Introduction

This project aimed to use undulatory locomotion produced by stingrays to produce a design that can swim towards a submerged neutrally buoyant body of interest and investigate it visually without the use of paddles or thrusters. Undulatory locomotion is due to one or more propulsive waves that pass along the length of a fin [1]. By producing a wave in the opposite direction to forward movement a thrust force is produced pushing the surrounding water back [2]. This mechanism combined with the drag of the body determines the velocity as illustrated in Figure 1.

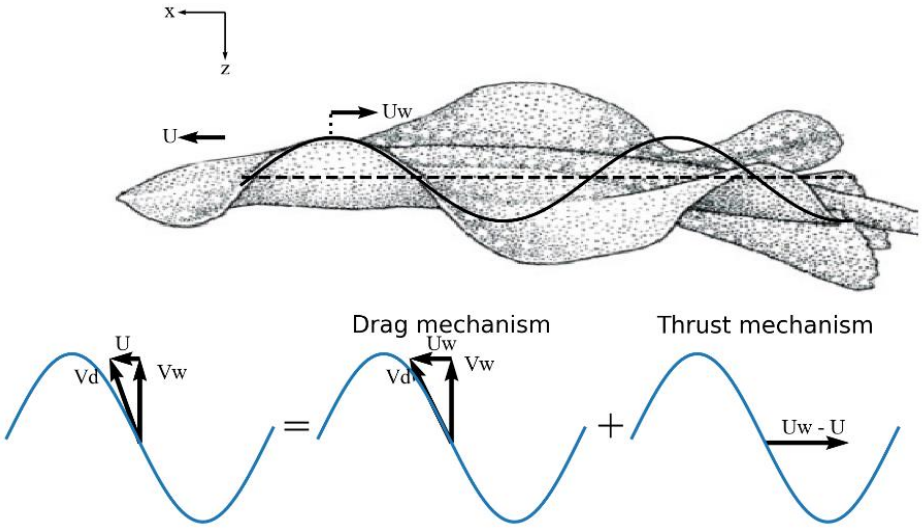


Figure 1. Free body diagram of thrust and drag components where U is the forward velocity, V_W is the lateral oscillatory velocity, U_w is the translating wave velocity and V_d is the velocity tangential to the surface [2]. Assuming that the travelling wave is an ideal sinusoid of constant amplitude, fin undulation produces primary thrust and straight body primarily produces drag. Figure adapted from [1].

Therefore, this paper aimed to (i) utilize undulatory locomotion in a robotic stingray, (ii) develop software that was capable of navigation and provided user feedback in the form of a live stream video.

The materials, design and design choices for the mechanical design will be further detailed in section 2.1 of this report. The electronics is shown in Table 1 with wiring shown in Figure 2. This setup allowed the robot to have the hardware facilities to sufficiently meet its objectives. Jetson-nano was chosen since it provided high compute power above our required frames rate of 6.85fps minimum for the object detection [3] with an expected rate of 4fps [4] and 30fps [5] for Yolov5s and Yolov5n (light-weight model) running at high resolutions. Furthermore, the general-purpose input/output simplified the messaging system in

comparison to using Arduino(s). Servo hats helped simplify powering and wiring of the robot with multiple being used to supply the required number of channels and to avoid race conditions. The intel RealSense D435i was chosen based on its successful underwater mapping applications and ROS (robot operating system) wrapper compatibility.

