ELEC330

Assignment 1 appendices

Turtle in water –Teri–Transformative, Efficient, Robust, Intelligent Underwater Bio-Robot

Date: ***October 20, 2024***

Abstract

This report describes the design and development of a biomimetic underwater robot inspired by sea turtles. The goal of the project is to create a versatile, low-cost, and reliable robot for underwater exploration and data collection. The robot uses a propulsion system similar to the flippers of sea turtles, enabling it to move flexibly in complex underwater environments, making it suitable for tasks such as exploring shipwrecks and archaeological sites, which are difficult for traditional underwater robots. The project utilizes Gazebo for simulation and ROS 2 for the control system, achieving efficient testing and development. The robot is equipped with four independently moving fins, providing 6 degrees of freedom. The mechanical structure of the robot is introduced in detail, along with design specifications. Finally, the report explains how to create a URDF file for simulating the robot's behavior in ROS 2 and Gazebo environments.

Contents

Report

Introduction

Due to the ocean's unpredictable and hazardous conditions, underwater exploration remains one of the most challenging and complex forms of exploration. The immense pressure, low visibility, and diverse terrain require innovative solutions for effective exploration. Biomimetic robots offer a promising approach, drawing inspiration from creatures that have naturally evolved over millions of years to thrive in underwater environments. By mimicking the movements and behaviors of creatures highly adapted to aquatic environments, the goal is to create a robot Teri (Transformative, Efficient, Robust, Intelligent Underwater Bio-Robot as shown in Fig 1) that can navigate agilely and efficiently in complex underwater environments, and perform tasks that traditional ROVs are not capable of, such as exploring shipwrecks or archaeological sites [1].

To facilitate the design, testing, and control of the robot, Gazebo and ROS (Robot Operating System) will play a crucial role. Gazebo will be used to create a realistic simulation environment for testing the robot's movements and behaviors, while ROS will provide the framework to integrate sensors, actuators, and control systems.

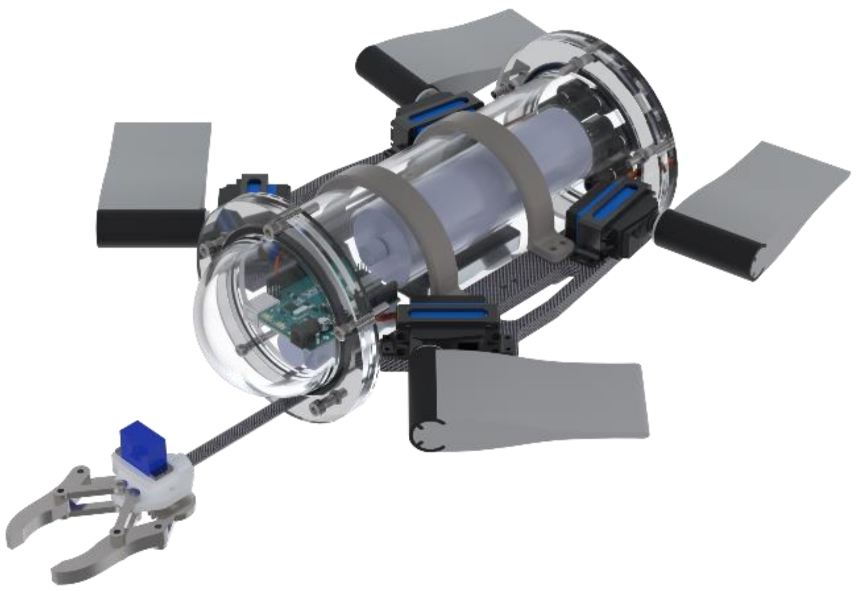


Fig 1. Bionic underwater robot - Teri

Aims

This project aims to create a robot with biomimetic inspiration in order to complete underwater exploration and data collection.
The aim of this project is to design and manufacture a biomimetic robot inspired by the natural movement of sea turtles, capable of underwater exploration and data collection. The project seeks to validate the feasibility of a versatile, low-cost, and reliable form of underwater exploration tool that can be deployed in various applications, including scientific research, fish farms, and marine archaeology [1].

Additionally, the development of this project requires the use of “Gazebo” for simulating underwater environments and “ROS 2” for developing the robot's control and communication systems. Project-based learning will enable developers to master these tools, allowing for more effective prototyping, simulation, and real-world deployment.

Background

For sensitive environments such as underwater archaeological sites, traditional ROVs or human divers are not well-suited for inspecting shipwrecks or enclosed areas. For human divers, entering a shipwreck is extremely dangerous, with high risks of getting lost or trapped. Additionally, the time a human diver can spend underwater is very limited, which means multiple dives or several divers are required, significantly increasing the cost of the mission.

Moreover, for companies or individuals with limited budgets, the cost of submersibles must be affordable. Some more experimental AUVs (Autonomous Underwater Vehicles) with hovering capabilities, like NESSIE [2], are capable of shipwreck exploration. They are relatively agile and carry imaging sonars that can help with obstacle avoidance in wrecks. However, inside shipwrecks, even with highly advanced positioning and mapping technologies, there is still a significant risk of losing the vehicle. Losing such robots would be too costly for exploration missions. Additionally, these vehicles are too large to effectively navigate through narrow corridors. Therefore, a smaller and more affordable solution is needed for exploration.

The robot designed in this project is inspired by turtles, animals that have thrived in marine environments through millions of years of evolution. Turtles use their long front flippers as paddles for propulsion, reaching speeds of up to 35 km/h, while their smaller rear flippers provide agility and steering capability. The streamlined shape of their shells provides excellent hydrodynamic performance, allowing them to move efficiently through the water [3]. These characteristics are essential to the biomimetic design of the robot.

