Achievement Standards

AS 91608: Undertake brief development to address an issue within a determined context.

AS 91610: Develop a conceptual design considering fitness for purpose in the broadest sense.

AS 91639: Implement complex interfacing procedures in a specified electronic environment.

AS 91640: Implement complex techniques in constructing a specified complex electronic and embedded system.

Tom Mitchell
13CTE

Context

Arduino is an open source software platform designed to be executed on the Arduino Uno micro-controller. It has rows of digital and analogue I/O pins where connected electronic components are interfaced using compatible software (A derivative of C), which is developed in the Arduino Integrated Development Environment (IDE). The Arduino Uno utilises DC electricity, so the board also has 5V and ground pins with the option of connecting an external power supply. Libraries can be imported to enable more complicated software interfacing procedures such as the communication between micro-controller and select third-party components. The large Arduino userbase of micro-controller and Arduino software means there are forums where example code and libraries are shared which will aid me in developing my solution.
The Arduino Uno optimizes for efficiency rather than versatility. It operates on a 10-bit architecture which means that data is represented in 10-bits, so can be executed more quickly. A restriction of this however is evident when data is being transmitted to and from external components. It can only detect 1024 states of the reference voltage, restricting the magnitude of data delivered.

The **Raspberry Pi** is an alternative development tool. It is a series of single-board computers. The Model A+ is the low-cost variant of the Raspberry Pi at $50 (compared to the Uno's $30). Unlike the Arduino, which is simply a micro-controller, the Raspberry Pi is a pocket computer which runs a full desktop OS (namely Windows). Code can be developed and executed on the Raspberry Pi itself which allows for deeper integration between software and hardware. This enables incredible versatility. While optimized, the raspberry Pi doesn't process data as quickly as the Arduino since it also must simultaneously run a full desktop client.
The Raspberry Pi does not come with the same set of I/O pins as the Arduino Uno. To connect components, it is necessary to purchase external I/O modules. Since networking isn't integral to the use of the Raspberry Pi, support for components is rarer and libraries aren't as flexible.

The Arduino Uno is the preferable controller for my context. Support for external components surpasses those of the Raspberry Pi. Arduino has a considerably larger development community, so libraries and forums provide access to easy learning and developing of my programming skills. Examples of code found in these places can be incredibly helpful in improving the efficiency of my program. The controller is also cheaper, enabling funds otherwise spent on acquiring a more expensive controller to be used for better quality materials and components. The 10-bit architecture should not pose any restrictions on my solution since I should not be needing to send large packets of data.