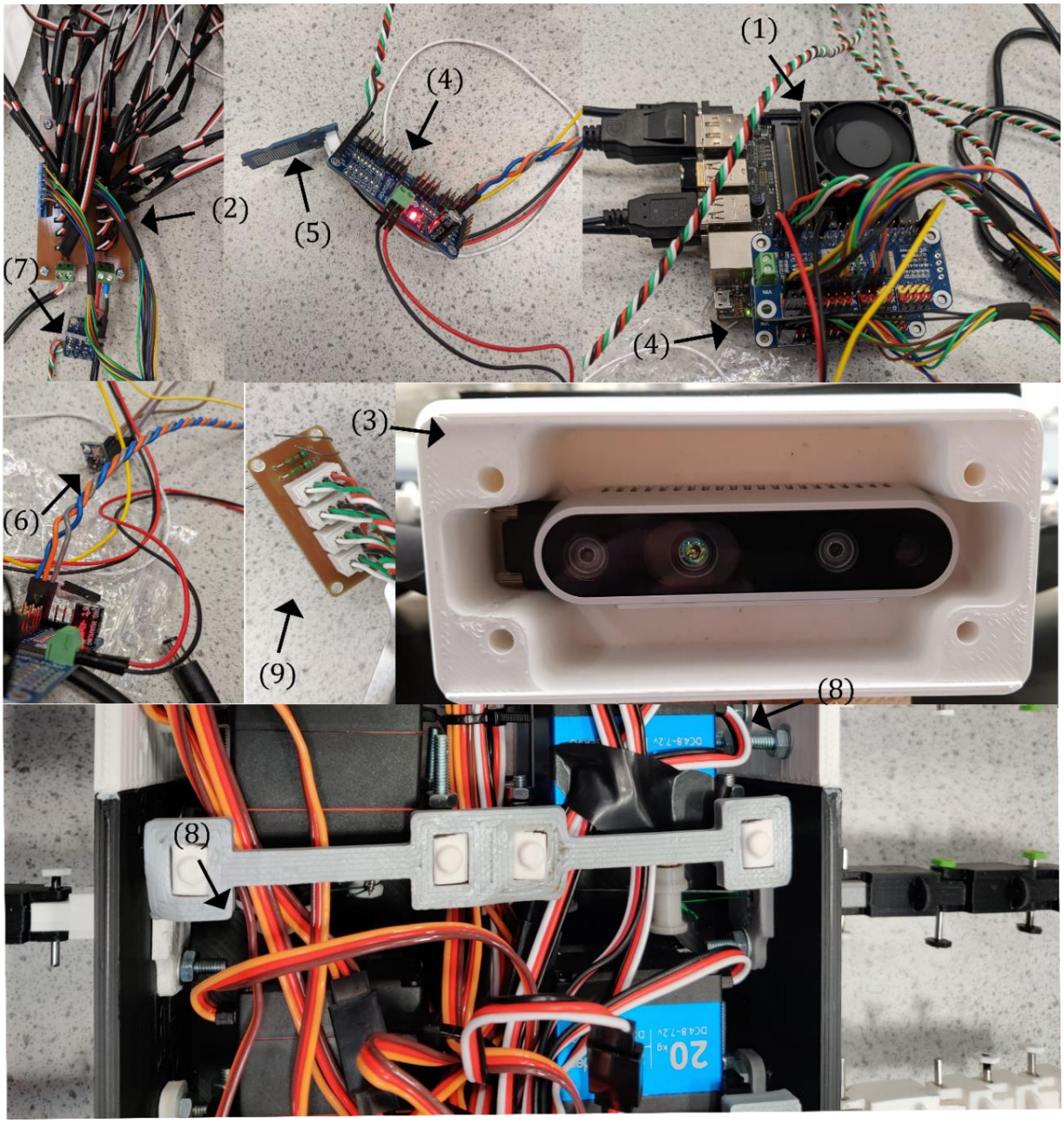


Figure 2. Stingray robot hardware

Table 1. Electronic components

Component	Function
(1) Jetson nano	Main computing unit
(2) Servo power PCB	Distributing power for servo motors
(3) Intel RealSense D435i	Vision and inertial measurements
(4) PCA9685	Servo PWM driver
(5) Grove water sensor	Determining water leak
(6) BMP180	Internal temperature monitoring
(7) Adafruit INA260 power meter	Servo power supply time response
(8) KS-3518/ SF3218MG digital servo motors	Ray and nose actuation
(9) I2C parallel splitter PCB	Share i2c wire across multiple devices with unique addresses

The 4GB Jetson nano was installed with jetpack 4.6 and programmed in ROS [6] with Python3.6/2.7/3.7 for non-compute-intensive tasks (sensor stream) and C++11 for low-level tasks (actuation). The system was deployed using serial console to reduce computational cost which was automated using GNU Screen written in the bash scripting language. The ROS code developed covers live streaming, simultaneous location and mapping (SLAM) [7] with fused RGB-D (RGB and depth image) [8] and filtered IMU odometry [9], goal object detection and 3D localization, obstacle avoidance, system monitoring, and goal location navigation encompassing both depth and yaw control. The system operates with odometry at a rate of 6Hz. Setting the upper limit for the control logic frequency. When queried, the object localization was found to operate at an acceptable level of 6.25Hz. The odometry method performed well indoors when the camera was moved at a steady rate similar that expected from the robot. The main APIs used are outlined below.

