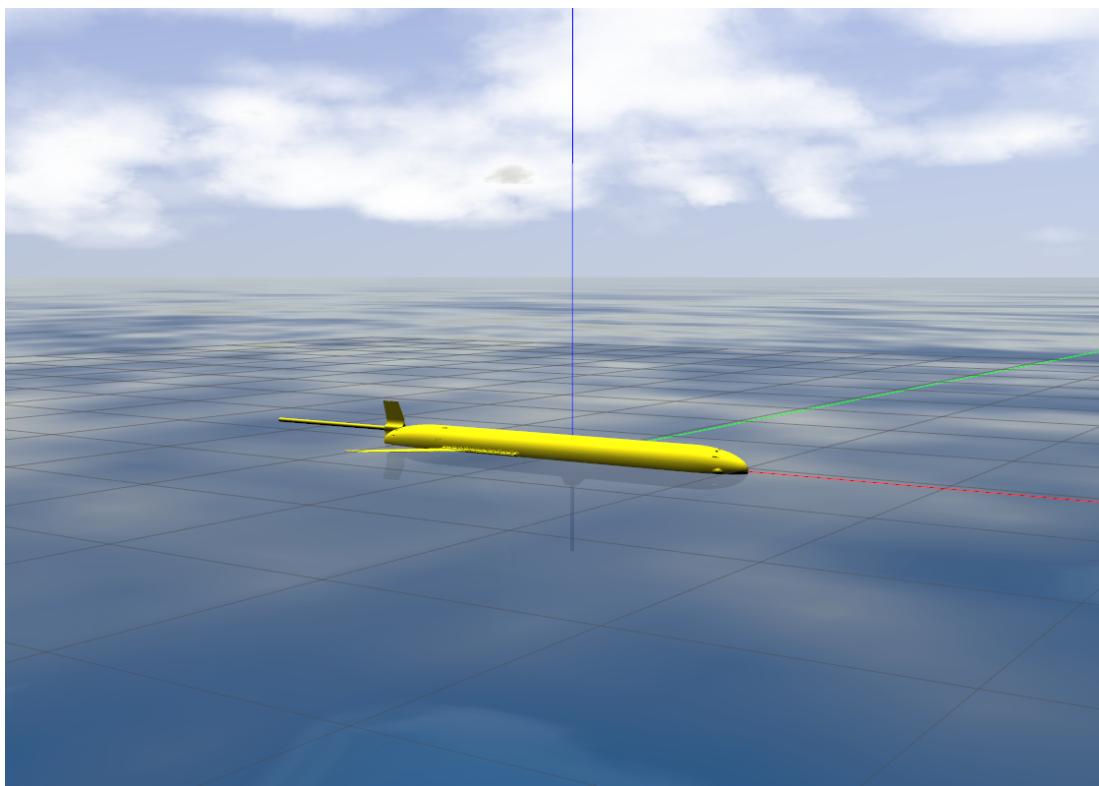


Fig. 10. Floating of the glider in the Gazebo World

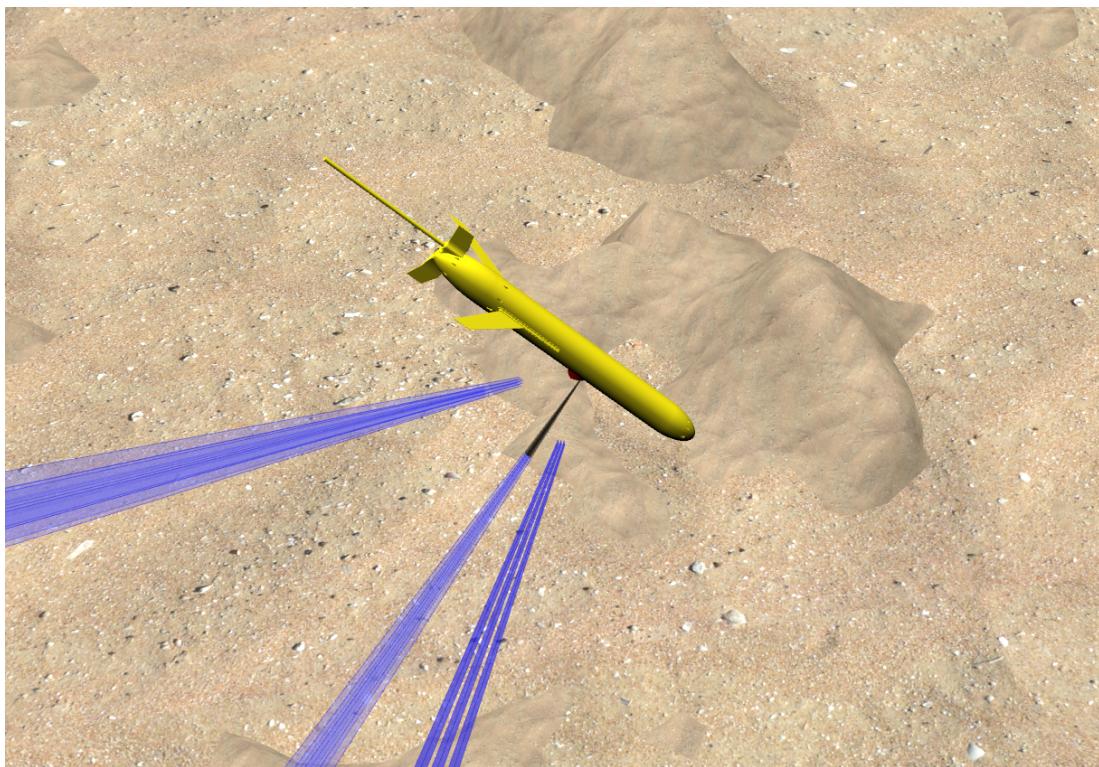


Fig. 11. Descending of the glider in the Gazebo World

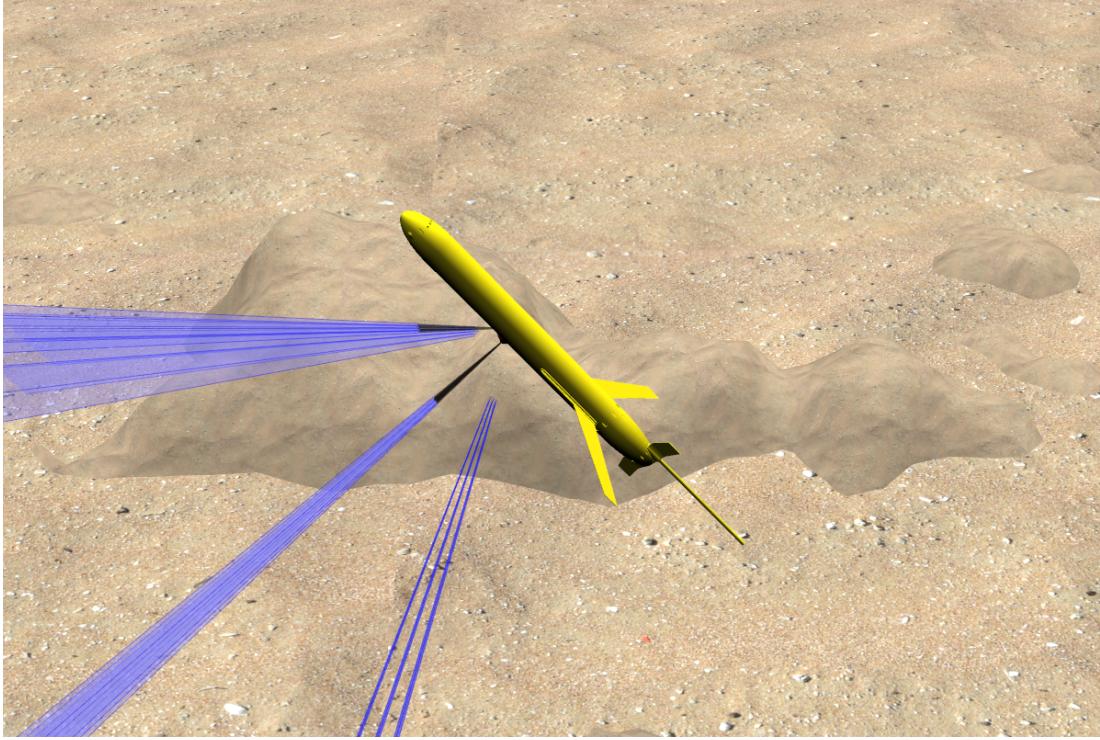


Fig. 12. Ascending of the glider in the Gazebo World

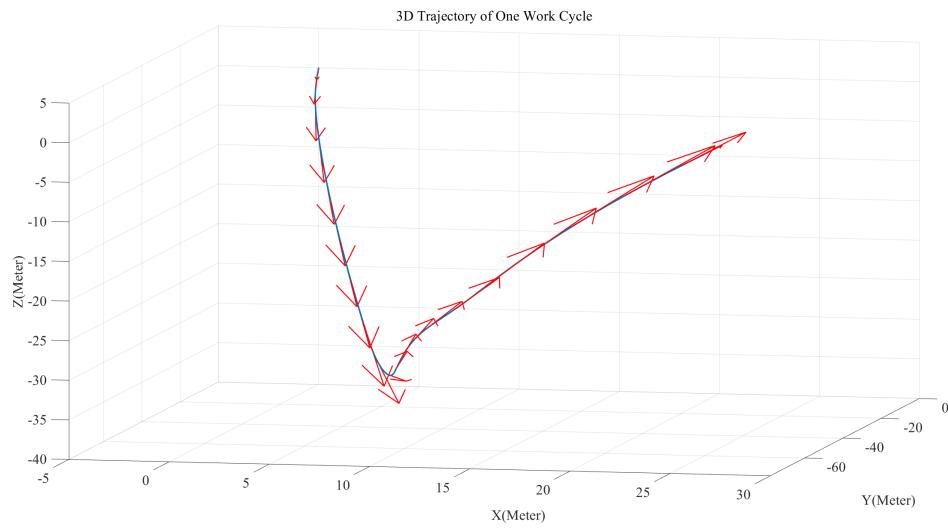


Fig. 13. 3D Trajectory of One Work Cycle

A. Orientation and Velocity Control

The regulation of the orientation and the forward velocity of is implemented via the LQR control module. As buoyancy-driven gliders lack the good manoeuvrability, the yaw control process is chronic, and lasts for multiple work cycles before reaching the desired angle. As shown in Fig.14, the desired yaw