Model Barrier Arm	Senior management at Westlake Boys High School has a pressing issue with the staff carpark. In the mornings before school and in the afternoons after school parents use the entrance to the carpark as a drop off zone. This introduces significant congestion among the cars in the carpark and threatens the safety of students who must cross the carpark to enter the school grounds. While rules exist to prohibit this behaviour, there is currently no mechanism enforcing this ruling. An ideal solution would involve an apparatus that allows permitted vehicles to enter the carpark while preventing those without consent. Development of a working prototype demands a great deal of time and resources. A proof of concept is a more appropriate outcome and so the aim of development will hence be the production of a working model. The skills required to develop the model will be of moderate level, likely utilising both hardware and software concepts. Solutions will likely require multiple sensors – machine readable 'key-cards', distance sensors, etc – and rotating mechanisms such as servo motors. To implement a form of key-card I would have to develop an algorithm to recognise patterns. I have limited experience with such concepts which may pose as an issue during development. I am however comfortable with the other skills being demanded. Possible stakeholders include senior staff at Westlake Boys. Among the senior staff Mr Gordon, the associate principle, is the most accessible. With the model I will try to prove the value of a barrier arm in serving the Westlake staff community.
Model Mechanical Lifting Apparatus	My Grandad, Mark Mitchell owns an orchard. To pollinate his fruit trees, he uses bees. These bees produce honey which Mark harvest annually. To avoid bee stings he wears a bee suit. Unfortunately, Mark has found that bees still find a way into the suit and so regardless of the suit, he still gets stung. This occurs most often when Mark reaches his hands directly into the hive to extract the honey frames. The bees will often find a way to squirm their way into the suit resulting in Mark inevitably being stung. Mark would like a way to retrieve the frame from within the beehive without having to put his hands in. An ideal solution would involve the use of some kind of mechanical lifting apparatus to fetch the frame from within the hive, so Mark can remain at a safe distance from the beehive and not get stung. This tool has the potential to be helpful for a whole community of beekeepers or agriculturists alike. However, the outcome of development is not likely to be a commercial product due to manufacturing restrictions. Therefore, the solution will merely be a proof of concept model in the hopes of serving Mark with his own problem. The skills demanded in the development of this model are likely to be of a moderate level, with software and hardware having to work together efficiently. While Mark is likely to be my main stakeholder, exploring this issue gives access to a large community of horticulturists and agriculturalists. I will be able to find stakeholders from the local community who have issues with bee keeping that need a remedy. I hope to serve the local beekeepers (with Mark among them) with my solution.
Model Automatic Vertical Transport System	The in-development language block at Westlake Boys high school lacks a vertical transport system to move people up and down the multiple floors. The departments upstairs are therefore inaccessible to those who are unable to use stairs or large trays of equipment that cannot be carried. Thus, to solve this issue a vertical transport system must be installed. This mechanism would resemble a lift. Development of a working prototype demands a great deal of time and resources and money. I will have to produce a working model as a proof of concept. The Westlake staff involved in the development of the new block would be my ideal stakeholders. The construction of this apparatus demands a great deal of resources, funding and time. Instead, the outcome will have to be a model to prove the validity of the concept. Programming will not likely require complex concepts, rather the making of the model requires the mathematical capacity for conceptualising the logistics of the program (with regards to the way the lift responds to multiple button presses across multiple floors). Among the staff Mr Gordon, the associate principle, is the most accessible. Access to the construction team is also critical to the development of the model. With the model I will try to prove the value of a barrier arm in serving the Westlake staff community. With the model I will try to prove the value of a vertical transport system in serving those in the Westlake community who may require the aid of a lift mechanism. My solution is not urgent however as other model lift systems exist that the school can license.

Model Guided Projectile	Jonathan Gow is an aspiring entrepreneur attempting to breach into the rocket industry. "There is only a limited amount of ways to [send people to the heavens]" he says, and he would like to be on the frontier of this research. He requires a concept device that proves his expertise in astrophysics. He does not require a demonstration of his robotics capabilities. Thus, he would like me to construct a rocket according to his specifications that he can use as test dummy for his physics work. To abide by the New Zealand rocketry requirements the rocket cannot fly higher than 400ft and so he requests that the model behave as a guided projectile, with guidance enabled by the thrust vectoring. This solution is rather complex. The model will not likely demand a great deal of resources and funding. However, due to its complexity, development does demand a great deal of time and skill. The model serves to benefit Jonathan himself, though wider production may enable similar aspiring astrophysicists to test their physics. The solution is not urgent as it doesn't serve to benefit a substantial userbase. Jonathan is likely to be my main stakeholder with the solution. Since my skills are mainly programming based, it will be necessary to find stakeholders schooled in the engineering field to aid me with more complicated robotics concepts.