Table 2 Core project APIs

API name	Function
ROSv1	Messages, build-system
Pytorchv1.10 [10]	Machine learning
OpenCV4	Image processing
PyZMQ [11]	TCP messaging client for live stream
Actionlib [12]	Event-based behavior

Individual Section - Cunningham, Alexander L

The core of the system was implemented as a client-server model using ROS's services. This model abstracts the system down to two types of asynchronous processes, the action server and the action client. The action server contained within the *action_server* package/program, handles the code for the controller and plant control system. Whilst the action clients handle goal and event-based behavior, such as searching for the goal location (*action_client*) or handling a water leakage (*safety_action_client*). The server and clients interact using messages and also rely on messages for defining their state as shown in Figure 3. Messages are organized using a tree structure with linked nodes for example */stingray/control* provides the base for control and connects to nodes such as */stingray/control/emergency_stop_flag*.

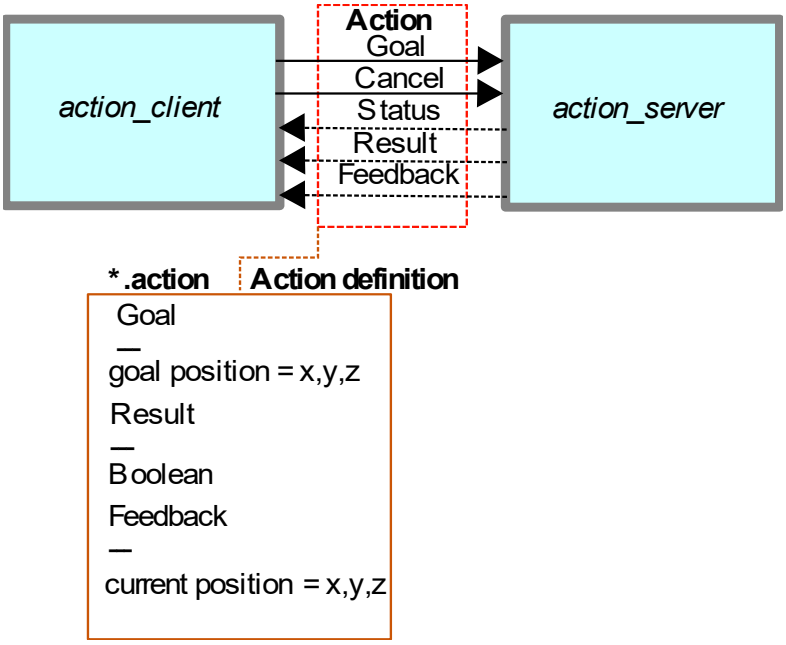


Figure 3. Action definition between action_client and action_server

Other components for functionality were performed using separate background executables with one-way topics that either publish data (sensors), receiving data (actuation) or both (safety manager). Simple request/response service is used for object detection, which responds to client requests. Each program was organized within a package with the code inside organized into classes. Code produced during this project is available at [13] with installation utilities provided at [14]. A non-extensive table of packages is shown in Table 3.

Table 3. Table of packages (programs)

Package name	**Description**
action_client	Manages goal-based behaviors
action_server	Receives goals and contains the controller
depth_control	Adjusts nose angle published by *action_server*
fin_calibration	Manually adjusts initial offset position for fine positioning of servos (default is 90 degrees)
left_fin_actuator/right_fin_actuator	Listens to *action_server* (*/stingray/control/frequency_left*) and adjusts the wave frequency on left/right fins
live_stream	Publishes live stream over transmission control protocol (TCP)
object_detection	Performs object detection then computes and returns the 3D position at the center point of detected box to the client
py_sensor	Publishes sensor topics (e.g., */stingray/pysensors/temperature*)
realsense_ros	Utility that publishes camera sensor messages (e.g., */camera/depth/color/points*)
safety_action_client	Checks for leaks/overheating electronics, user in loop interface

Servos were operated using PCA9685 drivers in C++ with software delays and PWM handled through an adapted implementation. The equation used for control is given by,

Equation 1

$$\alpha_{links}(t) = \alpha_{max} \sin 2\pi f t$$

where at time t the angle of a single link is given by α_{links}, α_{max} is the maximum possible angle of a link, and f is the desired waveform frequency. From this, the wire length change can be calculated based on the rotation of the link using,

Equation 2

$$L_{wire}(t) = \frac{Nw}{2} \sin \frac{\alpha_{links}(t)}{2}$$

where L_{wire} is the change in the wire length, w is the width of the link, and N is the number of links. Using L_{wire}, the angle of the servo can be calculated based on the radius of the winder r_{servo} at a time t.