angle was 1.5 radians, the glider might take over 1000 seconds to accomplish the goal from -1.5 radians. Within the process, the glider had experienced four work cycles, and each variation of the pitch angle affected the heading angle(or yaw angle), because of the gravity centre changing dramatically.



Fig. 14. Heading Angle Control

The pitch angle control is much more smooth than that of yaw angle, since it is affected directly from the gravity centre which is adjusted through the translational position of the battery package and the ballast mass. Fig.15 shows the variation of the pitch angle after the controller was given -0.3 desired pitch angle. Due to the changing of the gravity centre, the glider stopping on surface had short-time oscillations about the y-axis of its body-fixed frame and got the desired pitch angle within 30 seconds. When the glider arrived at the desired depth, the pitch angle was regulated to 0.3 within 20 seconds, to make sure the glider tilt up. Once the glider was submerged, water would give the glider the additional damping. Hence, in the second transition, the oscillation is smaller.

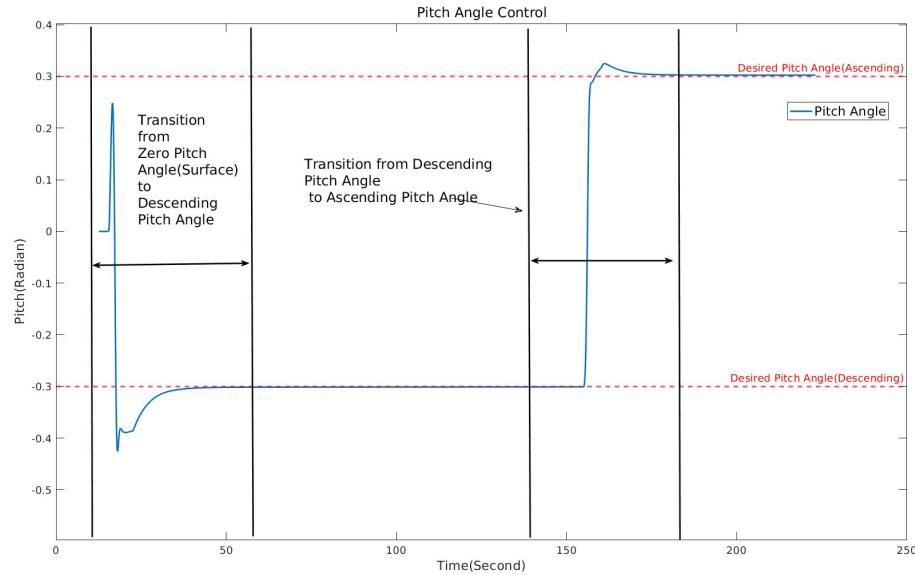


Fig. 15. Pitch Angle Control

In the instance, the pitch angle control range is from -0.6432 to 0.755 radians. This is because the restriction of the linear translation of the battery package and the ballast mass, coming from Petrel-II's mechanical layout.