Decision Making

In order to make a final decision I must prioritise the relevant factors in terms of importance. Complexity and relative skill level seem the most significant, so I can immediately discount the possibility of developing the model guided projectile. Potential difficulties are likely to slow development and so demand a great deal of time. Access to stakeholders is also critical. While getting a hold of Mr Gordon for development of a model automatic vertical transport system should be relatively easy, getting a hold of the construction team will prove much harder. Also, designs for automatic vertical transport systems already exist so manufacturers already provide a cheaper, more effective solution than any solution I could develop. That leaves the model barrier arm and the mechanical lifting apparatus. Similar to the vertical transport system barrier arms already exist. It would prove inconsequential for me to develop an alternative since manufactures have a much greater capacity for research and development with multiple hired professionals and superior manufacturing equipment. I must therefore solve Mark's bee problem. The outcome is likely to resemble a lifting apparatus with honey frames being extracted by a lifting mechanism. The proposed solution is not likely to be a commercial product due to manufacturing restrictions. Therefore, the solution will merely be a proof of concept model in the hopes of serving Mark with his own problem. This means I will not encounter any monetization opportunities. Nevertheless, the tool still has the potential to be helpful for a whole community of beekeepers or agriculturists alike. I should be able to find stakeholders from the local community who share this or similar issues. I will be sure to reflect on ongoing stakeholder considerations and feedback to develop a solution fit to their needs. Compared to the other potential problems the mechanical lifting apparatus is a good middle ground in terms of complexity where my skills are likely proficient to solve the problem. This however is not to say that I will not be learning anything either. If I want the solution to work as effectively as possible, I will have to pursue further study. Development will likely involve a deal of object-oriented programming, which is a programming concept I have not had much opportunity to explore. This experience will help me improve my programming skills. Further learning opportunities are likely to occur in the modelling phase of development. Thus, my final decision is the development and construction of a mechanical lifting apparatus with the hopes of serving the local beekeepers (with Mark among them) with my solution. I hope to encounter many opportunities for learning in finding and constructing a solution to this problem.

Conceptual Statement

My Grandad, Mark Mitchell owns an orchard in Queenstown where he cultivates bees which aid in the fertilisation of his fruit trees. These bees produce honey in which Mark harvests annually.
When it comes time to harvest this honey there is the trouble of not wanting to be stung by the bees. While Mark does use a beekeeping suit it can be a challenge when reaching his hand directly into the hive. The bees will often find a way to squirm their way into the suit resulting in Mark inevitably being stung.
The problem is that Mark would like a way to retrieve the frame from within the beehive without having to put his hands in.
An ideal solution would involve Mark using some kind of tool to fetch the frame from within the hive, so Mark can remain at a safe distance from the beehive and not get stung. It must be durable – but not too heavy – and not be easily damaged from an abundance of bees crawling all over it.

Attributes

- Provide up to half a metre of distance from the hive
- Withstand a plethora of bees crawling over it
- Light and portable
- Durable
- Can lift an entire frame
- Easy to hold
- Mobile – doesn't get caught on things
- Minimal maintenance
- Suitable outdoors

Stakeholders

Stakeholder	Interest	Information
Mark Mitchell	**Main Stakeholder -** Needs a solution to his bee problem.	Mark, my main stakeholder, would like a tool that aids in the extracting of the honey frames from his hives. He would like to be able to remain a distance (at least half a metre) from the hive to avoid being stung. He would like the tool to have a long life with minimal maintenance and so shouldn't need to be replaced.
Mr Enefer	**Technical advisor -** As an advisor, he is there to give advice in the development of my project. Also has a vested interest in the success of my project as student success may see his rise in the school staff hierarchy.	Mr Enefer requests that the model is durable, so it doesn't collapse under its own weight and not easily damaged. He also suggests that I consider, if the tool is to be hand-held, the ease and comfort of a potential handle. Mr Enefer is available to me for further advice as I develop my model.
Harry Simpson	**An aspiring designer -** Harry acts as a representative of the designer community. A reference as a stakeholder aids him in his studies.	Harry suggests that the model should be aesthetically pleasing. To appear suitable for the model's purpose, he recommends a red and black colourway. Also, to avoid damage, he suggests that electrical components and wiring should be screwed/strapped down tightly.
Nick Hunt	**A schooled agriculturist -** Nick acts as a representative of the agriculturist community. He would like to improve the efficiency of harvesting his honey frames.	Nick requests that the model has some form of automation. If possible, he would like it to be automated from end to end with minimal human action. He recognises the importance of portability. As such, he suggests that the model be light and mobile.
Mr O'Brien	**Coding expert -** Also a technical advisor, so he is there to give advice in the development of my project.	"[Digital technology] is the way of the future." Mr O'Brien implies here that I should embrace the use of technologies in the development of the model. As a coding expert, Mr O'Brien can aid me in the logistics of the programming for my project.