Equation 3

$$\alpha_{servo}(t) = \frac{L_{wire}(t)}{r_{servo}}$$

Equation 3 provided a limit on the producible wave form given the minimal resolution for the PWM length. Using this, the radius range for the winders was used in choosing the appropriate servos along with a force analysis. Forces included in the analysis were the link weight, hydrostatic force, pulley support reaction force, friction force from the wire as shown in Figure 4. Dynamic forces and viscoelastic forces were estimated using D'Alembert's principle and the Kelvin-Voigt model, both were found to have a negligible effect and are not shown in the following analysis.

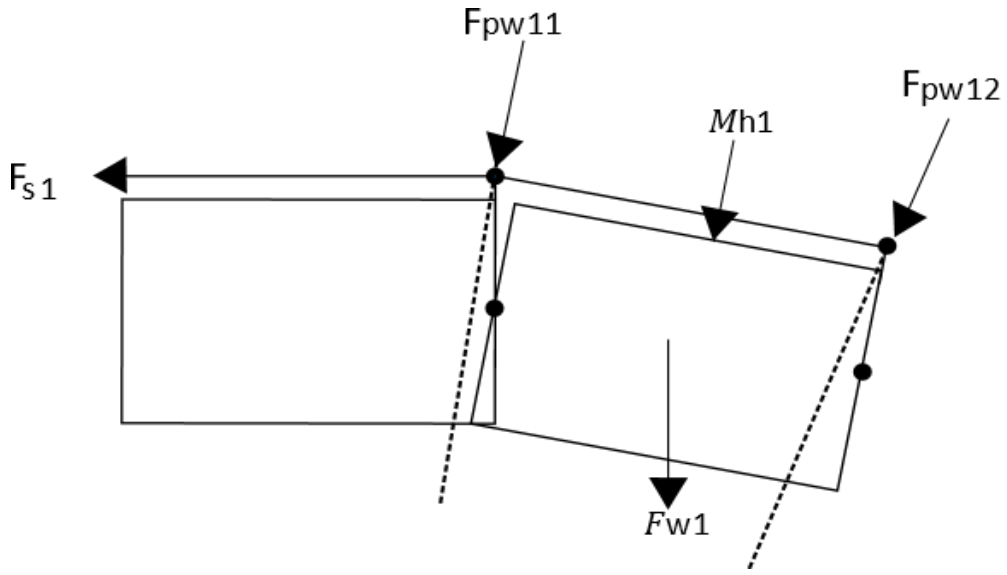


Figure 4. Moment diagram of the first link for the upper wire (lower wire is considered in assumptions). Where F_{s1} is the force of the top pulley wire, Mh_1 is the hydrostatic force acting on the first link, F_{w1} is the force due to the weight of the link and Fpw_{11} is the pulley reaction support force of the first wire on the first link.

Assumptions made in the analysis are as follows:

1. The pins are perfect pinned supports
2. The shape of the fin is rectangular of height 0.4 m and width equal to the ray length
3. The force due to the weight of ray i is equal to the weight of a slice taken across the height of the rectangular fin which acts downwards from the centre of mass of each ray
4. Each ray link sweeps $\alpha/2$
5. Friction and forces are equally distributed
6. The resistance of the lower wire is $F_{pw2} = 0.2F_{pw1}$ (wires move synchronously)

Integrating the hydrostatic force on a single ray across the length of the ray gives,

Equation 4

$$M_{hi} = \frac{1}{3}\rho_{water} g H y_i^3 w$$

The force due to the weight of the fin the moment is given by,

Equation 5

$$F_{wi} = m_{si} g$$

The moment due to the pulley support force can be characterised with the following,

Equation 6

$$F_{sp} = 2\mu F_{s1} cos\left(90 - \frac{\alpha}{2}\right)$$

From this the overall moments can be described by,

Equation 7

$$F_{s1}\left(\frac{w}{2}\right) + F_{sp}\frac{w}{2}sin\left(\frac{\alpha}{2}\right) = F_{sp}\sum_{i=2}^{n} D_{spi} + \sum_{i=2}^{n} Mh_i D_{hi} + g\sum_{i=2}^{n} m_i D_{mi}$$

where the distances D_{spi}, D_{hi} and D_{mi} are the distance between the parallel line running through the first pinpoint and the direction of the force in question (Euclidean distance). This can be solved for F_{pw1} giving a servo torque estimate for radius r. Parameters used in the servo specification analysis are shown in Table 4.

