AAM-SEALS: Developing Aerial-Aquatic Manipulators in SEa, Air, and Land Simulator

Tomer Atzili*, **Abhinav Bhamidipati***, **Yashveer Jain***, **William Wang Yang***
Sri Kiran Kommaraju, Karthikeya Kona, Xiaomin Lin, Yantian Zha
University of Maryland, College Park
{tatzili,abhinav7,yashveer,wyang124,komkiran,kkona,xlin01,ytzha}@umd.edu

Abstract: Current simulators lack the ability to accurately model integrated environments that encompass sea, air, and land. To address this gap, we introduce Aerial-Aquatic Manipulators (AAM) in SEa, Air, and Land Simulator (SEALS), a comprehensive and photorealistic simulator designed for AAMs to operate and learn in these diverse environments. The development of AAM-SEALS tackles several significant challenges, including the creation of integrated controllers for flying, swimming, and manipulation, and the high-fidelity simulation of aerial dynamics and hydrodynamics leveraging particle physics. Our comprehensive evaluation demonstrates that our AAM operates smoothly in the SEALS, reflecting photorealistic transitions across air, water, and air-water interfaces. We quantitatively demonstrate the benefits of particle-based hydrodynamics by comparing position-tracking errors across different dynamic systems. AAM-SEALS promises to benefit a broad range of robotics communities, including robot learning, aerial robotics, underwater robotics, mobile manipulation, and robotic simulators. We will open-source our code and data to foster the advancement of research in these fields. Demonstration videos can be accessed through this anonymous Google Drive link: https://shorturl.at/0VAVm

Keywords: Aerial-Aquatic Manipulators, Photo-realistic Robotics Simulators

1 Introduction

Mobile manipulation is a crucial and rapidly advancing field in robotics, offering the potential to revolutionize various industries by enabling robots to interact with and manipulate their environments. This capability is especially valuable in scenarios that are tedious, hazardous, or challenging for humans. Despite its significance, current research has focused predominantly on mobile manipulation in isolated environments – either in the sea, air, or on land. For instance, aerial manipulation involves robots performing tasks while flying, underwater manipulation focuses on submersible robots operating in aquatic environments, and ground-based mobile manipulation deals with robots navigating and interacting on terrestrial surfaces.

However, many real-world applications require robots to operate seamlessly across different environments. For instance, an ideal robotic system for search and rescue missions might need to take off from the ground and navigate through the air for most of the journey, before diving into water to reach and assist victims efficiently. This necessitates mobile manipulators capable of transitioning and functioning effectively across water, air, and land boundaries, as illustrated in Fig. 1.

To address this need, we propose a novel class of robots called Aerial-Aquatic Manipulators (AAMs). AAMs combine the capabilities of aerial manipulators [1, 2, 3, 4, 5], underwater manipulators [6, 7, 8, 9, 10], and aerial-aquatic quadrotors [11, 12, 13, 14, 15]. Our AAMs have unique advantages, such as the ability to navigate large areas efficiently and adaptively select the safest or

*The first four authors contributed equally, and are listed in alphabetical order.

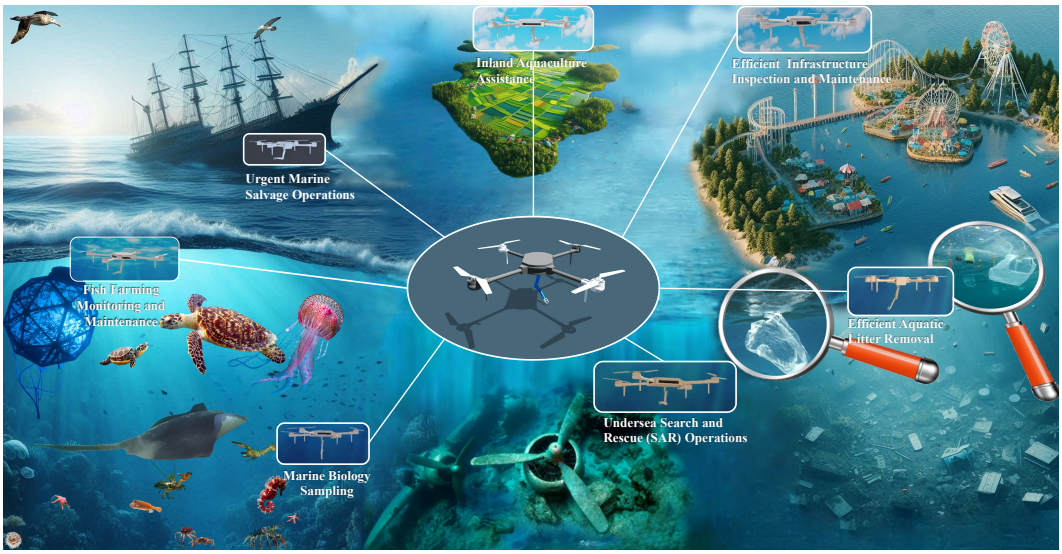

Figure 1: Aerial-Aquatic Manipulators: Demonstrating a wide range of critical applications that leverage their unique capabilities across sea, air, and land environments

most efficient path. For instance, an AAM can fly out of a debris-filled water area, travel through the air to a new location, and then re-enter the water to reach a target area.

The design and construction of such advanced robots poses significant challenges. Directly developing a physical AAM is complex and expensive, involving intricate designs for sensors, mechanics, morphologies, kinematics, and robot-environment interactions. To mitigate these risks and costs, we first propose developing AAMs within a high-fidelity simulation environment. This approach allows us to validate our designs and refine them iteratively, enabling testing, evaluation, validation, and verification (TEVV) before physical implementation.

In this paper, we introduce AAM-SEALS, a comprehensive and photorealistic simulator built on top of NVIDIA Isaac Sim [16]. AAM-SEALS enables Aerial-Aquatic Manipulators (AAMs) to operate and learn in integrated environments that encompass sea, air, and land (SEALS). Developing AAM-SEALS involved addressing several significant challenges, including the creation of integrated controllers for both flight and manipulation, as well as the high-fidelity simulation of aerial dynamics and hydrodynamics using particle physics [17, 18, 19, 20]. Particle-based hydrodynamics directly models fluids as a set of particles that interact with and constrain each other and surrounding objects, enabling the simulation of complex fluid dynamics and object-fluid interactions. This approach aligns with our goal of enabling AAMs to operate effectively in free-surface flows [21, 22].

We comprehensively evaluate our AAM-SEALS system across various aspects. Firstly, we showcase the teleoperation prowess of our AAM within the SEALS environment. Secondly, we conduct a quantitative comparison between the hydrodynamics of our SEALS, employing particle physics, and the conventional rigid body hydrodynamics used in the cutting-edge photorealistic underwater simulator, UNav-Sim [23]. This comparison entails analyzing position tracking and discerning photo-realism disparities, thus demonstrating the high-fidelity and photorealistic fluid simulations achievable with SEALS. Third, we demonstrate the potential of using AAM-SEALS for robot learning by conducting visual reinforcement learning experiments.

Our contributions are threefold. First, we introduce a novel class of robots, Aerial-Aquatic Manipulators (AAMs), capable of performing a wide range of highly valuable applications. Secondly, we develop a photorealistic, high-fidelity simulation environment tailored specifically for AAMs. Lastly, we demonstrate the effectiveness of our AAM and SEALS by successfully teleoperating our AAM to perform long-horizon mobile manipulation tasks and conducting visual reinforcement learning within our SEALS simulator.

2 Related Work

Aerial-Aquatic Quadrotors: The development of hybrid aerial-aquatic quadrotors has recently gained significant interest due to the popularity of quadrotors and the broad needs of tasks such as filming and aerial-aquatic environmental monitoring. Tan and Chen [12] developed a morphable aerial-aquatic quadrotor with symmetric thrust vectoring to adapt thrust direction for optimal performance in both air and water. They further explored this concept by integrating multi-rotors to refine propulsion systems and mechanical design [11]. Alzu'bi et al. [2] introduced the Loon Copter, a hybrid vehicle with active buoyancy control for smooth transitions between air and water, suitable for underwater exploration and environmental monitoring. Wu et al. [13] demonstrated a tandem dual rotor aerial-aquatic vehicle focusing on efficient propulsion and maneuverability. Liu et al. [15] advanced the field with the TJ-FlyingFish, which features tilt-able propulsion units for improved stability and control in both environments. These works collectively highlight significant progress in hybrid aerial-aquatic vehicles, showcasing innovative approaches to overcome the unique challenges of operating in both air and water. However, they have not considered the addition of manipulators which would drastically enlarge the number of tasks, and effective simulation tools are essential for the further development and testing of these hybrid systems.

Photorealistic Aerial or Underwater Simulators: Simulation environments are crucial for both gathering data and fostering the acquisition of new capabilities by robots. Advanced aerial robotics simulators such as Pegasus [24], built on IsaacSim [16], and AirSim [25] provide high-fidelity rendering. However, simulating underwater environments presents greater challenges. Recent advances have targeted complex underwater environments and maritime scenarios. For example, Zwilgmeyer et al. [26] use Blender to generate underwater datasets, while platforms such as UUV Simulator [27] and UWSim [28] model underwater physics and sensors. Despite their progress, these efforts have been discontinued. DAVE [29] seeks to bridge this gap but struggles with rendering limitations.