The turtle has been used as the inspiration for other robots and these will be used for inspiration for the design of the project. One example is the U-CAT, which has demonstrated the potential of using biomimetic principles for underwater exploration [4]. U-CAT's fin-based propulsion system enables movement with six degrees of freedom which minimizes environmental disturbance and improves manoeuvrability in confined spaces. Unlike traditional ROV that rely on propellers, which stir up sediment thus reducing visibility, fin-driven robots like “Teri” offer quieter, more controlled movement, avoiding the risk of entanglement or harming surrounding marine life. This further emphasizes the use of fin propulsion is well-suited for sensitive applications like archaeological sites, exploration, and fish farming.

Prior to mechanical production, this project will utilize “Gazebo” and “ROS 2” to facilitate the development process. Gazebo allows the creation of accurate 3D simulations in which the robot's underwater behaviour can be tested, providing a safe, controlled environment to refine the design before real-world deployment. By using Gazebo to refine the design, the cost of the project will be reduced as the number of iterations required of the physical robot will be reduced. ROS 2, will be used as the operating system for the robot, will allow for the integration of the control systems, sensors, and actuators, ensuring communication and control across the robot's components. Thus through the use of Gazebo and ROS 2, the project can optimize prototyping, test control algorithms, and enhance the robot's movement and data collection capabilities in simulated underwater conditions before transitioning to real-world testing.

MoSCoW Specification

To ensure that the project meets the aims, a specification has been created to provide clarity during the design and creation as well as to develop the precise requirements that will make the project successful. To develop this specification the MoSCoW (Must have, Should have, Could have, Won't have) model was followed as well as creating a technical specification.

Must have

- Waterproofing: The robot needs to travel underwater, so its sealed enclosure must be waterproof to prevent short circuits. The waterproof rating must be at least IP68.
- Degrees of Freedom: For a robot that moves in three-dimensional space, it must have 6 degrees of freedom: translation along the x, y, and z axes, and rotational movement of the head, roll and pitch (as shown in Fig 2).

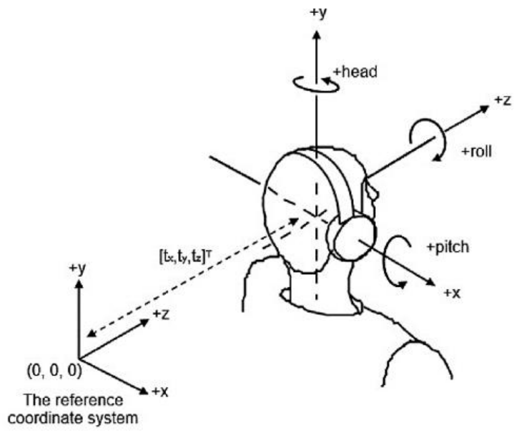


Fig 2. The reference coordinate system

- Power Supply: The robot must have a high-performance battery capable of storing enough energy to support all electronic components and actuators for the duration of the mission (at least 1 hour). This is critical for underwater operations where a power failure could lead to mission failure.
- Remote Control: The robot needs to receive commands and execute specific tasks, so it must be capable of remote control. To achieve this, it is necessary to ensure that the remote control system is compatible with underwater environments, potentially utilizing tethered communication or acoustic communication technology, as traditional radio signals are ineffective underwater.
- Self-Stabilization: This robot should have a self-stabilizing function. When designing the structure, the center of gravity should be kept as low as possible, allowing the robot to maintain balance in calm water even without the intervention of attitude sensors or actuators. This feature will provide the robot with basic posture maintenance capabilities (passive posture control), laying the foundation for the development of active posture control. This will enable the robot to collect clearer images and make it easier for the operator to complete tasks.
- Biomimetic Design: This robot is inspired by a turtle, so it will need to be designed based on the features of a turtle that have evolved over thousands of years, contributing to its biomimetic nature.

Should have

- Posture and Obstacle Avoidance: Due to the complexity of water environments, human operation may lead to mistakes. To maintain the robot’s posture and prevent collisions, the robot requires sensors such as an IMU (Inertial Measurement Unit) and Time of Flight sensors, giving it the ability to observe its environment, collect data, avoid obstacles, and assist the operator in completing tasks.

- Fail-safe Mechanism: In case of signal loss or the battery on the robot runs low, the robot must be able to recognize the situation and take emergency surfacing measures to avoid unnecessary damage.
- Modular Design: The robot should have a modular design, with its components designed for easy access and replacement, allowing for quick repairs or upgrades without the need to disassemble the entire system.
- Mechanical Claw: The robot should have a mechanical claw (or gripper) to perform tasks such as picking up objects or interacting with the underwater environment. This will enhance the robot’s ability to perform a variety of exploration and manipulation tasks.

Could have

- Corrosion Resistance: The project should be made of corrosion-resistant material. This is important because the ocean is filled with saltwater which is highly corrosive. It should remain unaffected after numerous long ocean exploration missions.
- Real-time Video Synchronization: Using video transmission equipment, the robot can remotely share the captured footage in real-time. Real-time video allows the operator to directly see the environment in which the ROV is located, accurately judge its position and surroundings, and precisely control its movements. This capability enables the robot to replace humans in exploring dangerous waters, greatly expanding the robot's range of activity and application scenarios.
- Extended Sensor Array: In addition to the IMU and Time of Flight sensors, consider adding water temperature sensors, pressure sensors, or salinity sensors to expand the robot’s data-gathering capabilities.

Won’t have

- Hydrodynamics: If there is time, the project could aim to develop a streamlined body by analyzing the hydrodynamics of the shape of the project. This will allow for the fastest movement in the water and a greater area to be covered in one mission.
- 3D Terrain Modeling: The robot's ability to develop 3D terrain modeling can also be explored, which is a key feature for underwater exploration and data collection. Acquiring the structural characteristics of the seabed or underwater objects is crucial for navigation.
- Autonomous Navigation: Implementing basic autonomous navigation algorithms would enable the robot to avoid obstacles and follow pre-programmed paths without human intervention.
- Depth Limitation: The robot will not operate at depths exceeding 20 meters. This is because, at greater depths, the impact of pressure on objects increases significantly, and for this project, it is not possible to find off-the-shelf actuators and sealed enclosures that can withstand pressures beyond this depth. The goal of this project is to validate the feasibility of the turtle-inspired biomimetic robot, so to control costs, it will be designed to operate only at a maximum depth of 20 meters.
- Wireless Control: Due to the limitations of radio signals in underwater environments, the robot in this project will not use wireless control (tethered communication will be used). In future developments, acoustic communication technology may be explored to ensure reliable control.