Table 4. Parameters used in the analysis

Parameter	**Name**	**Value**
Density of water	ρ_{water}	$1000\ kg/m^3$
Density of silicone [15]	$\rho_{silicone}$	$2330\ kg/m^3$
Depth of water	H	$2\ m$
Length of long ray joints (joints 1,2)	$y_i \in (y_1, y_2)$	$0.03\ m$
Length of short ray joints (joints 3,4,5)	$y_i \in (y_3 : y_5)$	$0.015\ m$
Width of the rectangular fin	w	$0.4\ m$
Acceleration due to gravity	g	$-9.81\ m/s^2$
Maximum angle of deflection for ray links	α_{max}	$15°$
Servo winder radius	r	$0.025\ m$
Ray thickness	t	$7\ mm$
Safety factor [16]	SF	1.75
Friction coefficient [17]	μ	0.2

Results are shown in Figure 5. Results showed that the maximum torque occurred at the neutral position requiring a torque of 18 $kgcm$ (including SF). Using this result and the result from the PWM calculations which showed that all angle change increments were possible suitable servos were purchased.

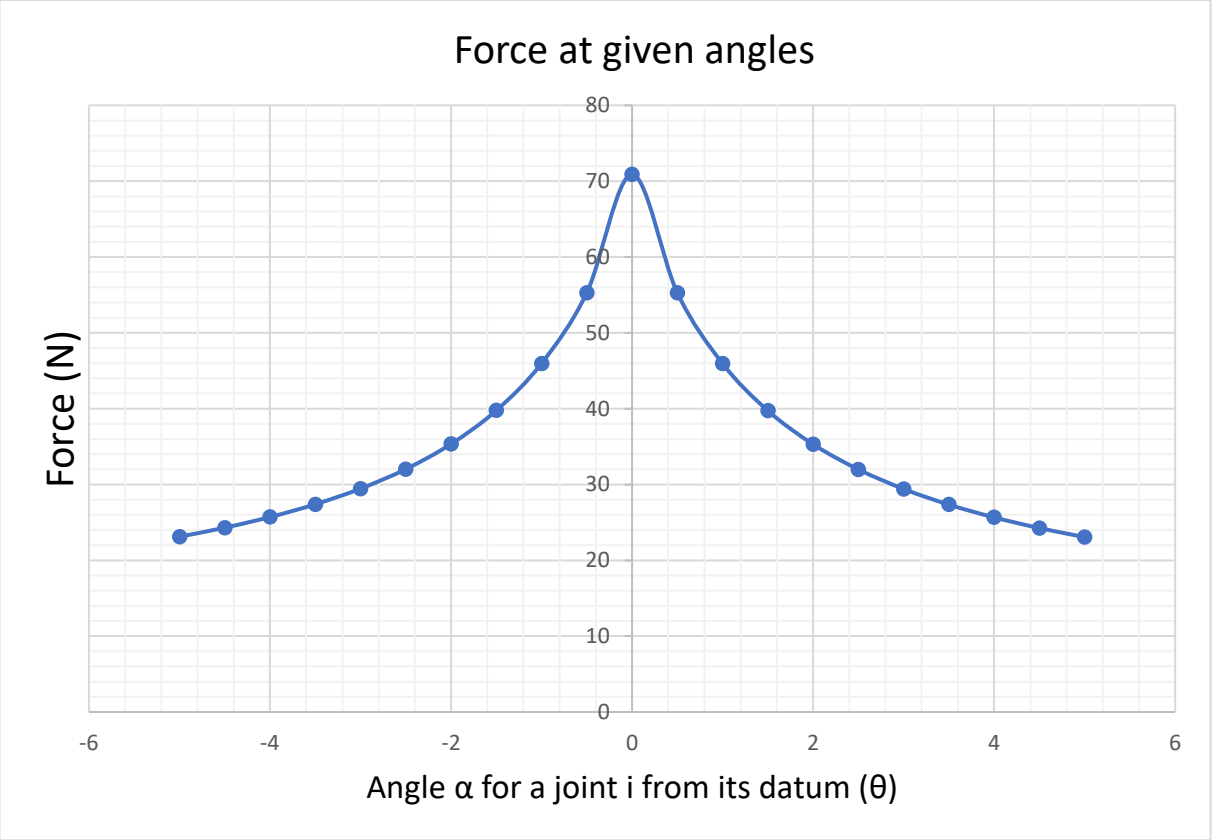


Figure 5. Servo specifications analysis excluding SF.