More recent simulators such as HoloOcean [30], MARUS [31], and UNav-Sim [23] have improved rendering realism but still struggle to simulate complex free-space fluids and object-water interactions without using particle physics. AuqaSim [32] focuses on near-water tasks, but lacks drone simulation above the water. Many simulators built on Unreal Engine face modifiability challenges and often do not release their original project files. ChatSim [33] integrates ChatGPT with OysterSim [34], enabling easy modifications of the simulated environment and generating photorealistic underwater settings. However, these simulators mainly address deep underwater tasks and often neglect aerial parts and air-water transitions.

3 Aerial-Aquatic Manipulator (AAM)

3.1 AAM Dynamics Modeling

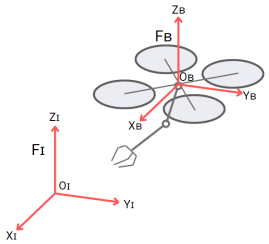

Figure 2: Schematic of a representative Aerial Aquatic Manipulator.

Aerial-Aquatic Manipulation is a novel concept introduced in this work, promising to open a new field of research. The general modeling of Aerial Aquatic Manipulators (AAMs) involves a cross-medium drone platform with n thrusters $T1...Tn$, a manipulator with m degrees of freedom (DoF), and a multi-finger gripper. For simplicity, we use a representative example of an AAM consisting of an aerial-aquatic quadrotor with four thrusters, a manipulator with three DoF, and a three-finger gripper. The quadrotor is chosen for its widespread use and robust performance in tasks such as object retrieval and handling. The manipulator's design enables precise and versatile operations, making it adaptable to a range of environments and tasks.

This AAM model is versatile and can be adapted to other aerial-aquatic platforms, such as hexacopters, or systems with different manipulator configurations and grippers, with minimal modifications. Our AAM serves as a prime example, and the simulator SEALS (discussed in Sec. 4) has been developed to allow researchers to test and refine various AAM designs with reduced costs and risks. We also included a guideline for future researchers to create their customized AAMs in Appendix. E.

The AAM's simulation follows the conventions outlined in the Isaac Sim simulator. Isaac Sim employs a right-handed rule convention where the Z-axis of the inertial frame points upwards, and the Y-axis is aligned with true North, adhering to the East-North-Up (ENU) coordinate system. For the vehicle's body frame, a front-left-up (FLU) convention is adopted. [24] This standardized coordinate system facilitates the integration and simulation of AAM's movements and operations within the virtual environment, ensuring consistency and accuracy in control and navigation algorithms.

In Fig. 3.1., the coordinate frame of inertial is denoted with $C_I : \{X_I, Y_I, Z_I\}$ with the Origin O_I, while the coordinate frame of the body is denoted with $C_B : \{X_B, Y_B, Z_B\}$ with the Origin O_B indicating the center of mass of AAM.

The angular velocity can be expressed as:

$$\dot{\omega} = J^{-1}(\tau - \omega \times J\omega) \tag{1}$$

where, J is the inertia tensor for vehicle expressed in C_B, ω denotes the angular velocity of C_B with respect to C_I expressed in C_B, τ denotes the total torque from each rotor. Following a recent work by Jacinto et al. (2023) [24], we can compute τ by multiplying the forces of individual rotors, represented as the vector $\mathbf{F} = [F_1, \ldots, F_N]$, with an allocation matrix $\mathbf{A}$:

$$\tau = \mathbf{A}\mathbf{F} \tag{2}$$

where the allocation matrix $\mathbf{A}$ is computed based on the quadrotor parameters including the arm length and rotor positions. We will define $\mathbf{A}$ in the following section.

However, most of the work on drone control assumes a fixed center of gravity (CoG), which cannot satisfy our needs of aerial manipulation. We now explain our improvements to handle dynamic changes of CoG in the next subsection, Sec. 3.2.

The dynamics of a 3-DoF manipulator is shown in Eq. 3 (essentially a kinematic equation). The kinematic equation connects how the joints move (joint velocities $\dot{q}$) to how the end of the robot moves (end-effector velocities $\dot{x}$). It uses a Jacobian matrix ($\mathbf{J}(q)$) to calculate these velocities. The manipulator motion is determined using inverse kinematics, allowing the calculation of the joint velocities $\dot{q}$ required to achieve the desired end-effector velocity $\dot{x}$.

$$\dot{x} = \mathbf{J}(q)\dot{q} \tag{3}$$

where $\dot{x}$ is the end-effector velocity, $\dot{q}$ is the joint velocity, and $\mathbf{J}(q)$ is the Jacobian matrix.

3.2 Handling the Change of Center of Gravity (CoG)

In our simulator, we handle dynamic changes in the center of gravity (CoG) of the quadcopter by continuously updating the allocation matrix at each time step. Therefore, our unique design of the allocation matrix $\mathbf{A}$ becomes:

$$\mathbf{A} = \begin{bmatrix} k_{T1} & k_{T2} & \cdots & k_{Tn} \\ (y_1 - y_{\text{CoG}})k_{T1} & (y_2 - y_{\text{CoG}})k_{T2} & \cdots & (y_n - y_{\text{CoG}})k_{Tn} \\ -(x_1 - x_{\text{CoG}})k_{T1} & -(x_2 - x_{\text{CoG}})k_{T2} & \cdots & -(x_n - x_{\text{CoG}})k_{Tn} \\ k_{R1}d_1 & k_{R2}d_2 & \cdots & k_{Rn}d_n \end{bmatrix} \tag{4}$$

where k_{Ti} is the thrust coefficient of the i-th rotor, x_i and y_i are the coordinates of the i-th rotor relative to the body frame, k_{Ri} is the rolling moment coefficient, d_i represents the rotor's rotational direction, and x_{CoG}, y_{CoG} are the coordinates of the center of gravity and will be updated per step[1].

To obtain the required rotor angular velocities ω, the inverse of the allocation matrix $\mathbf{A}^{-1}$ is calculated and applied to the vector of desired force and torques $[\mathbf{F}, \tau_x, \tau_y, \tau_z]^T$:

[1]While in this work x_{CoG} and y_{CoG} are obtained from our simulator, we are aware of methods that can estimate x_{CoG} and y_{CoG} in real world, such as [35, 36]. That said, developing on-line parameter estimators for aerial manipulation is still an open problem, and such features can be added in the future version of this work.

$$\omega^2 = \mathbf{A}^{-1} \begin{bmatrix} \mathbf{F} \\ \tau_x \\ \tau_y \\ \tau_z \end{bmatrix} \tag{5}$$

The squared angular velocities ω^2 are then processed to ensure they are non-negative, followed by normalization if any value exceeds the maximum permissible squared velocity. Finally, taking the square root of these values gives the rotor angular velocities in radians per second.

This dynamic adjustment ensures that our simulator accurately reflects the quadcopter's behavior as its CoG shifts due to varying payloads or changes in configuration, maintaining precise control and stability throughout its operation.

3.3 Control Development

The block diagram showing the AAM control system is presented in Fig. 3. The desired velocity and the desired joint angles for the quadrotor and manipulator, respectively, are calculated using the PID (Proportional-Integral-Derivative) and PD (Proportional-Derivative) controllers.

A PID controller was designed to regulate the velocity state of the drone, drawing inspiration from the work presented in [37]. The performance of the PID controller indicated relatively good attitude stabilization. Equ. 6. was employed to compute the control force using the PID controller for the quadrotor. This force was then allocated to individual rotors to determine their respective angular velocities, as mentioned in Sec. 3.1 and similar to this work [37].

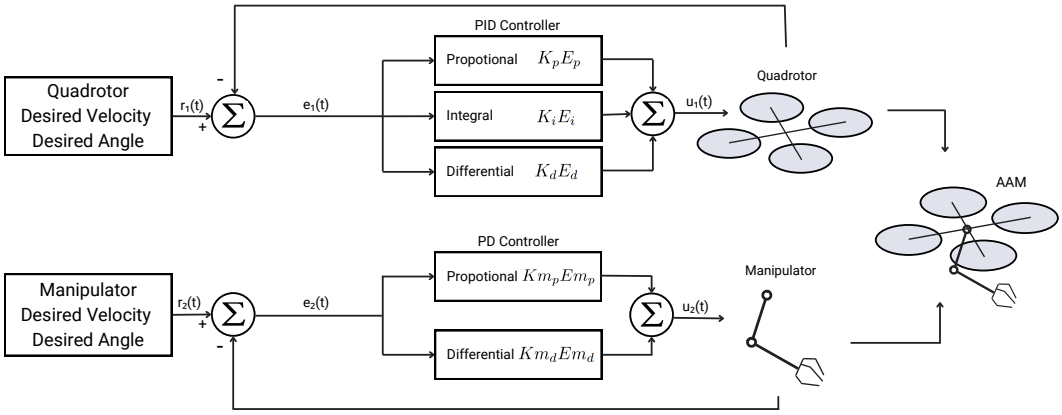

Figure 3: Block diagram of controller for AAM

$$F = K_pE_p + K_dE_d + K_iE_i + [0, 0, m_1g] + m_1a_{ref_1} \tag{6}$$

where

- F is the force for Quadrotor.
- K_pE_p is the Proportional term of PID Controller. K_p is the Proportional Gain and $E_p = v - v_{ref}$, is an error between drone velocity(v) and desired or reference velocity(v_{ref}).
- K_dE_d is the Derivative term of PID Controller. K_d is the Derivative Gain and $E_d = (v - v_{prev})/d_t - a_{ref_1}$, is an error between drone acceleration $((v - v_{prev})/d_t)$ and desired or reference acceleration(a_{ref_1}), and d_t is the time step.
- K_iE_i is the Integral term of PID Controller. K_i is the Integral Gain and E_i is the cumulative summation of E_p at each time step.
- $[0, 0, m_1g]$ is the gravitational force acting on the quadrotor.

