



MECHATRONICS FINAL REPORT

Modular Robotic Snake

Sara Djoudi C1617272 - Jack Frampton C1428611 - Nor Binti Mohd Ridzuan C1622148
Angelina Murphy C1516741 - Toby Wright C1540014 - Chian Sye Chia C1575242
Thomas Scammell C1621307 - Thomas Alexander Hulse C1648581

School of Engineering 2019 – 2020
Supervisors: Ze Ji, Emmanuel Brousseau

Abstract

In nature, snakes possess the ability to change their method of locomotion depending on their environment for efficiency. Work has been undertaken to design and build a snake-like robot consisting of individual identical modules connected in a chain-like formation to replicate three of these motions (rectilinear, serpentine and sidewinding). A module housing was designed, and 3D printed using selective laser sintering to package all components necessary to replicate the motions within it, including a removable tray for ease of access to these internal components. The modules were designed to allow for a 180° unimpeded rotation to the adjacent housing. Each of these individual units were to be designed such that they could be operated individually or linked together with inter-modular communication to form systems of varying numbers of modules. Two prototypes were assembled, one to replicate rectilinear motion, the other to replicate serpentine and sidewinding motion. Prototype 1 used micro linear actuators to extend a silicone skin enabling its scales to extrude, increasing friction in one direction, thus propelling the snake forwards. Prototype 2 used a series of Dynamixel servomotors controlled with a Robotic Operating System (ROS) network to replicate motion. The first prototype was shown to be able to push one module forwards at low speeds, improvements to this prototype have been suggested to help increase the speed and effectiveness of its movement primarily by introducing more modules and undergoing weight reduction. The second prototype was shown to successfully carry out both serpentine and sidewinding motions allowing for forward, backwards, sideways and circular motion. The use of sensors within a closed-loop feedback control has also been investigated to allow the robot to detect objects in its environment and react accordingly. Force sensitive resistors were selected and tested, proving that implementation between the skin and the housing was a suitable method of contact detection. Due to the Covid-19 pandemic, not all work could be completed out as desired including the full build of the first prototype and motion analysis of each. Suggested future work included design optimisation and the integration of both prototypes into one robot to combine all three types of motion.

Contents

1. Introduction	1
2. Modular Snake Robot Design Overview	1
2.1. Replicating snake motion	1
2.1.1. Rectilinear motion	1
2.1.2. Serpentine motion	2
2.1.3. Sidewinding motion	2
2.2. Component Overview	3
2.3. Management of Tasks	4
3. Hardware Design	6
3.1. Skeleton	6
3.1.1 Internal tray design	6
3.1.2 External housing development and prototyping	7
3.1.3 Final module design and assembly	8
3.2. Skin	9
3.2.1. Initial Investigation outcomes	9
3.2.2. Skin Testing	9
3.2.3. Skin attachment to housing	13
3.3. Kinematic models	14
4. Control, Sensors and Software	17
4.1. Closed-loop feedback control	17
4.2. Motors (Linear actuators and Servos)	18
4.2.1. Rectilinear motion (Linear actuators)	18
4.2.2. Serpentine and Sidewinding motion (Servo motors)	20
4.3. Sensors	31
4.3.1. Sensor selection	31
4.3.2. Sensor Hardware	36
4.3.3. Sensor Implementation plan	37
5. Integration and Testing	37
5.1. Prototype 1 – Rectilinear motion	37
5.1.1. Actuator Testing	37
5.1.2. Frictional Testing of Skin	39
5.2. Prototype 2 – Serpentine and Sidewinding motion	40
5.2.1. Motor Testing	40
5.2.2. Sensor Component Testing	43
5.2.3. Sensor integration Testing	44
6. Testing Results and Discussion	45
6.1. Prototype 1	45
6.1.1. Skin Testing	45
6.1.2. Actuator Testing	45
6.1.3 Frictional testing of Skin	48
6.2. Prototype 2	50
6.2.1. Housing Assembly	50
6.2.2. Motor Testing	51
6.2.3. Sensor Component Testing	56
6.2.4. Sensor integration Testing	57
7. Future work	58
8. Conclusion	59
9. References	61
10. Appendix	63

1. Introduction

Despite a lack of limbs, snakes possess the ability to move efficiently over rough terrains and through tight spaces using their great flexibility and various methods of locomotion. By imitating these motions, snake-like robots can have numerous applications in an industrial context. Such examples include travelling over, or through, hazardous terrain for rescue missions (Carnegie Mellon University, 2017), the enhancement of medical equipment for invasive surgical uses due to their flexion capabilities (Choset, n.d.) or even a role in facilitating complex tasks required by space rovers exploring extra-terrestrial environments (Liljeback, et al., 2013).

A project brief was devised to design and build a modular-based robot capable of recreating some of these methods of snake locomotion using several components housed within a series of identical modules. Each of these individual units were to be designed, such that they could be operated individually or linked together with inter-modular communication to form systems of varying numbers of modules. This would lead to a smart robot whose capabilities are expanded due to its motion and length adaptabilities. It was decided that the modules should be chained together, with the connections between components made to be as simple as possible, to minimise the time and complexity of setting up each iteration of the system, in addition to ease of component replacement if any issues were to occur.

The robot design would be able to carry out three different motions: serpentine (lateral undulation), sidewinding and rectilinear. To create a system capable of all three methods, two systems of control would be required to perform all three motions, these are linear actuator motors and smart servo motors. The final aim was to have one working prototype to combine the two systems of control, however, due to timing constraints, this integration did not occur. Instead, two working prototypes of each system were built and tested to demonstrate the capabilities of the final design.

2. Modular Snake Robot Design Overview

2.1. Replicating snake motion

2.1.1. Rectilinear motion

Rectilinear motion is useful when the snake is travelling through tight spaces. It entails the snake moving in a straight line by contracting muscles to control the movement of the scales. The scales are alternatively lifted slightly from the ground, and then pulled downward and backwards; the friction between the scales and the ground is used to pull the body forwards (Moon, 2001). This occurs at several points along the body. When the scales are lifted forwards, they are stretched out creating a rougher surface in contact with the ground.

To replicate this motion, a silicone skin layer would be developed to mimic the movement of scales based on a method called Kirigami. To simulate the contracting muscles that control the movement of the scales, micro linear actuators will be used. These will extend and retract the housing which in turn, will stretch the skin layer from its normal state. At the normal state, the housing is retracted, the skin relaxed and the scales lying flat, on housing extension the skin stretches, and the scales protrude out. With the scales popped out, a higher coefficient of friction would be created in one direction, propelling the snake forwards upon housing retraction.

The silicone scales will be cut using a water jet cutter. This will encompass each module individually allowing for ease of replacement of a module if one were to break.

2.1.2.Serpentine motion

Serpentine motion, also known as lateral undulation, is the most common snake motion. It consists of waves of lateral bending being propagated along the body from head to tail using objects on the ground to propel itself forwards (Moon, 2001).

To simulate this motion, smart servomotors will be used to instigate the lateral wave created by the 180° rotation between adjacent housing units within the y-axis. This is achieved by inputting a lateral sinusoidal motion through the motors with a delay between each one, creating a slithering like motion.


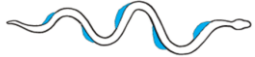
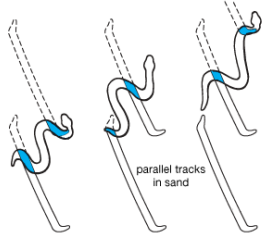
2.1.3.Sidewinding motion

When the surface is slippery, such as sand, the most common type of motion used is sidewinding, where two parts of the body are solely in contact with the ground with the remainder being held above it (Zug, 2001). The body is propelled laterally from these anchor points creating new anchor points a fixed distance away. Sidewinding is a complex motion that can be simplified by adding a vertical wave to the lateral existing wave of serpentine motion.

Similar to serpentine motion, this movement will be replicated using the servomotors. To recreate this motion, the same sinusoidal input as the serpentine motion will be implemented, as well as a similar vertical sinusoidal motion through the modules. This vertical motion will be offset such that the apexes of the lateral sine wave will be in contact with the ground, with the areas in between lifted above these anchor points.

To more easily compare the locomotion methods and understand the work that is needed to implement this in the robot, a summary for each motion is shown in Table 1.

Table 1. Description of each motion.

Locomotion	Rectilinear motion	Serpentine motion	Sidewinding motion
Images for illustration	 <p>(Encyclopædia Britannica Inc, 2012)</p>	 <p>(Encyclopædia Britannica Inc, 2012)</p>	 <p>(Encyclopædia Britannica Inc, 2012)</p>
Resistance to motion	Friction forces caused by micro skin motions	Friction forces caused by the housing surface	Friction forces caused by the housing surface and possible loss of motor power in lifting modules off ground
Resultant motion	Linear translation parallel to body	Linear translation parallel to body	Linear translation at an angle almost perpendicular to body
Work required	<ol style="list-style-type: none"> 1. Establish Skin design 2. Develop synchronous control of linear actuators 3. Integrate linear actuator and skin into housing 	<ol style="list-style-type: none"> 1. Develop control of servomotors 2. Establish motion replication of motors 3. Package motors into housing 	<ol style="list-style-type: none"> 1. Develop control of servomotors 2. Establish motion replication of motors 3. Package motors into housing
Additional functions	<ol style="list-style-type: none"> 1. Closed loop touch feedback to allow for: <ul style="list-style-type: none"> - obstacle collision detection - to detect contact with the ground for motion analysis 2. Communication with a master PC for control and monitoring 		

It was decided that two prototypes were to be built, one to instigate rectilinear motion, the other to perform serpentine and sidewinding motion using the servomotors. The second prototype would also integrate sensors into the robot surface for collision-detection and to detect contact with the ground. The collision-detection feature would create a closed-loop feedback system allowing the robot to respond to object detection by rerouting itself around the hazard. The detection of contact with the ground is an element that aids in the robot motion analysis during the research stage of future phases.

2.2. Component Overview

The predominant component propelling forward the first prototype during rectilinear motion are PQ12 Micro Linear Actuators. These require a Linear actuator control (LAC) board and an Arduino for control, in addition to a 6 V supply. In order to connect the actuator to the LAC board, a cable adapter is required to fit the ribbon connector to the female pin connector of the LAC board. The actuator and its relevant components are depicted below.

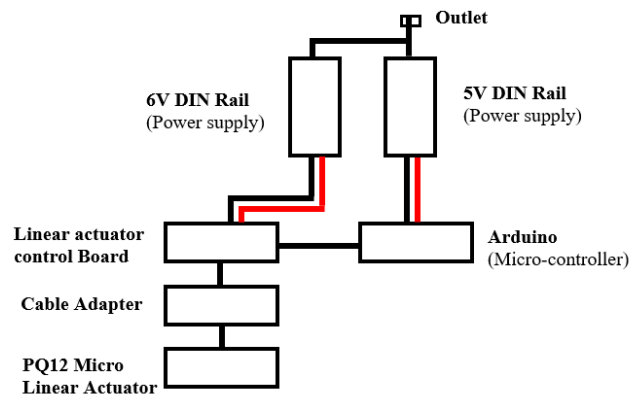


Figure 1. Components for prototype performing rectilinear motion.

For the second prototype performing serpentine and sidewinding motion, the component driving the motions are Dynamixel AX-12A servomotors. The motors require a 12 V supply and are sent commands via a laptop. For ease of wiring, a 6-port power hub is used to join the laptop and 12 V supply wiring connections to ensure the robot is tethered at one place. In addition to the motors and its associated components, the sensors also are incorporated into this prototype. The Force Sensitive Resistors (FSRs) require an Arduino to read the values outputted, and a 5 V supply. The components for this prototype are shown in Figure 2 below.

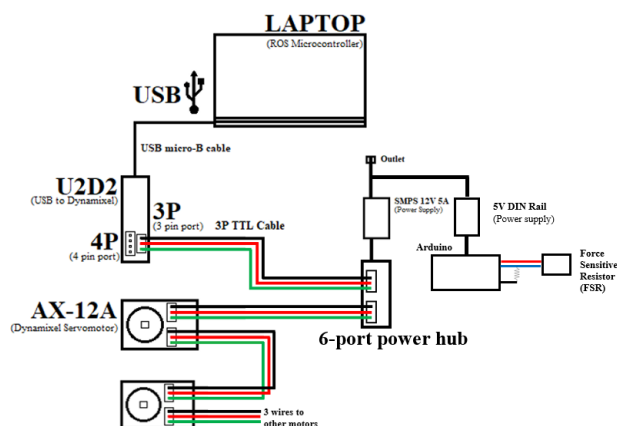


Figure 2. Components for prototype performing serpentine and sidewinding motion.

When integrating the two prototypes into one, all of these components would sit within the housings with the sensors and actuators sharing an Arduino.

2.3. Management of Tasks

To achieve the build of these two prototypes, the following objectives were set from project initiation:

- 1) Construct a self-contained module unit that contains relevant components
- 2) Establish a system whereby multiple modules can be physically interlinked and programmed simultaneously
- 3) Develop code that allows the module chain to move in serpentine, sidewinding and rectilinear sequences
- 4) Design an outer skin layer that can aid with the rectilinear motion
- 5) Embed the outer skin layer with sensors that allow the robot to detect the collision of obstacles

Semester 1 focused on meeting project objectives (1) and (4). This was accomplished by splitting the team into three sub teams: Skeleton, Skin and Software. The project objectives and work conducted of each sub-team from semester 1 are detailed below.

Table 2. Semester 1 Project objectives of each sub team.

Sub team	Sub team Project Objectives
Skeleton	Designing the module housings
Skeleton	Designing the bracket links
Skeleton	Using kinematics to calculate the motions
Skin	Selecting skin material
Skin	Designing skin structure
Skin	Implementing linear actuators to aid in skin extension and retraction
Software	Enabling motor control and communication between them
Software	Implementing timed communication control for linear actuators
Software	Measuring and responding to obstacle detection using sensors

Skeleton:

By first conducting a literature review of robots in a chain-like formation, it was determined that a box shape housing design with mechanical hinges would permit the imitation of the snake motion. This led to the development of a housing unit to contain all relevant hardware to produce the motions. Furthermore, this model contained a suitable design for bracket links to allow for 180° movement between units within a specified axis.

A simplified calculation was also performed to determine the maximum number of modules that could be present in the robot if one motor were used to lift the entire robot. The result was 4 modules, this was an overestimate due to more than one module being in contact with the ground at one time but provided a good base to start with in terms of calculating the optimum number of modules.

Skin:

A second literature review was performed to explore skin structure designs and materials used to produce a suitable frictional force to allow a robot to propel forwards. The research led to several considerations: what mechanism will be used to simulate muscle force in order to propel the robot forwards, how to replicate the anisotropic frictional interaction property of snakeskin scales', if the skin will be one sheath or solely coat each module individually, and finally, the method to attach the skin to the housings.

It was determined that the muscle force will be simulated by extending and retracting the housing modules. This in turn, would stretch the skin layer allowing for scale protrusion on full extension. The

components chosen to produce this muscle force simulation were PQ12 Linear Actuators due to their compact size, light weight and capability to withstand high loads whilst maintaining a high drive speed in comparison to competitors.

The scale pattern was chosen based on research produced at Harvard University (Rafsanjani et al. 2018). The scale dimensions were manipulated to overcome the accuracy of the waterjet cutter and tension force acting on the scales. The skin material selected was silicone due to its high stiffness and ease of manufacturability. It was also determined that the skin would coat each module individually for ease of module replacement and internal component packaging.

Software:

To successfully replicate the serpentine and sidewinding motion, AX-12A DYNAMIXEL Servomotors were selected to produce 180° rotations between adjacent housings. These motors produced a high torque output at a reasonable cost. These smart motors also allowed for additional features such as position feedback and the capability to be connected in series reducing wiring complexity.

The motors were interfaced with the laptop allowing for a Robot Operating System (ROS) environment to be created. By creating a node in ROS to communicate with the motors, the following measurements can be obtained: target position, velocity and position feedback. Communication between the laptop and the motor was achieved allowing for motor manipulation and position feedback control.

This semester focused on assembling the two prototypes. This required tasks primarily based on motor control (section 4.2.2), skin design (section 3.2), skin mechanism (section 4.2.1), housing design (section 3.1) and sensor integration to create touch closed-loop feedback (section 4.1). New sub teams were created to complete these tasks, the project objectives of each sub team are detailed in Table 3 below.

Table 3. Semester 2 Project objectives of each sub team.

Sub team	Sub team Project Objectives
Skeleton	Developing and manufacturing the module housing design
Skeleton	Developing and manufacturing the bracket links
Skeleton	Using kinematics with simplifying assumptions to calculate motion
Skin	Testing several skin designs
Skin	Selecting final skin design
Skin	Development and testing of skin response using actuators
Motors	Enabling motor control including position feedback
Motors	Enabling communication between motors
Sensors	Measuring and responding to obstacle detection using sensors

The work achieved by each sub team including testing will be discussed in the subsequent sections with a detailed discussion on any issues and improvements detailed for future stages. The sections discussed within this report include:

- Designing an internal tray within the skeleton to house all components (section 3.1.1)
- Finalising the housing design to support all three methods of snake locomotion (section 3.1.3)
- Finalising the dimensions of the skin cutting pattern and testing methods of reinforcement to strengthen the scales (section 3.2.2 & 6.1.1)
- Selecting a method of attachment of skin to housing (section 3.2.3)
- Carrying out kinematics of the robot to aid in controlling its motion (section 3.3)
- Creating a prototype to carry out rectilinear motion using micro linear actuators:
 - Control of micro linear actuators (section 4.2.1 & 6.1.2)

- Testing of micro linear actuators (section 5.1.1)
- Frictional testing of skin (section 5.1.2 & 6.1.3)
- Creating a second prototype with sensor integration to carry out serpentine and sidewinding motion using servomotors:
 - Control of servomotors (section 4.2.2)
 - Testing of servomotors (section 5.2.1 & 6.2.2)
 - Control of sensors (section 4.1 & 4.3)
 - Testing of sensors (section 5.2.2, 5.2.3, 6.2.3 & 6.2.4)

3. Hardware Design

3.1. Skeleton

The design work in the first semester was aimed at developing the module housing and a bracket to enable module connection. The module housing consisted of 3 parts: the external and internal housing, and the internal tray. The external and internal housing allowed for module extension (separation of the two parts) and module retraction (sliding of the external part over the internal part to form a closed cube shape), these are shown in top and bottom images in Figure 5, respectively. The final part, the internal tray, a sliding insert for internal components to be mounted. All 3 parts were designed with semester 2 focusing on design optimisation using rapid prototyping techniques.

Final modules were then produced using the SLS printer in the additive manufacturing lab before a working robot prototype was assembled and the modules underwent physical testing. This section discusses the processes involved in the design and development of the insert, the prototyping and development of the external housing and the functional capabilities of the final module design, shown in Figure 5.

3.1.1 Internal tray design

3.1.1.1 Parameters

The insert itself must provide a mounting point for all components, whilst conforming within three main parameters; motors must be positioned correctly for unimpeded robot function, components must fit within as small an effective volume as possible and the insert must provide channels for correct cable routing. Figures 2 and 3 highlight the components needed for each robot motion, with Table 4 showing the list of components that each module has been designed to house their volumetric dimensions and masses.

Table 4. Robot internal component list.

Component name	Length / mm	Width / mm	Height / mm	Mass / g
Dynamixel Ax-x12	50	40	32	55
Actuonix PQ12	48	21.5	13	15
LAC board	50	12	50	14
Arduino nano	43.2	10.5	17.8	7

It was determined that a volume with length 50 mm, width 52 mm and height 52 mm would be appropriate to house all components and wiring. The tray would fit within the main module with an extension for the servo motor to be mounted externally housing. The tray extension for the servomotors sits within the brackets between modules.

The internal tray design is shown in Figure 3a with Figure 3b showing the manufactured tray with the internal components packaged with several distinct features to accommodate the dimensions and wiring of each component, this can be seen in Figure 4. For the LAC board there are appropriately positioned mounting holes and an indent, allowing space for components that slightly overhang the

board where it contacts the insert. The raised bracket has a mounting point for the linear actuator and positions it in line with its second mounting point, to the external housing. The servo motor bracket was strengthened due to several failures during the prototyping phase of the insert and has several mounting holes as well as a cut out where the motor mounts to the bracket of the next module. There are also three mounting holes for the Arduino nano. Additionally, a mounting track for the insert was created with mounting holes for secure fixing to the external housing.

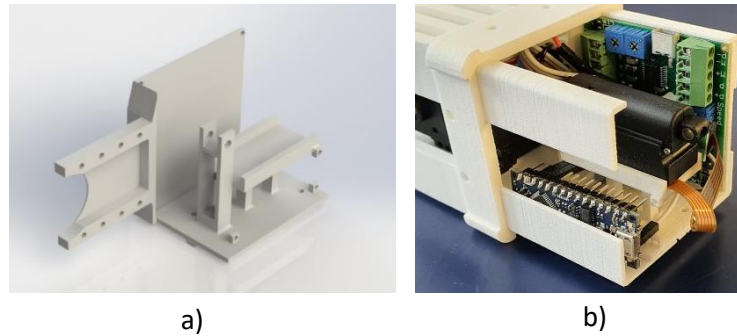


Figure 3. Module insert a) final design of the internal tray b) manufactured tray with components packaged

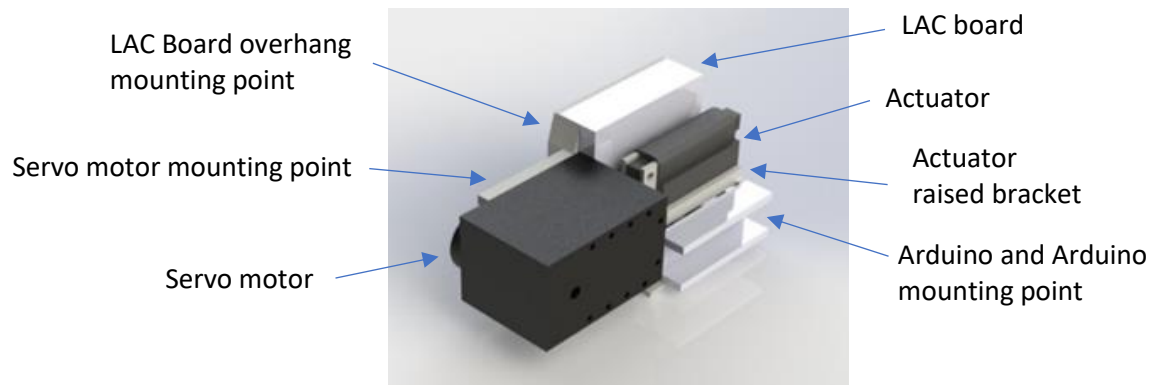


Figure 4. Module insert with components packaged

The internal tray had a mass of approximately 17 g. With the addition of the internal components , the total mass, excluding the mass of wiring which adds approximately a further 5 g, was 108 g.

3.1.2 External housing development and prototyping

3.1.2.1 Development in CAD

The external housing design underwent several iterations for weight reduction, internal tray packaging and wiring accommodation, whilst maintaining the same functional bracket design. Figure 5a below shows the external housing design as of the end of semester 1 and Figure 5b the final housing design for manufacture. Changes between the two designs were as follows:

- The internal volume was changed to accommodate for the insert.
- Further material was removed internally to reduce mass.
- Holes were added externally to allow for the sensor wiring to pass through the housing
- The external shape was adapted to fit the finalised skin dimensions
- Mounting points were added to securely attach linear actuator and servo motors.

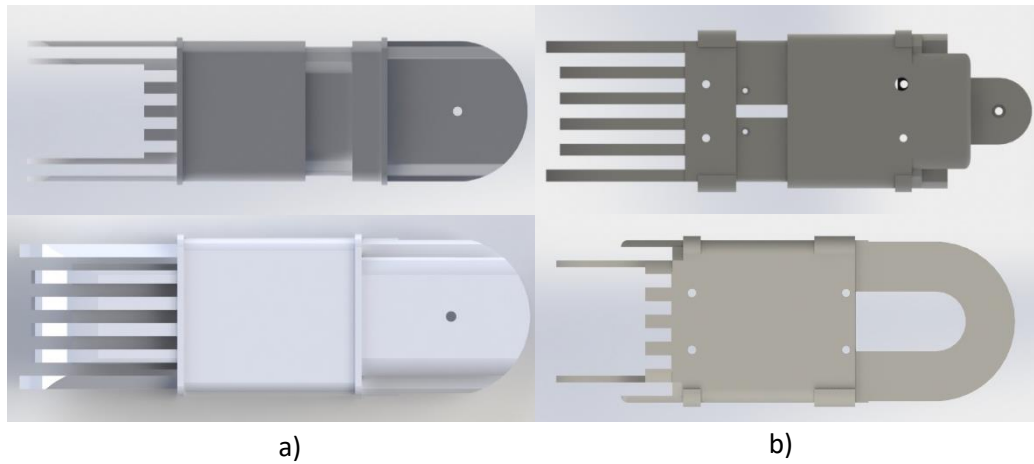


Figure 5. Housing design: a) housing design at the end of the first semester b) final housing design.

The final design was 3D printed using Ultimakers to identify areas where the housing may require further development. This led to further changes:

- The bracket was optimised, and material added to the base of the fins to increase rigidity where they snapped on several prototypes.
- Material removed from the fins where appropriate to offset the mass gain from the added material at the fins.
- The bracket mounting points finalised and altered for the appropriate screw dimensions.
- For manufacturing, the dimensions of all sliding parts and the bracket fins were altered to account for the tolerances of the SLS printer.

3.1.3 Final module design and assembly

As mentioned above, the SLS 3D printer was used to manufacture the finalised modules for the robot, comprising of three parts, their masses are outlined in Table 5 below.

Table 5. Masses of module components.

Part name	Mass / g
Insert	17
External housing inner	81
External housing outer	56

The total mass of the module components was of 154 g, with the addition of internal components and the skin layer, adding approximately 15 g, the final mass of a fully functional module was ~260 g.

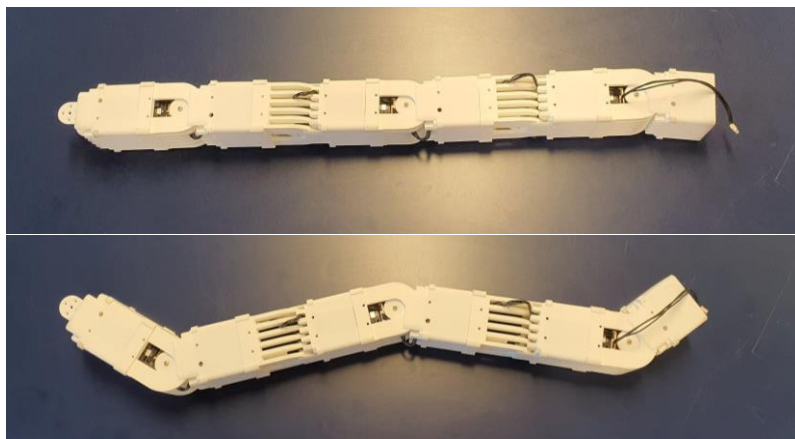


Figure 6. Assembly of modules in functioning robot.

Figure 6 shows an assembly of five modules with the housings conforming to the parameters for the robot to function effectively. With components mounted internally, the skeleton itself was shown to be fully functional during physical testing.

3.2. Skin

3.2.1. Initial Investigation outcomes

To facilitate the rectilinear motion, it was decided that a skin be used to replicate the opening/closing of scales to propel the robot forwards. This would be accomplished using a cutting technique called Kirigami and a water-jet cutter to create the cuts. By cutting the scales into the skin using a particular pattern, during material extension the scales will pop out and make contact with the ground. With the scales protruded, a higher coefficient of friction with the ground in the direction opposite to the desired motion will be created. This will permit the robot to push forwards off the ground when the modules are retracted.

Silicone was selected for the skin material due to its good flexibility and anti-adhesive properties, minimising the coefficient of friction with the ground when flat (in the direction of motion). It was also decided that, to aid accessibility to the inside of the modules, each module would have its own sections of skin attached to each side. This would leave the ends of the modules open for access to the internal components and wiring. In order to extend and retract the modules, a method of using micro linear actuators (PQ12 Linear Actuators) was chosen due to their small size and good controllability.

To continue the development of the skin design, the following tasks were set out:

- 1) Test methods of strengthening the silicone to reduce the chance of scales buckling when extended.
- 2) Test the silicone samples under tension to find out whether they can withstand the loads experienced when extending the modules and determine the stiffness of the silicone and the effect that different patterns of cut would have on the skin stiffness.
- 3) Determine a method of attaching the skin to the modules, considering the anti-adhesive properties of the material.
- 4) Conduct frictional testing of the skin to determine how effective the skin will be in aiding the rectilinear motion over various different substrates.
- 5) Undergo the final testing of the skin on the modules being controlled by the micro linear actuators.

These tasks are discussed in the following Skin sections.

3.2.2. Skin Testing

3.2.2.1. Combining Acetate with Silicone

In the initial investigation stage* (Appendix 4), strengthening silicone with an acetate layer was attempted using two methods:

1. Binding using double-sided adhesive tape
2. Curing liquid silicone with an acetate layer

The first method was conducted as shown in Figure 7 below. When in tension, the binding was seemingly secure (Figure 7a). However, it was found that the layers could easily be prised apart when pulled by hand (Figure 7b). This revealed that the tape was only adhering to the acetate and not the silicone. Testing other types of double-sided adhesive tape was considered, but even if the adhesion was successful, this would not guarantee the bonding would last once patterns were cut into the sheet with the water-jet cutter, thus was not considered as the final method.

*Note: Initial Investigation section refers to work conducted between October - December 2019

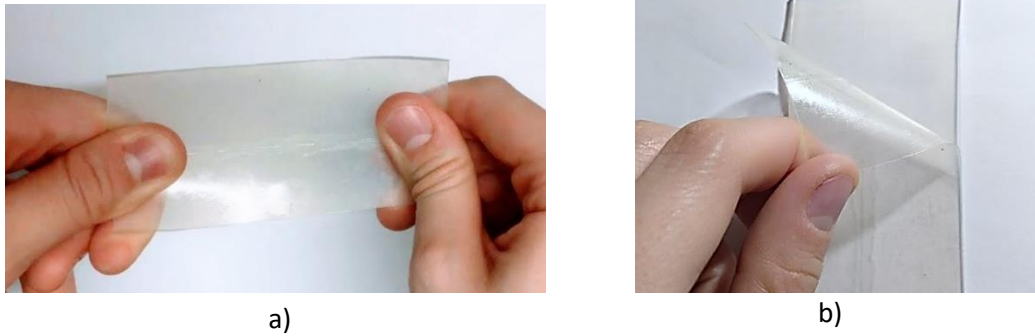


Figure 7. Adhesion of Silicone sheet with Acetate layer: a) Silicone-acetate layer sample b) Peeling apart of the sample's layers

The second method utilised Smooth-Sil™ silicone rubber, a liquid mixture that cures at room temperature with negligible shrinkage (Smooth-On 2020). This was inspired by a study that had used a similar method for an Earthworm robot (Liu et al. 2019). Samples were made, including curing the silicone layer on top of acetate, curing the silicone underneath the acetate and sandwiching the acetate between 2 layers of curing silicone (Figure 8a). However, all of these methods resulted in there being no bonding between acetate and silicone as shown in Figure 8b below. One of the properties of silicone is that it is generally non-adhesive with other surfaces. Liu et al. (2019) successful adhesion was due to the particular type of plastic layer they used to combine with the cured silicone.