Requirement specifications.

Specification	Justification	Test
Maximum Payload - 100g	The robot needs to be able to carry a certain amount of weight in a particular task	Test whether the robot can still move normally by attaching additional weights to it.
Body weight - 1.5kg	The robot needs to be lightweight enough so that the motors can propel the body efficiently.	Weigh using scales
Max speed – 30cm/s	Enough speed to do underwater exploration.	Find the time taken for the robot to swim for a fixed distance using the formula: $v = \frac{s}{t}$ to calculate the average speed.
Max Depth - 20m	The robot must be pressure-resistant enough to safely perform tasks underwater.	Measure depth swam down with a ruler
Operation time - 1 hours	Sufficient battery life allows more time and a wider exploration radius to complete tasks.	Using a stopwatch to time the duration of the operation
Drive Configuration - 4 finns	Mimic the movement patterns of a turtle.	Visual confirmation
Max Size - 480 x 420 x 140mm	The robot's size must not be too large to facilitate testing and reduce costs.	Measure with ruler
Navigation and obstacle avoidance - Time of Flight sensor	Traditional sonar detectors may fail when obstructed by the robot's external enclosure.	Conduct tests to evaluate the obstacle avoidance performance.
Waterproofing - IP68	For robots that need to operate in water, its waterproof rating must be IP68.	After submerging it in 1 meter of water for one hour, check for water ingress and ensure its functionality remains intact.

Robot design.

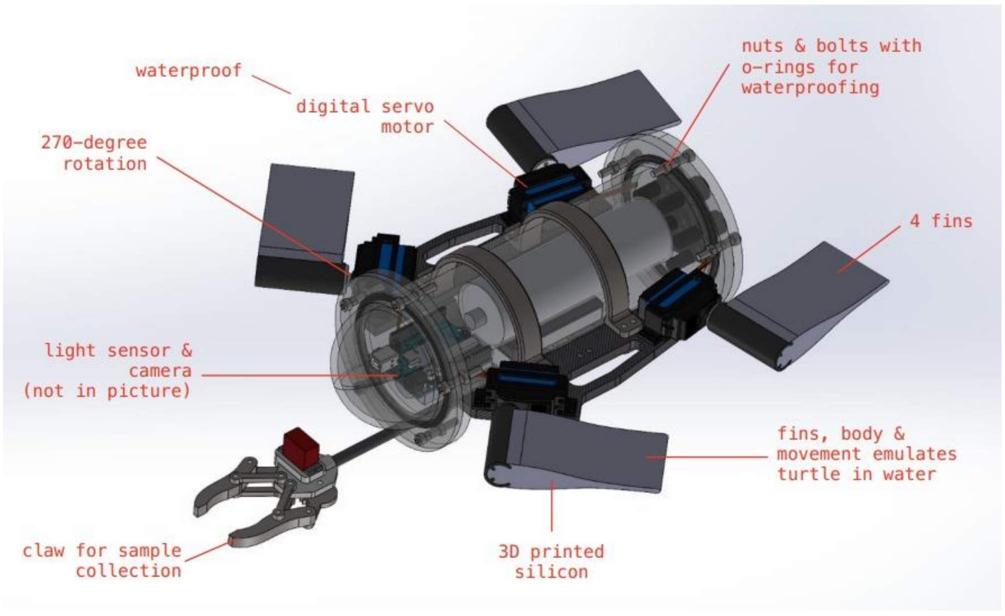

waterproof
digital servo motor
nuts & bolts with o-rings for waterproofing
270-degree rotation
4 fins
light sensor & camera (not in picture)
fins, body & movement emulates turtle in water
claw for sample collection
3D printed silicon

Fig 3. External structure

- Propulsion Method: The biomimetic robot, Teri, was designed according to the above requirements, as shown in Fig 3. Teri uses four fins for propulsion. The fins are connected to the body via the tail joints, and each fin can independently oscillate around an axis (side-to-side motion). The four fins of Teri are positioned at specific angles, ensuring that the thrust vectors of the fins are not collinear. This design enables the robot to control movement in six degrees of freedom.
- Center of mass: To increase stability and simplify control, we prevent unnecessary pitching and rolling by positioning the center of mass below the center of buoyancy.
- Fin Design: The propulsion force is generated by the fins' oscillating motion, producing a reactive force from the water (perpendicular to the surface of the fins). The radial component of this force relative to the axis generates the propulsion. To achieve higher propulsion efficiency, the fins are made of flexible materials such as silicone and TPU. During the oscillation, due to water resistance, the fins bend in the opposite direction of rotation, increasing the angle relative to the radial axis, which enhances the propulsion force (as shown in Fig 4).