- $m_1 a_{ref_1}$ is the force acting on the quadrotor, where m_1 is the mass of the quadrotor, and a_{ref_1} is the reference acceleration, which is the desired acceleration of the quadrotor.
- Dimensions for v, v_{ref}, and a_{ref_1} represent the x, y, z directions in 3D space, denoted as $\mathbb{R}^3$.

The manipulator joints are controlled using the PD controller described by Equ. 7. The function 'set_dof_target_pos()', in Isaac Sim, is employed to define the target joint angle positions for the manipulator, which calculates the desired velocity and angle for the manipulator.

$$F^m = K_p^m E_p^m + K_d^m E_d^m \tag{7}$$

where

- F^m is the force applied to a manipulator joint.
- $K_p^m E_p^m$ is the proportional term of the PD controller, where K_p^m proportional gain of the joint and $E_p^m = x_{ref} - x$ is the error between the desired (reference) angular position x_{ref} and the current angular position x of the joint.
- $K_d^m E_d^m$ is the derivative term of the PD controller, where Km_d is the Derivative Gain and $E_d^m = (x - x_{prev})/d_t$, with x_{prev} being the previous angular position and d_t the time step.

4 SEa, Air, and Lands Simulator (SEALS)

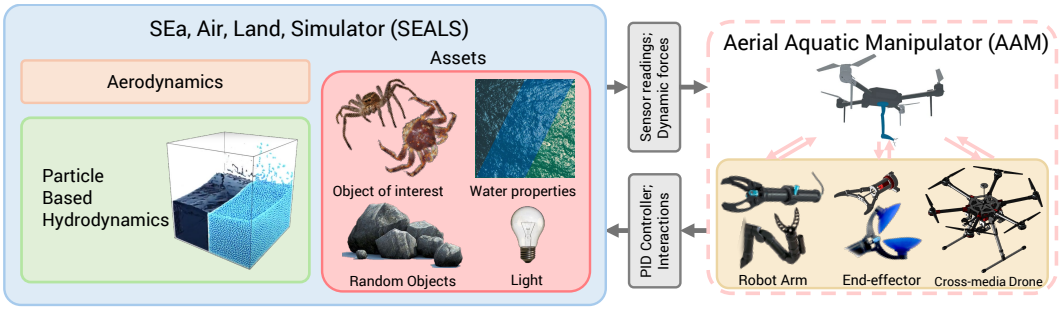

Figure 4: Overview of our SEa, Air, and Lands Simulator (SEALS).

4.1 Aerodynamics Development

Similar to the approach adopted by Jacinto et al. in [24], a simplified linear drag force model is employed to represent the aerodynamic effects that act on the drone. The influence of this linear drag force on our AAM can be expressed using the following equation (Equation 8):

$$F_d = c\mathbf{v} \tag{8}$$

where:

- F_d denotes the drag force with units of N (dimension $\mathbb{R}^3$).
- $\mathbf{v} = [\dot{x} \quad \dot{y} \quad \dot{z}]^T$ represents the linear velocity of the body frame (F_B) with respect to the world frame (F_I).
- c is a constant vector with units of $N/(m/s)$ (dimension $\mathbb{R}^3$), representing the drag coefficient dependent on the velocity acting on the body along each axis. Each element of c lies within the range $[0, 1)$.

4.2 Underwater Dynamics Development

One of the most challenging aspects of building a high-fidelity simulator that features an underwater domain is accurately modeling hydrodynamics and hydrostatics. Water particles behave in complex,

often unpredictable, movement and collision. As such, modeling the forces acting in resistance to underwater motion of a rigid body cannot be accurately calculated by rigid body hydrodynamic equations as used in simulators such as MARUS [31] and UNav-Sim [23] across various underwater environments. To increase the fidelity of the underwater simulation, smoothed particle hydrodynamics (SPH) [38] has been used to simulate the behaviors of individual fluid particles and how they interact with each other and the environment. This method is particularly viable for representing complex fluid interactions such as oceanography, currents, waves, and boundary conditions concerning hydrodynamics [17, 39, 40]. In addition, there are a variety of applications in which SPH excels, including computational biology [41], simulation of underwater landslides [40], and modeling of ice formations in a sea [42].

The variety of useful and high-fidelity applications of SPH makes it an attractive choice to model hydrodynamics for SEALS, but there are some core stability and computational issues, as noted by Macklin and Müller [20]. To address this, Macklin and Müller introduced a method titled position-based dynamics (PBD). This technique incorporates SPH, but introduces a constant density constraint that enforces particle incompressibility, allowing for longer timesteps in calculation and better performance when scaled [20, 43]. It is for these reasons that we chose Isaac Sim's PhysX engine to simulate high-fidelity hydrodynamics using PBD [19]. This system gives SEAL a strong and cutting-edge balance of realistic dynamics, breadth of application, and computational efficiency, all of which will only increase as hardware improves. We have included an overview of the hydrodynamics in Appendix. B.

4.3 Simulation Realism

The underwater part of our SEALS has the unique feature of enhancing realism as follows.

Realistic Air-Water Transition:

Our SEALS system uses particle-based hydrodynamics to achieve highly realistic air-water transitions, capturing both dynamic interactions and detailed rendering. As discussed in Appendix B, this approach simulates cohesion and surface tension, allowing realistic interactions between fluid and solid surfaces. Grounded in solid theoretical principles, we observe that particle-based hydrodynamics in SEALS effectively simulate water splashes when the AAM impacts the water and damping effects as it transitions from air to water. Figure 10 illustrates the splash of water, while the damping effect is demonstrated by deactivating the AAM's thrusters and allowing it to descend into the water under gravity. This effect causes a sudden change in acceleration as the AAM enters the water. Due to disturbances in the surface of natural water, such as wind-induced waves, the AAM loses the balance it maintains in the air once it submerges. For a detailed demonstration, please refer to the accompanying video: `https://shorturl.at/XG9Cr`

Light Attenuation:

Light attenuation in water refers to the gradual reduction in the intensity of light as it travels through water. This phenomenon occurs as a result of the absorption and scattering of light by water molecules and suspended particles. Different wavelengths of light are absorbed at varying rates, with longer wavelengths like red being absorbed more quickly, while shorter wavelengths like blue penetrate deeper. This results in objects appearing bluer and more blurred as they move farther away from the observer.

Figure 5: Visual representations of light attenuation when the robot dives deeper.

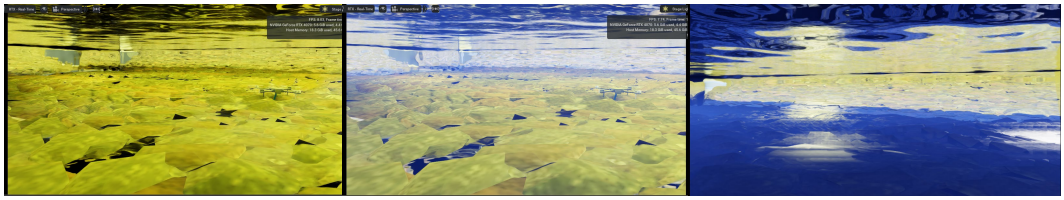

Figure 6: Visual representations of an underwater environment illustrating the effects of wavelength absorption, and light scattering (caustics).

In addition to absorption, light scattering, often observed as caustics, occurs when light rays bend and disperse as they pass through varying densities in the water, creating intricate patterns of light and shadow on underwater surfaces. As a result of these effects, objects appear less bright and can even change color as they move further away from the light source in underwater environments.

Our SEALS system supports these features to further enhance realism beyond what is offered by Isaac Sim. As a result, objects appear less bright and can even change color as they move further away from the light source in underwater environments. Our SEALS supports such features to further enhance realism over Isaac Sim. Fig. 5 shows the control of light intensity with respect to water depth, while Fig. 6 shows the light absorption effects and the light scattering (caustics) effects.

Realistic Wave-Drone Interaction:

The causes of ocean waves are diverse and winds can also vary significantly. While wind is the primary driver, generating waves by transferring energy to the water's surface, other factors also play a role. Seismic activity, such as underwater earthquakes, volcanic eruptions, or landslides, can produce tsunamis that may reach heights exceeding 100 feet (30 meters) in extreme cases. Additionally, the gravitational pull of the moon and sun creates tidal waves, which are typically more gradual and predictable compared to wind-driven waves and tsunamis. Therefore, simulating water waves in a controllable way is an important feature of our SEALS to enhance realism.

In this demo video: `https://shorturl.at/aT8yf`, we showcase an AAM in free fall that is unexpectedly struck by an ocean wave. The sudden shifts in acceleration caused by the wave impact are quantitatively illustrated in Fig. 12.

Realistic Aquatic Animals:

While most photorealistic simulators focus primarily on sensory realism, our SEALS system is the first to also emphasize the realistic simulation of environmental animal behavior. We have meticulously developed detailed meshes and kinematic models of aquatic animals, such as crabs and sea spiders, and equipped them with controllers that facilitate robot learning for developing control policies, as shown in Fig. 7 and 8.

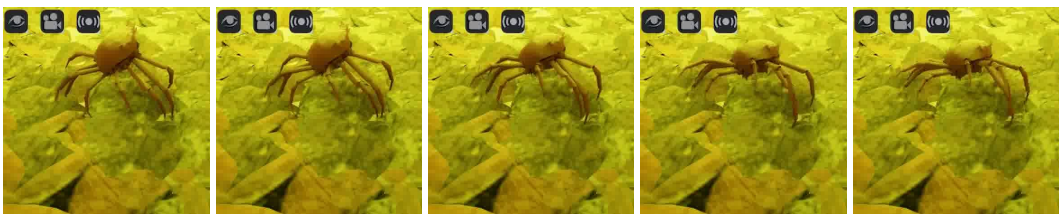

Figure 7: Visual representations of a simulated crab slowly walking on a sea floor.