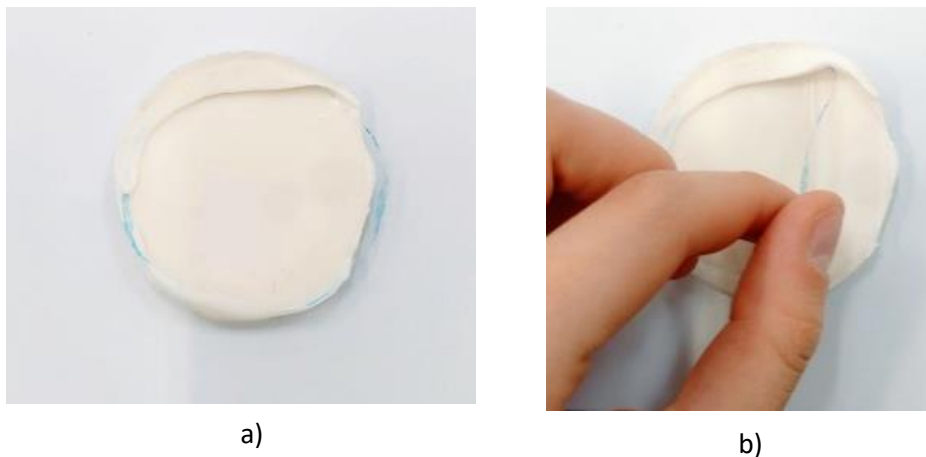


Figure 8. Adhesion of cured Silicone with Acetate layer: a) Silicone and acetate combining sample b) Peeling away of acetate from cured silicon

It was concluded that the acetate reinforcement would not be a feasible option for this project. After assembling the robot modules, if the silicone scales proved to not be stiff enough to lift the weight of the housing, methods for weight reduction was to be explored.

3.2.2.2. Tension testing results

In the initial investigation stage (Appendix 4), the aim was to perform tension tests to ultimately obtain the Young's modulus of the original silicone sheet and to conduct a comparison between the snake-skin samples in terms of flexibility and stiffness. The Young's modulus of the uncut silicone sheet was measured and calculated using a dog bone sample as shown in Figure 9a (height=38 mm, width=7 mm). Figure 9b shows the set up for the snake-skin samples, each sized at dimensions of height=70 mm and width=48 mm. Each sample underwent an extension under a load of 20 N at 8 mm/s (the peak efficiency settings of the actuator utilised to create the tension force) until complete failure.



a)



b)

Figure 9 – Tension testing of silicone samples a) Dog bone sample of the silicone sheet b) Tension testing of silicone samples

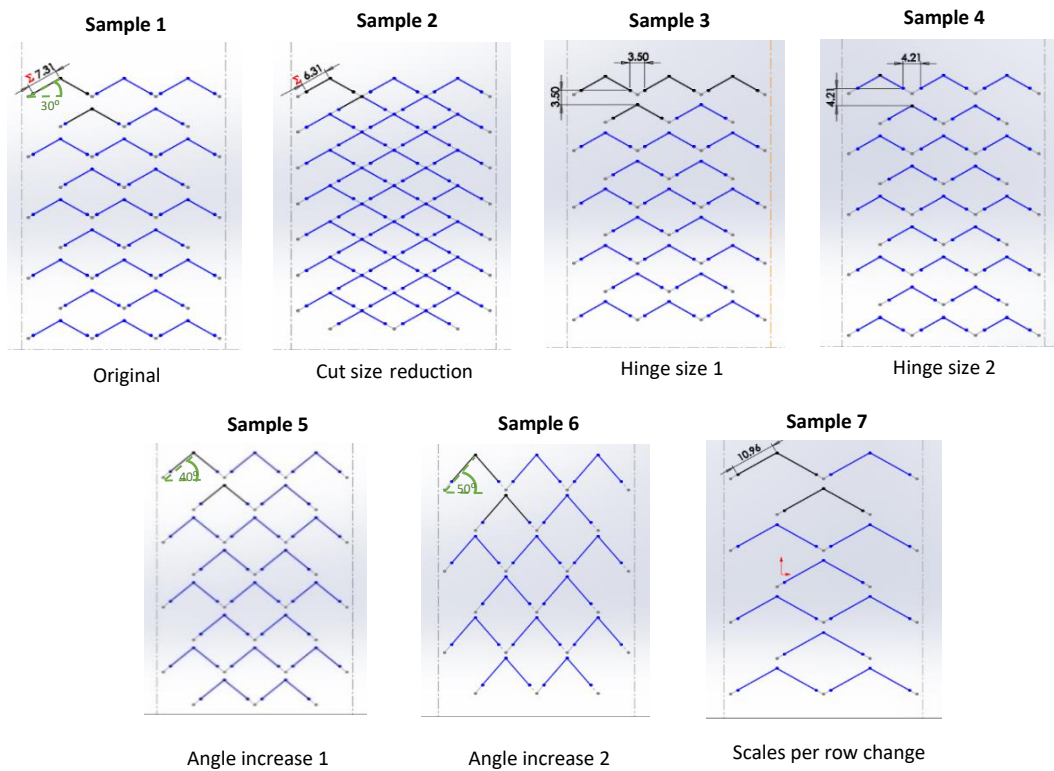


Figure 10. Snake-skin samples designed to the approximately height=70 mm and width=48 mm.

Seven different sample designs were cut out for experimentation. Sample 1 depicts dimensions that are proportional to the derived equations found in the Kirigami research undertaken at Harvard (Rafsanjani et al. 2018). As for the subsequent samples, factors that were varied included adjusting the cut size, the hinge size, the angle of the cut with respect to the horizontal plane and the number of scales per row (Figure 10). Table 6 below lists the factors changed for each sample.

Table 6. Description of each skin sample tested.

Sample Number	Sample Description
1	Original dimensions (cut size = 7.31 mm, cut angle = 30°)
2	Cut size reduced to 6.31 mm
3	Hinge size made to uniform dimensions of 3.50 mm in height and width
4	Hinge size made to uniform dimensions of 4.21 mm in height and width
5	Cut angle increased to 40°
6	Cut angle increased to 50°
7	Number of scales per row reduced to 2 scale/1 scale arrangement

The results for the tension testing for the dog bone specimen are displayed in Figure 11 below:

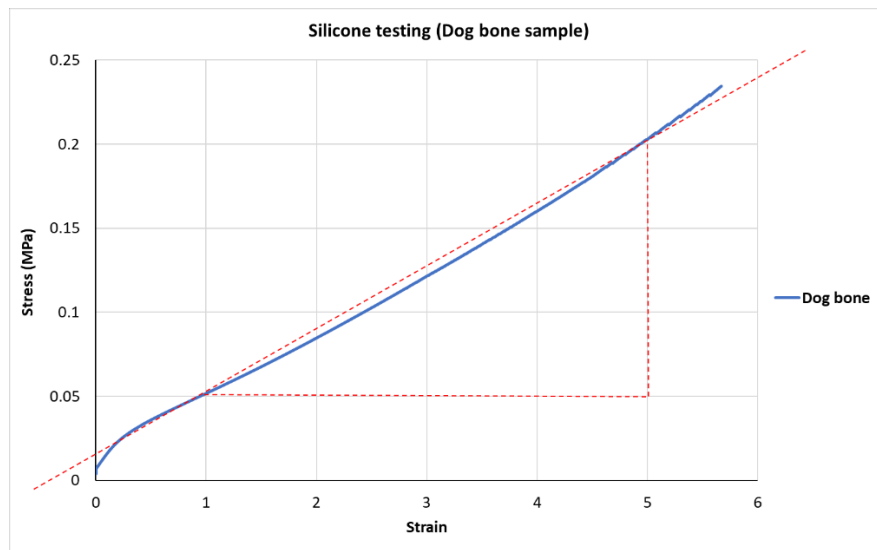


Figure 11. Dog bone sample graph (red dotted lines depict the range used to derive the Young's modulus value).

From Figure 11, the Young's modulus of the Silicone sheet was found to be:

$$\begin{aligned}
 E &= \frac{(0.2 - 0.05)}{4} \\
 &= 0.0375 \text{ MPa}
 \end{aligned}
 \tag{1}$$

This is a lot more compliant than what the typical Silicone range (1 - 50 MPa) is cited to be (Azo materials 2019a). Furthermore, it was found that the stiffest snake-skin sample had a modulus of 0.012 MPa (see data from Figure 75 (Appendix 1). This demonstrates that the silicone will retain, at most, a third of its original stiffness when scales are cut into it. Therefore, it is important to keep in mind because, the more compliant the snakeskin becomes, the more difficult it will be to produce the protrusion effect of the scales.

A comparison between all snake-skin samples can be drawn from Figure 12 below:

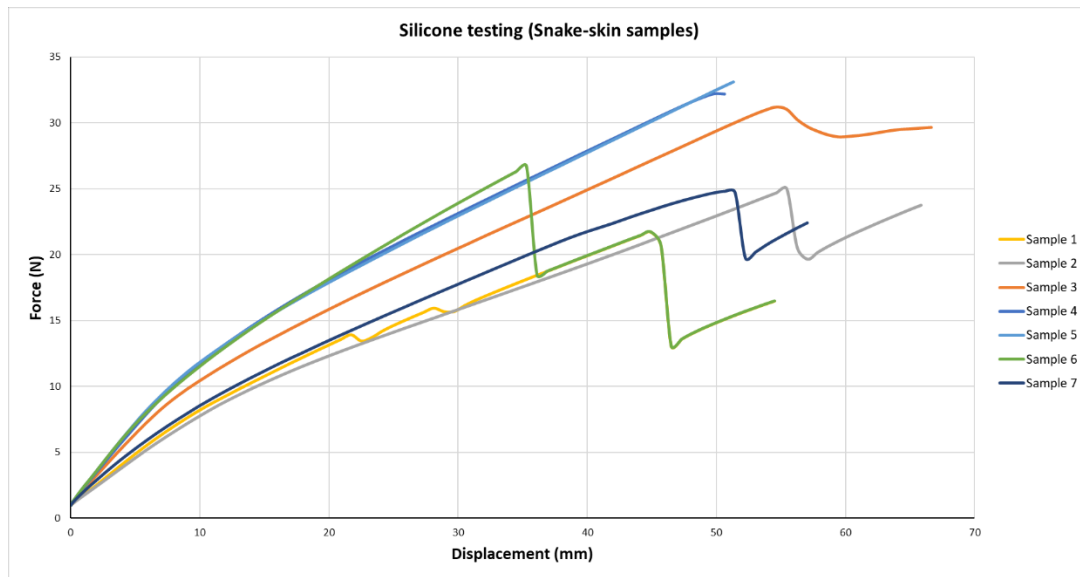


Figure 12. Snake-skin samples graph (used to compare the behavioural properties of each sample).

As demonstrated by some samples, the skin extended until a single hinge failed (usually on the side edge), which caused a drastic decrease in the stiffness. For example, this is depicted by the sharp drop in force for sample 6 at a displacement of 35 mm and then again at 45 mm, when more hinges start failing. Therefore, this sample was ruled out as its hinges clearly would not be durable over multiple extension cycles undergone to produce robot movement. Other conclusions that could be drawn from the results were:

- Cut size had minimal influence on the stiffness (illustrated by samples 1 and 2).
- There was a positive correlation between hinge size and the stiffness (illustrated by samples 3 and 4).
- As the angle size increase, the stiffness also increased (illustrated by samples 2, 5 and 6).

The criteria to define the optimal snake-skin sample was complex. It was desirable for the skin to be as flexible as possible so that the layer may be extended easily by the actuator, but it was also desirable for the skin to be as stiff as possible to allow for the hinges to trigger the protrusion of the scales, without any failure propagation through the skin.

Thus, a couple of options were provisionally chosen for a final test. At 20 N, sample 2 was chosen as it had the greatest extension before hinge failure. Furthermore, sample 3 was also selected, as it was feared that if the stiffest sample was selected, the full extension of actuator power may not be able to fully extend it, meaning the scales would not protrude optimally. Sample 3 was tested first with the actuator and housing. From the results (as detailed in section 6.1.2.), the skin stiffness did not hinder actuator extension and thus it was no longer required to test sample 2, which would have been a weaker, and therefore less favourable design.

3.2.3. Skin attachment to housing

To attach the silicone skin to the modules, a method of attachment that would fulfil a series of requirements was essential. These included:

- Secure attachment of the skin.
- The attachment method should not overlap above the skin layer as to not interfere with the contact between the scales and the ground.
- It would be preferable to have a method that facilitates the removal of the skin for accessibility of the module interior or to replace damaged skin sections.

Initially, ideas to attach the skin to the module temporarily using either hooks or clamps were considered. These were deemed not viable as they would involve parts of the clamps/hooks sticking out over the skin. This would interfere with the contact between the scales and the ground. As a result, the friction coefficient between the scales and the ground may be reduced. Lowering this coefficient of friction would reduce the effectiveness of the scales and thus the movement of the robot. Therefore, more permanent methods using commercially available adhesives were examined.

First a polyurethane adhesive, Gorilla Glue, was tested by gluing a silicone sampled to the 3D printed housing module made from a Nylon 66 Polyamide. When dried, this method seemed like a good choice when a shear force was applied, however the skin could be detached easily when peeled. This was due to the 'non-stick' characteristics of silicone. To overcome this, a silicone-based adhesive, Sil-Poxy, was identified as a viable option. Conventionally, it is used for the gluing of silicone rubber to more silicone and some other substrates. When tested, this method was also good in tension, but could still be peeled off if a moderately high force was applied. This issue was mitigated by roughening up the surface of the silicon first using sandpaper, then applying the adhesive. This resulted in the force required to peel the silicone off the substrate to increase to a level where it would only be possible if the user wanted to do so. This was deemed a viable method of attachment. As the forces that the skin would be experiencing were low, further tension and peel tests were not required to quantify the values of the forces required to detach the skin from the module.

3.3. Kinematic models

For demonstration purposes, forward kinematics for 3 linked DYNAMIXEL motors (Figure 13) was calculated. With 3 motors, it can be assumed to be a robotic manipulator with 3 degrees of freedom.

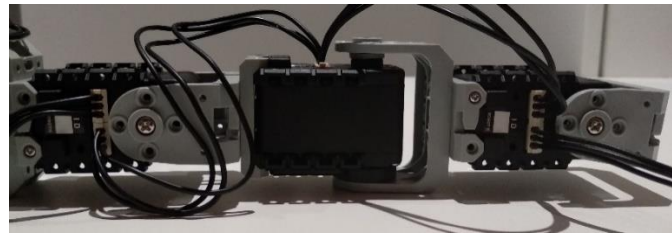


Figure 13. Side view of 3 linked DYNAMIXEL motors.

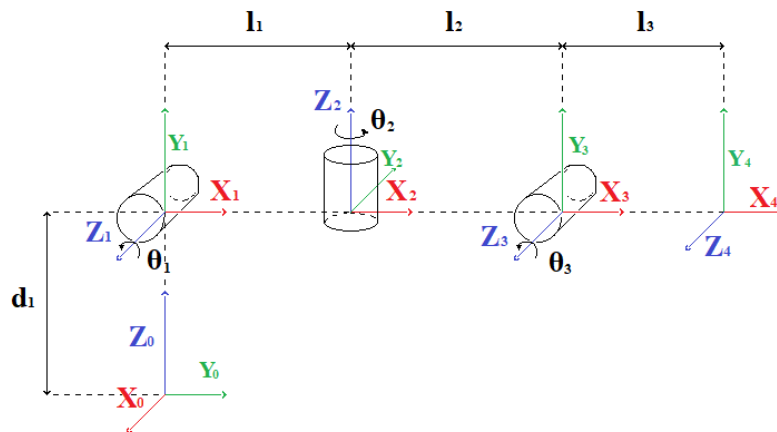


Figure 14. 3 linked DYNAMIXEL snake robot base coordinate system.

The kinematics for a 3 linked DYNAMIXEL snake robot was derived from the Denavit-Hartenberg (D-H) parameters method (Spong and Vidyasagar, 1989). As shown in Figure 14 above, all 3 joints are revolute joints (Table 7). Following the assumptions of kinematic chains, n number of joints will have $(n+1)$ number of links. Hence, this system will have 4 links.

Table 7. 3 linked DYNAMIXEL snake robot joint specification.

Joint	Description	Type
1	DYNAMIXEL AX-12A	Revolution
2	DYNAMIXEL AX-12A	Revolution
3	DYNAMIXEL AX-12A	Revolution-End effector

First, the base coordinate system of the 3 linked DYNAMIXEL snake robot is interpolated. The z-axis is placed perpendicular to the revolving direction of the joint (θ in Figure 14). The direction of the x-axis it depends on the direction of the common normal if it satisfies one of the following 4 cases:

- Case 1: not intersecting, not parallel
- Case 2: intersecting, not parallel
- Case 3: not intersecting, parallel
- Case 4: intersect, parallel

Under the right-hand coordinate system: link 1 falls under case 1 thus x_1 follows the common normal. Link 2 falls under case 1 therefore x_2 follows the common normal. Link 3 falls under case 1, x_3 therefore follows the common normal. Link 4 falls under case 3, so x_4 follows any common normal. The coordinate system is then kept the same as the previous coordinate system to simplify the D-H parameters. The D-H table was then be tabulated where $L_1 = L_2 = 66$ mm, and $L_3 = 25$ mm, for the basic motor brackets (Table 8).

Table 8. 3 linked DYNAMIXEL snake robot D-H parameters.

Link [#]	Joint Angle θ_i [rad]	Link Off-set d_i [mm]	Link Length a_i [mm]	Twist Angle α_i [rad]
1	0	d_1	0	$\pi / 2$
2	θ_1	0	66	$-\pi / 2$
3	θ_2	0	66	$\pi / 2$
4	θ_3	0	25	0

The link off-set for link 1, d_1 , separates the base coordinate and the first joint. d_1 is variable and can also have no value (0). Figure 14 represents the coordinate system in this way to make a clear distinction between the links and joints in the system. The transformation matrix, A_i , for each link was formed. A_i follows the standard form:

$$[A_i^{i-1}] = \begin{bmatrix} \cos \theta_i & -\cos \alpha_i \sin \theta_i & \sin \alpha_i \sin \theta_i & a_i \cos \theta_i \\ \sin \theta_i & \cos \alpha_i \cos \theta_i & -\sin \alpha_i \cos \theta_i & a_i \sin \theta_i \\ 0 & \sin \alpha_i & \cos \alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (2)$$

Where θ_i is the joint angle, d_i is the link off-set, a_i is the link length, and α_i is the twist angle. Hence:

$$[A_1^0] = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (3)$$

$$[A_2^1] = \begin{bmatrix} \cos(\theta_1) & 0 & -\sin(\theta_1) & 66 \cos(\theta_1) \\ \sin(\theta_1) & 0 & \cos(\theta_1) & 66 \sin(\theta_1) \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (4)$$

$$[A_3^2] = \begin{bmatrix} \cos(\theta_2) & 0 & \sin(\theta_2) & 66 \cos(\theta_2) \\ \sin(\theta_2) & 0 & -\cos(\theta_2) & 66 \sin(\theta_2) \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (5)$$

$$[A_4^3] = \begin{bmatrix} \cos(\theta_3) & -\sin(\theta_3) & 0 & 25 \cos(\theta_3) \\ \sin(\theta_3) & \cos(\theta_3) & 0 & 25 \sin(\theta_3) \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (6)$$

A T matrix was then constructed to show the rotation and position of each joint and the end-effector:

$$\text{Joint 1} = [A_1^0] \quad (7)$$

$$\text{Joint 2} = [A_1^0] \times [A_2^1] \quad (8)$$

$$\text{Joint 3} = [A_1^0] \times [A_2^1] \times [A_3^2] \quad (9)$$

$$\text{End Effector} = [A_1^0] \times [A_2^1] \times [A_3^2] \times [A_4^3] \quad (10)$$

The T matrix for each joint will take the standard form where the upper left 3x3 matrix represents the rotational matrix of the joints relative to the origin and the upper right 3x1 matrix represents the translation components in 3 axes relative to the origin:

$$T \text{ matrix} = \begin{bmatrix} R_{xx} & R_{xy} & R_{xz} & P_x \\ R_{yx} & R_{yy} & R_{yz} & P_y \\ R_{zx} & R_{zy} & R_{zz} & P_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (11)$$

Through interpolation of the transformation matrices, the T matrix for each joint and the end-effector, the angular rotation of joint 3 (θ_3) only affects the end-effector's rotation and position so that previous joints in the system are not affected. This is shown in Equation 6 ($[A_4^3]$) – the only transformation matrix to contain a coefficient for θ_3 – and Equation 10 (the T matrix for the end effector) – the only T matrix in the system to be a product of $[A_4^3]$.

The T matrix equations for each joint fit the assumptions of a robotic manipulator for calculations in forward kinematics, however the movement of a snake requires the interaction of objects in the environment as well as the ground itself. This proves an issue to utilise kinematics to represent the robot's motion.

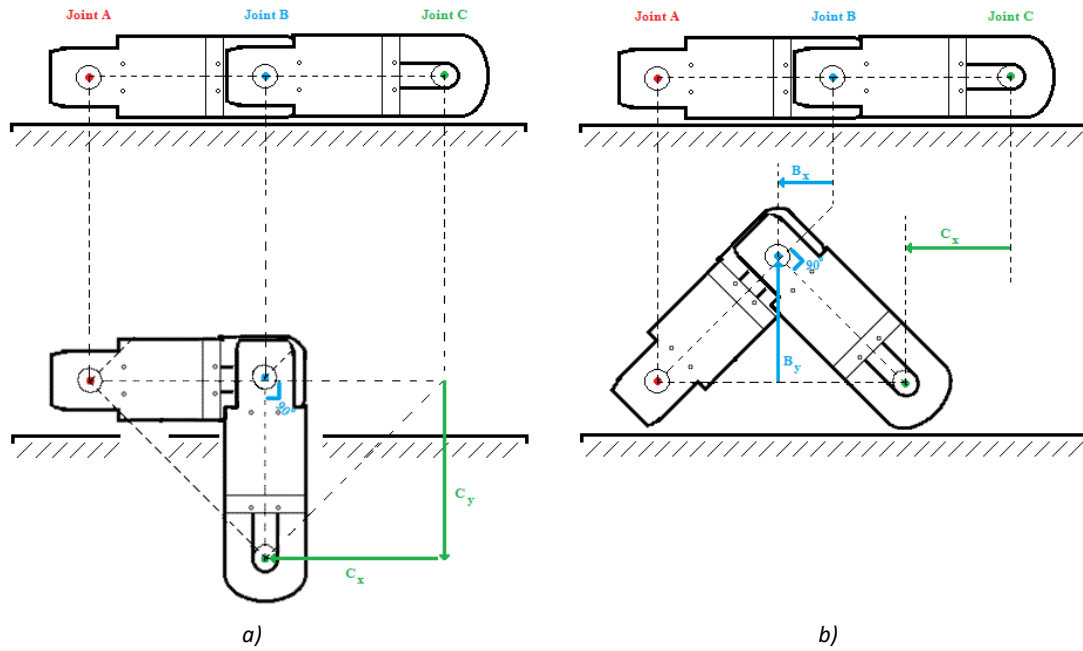


Figure 15. 90° joint B rotation: a) D-H Kinematics model b) Realistic model

Figure 15a demonstrates the result of joint B rotating 90° clockwise under the assumption of the D-H kinematics model outlined above. Figure 15b demonstrates the result of joint B rotating 90° clockwise under a realistic model – which accounts for the interaction with the flat ground level surface. These show that the 90° clockwise rotation of joint B will affect the end position of joint C (an end-effector) – as predicted by kinematics. However, joint C's contact with the ground (a flat surface) will change the displacement of previous joint B as shown in Figure 15b which is not accounted for in D-H kinematics (Figure 15a). Furthermore, the calculations assume joint A is fixed, in reality this is not true thus creating further displacement complications. The inaccuracy of kinematics modelling increases with more motors and modules added. As a result, it was deemed that the inaccuracy of using kinematics modelling for snake-like motion is too inaccurate, thus is not the ideal method for modelling motion profiles or planning movements.

4. Control, Sensors and Software

4.1. Closed-loop feedback control

By incorporating a closed-loop feedback control into a robot, the device can become smarter by monitoring internal components and its environment. For example, a closed loop for control to enable position feedback to ensure the component, i.e. a motor, reaches the position commanded by the controller removing inaccuracy (LeGrand, 2004). Another method commonly used within robots is the monitoring of the environment by sensors for obstacle detection. The sensors can be used to detect an obstacle then ensure the robot responds by changing its motion, for example, coming to a stand-still.

Within this project, the latter method was used to enable the second prototype (powered with servo motors) to navigate around an obstacle that had been detected through a head-on collision. The hardware utilised within this feedback loop are connected through a ROS environment and communicate via ROS nodes, this set-up is discussed in detail in section 4.2.2.3. The communication transfers and hardware of the control loop is shown in Figure 16 below.

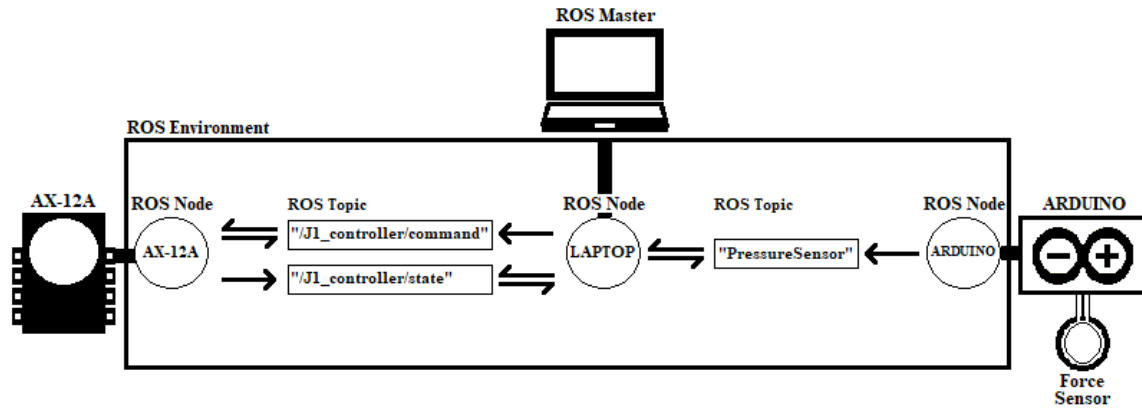


Figure 16. Hardware and communication transfers within closed-loop feedback.

The role of FSRs were to detect the collision of the robot during movement by embedding sensors into the 'head' of the snake. During a collision, the sensors will make contact with the obstacle leading to a decrease in the resistance. Since the raw sensor data cannot be directly inputted into the ROS environment an intermediary component, a microcontroller, is required. Arduinos were chosen due to their simplicity, cost and most importantly, compatibility with ROS. The change in resistance during a collision would be recorded by the microcontroller and published to the 'Arduino' node in the ROS network. If the value exceeds the threshold, it will be deemed a collision. This results in a command being sent to the motors via ROS to initiate a protocol to reroute the robot around the obstacle.

There are two main methods for integrating an Arduino into a ROS system, the simplest approach is to upload the code using the Arduino IDE software, similar to any other standard Arduino project. Utilising the `rosserial_arduino` package, the `rosserial` communication protocol is launched by including the library during the set-up section of the code. Although useful for testing, this method is generally considered inadequate for more complex systems where it is more useful to have code located and uploaded from a central directory. The other approach is to upload the code within the catkin workspace. Once setup this allows quick and easy alterations to the Arduino code, or any other code integrated within the workspace. The code remains almost the same as before however it needs to include the "Arduino.h" header and is saved within the projects source files. A part of the catkin workspace requires a `CMakeList.txt` file, this contain instructions on how to handle source files. This code will effectively act in place of the Arduino IDE software. Both methods were tested and are discussed in section 5.2.3.

4.2. Motors (Linear actuators and Servos)

4.2.1. Rectilinear motion (Linear actuators)

Micro-linear actuators were used to provide rectilinear motion where a single actuator movement simulates a segment of body movement of the snake. When using several actuators, the snake motion can then be simulated. Actuatorix PQ12 Linear Actuators were chosen to be the most desirable for this project. It weighs 15 g and is capable of smooth and consistent linear stroke length of up to 20 mm with a load up to 50 N. It also has an internal potentiometer which can provide positional feedback. In this project, there were 4 modules to represent the rectilinear motion.

4.2.1.1. Concept Design

The hardware workflow of the actuator control is shown in Figure 17 below where it has been simplified to demonstrate the wiring of one actuator to the control board. Each PQ12 Linear actuator is controlled by an Actuatorix Linear Actuator Control Board (LAC Board). The control board allows actuator adjustments for: speed, limits (minimum and maximum actuator positions) and accuracy

(changing the magnitude of actuator movement increments). Each module housing contained a PQ12 Linear actuator and a Linear Actuator Control Board, and one housing contained the Nano Arduino.

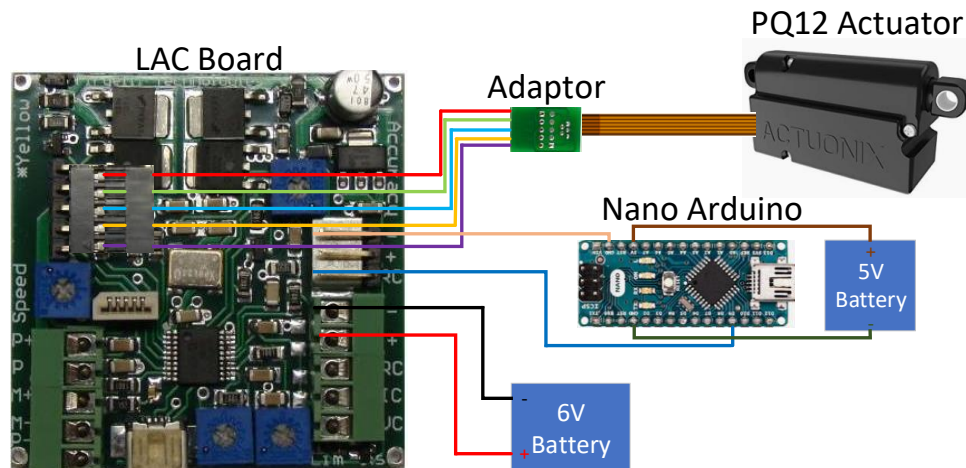


Figure 17. Hardware workflow of PQ12 Linear Actuator.

Figure 18 shows the wiring from the adapter to convert the PQ12 actuator ribbon to wires that can be plugged into the LAC board. The pins leaving the adapter and connecting to the LAC Board are: P+ and P- feedback potentiometer reference rail (+ve, -ve), M- and M+ actuator motor power and feedback potentiometer. Additionally, in order to create actuator movement, the LAC board and the Arduino nano (micro-controller) must have a common ground. This is shown by the pink wire in Figure 17 above.

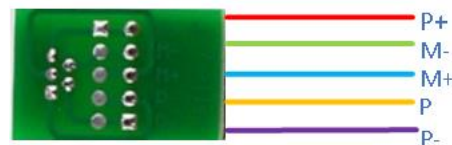


Figure 18. Wiring diagram of adaptor.

4.2.1.2 Final Design Proposal

To reduce the time lag between actuators, it was decided that a shared Arduino would be used to control each actuator. This entailed connecting multiple actuators to a Nano Arduino and a power supply. In this design, the number of Arduinos used was reduced from four to one. This provided a weight saving of 7 g per snake module in addition to reducing the overall material cost by £60.

