Biomimetic Snake Robot for Precise Locomotion

CS39440 Major Project Report

Author: Jamaine Christian (jac105@aber.ac.uk)

Supervisor: Dr Myra Wilson (mxw@aber.ac.uk)

03th May 2024

Version 3.0

This report is submitted as partial fulfilment of a BSc degree in Artificial Intelligence and Robotics (GH76)

Department of Computer Science

Aberystwyth University

Aberystwyth

Ceredigion

SY23 3DB

Wales, UK

Declaration of originality

I confirm that:

- This submission is my own work, except where clearly indicated.
- I understand that there are severe penalties for Unacceptable Academic Practice, which can lead to loss of marks or even the withholding of a degree.
- I have read the regulations on Unacceptable Academic Practice from the University's Academic Registry (AR) and the relevant sections of the current Student Handbook of the Department of Computer Science.
- In submitting this work, I understand and agree to abide by the University's regulations governing these issues.

Name Jamaine Christian

Date 03/05/2024

Consent to share this work

By including my name below, I hereby agree to this project's report and technical work being made available to other students and academic staff of the Aberystwyth Computer Science Department.

Name Jamaine Christian

Date 03/05/2024

Aknowledgements

I am thankful to my supervisor Myra Wilson for overseeing the project and sharing her knowledge with me to help guide me through the process.

Abstract

The goal of this project is to build a biomimetic snake robot that can mimic a biological snake's locomotor behaviours. Through design iterations and algorithmic advancements, the goal is to develop a robot that can navigate complex environments more effectively than traditional designs.

The initial stages involved constructing a basic wheeled robot, which evolved into a modular segmented design controlled by advanced algorithms to replicate snake-like motion. Challenges arose in accurately mimicking biological locomotion, requiring iterative adjustments and highlighting the complexity involved.

The Webots simulation environment provided a stable platform for testing, while real-time processing and C++ programming facilitated algorithm implementation. Despite achieving success in mimicking various snake gaits, challenges persisted in stable forward motion and obstacle navigation.

The project demonstrates innovation in utilizing simple harmonic motion and phase-shifts for locomotion control but highlights the gap between theoretical principles and real-world implementation. Skills gained include robotics, programming, and sensor integration, with insights into biomimetic design and sensor-controlled algorithms.

Future work should aim to refine control systems, introduce machine learning algorithms, and assess broader ecological and societal implications. Overall, the project highlights the complexity of using theoretical knowledge in practical application in robotics.

Table of Contents

BACKGROUND/ANALYSIS/PROCESS

Overview of Biologically Inspired Robotics:

Introduction to Biologically Inspired Robotics

Biologically inspired robotics is an interdisciplinary field bridging robotics and biology, comprising of two main technical areas. The first area involves applying biological concepts to solve engineering challenges in robotics. In this area, biomimetics, robots are designed to emulate living organisms, use sensors in ways like biological sensing systems, creating control algorithms in similar ways to biological sensor-motor schemes, and designing actuators that work like muscles [1].

The second technical area, known as bio-robotic modelling/analysis, focuses on using robotics to understand biological systems and behaviours. This involves robotics modelling of biological processes, simulations of biological behaviours, and exploration of biological recognition and motor functions. This area aims to leverage robotics technology to help advancements in biology [1].

Robots inspired by biology such as robot snakes for rescue operations in disasters, snake-like endoscopes for medical diagnosis, and artificial muscles for limb motor function restoration, highlight the practical applications of biologically inspired robotics [1].

Snake Robots

Design Inspiration from Biological Snakes

Snake, robots emulate the complex locomotion of biological snakes. These robotics systems mimic the flexible, limbless movement of the snakes, which allows them to navigate through uneven terrain and tight spaces with efficiency. The design of a snake robot typically involves multiple segments connected in chain by joints, resembling the vertebrae of a snake. This design allows smooth, continuous, precise motion mimicking snake-like undulation.

Snake locomotion involves contracting and expanding its muscles to create lateral waves that propagate down the body. For a snake-like robot, this motion is replicated through actuators positioned along the segments, which simulate the snake’s muscle contractions. These actuators are often controlled by algorithms to create a natural looking snake movement.

Snake Locomotion

Snakes are popularly associated with only one type of movement style, i.e slithering, however, they can locomote with four very different gaits, such as, lateral undulation, sidewinding, concertina movement, and rectilinear locomotion.

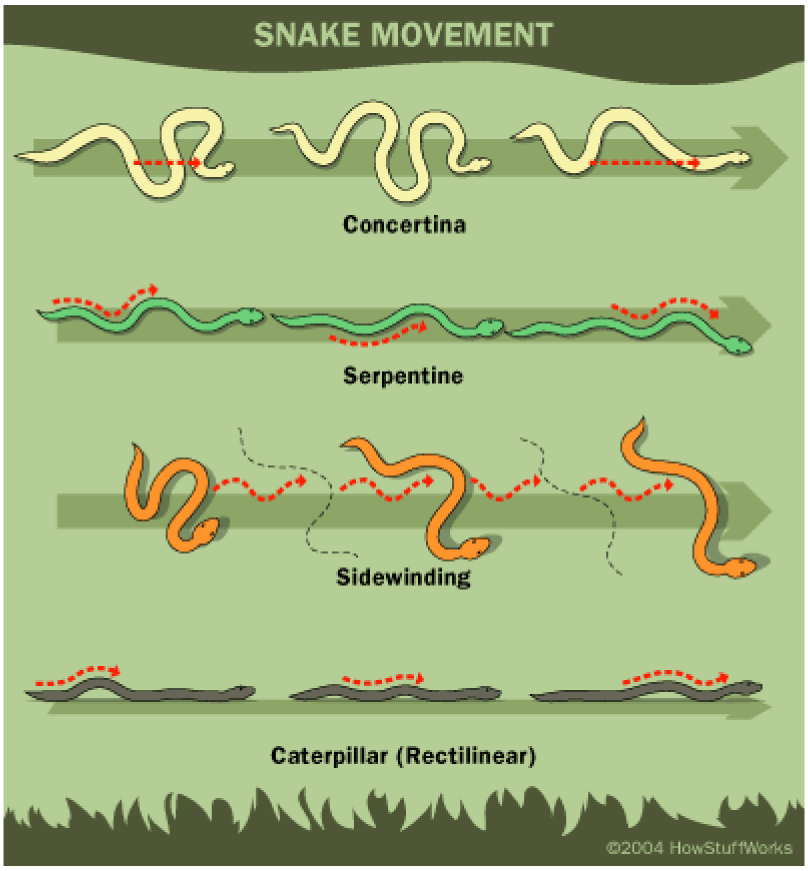
Almost all species of snakes can perform any of these four movement styles. In different scenarios and terrains, different locomotive movement styles are more effective than others. In addition, snakes often combine two or more types of locomotion together to adjust to the complexity of its environment. Thus, studying how snakes adapt to certain terrains may provide hints on how snake robots can be designed to adapt these environments as well [2].

“Serpentine: the most characteristic movement and the one that enables them to move at the greatest velocity. The serpent advances like a wave, through a sinusoidal movement of its body” [3].

“Concertina or accordion: the snake forms volutes or twists with its body contracting and expanding successively like a spring or accordion, moving from one anchor or impulse point to the next” [3].

“Lateral displacement or sidewinding: typical of the species that inhabit the desert (and in particular the sidewinder rattlesnake). The snake forms vertical waves to minimize the contact areas with the scorching surface and in this way it moves laterally” [3].

“Rectilinear: this mechanism has recently been unravelled and is the only one that does not fit the initial definition, since the snake crawls in a straight line with its body stretched. It is typical of the largest and more voluminous of the species as it allows them to access the narrow



burrows of their potential prey” [3].

[4]

Control Systems and Navigation

Controlling a snake robot involves sophisticated software algorithms that coordinate the movements of its body segments to achieve smooth, snake-like motion. Incorporating various sensors to detect the environment and adjust the robot's movement in real time is one method to achieve this. The more combination of sensors you have the more accurately you can adjust the robot to the environment, however this will increase the runtime, cost and complexity of the control algorithms.

Another would be incorporating path planning algorithms, like A* path planning, to determine the optimal path for the robot to reach its goals while avoiding obstacles. The robot could also be made to learn and adapt from past experiences by implementing machine learning techniques to help the robot navigate complex environments.

Hirose's Formulae.

Work by Hirose and Umetami, in the early 1970's, was among the first to explore and develop limbless locomotors. Hirose focused on developing robots that could perform lateral undulation and later developed a series of wheeled coupled-mobility devices that followed from this work. [5]

$$x(s) = sJ_o(\alpha) + \frac{4l}{\pi}\sum_{m=1}\frac{(-1)^m}{2m}J_{2m}(\alpha)\sin\left(m\pi\frac{s}{l}\right)$$

$$y(s) = \frac{4l}{\pi}\sum_{m=1}^{\infty}(-1)^{m-1}\frac{J_{2m-1}(\alpha)}{2m-1}\sin\left(\frac{2m-1}{2}\pi\frac{s}{l}\right)$$

[5]

Hirose's development of modelling and control first derived expressions of force and power as functions of distance and torque along the curve described by the snake. The curve was then derived and compared with results from natural snake locomotion. The curve, termed serpenoid, has curvatures that vary sinusoidally along the length of the body axis. These equations are shown below: [5]

This curve is different from sinusoidal curves. Comparisons with natural snakes across constant friction surfaces showed close agreement between the serpenoid curve and the empirical data. [5]

Simulation Environment

Why Webots?

Webots offers a complex simulation environment for creating and testing robotic applications. It is an essential tool for robotics research and development because of its extensive toolkit, which enables the simulation of intricate interactions between robots and their environments.

Features and Advantages of Webots

High-Accuracy Simulation: Webots' physics engine accurately replicates real-world physics, ensuring that simulations provide a reliable representation of how robots would behave in actual conditions.

Sensor and Actuator Simulation: The ability to simulate a wide range of sensors and actuators allows developers to test their systems under various scenarios before physical deployment.

Ease of Use and Accessibility: Webots is designed to be user-friendly, offering a graphical interface and support for multiple programming languages, which lowers the barrier to entry for new users.