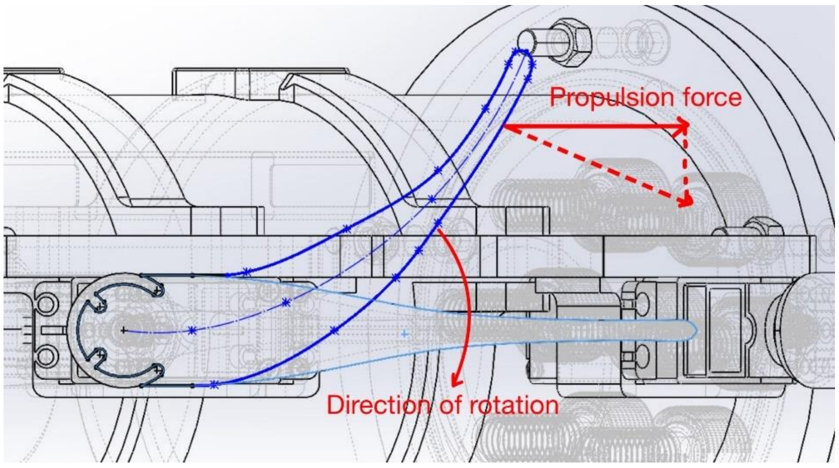


Fig 4. Bend in the opposite direction of rotation

The fins, made from flexible materials, act as an equivalent of infinitely small segments, and they can be 3D-printed for easy customization and adjustment. However, due to the use of flexible materials, the fins are prone to unwanted deformation under the influence of water currents when lacking proper constraints. This can weaken or impair the control performance. For example, when the motor rotates, the part of the fin near the motor moves accordingly, but the section farther from the motor may twist in the opposite direction or deform axially during ascent or descent. To prevent such occurrences, the fins will be mounted on a fin holder that connects to the motor. The fin holder, made of rigid material, provides better constraint for the fins. Additionally, this design allows for easy replacement and adjustment of the fins.

- Actuators: A mounting slot for an actuator (such as a mechanical claw in Fig 3) is reserved on the bottom of the robot. With different actuators, the robot can perform various tasks, making it versatile for different scenarios.

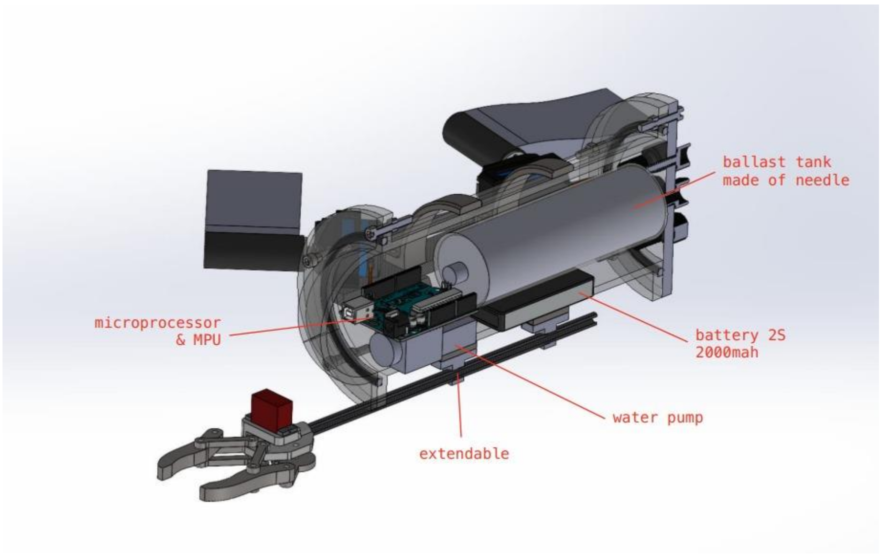


Fig 5. Internal structure

- Ballast Tank: As shown in the robot’s internal structure diagram (Fig 5), the sealed enclosure contains a buoyancy control piston (ballast tank). By pumping water in or out, the vehicle can adjust its weight to maintain neutral buoyancy. The buoyancy control system helps conserve energy, as the vehicle does not need to constantly compensate for non-neutral buoyancy using its fins. This also saves significant energy during diving and surfacing operations. Additionally, the ballast tank helps to increase the robot's payload. When the mechanical claw grabs objects or carries extra weight, the ballast system enhances maneuverability. Furthermore, the ballast tank functions as the robot’s fail-safe mechanism; in case of signal loss or low battery, the robot will ascend by draining the ballast tank, avoiding unnecessary damage.
- Materials: Due to the difficulty of manufacturing a waterproof sealed enclosure and the need to accommodate sensors such as ToF and cameras in the future, the robot's enclosure will be made from a combination of off-the-shelf acrylic tubes and acrylic hemispheres to ensure transparency. Additionally, the enclosure will be secured using custom 3D-printed tube clamps that attach the enclosure to the robot's side plates (carbon fiber or fiberglass), which serve as the mounting points for the digital servos.

Discussion and reflection design and model.

Advantage

Compared to conventionally propeller-driven ROVs, Teri offers the following advantages:

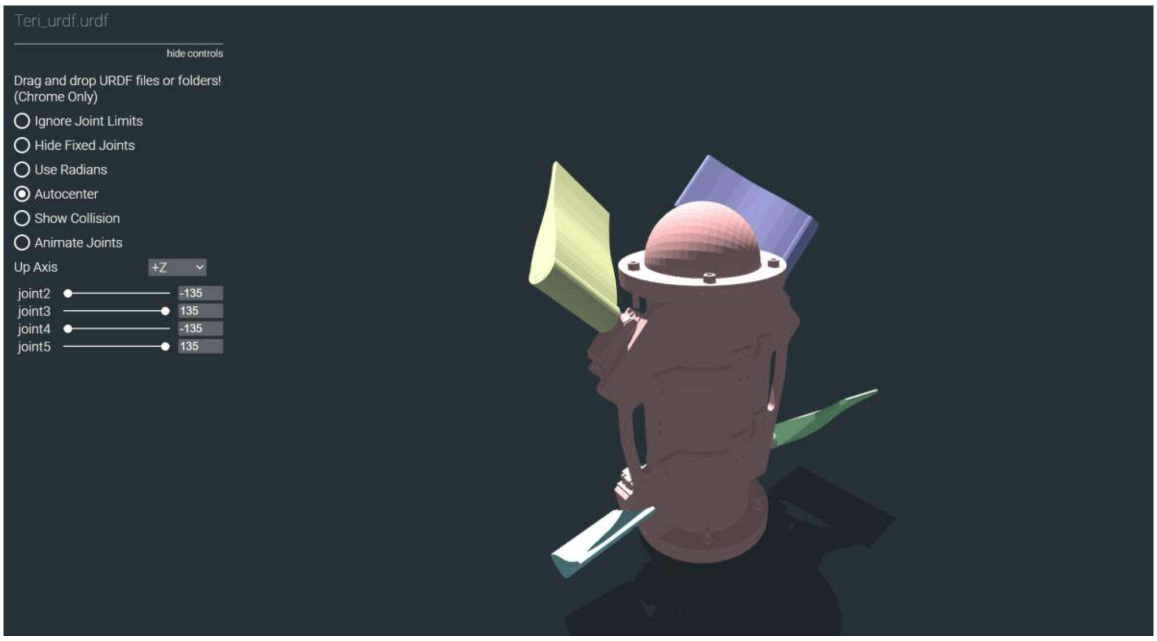
- Tangle Avoidance: Propellers can easily get tangled with surrounding ropes or plants. However, fins do not continuously rotate in one direction, thus avoiding this issue.
- Fewer Actuators and Lower Cost: Teri can achieve 6 degrees of freedom (DOF) with only 4 motors, reducing the number of actuators and lowering overall cost.
- Enhanced Safety: Unlike propellers, fins are safer for marine life and divers around the robot. This makes Teri suitable as a dive companion and usable in environments like fish farms without causing harm.
- Clearer Video Recording: The fin-driven propulsion does not generate strong water currents, so when the robot operates near the seabed, it will not stir up sediments. This ensures clearer video footage, which is beneficial for later analysis
- Fault Redundancy: Even if 2 motors fail, the remaining 2 fins can still ensure normal operation of the robot.