To actuate the entire wave a phase difference was applied given the separation of the actuators. Wave production was validated by analyzing tracking data. Three stop levels were implemented to reduce system stress. These levels were controlled by *stingray_obstacle_avoidance* (*stingray/control/emergency_stop_level*). For low severities (default) the actuator created a decaying wave based at a rate specified by the number of periods passed before a goal frequency and amplitude was reached given by,

Equation 8

$$f, \alpha = f_{initial} e^{\zeta t}, \alpha_{initial} e^{\zeta t}$$

where $\zeta = \frac{\ln (f_{goal}/f_{initial})}{1/f_{initial}}$ and t was in step increments until the f_{goal} was reached at a specified time as a fraction of the $1/f_{initial}$. For medium level severities the actuators stop at the end of the current waveform. Whilst for high level severity flags the actuators create a

shield with the actuator. Stopping is flagged by the *safety_action_client* (system wide termination), *stingray_obstacle_avoidance* to avoid collisions, and *action_server* if preempted by the *action_client* or if the current goal has been achieved. The *safety_action_client* uses a human-in-the-loop provision for greater safety compliance [18]. The user is notified and provided the option to cancel the current goal and terminate all programs if the internal temperature reaches the maximum operating temperature of the servo motors or if a leak has occurred.

The goals defined within the action client were searching for the plastic bottle(s), going to the plastic bottle goal location and returning back to the home initial location as is shown in Figure 6.

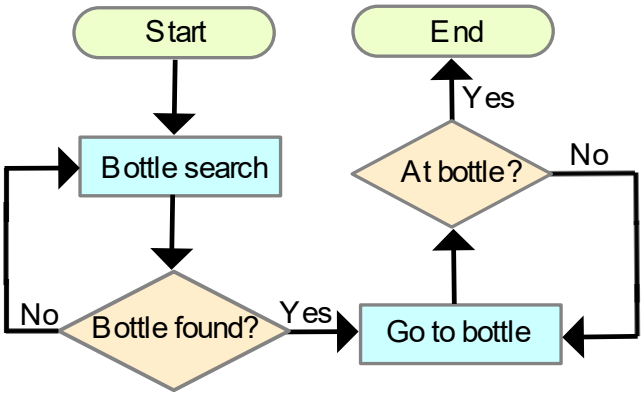


Figure 6. High-level logic flow overview

Initially, the goal client queries the object detector server for the goal location of the bottle (bottle search). The server then inputs the image (*/camera/color/image_raw*) into the object detection model outputting a bounding box. Using the pixel index for the center of the box the 3D point cloud string provided in ordered pixel format was indexed (deprojection of depth image */camera/depth/color/points*). This coordinate was then transformed from the camera frame to the map frame by defining a translation matrix using camera orientation and position from the origin (initial position). This information is provided by the SLAM (simultaneous localization and mapping) utility robot localization which uses RGB-D (RGB and depth image) and IMU data provided by the Realsense-D435i camera. Object detection is synchronized for point cloud data, camera image and transformation matrix sampling using a message queue. The action server publishes control signals until the object is found, or the goal is cancelled due to a convergence speed below a predefined limit.

Upon detection the action server is informed of the result and begins trajectory control. Trajectory control consists of depth and yaw control. The yaw control system expands upon

the proportional controller developed by Cloitre et al. [19]. The control system adjusts the pectoral fin frequency depending angle between the current heading calculated from */odometry/filtered* message and goal position supplied by the *object_detection* package.