There were two methods for wiring up the actuators to the one Arduino:

- 1) Each actuator has its own pin to connect to on the Arduino
- 2) All actuators share one pin of the Arduino

There were advantages and disadvantages of each option. The main advantage of the first choice is that if it were desired for the housings to extend and retract at different times or for different amounts, each actuator would require its own separate command signal, and so its own pin. This would be useful for future development of the robotic snake. However, for the current design, the key aim was for the actuators to move to the same maximum extension at the same time. Thus, the main criterion was to produce in-sync actuator movement. Through experimental testing, the second design Figure 19 below was chosen due to its minimal lag between actuators. However, from time to time, a delay between actuator motion was identified. This issue will be discussed further in Section 4.4.2. along with a plan for mitigation.

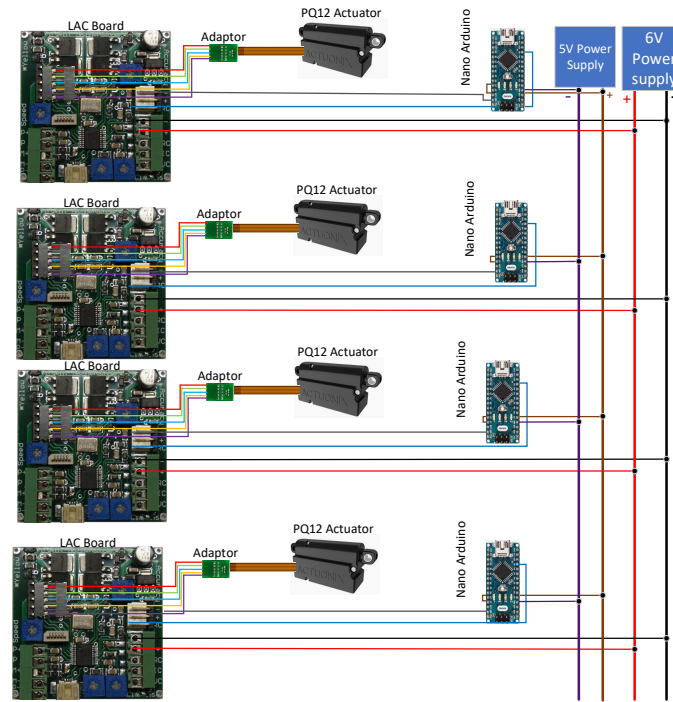


Figure 19. Multiple actuators connected to on pin each of the Arduino Nano.

4.2.2. Serpentine and Sidewinding motion (Servo motors)

4.2.2.1. Motor Selection

Servo motors were chosen due to their precise control of angular position, velocity, and acceleration. Additionally, servos have a high power-output to size and weight ratio which is ideal for a modular-based setup. Position feedback can also be provided by “smart” servo motors, a useful feature when producing robot motion.

DYNAMIXEL was chosen as the ideal selection due to software stability, greater support frameworks within software packages (such as ROS), and wider documentation online. Various models can be seen in Table 9 below. The AX-12A motor was the final choice due to its lower price and acceptable torque requirement

Table 9. AX Series DYNAMIXEL Motor Specifications.

Motor	Voltage [V]	Torque [N m]	Current [A]	Speed (No Load) [rpm]	Link	Size [mm]	Weight [g]	Price [£]
AX-12A	9 ~ 12	1.50	1.5	59	TTL	32x50x40	54.60	40
AX-12W	9 ~ 12	0.21	1.4	470	TTL	32x50x40	52.90	40
AX-18A	9 ~ 12	1.80	2.2	97	TTL	32x50x40	55.90	87

4.2.2.2. Wiring Diagram

The motors were connected in series using a daisy-chain formation in order to simplify the wiring. The initial hardware workflow map of the components for the motor control are shown in Figure 20, this has been simplified to show only one motor. The laptop acted as the microcontroller using a ROS environment for control. The U2D2 device allowed the integration of the AX-12A motors with the laptop by connecting to the 3-pin port. The 3-pin port then connects the input/output wires (GND, V_{DD}, SIGNAL) of the motors. The motors were then daisy-chained together. The SMPS2 DYNAMIXEL was connected to the last motor and acted as a power supply adapted to link the SMPS 12 V 5 A power supply to the AX-12A motors.

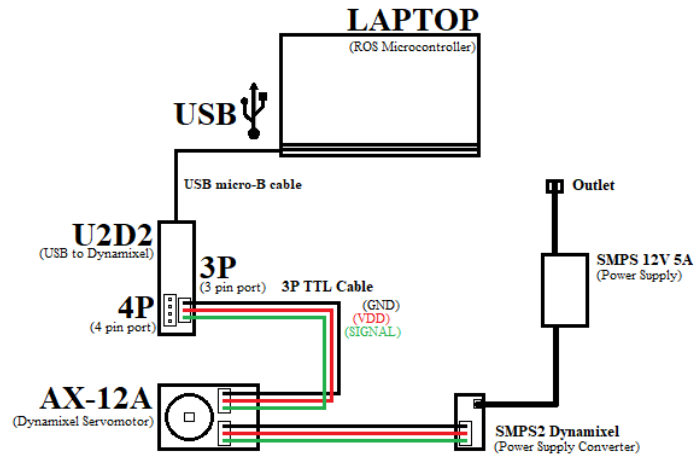


Figure 20. Initial hardware workflow map of motor control.

The laptop (with Xenial Ubuntu 16.04 and ROS Kinetic) acted as the microcontroller for the motors by creating scripts that are hosted on “nodes” through ROS kinetic. This sent a list of instructions to control the motors. Additionally, the servomotors provide position feedback that was sent to the laptop allowing to fine tune the control.

When daisy-chaining the motors together, the current drawn by each motor is summed together. If this sum value exceeds the current of the power supply, the servomotors will under-perform and thus conduct slower movements. Fortunately, as only 5 motors were to be used, the sum of the current was less than the current capacity of the power supply.

To further reduce complex wiring, a 6-port power hub was integrated into the hardware workflow of the motor control. This enabled the power supply and laptop to both be tethered at one end of the chain therefore the movement of the snake will not be constrained. The new hardware workflow of the motor control is shown in Figure 21 below.

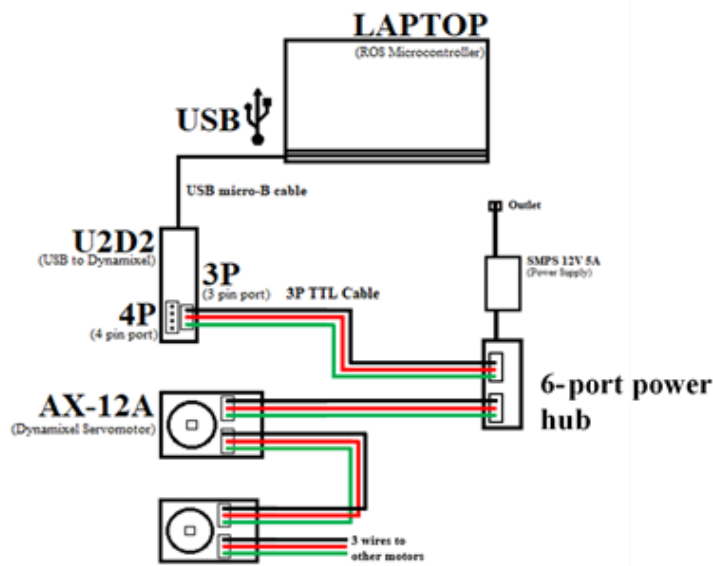


Figure 21. New hardware workflow map of motor control.

4.2.2.3. Motor Control and Configuration

ROS

The modular snake robot in this project utilised the Robot Operating System (ROS). ROS is named as an operating system, but it is more suitably described as a collection of software frameworks. These software frameworks are designed to act as a middle ground between hardware devices of different types and quality such as motors and sensors. One advantage of ROS is its wide variety of applications: simulation, where ROS integrates with Gazebo (a 3D simulator) that can create 3D scenarios to test complex indoor and outdoor environments; mapping and localization of environments; environment navigation; ROS nodes and ROS topics, etc. The disadvantage of ROS is its limited compatibility with a few Linux distributions, and little support for other operating systems such as Mac OS and Windows, as mentioned briefly within the initial investigations conducted (Appendix 4).

For the purpose of this project, ROS was being utilised in its application of ROS nodes and ROS topics, controlled via a laptop using the Xenial Ubuntu 16.04 Linux operating system. ROS nodes can either subscribe to another node to receive information or publish information to other nodes. Topics facilitate the communication between nodes and act as the “bridge” where messages are sent or received. These messages can consist of instructions for motor movements, diagnostic information on motors, or sensor feedback data.

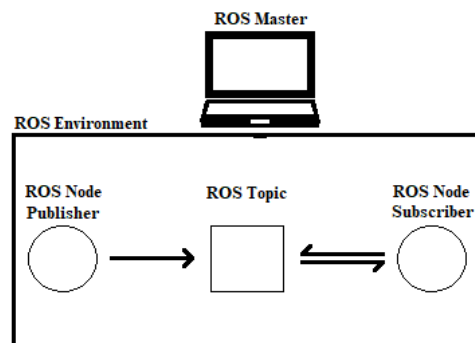


Figure 22. Basic ROS node and ROS topic setup.

Figure 22 is an example of a basic ROS node and topic setup. A ROS compatible device can act as the ROS Master, which creates a ROS environment, that tracks the nodes and topics linked to the IP address of the ROS Master device. A ROS node can publish a message to a ROS topic and is referred to as a publisher node, a ROS subscriber node listens to and receives information from this ROS topic. It is also possible for a single ROS node to act as both a subscriber and a publisher. Appropriate actions can then be scheduled depending on the type and content of the information received.

By utilising ROS nodes and topics, almost instantaneous feedback is provided between the force sensors and motor control. The network of nodes and topics also has the additional benefit of future proofing the project; additional sensors, motors, or various hardware components can be integrated into the existing ROS Master environment to communicate with existing devices in the network in the future.

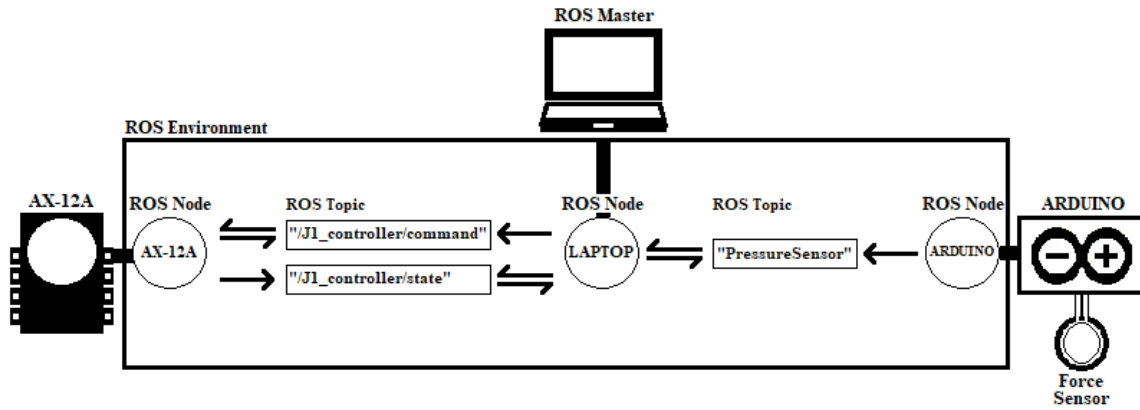


Figure 23. Modular Snake Robot ROS Master environment example.

Figure 23 is an example of the ideal ROS environment setup between the ROS Master, a DYNAMIXEL AX-12A motor, and an Arduino with a force sensor. The “LAPTOP” node can simultaneously read inputs from the “/J1_controller/state” and “ForceSensor” topics which can then influence the information sent to the “/J1_controller/command” topic to control the movement of the motor.

ROS initialization of motors

Every DYNAMIXEL motor possesses an ID number, at default this is set to 1. The ID is important in distinguishing between the different motors in any given setup. A “.launch file” can be used to identify, initialise, and set parameters to motors based on their ID number. If two motors possess the same ID number, then only one motor will be initialised and have its parameters set whilst the other motor is disregarded. Since all motors default to an ID number of 1 it is important to manually set the ID of each motor individually. This can be done by using the “DYNAMIXEL Wizard” software package (ROBOTIS, 2020).

It was mentioned in initial motor selection that the benefit of using a DYNAMIXEL motor is with the in-built control board that can provide position feedback. Another benefit of using a DYNAMIXEL motor is that there are official software packages from the manufacturer that allow the process of initialising the motors to a ROS Master environment to be more streamlined.

```

1 <!-- -*- mode: XML -*- -->
2
3 <launch>
4   <node name="dynamixel_manager" pkg="dynamixel_controllers" type="controller_manager.py" required="true" output="screen">
5     <rosparam>
6       namespace: dxl_manager
7       serial_ports:
8         pan_tilt_port:
9           port_name: "/dev/ttyUSB0"
10          baud_rate: 1000000
11          min_motor_id: 1
12          max_motor_id: 12
13          update_rate: 20
14     </rosparam>
15   </node>
16 </launch>

```

Figure 24. “controller_manager.launch” script.

A controller launch file, as seen in Figure 25, is executed by using the “roslaunch” tool within a command terminal by applying a terminal command with the following syntax:

```
roslaunch <catkin workspace source folder> <file-name.launch>
```

The “controller_manager.launch” script (Figure 24) was then used to identify and initialise the motors connected via the USB port.

```

1 <launch>
2 <!-- Start tilt joint controller -->
3 <rosparam file="$(find snake_robot)/tilt.yaml" command="load"/>
4 <node name="tilt_controller_spawner" pkg="dynamixel_controllers" type="controller_spawner.py"
5     args="--manager=dxl_manager
6         --port pan_tilt_port
7         J1_controller
8         J2_controller
9         J3_controller
10        J4_controller
11        J5_controller"
12     output="screen"/>
13 </launch>

```

Figure 25. “controller.launch” script.

The parameters for each motor was initialised with the “controller.launch” script, executed using the “roslaunch” tool following the same syntax utilised in Figure 25.

```

1 J1_controller:
2   controller:
3     package: dynamixel_controllers
4     module: joint_position_controller
5     type: JointPositionController
6     joint_name: snake_head
7     joint_speed: 3
8   motor:
9     id: 1
10    init: 512
11    min: 0
12    max: 1023

```

Figure 26. “tilt.yaml” script excerpt.

The “controller.launch” script initialises the parameters including the motors ID, joint speed, and controller type via the referenced “tilt.yaml” script as seen in Figure 26. The referenced controller package within the “tilt.yaml” script initialised the ROS nodes and appropriate ROS topics to the ROS Master environment. For example, using Figure 26 above, the “tilt.yaml” script created a publisher ROS node and subscriber ROS node. The publisher ROS node sent motor diagnostic information to the ROS topic “/J1_controller/state”. The subscriber ROS node listened to the ROS topic “/J1_controller/command”, once it received a message of the type “std_msgs/Float64” it called a service to move the motor to the angle (in radians) specified in the message.

ROS Node Manipulation

The existing ROS topics within the ROS Master environment was determined by executing the terminal command with the following syntax: `rostopic list`. This shows a list of the ROS topics present within the ROS Master environment.

ROS initialisation of motors discussed the method of ROS nodes and ROS topics being created to the ROS Master environment with the DYNAMIXEL software packages. However, using Figure 23 above as an example, sending a message to the ROS topic “/J1_controller/command” (to manipulate the motors position) or receiving messages from the ROS topic “/J1_controller/state” (to utilise position feedback) required the creation of additional ROS nodes. Publisher and subscriber ROS nodes are commonly created with C++ or Python scripts. For the purpose of this project, Python was used in conjunction with the “rospy” libraries to manipulate the ROS Master environment.

Publisher ROS Node

In a ROS environment Python scripts were called with the “roslaunch” command rather than the Python interpreter. Since the Python scripts were called without the preceding language interpreter, the

Python script began with the following syntax: `#!/usr/bin/env python`. This allowed the script to call the Python interpreter which then compiled and ran the rest of the script.

The next step was to import the dependant libraries. For a publisher ROS node there are two important libraries:

```
import rospy
from std_msgs.msg import Float64
```

“rospy” is a Python library that enables the interaction between ROS. As mentioned in the previous section, importing “Float64” from “std_msgs.msg” allowed the message type “std_msgs/Float64” to be reused.

```
snake_head = rospy.Publisher('/J1_controller/command', Float64, queue_size=1)
rospy.init_node('Publisher')
```

The above syntax details that variable “snake_head” published to the topic named “/J1_controller/command” with the message type “std_msgs/Float64”. A publisher ROS node was then initialised under the name “Publisher”. It is possible for one publisher ROS node to advertise messages to multiple topics.

```
snake_head.publish(0)
```

The “publish” command linked to the “snake_head” variable then advertised the message contained within the closed brackets to the “/J1_controller/command” topic. The initialised subscriber ROS node interpreted the message as 0 radians and moved the specified motor to the appropriate radial position.

Subscriber ROS Node

As mentioned previously, the ROS node can act as both a publisher and subscriber within the Python script. Therefore, following on from the previous section, another dependant library was included:

```
from dynamixel_msgs.msg import JointState
```

The subscriber declaration that subscribed to a particular topic is as follows:

```
rospy.Subscriber('/J1_controller/state', JointState, callback_function)
```

When this command is run, it takes the message contained within the topic “/J1_controller/state” of the type “dynamixel_msgs/JointState” and is was used as an argument in the “callback_function”.

There are a few ways to design the callback_function, in this case, it is more important to get the message contained within the topic “/J1_controller/state” and assign it to a Python variable. This allowed manipulation or execution of certain functions once a certain condition or value is assigned.

```
def callback_function(message):
    global motor_pos
    motor_pos = message.current_pos
```

The “callback_function” assigned “motor_pos” as a global variable so that when it was manipulated its value was used within the main function. The value assigned to “motor_pos” took the member “current_pos” of the “message” argument. This function allowed the position of the motor to be read whenever it is called.

The “message” argument is of type “dynamixel_msgs/JointState” and had the following members:

```

string name          # joint name
int32[] motor_ids    # motor ids controlling this joint
int32[] motor_temps  # motor temperatures, same order as motor_ids
float64 goal_pos     # commanded position (in radians)
float64 current_pos  # current joint position (in radians)
float64 error        # error between commanded and current positions (radians)
float64 velocity     # current joint speed (in radians per second)
float64 load         # current load
bool is_moving       # is joint currently in motion

```

In the above example only the “current_pos” member was captured, other members can also be used to execute relevant functions once a specified condition had been met.

4.2.2.4. Motion Control

The purpose of the motors was to mimic the movements of a snake. More specifically, the serpentine and sidewinding motions as mentioned in section 2.1. The motors received inputs that translated the motors position to the radian specified. The modular snake robot was loosely defined as a “robotic manipulator composed of a set of links connected together by joints” it was assumed that the use of forward and inverse kinematics provides valuable information in terms of motion planning. By knowing the radian inputs, the current position of each joint was then calculated with forward kinematics; inverse kinematics gave the desired radian inputs given required joint positions. However, this assumption was not entirely accurate, and is discussed in more detail within section 3.3.

To achieve the expected sidewinding and serpentine motions, the modular snake robot modelled a sine-wave pattern for the motors to replicate.

Imposed Sine-wave Motion

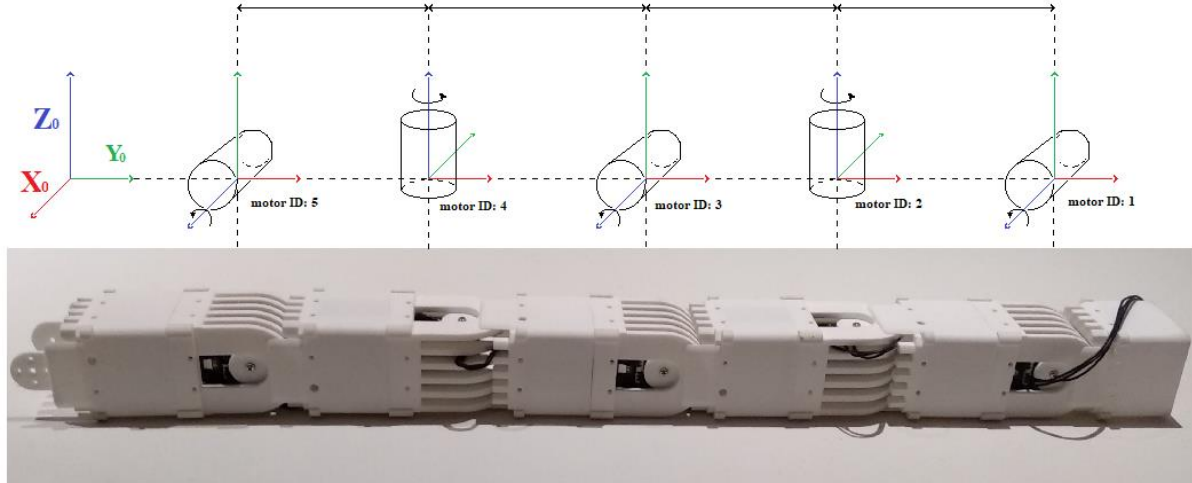


Figure 27. Orientation for 5-motor prototype.

The key method in replicating the movement of a snake in either sidewinding or serpentine motions was to induce a sine-wave pattern for the motors to follow as supported by Mu et al. (2017). With reference to Figure 27, and taking (X_0, Y_0, Z_0) as the reference base coordinate, the motors for a 5 module snake were sequentially off-set by 90° such that motors 1, 3, 5 rotate on the X_0 axis and motors 2, 4 rotate on the Z_0 axis. Therefore, grouped motors 1, 3, 5 and 2, 4 followed a separate sinewave. Taking each module of the snake robot to be sequentially numbered ($n = 1, 2, 3, 4, 5$), where $n = 1$ at the head of the snake and $n = 5$ at the tail of the snake, the equation of motion was represented as (Mu et al. 2017):

$$\theta(n, t) = \begin{cases} \sin(V_n t) & , n = 1, 2, 5 \\ \sin(V_n t + \alpha) & , n = 3, 4 \end{cases} \quad (12)$$

Where n is the joint number, V_n is the joint speed of joint n , α is the phase difference, t is the elapsed time, and θ is the current angle of the joint.

The equation of motion had been adapted into Python in the form of matrices.

```
# Initial Backward Movement Matrix
# ---
FiveMotor_BackInit = np.array([
    [ pos_2, 0, 0, 0, pos_2 ],
    [ 0, 0, 0, 0, 0 ],
    [ 0, 0, pos_1, 0, 0 ]
])
```

Figure 28. "Backwards" initial movement matrix for 5 module snake robot.

```
# Backward Movement Matrix
# ---
FiveMotor_BackMov = np.array([
    [ pos_1, 0, 0, 0, pos_1 ],
    [ 0, 0, 0, 0, 0 ],
    [ 0, 0, pos_2, 0, 0 ],
    [ 0, 0, 0, 0, 0 ],
    [ 0, 0, 0, 0, 0 ],
    [ pos_2, 0, 0, 0, pos_2 ],
    [ 0, 0, 0, 0, 0 ],
    [ 0, 0, pos_1, 0, 0 ],
    [ 0, 0, 0, 0, 0 ],
    [ 0, 0, 0, 0, 0 ],
    [ 0, 0, 0, 0, 0 ]
])
```

Figure 29. "Backwards" movement matrix for 5 module snake robot.

The initial movement matrix, as shown in Figure 28, adjusted the motors so that the movement matrix (Figure 29) acted as a repeating loop, as necessary. With reference to Figure 29, each row represented a value in the time domain, and each column represented one of the 5 motors in the 5-module snake robot setup. The motors were expected to float between two specified positions, "pos_1" and "pos_2", set by the user. In this example, "pos_1" and "pos_2" were set to 0.785 radians and -0.785 radians, respectively.

```
def motion(mov_array):
    columns = np.size(mov_array,1)
    rows = np.size(mov_array,0)
    time.sleep(time_delay)
    # Rows
    for i in range(0,rows,1):
        # Columns
        for j in range(0,columns,1):
            # Pass Filter
            if mov_array[i][j] == 0:
                continue
            else:
                # Snake Head
                if j == 0:
                    snake_head.publish(mov_array[i][j])
                # Snake Body 1
                if j == 1:
                    snake_body1.publish(mov_array[i][j])
                # Snake Body 2
                if j == 2:
                    snake_body2.publish(mov_array[i][j])
                # Snake Body 3
                if j == 3:
                    snake_body3.publish(mov_array[i][j])
                # Snake Body 4
                if j == 4:
                    snake_body4.publish(mov_array[i][j])
        time.sleep(time_delay)
    signal.signal(signal.SIGINT, signal_handler)
```

Figure 30. Motor matrix motion function.

The motor motion function, as seen in Figure 30, was standardised so that it can be used for any sized matrix. Hence, the motor function can be applied to a modular snake robot of any size, by calculating the number of columns and rows of the matrix argument. The motor function then cycled through the matrix by using “for” loops. Note that a “0” within the matrix does not represent a position of 0 radians, it was treated as a pass condition. Once it iterated through the columns, a time delay of 0.0872 seconds is induced before iterating through the next row. The total time elapsed between the two specified points “pos_1” and “pos_2” was calculated by dividing the total distance (in radians) by the joint speed (radians per second). Therefore, the time delay between each row iteration was calculated in the standard form as:

$$\text{Row time delay} = \frac{\left(\frac{(|\text{position 1}| + |\text{position 2}|)}{\text{Joint Speed}} \right)}{\left(\frac{\# \text{ rows in movement matrix}}{2} \right)} \quad (13)$$

This equation holds, and was more suitable, when the number of rows is kept even.

Note that Equation 12 is a general formula for snake-like motion, the pattern and sequence differed for the different snake-like motions – serpentine and sidewinding. Take the matrix and motion profile for the “left sidewinding” movement as an example shown in Figures 31 and 32.

```
# Initial Left Movement Matrix
# ---
FiveMotor_LeftInit = np.array([
    [ pos_2,    pos_2,    0,    0,    pos_1 ],
    [ 0,        0,        0,    0,    0 ],
    [ 0,        0,        pos_1,    pos_1,    0 ]
])
```

Figure 31. “Left sidewinding” initial movement matrix for 5 module snake robot.

```
# Left Movement Matrix
# ---
FiveMotor_LeftMov = np.array([
    [ pos_1,    pos_1,    0,    0,    pos_2 ],
    [ 0,        0,        0,    0,    0 ],
    [ 0,        0,        pos_2,    pos_2,    0 ],
    [ 0,        0,        0,    0,    0 ],
    [ 0,        0,        0,    0,    0 ],
    [ 0,        0,        0,    0,    0 ],
    [ pos_2,    pos_2,    0,    0,    pos_1 ],
    [ 0,        0,        0,    0,    0 ],
    [ 0,        0,        pos_1,    pos_1,    0 ],
    [ 0,        0,        0,    0,    0 ],
    [ 0,        0,        0,    0,    0 ],
    [ 0,        0,        0,    0,    0 ]
])
```

Figure 32. “Left sidewinding” movement matrix for 5 module snake robot.

The motors followed a non-sinusoidal waveform when it rotated between two defined angles (in radians). This is because the joint speed of the motors was constant and the general formula for snake-like motion was adapted via matrices in Python.

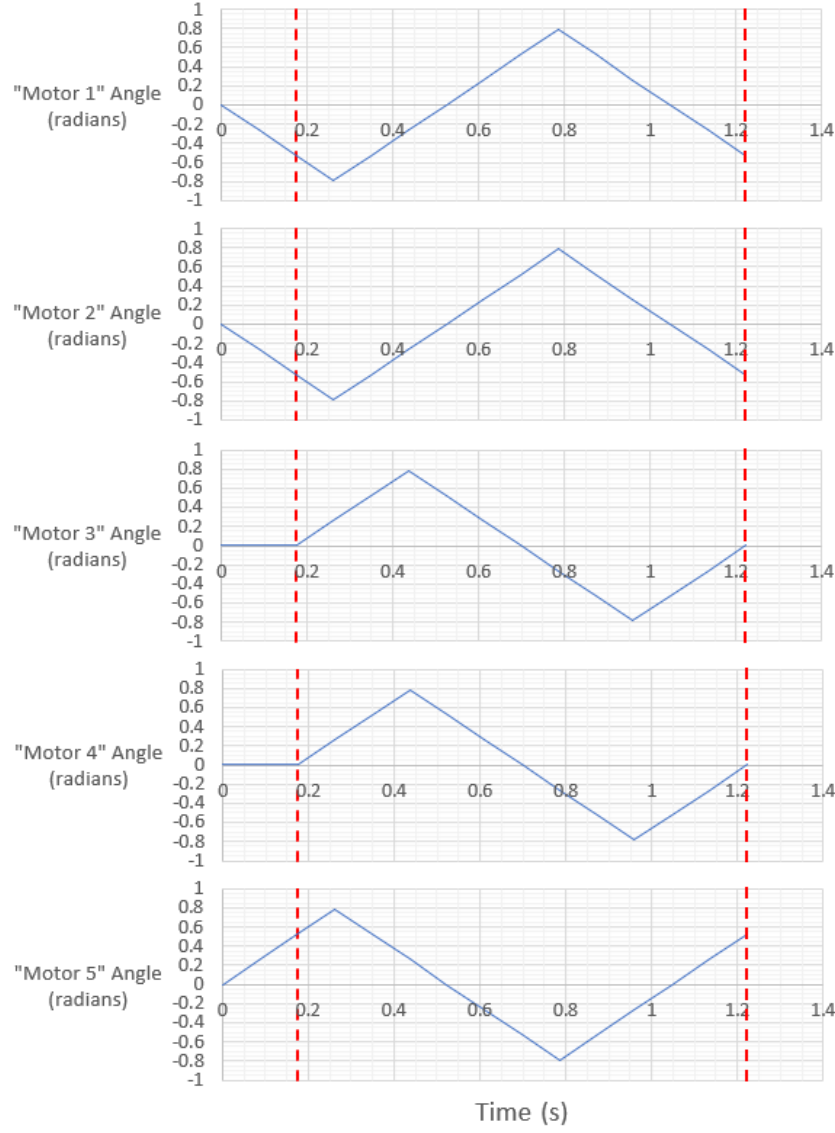


Figure 33. "Left sidewinding" movement motion profile for 5 module snake robot.

Figure 33 is an example of motors following a triangle wave of motion in the "left sidewinding" movement. The initial movement matrix (Figure 31) produced the movement that precedes the initial red dashed line and allowed the motors to then follow the movement matrix (Figure 32) which produced a symmetric motion shown within the dotted red line boundary. The triangular waveform followed the general equation:

$$\theta(n, t) = \pm \begin{cases} V_n(t - \delta_t) & , \delta_t \leq t \leq \frac{T}{4} + \delta_t \\ -V_n(t - \delta_t) + \frac{\pi}{2} & , \frac{T}{4} + \delta_t \leq t \leq \frac{3T}{4} + \delta_t \\ V_n(t - \delta_t) - \pi & , \frac{3T}{4} + \delta_t \leq t \leq T + \delta_t \end{cases} \quad (14)$$

Where θ is the current angle of motor n at time t , V_n is the joint speed of motor n , t is the elapsed time, δ_t is the time delay between motors, T is the period of the waveform.

Function Process Time

The motion of the snake, in the form of induced triangle waves to the motors, was extremely time sensitive. The snake propelled forward only if certain sections of the snake were in contact with the ground at any given moment. This raised a concern over the use of matrices within Python. In theory, each row represented a different value in the time domain. For example, row 1 was expected to be at $t = 0$ seconds. This was not entirely accurate for each column as each iteration of the function had a process time. Hence, whilst the first column was accurate to be a reference as $t = 0$ seconds, the position of the 5th column in the time domain was more accurately portrayed as:

$$t_c = R(\text{row time delay}) + A_c \delta \quad (15)$$

Where t_c is the position in the time domain for a certain column, c , R is the current row (starting at $R = 0$), row time delay as calculated in Equation 13, A_c is a constant depending on the current column – column 5, A_5 , will have a maximum value of 4, δ is the process time.