The Rationale for Choosing C++ in Robotics

When completing a sophisticated project that dives into biologically inspired robots, particularly the study of snake locomotion in robotics, the selection of a programming language is vital. The language must not only facilitate complex algorithm implementation but also ensure smooth integration with the hardware. C++ stands out as the best in this context, and here's why.

High Performance and Hardware-Level Access

Its capacity to operate close to the hardware layer enables efficient resource management, a critical feature in the field of robotics where timing and responsiveness are non-negotiable. Real-time processing and fast reaction times are the backbones of robotic systems, and C++ ensures that these requirements are met with precision.

Object-Oriented Programming for Complex Systems

C++ being object orientated aligns perfectly with the complexities of robotic design. By encapsulating the various components of a robot, such as its sensors, actuators, and control systems into classes, C++ helps maintain an organized and scalable code. This approach is beneficial in managing and evolving sophisticated systems like snake robots, which consist of numerous interdependent parts.

Cross-Platform Compatibility

Code can be compiled across multiple platforms, making it an ideal choice for robotics applications destined for a range of operating systems and embedded devices. This flexibility is invaluable in a dynamic field where applications must adapt to varying environments and technologies.

Library Ecosystem and Integration

With access to an abundance of libraries for simulation (such as the Robot Operating System - ROS), machine learning, and control algorithms, C++ provides a solid foundation for developing advanced robotic capabilities. In addition, its compatibility with other languages and tools, including APIs from simulation environments like Webots, ensures that C++ can be a central part of a versatile and powerful development toolkit.

Algorithmic Efficiency

When it comes to implementing the complex algorithms necessary for simulating snake-like locomotion, C++ excels. Its ability to handle elaborate computational tasks efficiently is essential for both the development and real-time execution of control strategies within robotic systems.

In conclusion, the selection of C++ for a major project focused on snake robots offers a convergence of benefits: from performance and system-level control to broad applicability and community support. It is a choice that lends itself to the development of robust, effective robotic systems, and prepares budding roboticists for the challenges of real-world applications.

Software Development Process Management in Robotics

Process Management in Robotics

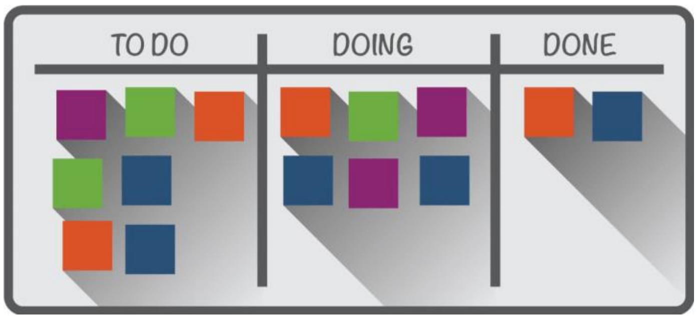
In the field of robotic development, process management plays a critical role in guiding the development of every step up until deployment. Effective process management ensures that robotic systems are functional, reliable, and delivered on time and within budget. To be a strong competitor in the industry, companies must adopt and adhere to process management techniques to keep the company running smoothly and hitting quotas in time.

Kanban Approach

Kanban is a workflow management method designed to help visualise work, maximise efficiency, and continuously improve the process. Originally developed in the Japanese automotive industry in the 1940s by Toyota, it was initially a scheduling system for lean manufacturing and just-in-time production [11,12]. Over time, Kanban has been adapted for use in various fields like software development and project management, due to its versatile and effective strategy.

Visualising Work

Kanban uses a physical or digital board to represent the workflow. The board is split into columns that represent different stages of the process, such as “To Do”, “In Progress”, and “Done”. Tasks are represented by cards or digital counterparts, which are moved from left to right as they progress through these stages [11].



[14]

Limiting Work in Progress (WIP)

One of the core principles of Kanban is to limit the number of tasks underway at any stage of the workflow. WIP limits help prevent bottlenecks by optimising resource allocation and reduce multitasking, which supports an increase in productivity [11].

Managing Flow

The aim of managing flow is to keep work items moving through the process with minimal delays. This practice includes monitoring movement of cards on the Kanban board and identifying any stages where bottlenecks occur. By managing flow teams enhance productivity and reduce the cycle time of tasks [13].

Kanban for Snake Robot Project

The Kanban approach for managing the development of a snake robot simulation project is ideal because it allows changes and adjustments without disrupting the overall flow of the project. For robotics, iterative testing and development are necessary and therefore as problems are encountered this approach allows for the flexibility to deal with them accordingly.

Using Kanban gives a clear view of the project's current state. In a project, like a snake-robot simulation, different components such as design, implementation and testing must be coordinated closely and therefore a Kanban approach would be ideal here.

In the context of designing a snake robot, developing, testing, and refining can be done incrementally. For example, after achieving basic locomotion the next step could be to integrate the obstacle avoidance, and each step can be developed and improved continuously without waiting for a previous section to be completely finalised.

By limiting the work in progress, Kanban prioritises tasks that offer the most value at the current stage of development. For the snake robot, prioritising tasks like refining locomotion algorithms over tasks like object detection will get the snake robot closer to the final goal.

Applications and Challenges of Snake Robots

Snake robots are part of the pioneering field of biologically inspired robots, characterised by their unique flexibility and ability to mimic sinuous movements of a snake. These types of robots are becoming high in demand due to their ability to navigate through confined or challenging environments that other robots or humans could not access. Their applications are diverse, ranging from medical procedures to archaeological exploration.

Surgical Assistance

Snake robots are making significant advancements in the medical field, particularly offering solutions in minimally invasive surgeries. They are used for a variety of procedures including cardiac, prostate and other surgeries involving difficult-to-access areas. The design of these robots often include miniature cameras, surgical tools like scissors and forceps, and additional sensing technologies to further assist surgeons [6].

Research has also focused on the development of continuum robots, which are a type of snake robot with highly flexible structures that can navigate through confined spaces within the human body. These robots use mechanisms like wire-actuating or concentric tube designs to achieve their flexibility and control, making them extremely useful for in surgeries where multiple degrees of freedom and high precision is needed [7].

During minimally invasive surgery, surgeons face several challenges using snake-like robots. The primary issues include managing multiple instruments like graspers and endoscopes from a single abdominal incision point. This constraint limits the instruments' degrees of freedom, complicating their manipulation. Another issue with using surgical tools instead of direct hand contact is that it leads to a loss in sensory feedback, such as feeling for texture and temperature [7].

Further advancements in the mechanical design of these robots include the integration of virtual fixtures into their control algorithms, which assist surgeons by limiting excessive movements and helping manage complex surgical tasks. This integration allows for precise, controlled movements that are essential for delicate surgical operations [8].

Industrial Inspection

In industrial inspection, snake robots are particularly useful for navigating the complex, confined spaces often found within machinery and piping systems. Companies like OC Robotics have pioneered the use of snake robots in industries such as aerospace and petrochemicals, where they inspect and maintain critical infrastructure. These robots can perform detailed inspections inside narrow pipes, ducts, and other hard-to-access areas, identifying issues like blockages, leaks, and structural weaknesses without the need for disassembly, physically entering hazardous areas or operational downtime [9].

The primary challenges in this application involve the harsh environmental conditions such as extreme temperatures, corrosive chemicals, and high-pressure conditions that threaten the function of the electronic and mechanical components.

To overcome these obstacles, snake robots are designed with durable materials and protective coating to withstand harsh conditions. These robots incorporate advanced control systems and AI algorithms to improve their ability to manoeuvre through complex spaces.

Archaeological Exploration

In archaeology, snake robots are used to explore and document sites without damaging their historical integrity. They can navigate through narrow tunnels and fragile ruins, equipped with advanced sensors and cameras to capture detailed images and data, trying to minimize physical contact with delicate structures [24].

One of the primary challenges in using snake robots in archaeology is the risk of damaging irreplaceable artifacts or structures. In an unstructured, fragile environment, a robot poses a risk with every manoeuvre it makes.

To address these challenges, researchers and engineers have designed snake robots with advanced control systems that improve precision in movement. For example, developments at Carnegie Mellon University have focused on mimicking the locomotion strategies of sidewinder snakes to improve the robots' ability to traverse sandy and uneven terrains [10].

Overall, snake robots exemplify the successful application of biological principles to solve real-world problems through engineering and technology. As research progresses, the scope of their capabilities and applications is expected to broaden, marking a significant advancement in the field of robotics.

DESIGN PROCESS

Initial Conceptualisation

The initial idea was to create a snake robot that could simulate the locomotive gaits of snakes in biology, such as slithering and sidewinding, for advanced manoeuvrability in constrained environments. This approach draws directly from biomimetics, where the goal is not just to imitate the biological form of the snake but also utilise the functional capacity of real-life snakes. The goal was to get the snake to move, and obstacle avoid, while observing the effectiveness of each gait while doing so.

Prototype Development

The design journey began with constructing a simple wheeled robot, consisting of one box for the body, four cylindrical hinge joints for the wheels and four distance sensors to detect walls and obstacles. This base design was used to familiarise myself with Webots and the robot design process.

I then continued to develop the prototype, by connecting four spherical joints, specifically hinge2Joints, and four more box links for the body of the snake. The hinge2Joints allows control over both the lateral and the vertical joint angles, a critical feature for simulating the multidirectional flexion within the muscles seen in biological snakes.

The links and joints were connected in a modular way to allow for greater range of motion and flexibility by allowing each link to bend and twist independently, enabling the robot to navigate through complex environments with obstacles and tight spaces.

This modular construction of the robot makes it easier to scale the robot up or down depending on the task or environment. In addition, if a component of the robot is damaged or does not function properly, the modular design allows for easy replacement of individual links and joints without having to deconstruct the entire robot, reducing downtime and maintenance costs.

Iterative Design Adjustments

Early tests quickly revealed that wheels limited the robot's ability to emulate true snake-like movements and the hinge2Joints did not bend as expected, which revealed the snake-like movements would need to be handled by the hinge2Joint as a motor rather than wheels to create the driving force. The wheels were consequently removed along with the distance sensors, which would be incorporated again later on.