Equation 9

$$\delta_f = K_{\mathrm{p}\Psi}\left(\Psi_{ref} - \Psi_{measured}\right)$$

Equation 10

$$f_{left} = f_{nominal} - \delta_f$$

Equation 11

$$f_{right} = f_{nominal} + \delta_f$$

where, Ψ_{ref} is the angle between the heading direction and the goal position and $\Psi_{measured}$ is the yaw determined through positioning. Both headings are updated with messages *stingray/control/frequency_left* and *stingray/control/frequency_right* corresponding to the variables $f_{left,right}$ received by *right/left_fin_actuator* packages. This adjusts the difference between positive and negative yaw moments. Ψ_{ref} was calculated by calculating the line from the current position to the next position based on the heading,

Equation 12

$$x_1, y_1 = x_2 + \cos\Psi_{measured}\,, y_2 + \sin\Psi_{measured}$$

Then taking two lines from the current position (x_2, y_2) to the next (x_1, y_1) and goal positions (x_3, y_3). From this, the lines were calculated in terms of direction ratios,

Equation 13

$$AB, BC\ = (x_1 - x_2)\hat{\imath} + (y_1 - y_2)\hat{\jmath}\,, (x_3 - x_2)\hat{\imath} + (y_3 - y_2)\hat{\jmath}$$

The angle can then be determined using,

Equation 14

$$\Psi_{ref} = \cos\left(\frac{AB \cdot BC}{|AB||BC|}\right)^{-1}$$

For headings over a specified threshold $\Psi_{threshold}$ the system pivots with one fin stationary until the error between the Ψ_{ref} and $\Psi_{measured}$ is sufficiently small.

The depth controller adjusts the initial offset θ_{offset} within the upper and lower bounds θ_{bounds} of the sine wave produced by the fins with messages */stingray/control/offset_left* and *stingray/control/offset_right*, hence creating an emphasized stroke relative to the central line of the robot. This has the effect of creating a moment about the center of gravity. For a backward moving wave (opposite direction to the nose), shifting the emphasis below the neutral position of the wave creates a counter clockwise torque raising the nose of the robot whilst moving the stroke emphasis above creates a positive torque downwards dropping the nose. The adjustment of the neutral position is coupled with adjusting the nose angle (*/stingray/control/nose_angle*). With a lowered nose creating a clockwise torque and a lowered nose creating a counter clockwise torque. The equation used for the depth control are shown below,

Equation 15

$$\delta_{zl} = \delta_z sgn(f_{left}), \delta_{zr} = \delta_z sgn(f_{right})$$

Where $\delta_z = K_{pz}(z_3 - z_2)$, z_3 is the goal location and z_2 is the current depth location and δ_{zl}, δ_{zr} are the offset angles applied and are both bounded by $\delta_{upper}, \delta_{lower}$.

Equation 16

$$\theta_{nose}(\delta_z) = \begin{cases} -26, & if\ z_{lower} > \delta_z \\ 0, & if z_{lower} < \delta_z < z_{upper} \\ 26, & if\ z_{upper} < \delta_z. \end{cases}$$

Where the nose angle is given by θ_{nose}. Values previously mentioned in the control algorithm implementation are given in Table 5.

Table 5. Control algorithm variables

Parameter	**Name**	**Value**
Yaw turning threshold	$\Psi_{threshold} \in (\Psi_{lower}, \Psi_{upper})$	$(-70,70)$ °
Frequency upper and lower bounds	$f_{bounds} \in (f_{lower}, f_{upper})$	$(-2,2)\ Hz$
Nose angle adjustment threshold	$z_{threshold} \in (z_{lower}, z_{upper})$	$(-0.1,0.1)\ m$
Wave offset upper and lower bounds	$\theta_{bounds} \in (\theta_{lower}, \theta_{upper})$	$(-10,10)$ °
Depth controller proportional constant	K_{pz}	0.165
Yaw controller proportional constant	$K_{\mathrm{p}\Psi}$	0.05
Nominal wave frequency	$f_{nominal}$	$1.4\ Hz$

Results

Software performance results are shown below in Figure 7 with qualitative results being performed during dry testing. Upon testing the integrated system a leak occurred resulting in the failure of the system. Further improvements to the design are needed for the system control to be optimized with the possible addition of integral and derivative terms depending on performance.