Limitation

- Due to hardware limitations, the digital servo cannot achieve continuous rotation, and each fin has a range of only 270 degrees, which imposes limitations on attitude adjustment.
- Additionally, due to a lack of understanding of acoustic communication and the absence of readily available solutions on the market, the robot could only use tethered control, which limits its exploration capabilities in terms of distance and complex terrain.

Assessment

- The design that has been created has incorporated all the above mentioned technical requirements as well as using the MoSCoW specification points for inspiration.All the technical specification points have been met. ‘Teri’ has been designed to be able to still function effectively with a 100g payload, collected by the claw, through the ballast which allows for stability and buoyancy control. Through the material and component choice the design will not exceed 1.5kg. The 4 fin design with servo motors ensure that the drive configuration and speed specification points are met. Furthermore the design is not greater than the maximum size outlined in the specification. The battery chosen ensures that the AUV is able to complete missions of up to one hour. Thus through considered design all the technical specification points have been met.
- In further iterations there is still the possibility of improving the hydrodynamics of the design, currently the design is more focused on functionality rather than higher performance aspects. This can be done by adding a shell to the design to further mimic the shape of a turtle and then testing this design using hydrodynamic simulation software. Additionally there can be further investigation into the possibility of the introduction of a greater sensor array and how these could be incorporated to the current design.
- After interference detection in SolidWorks, it was proven that all parts can be assembled.
- In the URDF viewer, the URDF file was tested, proving that the joint relationships are correct and the rotation angles are properly limited (shown in Fig 6).


Teri_urdf.urdf

hide controls

Drag and drop URDF files or folders!
(Chrome Only)

Ignore Joint Limits
Hide Fixed Joints
Use Radians
[✓ Autocenter]
Show Collision
Animate Joints

Up Axis +Z

```
joint2  -135
joint3   135
joint4  -135
joint5   135
```

Fig 6. Joints limitation

Obtain URDF file

Since the software used for design and modeling is SolidWorks, in order to conveniently and intuitively obtain URDF files, it is necessary to use an open-source plugin available on GitHub called "SolidWorks to URDF Exporter"[5] during this process.

1. First, the robot needs to be divided into subassemblies in SolidWorks, each consisting of multiple fixed parts. For Teri, it can be split into five subassemblies that will move relative to each other during operation, as shown in Fig 7.

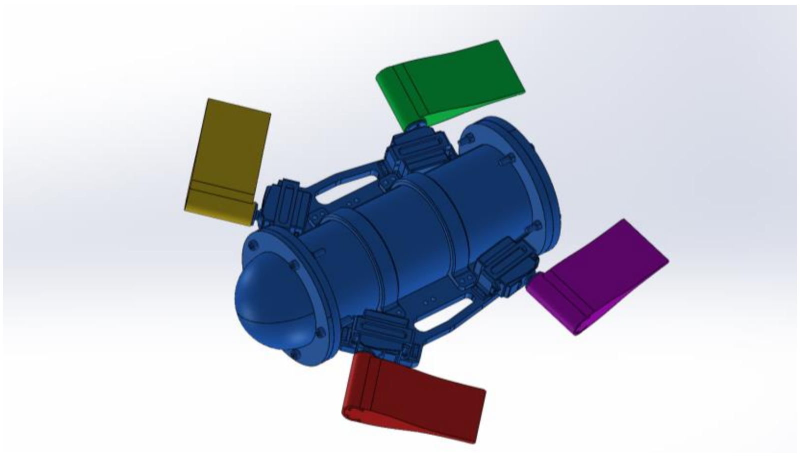


Fig 7. Five subassemblies

2. In URDF (Unified Robot Description Format), joints and links define the structure and movement of the robot, and they are closely related to the TF (Transform Frames) system. Each link represents a rigid body part of the robot, corresponding to a coordinate frame in the TF system. Joints define the motion relationship between two links, specifying the relative motion type (e.g., rotation or translation) between the parent and child links. In the TF system, joints generate the transformation from the parent link to the child link, dynamically updating the relative position and orientation of different parts of the robot.
   Therefore, to generate a correct URDF file, it is necessary to define the coordinate frames for the five subassemblies in SolidWorks (Fig. 8). During this process, a series of reference points,

axes, and planes need to be established to define the coordinate frames (Fig. 9).