The focus was to use the hinge2Joints as the driving force of the snake robot, however, after removing the wheels, I noticed that the joints were too large and were in contact with the ground and therefore had to be reduced in size.

The change in the way the snake robot creates its driving force focused efforts on perfecting joint movements using principles of harmonic motion to simulate sinuous, snake-like undulation.

The snake was slithering on the spot but was not creating the right forces to drive the snake forwards. To deal with this, I changed the segments of the snake from boxes to a cylindrical design with semi-sphere at the head and tail to round of the cylinders.

At this point, the robot had some sort of snake-like movement but not in the desired direction, therefore the integration of an advanced sensory system was used to try and keep the robot moving in the desired direction.

Integration of Advances Sensory System

Devices for Locomotion control

At this point, the robot could move with a serpentine gait, but in not in the desired direction. After testing the effects of changing the amplitude of the waves the snake robot was creating and testing how the movement was affected by changing the different phase-shifts between the different joint angles, it became clear that the lateral phase-shifts, vertical phase-shifts, and amplitude were very important factors for deciding the direction of the snake's movement. Changing the amplitude created dramatic changes, either allowing less friction which made the robot move faster or with larger increments the robot would lose balance and roll over. Adjusting the phase-shifts on the lateral joint angles controlled the direction the robot would go in and the vertical phase shifts would need to be adjusted with these two to get specific directions. Therefore, a device which could measure the current yaw, roll, and pitch would be needed to be integrated, so that the phase-shifts could be adjusted based on this error value.

I began looking into different robotic sensors and devices that could be used to measure the roll, pitch, and yaw of a robot. Through research I came across Inertial Measurement Units, which are, electronic devices that measures and reports acceleration, orientation, angular rates, and other gravitational forces. It is composed of 3 accelerometers, 3 gyroscopes, and depending on the heading requirement, 3 magnetometers. One per axis for each of the three robot axes: roll, pitch, and yaw [15].

I added an Inertial Unit device in Webots to head of the snake so that the yaw, roll, and pitch of the snake could be measured. The reasoning behind placing the inertial measurement unit in the head of the snake is that the robot is built in a modular structure therefor the rest of the body will follow the movement of the head. Being able to control the head movement should help to control the locomotive behaviour of the snake robot.

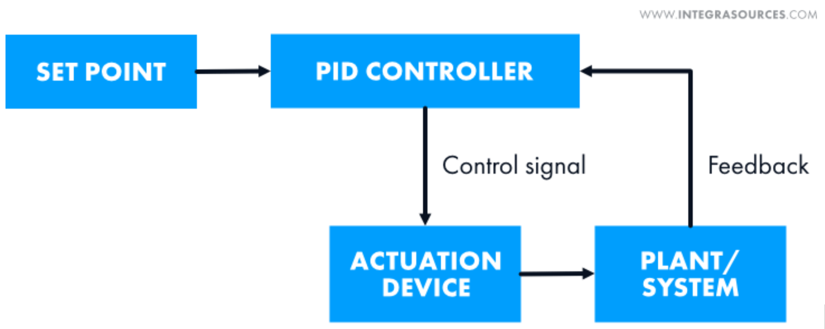
I then created a function to detect the current yaw, roll, and pitch of the robot, using the IMU device in the head of the snake, this function was called every time the move forwards function was called. This 'desiredDirection' variable was set as zero, as a zero yaw in the head of the snake robot would keep the robot moving forwards in a straight line. The current yaw was compared against the desired direction (yaw = 0, to go straight) and the error value was returned. I was struggling to find correct amplitude and phase-shift values for turning the robot left and turning the robot right, therefore, using fixed phase-shift values at certain error thresholds would not work for adjusting the robot to keep its direction straight. Due to this, I had to look more into proportional integral derivative controllers to dynamically adjust the phase-shifts based on the current yaw error.

A PID (Proportional – Integral – Derivative) controller is a widely used instrument in industrial control systems, deployed to regulate various process variables like temperature, flow, pressure, and speed. It utilizes a feedback mechanism from the control loop to keep the process output as close as possible to a desired setpoint, ensuring high accuracy and stability [16].

The main job of a PID controller is to adjust the process in response to differences in a setpoint, which might be caused by adjustments to the mass, load, or energy source. In essence, the aim is to match the feedback to a predetermined target, such as maintaining a straight path in the snake movements [16].

The function of PID controllers involves three key actions: proportional, integral, and derivative tuning. Proportional tuning adjusts the control action in proportion to the difference between the desired and actual outputs. Integral tuning addresses cumulative errors by increasing the correction over time, which can lead to overshooting the target. Derivative tuning is used to smooth out this overshoot by decreasing the correction as the target is approached, enhancing response time and stability [17].

These controllers are particularly effective in closed-loop systems, also known as feedback control systems, where the output is continuously monitored and adjusted to achieve consistent and precise control over various process variables. This setup includes sensors to measure outputs, a control unit to compare outputs against targets, and actuators to adjust the system's operation based on these comparisons [17].



[18]

After incorporating the IMU device and the PID controller, the snake robot was adjusting the yaw dynamically. An issue arose, the PID controller was over adjusting due to the sinusoidal movement causing a wrap around effect, which caused the robot snake to wrap around itself and stop functioning correctly. Limits were created to stop this wrap around effect, which constrained joint angles from going past realistic physical boundaries.

The PID controller worked at keeping the yaw of the robot head straight and stopped the robot from turning however, this did not keep the robot moving forwards in a straight line. I increased the amplitude of the vertical phase offsets to try and create more force on the bottom of the snake for the driving force.

To measure the force of collision against the floor as the amplitude and vertical phase offsets were adjusted, I installed a touch sensor on the bottom of each segment of the snake, which acted as a force sensor reding the impact force. After adjusting the amplitude and vertical phase offsets, the force reading did not change and showed either the sensor was not working correctly or that the amplitude did not affect the force. Certain amplitudes and vertical phase offsets caused the robot to roll over.

I then considered adding a compass to measure the position of the snake robot relative the world and use this to detect forward motion. At this point, the PID controller was removed because the robot would not move forwards.

I created a function to detect the current direction and used this the first time the move forwards function was called and set the desired direction variable as this direction. This variable would only be updated when move forwards was called after performing a turning movement. I could not get this to work properly through my implementation.

I then manually adjusted the different phase offsets and observed the snakes behaviour, until I had the snake moving forwards, however, it would roll over as it progressed and stop moving properly.

To resolve this, I implemented another PID controller, but this time adjusting the vertical phase offsets based on the error in the pitch from the IMU readings , to keep the snake upright. I used pitch instead of roll because I placed the IMU in the wrong orientation. This worked but affected how the snake progressed. Further phase offset adjustment was necessary.

Devices for obstacle avoidance

After refinement of the snake's movement and different gaits, obstacle avoidance was the next task. To accomplish this, I added ten distance sensors around the snake robot to detect the obstacles. This stopped the snake movement because as the snake moved it would roll over slightly and the distance sensors would be triggered by the floor surface. To fix this, I adjusted height and the angle of the distance sensors, so they did not contact the ground.

The distance sensors would not detect an obstacle from a distance great enough to allow the snake to avoid the obstacle. I tried to adjust the range but could not get this to work, so instead I removed the distance sensors and added four range finders to detect the obstacles, as these worked at longer ranges. The four distance sensors were placed around the robot, one at the head and tail and one at either side of the middle segment of the snake robot.The increase in mass from the range finders caused the robots movement to be affected, therefore further refinement of the locomotive function and PID controller was needed.

The snake robot could now avoid most obstacles however if the obstacle was approaching from certain angles the robot would not detect it. Therefore, I added four distance sensors again, two at the head and two at the tail pointing outwards into the blind spots. The size of the distance sensors was small enough not to affect the locomotive behaviour of the snake robot. This helped but again, the distance sensors did not detect obstacles from a distance far enough for the snake to adjust in time to avoid collision. After figuring out how to adjust the range, I had obstacle avoidance working.

Devices for hunting

After obstacle avoidance, the next task was to simulate act a snake hunting for its prey. To accomplish this, I added four cameras on the middle segment of the robot elevated on a pole, so they don't face the ground while moving. These cameras were positioned so that they cover and angle of $\pi/2$ and each faced out in a different direction to cover every angle of the robot. I try to implement object recognition on each of these to get the robot to adjust its direction to a approach (hunt) an obstacle. I could not get recognition to work on the robot.

Design Analysis

Comparative Analysis of my Snake Robot vs Underwater Undulating Propulsion Robots

Advantages of my design

The snake robot's ability to mimic complex snake-like undulations and moving with different gaits allows it to navigate through tight constrained spaced more effectively than many underwater robots that rely on fin movements. The modular design of the snake robot allows for redundancy in the joints to create more refined movement and manoeuvre through hard to access areas. This also allows for quick replacements of parts, minimising down time. This modular design is less common in rigid-bodied underwater robots. The snake robot being designed for land use rather than water provides a broader range of applications [19].

Disadvantages of my design

The reliance on precise control of multiple hinge2Joints for movement complicate the programming process and is more likely to have a mechanical or software issue than simpler locomotive designs seen in robot fish. These simpler locomotive designs in the fish robots allow for higher speed movement than my snake robot which moves precisely but slowly. My robot design has not considered energy consumption and therefore will be a lot less efficient than that of a fish made for efficient high-speed locomotion [19].

Advantages of Biomimetic design of a soft robotic fish for high-speed locomotion

The fish robot will exhibit higher energy efficiency than my snake robot due to it being in aquatic environments which minimises the drag and maximize the effectiveness of the undulating movements. The design of these robots allows for more stability and manoeuvrability in water, taking advantage of fluid dynamics, being able to have control in turbulent conditions. My snake robot would not be able to withstand high wind environments as effectively. Underwater robots like a fish robot with fins usually have more simpler control algorithms than a snake-like robot and therefore will less likely experience software and hardware issues for the locomotion[19].