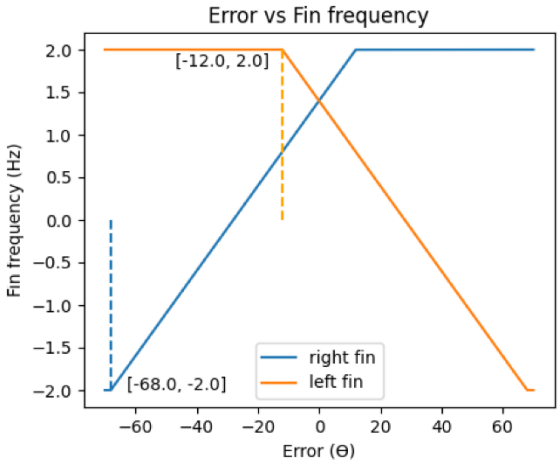
Software performance

Quantitative

Performance metrics	
Total power usage (2Hz wave)	20 $Watts$
Object detection (during SLAM)	6.5 Hz
Point cloud/RGB image/SLAM	6 Hz @ 424×240
Detection confidence threshold on test dataset	80% $confident$
Odometry $average\ std$	$\pm 3\ mm$
Produced wave frequency	$f(1) \approx 0.987 \pm Hz$

Limiting factors	
Point cloud processing (deprojection)	Occasional spike in dropped frames
Object detection model loading	Start-up time increased to 20 seconds
Object detection memory usage	Only possible to run headless
Object detection underwater image quality	Clear water is required for the nano model

Qualitative



- Correct frequency adjustments to a known goal location $(\Psi_{ref} - \Psi_{measured})$
- Waveform for desired wavelengths accurate

Figure 7. Software performance results.

References

[1] E. Blevins and G. V. Lauder, “Rajiform locomotion: three-dimensional kinematics of the pectoral fin surface during swimming in the freshwater stingray Potamotrygon orbignyi,” *The Journal of Experimental Biology,* vol. 215, no. 18, pp. 3231-3241, 2012.

[2] R. Bale, A. Shirgaonkar, I. D. Neveln, A. P. S. Bhalla, M. A.MacIver and N. A.Patankar, “Separability of drag and thrust in undulatory animals and machines,” *Scientific Reports,* vol. 4, no. 1, 2014.

[3] J. Sattar, M. Fulton and M. Islam, “Toward a Generic Diver-Following Algorithm: Balancing Robustness and Efficiency in Deep Visual Detection,” *IEEE Robotics and Automation Letters,* 2018.

[4] “YoloV5 Jetson Nano,” 11 January 2022. [Online]. Available: https://github.com/Qengineering/YoloV5-ncnn-Jetson-Nano.

[5] “YOLOv5n Nano Study,” 2 October 2021. [Online]. Available: https://github.com/ultralytics/yolov5/discussions/5027.

[6] S. A. I. Laboratory, “Robotic Operating System,” [Online]. Available: https://www.ros.org.

[7] “robot_localization,” [Online]. Available: http://wiki.ros.org/robot_localization.

[8] “rtabmap_ros,” [Online]. Available: http://wiki.ros.org/rtabmap_ros.

[9] “imu_filter_madgwick,” [Online]. Available: http://wiki.ros.org/imu_filter_madgwick.

[10] A. Paszke, S. Gross, F. Massa, A. Lerer and J. Bradbury, “Pytorch: An Imperative Style, High-Performance Deep Learning Library,” in *International Conference on Neural Information Processing Systems*, Vancouver, 2019.

[11] “PyZMQ Documentation,” 2022. [Online]. Available: https://pyzmq.readthedocs.io/en/latest/.

[12] “actionlib,” 2018. [Online]. Available: http://wiki.ros.org/actionlib.

[13] A. Cunningham, “ROSStingrayRobot,” [Online]. Available: https://github.com/alc9/ROSStingrayRobot.

[14] A. Cunningham, “my jetson setup,” [Online]. Available: https://github.com/alc9/my-jetson-setup.

[15] “MatWeb Material Property Data,” [Online]. Available: http://www.matweb.com/index/aspx.

[16] “Selection Calculation for motors: Technical Reference,” 28 08 2015. [Online]. Available: https://www.orientalmotor.co.in/file/etc/Selection_Calculations_For_Motors.pdf. [Accessed 30 03 2021].

[17] H. Jostlein, “Friction of Steel Wire on Pins, Warm and Cold,” 30 12 2014. [Online]. Available: https://lartpc-docdb.fnal.gov/cgi-bin/RetrieveFile?docid=10&filename=wirefriction.pdf&version=1. [Accessed 01 04 2021].

[18] D. Mitchell, J. Blanche, O. Zaki, J. Roe and L. Kong, *IEEE Access,* vol. 9, 2021.

[19] A. Cloitre, N. M. Patrikalakis, V. Subramaniam and P. V. Alvarado, “Design and control of a field deployable batoid robot,” in *The Fourth IEEE RAS/EMBS International Conference*, Roma, 2012.