Figure 8: Video frames that show a simulated sea spider caught by our Aerial-Aquatic Manipulator.

This behavioral realism is crucial for practical applications in in-land aquaculture assistance, marine biology sampling, and fish farming, as shown in Fig.1. By creating digital replicas of real-world animals, SEALS allows for more realistic and effective training of the Aerial-Aquatic Manipulator (AAM). However, achieving realistic aquatic animal simulations is challenging, requiring the construction of detailed meshes, accurate segmentation into parts, precise joint definitions to enable realistic movements, and the integration of joint controllers with reinforcement learning to develop control policies. We provide guidelines in Appendix D on how we accomplished this.

4.4 Sensors and Perceptual Modalities for Robots

In this initial version of AAM-SEALS, We implemented the following sensors.

Figure 9: Camera positions (red, green, and blue arrows are X, Y, and Z axes, respectively) and views of the cameras

Contact Sensor: This sensor detects physical contact between the gripper attached to drone manipulator and other rigid bodies in the environment. When a force exceeding a predefined threshold is applied to the body where the sensor is attached, the sensor transmits a signal indicating contact. The Contact Sensor extension utilizes the PhysX Contact Report API to generate a reading comparable to real-world contact cells or pressure sensors. For this experiment, contact sensors were positioned at each gripper fingertip.

Camera Sensors: The drone is equipped with two RGB-Depth camera sensors, as depicted in Fig. 9. The first is a front-facing camera mounted on the side of the drone, providing a forward view. The second is a downward-facing camera located on the belly near the edge of the drone, designed to capture a view with maximum overlap of the manipulator workspace.

4.5 Control Interfaces to Robots

The control interface between Reinforcement Learning and the Isaac Simulator involves a system in which the application sends commands for quadrotor velocity, joint angles, and gripper action to the Actions module. These actions are managed by the Controller module, which includes a PID Controller for quadrotor velocity, a PD Controller for manipulator joint angles as mentioned in Sec. 3.3, and a gripper command module. The PID Controller calculates the quadrotor's velocity, simulates the drone dynamics, converts force on the drone into rotor forces using the Allocation Matrix, and calculates rotor torques. The PD Controller manages the manipulator dynamics by determining joint angles, while the gripper command controls the open/close actions. These outputs

are sent to the Drone Dynamics and Manipulator Dynamics models, which process the physical behavior of the drone and manipulator. Finally, the results are sent to the NVIDIA Isaac Sim, enabling real-time simulation and control of the drone and manipulator.

5 Evaluation

Our AAM-SEALS system includes both the robot and the simulator, guiding our evaluation to address the following questions:

1. Can our AAM be teleoperated to both navigate and manipulate effectively?
2. How accurately can the AAM follow a predefined trajectory?
3. Does the particle-based fluid dynamics in our SEALS provide high-fidelity simulations?
4. Is AAM-SEALS suitable for robot learning?

To address these questions, our evaluation focuses on both control and learning aspects. Specifically, we use our teleoperation system to collect several trajectories and replay them in AAM-SEALS. To quantitatively assess the fidelity of particle-based underwater dynamics, we implemented position-based water dynamics from the recently published photorealistic underwater simulator, UNav-Sim [23]. We then compute position tracking and compare between particle-based and position-based dynamics, thus addressing Questions 1, 2, and 3. Additionally, we conducted visual reinforcement learning experiments to answer question 4.

5.1 Teleoperation

In Fig. 10, we demonstrate the teleoperation capability of our AAM within the SEALS simulator, showcasing its ability to fly into and out of water. The figure highlights the realistic water splashes generated by the interaction between the AAM and the water surface.

Figure 10: Teleoperated demonstration showcasing the Aerial-Aquatic Manipulator (AAM) seamlessly transitioning between air and water, creating dynamic and visually striking water splashes.

5.2 Particle-based Hydrodynamics

Given the complexities of achieving high-fidelity hydrodynamics, it is crucial to validate the quality of our simulations. As detailed in Sec. 4.2, our simulator utilizes position-based dynamics to model the interactions between water particles and rigid bodies. This approach contrasts sharply with systems that rely on motion equations to apply forces directly to rigid bodies, such as those used in the state-of-the-art underwater simulator UNav-Sim [23]. We refer to the approach used in UNav-Sim as *rigid-body hydrodynamics*.

We evaluate the dynamic performance of the AAM in SEALS from two key perspectives: 1) air-water transition, 2) waves-AAM interaction, and 3) underwater behavior. To quantitatively assess the air-water transition and waves-AAM interaction dynamics, we disabled the AAM's thrusters and allowed it to fall freely from the air into the water. We recorded the changes in the AAM's acceleration over time, as shown in Fig. 11, which clearly demonstrates a sudden shift in acceleration at the moment the AAM enters the water, caused by water damping effects.

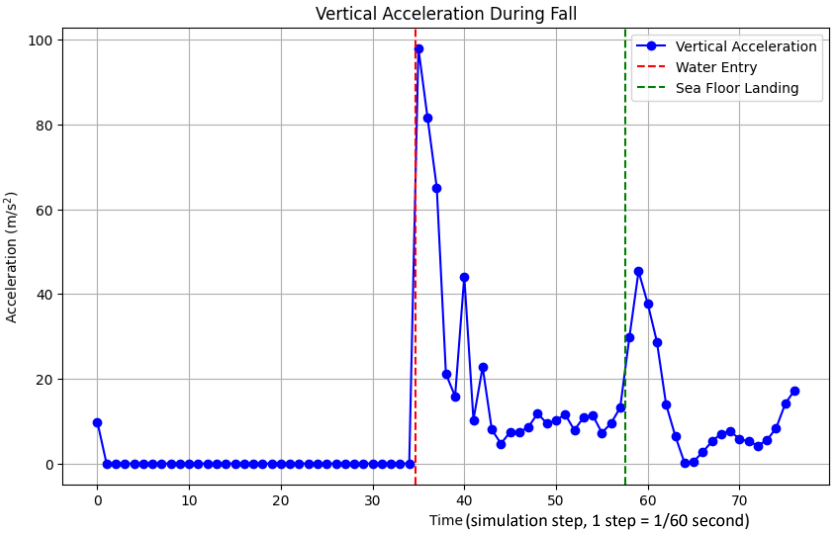

Figure 11: Plot of acceleration along z-direction over time while AAM falls freely from the air into the water.

Likewise, the Fig. 12 shows drastic shifts in accelerations along x, y, and z directions at the moment when AAM was hit by an ocean wave.

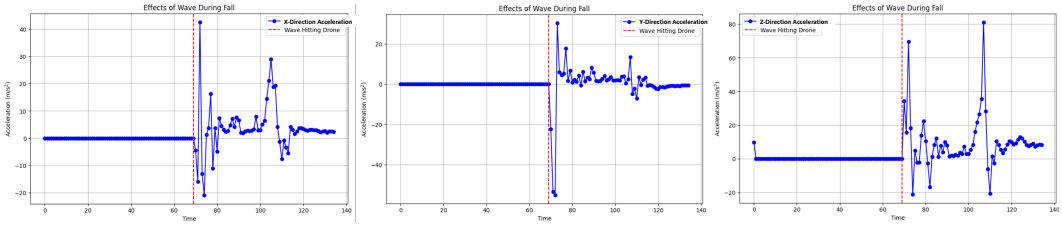

Figure 12: Plot of acceleration along x, y, and z directions over time while AAM falls freely from the air and gets hit by a wave.

To quantitatively evaluate the underwater dynamics, we ran the AAM along the same trajectory in three different environments: aerodynamic, position-based hydrodynamics, and UNav-Sim's rigid-body hydrodynamics. Given that UNav-Sim is a state-of-the-art underwater simulator, we would expect the position tracking in our simulator to closely resemble that of UNav-Sim's hydrodynamics. The environment with aerodynamic dynamics serves as a critical reference for this comparison. The position tracking results, shown in Fig. 13, indicate that at low velocities, our position-based dynamics (PBD) closely aligns with the behavior of the recently published UNav-Sim's rigid-body hydrodynamics, while differing significantly from the aerodynamic environment, as highlighted in the red rectangles. This comparison demonstrates that our hydrodynamics simulation is both similar to UNav-Sim's and distinctly different from the aerodynamic environment, underscoring the validity and accuracy of our hydrodynamics model.

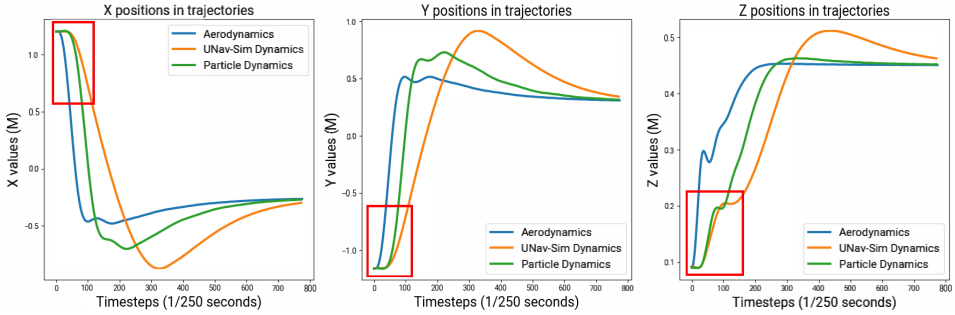

Figure 13: Position tracking of the trajectory after executing the same path across three different dynamic environments: aerodynamics, position-based hydrodynamics (in our SEALS), and UNav-Sim's hydrodynamics.