Disadvantages of Biomimetic design of a soft robotic fish for high-speed locomotion

While effective in water, these fish robots are limited to aquatic applications and therefore their design can only be used for specific applications, whereas the snake robot can be used for many more applications such as environment traversal, environment mapping, surgical applications and even be used in space. The fish robot can become more complex in control algorithms due to the effects of water density, which the snake robot does not have to deal with. In addition, setting up a simulation environment for an underwater robot is a lot more complex as you will need to set up more complex physics such as fluid dynamics and water density [19].

Comparative Conclusion

In this study between the design of my snake robot and biomimetic soft robotic fish, the strengths and weaknesses have been thoroughly examined. The findings highlight the snake robot's ability to adapt and operate in diverse and confined environment due to its modular design and redundancy features. However, its slower movement and higher energy consumption, compared to the robotic fish design suggest an area of improvement. With more time, the focus could be allocated to improving the snake robots speed and efficiency.

Conclusion of Snake Robot Design

The design process of the snake robot demonstrates a journey from conceptualisation through iterative adjustments to refinement and application. The goal was to create a robot that not only mimicked the complex movements of a biological snake but also navigated efficiently through different environments. The iterative design approach was crucial for developing the snake robot, changing the robot from a basic wheeled robot to an advanced biomimetic system.

Incorporating an Inertial Measurement Unit (IMU) at the head and utilising a PID controller allowed for real-time adjustments of the robot's orientation, which allowed navigation without the robot rolling over or deviating from the desired direction. This closed-loop feedback mechanism helps the robot to navigate complex environments precisely. The modular design also significantly increased the robot's ability to manoeuvre through difficult paths and allows for easy maintenance.

Comparatively, the complexities of its control system, while enabling sophisticated

movement, also introduce complexities in programming and maintenance. Furthermore, the energy efficiency and speed of the robot do not match those of underwater biomimetic designs like robotic fish, which benefit from hydrodynamic environments.

In conclusion, this project not only showcases the potential of biomimetic designs in robotic applications but also the complexities. The snake robot, with its advanced sensory and control systems, showcases the potential of robots emulating biological behaviours and how their complex design can be used to simplifier specific tasks.

High-level representation of snake robot shown in Figure 7 in the appendix.

IMPLEMENTATION

Basic Wheeled Robot Controller

In the beginning, I had a basic wheeled robot that had a box shape as the body and two cylinders as hingeJoints on each side, acting as wheels. This robot had a distance sensor on the front to detect obstacles. To implement this, I first included the Webots robot library, "<webots/Robot.hpp>", and initialised the robot,"Robot *snakebot = new Robot()". I also set up an int called timeStep, which gets the basic time step of the robot, "int timeStep = (int)snakebot->getBasicTimeStep();", and is used in future functions and calculations.

Next, I set up the motors using an array of Motors, "Motor *wheelMotor[4]", To use the type Motor, I needed to include the Webots motor library, "<webots/Motor.hpp>" and the C++ array include "<array>". I then added the C++ <string> include so that I could set up an array of the wheel names – std::string wheelNames[4] = {"wheel1","wheel2",etc. }. I then initialised each motor in a for loop that went through each index of the Motor array (up four at this stage) and got the motor of the robot which had the same name as the current wheelName array index. I also initialised the velocity as one, "wheelMotor[i]->setVelocity(1.0)", and their positions to infinity, "wheelMotor[i]->setPosition(INFINITY)", so that the wheels could have continuous motion.

Then, I setup and initialised the distance sensor, "DistanceSensor *distanceSensor = snakebot->getDistanceSensor("distanceSensor")", and set it to have callback every time step, "distanceSensor[i]->enable(timeStep)".

To accomplish basic obstacle avoidance, I followed the Webots User Guide and used Tutorial 6: 4-Wheeled Robot [20]. This tutorial creates a 4-wheeled robot similar to my design at this point but mine has an extra distance sensor. I used their basic avoidance logic within the while loop but adapted it for my robot:

```
"double left_speed = 1.0;
double right_speed = 1.0;
if (avoid_obstacle_counter > 0) {
 avoid_obstacle_counter--;
 left_speed = 1.0;
 right_speed = -1.0;
} else { // read sensors
 double ds_values[2];
 if (distanceSensor->getValue() < 950.0)
  avoid_obstacle_counter = 100;
}
wheelMotor[0]->setVelocity(left_speed);
wheelMotor[0]->setVelocity(right_speed);
wheelMotor[0]->setVelocity(left_speed);
wheelMotor[0]->setVelocity(right_speed);"   [20]
```

Controlling modular wheeled robot

After accomplishing basic obstacle avoidance of a basic 4-wheeled robot, I added 4 more segments of the same size and shape as the body with 4 more extra wheels on each and connected these segments in a modular way using spherical hinge2Joints.

The idea was to control the wheel velocity to cause the hinge2Joints to bend in oscillation and as it progressed to simulate to locomotive slithering movement of snakes. I adjusted the arrays to accommodate for the additional wheel motors and increased the limit on the for loop to 20. I also removed the distance sensors and the obstacle avoidance logic at this point to focus on robot locomotion.

To control the joint angles, I needed to first setup the motors controlling each and done this in the same way as the wheel set up earlier but for both lateral and vertical joint motors putting them separately in an array of lateral joint motors and an array of vertical joint motors. This caused the robot to continuously wrap around itself breaking. To resolve this, I changed the position from INFINITY to zero to stop the motors causing their own torque.

I then created a function, serpentineWheelMovement(double speed, Motor *wheelMotor[], double time), that took in the wheel motor array and the time step and used this to set the wheel motors of each segment to opposite speeds of the last to create torque around the hinge2Joint.

In the main function, I changed the obstacleAvoidanceCounter to a turnCounter and reduced the value from 100 to 50 because the obstacle avoidance, completed a 180 degree turn but I want my snake robot to oscillate in a range of 90 degrees.

Inside the serpentineWheelMovement functions for loop:

```
If (i<4){
        wheelMotor[i]->setVelocity(speed);
}
Else if (i < 8){
        wheelMotor[i]->setVelocity(-speed);
}
Else if (i < 12){
        wheelMotor[i]->setVelocity(speed);
}
Else if (i < 16){
        wheelMotor[i]->setVelocity(-speed);
}
Else wheelMotor[i]->setVelocity(speed);
```

Inside the while loop:

```
  if (turnCounter> 25) { // 45 degrees on way
   int speed = 1
    turnCounter--;
    serpentineWheelMovement(speed, wheelMotor[20], double time)

  }
  else if (turnCounter > 0) { //45 degrees the other way
    int speed = -1
    turnCounter--;
    serpentineWheelMovement(speed, wheelMotors[20], double time)
  }
```

This caused the robot to rock backwards and forwards and did not do what I expected. So I changed the serpentineWheelMovement logic to apply opposite turning velocities to each segment rather than a forwards driving speed. To do this, I added additional conditions inside the if statements for each segment to check if it was a left or right wheel. I used "if (i%2 == 0)" to check this condition and apply opposite speeds, making sure each segment would produce an opposite torque from each other. Unfortunately, I was left with the same result; the robot rocking backwards and forwards.

At this stage I realised that my wheeled robot approach was not going to work and therefore the wheels were consequently removed.

Joint Angles Control

To accomplish snake-like locomotion, I would need to control the joint angles of each joint connecting the segments. Using the joints to control the snake rather than the wheels mimics the natural locomotive behaviour of a snake contracting and expanding its muscles [2].

I created two functions startRobot(Motor *lateralJointMotors, Motor *verticleJointMotors, int numJoints) and stopRobot(Motor *lateralJointMotors, Motor *verticleJointMotors, int numJoints), which used a for loop up until the number of joints, setting every Motors velocity to 10 (startRobot) or zero (stopRobot).

Further looking at the locomotive behaviour of a snake, it is apparent that the sinuous movement of the snake is caused by muscle contractions and expansions of the snake. Therefore, I began investigating different algorithms that could be used and I came across work by Hirose and Umetami, which used sinusoidal formulae to control snake-like movements. Hirose and Umetami's formulae were too difficult for me to fully grasp and would not work for my robot anyway [5].

I looked further into physics formulas of sine waves to try and come up with my own algorithm similar to that of Hirose and Umetami's. I came across simple harmonic motion, an oscillatory motion that revolves around an equilibrium point. Many systems such as pendulums and atoms exhibit SHM, and its purpose is to describe the movement of oscillating objects. [21]. Here is the algorithm for displacement of this point: $X = A\sin(wt)$ [21]. $W = 2 \times Pi \times frequency$ [21]

Using SHM formula, I created my own function to control the displacement of the joint angles within the while loop:

```
Void setJointAngles()
for (int i = 0; i<4; i++){
double jointAngleLateral = amplitudeLateral * sin(2.0 * M_PI * frequency * timeStep);
```

lateralJointMotors[i]->setPosition(jointAngleLateral);
}

For this to work I needed to introduce an amplitudeLateral which was set to 0.5 and to be able to use M_PI I needed to include C++'s math library, <cmath>.

Using simple harmonic motion for the snake joint motors caused the snake robot to oscillate on the spot in a sinuous way mimicking that of snake's movement but did not yet create any motion forwards or sideways.

To tackle this, I introduced a phase shift to each joint motor so that the forces were not equal and so the robot could create a driving force in the direction of the head of the snake [22]. I first needed to set up the phase offsets for each angle: double lateralPhaseOffsets[] = {M_PI/2.0, M_PI/2.0 , M_PI/2.0 , M_PI/2.0 }
jointAngleLateral = amplitudeLateral * sin(2.0 * M_PI * frequency * timeStep + lateralPhaseOffset[i]).
This was implemented in the setJointAngle function, the parameters were adjusted accordingly.

Adding the phase offsets, the robot began to have motion, while keeping its sinuous oscillations, but the motion was lateral instead of the robot moving forards. To deal with this I used the exact same algorithms as the lateral joint motors but done this for the vertical joint motors also:

double jointAngleVertical = amplitudeVertical * sin(2.0 * M_PI * frequency * timeStep);
VerticalJointMotors[i]->setPosition(jointAngleVertical);

Implementation of control over the vertical joints allowed the robot to move with a slight forwards motion but was turning to the right. I tried to manually adjust both the lateral and verticle phase offsets and managed to get the robot to turn left, turn right, move left laterally (sidewind), and move right (laterally) but could not get the right phase offsets to get the robot to move directly forwards with sinuous movement. The design of the robot was changed at this point, to resemble that of a biological snake using cylindrical segments rather than rectangular boxes. This allowed slightly more movement but it was not a drastic change.