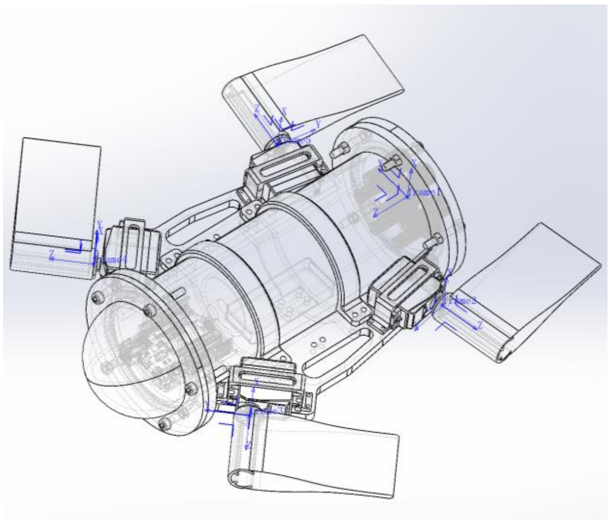


Fig 8. Coordinate frames

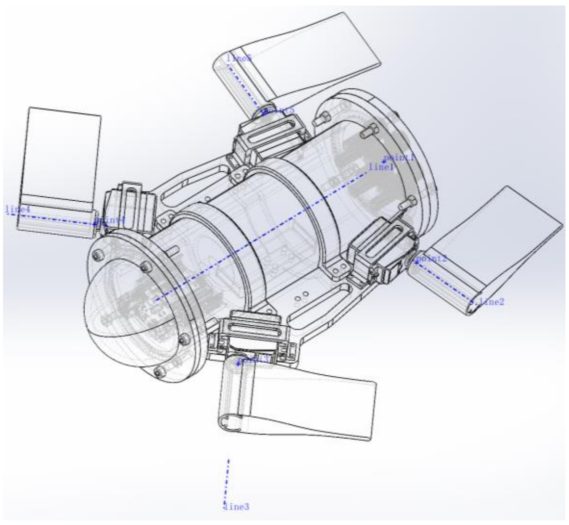
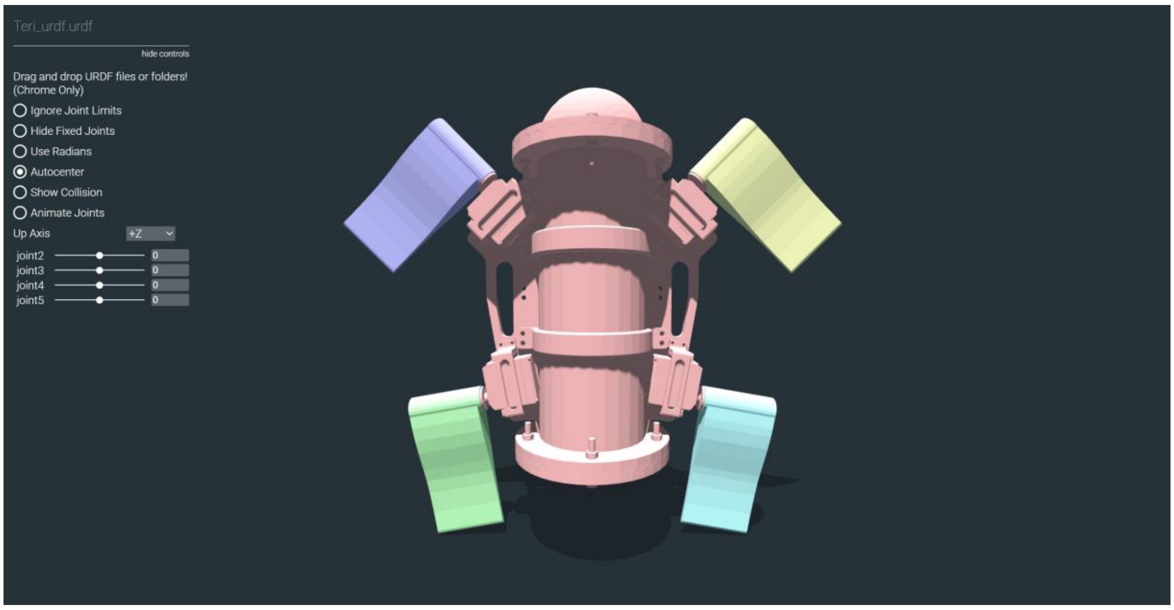


Fig 9. Reference points, axes

3. Finally, in the plugin, parameters such as the part's mass, color, joint types, and movement Limits (prevent unrealistic movements.) are set. Once these parameters are defined and exported, a fully defined URDF file is generated. However, it is important to note that the URDF exported from SolidWorks is in ROS format and needs to be converted to the ROS 2 version using the sw2urdf_ros2 tool available on GitHub [6].The file can be checked for correctness using an online URDF viewer [7] (Fig. 10).


Teri_urdf.urdf

hide controls

Drag and drop URDF files or folders! (Chrome Only)

- Ignore Joint Limits
- Hide Fixed Joints
- Use Radians
- [✓ Autocenter]
- Show Collision
- Animate Joints

Up Axis +Z

joint2 0
joint3 0
joint4 0
joint5 0

Fig 10. Teri in online URDF viewer

Launch the URDF file in Gazebo

After designing and generating the URDF file for the biomimetic robot Teri, the next step is to load the robot model into Gazebo. The URDF file and other necessary resources need to be organized into a folder and placed in the src directory of the ROS 2 workspace. The folder includes the following files:

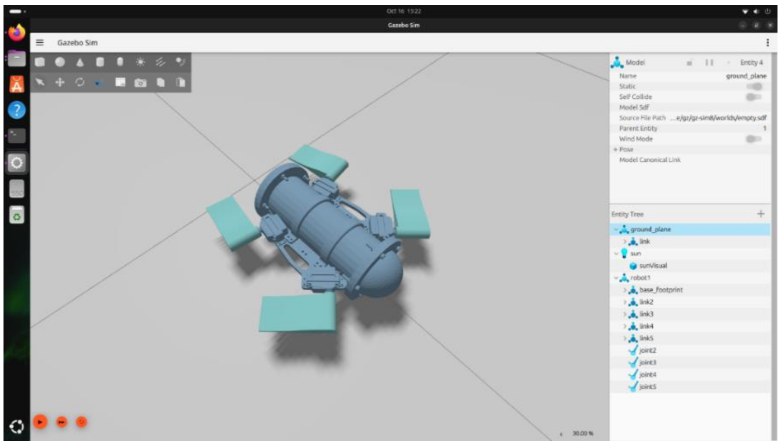
- launch folder (shown in the Documented code section): This folder contains ROS 2 launch files. These files define how to launch the entire system, including the Gazebo simulation, the

robot state publisher, etc. Running these files allows the entire system to be initiated.
In this ROS 2 launch file, the robot_state_publisher node is responsible for publishing the robot's URDF description, joint states, and transformation data (robot_description, joint_states, and tf topics), enabling other nodes to subscribe to and use this information. The ros_gz_bridge node acts as a bridge, synchronizing joint states and velocity commands from the Gazebo simulation with ROS 2 topics (joint_states and cmd_vel), allowing data sharing and communication between ROS 2 and Gazebo to control and provide feedback on the robot's status in the simulation environment.

- meshes folder: This folder contains 3D mesh files of the robot model. Gazebo uses these mesh files to visualize the robot and its components, representing the robot's visual and collision models.
- resource folder: Stores ROS 2 package identification files, which help ROS 2 locate and utilize the package.
- Teri_bot folder: Implements Python code related to the robot, including logic for controlling the robot and publishing/subscribing to ROS 2 messages.
- urdf folder: This folder contains URDF files that define the robot's physical structure and joints. The URDF files describe each part of the robot, the connections between the joints, physical properties (such as mass and friction), and the positions of sensors and actuators.
- package file: This is the metadata file for the ROS 2 package. It contains information such as the package's name, version, maintainer, and dependencies. ROS uses this file to manage the package and ensure proper installation and resolution of dependencies.
- setup.py (shown in the Documented code section)and setup.cfg files: These files define the steps for building and installing the Python package. Using these files, colcon build can correctly install the package into the workspace. The files specify the package dependencies, file paths to be installed, and any entry points.

Once these files are properly configured, the following commands can be used to load the URDF file into Gazebo:

- colcon build
- source install/setup.bash
- ros2 launch Teri_bot launch.py



Name ground_plane
Static
Self Collide
Model Sdf
Source File Path ...e/gz/gz-sim8/worlds/empty.sdf
Parent Entity 1
Wind Mode
Pose
Model Canonical Link

ground_plane
link
sun
sunVisual
robot1
base_footprint
link2
link3
link4
link5
joint2
joint3
joint4
joint5

Fig 11. Teri in Gazebo

As shown in the Fig 11, after executing the command, the Gazebo simulation environment was generated, and the Teri robot was successfully launched. The structure of the robot is correctly displayed within the simulation environment. It was observed that the fins of the Teri robot sag under the influence of gravity, verifying the correctness of the URDF file and demonstrating the robot's realistic physical interaction in the simulation environment.

Conclusion

This first assignment serves as the foundation of the biomimetic robot. It focuses on designing the robot through CAD software and simulating the designed robot in Gazebo. Since this robot is inspired by a turtle, it needs to be designed to have fins for propulsion, be waterproof, and many more. To achieve this, we utilize SolidWorks to model the robot and convert it into a URDF file. This file is used to define the robot's structural components, such as nodes and joints. The launch file isl then successfully spawn the robot in an empty Gazebo environment.

Reference

[1] Salumae, T. et al. (2014) 'Design principle of a biomimetic underwater robot U-CAT', 2014 Oceans - St. John's, 129, pp. 1–5. doi:10.1109/oceans.2014.7003126. Available at: https://ocean.si.edu/ocean-life/reptiles/sea-turtles

[2] Cartwright, J. et al. (2008) Nessie III autonomous underwater vehicle for sauce 2008, Nessie III Autonomous Underwater Vehicle for SAUCE 2008. Available at: https://www.researchgate.net/publication/239592677_Nessie_III_Autonomous_Underwater_Vehicle_for_SAUCE_2008.

[3] Larisa Bennett Reviewed by Connie Y. Kot (Duke University Marine Laboratory) (2023) Sea turtles, Smithsonian Ocean. Available at: https://ocean.si.edu/ocean-life/reptiles/sea-turtles .

[4] Salumae, T., Chemori, A. and Kruusmaa, M. (2019) 'Motion control of a hovering biomimetic four-fin underwater robot', IEEE Journal of Oceanic Engineering, 44(1), pp. 54–71. doi:10.1109/joe.2017.2774318. Available at: https://www.researchgate.net/publication/321814038_Motion_Control_of_a_Hovering_Biomimetic_Four-Fin_Underwater_Robot

[5] solidworks_urdf_exporter*: This is a tool that* export URDF file in SolidWorks. Available at: https://github.com/ros/solidworks_urdf_exporter/releases

[6] xiaoming-sun6 *Xiaoming-Sun6/sw2urdf_ros2: This is a tool that can be used in to change Ros URDF files to ROS2*, *GitHub*. Available at: https://github.com/xiaoming-sun6/sw2urdf_ros2

[7] URDF viewer example. Available at: https://gkjohnson.github.io/urdf-loaders/javascript/example/bundle/