However, the visual demonstration of the AAM running in UNav-Sim's rigid-body hydrodynamics, shown in Fig. 14, , lacks the visual effects during air-water transitions. In contrast, our particle-based hydrodynamics, as illustrated in Fig. 10, captures these transitions with striking realism. The difference highlights the superior photorealism achieved with position-based dynamics. Thus, overall, position-based hydrodynamics offers better performance.

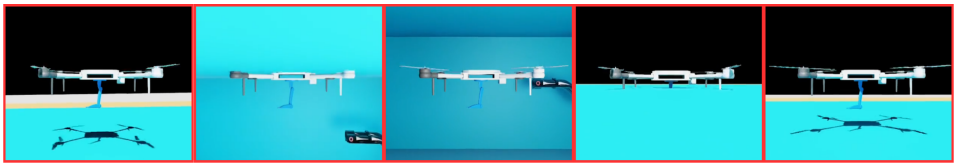

Figure 14: Air-Water Transition in an environment using UNav-Sim's Rigid-Body Hydrodynamics, which lacks realistic water splash effects.

5.3 Visual Reinforcement Learning Evaluation

We applied the Soft Actor-Critic algorithm [44], a robust reinforcement learning (RL) method for continuous control, to train our AAM in the SEALS environment to reach objects such as crabs in Fig. 9, using distance-based rewards. The observation space included two 128 x 128 RGB images from the front and bottom cameras on the AAM (resulting in six channels in total) and the global poses of the AAM and the target object. The cumulative average rewards over the past 100 episodes, shown in Fig. 8, indicate that the reinforcement learning converges. For more detailed robot learning results, please refer to Appendix C.

6 Conclusion

Our work makes significant contributions to the field of robotics by introducing a new class of robots, Aerial-Aquatic Manipulators (AAMs), and developing the first high-fidelity simulator that integrates sea, air, and land environments. The unique benefits of this work include enabling AAMs to seamlessly transition between different environments and perform complex tasks that require multi-terrain capabilities.

Limitations: While our research presents a robust simulation environment, we acknowledge the limitation of not being able to verify the Sim2Real transfer fully. Developing a physical AAM involves substantial research efforts that

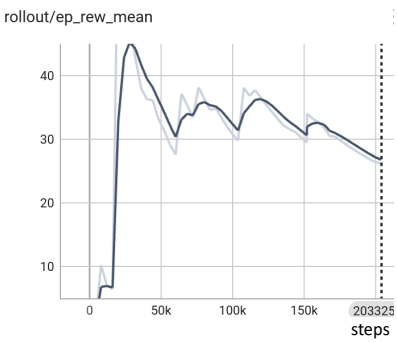

Figure 15: Visual reinforcement learning results

could constitute a separate study. In essence, building physical and simulated AAMs presents a chicken-and-egg problem, where the development of physical and simulated AAMs influences each other. However, it's important to note that many highly regarded works that develop expensive photorealistic simulators, such as [45, 46, 47, 48, 23, 49], do not include physical robot experiments due to the significant challenges involved. Our AAM-SEALS distinguishes itself from these works by not only providing a state-of-the-art simulation environment but also introducing a novel class of robots with carefully designed morphology, kinematics, dynamics, and control systems. We believe AAM-SEALS opens a broad avenue for future research efforts:

- **Supporting the Development of Physical AAMs:** Leveraging AAM-SEALS to aid in the development of physical AAMs is an ongoing project. This simulator will provide critical insights and validation before constructing physical prototypes.
- **Modeling Power Depletion Effects:** Future versions of AAM-SEALS could incorporate the impact of battery depletion on motors, control systems, and planning, as exemplified in [50]. This addition would enhance the realism and practical utility of the simulations, allowing for more accurate modeling of how power limitations affect AAM performance.
- **Enhanced Manipulation:** Attaching a second manipulator to the AAM and exploring the resulting novel opportunities within AAM-SEALS could significantly expand the robot's capabilities.
- **Efficient Simulation:** Simulating particle physics is computationally expensive. Future research could focus on developing a hierarchical simulation approach, where high-resolution simulations are limited to the local region around the AAM. This localized high-resolution simulation would be informed by an outer, lower-resolution simulation that incorporates broader environmental factors such as temperature, depth, and ocean flow.

References

[1] H. B. Khamseh, F. Janabi-Sharifi, and A. Abdessameud. Aerial manipulation—a literature survey. *Robotics and Autonomous Systems*, 107:221–235, 2018.

[2] H. Alzu'bi, I. Mansour, and O. Rawashdeh. Loon copter: Implementation of a hybrid unmanned aquatic–aerial quadcopter with active buoyancy control. *Journal of field Robotics*, 35 (5):764–778, 2018.

[3] K. Abdulmajeed. Autonomous control of a quadrotor-manipulator; application of extended state disturbance observer. *arXiv preprint arXiv:1910.09052*, 2019.

[4] Z. Samadikhoshkho, S. Ghorbani, F. Janabi-Sharifi, and K. Zareinia. Nonlinear control of aerial manipulation systems. *Aerospace Science and Technology*, 104:105945, 2020.

[5] N. Imanberdiyev and E. Kayacan. A fast learning control strategy for unmanned aerial manipulators. *Journal of Intelligent & Robotic Systems*, 94:805–824, 2019.

[6] M. Lee and H.-S. Choi. A robust neural controller for underwater robot manipulators. *IEEE Transactions on Neural Networks*, 11(6):1465–1470, 2000.

[7] W. Zhang, H. Xu, and X. Ding. Design and dynamic analysis of an underwater manipulator. In *Proceedings of the 2015 Chinese Intelligent Automation Conference: Intelligent Technology and Systems*, pages 399–409. Springer, 2015.

[8] D. Youakim and P. Ridao. Motion planning survey for autonomous mobile manipulators underwater manipulator case study. *Robotics and Autonomous Systems*, 107:20–44, 2018.

[9] S. Sivčev, J. Coleman, E. Omerdić, G. Dooly, and D. Toal. Underwater manipulators: A review. *Ocean engineering*, 163:431–450, 2018.

[10] P. Cieslak, P. Ridao, and M. Giergiel. Autonomous underwater panel operation by girona500 uvms: A practical approach to autonomous underwater manipulation. In *2015 IEEE International conference on robotics and automation (ICRA)*, pages 529–536. IEEE, 2015.

[11] Y. H. Tan and B. M. Chen. Design of a morphable multirotor aerial-aquatic vehicle. In *Oceans 2019 Mts/IEEE Seattle*, pages 1–8. IEEE, 2019.

[12] Y. H. Tan and B. M. Chen. A morphable aerial-aquatic quadrotor with coupled symmetric thrust vectoring. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2223–2229. IEEE, 2020.

[13] S. Wu, M. Shao, S. Wu, Z. He, H. Wang, J. Zhang, and Y. You. Design and demonstration of a tandem dual-rotor aerial–aquatic vehicle. *Drones*, 8(3):100, 2024.

[14] I. Semenov, R. Brown, and M. Otte. Control and dynamic motion planning for a hybrid air-underwater quadrotor: Minimizing energy use in a flooded cave environment. *arXiv preprint arXiv:2301.00936*, 2023.

[15] X. Liu, M. Dou, D. Huang, S. Gao, R. Yan, B. Wang, J. Cui, Q. Ren, L. Dou, Z. Gao, et al. Tj-flyingfish: Design and implementation of an aerial-aquatic quadrotor with tiltable propulsion units. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7324–7330. IEEE, 2023.

[16] J. Liang, V. Makoviychuk, A. Handa, N. Chentanez, M. Macklin, and D. Fox. Gpu-accelerated robotic simulation for distributed reinforcement learning. In *Conference on Robot Learning*, pages 270–282. PMLR, 2018.

[17] C. A. D. Fraga Filho, C. A. D. Fraga Filho, and Castro. *Smoothed Particle Hydrodynamics*. Springer, 2019.

[18] M. Macklin and M. Müller. Position based fluids. *ACM Transactions on Graphics (TOG)*, 32 (4):1–12, 2013.

[19] M. Macklin, M. Müller, N. Chentanez, and T.-Y. Kim. Unified particle physics for real-time applications. *ACM Transactions on Graphics (TOG)*, 33(4):1–12, 2014.

[20] M. Müller, B. Heidelberger, M. Hennix, and J. Ratcliff. Position based dynamics. *Journal of Visual Communication and Image Representation*, 18(2):109–118, 2007.

[21] A. King and M. Bloor. Free-surface flow over a step. *Journal of Fluid Mechanics*, 182:193–208, 1987.

[22] D. Violeau and B. D. Rogers. Smoothed particle hydrodynamics (sph) for free-surface flows: past, present and future. *Journal of Hydraulic Research*, 54(1):1–26, 2016.

[23] A. Amer, O. Álvarez-Tuñón, H. İ. Uğurlu, J. L. F. Sejersen, Y. Brodskiy, and E. Kayacan. Unav-sim: A visually realistic underwater robotics simulator and synthetic data-generation framework. In *2023 21st International Conference on Advanced Robotics (ICAR)*, pages 570–576. IEEE, 2023.

[24] M. Jacinto, J. Pinto, J. Patrikar, J. Keller, R. Cunha, S. Scherer, and A. Pascoal. Pegasus simulator: An isaac sim framework for multiple aerial vehicles simulation. *arXiv preprint arXiv:2307.05263*, 2023.