Fixing Forwards Movement

The robot could not progress forwards at this point, but I knew the phase offsets were pivotal for controlling how the snake moved and turned. To be able to monitor the error in the direction the robot was going compared to where it was intended, I needed to implement an Inertial Measurement Unit (IMU) device in the head of my snake to be able to detect the direction the head was facing.

First, I setup and enabled IMU device:

```
    InertialUnit *imu = snakebot->getInertialUnit("imu");
    imu->enable(timeStep); // get imu reading every timestep
```

Next, I removed the control of the joints from the while loop to a function, “moveForwards(Motor *lateralJointMotors[], Motor *verticleJointMotors[], double time, int numJoints,InertialUnit *imu)”. In this function, I set a yaw float variable as the current yaw reading from the imu device: “float yaw = imu->getRollPitchYaw()[2]”. This yaw value would be used against the desired value (0 for head being straight) that kept the snake head facing forward.

The other locomotive behaviours turnLeft, turnRight, moveLeft, and moveRight were then created as functions in the same way as the moveForwards function but with the correct phase offsets for their movement type.

I attempted to transition between turning right and turning left to straighten the snake robot because the moveForwards function had some sort of forwards motion but would turn to the left as it progressed. I used a constant value, YAW_THRESHOLD, to have a range the head could oscillate in without being considered turning. Here is the implementation:

```
// If yaw error exceeds the threshold, call turning functions to correct it
   if (yawError > YAW_THRESHOLD) {
      cout << "turning right, to straighten" << endl;
       turnRight(lateralJointMotors, verticleJointMotors, time, numJoints);
   } else if (yawError < -YAW_THRESHOLD) {
       cout << "turning left, to straighten" << endl;
       turnLeft(lateralJointMotors, verticleJointMotors, time, numJoints);
   /} else {
      // Normal forward movement using sinusoidal joint angles
```

Using this implementation helped the robot drive straight but was not good enough and would still steer off to the side. This part was commented out.
I then tried to introduce a Proportional-Integral-Derivative (PID) controller, to dynamically adjust the phase offsets based on the error of the snake from its intended path. First, I added global variables for the different PID gains [23]:

```
double yawKp = 0.1; // Proportional gain
double yawKi = 0.01; // Integral gain
double yawKd = 0.00; // Derivative gain
```

Then I implemented the PID output algorithm within the moveForwards function:

```
  // error in the angle of the neck of the snakebot
  double yawError = desiredYaw - yaw;

 // Integral of error (sum of Error)
 yawIntegralError += yawError * (time)/1000; // need ms->s conversion

 // Derivative of error (difference in error)
 double yawDerivativeError = (yawError - yawPreviousError) / (time / 1000.0); // need ms->s
conversion
```

```
//PID output config
 double yawPIDOutput = yawKp * yawError + yawKi * yawIntegralError + yawKd *
yawDerivativeError;

  // //Save current error as previous error for next iteration
  // yawPreviousError = yawError;
```

This algorithm is placed before the jointAngleLateral = amplitudeLateral * sin(2.0 * M_PI * frequency * time + lateralPhaseOffsets[i]) line and the yawPIDOutput is added inside this line after the lateralPhaseOffsets[i] inside the brackets to further adjust the offsets.

Next, I began fine tuning the gain values until the robot began maintain its head with a yaw of zero. At first, the robot would over adjust at certain points due to the PIDOutput causing the robot to wrap around itself. To fix this, I implemented limits to stop this:

```
  //stop snake wrapping around, ensure shortest turning angle
     if(yawError > M_PI) yawError -= 2.0 * M_PI;
     if(yawError < -M_PI) yawError += 2.0 * M_PI;
```

This fixed the issue and done its purpose as it was implemented but did not get the robot to move forwards, instead, the robot oscillated in smaller angles keeping the head straight but with no movement. It was evident, this implementation of a PID controller was not going to work and was thus removed.

Afterwards, I added 5 touch sensors, one to each segment to detect the impact collision with the ground. These sensors were set up and were called back every time step :

```
// set up touch sensors
  TouchSensor *forceSensors[5];
  std::string forceSensorNames[5] = {"neckTouchSensor", "segment1TouchSensor",
"segment2TouchSensor", "segment3TouchSensor", "segment4TouchSensor"};
  for (int i = 0; i < 5; i++) {
    forceSensors[i] = snakebot->getTouchSensor(forceSensorNames[i]);
    forceSensors[i]->enable(timeStep);
  }
```

I then printed the values on each segment to the terminal to observe any patterns in the force applied at different vertical offsets or vertical amplitudes. I could not get any valuable information from this, so I decided to use a PID controller to adjust the vertical amplitudes at the joint motors in relation the amount of force that was being applied at each joint.

I created a function to adjust the vertical offsets. This function takes in an array of doubles (verticlePhaseOffsets), an array of Touch sensors (force sensors), and an int of the number of joints. It creates a forceReading array of type double with a capacity of the number of joints plus one (one more segment than joint) and a double of the total force initialised as zero. Then it loops through the number of joints, reading the force value on the segment with the same index. If this value is nonzero it is added to the total force. It then loops again but up until the number

of joints this time. It divides the current force on the joint by the total force to get the ratio, if it is the last joint average the current force on the last two segments before getting the ratio.
The vertical phase offset is then adjusted in proportion to ratio of force:

```
verticalPhaseOffsets[i] += (M_PI/2.0) * (1.0 - ratioOfForce)
```

This was the added inside the moveForwards function before the vertical joint angles are set but did not get the robot to progress and just got the robot to start bouncing and rolling, so was removed.

Hunting

The robot would not move forwards directly and would steer off still, despite attempts to resolve this, so a different approach was attempted. Instead of fixing the forward motion, I tried implementing object detection and get the robot to adjust left and right based on the error from the camera. This implementation was supposed to be a later objective on the kanban to-do list, however, due to the bottleneck of getting the robot to progress forwards, this was used for maintaining a path.

First, I added 4 cameras around the robot in the middle segment extended on a pole so they would not face the ground and could see at a greater range and set these up in the main function to have a callback every time step. For object recognition, I needed to enable recognition withing the cameras:

```
camera[i]->recognitionEnable(timeStep);
```

I could not get the snake robot to detect objects of any colour even after adjusting parameters, so the idea of hunting and using this as a way of controlling the snake's movement did not work.

Final phase offset adjustments

I began manually adjusting the phase offsets again to try and get the forward motion. Eventually, I had to robot moving forwards, but it would roll over as it progressed, therefore, another PID controller was introduced in the same way as before but instead of the yaw error adjusting the lateral phase offsets, the pitch is used to control the vertical offsets to stop the robot from rolling over. The pitch was needed in the implementation instead of the roll because the IMU device was placed incorrectly in the head of the snake.

This IMU device to control the pitch worked in stopping the robot from rolling over but the adjustment in the phase offsets to do so, also stopped the robot from locomoting in the same way.

After additional adjustments of the offsets to try and get the robot to progress straight again, the PID controller gains needed adjusting and was not needed for specific offsets. The final value for move forwards was found after trial and error with no need of the PID controllers adjustment.

Obstacle Avoidance with Integrated sensory system

After accomplishing the basic locomotion function for move forwards, move left, move right, turn left and turn right, the next task was to implement obstacle avoidance using sensors to detect objects and trigger specific movement logic in relation to these conditions.

First, I added 4 range finders around the snake robot one at the head and tail and one either side of the middle segment, so they covered as much areas as possible. I used range finders instead of distance sensors because the distance sensors earlier were not detecting objects from a distance far enough away that would allow the snake robot to change its oscillations to avoid the obstacle. I set up the range finders in the same way as the other devices and enabled it every timestep.

I then created a function, getAverageDistance(const *RangeFinder), to take the range values the range finders gets and averages them into one value. This used the getWidth() and getHeight() methods from the range finder library to find the resolution of the range finder. It then initialises a count (int variable) and a sum (int variable) as zero. A for loop then begins going from zero the resolution value, adding the value to the sum and adding one to the count if the value received from the range finder is nonzero. This function would then return the average value.

Serpentine avoidance

I then began the obstacle avoidance logic for serpentine locomotion, serpentineAvoidance(RangeFinder *rangeFinder[], DistanceSensor *distanceSensor[], Motor *lateralJointMotors[], Motor *verticleJointMotors[], double time, int numJoints), checking the average reading of sensors and performing specific movements based on the conditions the snake robot is in.
Heres the pseudo code:

```
If there was an object in front – turn right
Else If object to left – move right
Else If object to right move left
Else {
        move forwards
}
```

This worked for obstacle avoidance in most cases, but the robot would get caught by some objects approaching at specific blind spot angles. To resolve this, I added 4 distance sensors, two at the head and two at the tail, pointing out into the robots' blind spots. I figured out how to increase the range, so my initial issue with them was resolved.

The logic thus changes:

```
If object in front – turn right (uses range finders)
Else If object in front left - turn right (uses distance sensors)
Else If object in front right – turn left (uses distance sensors)
Else If object to left – move right (uses range finders)
Else .If object to right – move left (uses range finders)
Else {
        startRobot()
        move forwards()
}
```

The snake robot could now move forwards and obstacle avoid using a serpentine gait, however, the speed that the robot moved at was very slow. I was confused at first because the basic locomotive function worked at a much faster speed, so I attempted at adding a delay to how often the call to the sensor values occured.

```
  // delay sensor update so the robot can move smoothly
  if (sensorUpdateCounter++ % updateInterval == 0) { // update every x timeSteps
   averageDistanceFront = getAverageDistance(frontRangeFinder);
   averageDistanceLeft = getAverageDistance(leftRangeFinder);
   averageDistanceRight = getAverageDistance(rightRangeFinder);
   averageDistanceBack = getAverageDistance(backRangeFinder);
   leftDS = distanceSensor[0]->getValue();
   rightDS = distanceSensor[1]->getValue();
   // Reset counter
   sensorUpdateCounter = 0;

 }
```

This was placed before the conditional logicto ensure the values that are being used as conditions only update every update interval. UpdateInterval and sensorUpdateCounters are created globally so they can be accessed and utilised in this function.