The process time was then estimated by using the following syntax:

```
import time
start = time.time()
# insert section of code here to be measured
end = time.time()
process_time = end - start
```

With the above syntax between the iteration of columns being used, the process time for the function to iterate through each column was then calculated. Note that the precision on the time library function on a Linux based system was estimated to be around $\pm 1 \mu\text{s}$ – and was considered negligible.

Table 10. Process time for each movement matrix.

Movement [type]	Average process time between each column iteration $\Delta [\mu\text{s}]$	Average process time between column 1 and column 5 $A_5 \delta [\mu\text{s}]$	Total time elapsed between “pos_1” and “pos_2” [s]	Percentage of $A_5 \delta$ against total time elapsed [%]
Forward Serpentine	33	132	0.523	0.0252
Backwards Serpentine	36	144	0.523	0.0275
Right Sidewinding	69	276	0.523	0.0528
Left Sidewinding	74	296	0.523	0.0566
Clockwise Sidewinding	62	248	0.523	0.0474
Anti-clockwise Sidewinding	60	240	0.523	0.0459

Table 10 details the process time for each type of movement and its percentage against the total time elapsed between the two specified motor positions. This setup contained 5 modules (hence, 5 motors) with joint speeds of 3 rad s^{-1} with “pos_1” set at 0.785 radians and “pos_2” set to -0.785 radians. This gave the total time elapsed between “pos_1” and “pos_2” as 0.523 seconds. Given that the highest percentage was “0.0566 %” from the “Left Sidewinding” movement type, the process time delay between the first and last motor was considered negligible.

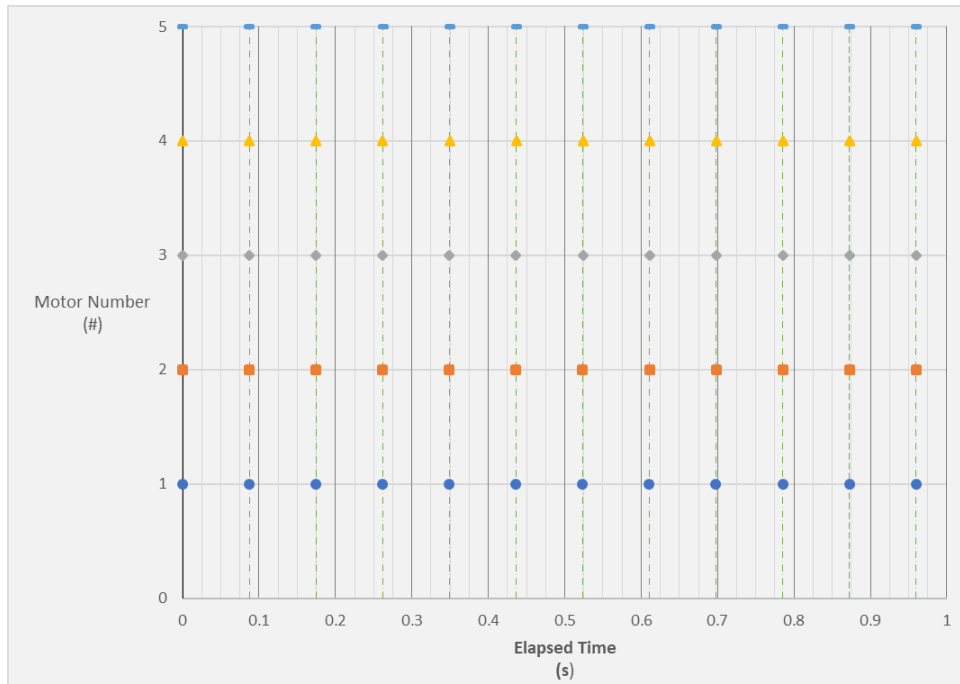


Figure 34. Goal position process execution profile for “left sidewinding” movement matrix.

Figure 34 is a visual representation of the motors receiving commands during the “left sidewinding” motion – taken as an example as this matrix profile contained the largest margin of error in terms of time delay between motors. The x-axis represents the elapsed time in seconds, and y-axis represents the motor number. Data points represented the time at which the motor received and executed its specified goal position (where it moved to either position 1, position 2, or not move at all) as shown in the matrix within Figure 32 – and included the time delay induced by the process time for the function shown in Figure 30. The vertical dotted lines show the time periods at which the motors must move to their next specified goal position. The distance between each dotted green line is the row time delay (0.0872 seconds) between each row of the movement matrix, given by Equation 13. Instructions for all 5 motors needed to be sent at the same time to ensure that there were no unaccountable delays between them. Figure 34 and Table 10 confirm that the process time delay induced by the matrix function was negligible and was assumed as being processed “at the same time”. Therefore, the movement of the snake robot was not affected by time delays inherent in a program’s processing time. It was understood that this delay will increase as the number of modules increase. However, as the number of modules used in order to meet the outcome of this project will not exceed five, this is not a current concern.

The matrices for the other motions (Forward serpentine, right sidewinding, clockwise sidewinding, and anti-clockwise sidewinding) can be found under Appendix 2.

4.3. Sensors

4.3.1. Sensor selection

Theoretically, sensors allow the robot to simulate touch, sight, have audible capabilities, and perform motion. Additionally, some sensors provide feedback regarding the environments and the terrain. The aim of incorporating sensors on the snake-like robot is to detect/avoid the collision of the robot during movement. This will be accomplished by positioning sensors on the first module (head) of the robot. These sensors will then detect obstacles/collisions and will signal the robot to stop moving and avoid the obstacle. Another objective is to detect which modules are in contact with the ground/surface and

determine the force acting at different points on the module to allow for in depth motion analysis of the robot. By incorporating these results, this will aid improvement of each motion.

Various sensors have been considered to achieve these objectives such as LiDARs and force/torque sensors. LiDARs are an example of vision and proximity sensors, and function similar to sensor types required in autonomous vehicles such as cameras, infrared, sonar, ultrasound and radar (FRANK, 2017). These sensors enable the robot to identify the location and size of an object, or to detect the presence of any obstacle near the robot without making physical contact. On the other hand, force/torque sensors (FT sensors) work fundamentally by making physical contact in order to gather information regarding the environment. Examples for FT sensors are tactile sensors and force sensitive resistor (FSR) sensors.

4.3.1.1. LIDARs

LiDAR sensors were taken into consideration to be used on the first module to act as the 'eyes' for the snake robot. LiDAR sensors are based on the Time of Flight (ToF) principle. This operates by projecting a small light onto a surface of an object and measuring the time it takes for the light to be reflected back to its source. This is illustrated in Figure 35.



Figure 35. TF Mini optical simulation of the optical path.

One of the advantages of this sensor is that it can detect an object from a certain distance precisely with high sensitivity and high-speed distance detection. A small LiDAR sensor (TF Mini LiDAR from Seeedstudio Grove) was considered for this project to enable packaging into the snake modules. The component of the sensor is as shown in Figure 36a and its size specification shown in Figure 36b.

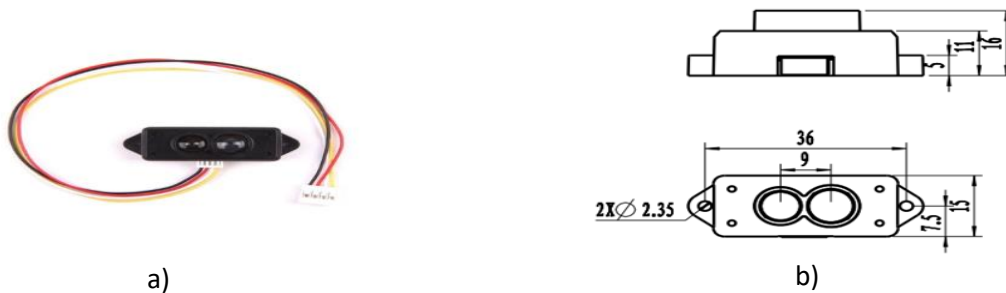


Figure 36. a) TF Mini LiDAR b) TF_mini left module size drawing (unit: mm).

This sensor is also incorporated with distinctive electrical and optical designs as shown in Table 11 and 12 below (SEEDGROVE, 2018). This mini LiDAR can operate within 0.3 m up to 12 m. These distances are dependent on the environment the measurements are taken in; the effective detection distance decreases when taken outside due to varying levels of illumination from the sun. This is depicted in Figure 37 below. Note: all distance parameters are set under the opposite direction with the object to be detected.

Table 11. TF_mini electrical characteristics.

Item	Symbol	Typical value	Unit
Input voltage	DC	5	V
Average power	P	≤120	mW
LED peak current	I_{\max}	800	mA
Serial port TTL voltage level	V_{TTL}	3.3	V

Table 12. Optical characteristics.

Parameter	Symbol	Condition or description	Typical value	Unit
FOV	β		2.3	Degree
Resolution	Re	Sensitivity to distance change	5	mm
Wavelength	λ		850	nm

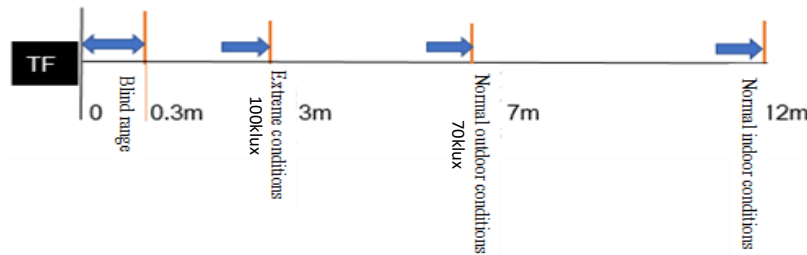


Figure 37. Measurement range schematic diagram.

The field-of-view (FOV) is defined as the angle of the emitted LiDAR signals that are covered by the sensor. Essentially, the FOV determines the side lengths of different detection ranges. For the TFmini model, the FOV is equal to 2.3. The acceptance angle per metre is illustrated in Figure 38 below.

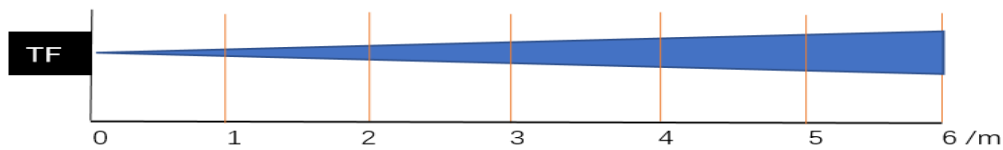


Figure 38. Detection range (acceptance angle) schematic diagram.

The area of the detection range per metre due to the FOV angle is detailed in Table 13 below. The distance represents the vertical length between the object to be detected and the sensor, stated in meters, whilst the side length of detection range is expressed in millimetres. Generally, if the side length of the object to be detected is greater than the detection range side length, the output data from LiDAR can be considered reliable, otherwise the output data of LiDAR may be varied and may cause the error to increase. (SEEDGROVE, 2018)

Table 13. Relationship between detection range and distance.

Distance/m	1	2	3	4	5	6
Detection range side length/mm	40	80	120	160	200	240

Although the LiDARs sensor can provide required information instantly at high accuracy, there are also several disadvantages of using this sensor for this project. These are detailed below:

1. This sensor is expensive compared to other type of sensors and has high operating costs.
2. Very large datasets that are difficult to interpret: LiDAR is a technology that collects a significant number of datasets which require high level of analysis and interpretation. For this reason, it may take a lot of time to analyse the data.
3. Because of the datasets collected are huge and complex, it may require skilful techniques to analyse the data which add up to the overall cost.
4. The operating range for this sensor is between 0.3 m to 12 m. If there is any obstacle sits below 0.3 m from the sensor, the sensor will not be able to detect it as it will fall into the blind range. Therefore, using this sensor on the snake robot would be ineffective.

5. These sensors are unreliable when used on water surfaces or where the surface is not uniform. The returned data will be inaccurate due to high water depth will affect the reflection of the pulses.

Primarily due to point 4, it was concluded that a LiDAR sensor was not relevant for this project. However, this type of sensor may be useful for future integration to allow the robot to choose the clearest path in hazardous conditions and for Search and Rescue (SAR) operations as it can detect obstacles from a distant.

4.3.1.2. Tactile Sensors

Tactile sensors are modelled based on the human sense of touch; the cutaneous sense and the kinaesthetic sense. The cutaneous sense is the sense related to skin which can detect stimuli in response to the mechanical stimulation, temperature, pain and pressure. Kinaesthetic sense is related to the touch and movement sense where the receptors inside the human's muscle, tendons and joints will receive sensor input (ElProCus, 2016).

Yuan, Zou and Sun (2019) proposed the use of triboelectric tactile sensors in their caterpillar-like soft robot. The sensors were attached to the back of the caterpillar robot so that the robot can detect when the back touches any obstacle. The sensors were manufactured from silicone rubber, this complex process is depicted in Figure 39.

Operating principle of triboelectric tactile sensors

The triboelectric tactile sensors are based on the coupling effect of triboelectrification effect and electrostatic induction (Dickey, 2017). These produce electrical signals which determine whether contact with an obstacle has occurred.

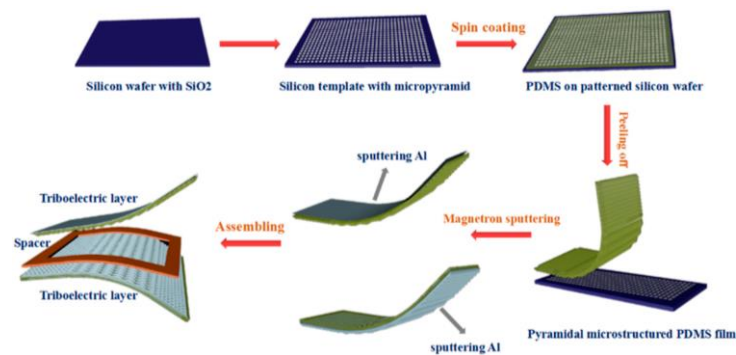


Figure 39. Triboelectric tactile sensor manufacturing process. (Wang et al., 2019)

Electrostatic induction refers to the phenomenon of redistribution of charge due to the existence of external charges. Triboelectrification effect is the transfer of charge when there is contact between two different materials of varying electron-binding ability. This charge transfer leads to the two materials having the equal quantity of opposite charges creating a potential difference (Figure 40a). The negative charge on the rubber surface of the sensor drives free electrons to the ground producing a current and voltage output (Figure 40b). As the object moves away from the sensor and the separation distance between the triboelectric tactile sensor decreases, the potential difference decreases (Figure 40c). This process causes the electrons to flow away from the ground to generating a voltage in the opposite direction. When the object is far enough away from the sensor, electrostatic balance is achieved and the electron stops moving and so no current and voltage is produced (Figure

40d)(Tao et al., 2018). It is the change in voltages that deem whether the sensor has made contact with an object.

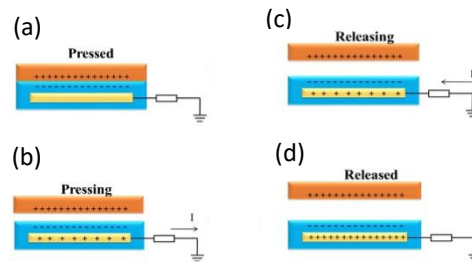


Figure 40. Working process of triboelectric tactile sensor: a) force applied on sensor, b) object starts to move away from sensor, c) object far enough from sensor causing electrostatic balance achieved & d) contact force detected

Although this type of sensor is helpful to achieve the objectives of the snake robot sensors because it can provide the information for collision detection more accurately, the manufacturing time and cost to produce the sensors is extensive. It also requires more complex equations to convert forces in the global coordinate system for data analysis which can be very time consuming. As a result, it was decided that this sensor was not suitable for the project.

4.3.1.3. Force Sensitive Resistors (FSR) sensors

A force sensitive resistor (FSR) sensor is a device which changes its resistance depending on the force and pressure applied to its surface. FSRs are very easy to produce and are cheap. The downsides of this sensor are that it is not accurate and not suitable for precision measurement. However, for most touch-sensitive applications they are good for detecting if a force has been applied on a surface or not. Since the amount of force applied is not a crucial aspect in detecting collision on the robot, this sensor can be considered fit for this project. The advantages of using FSR sensors are listed below (www.robotshop.com, n.d.):

- Can be used to provide a 0 – 5 V output allowing for it to be measured by the micro-controller (Arduino)
- Light weight: approximately 0.01 kg each
- Actuation Force as low as 0.1 N and sensitivity range to 10 N
- Highly Repeatable Force Reading - as low as 2 % of initial reading with repeatable actuation system
- Cheap: roughly £ 4 each
- Ultra-thin allowing for ease of packaging on the housing 0.35 mm

Operation Principle of FSRs

FSRs are created by two substrate layers followed by a conductive film and a plastic spacer, which contains an opening aligned with the active area. After the spacer layer there is a conductive print on substrate. This is illustrated in Figure 41.

When external force is applied to the sensor, the conductive film is deformed against the substrate. Air from the spacer opening is pushed through the air vent, and the conductive film touches with the conductive print on the substrate. The greater the contact area of these surfaces, the smaller the resistance. Thus, the more pressure applied on the sensor, the more the layers come into contact with the conductive film causing the resistance to decrease.

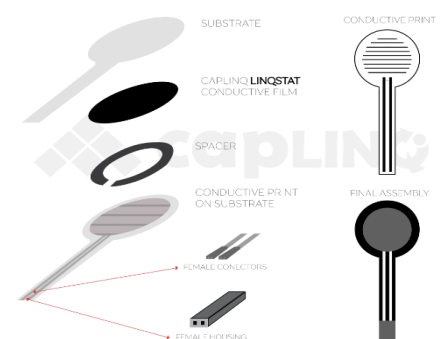


Figure 41. FSR sensor components. (CAPLINQ Blog, 2016)

4.3.2. Sensor Hardware

From research, it was decided that Force Sensitive Resistors (FSRs) would be used. The hardware configuration of the sensor to the Arduino is shown in Figure 42 below. It entails two connections from the FSR at one end of the FSR sensor to power input (5 V) and the other end to Analog 0 (A0) and ground (GND) via a 10k resistor. This creates the voltage divider circuit required to obtain varying resistance values when the FSR is loaded.

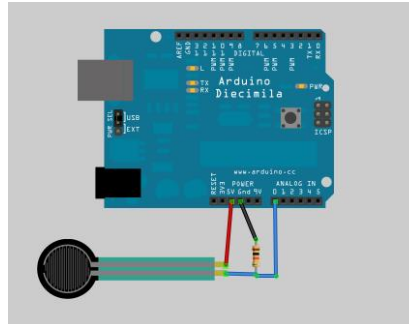


Figure 42. Wiring diagram of how sensor is connected to the Arduino board.

Using Arduino, these resistance values can be interpreted as a value 0 to 1023 (the maximum range of the 10-bit ADC in the Arduino), which can then be converted into a voltage between 0 – 5 V for ease of error diagnostics using the code shown in Figure 43 below. When no object is in contact with the conductive film, the Arduino will return the value as zero. When contact is made with conductive film, such as the ground during motion or the collision of a hazard, a voltage will be read. These results are printed in the Serial Monitor for real-time acquiring. If the voltage is greater than the threshold value of 0.5 V, it will be deemed as a collision for obstacle detection, or as a module in contact with the ground during motion analysis.

```
void setup() {  
  
  // initialize serial communication at 9600 bits per second:  
  Serial.begin(9600);  
}  
  
void loop() {  
  // read the input on analog pin 0:  
  int sensorValue = analogRead(A0);  
  // Convert the analog reading (which goes from 0 - 1023) to a voltage (0 - 5V):  
  //float voltage = sensorValue * (5.0 / 1023.0);  
  
  float voltage = map(sensorValue, 0, 1023, 0, 5);  
  Serial.println("Voltage = ");  
  Serial.println(voltage);  
  
  if (voltage>0) {  
    Serial.println("collision detected");  
  }  
}
```

Figure 43. Arduino code to achieve the threshold values for the sensor.

Using Arduino, these resistance values can be interpreted as a value 0 to 1023 (the maximum range of the 10-bit ADC in the Arduino), which can then be converted into a voltage between 0 – 5 V for ease of error diagnostics using the code shown in Figure 45 below. When no object is in contact with the conductive film, the Arduino will return the value as zero. When contact is made with conductive film, such as the ground during motion or the collision of a hazard, a voltage will be read. These results are printed in the Serial Monitor for real-time acquiring. If the voltage is greater than the threshold value of 0.5 V, it will be deemed as a collision for obstacle detection, or as a module in contact with the ground during motion analysis.

4.3.3. Sensor Implementation plan

The objectives of the sensors were to firstly detect the collision of the robot during movement by embedding sensors into the 'head' of the snake. During a collision, the sensors will make contact with the obstacle leading to a decrease in the resistance. This change in resistance would be recorded by the micro-controller (Arduino) and published to the Arduino node in the ROS network, as shown in the Testing section. If the value exceeds the threshold, it will be deemed a collision. This results in a command being sent to the motors via ROS to initiate a protocol to reroute the robot around the obstacle.

Another objective of the sensors was to detect which modules were in contact with the ground, and the magnitude of force acting on certain points of that module during motion. This would allow for further understanding of motion analysis in order to improve the robot movement. This would be accomplished by placing four sensors under the skin of each side of all modules. As the sensors will be plugged into the Arduino, the sensor values can be obtained real-time through the serial monitor function in the Arduino software.

Due to timing constraints, the initial plan to meet the objectives mentioned above were adapted. Thus, it was decided that prototype 2 for serpentine and sidewinding motion would only have sensors embedded into the first module (the robot's 'head') for collision detection. Additionally, the attachment methodology of the sensors to the housing surface under the skin will be determined through experimental testing. This was to dispute concerns that the sensors may not have secure enough attachment under the skin, or that the forces acting on the modules will not be measured efficiently due to the placement under the skin. This testing is discussed further in Section 5.2.2.

5. Integration and Testing

5.1. Prototype 1 – Rectilinear motion

5.1.1. Actuator Testing

Three tests were performed in order to assess the performance of the skin mechanism to extend and retract the housing and its effect on the skin layer. The areas that needed to be examined are listed below:

- Actuator retraction-extension motion
- Housing retraction-extension motion
- Function of the scales when the skin is in tension

5.1.1.1. Actuator motion testing

This test was used to ensure the actuator performed the motion required to extend and retract the housing to allow the protrusion and flattening of the scales on the housing surface. This testing had two objectives:

- 1) Determine the value to produce maximum extension and retraction of the actuator.
- 2) Ensuring the cyclic motion of switching between maximum extension and retraction is smooth.

As the RC mode of the actuator is based on the servo library, it was assumed that the maximum retraction value of the actuator was equal to 0° , the known minimum position of a servo when utilising the servo library, and the maximum extension value of the actuator was equal to 180° position, the known maximum position of a servo when utilising the servo library. By uploading the code in Figure 44 below to the Arduino, the actuator was commanded to extend to the equivalent value of 180° , subsequently the physical extension of the actuator was measured. The same process was then repeated for the value of 0° to determine the full retraction value.


```

#include <Servo.h>

Servo actuator;
int pos = 0; // variable to store the actuator position

void setup() {
  actuator.attach(9);
}

void loop() {
  actuator.write(180); // testing value for full extension
  delay(1000);
  actuator.write(0); // testing value for full retraction
}

```

Figure 44. Arduino code to establish full extension/retraction values.

To produce the cyclic motion of switching between maximum extension and retraction of the actuator, a sweep function was used. This is shown in Figure 45 below. The code included the values for maximum extension and retraction established in the first bit of testing. It commanded the actuator to start at maximum retraction and move in increments of 1 up to maximum extension, then back down again. As this code was written in the *main void loop*, this code ran continuously when all components were wired up and connected to the correct power supply.

```

#include <Servo.h>
Servo actuator;
int pos = 0; // variable to store the actuator position

void setup() {
  actuator.attach(9);
}

void loop() {
  for (position = 0; position <= 180; [position += 1)
  {
    actuator.write(position);
    delay(20);
  }
  for (position = 180; position >= 0; position -= 1)
  {
    actuator.write(position);
    delay(20);
  }
}

```

Figure 45. Sweep function of the actuator.

The next step was to ensure this motion was smooth by manipulating the magnitude of the delay and the size of the increment. It was determined that by increasing the increment from 1, the movement became heavily indexed affecting the smoothness of the actuator motion. It was decided that the step value should therefore stay at 1. When the magnitude of the delay was altered, the actuator was unable to complete its full cyclic extension-retraction motion, thus the maximum extension and retraction limits were not reached. By using 20 ms time delay, the actuators were able to produce the cyclic motion reaching the established limits at the maximum velocity.

5.1.1.2. Actuator in housing testing

The second test for the skin mechanism was used to determine whether the actuator cyclic movement would cause the housing to extend and retract. The objective of the test was to firstly ensure the actuator fit securely into the housing at both ends, and to also check that the housing extension and

retraction was consistent as a repetitive motion of the housing would ensure the scales of the skin would react the same every time.

5.1.1.3. Actuator in housing with skin attachment testing

The third, and final test for the skin mechanism was to test the housing extension with the skin attached to the module. This would deem whether the attachment would be able to withstand the tension force indirectly produced by the actuator, and how the scales would behave during the motion.

5.1.1.4. Prototype full build and testing

Once the housings were manufactured using selective laser sintering (SLS), the actuator components would have been inserted and the prototype made up of four modules, built. Unfortunately, the project ended early due to Covid-19, as a result we were unable to retrieve our SLS parts and complete the build. With more time, this testing would have been completed and a full prototype made for rectilinear motion.

Based upon our testing of one housing, we can assume that this prototype would have worked. With more housings, the robot would have had a higher pushing force therefore during rectilinear motion, the robot may have moved a greater distance under each cycle of the actuator extension-retraction motion.

5.1.2. Frictional Testing of Skin

The snake-skin layer function is heavily reliant on its interaction with the surface it is travelling on. Therefore, it was imperative that frictional testing was conducted in order to quantify these interactions through obtaining the static (μ_s) and kinetic (μ_k) coefficients of friction between the silicone layer and various substrates.

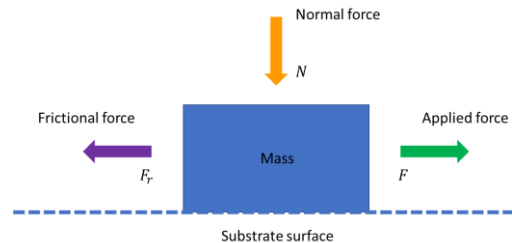


Figure 46. Diagram illustrating the force interactions to consider for a given mass moving along a substrate surface.

These frictional coefficients can be best described using the diagram above (Figure 46). When a force is applied to an initially stationary mass, the frictional resistance will increase to counteract any applied force until a certain limit, where friction between the surfaces is overcome and motion begins. It is this threshold of motion which is characterised by μ_s . Once moving at a constant velocity, the frictional force will also remain constant, although it will usually be less than the force required to start the mass moving. This is characterised by μ_k . Both coefficients can be substituted into the following equation for calculation:

$$F_r = \mu N \quad (16)$$

Where F_r = Frictional force, μ = coefficient of frictional, and N = Normal force.

Normal force is equivalent to the weight of the mass, assuming the applied force is applied parallel to the substrate surface.

5.1.2.1. Experimental procedure

The aim was to record data for a flat silicone layer and a silicone layer with scales in order to be able to compare their interactions for a variety of substrate surfaces. Although the pin-on-disc method was initially considered (as mentioned in the initial investigations in Appendix 4), it was not pursued as it would not be compatible with testing the skin layer with the scales protrude out. Therefore, the set-up shown in Figure 47 was employed instead.

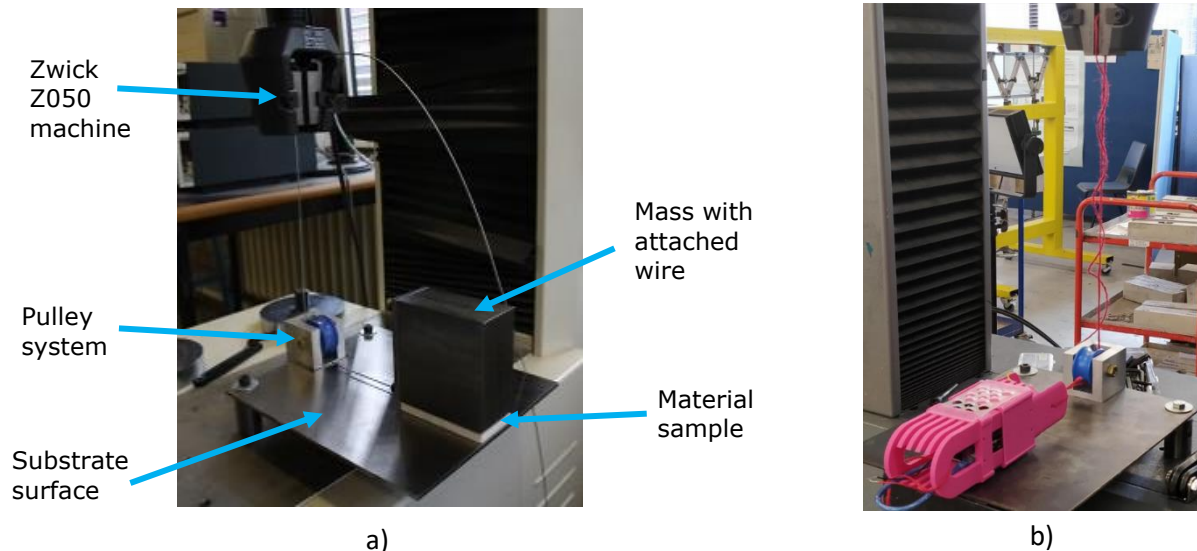


Figure 47. Frictional testing rig (a) Set-up for the flat silicone sample (b) Set-up

The flat silicone sample was attached under a 3.5 kg mass block, which was not the case with the scaled sample. It was expected that the scales would have been crushed due to not being able to sustain the block's weight. Instead, this sample was attached to a module housing prototype weighing 171.91 g. The difference in weight was not an issue, as any mass difference is accounted for when calculating the coefficients of friction using Equation 16, from which the analysis would be derived.

With both set-ups the mass had a wire attached, which linked through the pulley system up to the Zwick Z050 machine clamp. The machine was then set to pull the wire up with a force of 20 N at 8 mm/s for a distance of 20 mm. The wire and pulley system ensured the applied force acted parallel to direction of motion. This allowed the mass to move along and the resistant force between the material layer and the substrate surface between to be measured.

The flat silicone sample was tested along four substrate surfaces (steel, carpet, acetate, laminate) only in one direction, as it was assumed that both directions would generate the same results.

Scaled samples were attached to the top and bottom of a module housing prototype (Figure 47b) and set in 'extension' mode, so the scales were permanently popped out. The skin layer was tested across steel and carpet, with both forwards and backwards motion recorded.

5.2. Prototype 2 – Serpentine and Sidewinding motion

5.2.1. Motor Testing

The motor testing came in two stages: 5-motor prototype with standard motor brackets, and a 5-motor prototype with 3D printed SLS brackets (the module housings). Both prototypes are shown in Figures 48 and 49 below.



Figure 48. 5-motor prototype with standard motor brackets.



Figure 49. 5-motor prototype with custom SLS brackets.