[25] S. Shah, D. Dey, C. Lovett, and A. Kapoor. Airsim: High-fidelity visual and physical simulation for autonomous vehicles. In *Field and Service Robotics: Results of the 11th International Conference*, pages 621–635. Springer, 2018.

[26] P. G. O. Zwilgmeyer. Creating a synthetic underwater dataset for egomotion estimation and 3d reconstruction. Master's thesis, NTNU, 2021.

[27] 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*, pages 1–8. IEEE, 2016.

[28] S. K. Dhurandher, S. Misra, M. S. Obaidat, and S. Khairwal. Uwsim: A simulator for underwater sensor networks. *Simulation*, 84(7):327–338, 2008.

[29] 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)*, pages 1–8. IEEE, 2022.

[30] E. Potokar, S. Ashford, M. Kaess, and J. G. Mangelson. Holoocean: An underwater robotics simulator. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 3040–3046. IEEE, 2022.

[31] I. Lončar, J. Obradović, N. Kraševac, L. Mandić, I. Kvasić, F. Ferreira, V. Slošić, D. Nad, and N. Mišković. Marus-a marine robotics simulator. In *OCEANS 2022, Hampton Roads*, pages 1–7. IEEE, 2022.

[32] J. Wu, X. Lin, S. Negahdaripour, C. Fermüller, and Y. Aloimonos. Marvis: Motion & geometry aware real and virtual image segmentation. *arXiv preprint arXiv:2403.09850*, 2024.

[33] A. Palnitkar, R. Kapu, X. Lin, C. Liu, N. Karapetyan, and Y. Aloimonos. Chatsim: Underwater simulation with natural language prompting. In *OCEANS 2023-MTS/IEEE US Gulf Coast*, pages 1–7. IEEE, 2023.

[34] X. Lin, N. Jha, M. Joshi, N. Karapetyan, Y. Aloimonos, and M. Yu. Oystersim: Underwater simulation for enhancing oyster reef monitoring. In *OCEANS 2022, Hampton Roads*, pages 1–6. IEEE, 2022.

[35] H. Lee, S. Kim, and H. J. Kim. Control of an aerial manipulator using on-line parameter estimator for an unknown payload. In *2015 IEEE international conference on automation science and engineering (CASE)*, pages 316–321. IEEE, 2015.

[36] H. Lee and H. J. Kim. Estimation, control, and planning for autonomous aerial transportation. *IEEE Transactions on Industrial Electronics*, 64(4):3369–3379, 2016.

[37] D. Mellinger and V. Kumar. Minimum snap trajectory generation and control for quadrotors. In *2011 IEEE International Conference on Robotics and Automation*, pages 2520–2525, 2011. doi:10.1109/ICRA.2011.5980409.

[38] J. J. Monaghan. Smoothed particle hydrodynamics. *Annual review of astronomy and astrophysics*, 30:543–574, 1992.

[39] R. Xi, Z. Luo, D. D. Feng, Y. Zhang, X. Zhang, and T. Han. Survey on smoothed particle hydrodynamics and the particle systems. *IEEE Access*, 8:3087–3105, 2019.

[40] V. Zago, G. Bilotta, A. Cappello, R. Dalrymple, L. Fortuna, G. Ganci, A. Hérault, and C. Del Negro. Simulating complex fluids with smoothed particle hydrodynamics. *Annals of Geophysics*, 2017.

[41] M. Toma, R. Chan-Akeley, J. Arias, G. D. Kurgansky, and W. Mao. Fluid–structure interaction analyses of biological systems using smoothed-particle hydrodynamics. *Biology*, 10(3):185, 2021.

[42] O. Marquis, B. Tremblay, J.-F. Lemieux, and M. Islam. Smoothed particle hydrodynamics implementation of the standard viscous-plastic sea-ice model and validation in simple idealized experiments. *The Cryosphere Discussions*, 2022:1–33, 2022.

[43] R. Andersson and E. Tjernell. Comparison between smoothed-particle hydrodynamics and position based dynamics for real-time water simulation, 2023.

[44] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pages 1861–1870. PMLR, 2018.

[45] M. Savva, A. Kadian, O. Maksymets, Y. Zhao, E. Wijmans, B. Jain, J. Straub, J. Liu, V. Koltun, J. Malik, et al. Habitat: A platform for embodied ai research. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 9339–9347, 2019.

[46] A. Szot, A. Clegg, E. Undersander, E. Wijmans, Y. Zhao, J. Turner, N. Maestre, M. Mukadam, D. S. Chaplot, O. Maksymets, et al. Habitat 2.0: Training home assistants to rearrange their habitat. *Advances in neural information processing systems*, 34:251–266, 2021.

[47] C. Li, F. Xia, R. Martín-Martín, M. Lingelbach, S. Srivastava, B. Shen, K. Vainio, C. Gokmen, G. Dharan, T. Jain, et al. igibson 2.0: Object-centric simulation for robot learning of everyday household tasks. *arXiv preprint arXiv:2108.03272*, 2021.

[48] C. Gulino, J. Fu, W. Luo, G. Tucker, E. Bronstein, Y. Lu, J. Harb, X. Pan, Y. Wang, X. Chen, et al. Waymax: An accelerated, data-driven simulator for large-scale autonomous driving research. *Advances in Neural Information Processing Systems*, 36, 2024.

[49] X. Puig, E. Undersander, A. Szot, M. D. Cote, T.-Y. Yang, R. Partsey, R. Desai, A. Clegg, M. Hlavac, S. Y. Min, et al. Habitat 3.0: A co-habitat for humans, avatars, and robots. In *The Twelfth International Conference on Learning Representations*.

[50] O. S. Bhatti, M. Rizwan, P. S. Shiokolas, and B. Ali. Genetically optimized anfis-based pid controller design for posture-stabilization of self-balancing-robots under depleting battery conditions. *Journal of Control Engineering and Applied Informatics*, 21(4):22–33, 2019.

[51] T. Kanehira, M. L. McAllister, S. Draycott, T. Nakashima, D. M. Ingram, T. S. van den Bremer, and H. Mutsuda. The effects of smoothing length on the onset of wave breaking in smoothed particle hydrodynamics (sph) simulations of highly directionally spread waves. *Computational Particle Mechanics*, 9(5):1031–1047, 2022.

[52] N. Akinci, G. Akinci, and M. Teschner. Versatile surface tension and adhesion for sph fluids. *ACM Transactions on Graphics (TOG)*, 32(6):1–8, 2013.

[53] K. von Szadkowski and S. Reichel. Phobos: A tool for creating complex robot models. *Journal of Open Source Software*, 5(45):1326, 2020.

This appendix aims to address potential concerns and questions detailed in Appendix. A, provide further details on position-based dynamics in Appendix. B, elaborate on AAM dynamics and control in Appendix. **??**, and discuss additional experiments in Appendix. C.

In our supplemental video, we showcase several aerial-aquatic trajectories used to complete various tasks. Due to hardware constraints and the high cost of running liquid simulations, reinforcement learning in aerial-aquatic environments is notably slow. Consequently, we will add more demonstration videos to the anonymous Google Drive: `https://drive.google.com/drive/folders/1ghEOsFIZ1hNrTdV5EBSUGiNVyWthGjrN?usp=sharing`

Appendix A Potential Questions and Our Answers

In this section, we aim to address potential concerns and questions and hope to clear any doubts or uncertainties that may arise.

You mentioned that your SEALS is based on Isaac Sim, while the AAM control is based on Pegasus. Can you specify your unique contribution?

To the best of our knowledge, there currently exists no high-fidelity simulator capable of effectively modeling movement both underwater and in the air. Although position-based dynamics [19, 20, 18] incorporated in the powerful NVIDIA Isaac Sim framework[2] seems promising, their application in the development of a high-fidelity underwater robotics simulator in fluids of free space has not yet been explored. Additionally, adapting these dynamics to support motion across both aerial and aquatic mediums, including quadcopter dynamics, presents further challenges. Our initial attempt to create such a simulator required significant effort.

To determine the suitability of leveraging position-based dynamics, one of our major tasks was to integrate traditional rigid-body-based hydrodynamics, as used in the cutting-edge photorealistic simulator UNav-Sim, into the Isaac Sim framework (specifically AAM-SEALS) alongside position-based hydrodynamics. This integration is non-trivial and allows us to compare the two hydrodynamics models, providing valuable insights.

The application of the control design from [37] to AAM control is also not straightforward, as it is designed solely for aerial robotics to generate trajectories, without manipulators or underwater environments. Our AAM-SEALS adapted the control logic from Pegasus [24] by incorporating manipulator control. We specifically tailored our design for the morphology, kinematics, sensing, and control of our AAM, featuring a thinner arm, a three-finger gripper, and two RGB-Depth cameras (one looking ahead and one looking down) for aerial-aquatic manipulation tasks.

Our final contribution is a comprehensive evaluation of both the robot and the simulator, including applications in visual reinforcement learning (RL).

Is there a weakness in not including experiments on Physical AAMs?

We strongly believe that the development of AAM-SEALS already constitutes a significant workload, and adding research and development of the first physical AAM would far exceed the typical workload for a top conference paper. It is also worth noting that many other papers on photorealistic robotics simulators, such as [47, 48], do not include physical robot experiments.

What's the action space and reward function for the RL?

Please refer to Appendix. C for details.

Could you explain more regarding position-based hydrodynamics?

Please refer to the Appendix. B.

Where can I find the description of manipulator modeling in Sec 3.1 AAM Dynamics Modeling? Also, what does the expression $E_i += E_p$ mean in Sec. 3.3 Control Development?