This implementation allowed the snake robot to move slightly faster, but it did not solve the issue of the robot moving slower than intended.

Sidewinder Avoidance

The next goal was to accomplish obstacle avoidance with a sidewinding gait. The functions for movement were already done, so just rearrangement of the serpentine logic was needed to get the snake to be able to obstacle avoid while locomoting with a sidewinding gait. The right side of the snake if now considered the leading part of the snake, therefore the head is the left and the tail is the right. The pseudo code is written with the sidewinder orientation.

Here is the pseudo code:

```
If object in front left - turn right (uses distance sensors)
Else If object in front right – turn left (uses distance sensors)
Else If object too close to right – turn left (uses range finders)
Else moveRight (this is the move forwards function in the sideWinder case)
```

This had the snake sidewinding and avoiding obstacles but there were a few cases where the robot would get stuck such as the head or tail in the corner so a few more conditional statements were added:

```
If tail of snake stuck in corner – turn left (uses distance sensors)
Else If head of snake stuck in corner – turn right (uses distance sensors)
Else If object in front left - turn right (uses distance sensors)
Else If object in front right – turn left (uses distance sensors)
Else If object too close to right – turn left (uses range finders)
Else moveRight (this is the move forwards function in the sideWinder case)
```

At this point, obstacle avoidance for serpentine and sidewinding gaits were working, so the next step was to add maze traversal logic to the snake to see if the movements are precise enough to move through confined spaces.

Maze Traversal

After accomplishing obstacle avoidance, the next task for both types of snake locomotion was to be able to traverse through a maze constructed with walls. The main adjustment between the obstacle avoidance and maze traversal, is using the walls to follow a path instead.

Serpentine Maze Traversal

distanceDifference = averageDistanceLeft - averageDistanceRight;

```
If distance difference < -TURN_THRESHOLD          - turn right
Else If distance difference > TURN_THRESHOLD      - turn left
Else If left turn available                       - turn left
Else If right turn available                      - turn right
Else                                              - move forwards
```

Although the obstacle avoidance worked, the turning values for the sidewinding maze traversal were not precise enough to get round the maze.

Sidewinder Maze Traversal

distanceDifference = averageDistanceFront – averageDistanceBack;

```
If distance difference < -TURN_THRESHOLD          - turn right
Else If distance difference > TURN_THRESHOLD      - turn left
Else If left turn available                       - turn left
Else If right turn available                      - turn right
Else                                          - move right  (this is move forwards for sidewinding)
```

Although the obstacle avoidance worked, the turning values for the sidewinding maze traversal were not precise enough to get round the maze, however, sidewinding worked better due to the angle it approached the corner at.

Conclusion of Implementation

In conclusion, the evolution of the snake robot from a simple wheeled robot to a sophisticated serpentine system is a prime example of the iterative nature of robotics and how it combines theoretical knowledge with practical applications. The project's initial focus on basic obstacle avoidance utilising wheeled locomotion established a fundamental knowledge base for sensor integration and motor control. But moving to a snake-like design, was a major step in the direction of more realistic locomotion, with advanced control mechanisms including joint angle modulation and sinusoidal movement algorithms.

While the integration of IMUs and PID controllers highlighted the difficulties and complexities of creating stable, directed movement, the use of Simple Harmonic Motion (SHM) and phase offsets demonstrated a sophisticated way to imitating the sinuous locomotion of snakes. The integration of touch sensors and advanced sensory systems highlighted a deeper sophistication in the robot's interaction with its surroundings, even despite failures with PID tuning and the limitations of wheeled segments.

After modifications and improvements, such as the integration of integrated sensory systems for obstacle avoidance, the project showcased the robot's potential for intricate navigation tasks like navigating through mazes. From basic wheeled movements to complex serpentine manoeuvring, the iterative design process highlights the need of adaptability and ongoing development in engineering designs while also reflecting the difficulties encountered in robotics.

This project highlights the dynamic interaction between design theory and practice in robotics and provides an example of how theoretical concepts may be turned into creative, workable solutions. The evolution of a simple wheeled robot into an intricate serpentine mechanism emphasises the value of flexibility and iterative improvement in engineering projects. The insights gained from the difficulties and solutions through this project pave the way for future projects and highlight the possibility of more complex and successful biomimetic solutions.

TESTING

Component Testing

The testing phase began with testing each component on their own to make sure they were functioning correctly.

Locomotion Using wheels

The testing phase began when the robot still was driven by wheels but had already been evolved into a modular design.

The first idea was to apply opposite directional force to each segment and inverse the speeds after a specific period of time, however, this method was ineffective and caused the robot to just rock backwards and forwards. I then tried to add more velocity to see if this helped but just caused the rocking motion to occur over a greater distance.

The next idea was to make each segment have opposite turning directions as their adjacent segments, but this produced a rocking back and forth motion. I considered that the robot may not be twisting like a snake because there was no velocity applied to the joint motors and maybe this locked them. However, this caused the segments to wrap around each other and break the robot.

Locomotion Using Joint Motors

The thing to be tested was the effect of a change in amplitude in the sinuous movement of the snake robot. Changing the lateral amplitude did not change how the snake moved, it was still oscillating but stationary at this point.

Next, I started manually adjusting the different phase offsets for each joint motor in both the lateral and vertical axis and running the simulation to observe the effects this had on the robot's movement (see Figure 1 & Figure 2 for examples).

Sensor Testing

The first sensor to be tested was the IMU device, which would be necessary for the yaw and pitch PID controllesr. This was a simple test to see if the IMU sensor was receiving roll pitch and yaw values. This was done using a debug print statement in the while loop.

The next sensor to be tested was the force sensor, which would be necessary for the adjustVerticlePhaseOffset function that adjusted the vertical phase offsets of the joint motors in proportion to the force applied on each segment. This test was done by simply printing the force values on the bottom of each segment to the terminal. This information was not valuable.

The adjustVerticlePhaseOffset function was then tested but this had no affect on whether the robot would move forwards or not so was then ignored.

After snake locomotion was accomplished, the next goal was to integrate sensors so that the snake could avoid different obstacles.

The main sensor that is used for obstacle avoidance and needed testing was the range finder, to do this, I first got the robot to stop, using the stopRobot() function, when an object was detected from a threshold distance. I adjusted the value and run the simulation iteratively until I got threshold values that would stop the robot when an object was detected from a distance far enough away.

The next sensor for obstacle avoidance and that needed testing was the distance sensor, which was used for detecting the blind spots the range finders had. These were tested in the same way as the range finders, however, no change in value changed the distance the object would be detected from. To solve this, I changed the range values inside the device in the scene hierarchy.

The range on the distance sensors were to far now and were causing the robot to stop because it was detecting the ground from a few metres in front of the snake robot. I then manually adjusted the threshold value for detecting an object until the ground was no longer detected.

PID Controller Testing

After implementing the PID controller, adjustment and testing of the yaw PID gain values were needed to be able to keep the robot head straight (see Figure 3). The gain values necessary for this were found by trial and error with incremental adjustments and watching the result when the simulation was run but this did not get the robot to move forwards as was intended; the PID controller was consequently removed.

The second PID controller was adjusting the vertical phase offsets using the pitch (pitch instead of roll because imu placed in wrong orientation). The intention behind this PID controller was to stop the robot from rolling over, while it progressed forwards (offset values were found to go forwards but the robot rolled over and stopped progressing). The gain values were adjusted, and the affects were observed when the simulation was run, the robot did eventually stop rolling over but this also changed the movement and was no longer headed straight (see figure 4).

The PID gain values were all zero for the robot moving forwards in the final implementation.

System Integration Testing

After accomplishing locomotion and testing each individual component the robot needed to be tested as an entire system.

This was done by running the obstacle avoidance and maze traversal logic and observing the affects in the simulation. The first value that needed testing and adjusting again was the object detection threshold to allow the snake robot enough room to turn or move away from an object without collision (see Figure 5).

This worked but the robot moved at very slow speed when running the obstacle avoidance logic so extra logic to only update the sensor values every x amount of timesteps. This value was manually adjusted, and the effects observed but there were no changes in the speed of the robot.

The next thing to be tested was the maze traversal logic, and the WALL_THRESHOLD constant needed adjusted so the robot would detect when there was a turn available this was done by changing the value and rerunning the simulation to observe the affects (see Figure 6). The phase offsets for the turning values did not produce turns precise enough to go round the maze. However, side winding proved to be superior in traversing the maze because it approached the corners at a different angle compared to the serpentine movements.

CRITICAL EVALUATION

This project embarked on the ambitious task of creating a biomimetic snake robot capable of emulating the intricate locomotive behaviours of biological snakes. The goal was to develop a snake-like robot that could mimic the movements of real snakes and then to be able to navigate complex environments more effectively than traditional robot designs by using a series of segments and joints controlled by advanced algorithms. This project delving in the study of biomimetics aimed to bridge the gap between theoretical robotics and practical applications such as search and rescue, medical assistance, and hazardous environment navigation.

Approach and Techniques

The initial design phase involved constructing a basic 4-wheeled robot to gain a foundational understanding of robotic devices and how they are controlled within the Webots simulation environment. This design quickly evolved to a modular segmented robot connected by joints and driven by wheel motors, however, these wheels proved to be ineffective at mimicking the locomotive behaviour of biological snakes so were removed and the focus shifted to controlling the joints through algorithms designed to replicate the sinuous motion of snakes, making use of biomimetic ideas.

This project encountered significant challenges that caused the need for numerous readjustments. This iterative process, while common in designing robots, highlighted an underestimation of the complexity involved in accurately replicating biological snake locomotion. Each iteration of design refinement improved the ability to mimic the locomotion of biological snakes but also indicated the need for more background research into biological snake motion, prior to designing the robot.

Tools and Technology

The Webots simulation environment was selected because they offered a stable platform for testing the robot in a controlled setting, allowing quick development through iteration without the expenses of physical models. The project required real-time processing, and C++ programming provided the required control over hardware for algorithm implementation.