The forward(x-axis on its body frame) speed control is still implemented by the LQR control module. The desired forward speed was set 0.5 m/s, which was reached within 20 seconds, as shown in Fig.16. The speed was affected by the transition from the descending to the ascending, for the the orientation and the gravity centre of the gilder was changing significantly

during the time. In practical situations, the forward speed control is not essential, particularly when the glider dose not equip with the DVL(Doppler Velocity Log), in order to achieve the low electric power consumption.

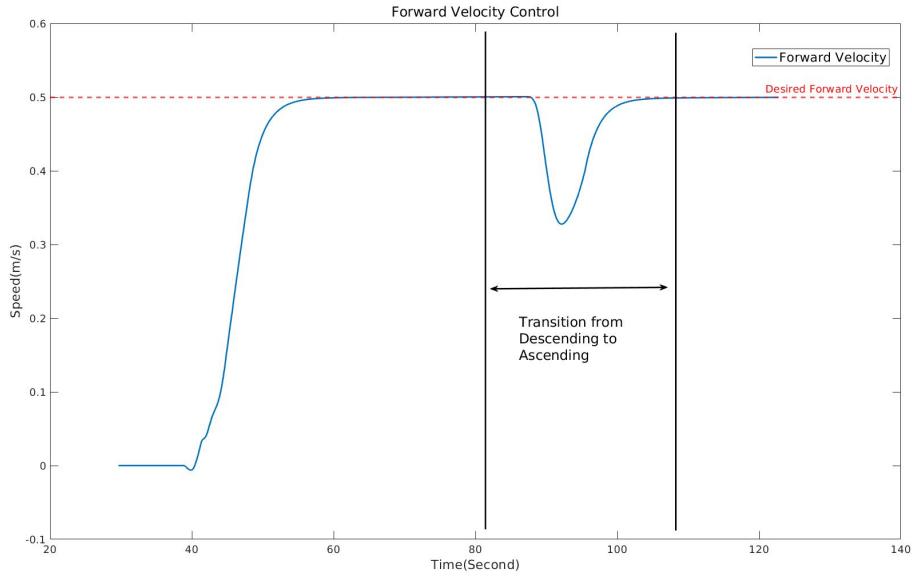


Fig. 16. Forward Velocity Control

B. Waypoints Tracking

The waypoints tracking is realised by the recursive guidance system. Due to the low manoeuvrability and the lack of high precision underwater positioning means, the buoyancy-driven glider would update its desired heading angle and pitch angles after each work cycle. There were five waypoints given, as shown in Fig.17 and Fig.18, the green arrows indicate the direction of the velocity at that position. The glider was able to go through all waypoints smoothly with the constant circle of acceptance $R = 15m$. However, when the constant circle of acceptance was switched to $R = 10m$, there was a failure, when the glider attempted to get in the adaptive circle of acceptance of the waypoint 4, because of low manoeuvrability. Hence, the glider turned around and made a second attempt to the target.