[2] https://developer.nvidia.com/isaac/sim

Please refer to the Appendix. **??** for details.

So far, the demonstration of underwater manipulation by the AAM has been limited.

Please refer to Fig. 16 and our supplemental video for a detailed demonstration of the AAM grasping objects underwater.

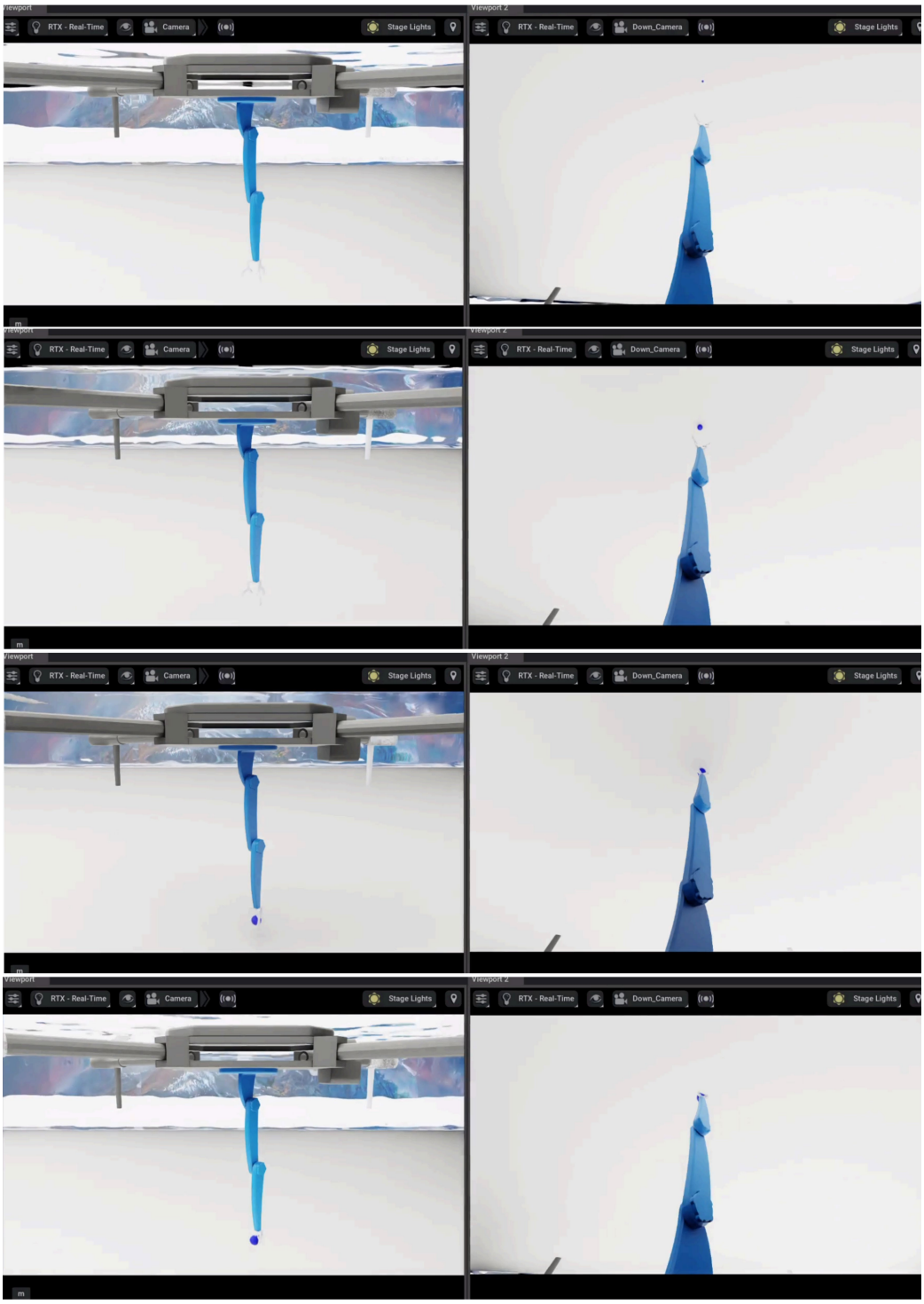

Figure 16: Video frames showcasing the object being grasped underwater by our AAM

How many trajectories did you collect for evaluating the position-tracking error?

We collected 10 trajectories and believe this is sufficient to demonstrate the position-tracking error between dynamics systems. With 10 trajectories we could capture a fairly representative sample of directions, magnitudes, and sequences of actions. In addition, most of the trajectories were evaluated at more than 700 time points, meaning the 10 collected trajectories hold a large amount of evaluative data. The results of all 10 trajectories follow the trends presented in Sec. 5.2 of the paper, as shown in Fig. 17.

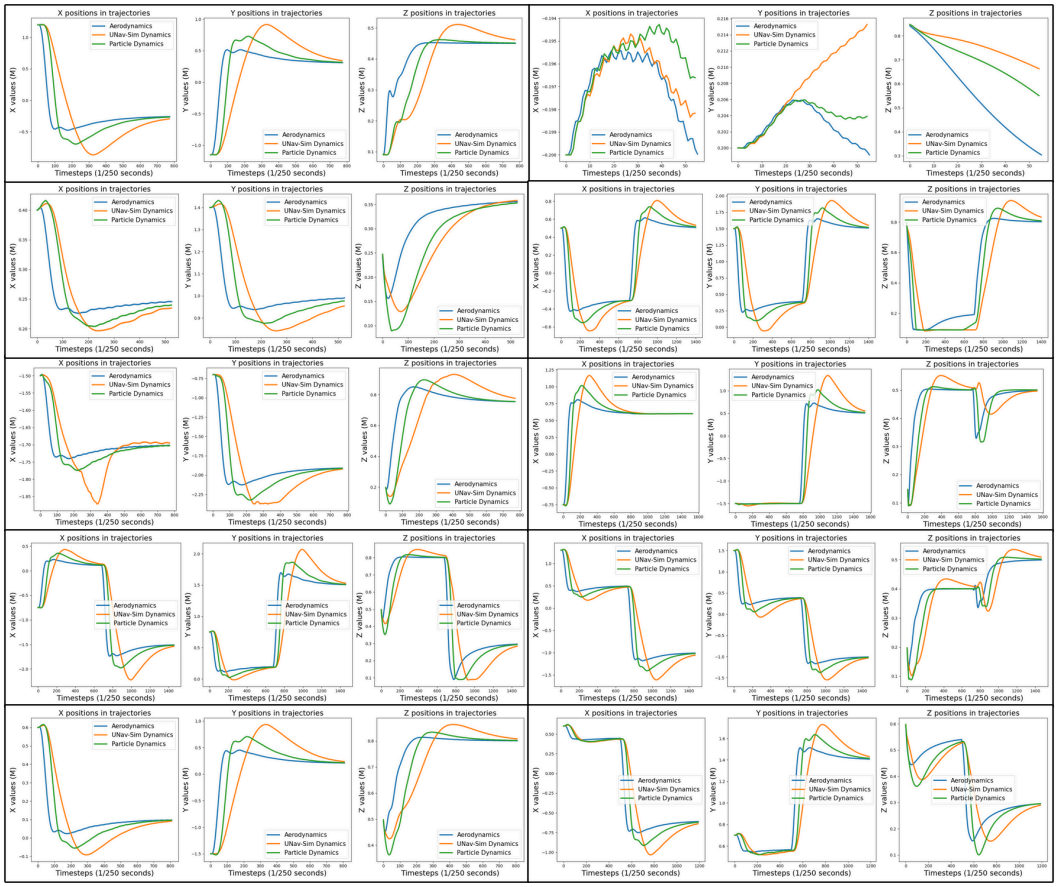

Figure 17: Dynamics trajectory comparisons across all 10 collected trajectories

Appendix B Preliminary Knowledge on Position-Based Dynamics

In the paper, the fluid simulation method uses the position-based dynamics (PBD) approach ([19, 20, 18]), which is closely related to the smoothed particle hydrodynamics (SPH) explained in [38, 51, 17, 40]. SPH is a well-known method that computes density and forces based on particle method for fluid simulation. However, SPH is sensitive to density fluctuations due to neighborhood deficiencies, and enforcing incompressibility is computationally expensive due to the unstructured nature of the model. SPH algorithms often become unstable if particles do not have enough neighbors for accurate density estimates. Typically, stability in SPH is maintained by taking sufficiently small time steps or using many particles, both of which increase computational costs.

In contrast, PBD improves upon these limitations by directly manipulating particle positions to satisfy physical constraints, specifically a density constraint given by:

$$C(x_1, ..., x_n) = \frac{\rho_i}{\rho_0} - 1 \leq 0 \tag{9}$$

where ρ_i is the density at particle i and ρ_0 is the rest density of the fluid. This ensures that particles maintain a proper distance from each other, effectively preventing clustering. PBD benefits from unconditionally stable time integration and robustness, making it popular with game developers and filmmakers. By addressing particle deficiencies at free surfaces and handling large density errors, PBD allows users to trade incompressibility for performance while remaining stable.

The PBD method also integrates additional effects such as cohesion and surface tension by adopting models such as those proposed by Akinci et al. (2013) [52]. For fluid-solid coupling, boundary particles are used to compute pressure forces between fluid and solid surfaces, ensuring accurate interactions. In the work [18], the density estimation for fluid particles includes contributions from both fluid and solid particles, represented as:

$$\rho_i = \sum_j m_j W(x_i - x_j, h) \tag{10}$$

where m_j is the mass of particle j, W is the smoothing kernel, h is width of the smoothing kernel W, and $x_i - x_j$ is the distance between particles i and j. This approach could then improved to a mass-weighted version of position-based dynamics as proposed in the unified position-based dynamics work [19]:

$$\rho_i = \sum_{fluid} m_j W(x_i - x_j, h) + s \sum_{solid} m_j W(x_i - x_j, h) \tag{11}$$

where a parameter s is introduced to account for the differing densities, which allows for the realistic simulation of buoyancy and sinking behaviors of objects with different densities.