The integration of various sensors, including IMUs and PID controllers for orientation and touch sensors for interaction with the environments, demonstrates sophisticated planning and technological application of different devices. However, the project's aspirations were lowered by technical limitations, such as difficulties controlling forward movement with the use of PID controllers and the difficulties of integrating the sensors simultaneously controlling all the joints to produce snake-like locomotion in real-time to be able to avoid obstacles.

Achievement of Objectives

The project succeeded successfully in achieving several of its initial objectives such as developing a snake-like robot capable of mimicking multiple snake gaits, such as serpentine and sidewinding. However, challenges reoccurred in achieving stable and directly forward motion. The robot's ability to navigate complex environments was sufficient in simpler tasks but lacked in more structured scenarios due to the difficulties in precise control and sensor integration. The difference in performance when tasked with sensor integration highlighted the complexity of translating biomimetic principles into working robotic movements in various scenarios.

Quality of Results

The robot demonstrated the potential of incorporating biological concepts into robotic designs by its ability to navigate around obstacles using a biomimetic method. However, the inconsistent performance, especially the slower mobility in the more complex tasks like obstacle avoidance and maze traversal, indicated problems with sensor responsiveness and algorithm efficiency.

Innovation and Creativity

This project showcases high level innovation, especially in the use of simple harmonic motion formulas to control the joint angles, and the application of phase-shifts to manage directional control. These solutions to obtaining snake-like locomotion, while theoretically sound, lacked in getting the robot to move properly and highlighted the gap between theoretical robotics and real-world implementation.

Skills Enhancement

This project was a big learning experience for developing practical skills in robotics, programming, sensor integration and research. The kanban process was particularly valuable in teaching problem-solving and adaptability when experiencing bottlenecks like controlling forward motion.

Knowledge Gained

An abundance of knowledge was gained in the field of biomimetic robotics, especially enhancing robotic functionality and design based on biological principles. This project also provided insight into the difficulties associated with sensor integration for real-time control algorithms. Through designing the robot, I learned that snakes locomote with different gaits and how complex their movements are.

Challenges and Solutions

The main challenge faced was achieving a stable and controlled forward motion, which was essential for locomoting with a serpentine gait. Solutions involved numerous iterations of adjusting the phase offsets of each joint angle based on feedback from each test run. The next challenge was obstacle avoidance and maze traversal which required numerous iterations in the design, such as adjusting the height and angle of each sensor and refining the control algorithms based on new problems found in test runs. These solutions, while not completely avoiding all obstacles, led to incremental improvements that were important for understanding of the project.

Ethical Issues

This project maintained an ethical approach by focusing on simulations, which can be used to reduce risk before any real-world applications. Future projects could consider a detailed assessment of the ecological impacts of deploying a snake-like robot in natural settings, making sure that biomimetic designs do not disrupt the habitats of real biological snakes.

Social Impact

The potential application of the snake robot has many potential positive social impacts such as being used for search and rescue, medical assistance, and hazardous environment navigation. However, broader implications, like replacement of employees in sectors likely to adopt these technologies, need further exploration to ensure that the developments contribute positively to society without unintended negative consequences.

Conclusion

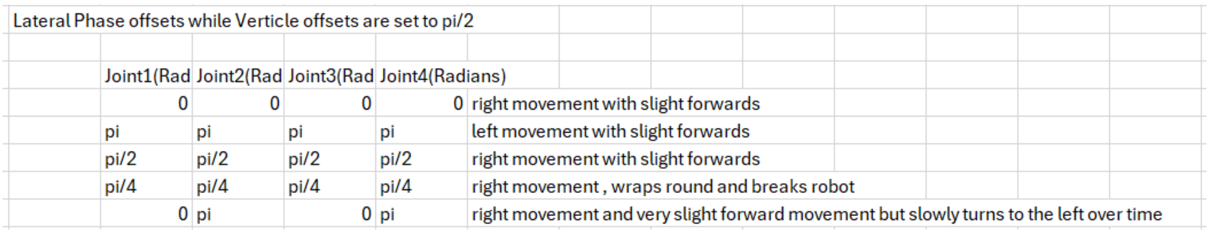
This project highlights the dynamic and complex nature of integrating theoretical knowledge with practical application in robotics. The iterative process from a simple wheeled robot to an advanced biomimetic system illustrates both the potential and the challenges of biomimetic robotics.

Despite not achieving flawless functionality, particularly in maze traversal and maintaining speeds while using an integrated sensory system, the project made significant advancements in understanding and applying complex control systems and sensor integration.

Future work should aim to refine these systems to improve the adaptability and efficiency of the robot and maybe introduce a state-machine or algorithms to switch between the different gaits of a snake. Future projects could also potentially incorporate machine learning algorithms to improve decision-making capabilities.

Appendix:

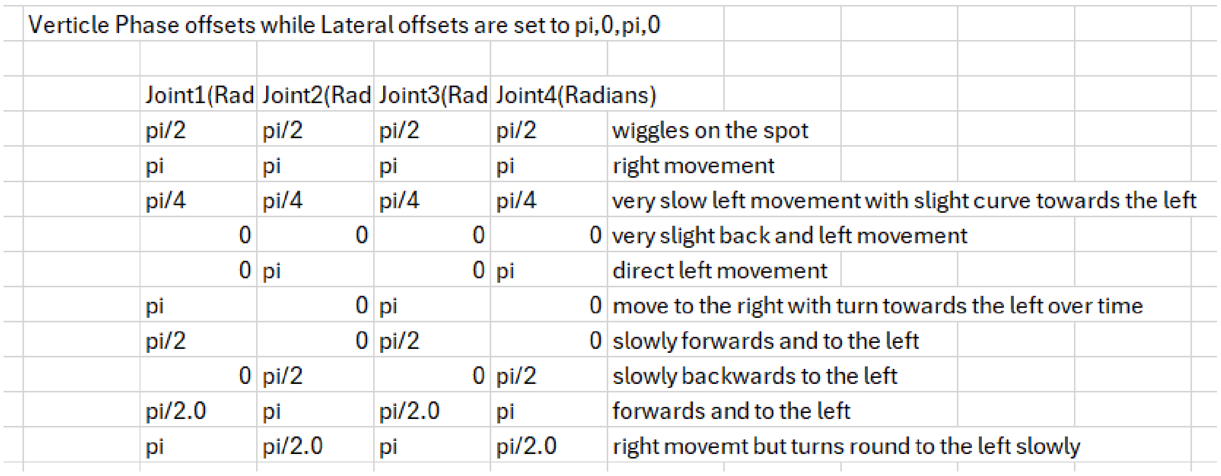
Figure 1 – lateral phase offset adjustments



Lateral Phase offsets while Verticle offsets are set to pi/2

Joint1(Rad	Joint2(Rad	Joint3(Rad	Joint4(Radians)	
0	0	0	0	right movement with slight forwards
pi	pi	pi	pi	left movement with slight forwards
pi/2	pi/2	pi/2	pi/2	right movement with slight forwards
pi/4	pi/4	pi/4	pi/4	right movement , wraps round and breaks robot
0	pi	0	pi	right movement and very slight forward movement but slowly turns to the left over time

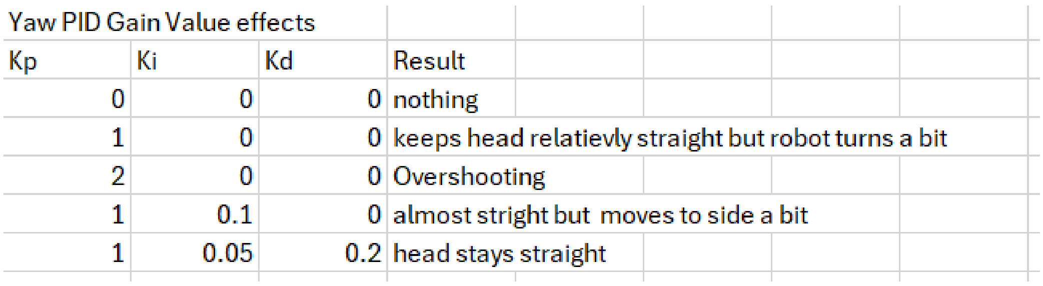
Figure 2 - vertical phase offset adjustments



Verticle Phase offsets while Lateral offsets are set to pi,0,pi,0

Joint1(Rad	Joint2(Rad	Joint3(Rad	Joint4(Radians)	
pi/2	pi/2	pi/2	pi/2	wiggles on the spot
pi	pi	pi	pi	right movement
pi/4	pi/4	pi/4	pi/4	very slow left movement with slight curve towards the left
0	0	0	0	very slight back and left movement
0	pi	0	pi	direct left movement
pi	0	pi	0	move to the right with turn towards the left over time
pi/2	0	pi/2	0	slowly forwards and to the left
0	pi/2	0	pi/2	slowly backwards to the left
pi/2.0	pi	pi/2.0	pi	forwards and to the left
pi	pi/2.0	pi	pi/2.0	right movemt but turns round to the left slowly

Figure 3 – yaw PID gain adjustments



Yaw PID Gain Value effects

Kp	Ki	Kd	Result
0	0	0	nothing
1	0	0	keeps head relatievly straight but robot turns a bit
2	0	0	Overshooting
1	0.1	0	almost stright but moves to side a bit
1	0.05	0.2	head stays straight

Figure 4 – pitch PID gain adjustments

Pitch PID Gain Value effects			
Kp	Ki	Kd	Result
0	0	0	nothing
1	0	0	nothing
2	0	0	nothing
5	0	0	overshooting
4	0	0	oscliatting side to side after a while
3	0.5	0.2	stops rolling over after a while but does not move forwards

Figure 5 – obstacle detection value adjustments

Object detection threshold adjustment effects	
Distance	Result
0.5m	object hit
1.5 m	object avoided from far away
1.0m	object avoided but still a bit far
0.75m	objected avoided but if object near another object issues may occur
0.9m	sweet spot for obstacle avoidance