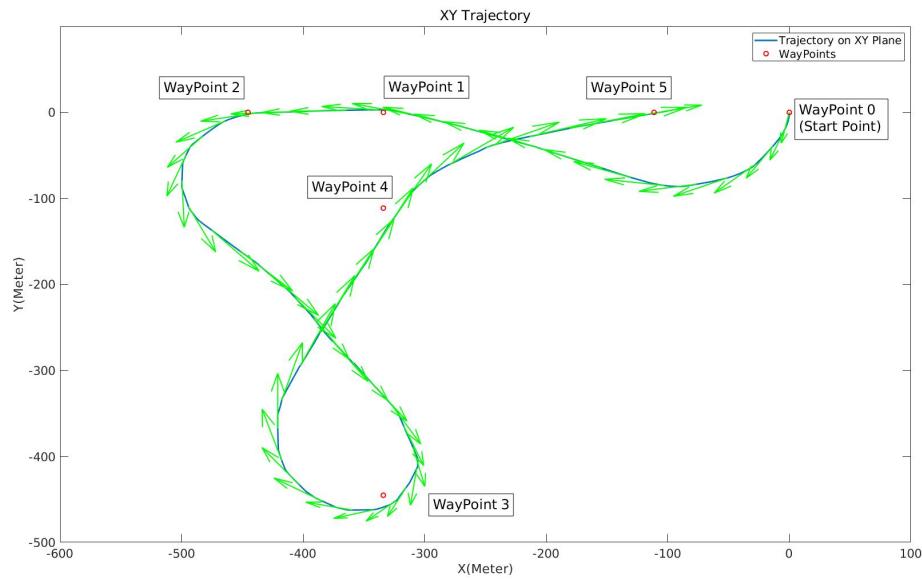


Fig. 17. Trajectory on XY-Plane of Waypoints Tracking with $R = 15m$

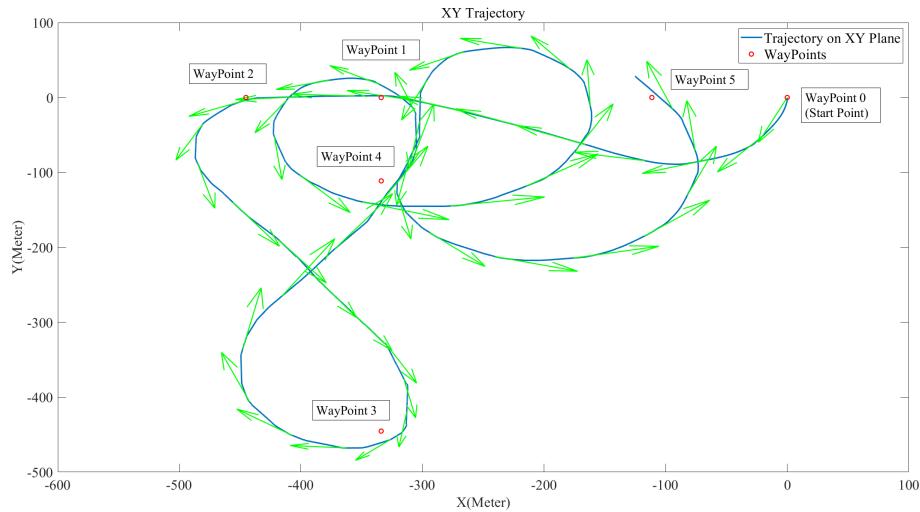


Fig. 18. Trajectory on XY-Plane of Waypoints Tracking with $R = 10m$

IX. CONCLUSION

In the paper, a novel simulator for buoyancy-driven gliders is proposed, which brings hydrodynamics and hydrostatics effects into the simulation. This work consisting of the kinetic module, the LQR control module, the recursive guidance module and the tool aiding to design waypoints, will accelerate the software development for low-manoeuvrability buoyancy-driven gliders. We use the Petrel-II glider as an example to present these features. Note that all of those are developed on ROS1 and Gazebo Classic using C++ language, since some sensors' package compatibility issues. In future, the simulator will be rewritten using Gazebo Sim and ROS2 software frameworks.