The aim of this testing stage was to observe the different motions performed by the robotic snake – in both the standard and custom SLS brackets – and identify associated issues. The hierarchy of potential issues includes ease of assembly, mechanical movement, motion stability and bracket durability. The test is an accumulation of techniques mentioned in section 4.2.2. including the use of a ROS environment to interface the 5-motor prototype and the matrix function (Figure 30) used to cycle through the 6 different motions: forward serpentine (Appendix 2 – Figure 76 and Figure 77), backwards serpentine (Figure 28 and Figure 29), left sidewinding (Figure 31 and Figure 32), right sidewinding (Appendix 2 – Figure 78 and Figure 79), clockwise rotation (Appendix 2 – Figure 80 and Figure 81), and anti-clockwise rotation (Appendix 2 – Figure 82 and Figure 83).

```
if __name__ == "__main__":
    # start main
    start()
    home()
    safety_check(pos_1, pos_2)

    time.sleep(2)

    # Initial Forward Movement
    motion(FiveMotor_ForwardInit)

    # Forward Movement
    # >Infinite loop
    # >Can be interrupted (w/ KeyboardInterrupt = Ctrl + C)
    try:
        while True:
            motion(FiveMotor_ForwardMov)
    except Skip:
        pass
```

Figure 50. Excerpt from main function for motor testing.

Figure 50 shows a section of the main program. It begins by calling the “start” function (Figure 51).

```
def start():
    global snake_head
    global snake_body1
    global snake_body2
    global snake_body3
    global snake_body4
    snake_head = rospy.Publisher('/J1_controller/command', Float64, queue_size=1)
    snake_body1 = rospy.Publisher('/J2_controller/command', Float64, queue_size=1)
    snake_body2 = rospy.Publisher('/J3_controller/command', Float64, queue_size=1)
    snake_body3 = rospy.Publisher('/J4_controller/command', Float64, queue_size=1)
    snake_body4 = rospy.Publisher('/J5_controller/command', Float64, queue_size=1)
    rospy.init_node('Publisher')
```

Figure 51. “start” function.

The “start” function initialises variables that point to the relevant ROS topic for each motor and creates a publisher node to enable the script to publish messages to these topics.

```
def home():
    snake_head.publish(0)
    snake_body1.publish(0)
    snake_body2.publish(0)
    snake_body3.publish(0)
    snake_body4.publish(0)
```

Figure 52. “home” function.

The “home” function is called next, this ensures that the snake is in a standard “resting” position, setting the position of all motors to 0 radians (Figure 52).

```
def safety_check(a,b):
    safe = True
    if a > 1.6 or a < -1.6:
        print("Position 1 must be between 1.6 [rad] and -1.6 [rad]")
        safe = False
    if b > 1.6 or b < -1.6:
        print("Position 2 must be between 1.6 [rad] and -1.6 [rad]")
        safe = False
    if safe == False:
        shutdown()
```

Figure 53. “safety_check” function

The “safety_check” function is then called (Figure 53). This function takes two arguments: the two specified positions that the motors will alternate between. To prevent any mechanical damage to the motors or the motor brackets, this function ensures that the two positions are not greater than +1.6 radians or less than -1.6 radians.

```
def shutdown():
    print("Shutting down")
    home()
    sys.exit(0)
```

Figure 54. “shutdown” function.

If any of the positions flag a false safety check, the “shutdown” function is then called (Figure 54) to allow the program to exit safely and appropriately.

The next part of the main program initiates the motion function for the initial part of the specified motion. This sets the motors up into a position that allows the main body of the specified motion to be run on a loop. The loop is set within a “try..except” block, which allows the script to test the block of code but handle any raised errors – in this case the error to be handled is “Skip” (Figure 55).

The “motion” function (Figure 30) contains the following syntax: `signal.signal(signal.SIGINT, signal_handler)`. This captures a signal of type SIGINT (essentially KeyboardInterrupt – pressing CTRL+C), and once captured it then calls the function “signal_handler” (Figure 55).

```

# Skip Definition
class Skip(Exception): pass

# Signal Handler
# --- --- ---
# >Capture KeyboardInterrupt
# >Set Snake to home position
# >Quit script appropriately
def signal_handler(sig, frame):
    print("Interrupted")
    home()
    raise Skip

```

Figure 55. Skip declaration and “signal_handler” function.

For the purpose of initial testing, it is ideal to have the ability to cycle through the motions as desired for ease of testing. The “signal_handler” function sets the motors to their home position and raises the exception: Skip. Skip is therefore declared as an exception and calls a pass statement allowing the robot to move onto the next motion.

All project manufacturing and testing was halted on 13/03/2020 due to the COVID-19 pandemic. Due to this, the initial motor testing was performed without the addition of the head and tail modules. There was also no time for adjustments or organising additional tests. Section 6.2.1.2. details the next plan of action following the issues raised in the initial tests performed.

5.2.2. Sensor Component Testing

5.2.2.1. Establishing the methodology for sensor attachment

The sensor component testing was used to determine the method sensor attachment to the housing. These preliminary tests were conducted using a 3D printed piece to replicate the adhesion to the module surface of the robot.

The first method entailed gluing the sensor directly onto the housing, then securing the skin directly onto the module surface to ensure secure attachment to the module. This is depicted in Figure 56 below.



Figure 56. Attachment of sensor in between module and skin layer.

The layer of glue used to create adhesion of the skin to the housing resulted in a pressure acting on the sensor despite the lack of an external additional force. This meant the sensor would not be able to detect any form of interaction by an external object such as the ground during motion, or an obstacle during collision detection. As a result, this method was determined to be inadequate. To mitigate this problem, the second method was used. This involved leaving a small gap of a few millimetres between the sensor and the skin material alleviating the sensor of the pressure produced by the adhesion. The sensor was glued directly onto the module surface replica and the skin material was attached either side of the sensor as shown in Figure 57 below.



Figure 57. Attachment of sensor on module without being in contact with skin layer.

When tested with the Arduino code detailed in Section 4.3.2, the Arduino returned a zero voltage when no force was applied to the sensor. When applying a pressure onto the skin layer, the Arduino returned integer values of 1-5 V to the serial monitor and indicated a ‘collision detected’. It was deduced that this method of attachment allows for accurate measurement of external forces applied to the sensor through the skin layer, thus was the chosen method of sensor adhesion.

5.2.2.1. Establishing the methodology for collision detection

The sensor hardware was tested to ensure their range of detection force was suitable for collision detection. This was conducted by just applying a force to the sensor through the fingertip to make certain the values changed in response. The second stage of testing entailed designing and manufacturing a housing for the snake head to enable the sensors to make contact with the obstacle if a head on collision was to occur to guarantee detection; the design is discussed in Section 7. Due to the project abruptly ending as a result of Covid-19, the snake head was not manufactured thus ceasing further testing. With more time, the plan was to integrate the sensors into the snake head and ensure the design allowed for accurate obstacle detection.

5.2.3. Sensor integration Testing

Utilising the code shown in Figure 58, the voltage of analog pin 0, where the sensor is connected was read. This signal was then interpreted and converted into a value from 1 – 1023 that dictated how much pressure was being applied to the sensor. These values were subsequently written in the form of a 16-bit integer equivalent under a topic in ROS to the ‘Arduino’ ROS node 10 times per second.

```
#include <ros.h>
#include <std_msgs/Int16.h>

//Set up the ros node and publisher
ros::NodeHandle nh;
std_msgs::Int16 int_msg; //defining message as number converted to 16-bit signed integer equivalent
ros::Publisher arduino("arduino",&int_msg); //defining publisher node "arduino"
int analogPin = 0; //connect FSR to analog pin 0

void setup()
{
  //initialising the node
  nh.initNode();
  nh.advertise(arduino);

  Serial.begin(9600);
}

long publisher_timer;

void loop()
{
  if (millis() > publisher_timer) {
    int_msg.data = analogRead(analogPin);
    arduino.publish(&int_msg);
    publisher_timer = millis() + 100; //publish ten times a second
    nh.spinOnce();
  }
}
```

Figure 58. Arduino ROS node code.

The second method tested was to upload the code within the catkin workspace. The code remains almost the same as before however it needs to include the “Arduino.h” header and is saved within the projects source files (Figure 59). This code also published the sensor values 10 times per second.

```

cmake_minimum_required(VERSION 2.8.3)

include_directories(${ROS_LIB_DIR})

# Remove this if using an Arduino without native USB (eg, other than Leonardo)
add_definitions(-DUSB_CON)

generate_arduino_firmware(hello
  SRCS arduino.cpp ${ROS_LIB_DIR}/time.cpp
  BOARD Arduino/Genuino 101
  PORT /dev/ttyACM0
)

```

Figure 59. – CMakeLists.txt(firmware).

Based on collision testing, a threshold value can be obtained to decide whether the robot has made a collision using a listener node. This node will then inform the motor node to change adapt its movement in order to move around the obstacle. This process is described in more detail within the section 6.2.4.

6. Testing Results and Discussion

6.1. Prototype 1

6.1.1. Skin Testing

When attached to the housing with the actuator fully extended, it was immediately noticed that the scales were collapsing under the weight of the module. This would reduce the desired increase in friction in the direction opposed to motion as the skin surface would be flatter and the tips of the scales were not digging into the ground as much as predicted. Due to the components within the module being unevenly distributed, this flattening was more prominent on one side of the module. To overcome this, the components within the module should be positioned in such a way as to ensure the centre of mass is as close to the centre of the module as possible. If the scales still collapse under the load of the housing, a thicker layer of silicone should be considered to increase the rigidity of the scales when extended. If this thicker layer still does not achieve the desired effect, then alternative methods of reinforcing the scales should be investigated, such as support material or changing the material of the scales all together.

It was also observed that, when initially extending, some of the scales would pop out in the wrong direction, towards the housing instead of towards the ground. This could be counteracted by manually popping the scales out in the correct direction before implementing motion, but a more permanent method would be ideal. This could consist of material put underneath the scales to ensure they pop out in the correct direction.

It was not investigated how the scales would cope with wear over repeated use over rougher surfaces. If it was found that the scales would wear down quickly, this could impact how effective they would be in increasing the frictional force with the ground and a more wear resistant material could be considered.

6.1.2. Actuator Testing

6.1.2.1. Actuator motion testing

When determining the values required to produce the physical extension and retraction limits of the actuator, the generic servo motor limits were taken, which were 0° and 180°. By implementing these values into the Arduino code to command the actuator to perform its physical equivalent, then measuring the physical distance the actuator had travelled, the actuator limits were established. At a command of 180°, the actual extension was equal to 20 mm (the full extension value listed by the manufacturer) meaning that 180° was equal to full extension. At 0°, the actuator performed full retraction and therefore the set limits were written into sweep motion code.

The parameters of step increment and delay within the sweep motion code were then adjusted until the actuator produced smooth cyclic motion. Having established the optimal values for these parameters, the code needed to be adapted and the delay function removed. This is because the delay function halts the program for the stated number of milliseconds before moving onto the next line of code. As a result, it blocks the program from performing any other task in the meantime. This means in future steps when the actuator control is integrated with sensors, the delay function would cease the micro-controller from constantly reading the values from the force sensitive resistors deeming the function of the sensors unreliable.

To mitigate this issue from affecting future work, a new function called *millis()* was used. This function works by returning the number of milliseconds that has passed since the program initiated (the Arduino was connected to a power supply). By using some basic maths, this function can be used to replace the delay without interrupting the loop. The final code for the actuators is shown in Appendix 3 (Figure 84). The code works by continuously writing the actuator position. The value of the actuator position is changed every 20 milliseconds and incremented/decremented by a value of one. In order for the Arduino to know when to command this change in position, the *millis* function is used. Within the if statement, a basic calculation is performed that subtracts the previous recorded time (millis) by the current record time (sweepMillis). If this value is greater than 20 milliseconds, the actuator position is changed. The current recorded value then replaces the 'millis' variable in preparation for the next count of 20 milliseconds. All code outside of the if statement will continuously run, so other written functions that may be incorporated in future such as continuously reading data from the sensors, would not be affected.

6.1.2.2. Actuator in housing testing

Figures 60 and 61 below show the maximum retraction and extension of the actuator, and thus the housing. The attachment of the actuator was secure and caused smooth motion of the housing and extension of the housing on all sides evenly. This meant the skin was pulled evenly causing no problems.

The actuator took 3 seconds to fully extend the housing to a 20 mm distance (Figure 60) and another 3 seconds to fully retract the housing (Figure 61). It would have been desired for this duration to be faster, however the actuators were already performing at maximum speed. This would not cause any issues with performing rectilinear motion but would just make the snake move slower. The actuators could be swapped out for quicker ones in the future if desired.



Figure 60. Actuator full extension pushing open the housing.



Figure 61. Actuator full retraction closing the housing.

6.1.2.3. Actuator in housing with skin attachment testing

The same process was followed as the previous tests with the actuator performing a sweep motion continuously from full extension to full retraction. The figures below show the skin behaviour under the limits of the motion. After several cycles, the skin layer remained attached to the skin, this showed that the attachment method was secure and did not need to be altered.

Under full extension, the scales simultaneously protruded outwards as shown in Figure 62 below. However, under the weight of housing along with the internal components, the scales are flattened. Fortunately, the cuts within the skin layer still create a higher coefficient of friction with the ground in the direction opposite to the desired motion. Therefore, on module retraction, the robot is able to push forwards off the ground. At full retraction, the scales flatten allowing for ease of motion as shown in Figure 63 below.

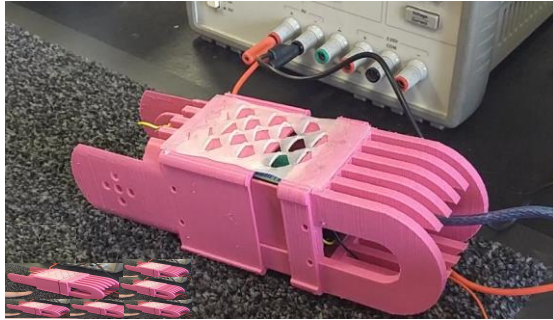


Figure 62. Housing extension with scales protruding.

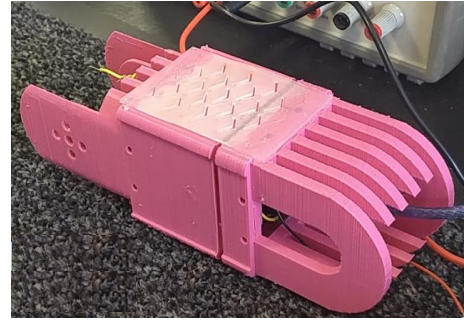


Figure 63. Housing retraction with scales flattened.

6.1.2.4. Skin mechanism issues

When multiple actuators were wired up and tested external to the housing for syncing capabilities (Figure 64), it was noted that at certain points, the individual actuators' motion was jagged. Due to the position feedback capability of actuator, each actuator was continuously trying to correct itself when it slightly missed the mark of the commanded value. This led to the actuators becoming out of sync. To mitigate the issue, the PQ12 actuators were to be replaced with more basic actuators with no position feedback. Although this would take the accuracy of actuator extension and retraction positioning away, it would not be a concern as it would have very little effect on the overall length of the actuator position.

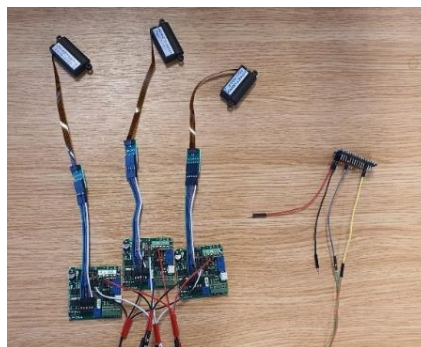


Figure 64. Multiple actuators set-up externally to the housing for testing.

By replacing the PQ12 actuators, the LAC board and cable adapters were also no longer required. By removing these components, the module could be redesigned and reduced in size. This would lead to weight reduction and potentially alleviate the issue of the scales buckling under the heavy weight of the original housing design, internal tray and the internal components.

As mentioned previously, another solution to ensure the scales do not flatten under the weight of the housing is to have a double skin layer of silicone sheets to increase the stiffness. However, there was concern that the actuator may not be able to withstand the required tension force to stretch the thicker silicone layer and so additional testing would be required.

Despite the issues listed above, the actuator-driven prototype of a single module developed allows for robot movement when performing rectilinear motion. The issues listed above are to augment the motions efficiency in moving a greater distance each extension-retraction cycle.

6.1.3 Frictional testing of Skin

6.1.3.1. Results

Figure 65 illustrates the performance of the flat silicone layers across four substrate materials.

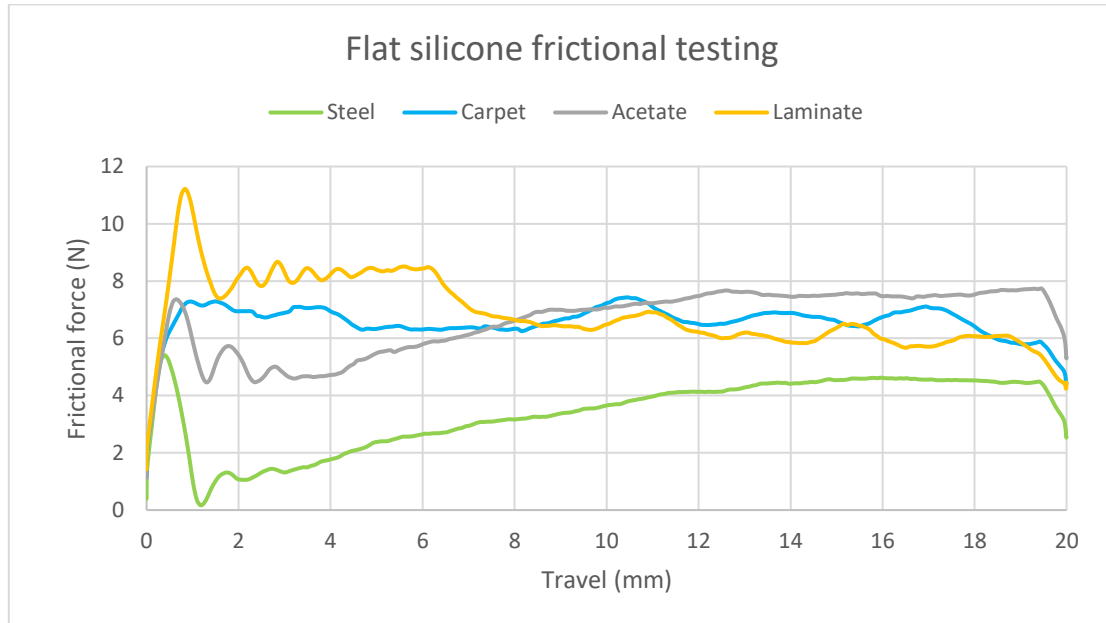


Figure 65. Frictional testing results for flat silicone sample attached to steel block mass.

For each substrate, μ_s was obtained using the force needed to initiate movement between the two surfaces. This is depicted by an initial peak force which is usually greater than any force exerted after the mass begins to move. However, in the case of the carpet sample where there was no distinct peak, the force at the beginning of the motion period (at ~ 1 mm) was used for calculation.

μ_k was determined using the average force exerted during the motion region. The results are summarised in Table 14.

Table 14. Coefficient of static and kinetic friction results for the flat silicone sample.

Substrate	Initial peak force (N)	Average force (N)	μ_s	μ_k
Steel	5.41	3.39	0.16	0.10
Carpet	7.28	6.72	0.21	0.20
Acetate	7.24	6.66	0.21	0.19
Laminate	11.22	6.86	0.33	0.20

The same method was applied for the scaled silicone results. Although the initial peak force does not equate to the maximum force exerted for three of the four scenarios, it was used for the calculation of μ_s . The subsequent force range across the rest of the travel distance was averaged to calculate μ_k . The results are displayed in Figure 66 and Table 15 below.

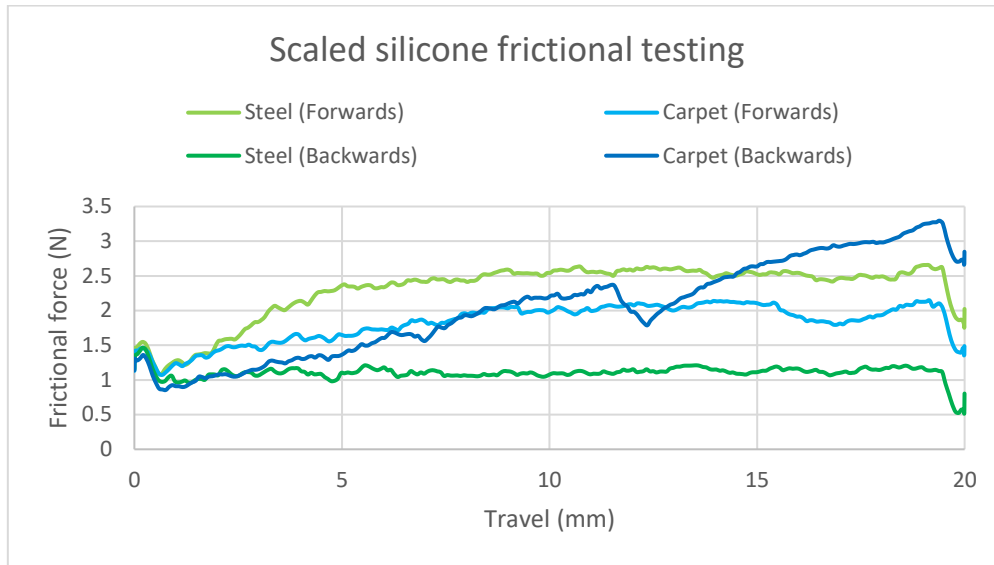


Figure 66. Frictional testing results for scaled silicone sample attached to module housing.

Table 15. Coefficient of static and kinetic friction results for the scaled silicone sample.

Substrate		Initial peak force (N)	Average force (N)	μ_s	μ_k
Steel	Forward	1.55	2.51	0.92	1.49
	Backward	1.44	1.11	0.85	0.66
Carpet	Forward	1.43	1.93	0.85	1.14
	Backward	1.36	2.32	0.81	1.38

6.1.3.2. Discussion

Flat silicone results

From Figure 65, it appears that the frictional resistance across carpet, acetate and laminate are relatively similar. This similarity in μ_k values is encouraging as it suggests that, when the scales are flat against the module housing, they will not hinder the robot's serpentine and sidewinding motion. However, it should be taken into consideration that the steel and acetate tests did not produce a stabilised force range during travel, and so repeat tests should be carried out to verify the average frictional resistance exerted on these surfaces.

Scaled silicone results

Evidently, three of the four tested cases did not display a consistent magnitude of frictional force during motion (Figure 66). Their μ_k values are all above 1, indicating that, on average, the measured frictional force was greater than the normal force. This may be due to the machine being insensitive to the module housing's weight.

In terms of the direction of the scales, for the steel surface, it is apparent that there is more resistance to the scales in the forwards direction than backwards (scale orientation depicted in Figure 67 below). This may be because on a seemingly flat surface, the scales have nothing to interlock with. They become compliant, increasing contact area with the steel substrate and therefore adhesion-mediated friction becomes the dominant interaction (Tramsen et al. 2018). In the backwards direction, only the points of the scales contact the surface so there is less frictional resistance due to adhesion in this direction.

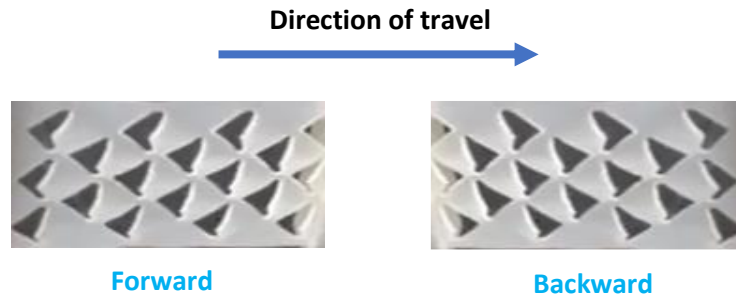


Figure 67. Scale orientation in the 'forwards' and 'backwards' direction, as referred to in the report.

For the carpet substrate, the opposite relationship is demonstrated, with μ_k in the backwards direction being larger than μ_k in the forwards direction. The scales in the forwards direction grip more effectively with the carpet, with the stiff points propelling the module housing ahead. In the backwards direction, the scale orientation and carpet roughness interlock opposingly, thus preventing smooth travel. This may explain the jagged nature of the frictional force data recorded in Figure 66.

Ultimately, it may be deduced that the scaled silicone skin will facilitate the modular robot moving rectilinearly on rough surfaces more efficiently. However, it would be prudent to repeat this experiment with more types of substrate surfaces in future to verify this conclusion.

6.2. Prototype 2

6.2.1. Housing Assembly

The second housing prototype was assembled for the final motor test (detailed in section 5.2.1), allowing for observation of its effectiveness as an assembly and during robot motion. As outlined in section 3.1.3, the housing enabled the full function of the robot without restriction, enabling the full rotation of servo motors at each bracket whilst providing the rigidity necessary for controlled motion. Although the skeleton was successful in its functionality, there was one issue that arose during the motor testing; Figure 68 shows the damage sustained by a wire where it interacted with the skeleton at one of the brackets. This can easily be prevented in future modules by removing some material from the end of the fin, preventing the rubbing from occurring.



Figure 68. Damage sustained to wire during robot testing.

Head and tail modules were designed for the robot, shown in Figure 69, with the head module allowing for the mounting of sensors onto the 'face' of the robot and the tail module facilitating the movement patterns of the robot more effectively than the standard module seen in the assembly (Figure 6). Unfortunately, due to the closure of the university they were not manufactured. These components could be prototyped and optimised to better facilitate the robot motion and ideal sensor position.

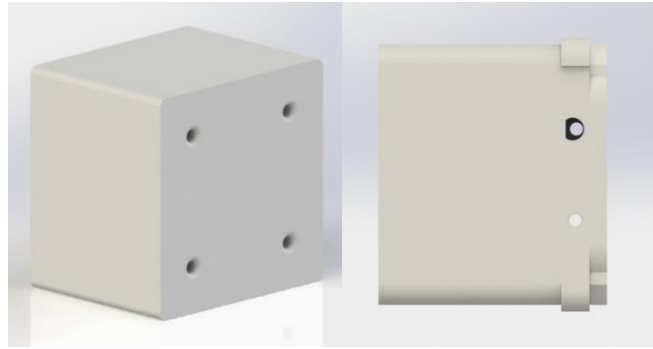


Figure 69. Renders of the head insert and tail module to be manufactured in the future.

6.2.2. Motor Testing

6.2.2.1. Initial Motor Control Issues

Several issues were encountered within the implementation and execution of motor control: position feedback error, inaccurate kinematics model, and the weight of the U2D2 and SMPS2 DYNAMIXEL Power Adapter tethering the snake.

Position Feedback

As discussed in section 4.2.2. detailing the ability to use ROS nodes to subscribe to certain topics, one potential advantage of using a ROS node as a subscriber is to receive messages from the “/J1_controller/state” topic to obtain the current position of the specified motor. This allows position feedback for each motor within the modular snake robot setup. The time elapsed between two specified motor positions can be estimated given the two positions and the joint speed, but due to friction and weight distribution, the actual value will differ from the theoretical value. Hence, the motor may be commanded to move to the second position before it has even reached the first position. Position feedback would allow smoother and more accurate position control.

Position feedback was tested with one motor, bouncing between two points, $A = 0.785$ radians and $B = -0.785$ radians. The intention was to command the motor to move to position A and subsequently position B once the position feedback gave the current motor position as equal to position A and vice versa. Within the first few rotations the aim was achieved, movements between point A and point B were smooth and accurate. As more time elapsed, the motor began to hang at each point before initiating movement. This hang time started to increase exponentially the longer the motor was in motion.

The error in position feedback was mitigated by using an alternative method to collect the current position data of the motor published to the ROS topic, compared to the initial method discussed in section 4.2.2.3. under ROS node manipulation. The following function instead waits for messages to be sent to a topic.

```
def PositionFeedback():
    msg = rospy.wait_for_message('/J1_controller/state', JointState)
    current_pos = msg.current_pos
    return current_pos
```

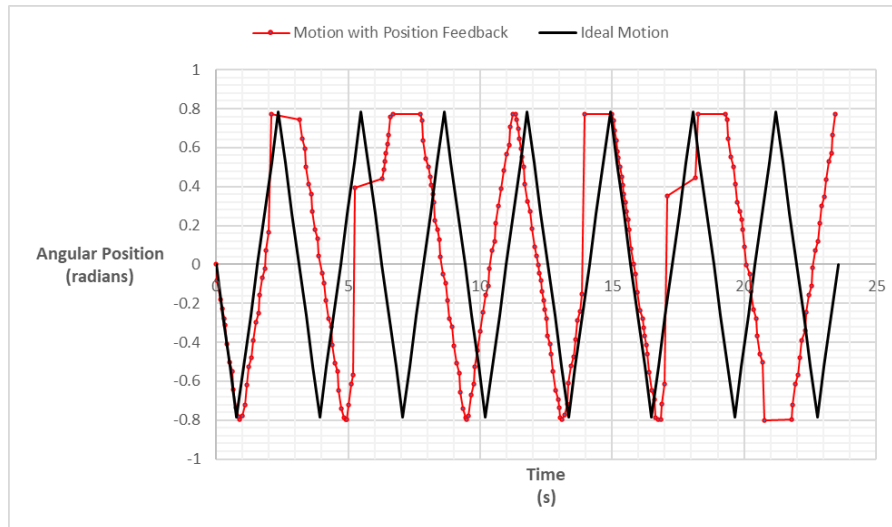


Figure 70. Position feedback motion against ideal motion.

Figure 70 shows the mitigation in position feedback error and compares the ideal motion of the motor against the motion using position feedback. There are multiple occurrences (time = 2.5, 7.5, 14, 17.5, and 21 seconds) where there is a delay between the position feedback and when the motor is scripted to move to the next position. The exact cause of this time delay is unknown and would require further testing.

Since the error in position feedback instigates after the first rotation, this issue has a high chance of occurring. The impact of this issue severely affects the motion of the snake, as the motion of a snake is heavily reliant on timed delays to allow contact with the ground at certain points to propel the snake forwards, hence the impact on the project as a whole is severe. This motion is also heavily reliant on all motors being in coordination with each other, and a delay in even one of the motors will severely affect the synchronised motion.

Position feedback is also important for the future of this project and in its application in the real world. There may be scenarios where it is physically impossible for the snake to perform certain motions or move certain motors into specific positions – i.e. a narrow-curved section of a cave. Position feedback would prevent the motors from moving to a physically impossible position that would otherwise damage the motors. The feedback can instead adjust the motion of the robot to be best suited to the environment – such as reducing the angles of rotation.

Mitigation requires further development and research – and ideally would involve an algorithm that can track each motors position and compensate the motion appropriately (which gets increasingly more complex with more motors in the system). However, an alternative solution is to keep the theoretically derived time method as detailed in section 4.2.2. with the use of matrices. Whilst this method has its limitations and inaccuracies due to the difference in value between the theoretical and real elapsed time, the error is negligible compared to the current position feedback method as shown in Figure 70 – making it the most accurate motor control method currently available – this method also allows the motion of each motor to remain synchronised.

Kinematics Model Inaccuracy

Section 3.3. covered the kinematics modelling of the snake up to 3 degrees of freedom. It was assumed that the modular snake robot could be defined as a robotic manipulator. To an extent, the modular robotic snake is a set of joints connected by links. However, a condition not considered in forward kinematics or inverse kinematics is the interaction that the end-effector – or any of the joints – has with its environment. Mathematically it was inferred that the angle of the last joint preceding the end-

effector has no effect on the position of previous joints. This assumption is inaccurate. The evaluation of joint positions does not recognise the scenarios where certain joints contact the ground. As the modular robotic snake relies heavily on joint placement to propel forwards, kinematics is not a suitable method to model the motion of the modular robotic snake.