Figure 6 – wall detection value adjustments

Wall detection threshold	
Distance	Result
0.25m	not detecting walls
0.5m	detcting walls but over adjusting not allowing forwards movement
1m	sweet-spot for wall following

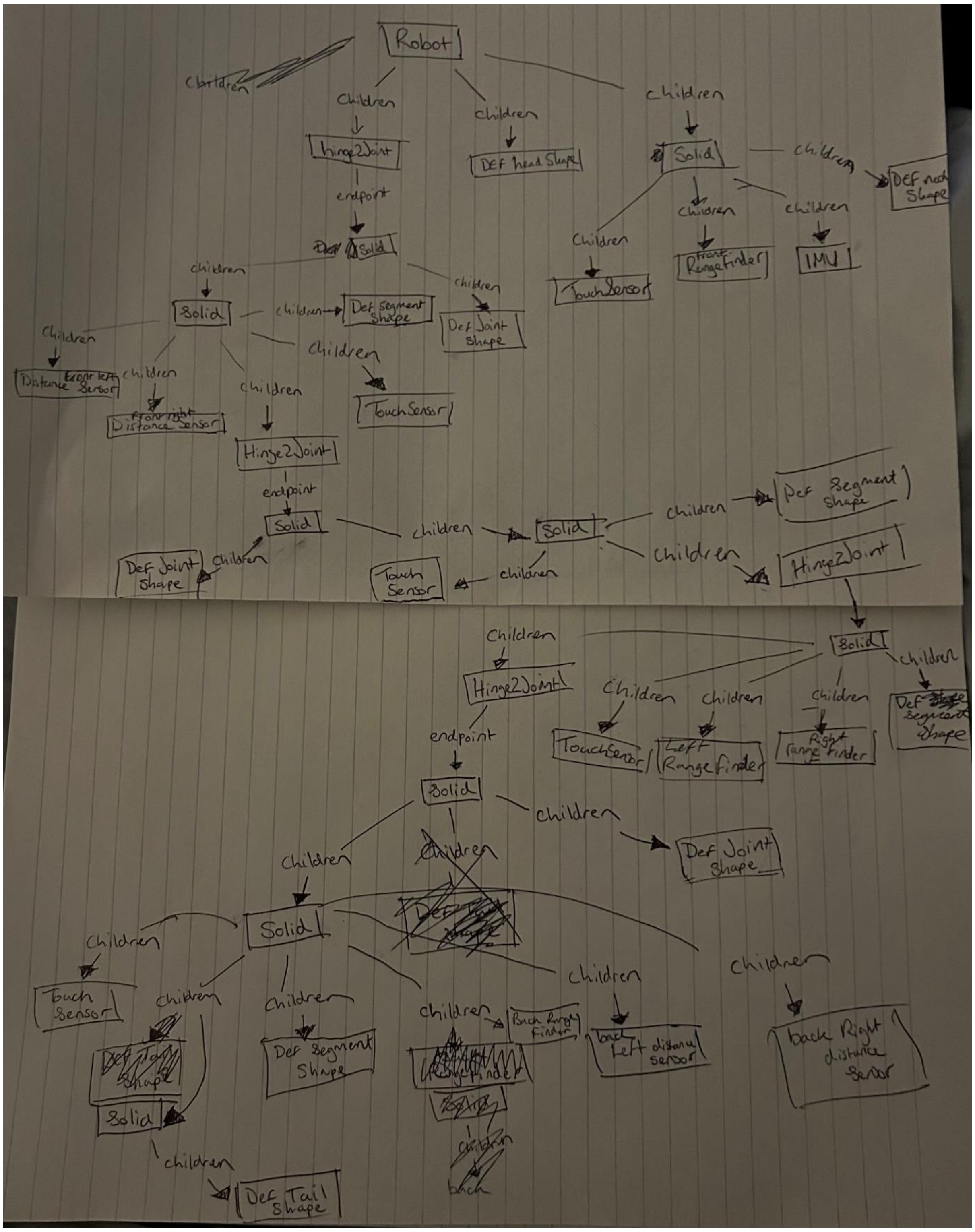


Figure 7 – High-level representation of snake robot

References:

[1] Liu, Y. and Sun, D. (eds.) (2012) Biologically Inspired Robotics. CRC Press. Available at: https://library.oapen.org/handle/20.500.12657/41667 (Accessed: 14 March 2024).

[2] Behavior Patterns. (n.d.) Snake Locomotion. Available at: https://snakelocomotion.wordpress.com/behavior-patterns/ (Accessed: 15 March 2024).

[3] Barral, M. (2018) How does a snake move? BBVA OpenMind. Available at: https://www.bbvaopenmind.com/en/science/mathematics/how-does-a-snake-move/ (Accessed: 15 March 2024).

[4] Lacy Perry. (2004) Snake motion animation [Online image]. In 'How Snakes Work', HowStuffWorks. Available at: https://media.hswstatic.com/eyJidWNrZXQiOiJjb250ZW50Lmhzd3N0YXRpYy5jb20iLCJrZXkiOiJnaWZcL3NuYWtlLW1vdGlvbi5naWYiLCJlZGl0cyI6eyJyZXNpemUiOnsid2lkdGgiOjI5MH0sInRvRm9ybWF0IjoiYXZpZiJ9fQ== (Accessed: 15 March 2024).

[5] Dowling, K.J. (1997) Limbless Locomotion: Learning to Crawl with a Snake Robot. Doctor of Philosophy in Robotics. Carnegie Mellon University. Available at: https://www.societyofrobots.com/robottheory/limbless_locomotion.pdf (Accessed: 20 March 2024)

[6] Begos, K. (2012) 'Scientists, doctors use snake robots for surgery, rescues, and exploration', Associated Press. Available at: https://news.yahoo.com/news/scientists-doctors-snake-robots-surgery-rescues-exploration-080012057.html (Accessed: 1 April 2024).

[7] Orekhov, A.L., Abah, C. and Simaan, N. (2019) 'Snake-Like Robots for Minimally Invasive, Single Port, and Intraluminal Surgeries'. Available at: https://arxiv.org/abs/1906.04852 (Accessed: 1 April 2024).

[8] Johns Hopkins University, CiiS Lab (2019) Snake Robot. Available at: https://ciis.lcsr.jhu.edu/doku.php?id=research.snake_robot (Accessed: 1 April 2024).

[9] OC Robotics (no date) Home page. Available at: https://www.ocrobotics.com/ (Accessed: 1 Aril 2024)

[10] Spice, B. (2015) 'Carnegie Mellon's Snake Robots Learn To Turn By Following the Lead of Real Sidewinders', *Carnegie Mellon University News*, 23 March. Available at: https://www.cmu.edu/news/stories/archives/2015/march/snake-robots-follow-sidewinders.html (Accessed: 1 April 2024).

[11] Radigan, D. (Year of publication). "Kanban." Atlassian. [Online] Available at: https://www.atlassian.com/agile/kanban (Accessed 10 March 2024).

[12] Toyota Blog. (2013). "Kanban – Toyota Production System guide." [Online] Available at: https://mag.toyota.co.uk/kanban-toyota-production-system/ (Accessed 10th March 2024).

[13] Kanban Blog. (2014). "Kanban and other techniques for developing better software faster." [Online] Available at: https://kanbanblog.com/ [Accessed 10th March 2024].

[14] Eby, K. (2016). IC Simple Kanban [Image]. Available at: https://www.smartsheet.com/sites/default/files/styles/900px/public/IC-simple-kanban-c.webp?itok=atpSsOOd [Accessed 20 April 2024].

[15] SBG Systems. IMU - Inertial Measurement Unit. Available at: https://www.sbg-systems.com/inertial-measurement-unit-imu-sensor/ [Accessed 4 April 2024].

[16] Omega, a DwyerOmega brand. What is a PID Controller? Available at: https://www.omega.com/en-us/resources/pid-controllers [Accessed 4 April 2024].

[17] Yuldashev, T., & Solovev, A. (2024). Basics of PID Controllers: Design, Applications, Advantages & Disadvantages. Available at: https://www.integrasources.com/blog/basics-of-pid-controllers-design-applications/ [Accessed 4 April 2024].

[18] Yuldashev, T., & Solovev, A. (2024). PID Closed Loop Control System [Image]. Available at: https://www.integrasources.com/media/files/PID_closed_loop_control_system.png.webp [Accessed 4 April 2024].

[19] Van den Berg, S.C., Scharff, R.B.N., Rusák, Z. and Wu, J. (2022) 'OpenFish: Biomimetic design of a soft robotic fish for high speed locomotion', *HardwareX*, 12, e00320. Available at: https://www.sciencedirect.com/science/article/pii/S2468067222000657 (Accessed: 22 April 2024).

[20] Webots. (2023). Tutorial 6: 4-Wheeled Robot (60 Minutes). Cyberbotics. Retrieved from https://cyberbotics.com/doc/guide/tutorial-6-4-wheels-robot?tab-language=c++. (Accessed 12 March)

[21] Kaylegian-Starkey, K., Wood, D. & Iverstine, W. n.d., 'Simple Harmonic Motion | SHM Meaning, Equation & Examples', AP Physics 1: Exam Prep, Study.com, Available at: https://study.com/learn/lesson/simple-harmonic-motion-smh.html. (Accessed 2 April 2024)

[22] Kaylegian-Starkey, K., Wood, D. & Iverstine, W. n.d., 'Simple Harmonic Motion | SHM Meaning, Equation & Examples', AP Physics 1: Exam Prep, Study.com, Available at: https://study.com/learn/lesson/simple-harmonic-motion-smh.html. (accessed 2 April 2024)

[23] Wilderness Labs. n.d., 'Standard PID Algorithm', Wilderness Labs Developer, Available at: https://developer.wildernesslabs.co/Hardware/Reference/Algorithms/Proportional_Integral_Derivative/Standard_PID_Algorithm/. (Accessed 4 April 2024)

[24] Dorminey, B. (2013, September 30). Robotic Snakes Slither Their Way Into Ancient Archaeology. *Forbes*. Available at: https://www.forbes.com/sites/brucedorminey/2013/09/30/robotic-snakes-sliver-their-way-into-ancient-archaeology/?sh=51751f1a79ea (Accessed 1 April 2024)