Appendix C Additional Reinforcement Learning Experiments

In this section, we report the settings and results of reinforcement learning using pose states.

State Space: Our state space includes the positions and orientations of both the object to be grasped and our AAM. The state space also includes the velocities of our AAM along the x, y, and z directions.

Action Space: The action space in our framework is defined as a 3-tuple: [velocity_x, velocity_y, velocity_z], representing the AAM's body movement with respect to the world frame. It is assumed that once the AAM's gripper reaches a desired position, there exists an engineered policy to automatically close the fingers, grasp the object, and then ascend out of the water.

Termination Condition:

In our RL environments, termination conditions are designed to determine the success or failure of an episode. Success is achieved when the AAM's gripper reaches within a distance of 0.01 meters of the target object.

Reward Function:

The reward function is defined on the basis of the distance between the AAM and the object, adjusted with a height offset to ensure clearance for grasping. It categorizes distances into three regions: outer (distance greater than 1 meter), inner (distance between 1 meter and d_t), and success (distance less than d_t). Each region employs a different reward calculation to provide dense rewards instead of sparse ones, with an added exponential growth factor to amplify the rewards as the AAM approaches the object. The reward formulations for each region are as follows:

- **Outer Region:** $r = \exp(-d)$
- **Inner Region:** $r = \frac{1}{d}$

- **Success Region:** $r = 1000\frac{1}{d_t}$

Here, d denotes the Euclidean distance between the AAM and the object, and d_t is the distance threshold below which the distance is considered a success. During our training, d_t was set to 1×10^{-2} meters.

To deter erratic behavior, penalty rewards are implemented for the RL agent. If the AAM's velocity surpasses a defined threshold, it incurs a penalty of -5. Furthermore, if the AAM achieves the success condition but subsequently leaves the success region in a specified number of steps, it receives a penalty equivalent to $-1000\frac{1}{d_t}$. This penalty is intended to enforce stability and keep the AAM's gripper within the success region during operation.

C.1 Training Results

The reinforcement learning results are presented in Fig. 18, illustrating the mean of episode cumulative rewards (ep_rew_mean) over steps, the success rate over steps, and the mean of episode length (ep_len_mean) over steps.

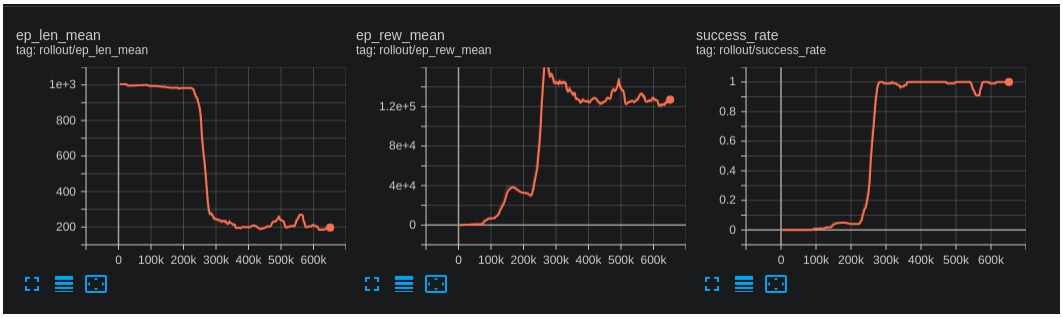

Figure 18: Reinforcement Learning Results

C.2 Reinforcement Learning Hyperparameters

In this section, we provide the values of the crucial hyperparameters listed in Table 1.

Parameter	Value
batch size	2048
total timesteps	10,000,000
Episode Length	1,000
distance threshold (d_t)	10^{-2}m
Isaac Sim physics simulation timestep (dt)	0.004
replay buffer size	1,000,000
learning rates for actor and critic	0.006
discount (γ)	0.99
exploration noise	0.1
minimal exploration noise	0
learning starts	100
number of hidden layers (all networks)	2
number of hidden units for layer 1 and 2	[256, 256]
nonlinearity	ReLU
seeds	0

Table 1: Reinforcement Learning Hyperparameters

Appendix D Modeling, Control, and Learning of Aquatic Animals

The crab mesh was created in Blender 3.3, where we used various mesh tools to sculpt the body, legs, and claws of the crab into a detailed and realistic model. Once the basic structure was complete, we activated the Phobos extension [53], a powerful tool for robotics modeling. With Phobos, we assigned joints to key parts of the crab, such as the leg bases and claw hinges, by selecting the corresponding mesh segments and establishing them as joints. We then linked these joints to simulate natural movements, ensuring that each leg and claw could articulate correctly. Fig. 19 Careful naming and organization of components within Phobos allowed for a clean and manageable hierarchy, which is essential for future robotics applications. Finally, we exported the entire setup, including joints and links, in URDF format using the Phobos export function, making the crab model ready for integration into the robotics simulation environment. To ensure everything worked as expected, we loaded the URDF into an Isaac Sim and fine-tuned the model, checking that the crab's movements were accurately represented and adjusting any discrepancies.

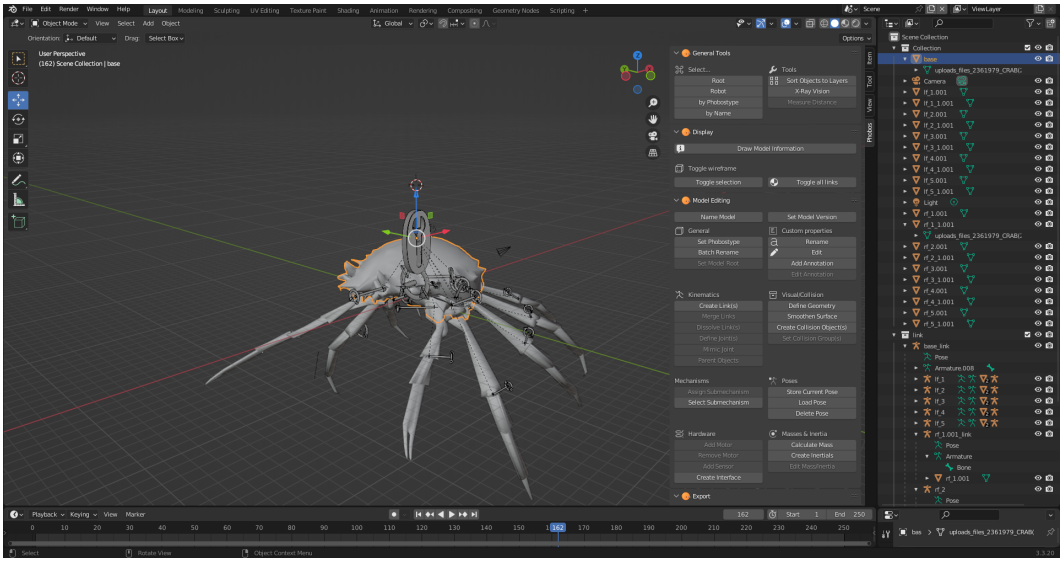

Figure 19: Crab Mesh in Blender

After creating the URDF and meshes of the crab, we then need to model it and assign controllers to joints. The crab model consists of 18 joints, resulting in a total of 18 degrees of freedom (DoF) for the body. Managing such a high-DoF agent can be challenging, so we employed a reinforcement learning (RL) based policy for control. The controller used for each joint is a position-based controller, defined by the following equation:

$$F = K_d(V_d - \text{current\_V}) + K_s(P_d - \text{current\_P})$$

Where:

- K_d is the damping coefficient,
- K_s is the stiffness coefficient,
- V_d is the desired velocity (typically set to 0),
- current_V is the current velocity of the joint motion,
- P_d is the desired position (angular position in radians),
- current_P is the current angular position of the joint.

The reward function used in the RL approach is similar to the one described in Appendix C. The objective is for the robot to reach a fixed target position, with the robot spawning from different starting positions each time. That said, future works could consider more advanced learning algorithms such as goal-conditioned reinforcement learning and even adversarial multi-agent reinforcement learning that can empower the crab with defensive strategies.

Appendix E Adding your Customized AAM

In this paper, we provide a high-level description how future researchers could create their own AAM and load into our SEALS. We will release a detailed tutorial online once the paper gets accepted.

To include a different robot model in our simulator, you will need a .usd (Universal Scene Description) file of the robot. The process involves several steps:

1. Designing the Robot Model:
   - Begin by designing the robot model in SolidWorks (a 3D CAD Design Software)
   - Create the individual parts and assemble them, ensuring all joints and kinematic properties are accurately defined
2. Generating the Mesh Files and .urdf File:
   - Create mesh files to represent the robot's physical structure visually and geometrically

     Note: These meshes provide a realistic appearance in the simulation and can be exported alongside the .urdf file
   - Export these meshes alongside the .urdf (Unified Robot Description Format) file

     Note: The .urdf file encapsulates the working joints, linkages, and their respective constraints
3. Importing into Isaac Sim:
   - Import the .urdf file into the Isaac Sim simulator
   - Convert the .urdf file into a .usd (Universal Scene Description) file

     Note: The .usd format is essential because it enables seamless integration and manipulation within the simulator, ensuring that all joints operate correctly and the robot's physical characteristics are preserved