Documented code

Launch.py

```
import os
from ament_index_python.packages import get_package_share_directory
from launch import LaunchDescription
from launch.substitutions import LaunchConfiguration, Command
from launch.actions import DeclareLaunchArgument, SetEnvironmentVariable
from launch_ros.actions import Node
from launch_ros.parameter_descriptions import ParameterValue
from launch.actions import IncludeLaunchDescription
from launch.launch_description_sources import PythonLaunchDescriptionSource

# Function to generate the launch description
def generate_launch_description():

    # Check if we're told to use simulation time; use_sim_time is a launch
argument
    use_sim_time = LaunchConfiguration('use_sim_time')

    # Get the path to the Teri_bot package share directory
    package_share_dir = get_package_share_directory('Teri_bot')

    # Specify the relative path to the URDF file
    path_to_urdf = os.path.join(package_share_dir, 'urdf', 'Teri_urdf.urdf')

    # Set the resource path to the workspace source directory
    # resource_path = os.path.join(os.path.expanduser('~'), 'project_ws',
'src')
    resource_path = get_package_share_directory('Teri_bot')
    set_gz_sim_resource_path = SetEnvironmentVariable(
        name='GZ_SIM_RESOURCE_PATH',
        value=resource_path + '/../'
    )

    # Create a robot_state_publisher node to publish the robot's state,
especially joint states and TF information
    node_robot_state_publisher = Node(
        package='robot_state_publisher',  # Use the robot_state_publisher
package
        executable='robot_state_publisher',  # The executable name
        name='robot_state_publisher',  # The node name
        output='screen',  # Output will be printed to the screen
        parameters=[{
            'robot_description': ParameterValue(Command(['xacro ',
str(path_to_urdf)]), value_type=str)  # Generate the robot description from
the URDF file
        }]
    )

    # Launch Gazebo simulator with an empty world
    gz_sim = IncludeLaunchDescription(
        PythonLaunchDescriptionSource(
            os.path.join(
                get_package_share_directory("ros_gz_sim"),  # Get the path
to the ros_gz_sim package
                "launch",  # Launch folder
                "gz_sim.launch.py"  # The Gazebo launch file
            )
        ),
        launch_arguments={"gz_args": "-r -v 4 empty.sdf"}.items(),  #
Arguments to pass to Gazebo: load empty.sdf with verbose logging
    )

    # Spawn the robot in Gazebo using the create service equivalent in
ros_gz_sim
    spawn_entity = Node(
        package="ros_gz_sim",  # Use the ros_gz_sim package
        executable="create",  # The 'create' executable to spawn models in
Gazebo
        arguments=[
            "-name", "robot1",  # The name of the robot in the Gazebo
simulation
            "-file", path_to_urdf,  # Path to the URDF file that defines the
robot
            "-x", "0", "-y", "0", "-z", "1.4"  # Initial position of the
robot in the Gazebo world (x, y, z)
        ],
        output="screen",  # Print the output to the screen
    )

    # Add the ros_gz_bridge node to bridge joint states and commands between
ROS 2 and Gazebo
    bridge = Node(
        package='ros_gz_bridge',  # Use the ros_gz_bridge package
        executable='parameter_bridge',  # Run the parameter bridge
executable
        name='ros_gz_bridge',  # Name of the bridge node
        arguments=[
            # Bridge the /joint_states topic from ROS 2
(sensor_msgs/JointState) to Gazebo (gz.msgs.Model)
            '/joint_states@sensor_msgs/msg/JointState[gz.msgs.Model',

            # Bridge velocity commands (if controlling with velocity)
between ROS 2 and Gazebo
            '/cmd_vel@geometry_msgs/msg/Twist[gz.msgs.Twist'
        ],
        output='screen'  # Print the output to the screen
    )

    # Return the full launch description
    return LaunchDescription([
        DeclareLaunchArgument(
            'use_sim_time',  # Declare the use_sim_time argument
            default_value='false',  # Default to false (no simulation time)
            description='Use simulation time if true'  # Description of the
argument
        ),

        # Set the environment variable for GZ_SIM_RESOURCE_PATH
        set_gz_sim_resource_path,

        # Start the joint state publisher, robot state publisher, Gazebo
simulation, and spawn the robot
        node_robot_state_publisher,
        gz_sim,
        spawn_entity,
        bridge  # Add the ros_gz_bridge node
    ])
```

setup.py

```
from setuptools import setup  # Import setuptools to configure and install
the Python package
from glob import glob  # Import glob to match file patterns

# Define the package name
package_name = 'Teri_bot'

# Call the setup function to configure the package metadata and install
information
setup(
    name=package_name,  # Name of the package
    version='0.0.0',  # Version of the package
    packages=[package_name],  # Modules or sub-packages included in the
package, typically the directory name
    data_files=[  # Used to specify files to be copied during installation
and their destination
        ('share/ament_index/resource_index/packages',  # ROS 2 resource
index directory
            ['resource/' + package_name]),  # Resource file for the package
(e.g., marker file), installed to the resource index
        ('share/' + package_name, ['package.xml']),  # Copy the package.xml
file to the share/package_name directory
        ('share/' + package_name + '/launch/', glob('launch/*.py')),  # Copy
all Python launch files from the launch folder to share/package_name/launch/
        ('share/' + package_name + '/urdf/', glob('urdf/*')),  # Copy all
files from the urdf folder to share/package_name/urdf/
        ('share/' + package_name + '/rviz/', glob('rviz/*')),  # Copy all
files from the rviz folder to share/package_name/rviz/
        ('share/' + package_name + '/meshes/collision/',
glob('meshes/collision/*')),  # Copy all mesh files from the collision
folder to share/package_name/meshes/collision/
        ('share/' + package_name + '/meshes/visual/',
glob('meshes/visual/*')),  # Copy all mesh files from the visual folder to
share/package_name/meshes/visual/
    ],
    install_requires=['setuptools'],  # Dependencies required to install
this package
    zip_safe=True,  # If True, the package can be distributed safely as a
zipped archive
    maintainer='ros-industrial',  # Name of the package maintainer
    maintainer_email='TODO:',  # Email of the package maintainer (should be
updated with a valid email)
    description='TODO: Package description',  # A brief description of the
package (should be updated with the actual description)
    license='TODO: License declaration',  # The type of license for the
package (should be updated with the appropriate license)
    tests_require=['pytest'],  # Dependencies needed for running tests,
typically a testing tool like pytest
    entry_points={  # Define console script entry points for creating
executable command-line tools
        'console_scripts': [
            # Add executable scripts here if needed
        ],
    },
)
```