REFERENCES

- [1] R. E. Davis, C. C. Eriksen, C. P. Jones *et al.*, “Autonomous buoyancy-driven underwater gliders,” *The technology and applications of autonomous underwater vehicles*, pp. 37–58, 2002.
- [2] J. G. Graver, R. Bachmayer, N. E. Leonard, and D. M. Fratantoni, “Underwater glider model parameter identification,” in *Proc. 13th Int. Symp. on Unmanned Untethered Submersible Technology (UUST)*, vol. 1, 2003, pp. 12–13.
- [3] K. I. M. Arshad, “Buoyancy-driven underwater glider modelling and analysis of motion control,” *Indian Journal of Marine Sciences*, vol. 41, no. 6, pp. 516–526, 2012.
- [4] D. C. Webb, P. J. Simonetti, and C. P. Jones, “Slocum: An underwater glider propelled by environmental energy,” *IEEE Journal of oceanic engineering*, vol. 26, no. 4, pp. 447–452, 2001.
- [5] J. Sherman, R. E. Davis, W. Owens, and J. Valdes, “The autonomous underwater glider” spray”, *IEEE Journal of Oceanic Engineering*, vol. 26, no. 4, pp. 437–446, 2001.
- [6] D.-Y. Xue, Z.-L. Wu, Y.-H. Wang, and S.-X. Wang, “Coordinate control, motion optimization and sea experiment of a fleet of petrel-ii gliders,” *Chinese Journal of Mechanical Engineering*, vol. 31, pp. 1–15, 2018.
- [7] E. H. Henriksen, I. Schjølberg, and T. B. Gjersvik, “Uw morse: The underwater modular open robot simulation engine,” in *2016 IEEE/OES Autonomous Underwater Vehicles (AUV)*. IEEE, 2016, pp. 261–267.
- [8] M. M. Zhang, W.-S. Choi, J. Herman, D. Davis, C. Vogt, M. McCarrin, Y. Vijay, D. Dutia, W. Lew, S. Peters *et al.*, “Dave aquatic virtual environment: Toward a general underwater robotics simulator,” in *2022 IEEE/OES Autonomous Underwater Vehicles Symposium (AUV)*. IEEE, 2022, pp. 1–8.
- [9] P. Cieślak, “Stonefish: An advanced open-source simulation tool designed for marine robotics, with a ros interface,” in *OCEANS 2019-Marseille*. IEEE, 2019, pp. 1–6.
- [10] M. M. M. Manhães, S. A. Scherer, M. Voss, L. R. Douat, and T. Rauschenbach, “Uuv simulator: A gazebo-based package for underwater intervention and multi-robot simulation,” in *Oceans 2016 Mts/Ieee Monterey*. IEEE, 2016, pp. 1–8.
- [11] T. I. Fossen, *Handbook of marine craft hydrodynamics and motion control*. John Wiley & Sons Ltd, 2011.
- [12] N. Koenig and A. Howard, “Design and use paradigms for gazebo, an open-source multi-robot simulator,” in *2004 IEEE/RSJ international conference on intelligent robots and systems (IROS)(IEEE Cat. No. 04CH37566)*, vol. 3. Ieee, 2004, pp. 2149–2154.
- [13] M. Prats, J. Perez, J. J. Fernández, and P. J. Sanz, “An open source tool for simulation and supervision of underwater intervention missions,” in *2012 IEEE/RSJ international conference on Intelligent Robots and Systems*. IEEE, 2012, pp. 2577–2582.
- [14] A. Birk, T. Doernbach, C. Mueller, T. Łuczynski, A. G. Chavez, D. Koehntopp, A. Kupcsik, S. Calinon, A. K. Tanwani, G. Antonelli *et al.*, “Dexterous underwater manipulation from onshore locations: Streamlining efficiencies for remotely operated underwater vehicles,” *IEEE Robotics & Automation Magazine*, vol. 25, no. 4, pp. 24–33, 2018.
- [15] E. Coumans, “Bullet physics simulation,” in *ACM SIGGRAPH 2015 Courses*. ACM, 2015, p. 1.
- [16] M. A. Sherman, A. Seth, and S. L. Delp, “Simbody: multibody dynamics for biomedical research,” *Procedia Iutam*, vol. 2, pp. 241–261, 2011.
- [17] J. Lee, M. X. Grey, S. Ha, T. Kunz, S. Jain, Y. Ye, S. S. Srinivasa, M. Stilman, and C. Karen Liu, “Dart: Dynamic animation and robotics toolkit,” *The Journal of Open Source Software*, vol. 3, no. 22, p. 500, 2018.
- [18] O. Kermorgant, “A dynamic simulator for underwater vehicle-manipulators,” in *International Conference on Simulation, Modeling, and Programming for Autonomous Robots*. Springer, 2014, pp. 25–36.
- [19] T. Watanabe, G. Neves, R. Cerqueira, T. Trocoli, M. Reis, S. Joyeux, and J. Albiez, “The rock-gazebo integration and a real-time auv simulation,” in *2015 12th Latin American Robotics Symposium and 2015 3rd Brazilian Symposium on Robotics (LARS-SBR)*. IEEE, 2015, pp. 132–138.