Mitigation requires further research and understanding. Mu et al. (2017) did research into snake-like motions for complex environments, however their kinematics was not conclusive. The use of kinematics is in its ability of motion planning and to create motion profiles for different movements – most notably for robotic arms. With current understanding, the method of inducing a triangle-wave is the appropriate alternative for motion profiling. Due to time limitations, the alternative to kinematics – in terms of motion planning – is achieved using force sensors (as discussed in section 4.3.) which will allow the robotic snake to detect environmental collisions. Motion planning can be taken further in future. With the use of an onboard camera, computer vision can be utilised for mapping unknown environments.

Tethered Snake Robot

With reference to Figure 20, the initial hardware workflow map for motor control, the snake robot is tethered at both the head and tail of snake via the microprocessor (in this case a laptop with Xenial Ubuntu 16.04) and the power supply. In motion testing, without human intervention, forwards and backwards serpentine motions had restricted movements. One step to mitigate the issue was to reduce the complexity of the wiring and therefore reduce the number of tethered points. Figure 21, the new hardware workflow map for motor control, is an attempt to reduce the complexity of the wiring by allowing the power supply and the U2D2 (USB to DYNAMIXEL) device to connect to the same hub via the “6 Port AX/MX Power Hub”. A more suitable mitigation, in the future of this project, is to replace the power supply with a portable 12 V 5 A battery pack and the laptop with a ROS compatible microprocessor – which would reduce the number of tethered points to zero and allow free movement.

6.2.2.2. Plan of Action due to Academic Disruption

Further testing of the modular robot snake – in terms of motor control – was planned; however, all project manufacturing and testing was halted on 13/03/2020 due to the COVID-19 pandemic. Despite this, details on the testing process, that would have been carried out otherwise, is outlined below, including a plan on how to integrate them into the current project.

Further Motion Testing with Code Optimisation

The 6 different motions: forward serpentine, backwards serpentine, left sidewinding, right sidewinding, clockwise rotation, and anti-clockwise rotation were tested. These movements were implemented using the methods and matrices detailed in section 4.2.2.

Initially a 5-motor snake robot was used to test the 6 motions using the standard manufacturer motor brackets. The bracket assembly, motion stability, and mechanical movement of the 6 motions demonstrated no issues with the initial 5-motor prototype, pictured in Figure 48, thus no programming adjustments were needed. Please refer to demonstration video file #3 for reference. A 5-motor prototype was then assembled with the custom SLS brackets (see section 3.1.) and tested with the 6 motions outlined above (demonstration video files #4 - #12).

The 6 motions demonstrated with the 5-motor prototype with custom SLS brackets, pictured in Figure 49, had a few issues. Primarily due to the fact that not all bracket designs were printed (COVID-19 pandemic), some key components were not produced – such as the head and tail of the snake. The movement of the snake robot relied heavily on its interaction with the ground and the surrounding environment. Without the head and tail components, some movements were less pronounced. Backwards serpentine was significantly slower than forwards serpentine, as this movement relied on

the tail contacting the ground and pushing off from it. Left and right sidewinding movements had some stability issues, the increased weight and length of the SLS motor brackets (in comparison to the standard brackets) meant that more weight was shifted onto the mid-section of the snake, causing the robot snake to turn onto its side. This can be seen with reference to video file #12 - before the end of the first rotation (within the first 2 seconds) the snake robot can be seen falling onto its side. It is still able to achieve the desired direction of movement, but with less efficiency.

Initial testing, performed upon the assembly of the prototype, showed that target joint angles greater than 0.785 radians (45°) produced smoother forwards and backwards serpentine motions – an angle of 45 degrees was originally chosen as it was observed to cause certain sections to not be in contact with the ground to assist the different motions. Due to COVID-19 this could not be formally tested with supporting data, but angles of 55° and 60° showed smoother movements – these greater angles allowed the “tail” end of the snake to contact the ground and push off. It was also noted that angles lower than 0.785 radians gave the sidewinding movements more stability – this had the effect of evening out the weight distribution. Further testing would be required to identify the exact causes within the movement issues identified, and to identify optimum angles and joint speeds for the different motions by assessing the speed and stability.

As mentioned in section 4.2.2., the matrices and matrix iterator function (Figure 30) produced allowed the manipulation and testing of the joint speed and the target joint angle for all motors. This meant that testing different angle and speed combinations only required the user to set the angle and speeds in the scripts themselves. However, the proposed testing briefly outlined above would require different joint speeds and different joint angles for different motors in the same setup. This is more apparent in the fact that forwards and backwards serpentine motions rely heavily on motors 1, 3, and 5, in that they must all follow the same speeds and target positions off-set by a time delay. The sidewinding and rotation movements rely more heavily on motors 2 and 4. It was suggested above that greater angles provide smoother serpentine motion, but that lower angles provide smoother sidewinding motion. The current matrix iterator setup will not allow for different target angle positions for different motors, instead the matrix is setup so that all motors follow the same pattern with time delays.

For future testing, the code must be further optimised to accommodate these additional changes, along with the ability to assign unique target joint angle positions to each motor. Rather than having one singular matrix to encompass an entire movement, this method will require an (N x 3) matrix to represent the motors (1,3,5) responsible for the serpentine motion and an (N x 2) matrix to represent the motors (2 and 4) responsible for the sidewinding motion. Where N is an arbitrary number of rows set by the user – determined by the precision of time delay needed. Take a 5-motor setup for the “left sidewinding” movement as an example:

```
leftSW_serpentine_matrix = np.array([ [ serp_pos1, 0, serp_pos1 ],
[ 0, serp_pos1, 0 ],
[ serp_pos2, 0, serp_pos2 ],
[ 0, serp_pos2, 0 ],
])

leftSW_sidewinding_matrix = np.array([ [ side_pos1, 0, ],
[ 0, side_pos1, 1 ],
[ side_pos2, 0, 1 ],
[ 0, side_pos2, 1 ],
])
```

Figure 71. Suggested serpentine and sidewinding matrix for the “left sidewinding” movement.

Note: “serp_pos1” and “serp_pos2” are two target joint angle positions for the motors responsible for serpentine motion, “side_pos1” and “side_pos2” are two target joint angle positions for the motors responsible for sidewinding motion and “0” represents a pass condition. Due to the change in the matrix formation, two matrix iterator functions will now need to be created. Note that the time delay

for each function is different and is defined before the main body of the script so that it is declared as a global variable.

```
def serpentine_motion(mov_array):
    columns = np.size(mov_array,1)
    rows = np.size(mov_array,0)

    # Rows
    for i in range(0,rows,1):
        # Columns
        for j in range(0,columns,1):
            # Pass Filter
            if mov_array[i][j] == 0:
                continue
            # Publish movements to motor
            else:
                # Motor 1: Snake Head
                if j == 0:
                    snake_head.publish(mov_array[i][j])
                # Motor 3: Snake Body 2
                if j == 1:
                    snake_body2.publish(mov_array[i][j])
                # Motor 5: Snake Body 4
                if j == 2:
                    snake_body4.publish(mov_array[i][j])
            time.sleep(serp_time_delay)
```

Figure 72. Optimised Serpentine Matrix Iterator Function.

```
def sidewinding_motion(mov_array):
    columns = np.size(mov_array,1)
    rows = np.size(mov_array,0)

    # Rows
    for i in range(0,rows,1):
        # Columns
        for j in range(0,columns,1):
            # Pass Filter
            if mov_array[i][j] == 0:
                continue
            # Publish movements to motor
            else:
                # Motor 2: Snake Body 1
                if j == 0:
                    snake_body1.publish(mov_array[i][j])
                # Motor 4: Snake Body 3
                if j == 1:
                    snake_body3.publish(mov_array[i][j])
            time.sleep(side_time_delay)
```

Figure 73. Optimised Sidewinding Matrix Iterator Function.

By splitting the serpentine and sidewinding motions into two separate functions, testing with different target joint angles and joint speed for each motor is possible. This function is not in its final form, since it cannot be tested at this stage, so it can be further optimised depending on its usage.

In the singular matrix, which encompasses all the motors, only one function call is needed within the main program. The concern in this new change is that creating the different movements requires two function calls for all 5 motors to move simultaneously. Python has a method of addressing this issue – through multiprocessing.

Import the dependant libraries for multiprocessing: `from multiprocessing import Process`

Within the main body of the script, the two functions can be called at the “same” time with the following syntax and reference to Figure 71, Figure 72, and Figure 73:

```
Process(target=serpentine_motion, args=(leftSW_serpentine_matrix,)).start()
Process(target=sidewinding_motion, args=(leftSW_sidewinding_matrix,)).start()
```

From this method, multiple joint angles and joint speeds can be tested for the serpentine and sidewinding motions simultaneously. The use of multiprocessing does have some drawbacks. It requires the “ROS Master” device to have multiple CPUs (depending on the number of processes). If there is the intention of replacing the “ROS Master” device with something more portable – such as a Raspberry Pi or Arduino – then the hardware specifications must be considered suitable for multiprocessing. And whilst multiprocessing allows two functions to run at the “same” time, there will technically be a process delay. This would require further testing and mitigation. It would also be ideal to test the different movements on different surfaces and slopes at this stage.

6.2.3. Sensor Component Testing

6.2.3.1. Force detection

When testing was conducted, it was noted that the sensors easily attached to the 3D printed housing using a thin layer of super glue. The method of attachment did not affect the readings obtained from the sensors; thus, no calibration was required. Additionally, an effective method of sensor integration to the housing under the skin layer was established ready to be implemented into the motor-powered prototype.

Although the sensors were able to effectively collect values when force was applied during experimental testing, one disadvantage of using force sensitive resistors as sensors for collision detection within this project is that the collisions can only be detected when the force applied is perpendicular to the conductive film (the active area of the sensor). Therefore, to increase the accuracy of the sensor for obstacle impact, a cotton bud or rubber material can be placed over the active area to ensure the applied force detected on the FSR can be distributed across the entire active area. Using the serial monitor of the Arduino to display voltage values obtained from the associated force produce by the collision, calibration can be used to ensure ‘zeroing’ with the added material layering the active area, and to adapt what is deemed to be a collision.

6.2.3.2. Force Measurement

The force versus resistance characteristic of an FSR can be very non-linear. However, there is an almost a linear relationship between the force applied on the FSR with the conductance ($1/\text{resistance}$).

Liljebäck et al. (2013), conducted an experiment to utilise this relationship to accurately calculate the force applied to the sensors by placing the FSR on a digital scale. The measurement results were shown as the dashed red line on the graph in Figure 74.

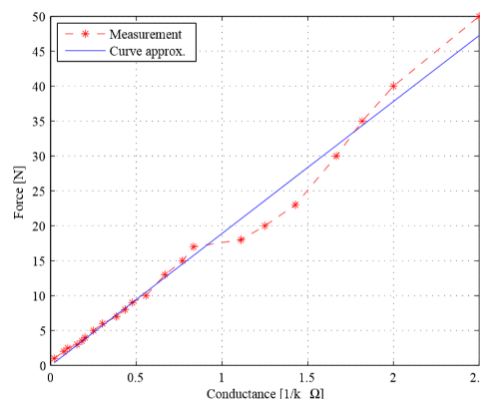


Figure 74. The measured conductance ($1/R$) of the FSR as a function of applied force.

The function of the digital scale is to weigh the force applied on the FSR and at simultaneously measure the electrical resistance through the FSR. The solid line on the graph represents the linear curve approximation to the two variables. It estimates the correlation between the force applied to an FSR

as a function of its conductance, G_{FSR} , and resistance, R_{FSR} . Based on the measurements obtained during experimentation, the expression for this linear curve was then estimated as:

$$F_{FSR} = 18.9 \times G_{FSR} = \frac{18.9}{R_{FSR}} \quad (17)$$

The value for the FSR resistance (R_{FSR}) can be obtained using the equation below, where V_{ADC} is the FSR measured voltage from the Arduino:

$$V_{ADC} = \frac{R_{FSR}}{R_{FSR} + 8.2} \cdot 4.5 \quad (18)$$

By rearranging the equation above for R_{FSR} , the following equation is reached:

$$R_{FSR} = \frac{4.5 - V_{ADC}}{8.2V_{ADC}} \quad (19)$$

Subbing Equation 19 into Equation 17, the magnitude of force applied on the FSR can be calculated using the following equation:

$$F_{FSR} = \frac{4.5 - V_{ADC}}{8.2V_{ADC}} \cdot 18.9 \quad (20)$$

This formula can be used to compare the experimental results with the theoretical result to prove the accuracy of the FSR. By experimental results of force measurement, it means pressing the active area of the FSR on digital scale with the same amount of force applied on the FSR during testing with Arduino.

The aim of the sensors located within the head of the snake are to demonstrate collision-detection on the front head. The current design therefore assumes that all contact forces are applied directly onto the front face of the head, however this may not be the case. A collision may occur at the sides of the front module, these can be detected by implementing additional sensors on the sides. With the FSRs' low cost and ease of use, utilising extra sensors is simple and cheap. However, due to Covid-19 outbreak, the method of testing the force measurement of FSR with the snake modules could not be implemented.

6.2.4. Sensor integration Testing

As mentioned in section 5.2.3, the Arduino was connected to the ROS network successfully using the Catkin workspace. This enabled the sensor values to be continuously read and published to the 'Arduino' node ten times a second. The next step was to connect additional sensors to the Arduino and ensure these values were readily obtained.

Utilising the values obtained by the current sensor connected to the Arduino, the threshold value classed as a 'collision' was to be determined. As a result of Covid-19, the project was ended early, so this step was not accomplished. The collision testing would have entailed simulating a collision and determining the threshold value. This value would then have assigned a range of acceptance, this scale would have also been obtained through further experimental testing. On establishing these values, a code in ROS was to be written to utilise this threshold and send a signal to the 'motor' node in ROS on detection of a collision.

When a signal is sent to the motor node in ROS, the motors will change its activity from continuous forward movement of the motors of serpentine motion to the following procedure:

1. Stop all motor motion
2. Reset the motors to 'home' position
3. Perform reverse serpentine motion for several cycles

4. Either move to the right or left using sidewinding for several cycles
5. Perform forward serpentine motion and continue along its path

This sequence allows for the robot to move away from the obstacle then reroute itself around it. To change from one type of motion to another, the motors are first stopped then reset to 'home' position because this is the reference for all motion types in terms of motor angles.

7. Future work

One full prototype was built, and the other was semi-built due to the project ending abruptly and early. If the project was not cut early, the assembly of the second prototype for rectilinear motion would have been completed and further analysing the motion of the first prototype performing serpentine and sidewinding motion would have been conducted. Tasks suggested for the future focuses on design optimisation, and are detailed below:

1) Actuator Number

The maximum number of actuators that can be connected to the Arduino nano requires testing. This will determine the limit of modules that can be linked together to build the prototype. During testing, the simultaneous motion of the actuators will also need to be tested to ensure the delays do not affect module extension and retraction movements. Additionally, the current drawn by the actuators needs to be monitored to ensure the current limit of the Arduino is not exceeded as this would lead to hardware failure.

2) Housing redesign

The housing requires a redesign for several reasons, the most predominant reason is if the current actuators being used (PQ12 Micro Linear Actuators) are replaced with a more basic actuator model. This would remove the feature of each individual actuator continuously correcting themselves via position feedback, thus permitting consistent syncing of all actuators. As a result, the need for a Linear Actuator Control Board will be removed. As this component dictates the minimum size of the housing due to its large dimensions, the housing size can dramatically be reduced on its removal. By scaling the housing down, the weight of the module will be reduced. This will optimise all motions, in particular, rectilinear motion as a reduction in weight will flatten the scales less making the scales more effective in producing a higher coefficient of friction, and such, propel the robot further forward. Additionally, further mass reduction can be accomplished by integrating lattice patterns into the main body structure. Finite element analysis of such designs can be done to optimise the mass of the skeleton as far as possible whilst retaining the necessary strength levels of components within the bracket.

3) Wireless control of the motors

To ensure the robot is not tethered to a laptop during movement, wireless control of the motors must be implemented. This would ensure the route of the prototype is not limited by the cabling system. To remove the wiring connection to the laptop, the laptop can be replaced with a Raspberry pi. This type of micro-controller will allow for continued access to the ROS network for inter-communication between other components within the closed-loop control.

4) Skin optimisation

As an initial prototype, the skin layer was successfully designed to adhere to the modular robot housing and to intermittently extend and contract without failure of the protruding scales. Of course, there are many avenues of improvement that can be explored to optimise its functionality further. As mentioned from the actuator testing feedback, the stiffness of the scales needs to be increased if the skin layer is to sustain the weight of the module housing with all its components inside. This can be solved by either manufacturing the skin with a thicker layer of silicone, adhering two layers of silicone skin together or thorough investigating ways of partially stiffening the silicone skin at critical areas, such as at the points of the scales.

5) Embedding LiDAR sensors into the skin layer

By utilising LiDAR sensors, the robot becomes aware of its surroundings and any obstacles in its path within a 0.3 – 12 m range. This allows for the robot to reroute itself around detected obstacles in real-time as an alternative to awaiting a collision to detect an obstacle ahead. Not only does this make the robot more efficient in hazardous terrain, but it reduces the probability of the robot incurring any potential damages due to collision.

6) Testing of varying prototype lengths

As mentioned in the introduction, the ability to change the length of the robot could be useful when adapting in response to the environment. This would lead to a smart robot whose capabilities are expanded due to its flexibility for varying scenarios. Therefore, future work would entail adapting the lengths of the prototypes and assessing their movement capabilities and noting the differences in movement efficiency in comparison to robot length. For the prototype performing rectilinear motion, the movement has been tested for an individual module and it works well. Therefore, it would be intriguing to see its capabilities with more modules. This was unable to be completed due to the project ending early but can be easily completed in future stages.

7) Integrating the two prototypes into one

The final task for future work is integrating the two prototypes built into one functioning robot. This would create a robot that can perform all three types of motion enabling it to increase its capabilities of moving over hazardous terrains. The skin layer may also aid the serpentine and sidewinding motion by adding friction to the positive direction of moving, thus acting as a surface for the motors to push off during motion. When combining the prototypes, the actuator control must be integrated into the closed-loop feedback control of the sensors and motors. This could be accomplished by using an Arduino to control all the sensors and actuators. Using Serial Peripheral Interface (SPI), the Arduino will act as the Master component and control the timing of sending commands out to the actuators and listening to the sensor readings. Additionally, the Arduino can be connected to the ROS network as demonstrated in Section 4.1.

By completing the tasks listed above, the movement of the snake will become more efficient through weight reduction and utilising the skin layer for the various motions. By integrating all three movements into one robot, the robot can adapt to its environment by selecting the optimal motion for the terrain. With the use of further sensors such as LiDAR's, the robot will be able to react to obstacles ahead in comparison to waiting for a collision to detect an obstacle in its path. This will make the robot more effective in tackling hazardous missions.

8. Conclusion

This project endeavoured to construct a snake-inspired robot based on the principles of a modular robotic system. Five objectives were initially specified at the beginning of the report (Table 16). In order to achieve them, three main areas were simultaneously explored, involving extensive research and development in both hardware and software design.

Skeleton work resulted in the development of an interlinking module housing chain with alternating degrees of freedom, thus fulfilling Objectives 1 and 2. Software work resulted in the creation of a ROS environment incorporating servo motor control and a separate system used for the actuator control, both programmed to facilitate three types of snake motion, as well as a sensor system to recognise and react to collisions suitably. This allowed for Objectives 2 and 3 to be achieved. Skin work resulted in the production of an extendible elastomer skin layer with retractable scales to exclusively facilitate the rectilinear motion of the robot. This design satisfied Objectives 4. As described in the report, initial testing was conducted in order to fulfil Objective 5. Although a model itself was not created (due to the project ending as a result of COVID-19), the analysis of the testing results demonstrated the ability

of model to be successful on build completion. Table 16 below summaries the objectives 1 – 5 and the outcome of the project efforts.

Table 16. Project Objective outcomes.

Objective	Attempted by	Completed
1) Construct a self-contained module unit that contains relevant components	Skeleton work	Yes
2) Establish a system whereby multiple modules can be physically interlinked and programmed simultaneously	Skeleton work Software work	Yes
3) Develop code that allows the module chain to move in serpentine, sidewinding and rectilinear sequences	Software work	Yes
4) Design an outer skin layer that can aid with the rectilinear motion	Skin work	Yes
5) Embed the outer skin layer with sensors that allow the robot to detect the collision of obstacles	Software work Skin work	No

From the result of these objectives, two prototypes were successfully assembled. Prototype 1 demonstrated a single actuator's ability to successfully extend and contract a module housing with a scaled skin layer fixed on, shifting the module housing forwards. Its shortcomings included lack of multiple actuator assembly, the risk of multiple actuators becoming out of sync when linked together, and the flattening of the skin scales under the housing weight. However, upon refinement, the robot's model's ability to travel rectilinearly through narrow, hazardous sections is a feature highly valuable to the firefighting industry and other sectors involving rescue missions.

Prototype 2 demonstrated the motors' ability to successfully allow a housing chain of 5 modules to perform both serpentine and sidewinding motion. Its shortcomings included the motor error in position feedback and limitations of the kinematic model in evaluating the housing chain motion. Nevertheless, the prototype produced performed six movements (forwards, backwards, left, right, clockwise rotation and anti-clockwise rotation) utilising the basis of serpentine and sidewinding motion. This allowed full control over the manoeuvrability of the robotic snake demonstrating its capability to travel around obstacles. This could be highly applicable to designing robots which explore extra-terrestrial environments to retrieve critical scientific data.

Combining the two prototypes in the future would allow for a functional robot that can detect obstacle collisions and manoeuvre itself around such impediments using its vast range of motion. Ultimately, if the shortcomings are resolved as specified in the future work section, it is expected that this single robot model combining both initial prototype capabilities would be beneficial to various industries for direct usage, such as during rescue and exploration missions, in addition to being used as an initial base for design purposes of equipment that require flexible properties.

9. References

- FRANK, R. (2017). What types of sensors are used in robots? SENSORTiPS. [online] 26 Sep. Available at: <https://www.sensortips.com/featured/types-sensors-used-robots/> [Accessed 26 Feb. 2020].
- SEEDGROVE, D.L. (2018). Seeedstudio Grove - TF Mini LiDAR Datasheet. [online] Available at: http://wiki.seeedstudio.com/Grove-TF_Mini_LiDAR/ [Accessed 26 Feb. 2020]
- Yuan, Xin, Zou, Jiakang and Sun, Lining (2019). Soft tactile sensor and curvature sensor for caterpillar-like soft robot's adaptive motion. In: . [online] RICAI 2019: 2019 International Conference on Robotics, Intelligent Control and Artificial Intelligence. Association for Computing Machinery New YorkNYUnited States, pp.690–695. Available at: <https://dl.acm.org/doi/abs/10.1145/3366194.3366318?download=true>.
- Wang, J., Qian, S., Yu, J., Zhang, Q., Yuan, Z., Sang, S., Zhou, X. and Sun, L. (2019). Flexible and Wearable PDMS-Based Triboelectric Nanogenerator for Self-Powered Tactile Sensing. *Nanomaterials*, 9(9), p.1304.
- Dickey, M. D. (2017). Stretchable and soft electronics using liquid metals. *Advanced Materials*, 29(27), 1606425
- Tao, J., Bao, R., Wang, X., Peng, Y., Li, J., Fu, S., Pan, C. and Wang, Z.L. (2018). Self-Powered Tactile Sensor Array Systems Based on the Triboelectric Effect. *Advanced Functional Materials*, 29(41), p.1806379.
- www.robotshop.com. (n.d.). Interlink Electronics 0.2" Circular FSR (Short). [online] Available at: <https://www.robotshop.com/uk/interlink-electronics-circular-fsr-short-34-00004.html> [Accessed 5 Mar. 2020].
- CAPLINQ Blog. (2016). How do force sensitive resistor (FSR sensor) work? [online] Available at: https://www.caplinq.com/blog/force-sensitive-resistor-fsr-sensor_1638/ [Accessed 5 Mar. 2020].
- Liu, B. et al. 2019. Kirigami Skin Improves Soft Earthworm Robot Anchoring and Locomotion Under Cohesive Soil. *2019 2nd IEEE International Conference on Soft Robotics (RoboSoft)* . doi: 10.1109/robosoft.2019.8722821.
- Rafsanjani, A. et al. 2018. Kirigami skins make a simple soft actuator crawl. *Science Robotics* 3(15), p. eaar7555. doi: 10.1126/scirobotics.aar7555
- Smooth-On 2020. Smooth-Sil™ 940 Product Information. Available at: <https://www.smooth-on.com/products/smooth-sil-940/> [Accessed: 5 March 2020].
- Liljebäck, P., Pettersen, K.Y., Stavdahl, O. and Gravedahl, J.T. (2012). A review on modelling, implementation, and control of snake robots. *Robotics and Autonomous Systems*, 60(1), pp.29–40.
- Moon, B. 2001. Snake locomotion. [online] Available at: <https://userweb.ucs.louisiana.edu/~brm2286/locomotn.htm> [Accessed: 27 November 2019].

Encyclopædia Britannica Inc. 2012. Snake Locomotion. [online] Available at: <https://kids.britannica.com/students/assembly/view/171904> [Accessed: 27 November 2019].

Spong, Mark W.; Vidyasagar, M. (1989). *“Robot Dynamics and Control”*. New York: John Wiley & Sons.

ROBOTIS. (2020). *“DYNAMIXEL Wizard 2.0”*. [online] Available at: http://emanual.robotis.com/docs/en/software/dynamixel/dynamixel_wizard2/ [Accessed: 1 December 2019].

Mu, Z., Wang, H., Xu, W., Liu, T., & Wang, H. (2017). *“Two types of snake-like robots for complex environment exploration: Design, development, and experiment”*. [online] Available at: <https://doi.org/10.1177/1687814017721854> [Accessed: 3 February 2020]

Tramsen, H. et al. 2018. Inversion of friction anisotropy in a bio-inspired asymmetrically structured surface. *Journal of The Royal Society Interface* 15(138), p. 20170629. doi: 10.1098/rsif.2017.0629

Renishaw, (2019), Lattice test structures built on Renishaw AM250 metal AM system at the university of Nottingham, as part of the Aluminium Lightweight Structures via Additive Manufacturing (ALSAM) project, available at: [https://resources.renishaw.com/details/ALSAM+project+aluminium+lattice+structures\(243282\)\(87500\)](https://resources.renishaw.com/details/ALSAM+project+aluminium+lattice+structures(243282)(87500)) [Accessed on: 3 April 2020]

LeGrand, R., 2004. Closed-loop motion control for mobile robotics, s.l.: Circuit cellar.

Carnegie Mellon University, 2017. Carnegie Mellon Snake Robot used in search for Mexico Quake Survivors. [Online] Available at: <https://www.cmu.edu/news/stories/archives/2017/september/snakebot-mexico.html> [Accessed 18 April 2020].

Choset, H., n.d. Medical Snake Robot. [Online] Available at: <https://medrobotics.ri.cmu.edu/node/128447> [Accessed 18 April 2020].

Liljebäck, P., Pettersen, K. Y., Stavdahl, O. & Gravedahl, J. T., 2013. *Snake Robots: Modelling, Mechatronics, and Control*, s.l.: Springer.

10. Appendix

10.1. Appendix 1 – Raw data from Tension Testing

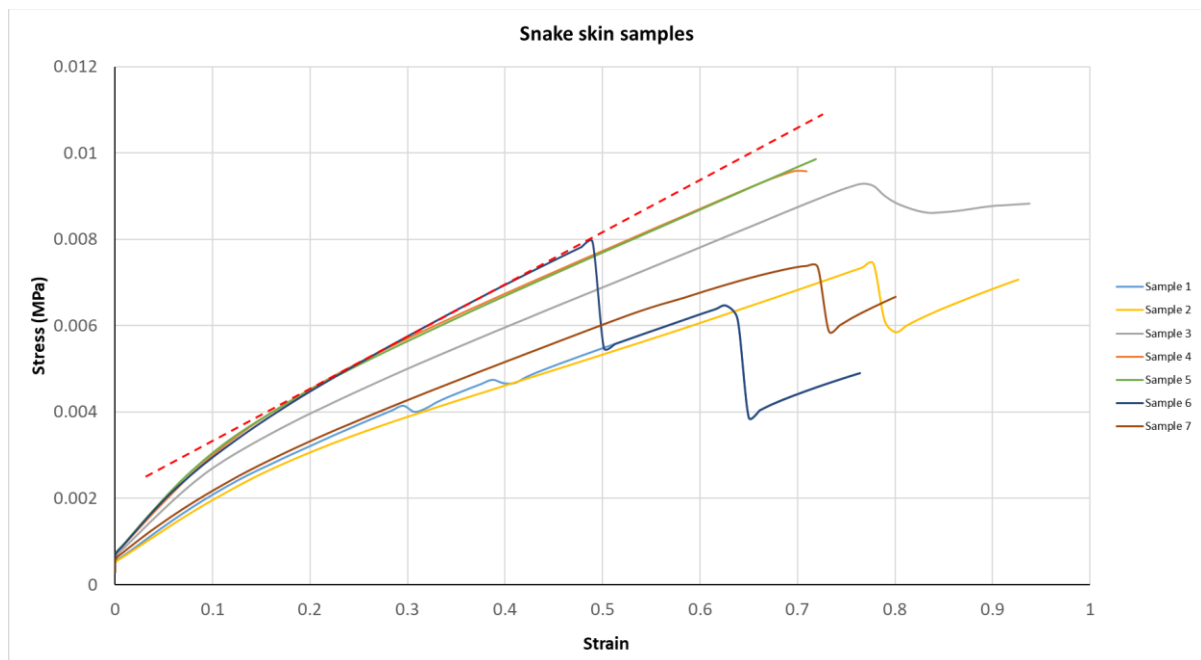


Figure 75. Snakeskin sample tension testing raw data (red dotted line is indicative of working out the Young's modulus of Sample 6)

Young's Modulus of Sample 6 is:

$$E = \frac{0.01 - 0.002}{0.65}$$
$$= 0.012 \text{ MPa}$$

10.2. Appendix 2 – Motion Profiles

```
# 5 MOTORS
# Initial Forward Movement Matrix
# ---
FiveMotor_ForwardInit = np.array([
    [ pos_2, 0, 0, 0, pos_1 ],
    [ 0, 0, 0, 0, 0 ],
    [ 0, 0, pos_1, 0, 0 ]
])
```

Figure 76. “Forward serpentine” initial movement matrix for 5 module snake robot.

```
# Forward Movement Matrix
# ---
FiveMotor_ForwardMov = np.array([
    [ pos_1, 0, 0, 0, pos_2 ],
    [ 0, 0, 0, 0, 0 ],
    [ 0, 0, pos_2, 0, 0 ],
    [ 0, 0, 0, 0, 0 ],
    [ 0, 0, 0, 0, 0 ],
    [ 0, 0, 0, 0, 0 ],
    [ pos_2, 0, 0, 0, pos_1 ],
    [ 0, 0, 0, 0, 0 ],
    [ 0, 0, pos_1, 0, 0 ],
    [ 0, 0, 0, 0, 0 ],
    [ 0, 0, 0, 0, 0 ],
    [ 0, 0, 0, 0, 0 ],
    [ 0, 0, 0, 0, 0 ]
])
```

Figure 77. “Forward serpentine” movement matrix for 5 module snake robot.

```
# Initial Right Movement Matrix
# ---
FiveMotor_RightInit = np.array([
    [ pos_2, pos_1, 0, 0, pos_1 ],
    [ 0, 0, 0, 0, 0 ],
    [ 0, 0, pos_1, pos_2, 0 ]
])
```

Figure 78. “Right sidewinding” initial movement matrix for 5 module snake robot.

```
# Right Movement Matrix
# ---
FiveMotor_RightMov = np.array([
    [ pos_1, pos_2, 0, 0, pos_2 ],
    [ 0, 0, 0, 0, 0 ],
    [ 0, 0, pos_2, pos_1, 0 ],
    [ 0, 0, 0, 0, 0 ],
    [ 0, 0, 0, 0, 0 ],
    [ 0, 0, 0, 0, 0 ],
    [ pos_2, pos_1, 0, 0, pos_1 ],
    [ 0, 0, 0, 0, 0 ],
    [ 0, 0, pos_1, pos_2, 0 ],
    [ 0, 0, 0, 0, 0 ],
    [ 0, 0, 0, 0, 0 ],
    [ 0, 0, 0, 0, 0 ],
    [ 0, 0, 0, 0, 0 ]
])
```

Figure 79. “Right sidewinding” movement matrix for 5 module snake robot.

```
# Initial Turn Right Movement Matrix
# ---
FiveMotor_TurnRightInit = np.array([
    [ pos_2, pos_1, 0, 0, pos_1 ],
    [ 0, 0, 0, 0, 0 ],
    [ 0, 0, pos_1, pos_1, 0 ]
])
```

Figure 80. “Clockwise rotation” initial movement matrix for 5 module snake robot.

```
# Turn Right Movement Matrix
# ---
FiveMotor_TurnRightMov = np.array([
    [ pos_1,      home_pos,      0,      0,      pos_2 ],
    [ 0,          0,          0,          0,          0 ],
    [ 0,          0,          pos_2,      home_pos,      0 ],
    [ 0,          0,          0,          0,          0 ],
    [ 0,          0,          0,          0,          0 ],
    [ 0,          0,          0,          0,          0 ],
    [ pos_2,      pos_1,          0,          0,          pos_1 ],
    [ 0,          0,          0,          0,          0 ],
    [ 0,          0,          pos_1,      pos_1,          0 ],
    [ 0,          0,          0,          0,          0 ],
    [ 0,          0,          0,          0,          0 ],
    [ 0,          0,          0,          0,          0 ]
])
```

Figure 81. "Clockwise rotation" movement matrix for 5 module snake robot.

```
# Initial Turn Left Movement Matrix
# -----
FiveMotor_TurnLeftInit = np.array([
    [ pos_2,      pos_2,      0,      0,      pos_1 ],
    [ 0,          0,          0,      0,      0 ],
    [ 0,          0,          pos_1, pos_2,      0 ]
])
```

Figure 82. "Anticlockwise rotation" initial movement matrix for 5 module snake robot.

```
# Turn Right Movement Matrix
# ---
FiveMotor_TurnLeftMov = np.array([
    [ pos_1,      home_pos,      0,      0,      pos_2 ],
    [ 0,          0,          0,          0,          0 ],
    [ 0,          0,          pos_2,      home_pos,      0 ],
    [ 0,          0,          0,          0,          0 ],
    [ 0,          0,          0,          0,          0 ],
    [ 0,          0,          0,          0,          0 ],
    [ pos_2,      pos_2,          0,          0,      pos_1 ],
    [ 0,          0,          0,          0,          0 ],
    [ 0,          0,      pos_1,      pos_2,          0 ],
    [ 0,          0,          0,          0,          0 ],
    [ 0,          0,          0,          0,          0 ],
    [ 0,          0,          0,          0,          0 ]
])
```

Figure 83. "Anticlockwise rotation" movement matrix for 5 module snake robot.

10.3. Appendix 3 – Final actuator code

```
//Code for actuator - full retraction to full extension
#include <Servo.h>

unsigned long sweepMillis = 0; //storage for the sweeping timer

Servo actuator;
//int pos = 0;

void setup() {
    actuator.attach(9); // attaches the actuator to pin 9 - must be PWM pin
}

void loop() {
    sweep();
}

void sweep() {
    static int pos; //variable to store the actuator position
    actuator.write(pos); //do this all the time
    if (millis() - sweepMillis > 20) //every 20ms, change the actuator position
    {
        sweepMillis = millis(); //replace the counter
        static int sweepDirection = 1; //storage for the direction
        pos += sweepDirection; //change the pos by 1/-1
        if (pos == 180 || pos == 0) //when pos gets to 0 or 180
        {
            sweepDirection = -sweepDirection; //change the direction
        }
    }
}
```

Figure 84. Code to produce smooth cyclic motion of the actuator.

10.4. Appendix 4 – Interim Report

Abstract

The body of a snake is comprised of long limbless vertebrae leading to flexible capabilities that allow for various types of movement including but not limited to rectilinear, sidewinding and serpentine motions. The extensive list of movement makes snakes adaptable to a variety of environments and as such, having the ability to swim in water, pass through small gaps and move on uneven terrain. By imitating the motion of a snake, these specialities can be utilised in several industries including the medical industry for endoscopes, hazardous rescue missions consisting of narrow spaces and the inspection of pipelines. This report explores the design of a modular robot with the modules connected in a chain, in order to imitate the mechanism of a snake. An outer skin layer is used to mimic the function of scales to provide additional friction during rectilinear motion. The desired movement abilities were determined, and a module design was created to meet such requirements. This project uses ROS (Robot Operating Systems) via a laptop for the control commands and feedback of both rotational movements using AX-12A Dynamixel servo motors, and linear movement to aid skin extension and retraction using Actuonix PQ12 linear actuators. This closed loop system allows for accuracy and synchronisation between all modules of the robot. The communication between modules will be attained by serial interface of the hardware components including a slave Arduino placed within each module with a master Arduino to converse back to the ROS network. Future steps for investigation include motor and actuator control, skin prototype testing and robot assembly. The associated risks to completion of the project have been explored with a contingency plan to mitigate these risks.

1. Introduction

The project brief was centred around designing a modular-based robotic system inspired by snakes. A modular robot is based on the concept of individual units; each unit can be equipped with several components such as motors, actuators and sensors. These systems can then be linked so that inter-modular communication and motion cohesion can be established. Their physical connections can be rearranged in several different formations to perform a variety of motion sequences and/or meaningful tasks, such as assuming a linear formation to move through a narrow pipe or to be configured in a structure with legs to move efficiently over uneven terrain.

The most common modular robot configuration are chain-type robots that tend to imitate snakes. Extensive research has been undertaken globally in order to mechanically and electronically imitate the movement and flexibility of the snake body. In an industrial context, snake-like robots can have numerous significant applications. One such example is travelling through hazardous areas for rescue missions. Another use is in the medical field, where the development of endoscopic equipment for surgery could be enhanced by the flexion capabilities of a snake-like device. Furthermore, the variety of motion sequences displayed by a snake could facilitate many complex tasks required by space rovers exploring extra-terrestrial environments.

A snake has several types of locomotion in order to adapt to different circumstances; the focus of this project is to successfully replicate the serpentine, sidewinding and rectilinear motion. In order to replicate these motions, determining how a snake manipulates its body to carry out such motions needs to be understood. Serpentine motion, also known as lateral undulation, is the most common

snake motion. It consists of waves of lateral bending being propagated along the body from head to tail using objects on the ground to propel itself forwards (Moon, 2001). When the surface is slippery, such as sand, the most common type of motion used is sidewinding, where solely two parts of the body are in contact with the ground, the remainder is held above it (Zug, 2001). The body is propelled laterally from these anchor points creating new anchor points a fixed distance away. Sidewinding is a complex motion that can be simplified by adding a vertical wave to the lateral one of serpentine motion. Rectilinear motion is useful when the snake is travelling through tight spaces. It involves the snake moving in a straight line by contracting muscles to control the movement of the scales. The scales are alternatively lifted slightly from the ground, and then pulled downward and backward, using the friction between the scales and the ground to pull the body forwards (Moon, 2001). This occurs at several points along the body. When the scales are lifted forwards they are stretched out, creating a rougher surface in contact with the ground.

For the chain-like robot to imitate such motion, the project focuses on exploring the joint design and the connection of the modules. As such, the main objectives are:

- Construct a self-contained module unit that contains relevant components
- Establish a system whereby multiple modules can be physically interlinked and programmed simultaneously
- Develop code that allows the module chain to move in serpentine, sidewinding and rectilinear sequences
- Design an outer skin layer that can aid with the rectilinear motion

Embed the outer skin layer with sensors that allow the robot to detect the collision of obstacle Initially four modules will be connected to focus on module integration and ensure the electronic components are compatible. Then eight modules will be linked to produce the three types of snake-like motion listed in the third objective above.

In order to accomplish such objectives, it was apparent that the challenge would consist of extensive software and hardware development. In terms of hardware, tasks included creating the module housings and designing bracket links that would not hinder the robot motion. Additionally, the outer skin design would entail appropriate material selection, experimentation with the dimensions to allow for efficient movement, and a linear actuator to aid the motion of skin extension and retraction. In terms of software, creating a functional control loop that can be programmed to manipulate the motion of each module as desired was the main aim.

2. Literature Review

The skeleton of a snake body consists of a long flexible, limbless vertebra with the associated ribs. This structure facilitates a variety of movement that enable locomotion in complex environments and harsh conditions. The different types of motion produced permit the snake to pass through small gaps on the ground, swim in water, and move on uneven surface terrain. By mimicking these adaptable abilities, snake-like robots can perform inspections of pipelines, conduct rescue work in hazardous environments and carry out surveillance in harsh conditions.

The first snake-like robot was invented by Shigeo Hirose in 1972 which was the ACM (Yamada, et al. 2006). The ACM is a snake-like robot that uses a method of pushing the body by twisting and turning to move. This robot has one degree of freedom (DOF) joints, which means its movement is constrained to a sine wave form on a flat surface. To stimulate the robot movements in water, paddle blades were attached to around the side of the body, as shown in Figure 1. As a result, a connection to the water was established by creating momentum for forward propulsion. To aid movement on the ground, small wheels were attached to the paddle blades.



Figure 1. ACM-R5 prototype (from Yamada, et al. 2006)



Figure 2. Robot with worm locomotion (from Akbarzadeh and Kalani 2012)

Further research was conducted to overcome the limitations of a snake-like robot with one DOF, to allow for additional types of locomotion. Akbarzadeh and Kalani (2012) designed a robot based on worm motion, with no wheels and spindle joints, which can move like a wave vertically as shown in Figure 2. Unfortunately, the success of the research was limited because the motion speed was very slow.

In order for the robot to perform diverse types of motion that are representative of a snake, researchers focused on developing a robot with orthogonal joints. Although the robot movement increased, the robot speed of movement decreased. Therefore, researchers explored designing the joints with springs (Lin, et al. 2005). However, the lateral moment of the spring restricted the distance travelled by the snake and thus increased the energy required for forward movement.

Yang, et al. (2015) proposed a snake-like robot with different shapes, structures and modular joints. The snake robot comprised of ten modular joints with two DOF and were arranged with a 'yaw-pitch-yaw-pitch' layout. The design consisted of two servo motors per module, with a drive motor in each joint, and eight rails fixed on the body. The design is shown in Figure 3 below.

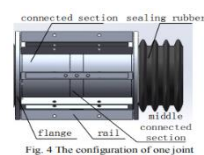


Fig. 4 The configuration of one joint

Figure 3. Amphibious snake-like robot with 10 joints and the configuration of one joint (from Yang, et al. 2015)

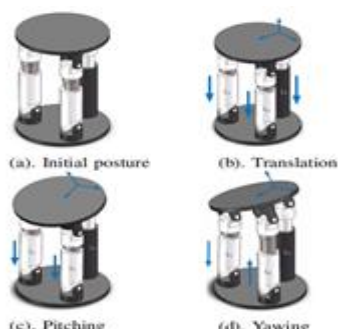


Fig. 5 Mechanism design of a snake-like robot (from Li, et al. 2016)

Another innovative snake-like robot includes three DOF that allows for translational and rotational movement of the joints, as shown in Figure 4 (Li, et al. 2016). The mechanism consists of a moving platform and three mechanical joints - one universal joint and two composite spherical joints. Two symmetrical prismatic legs are connected to the sphere joints to replicate the snake's intercostal muscles. These joints eliminate torque which is normal to the moving platform to protect the universal joint. To keep the original length of snake segments, the parallel joint is designed to prevent

simultaneous yaw and pitch movement. The rotating axis is the cross axis of the universal joint, which entails two rotating directions perpendicular to each other.

The Carnegie Mellon University created a modular snake robot which can move in a 3D manner enabling the robot to perform different types of motion: climbing, linear progression, sidewinding, rolling, cornering and pipe rolling (Muhammed, et al. 2016). Each module contains a single rotational joint with one DOF, but when linked together, it gives the robot N DOFs, relative to the length of the robot. Each module is capable of rotating 90° with respect to the previous module with the rotation axis being perpendicular with respect to the previous module. This robot is shown in Figure 5.



Figure 5. Robot design (from Muhammed, et al. 2016)

Inspired by Carnegie Mellon University's design, a snake-like robot that consists of a few modules linked together with several internal components was adopted to our design. The box shape of the module was replicated along with the concept of mechanical hinges to replicate the snake-like motions. Additionally, the inclusion of an outer skin layer was proposed within the design stage, to aid with the rectilinear motion. To allow the robot to detect obstacles present in the surrounding environment, the embedding of sensors into the skin are to be investigated.

3. Management of Tasks

To ensure the main project objectives are met, the team was split into three sub-teams to reflect the expertise required of the following areas: the skeleton, the skin and software. The allocation of sub-teams helped to exploit vital knowledge and the completion of specific technical tasks, enabling a higher standard of work to be produced. The focus of each sub-team is shown in Table 1 below.

Table 1. Displays project objectives of each sub-team

Sub-team	Sub-team project objective
Skeleton	Designing and building the module housings
Skeleton	Designing and building bracket links
Skeleton	Using kinematics to calculate the motions
Skin	Selecting skin material
Skin	Designing skin structure
Skin	Implementing linear actuators to aid in skin extension and retraction
Software	Enabling motor control and communication between them
Software	Implementing timed communication control for linear actuators
Software	Measuring and responding to obstacle detection using sensors

Each sub-team created a list of tasks to meet the sub-team project objectives which were used to create a Gantt Chart. A simplified version is shown in Appendix 1. Gantt Chart.

A leader was assigned to each sub-team to firstly allocate tasks within each area and, additionally, to tackle any issues associated with communication and task overlap between teams. Several regular meeting slots were organised to ensure these issues were being tackled, and are as follows:

- Leader meeting – to discuss work being performed by each sub-team and highlight any associated issues that need to be resolved
- Sub-team meeting – to allocate tasks to team members
- Supervisor meeting – to update the supervisor on on-going project work and receive any help where required
- Team meeting – to discuss important details that arose during the supervisor meeting

During the Leader meeting, an Action Tracker management tool was used to identify any apparent issues using a traffic light system. Red indicated a problem where no plan has been identified, orange was a task in progress of being completed and green showed that a task had been completed. By using the traffic light system, tasks falling behind schedule are identified and further resources allocated, or deadlines extended where appropriate. Additionally, sub-team leaders were able to discuss any 'red' actions to help provide a plan to tackle the task using the expertise of others.

The current work of each sub-team will be discussed in the following section with any risks of completion highlighted in a separate section afterwards.

4. Work achieved to date

4.1. Skeleton

4.1.1. Robot Design

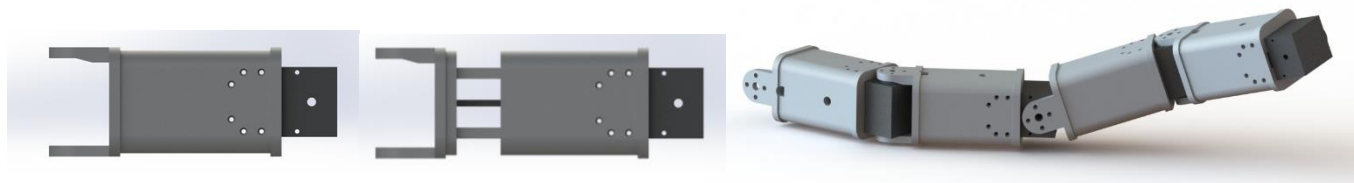
The robot skeleton itself is comprised of multiple identical modules, effectively acting as vertebrae within the skeleton. Each module must be designed to meet three primary functions: component housing, robot motion and skin integration.

Component housing: This is characterised by the design of the module interior. Simplistically, all the necessary components for module function must be contained within the housing; outlined in appendix 3. They must be mounted securely and must be easy to access for module assembly and maintenance. This should be done as space efficiently as possible to minimise the weight and size of the housings.

Robot motion: This is characterised by the shape design of the module bracket. The bracket itself must facilitate both linear and rotational motion. The actuator must be able to function within its programmed path without restriction to an extension of 20 mm. This will lead to stretching the skin of the robot (discussed in detail within Section 4.3) exposing the scale of this outer layer creating additional friction of the modules to aid rectilinear motion. The 180° sweep of the rotational actuators (servo motors) must also not be inhibited by the bracket design. This sweeping movement will aid in the last two types of motion (serpentine and sidewinding) to create the required sine waves of the housings. Furthermore, the module weight should be reduced as far as possible whilst still maintaining necessary strength to ensure the actuators can function as effectively as possible.

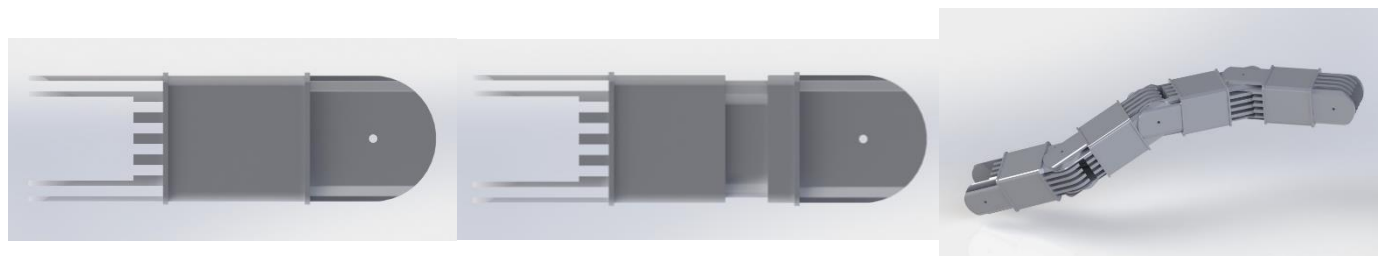
Skin integration: This is characterised by the module exterior. The skin must be able to function fully under extension and compression of the module. The housing must have mounting points for the skin as well as a recess to prevent the skin from sustaining damaged when in its passive state.

The initial skeleton design, shown in Figure 6 (a, b and c) below, was a basic design that met the design criteria listed above. This model entailed a two-part module design and a skin recess. The skin recess was to be developed in order to aid the smooth motion of the snake by ensuring a flat-as-possible skin surface is maintained when the module is retracted. It was noted that this design only allowed for a rotational range of 120°. In order to achieve the three types of motion set out within the project objectives, this range was to be increased to 180°. Additionally, an internal structure was to be developed to house the updated components list to include the linear actuator control board (50 mm x 50 mm).



Figures 6. Design Iteration 1 in SolidWorks (a), (b) and (c)

The second design iteration, shown in figure 7 (a, b and c) below, focused on the development of the bracket. This involved the implementation of a fin-like design to accommodate the 180° rotation of the actuator and implementing smooth edges to ensure that the bracket does not catch on any surface and impede the movement of the robot. Furthermore, the internal dimensions were altered to accommodate for the linear actuator control board, Arduino, and to provide more space for additional wiring. A sliding internal structure was incorporated that allows for components to be mounted external to the module then slid in, simplifying the assembly and maintenance process. Technical drawings for this design can be seen in Appendix 2.



Figures 7. Design Iteration 2 in SolidWorks (a), (b) and (c)

4.1.2. Determining the number of modules

To determine the maximum number of modules that could be present in the robot, a simplified approach was taken where one motor would be used to lift the whole robot. This equation was based on a single motor creating a pivot in the horizontal position to form a cantilever beam as shown in Figure 8 below, where the single motor will experience the maximum force.

N	Number of modules
m	Mass of each module (0.18 kg includes motor, actuator and housing of the module)

Table 2. Defining the variables

X	Length of each module (0.107 m taken from CAD drawing)
L	Total length
T _{max}	Maximum torque output of the motor (1.50 Nm)

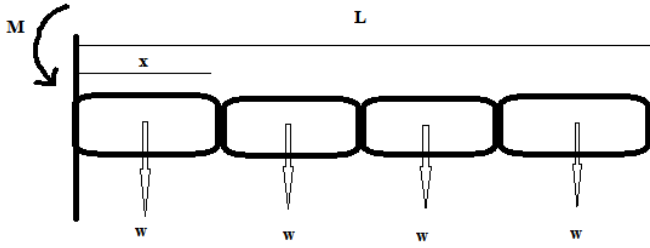


Figure 8. Diagram of the modules

The calculation is as follows with the variables defined in Table 2 above:

Determining weight (w),

$$w = mg = 0.18 \times 9.81 = 1.77 \text{ [N]}$$

Moment at the cantilever constraint, M , is considered equals to the torque output of AX-12A servomotor. Therefore,

$$M = NXw \times \frac{N}{2}$$

$$M = \frac{wXN^2}{2}$$

$$1.5 = \frac{1.77 \times 0.107 \text{ N}^2}{2}$$

$$1.5 = 0.094 \text{ N}^2$$

$$\text{N}^2 = 15.96$$

$$\text{N} = 4 \text{ modules}$$

As a result, a single motor allowed 4 modules to lift the entire robot. However, the motion of the robot within the project context requires only one or two modules to be lifted at a time with at least one module or more in contact with the ground, as such the entire robot will never be required to be lifted by one motor. Thus, this result is an overestimate but a good base to start with.

4.2. Software

4.2.1. Hardware workflow map of motors

Figure 9 shows the initial hardware workflow map of the components for the motor control, these components are listed in Appendix 3 under Component list for the Motors. Figure 9 has been simplified to show only one motor with its output cable connecting to the SMPS2 Dynamixel. The laptop (with Xenial Ubuntu 16.04 and ROS Kinetic) acts as the microcontroller for the motors – scripts are created and hosted on “nodes” through ROS kinetic that sends a list of instructions to control the motors. The U2D2 device allows the integration of the AX-12A motors with the Laptop, since it has no inherent way of connecting to 3 pin input/output cables. The AX-12A motor acts as a “smart” servomotor and provides position feedback which can be captured by the laptop to fine tune the control. Figure 9 shows the U2D2 connecting to the AX-12A motor with a 3P TTL cable at its input, and the output cable connecting to the SMPS2 Dynamixel. The SMPS2 Dynamixel acts as a power supply adaptor to link the SMPS 12V 5A power supply to the AX-

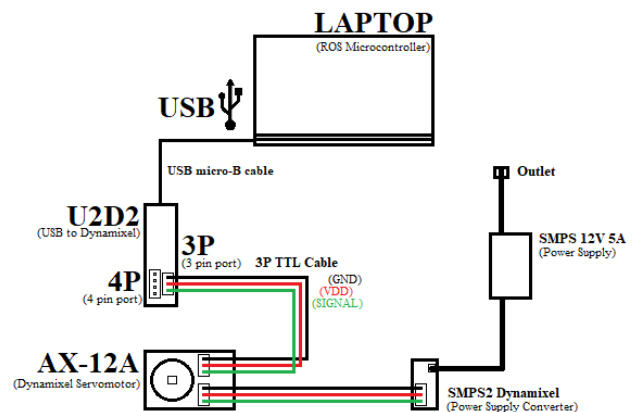


Figure 9. Hardware workflow of the motor control

12A motors. The laptop only acts as a microcontroller for the motors, which is why a separate power supply is used.

The AX-12A motors can also be daisy-chained together to reduce wire complexity by connecting the AX-12A's output cable to the input of another AX-12A motor and the output of the final AX-12A motor connects to the power supply adaptor. This daisy-chain can be repeated until the number of motors exceeds the power supply's capacity. This constraint is caused by resistance in the wires and the limit of the current drawn that leads to a drop in the power further down the chain. The number of motors required for a noticeable difference in performance is unknown and not well documented – requiring further experimentation. A possible solution, currently under investigation, is to use a 6 port AX/MX power hub to replace the SMPS2 Dynamixel power supply converter. This will allow multiple daisy chains to tether from the microcontroller port. To meet the final design of 8 modules, the 5 free ports on the AX/MX power hub offer a few options: 4 daisy chains of 2 motors or 2 daisy chains of 4 motors, depending on the daisy chain limit.

4.2.2. Motor selection

Servo motors are the most desirable type of motor for a robot and its individual joints. It allows precise control of angular position, linear position, velocity and acceleration. It is also common for servo motors to have a high-power output to size and weight ratio making it ideal for modular setups. However, the most common issue associated with servo motors is that they are typically unidirectional in communication. This means a controller sends a packet to the motor for instructions of movement and the motor executes in an open loop format. For a servo that lacks a position feedback loop, the controller will receive no feedback therefore it is unknown whether the motor has reached its target position and its current position. The wiring can also be complex with the controller using 3 pin ports for each motor.

Fortunately, these issues can be resolved using a “smart” servo motor. A “smart” servo motor performs the same functions as a regular servo, but they also allow additional features such as:

- Position feedback (more accurate movement and other uses – *mimicry*, *home position*, *reset*)
- Motors can be joined in series – daisy chaining (less complex wiring)

Mimicry: Due to position feedback, it is possible to physically manipulate the motors manually in real-time and, with the correctly setup program, record the current position of the motors. Because of this, a series of motors can be setup and each position recorded allowing for simple acquisition of starting positions and tracking the movement of the robot.

Home position: With position feedback the starting position can be set for the robot allowing it rest in a certain position before it begins a specified motion or action. This improves the accuracy and reliability of the task to be followed by the robot especially as the motors can be easily manipulated whilst the program is off thus affecting the pre-programmed position settings.

Reset: This is similar to home position however it resets all motors to 0 degrees. Position feedback allows the creation of a bespoke “reset” position.

“Smart” servo motor options

There are several “smart” servo motors on the market made by various manufacturers, with the main options identified as: Dynamixel, Herkulex and Kondo.

Herkulex and Kondo are cheaper alternatives to Dynamixel (DXL), but their software stability is unknown and less documented. DXL are slightly more expensive, but the support framework for DXL is more stable meaning certain software or programs used by DXL motors are kept up to date making it more reliable. Additionally, Dynamixel have a wide range of skeletal brackets available to physically link motors together. By initially using these skeletal brackets, resources can be used elsewhere to aid project progress with a possibility of using bespoke brackets in later stages of the project.

Dynamixel Servomotor options

Dynamixel hosts a wide range of servomotors all with different purposes, however for this project the AX series of servomotors was chosen based on the motor details outlined in Table 3 below. The main requirement for the motors was a high torque output with little concern for the magnitude of the RPM, thus the AX-12A model was selected. Additionally, this model was within the allocated budget range in comparison to other series that DXL offered.

Table 3. AX Series Dynamixel Motor Specifications

Motor	Voltage [V]	Torque [N m]	Current [A]	Speed (No Load) [rpm]	Link	Size [mm]	Weight [g]	Price [£]
AX-12A	9 ~ 12	1.50	1.5	59	TTL	32x50x40	54.60	40
AX-12W	9 ~ 12	0.21	1.4	470	TTL	32x50x40	52.90	40
AX-18A	9 ~ 12	1.80	2.2	97	TTL	32x50x40	55.90	87

4.2.3. Control loop of micro-controller to actuator

The hardware workflow of the actuator control is shown in Figure 10 below where it has been simplified to demonstrate the wiring of two actuators to their control board. It entails the PQ12 Linear actuator, an actuator control board per actuator, a slave Arduino per actuator and a single master Arduino that connects to the computer through a USB connection. This link to the computer allows for the actuator control to be part of the ROS network represented as a node by using the master Arduino. One actuator along with its control board and slave Arduino sits within each module of the robot, bar one module that also contains the master Arduino.

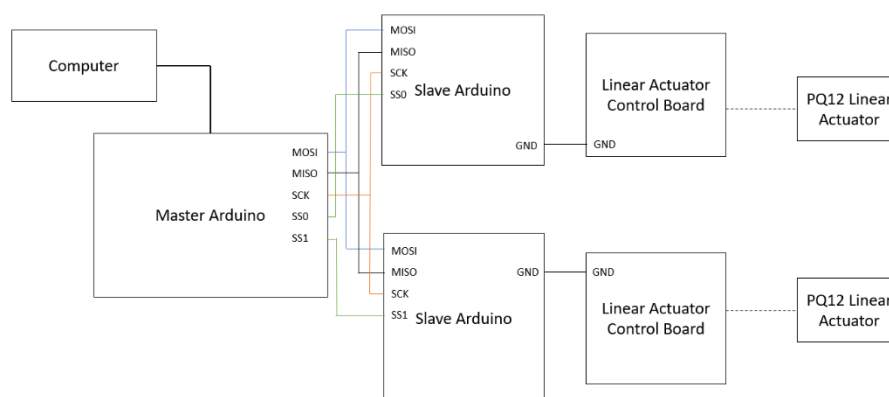


Figure 10. Hardware workflow of PQ12 Linear Actuators

Linear Actuator Control Board

Each PQ12 Linear actuator is controlled by an Actuonix Linear Actuator Control Board. The control board allows actuator adjustments of the speed, limits (minimum and maximum actuator positions) and accuracy (changing the magnitude of actuator movement increments).

The control board has several control modes. However, for this project, the two potential control modes being considered are the RC Servo Interface mode and PWM mode. The RC Servo Interface mode works in a similar way to servos using the standard RC cable (three wires – red, black, white) and the servo library within the Arduino software. A 1 ms pulse commands the controller to fully retract the actuator, and a 2 ms pulse signals full extension. On the other hand, the PWM mode allows for the control of the actuator using a single output pin from the slave Arduino. The present duty cycle sets the actuator position to the same percent of full stroke actuator extension – 100% duty cycle represents full extension and 0% duty cycle represents full retraction. The ease of both set-ups will be tested during actuator testing then the final mode chosen.

SPI control between the slave and master Arduino

Both control board modes connect to the slave Arduino. This Arduino is then connected to a second Arduino known as the master Arduino. This master Arduino determines which slave Arduinos are activated for actuator movement and ultimately, controlling the skin movement.

The Serial Peripheral Interface (SPI) bus has been chosen as the form of communication between all Arduino's due to its high speed and reduced wire (only 4 wires) advantages. As shown in Figure 10 above, the four connections are: MOSI (Master Out Slave In), MISO (Master In Slave Out), SCK (Slave Clock) and SS (Slave Select). MOSI, MISO and SCK can be daisy-chained for each Arduino with solely SS being a new wire per module. The SPI communication entails the Master Arduino communicating with one slave device per time by setting the SS wire. Usually this input is an active low signal; hence, the master must send a logic 1 on this signal to select the slave device (Dhaker, 2018).

SSH control between master Arduino and computer

The most straight-forward method of connecting the master Arduino to the computer is to use a Secure Shell network protocol (SSH). An SSH interface between two devices allows the initiator of the SSH protocol to do any of the following: tunnelling, share files, and execute commands via the command line. In this case, execution of commands via the command line is the most important. In order to establish an SSH connection between two devices, it is required to have the following: host names for both devices, and a network connection between the two devices (either connected to the same WiFi network or connected via Ethernet). Once an SSH connection is established, the laptop can then execute a command via SSH to configure the master Arduino's ROS URL to match the ROS URL the computer in order to join the existing ROS environment of the Dynamixel motors. As a result, a "talker"/"listener" node is created to permit data transformations.