- [20] D. Burns and R. Osfield, "Open scene graph a: Introduction, b: Examples and applications," in *Virtual Reality Conference, IEEE*. IEEE Computer Society, 2004, pp. 265–265.
- [21] I. Vaughn and S. Suman. (2004) Deep submergence ds_sim. [Online; accessed 31-October-2024]. [Online]. Available: http://github.com/Field-Robotics-Lab/ds_sim
- [22] K. Suzuki and K. Kawabata, "Development of a simulator for underwater reconnaissance tasks by utilizing remotely operated robots," in *2020 IEEE/SICE International Symposium on System Integration (SII)*. IEEE, 2020, pp. 1100–1106.
- [23] S. Nakaoka, "Choreonoid: Extensible virtual robot environment built on an integrated gui framework," in *2012 IEEE/SICE International Symposium on System Integration (SII)*. IEEE, 2012, pp. 79–85.
- [24] E. Games. (2024) Unreal engine. [Online; accessed 31-October-2024]. [Online]. Available: <https://www.unrealengine.com/>
- [25] U. Technologies. (2024) Unity 6. [Online; accessed 31-October-2024]. [Online]. Available: <https://unity.com/>
- [26] G. Echeverria, N. Lassabe, A. Degroote, and S. Lemaignan, "Modular open robots simulation engine: Morse," in *2011 ieee international conference on robotics and automation*. IEEE, 2011, pp. 46–51.
- [27] T. I. Fossen and T. Perez. (2004) Marine systems simulator(mss). [Online; accessed 31-October-2024]. [Online]. Available: <https://github.com/cybergalactic/MSS>
- [28] ROS. (2023) Ros wiki-urdf. [Online; accesseed 12-February-2023]. [Online]. Available: <https://wiki.ros.org/urdf>
- [29] N. E. Leonard and J. G. Graver, "Model-based feedback control of autonomous underwater gliders," *IEEE Journal of oceanic engineering*, vol. 26, no. 4, pp. 633–645, 2001.
- [30] Y. Song, W. Shi, Y. Wang, H. Wu, S. Yang, H. Hou, and Y. Xu, "Evaluation of energy consumption and motion accuracy for underwater gliders based on quadrant analysis," *Ocean Engineering*, vol. 285, p. 115399, 2023.
- [31] G. Sim. (2020) Spherical coordinats. [Online; accesseed 12-February-2024]. [Online]. Available: <https://gazebosim.org/api/sim/9/tutorials.html>
- [32] B. Ullah, M. Ovinis, M. B. Baharom, M. Javaid, and S. Izhar, "Underwater gliders control strategies: A review," in *2015 10th Asian Control Conference (ASCC)*. IEEE, 2015, pp. 1–6.
- [33] H. Wu, W. Niu, S. Wang, and S. Yan, "A feedback control strategy for improving the motion accuracy of underwater gliders in currents: Performance analysis and parameter optimization," *Ocean Engineering*, vol. 252, p. 111250, 2022.
- [34] M. M. Noh, M. R. Arshad, and R. M. Mokhtar, "Depth and pitch control of usm underwater glider: Performance comparison pid vs. lqr," 2011.
- [35] S. Zhang, J. Yu, A. Zhang, and F. Zhang, "Spiraling motion of underwater gliders: Modeling, analysis, and experimental results," *Ocean Engineering*, vol. 60, pp. 1–13, 2013.
- [36] python control.org. (2023) Python control systems library. [Online; accesseed 14-February-2024]. [Online]. Available: <https://python-control.readthedocs.io/en/latest/index.html>
- [37] T. S. community. (2025) Scipy documentation. [Online; accesseed 3-February-2025]. [Online]. Available: <https://docs.scipy.org/doc/scipy/index.html>
- [38] E. R. Frank Warmerdam. (1998-2025) Gdal-stable. [Online; accesseed 24-October-2024]. [Online]. Available: <https://gdal.org/en/stable/>
- [39] L. Liu, L. Xiao, S.-Q. Lan, T.-T. Liu, and G.-L. Song, "Using petrel ii glider to analyze underwater noise spectrogram in the south china sea," *Acoustics Australia*, vol. 46, pp. 151–158, 2018.
- [40] H. Wu, W. Niu, S. Wang, and S. Yan, "An optimization method for control parameters of underwater gliders considering energy consumption and motion accuracy," *Applied Mathematical Modelling*, vol. 90, pp. 1099–1119, 2021.