4.3. Skin

The primary purpose of the skin design was to facilitate rectilinear motion through friction as is demonstrated by snakeskin scales in nature (Lissmann 1950, Abdel-Aal 2018). Thus, the skin model considerations were as follows:

- 1) What mechanism would be used to simulate muscle force to propel robot forwards?
- 2) What mechanism would be used to simulate the snakeskin scales' anisotropic frictional interaction between the robot body and the surface it is travelling on?
- 3) Would the skin be a sheath surrounding the chain of modules or coat each module individually?
- 4) How would this skin be attached to the housings?

Question 1 - While there are numerous studies that explore pneumatic avenues (Liljebäck et al. 2008, Marvi et al. 2011, Marchese et al. 2015), it was decided not to pursue such a system as this would require additional testing and fitting of several controller boards, a pressure sensor system and an air pump within the housing of at least one of the modules. Consequently, this would increase the housing size drastically and ensuring the air-tightness of the system would also prove to be a difficult task. Thus, a pneumatic system was concluded to be too impractical to implement successfully in the given timeframe. Therefore, the alternative method selected was to simulate muscle motion of the snake robot using electrical linear actuators.

Question 2 – A plausible option identified through research was proposed by Tramsen (2018), shown in Figure 11, where nibs allow for smooth and swift motion in one direction, but high friction in the opposite direction. However, while a particular scale orientation would facilitate rectilinear motion perfectly, concern was expressed over how this might interfere with the other motion sequences the robot was intending to achieve.



Figure 11. Multi-material snake scale design

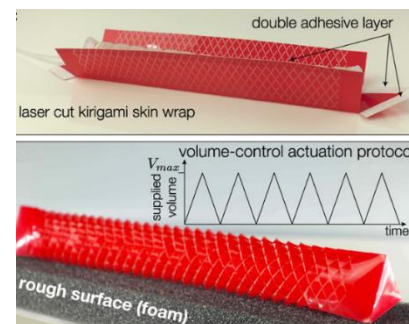


Figure 12. Kirigami – patterned skin layer was formed for a pneumatic robot system

Therefore, another design investigated by researchers at Harvard University (Rafsanjani et al. 2018) was considered. This used the art of kirigami to create a sheet of scales that buckle and pop out when the sheet is extended as shown in Figure 12 above. Since the system was optimised to function with a pneumatic elastomer actuator, consideration had to be taken in order to make this mechanism work using electrical linear actuators.

Question 3 - As for question 3, it was decided to have individual skin coverings around each module because it was likely that the power capacity of the actuators would not be enough to generate sufficient extension of the skin if the skin was one large sheath encapsulating the chain of modules. Additionally, it was predicted that a single skin sheath may restrict the modules from performing the other target types of motion.

Question 4 - Question 4 has yet to be determined at this stage, but the technique that is to be decided on should ensure the skin is firmly fixed to the housing without hindering the skin's extension and the scales' interaction with the ground surface.

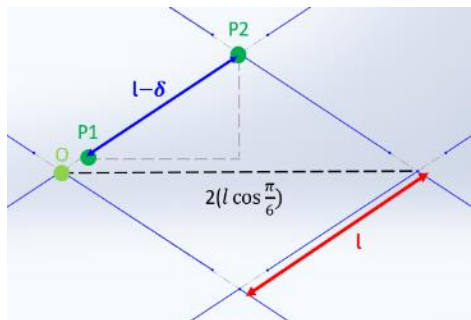
4.3.1. Skin design with material selection

After answering the questions specified above, the final decision to be made was the material(s) that would allow the skin design and functional requirements to be met. Below is a table summarising the properties of the materials explored and why they might be good for the purpose of this project's objectives. Initially, silicone and acetate were acquired because they appeared to be the most promising options.

Table 4. Material properties

Material	Stiffness	Uses	Manufacturability	Advantages	Concerns
Silicone	Between 0.001 - 0.05 GPa (Azo materials 2019a)	Wide application in mechanical, electrical, medical industries	Purchasable in 600x600 mm sheets with varying thicknesses (1.5-6 mm etc.)	Good flexibility, anti-adhesive properties and high gas permeability	May be too deformable to frictionally interact with ground
Braided fibreglass	Unknown (RS PRO 2019a)	Bandages, cable sleeving	Purchasable as cable sleeving with potentially a maximum diameter of 20 mm	Good flexibility and fray resistant when cut	Making cuts in the material may cause the structure to unravel
Ninjaflex	0.1124 GPa/ 0.15 GPa (under tensile/compressive loading) (Hanifpour et al. 2018)	Lattice structure manufacturing, prototype development etc.	Purchasable as cartridge roll form which a skin design can be additively manufactured	Highly elastic (up to 600% elongation capability), additive manufacturing means it can be used to create complex designs	May be too stiff for the actuator to expand and contract it
Polyester plastic	Between 2 – 4 GPa (Azo materials 2019b)	Capacitors, graphics, recording tapes	Purchasable as 457x305 mm sheets with varying thicknesses (0.05-0.51 mm)	Comes in various thicknesses, therefore ideal for experimentation	May be too thin for scales to buckle up underneath robot body
Cellulose acetate	Between 2.4-4.1 GPa (Azo materials 2019c)	Photographic film, wound dressing,	University teaching office has A4 supplies	Good toughness, reasonably hard for a thermoplastic, high stiffness for such thin sheets	May be too thin for scales to buckle up underneath robot body

Using the dimensions specified by Harvard researchers (Rafsanjani et al. 2018), as shown in Figure 13, and taking $\mathbf{O}=[0,0]$, $\mathbf{P2} = \left[l \cos \frac{\pi}{6}, l \sin \frac{\pi}{6} \right]$ and $\mathbf{P1} = [\mathbf{P2}] \frac{\delta}{l}$, the cut length $[l - \delta]$ was able to be determined, where δ is the hinge size. Then, this scale pattern was created using SolidWorks 3D. The initial housing design had four flat-surfaced sides of lengths 40 mm each (as explained in Figure 14). Hence the horizontal length $l \cos \frac{\pi}{6}$ (as indicated in Figure 13) was set to 5 mm and used as the driving dimension for the rest of the parameters in order to fit this constraint. The GlobalMAX waterjet cutting machine (accuracy to the nearest 0.1 mm) was then used to manufacture the pattern.



Cut length —————

Construction net length —————

Figure 13. Scale dimensions

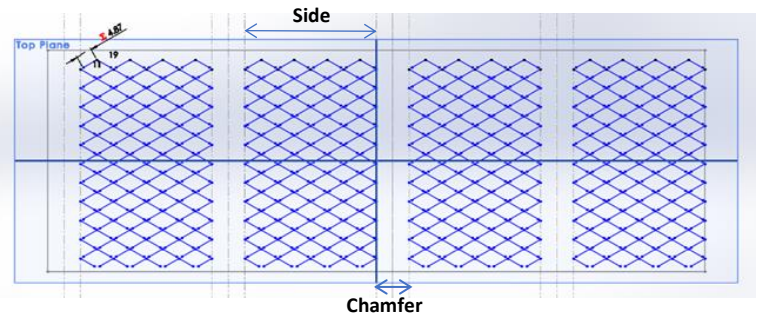


Figure 14. – This is the whole sheet design for a single module housing unit with 4 sides and chamfered edges (as shown in Figure 6c). In order to fit 4 scales per side, the total horizontal length was set to $2(l \cos \frac{\pi}{6}) = 10$ mm. Therefore, proportional cut sizes were derived from this term

Unfortunately, after manufacturing the first sample, it was found that the hinge sizing was too small, causing the scales to break off when extended. Ideally, cut sizes also needed to be bigger in order to effectively buckle under the weight of body, but not too large so to deform and flatten against the ground surface (Tramsen et al. 2018). Thus, in order to increase these two parameters, a new housing was designed with larger flat surface side lengths of 50mm. Scale numbers of 2 and 3 per row were experimented with, along with hinge sizes (δ) ranging between $\sim 1.5 - 2.5$ mm.

Silicone samples are currently being prepared and printed, ready for testing with the actuators next semester. Future steps include reinforcing the silicone sheet with acetate attached together by double adhesive tape, then testing this combination against the original, untampered silicone. It was agreed to pursue this avenue as the acetate sheets alone were anticipated to be too thin for functioning under the weight of the robot housing. Also, as suggested by literature (Tramsen et al. 2018), silicone scales may end up pressing against the ground instead of gripping it if the stiffness is not increased.

4.3.2. Actuator selection

Actuators - To extend the module housing, and therefore the skin, a linear actuator will be used. This motion is shown in Figures 7a and 7b in Section 4.1. The chosen actuator must meet a range of requirements for the functionality of the module. It must be able to retract and extend while withstanding resistive forces due to the weight of the other modules, elasticity of the skin and sliding friction within the module housing. It must also be as small as possible to minimise the size and mass of each module. Additionally, the ability to be controlled individually would be advantageous as it allows different patterns to be propagated throughout the entire body. Finally, position feedback would also be useful to determine how the snake is moving compared to how it has been commanded to be moved.

Three types of linear actuators have been considered for this project; hydraulic actuators, pneumatic actuators and electromechanical actuators.

Hydraulic linear actuators consist of squeezing an incompressible liquid within a cylinder to move a piston producing mechanical work. To control this compressing motion a pump is used to increase the pressure in one side of the chamber by forcing more fluid into it. By using incompressible fluids, hydraulic actuators can act under high forces and hold their force and torque at constant values without the addition or removal of fluid. However, hydraulic actuators require a lot of maintenance due to leakage and contamination. Temperature changes and leakage can also alter the accuracy and position of the piston positioning. Furthermore, hydraulic actuators require a lot of extra equipment (tank, motor, pump, release valves) adding to the cost and space required for packaging (Rosenfield, 2017).

Pneumatic linear actuators work in a similar way to the hydraulic type, consisting of a piston inside a cylinder, but being controlled by a pressurised gas. Again, an external pump or compressor is needed to control the flow of gas into the cylinder. The advantages of using pneumatic actuators are that they are relatively cheap, can work at high speeds and forces and can resist shocks with minimal damage. However, they are less efficient than other methods of actuation due to pressure loss and the challenges associated with compressing air (Rosenfield, 2017). For pneumatic actuation to be accurate and efficient it needs to be designed to fit its purpose, requiring proportional regulators and valves, adding to the overall cost.

Electromechanical linear actuators take the rotational force of a motor and convert it to linear movement (Rosenfield, 2017). Electric actuators are easily controlled and do not require the external pumps or compressors of the other methods listed above. Multiple actuators can be easily controlled synchronously at once using a micro-controller. As the speed of the movement of the actuators can be easily controlled, they can be programmed to stop and start with an increasing deceleration and acceleration respectively, reducing shock loadings at the beginning and end of movements. They do not use fluids so there is no chance of leakage and are not hugely affected by temperature. On the other hand, they cannot compete with the higher forces and speeds that pneumatic and hydraulic actuators can. If shock loadings do occur, they can affect the components within it, leading to permanent losses in performance.

For this project, electromechanical actuators have been chosen as they do not require the extra equipment and range of high forces produced by hydraulic and pneumatic actuators. They also do not require sealing, which could be difficult to do on a small scale for the individual modules.

To decide on a commercially available electric actuator, a few were taken into consideration. The specifications of these actuators are listed in Table 5 below.

Table 5. Comparison of commercially available electric actuators

Model	PQ12 Linear 100:1 6v	Crouzet Linear Actuator 5.6v dc	Inline Nexus Linear Actuator, 25 mm, 12 V w/ Limit Switches
Cost [£]	61.42	42.07	61.40
Mass [g]	15	90	59
Stroke Length [mm]	20	10	25
Max Speed [mm/s]	10 (No load), 6 (40N load)	1.67	0.51
Retracted Dimensions [mm]	47.5 * 21.5 * 15	56.4 * 47.2 * 46.0	82 * 23 * 18.5
Extended Dimensions [mm]	67.5 * 21.5 * 15	66.4 * 47.2 * 46.0	107 * 23 * 18.5
Input Voltage [V]	6	5.6	12
Max Load [N]	50	43	25

The PQ12 Linear actuator has been chosen as it is small and light while still having the largest maximum load and drive speed of the considered actuators.

5. Plans moving forward

5.1. Software

5.1.1. Motor testing

The aim of these motor tests is to form the motion of the modular snake robot. To achieve these aims, the following objectives are required:

1. Install and setup Xenial Ubuntu 16.04 to a laptop
2. Install and setup ROS Kinetic to the laptop running Xenial
3. Interface the Dynamixel motor with the laptop with the components outlined in Appendix 3

4. Setup a ROS environment and ROS nodes for the Dynamixel motor to advertise diagnostics (goal position, position, velocity, etc.)
5. Publish to a node manually via the terminal to achieve motion of a Dynamixel motor
6. Create a “talker” node, configured using Python, to send information on position to achieve planned motion of a Dynamixel motor
7. Create a “listener” node, configured using Python, to receive information on position feedback from the Dynamixel motor’s advertised diagnostics
8. Achieve above objectives with multiple linked Dynamixel motors using basic motor brackets
9. Achieve above objectives using the 3D printed skeletal bracket

Background - The objectives outlined above will be standalone. The steps to achieve each objective is more involved, but each objective is dependent on the project aim to be able to build a platform for motion for the modular snake robot. As an example: ROS Kinetic is only compatible with a few Linux distributions, and there is little support for other operating systems such as Mac OS and Windows. ROS Kinetic is an important aspect to the project as a whole and can future proof potential additions to the project. Once a ROS environment is setup, the chain of communication between devices is more streamlined, so that if more sensors are tuned in at later stages, they can “communicate” within the already existing network. By advertising sensor feedback to a ROS node, a “listener” node can then pick up this new information and then a “talker” node can change the set of instructions sent to the Dynamixel motors. So, for the purpose of the report, motor testing will be summarised in Table 6 below to show current progress and future plans.

Table 6. Motor testing summary

Objective No.	End Goal	Completion date	Completed
1	Working Linux OS	29/10/2019	Y
2	Working ROS Kinetic	29/10/2019	Y
3	Working connection between Dynamixel and Laptop	12/11/2019	Y
4	Advertise Dynamixel diagnostics	19/11/2019	Y
5	Manual manipulation of Dynamixel	19/11/2019	Y
6	Automatic manipulation of Dynamixel	26/11/2019	Y
7	Allow position feedback control for Dynamixel	04/12/2019	Y
8	Achieve objectives 3 to 7 for multiple Dynamixel motors	Spring Semester	N
9	Test 3D printed skeletal bracket under the motion of the modular snake robot	Spring Semester	N

5.1.2. Actuator control assembly

The mode for the LAC control board to the actuator will be selected during set-up. The decision will be made based on the method with simpler wiring and ease of coding. Time will then be spent on determining the actuator control loop of master Arduino to slave Arduino communication and ensuring the master Arduino has a present node within the current ROS network.

5.1.3. Sensor Research

Research to the type of sensor to be used to allow the robot to be aware of its surroundings will be conducted. The types of proximity sensors to be investigated include, but not limited to, force sensitive resistors, LIDARs and tactile switches. The investigation includes understanding how these sensors work including the electronics and software side, how to integrate these sensors into the robot by embedding into the skin or attachment to the housing, and the feedback loops of the sensors.

5.2. Skin

5.2.1. Skin material testing

Once samples have been manufactured, at least three tests would have to be conducted, these include material characterisation, locomotion testing and frictional testing.

1) Material characterisation

This would consist of using the Zwick Z050 machine (accuracy to the nearest 0.001 mm) to perform a series of uniaxial tension tests on different design samples. Force and displacement data would be obtained, from which a Young's modulus value could be derived. Therefore, by identifying the difference in stiffness between normal samples and the scaled samples, this will clearly quantify how the materials' behaviour changes as a result of the cut design dimensions. Additionally, by conducting this test at the actuator peak efficiency settings (20 N at 8 mm/s), this will give an indication of whether the scales will successfully buckle and pop out without the hinges breaking if the material was extended by the actuator force.

2) Locomotion testing

This would involve identifying the distance the robot travels in one gait cycle (where gait cycle can be defined as a single extension and retraction sequence of the modules). The actuator settings could then be changed (i.e. applying forces in intervals up to its maximum force of 50 N) therefore establishing whether a greater force exerted by the actuator results in a greater rate of movement by the robot. Additionally, this procedure could be repeated on different ground surfaces of varying friction in order to compare how well the robot functions on different terrain.

3) Frictional testing

After conducting the locomotion testing on various terrains, it will also be useful to obtain the frictional coefficient μ of each material sample for further analysis. This was partially explored by a study in 2018 (Tramsen et al. 2018), where one of the tests included a stiff and soft sample being tested against a soft surface. The soft sample was then tested against a variety of other surfaces, but not the stiff sample. Therefore, this project would seek to investigate all possible interactions between the different types of robot skin designs and ground surfaces, in order to obtain the optimal interaction conditions. This would be conducted using a pin on disk tribometer, which consists of a stationary pin loaded against a rotating disc. The frictional coefficient is then determined by the ratio of the frictional force to the loading force on the pin.

5.3. Skeleton

5.3.1. Kinematics

To increase the accuracy of the kinematic calculations, future steps include using the Jacobian matrix and inverse kinematic calculations to define the dynamic relationship between two different modules: displacement and rotation. These values will then be used within the code for the motor to produce the commands to mimic the snake motion.

5.3.2. Housing Design

Future steps of the housing design include further optimisation of the module interior when the full component list is finalised to ensure an effective lay out of all components. The exterior dimensions may also be altered to best accommodate the skin design once testing has taken place and the skin

layer has been optimised. The design must also be tested, and topology optimised to reduce weight. Finally, the manufacturing method must be chosen, and a manufacturing plan implemented.

6. Risks to completion

The management risks and risks specific to sub-team tasks were determined and filled out in the Risks Table shown in Appendix 4. Each risk was categorised by the probability of it occurring and the severity of consequence if it were to occur. A contingency plan to mitigate each risk was then formed.

7. Conclusion

This semester firstly concentrated on becoming familiar with the design of chain-type modular robots by conducting a literature review. Based off the learnings taken from these designs, the first module iteration was produced. Focusing on the required capabilities of motion of the project, this design was modified to increase the range of rotational movement and enhance extension movement for the outer skin.

In terms of rotational movement, servo motors were deemed the most suitable motor to replicate the motion of a snake as they allow precise control of angular position and velocity. Dynamixel AX-12A “Smart” servo motors were chosen to allow position feedback and to reduce the complexity of the wiring via daisy-chaining of motors. These motors were low in cost whilst also meeting the general torque requirements. Calculations based on the torque of the AX-12A motor determined the maximum number of modules to be 4, this was based on one motor independently lifting the entire chain. As the robot will always have contact points with the ground during all three types of motion, this was an overestimate but a good base to start with.

The control of the motors uses a laptop with Xenial Ubuntu was used to act as the microprocessor. ROS was used as the common environment to connect different devices and software via nodes which allow the transfer of information and manipulation of devices within the ROS environment. This allowed for easier integration of additional devices or sensors in the future of this project. The following have been achieved as a result: setup of a ROS environment with linux; a connection established between the laptop and a dynamixel motor leading to manual and automatic manipulation of the motors position via a Python script through a ROS node and a closed loop system to enable position feedback to allow smoother motion for current and more complex motions in later stages of the project. However, there is a concern of the daisy-chain limit for the Dynamixel motors which is currently under investigation. This and the connection of motors to work simultaneously, then to achieve the three types of motion are areas of investigation for next semester.

With regards to the skin design progress, linear actuators have been selected to simulate muscle propulsion and kirigami-inspired patterns have been experimented with to mimic snake scale behaviour. This semester has focused on the refinement of the kirigami pattern dimensions and successfully manufacturing these intricate cuts onto the silicone material. The next semester will focus on developing the control of the actuators and then fitting the actuators into the module housing; attaching the skin to the housing and ensuring these components work cohesively to provide additional friction effectively to perform rectilinear motion.

8. References

Abdel-Aal, H. 2018. Surface structure and tribology of legless squamate reptiles. *Journal of the Mechanical Behaviour of Biomedical Materials* 79, pp. 354-398. doi: 10.1016/j.jmbbm.2017.11.008.

Adafruit Learning System. (2019). Force Sensitive Resistor (FSR). [online] Available at: <https://learn.adafruit.com/force-sensitive-resistor-fsr/overview> [Accessed 18 Nov. 2019].

Akbarzadeh, A. and Kalani, H. 2012. Design and Modelling of a Snake Robot Based on Worm-Like Locomotion. *Advanced Robotics* 26(5-6), pp. 537-560. doi: 10.1163/156855311x617498.

Azo materials 2019a. Properties: Silicone Rubber. Available at: <https://www.azom.com/properties.aspx?ArticleID=920> [Accessed: 26 November 2019].

Azo materials 2019b. Polyethylene Terephthalate Polyester (PET, PETP) - Properties and Applications. Available at: <https://www.azom.com/article.aspx?ArticleID=2047> [Accessed: 27 November 2019].

Azo materials 2019c. Properties: Cellulose Acetate. Available at: <https://www.azom.com/properties.aspx?ArticleID=1461> [Accessed: 26 November 2019].

Black Stem Tactile Switch, S. (2019). SKHHAJA010 | Black Stem Tactile Switch, Single Pole Single Throw (SPST) 50 mA @ 12 V dc 4.3mm | RS Components. [online] Uk.rs-online.com. Available at: <https://uk.rs-online.com/web/p/tactile-switches/7581922/> [Accessed 22 Nov. 2019].

Circuit-diagram.org. (2019). Circuit Diagram Web Editor. [online] Available at: <https://www.circuit-diagram.org/editor/> [Accessed 20 Nov. 2019].

Dhaker, P. 2018. Introduction to SPI Interface | Analog Devices. Available at: <https://www.analog.com/en/analog-dialogue/articles/introduction-to-spi-interface.html#> [Accessed: 27 November 2019].

Encyclopaedia Britannica. 2020. Snake Locomotion. Available at: <https://kids.britannica.com/students/assembly/view/171904> [Accessed 30th March 2020]

Hanifpour, M. et al. 2018. Mechanics of disordered auxetic metamaterials. *The European Physical Journal B* 91(11). doi: 10.1140/epjb/e2018-90073-1.

Li, M. et al. 2016. 3-DOF bionic parallel mechanism design and analysis for a snake-like robot. *2016 IEEE International Conference on Robotics and Biomimetics (ROBIO)* . doi: 10.1109/robio.2016.7866272.

Liljebäck, P. et al. 2008. Modular Pneumatic Snake Robot: 3D Modelling, Implementation And Control. *Modelling, Identification and Control: A Norwegian Research Bulletin* 29(1), pp. 21-28. doi: 10.4173/mic.2008.1.2.

Lin, Y.C. and Chou, J.J. 2005. Development and analysis of a snake robot with flexible connectors. *Proceedings, 2005 IEEE/ASME International Conference on Advanced Intelligent Mechatronics*. doi: 10.1109/aim.2005.1511147.

Lissmann, H. 1950. Rectilinear locomotion in a snake (Boa Occidentalis). *Journal of Experimental Biology* 26(4), pp. 368-379.

Marchese, A. et al. 2015. A Recipe for Soft Fluidic Elastomer Robots. *Soft Robotics* 2(1), pp. 7-25. doi: 10.1089/soro.2014.0022.

Marvi, H. et al. 2011. Scalybot: A Snake-Inspired Robot With Active Control of Friction. *ASME 2011 Dynamic Systems and Control Conference and Bath/ASME Symposium on Fluid Power and Motion Control, Volume 2* . doi: 10.1115/dscc2011-6174.

Mohammed, I. et al. 2016. Design and control architecture of a 3D printed modular snake robot. *2016 World Automation Congress (WAC)* . doi: 10.1109/wac.2016.7583009.

Moon, B. 2001. Snake locomotion. Available at: <https://userweb.ucs.louisiana.edu/~brm2286/locomotn.htm> [Accessed: 27 November 2019].

Rafsanjani, A. et al. 2018. Kirigami skins make a simple soft actuator crawl. *Science Robotics* 3(15), p. eaar7555. doi: 10.1126/scirobotics.aar7555

Robot Shop. (2019). [online] Available at: <https://www.robotshop.com/media/files/pdf/single-point-10mm-force-sensor-solder-tab-datasheet.pdf> [Accessed 20 Nov. 2019].

Rosenfield, S. 2017. Pros and Cons of Pneumatic, Hydraulic, and Electric Actuation. Available at: <https://electronics360.globalspec.com/article/9480/pros-and-cons-of-pneumatic-hydraulic-and-electric-actuation> [Accessed: 27 November 2019].

RS PRO 2019a. RS PRO Expandable Braided Fibreglass Natural Cable Sleeve, 20mm Diameter, 5m Length | RS Components. Available at: <https://uk.rs-online.com/web/p/cable-sleeves/6681298/> [Accessed: 26 November 2019].

RS PRO 2019b. RS PRO Shim Kit, Plastic - Black, Blue, Green, Grey, Natural, Orange, Red, Yellow | RS Components. Available at: <https://uk.rs-online.com/web/p/products/0770816/> [Accessed: 26 November 2019].

Tramsen, H. et al. 2018. Inversion of friction anisotropy in a bio-inspired asymmetrically structured surface. *Journal of The Royal Society Interface* 15(138), p. 20170629. doi: 10.1098/rsif.2017.0629

Yamada, H. and Hirose, S. [no date]. Study on the 3D shape of active cord mechanism. Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006. . doi: 10.1109/robot.2006.1642140.

Yang, B. et al. 2015. A modular amphibious snake-like robot: Design, modeling and simulation. *2015 IEEE International Conference on Robotics and Biomimetics (ROBIO)* . doi: 10.1109/robio.2015.7419054.

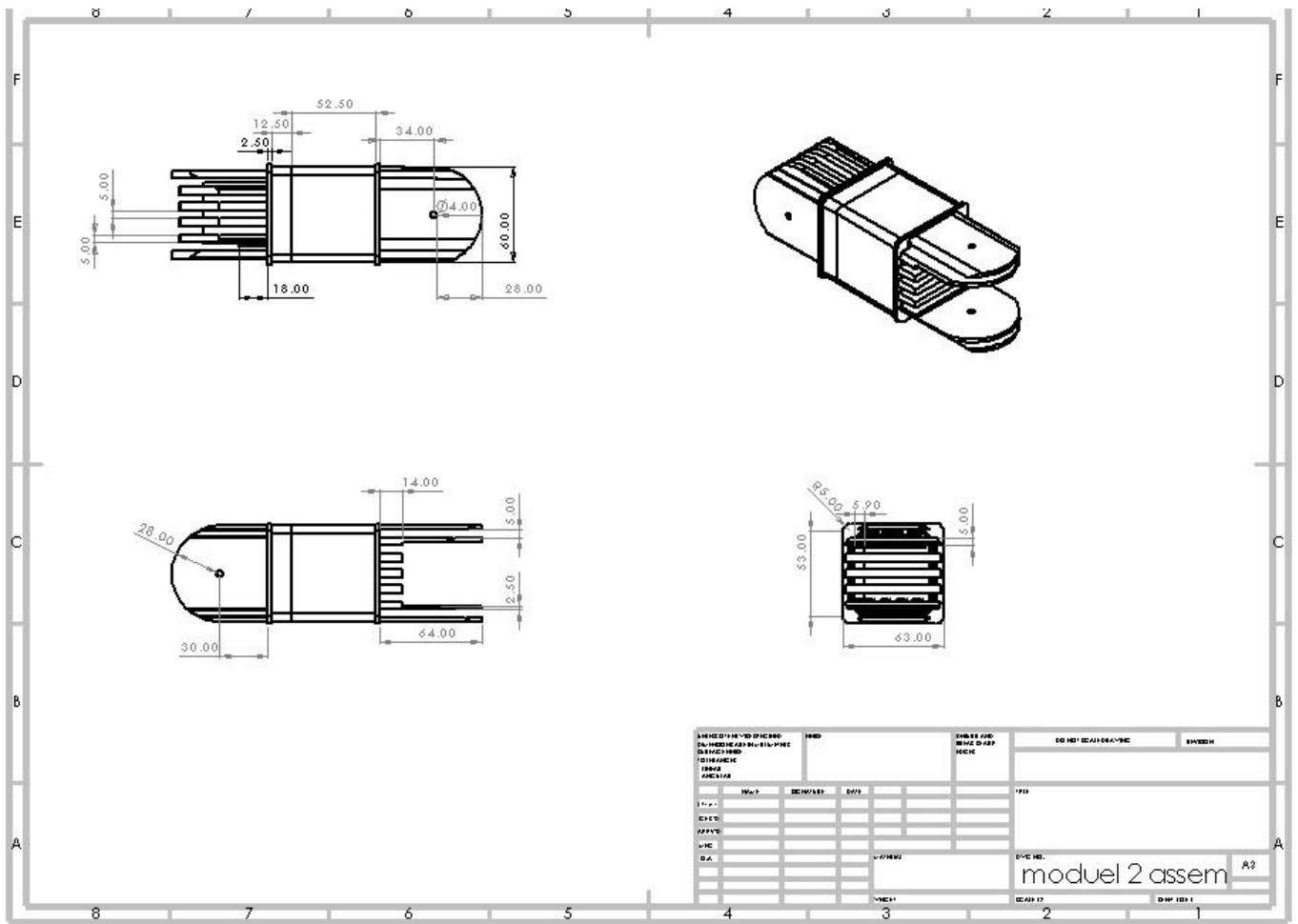
Zug, G. 1999. Locomotion | behaviour. Available at: <https://www.britannica.com/topic/locomotion> [Accessed: 27 November 2019].

9. Appendix

Appendix 1 – Gantt Chart

Tasks	Autumn Semester											Christmas Break			Exams			Spring Semester								
	Wk 1	Wk 2	Wk 3	Wk 4	Wk 5	Wk 6	Wk 7	Wk 8	Wk 9	Wk 10	Wk 11	C1	C2	C3	Exam 1	Exam 2	Exam 3	Wk 1	Wk 2	Wk 3	Wk 4	Wk 5	Wk 6	Wk 7	Wk 8	Wk 9
Material Selecting (including testing)																										
Design & Manufacturing																										
Testing																										
Integration																										
Housing Design																										
Housing manufacturing & assembly																										
Housing connections																										
Motor Control																										
Actuator Control																										
Sensor Control																										
Integration																										

Appendix 2 – Technical Drawings of Design Iteration 2



Appendix 3 – Component Lists

Component list for the motor:

- Laptop (Xenial Ubuntu 16.04 with ROS Kinetic)
- USB micro-B cable
- U2D2 (USB to Dynamixel adapter)
- 3 pin TTL cables
- AX-12A Dynamixel Motors
- SMPS2 Dynamixel (Power supply adapter)
- SMPS 12V 5A (Power supply)

Component list for the actuator:

- PQ12 Linear actuator
- Actuonix Linear Actuator Control Board
- Slave Arduino
- A Master Arduino

Components within the modules:

- 3 pin TTL cables
- AX-12A Dynamixel Motor
- PQ12 Linear actuator
- Actuonix Linear Actuator Control Board
- Slave Arduino
- A Master Arduino

Appendix 4 – Risk Table

Risk	Probability (P) 1 = Remote or improbably 2 = Probably or occasional, 3= Certain or frequent	Severity (S) 1 – High 2 – Medium 3 – Low	Overall risk Low: 1-3 Medium: 4-6 High: 7-9	Mitigation/ Contingency plan
Budget Risk	1	5	5	BOM tracker to ensure costs do not overrun
Integration Risk	3	2	6	Group meetings and sub-team leader meetings to ensure sub-teams integration
Resource Risk	2	3	6	Continuous assessment of resource allocation
Schedule Risk	2	3	6	Continuously assess and update Gantt Chart