Stakeholder Interview

What would you like to see from this device?

Mark: *"I'd like a device that can help me remove the frames from my hives, without getting stung."*

Nick: *"I'd like to see if you can help me streamline my honey gathering processes with some kind of gadget to help me more efficiently extract my honey frames."*

Out of the attributes discussed, which are essential, and which would you be willing to sacrifice, if necessary?

Mark: *"Mobility and ease of use are most important. Minimal maintenance also. I'm hardly likely to collect my honey while the weather's not so good, so it doesn't need to be designed to withstand 'the elements'."*

Nick: *"Efficiency is most important. Mobility also. I don't mind what the thing looks like, as long as it works."*

Any material requirements?

Mark: *"Strong, yet light weight. Something like aluminium or MDF, or maybe acrylic."*

Nick: *"Light and durable; a polymer. Perhaps acrylic."*

What are some potential hazards that I should be aware of?

Mark: *"Some of the bees remain on the frames while I remove them. I wouldn't want them squashed."*

Nick: *"I'd like to avoid hurting my bees."*

Do you think sustainability is important? If so, what sustainable aspects would you like to see implemented?

Mark: *"Sure. Maybe try to make the tool recyclable."*

Nick: *"Yeah, sustainability's important. During your design process you could try cut down on waste by prototyping on more environmentally materials. Also, you could minimise waste by making the most out of available material."*

I'd like to know what you think is most important, safety, functionality or sustainability?

Mark: *"First and foremost I'd like a tool that works. It'd be good to sure its safe to use though."*

Nick: *"Safety comes first, but don't let it get in the way of functionality. Find a way to make a safe product that works!"*

Post research (below): Which conceptual design (from concept research) do you like the most, and why?

Mark: *"Definitely the arm-like device. Attaching the other one would involve me getting too close to the hive."*

Nick: *"The claw device. Attaching the sluice-gate-like device would be so inefficient."*

Which product (from product research) would you most like the device to resemble, and why?

Mark: *"The loading arm. Seems the most practical. I don't need the all the fancy tech from the others."*

Nick: *"The printing device looks to be the most flexible which I like, however the side-loading arm is probably more realistic."*

Specification

1. **Safety**

1.1 The tool should not pose any hazards to the bees or user.
1.2 Any moving parts should be covered to prevent bees from getting squashed by these.
1.3 There should be no sharp edges.
1.4 I hope to show my testing practices are ethical and that my trialling procedures are culturally appropriate.
1.5 In testing, I will use empty hives to minimise the risk imposed by mechanical parts on the bees.
1.6 I will implement an easy method of terminating the mechanical procedures in case of an emergency.

2. **Properties**

2.1 My stakeholders want the arm to provide the holder up to half a metre of distance from the hive.
2.2 The tool must have enough grip on the frame so is not dropped and the honey is not contaminated.
2.3 It must also be able to successfully lift the frames without collapsing under its weight.
2.4 Must be supplied with enough power to extract all the frames without running out.
2.5 Cables must feed neatly down each the tool, so they do not get damaged.
2.6 Cables that provide power and signal to the components.
2.7 A sturdy handle.

3. **Materials**

3.1 The materials used would either aluminium, MDF or acrylic.
3.2 Materials must be that they and durable, strong, long lasting.
3.3 They must be able to support the weight of the tool as well as a honey frame without collapsing.
3.4 More socially acceptable materials such as wood are preferable, where possible.
3.5 I will try to reuse sheets of aluminium, MDF and acrylic to improve sustainability and social acceptability.

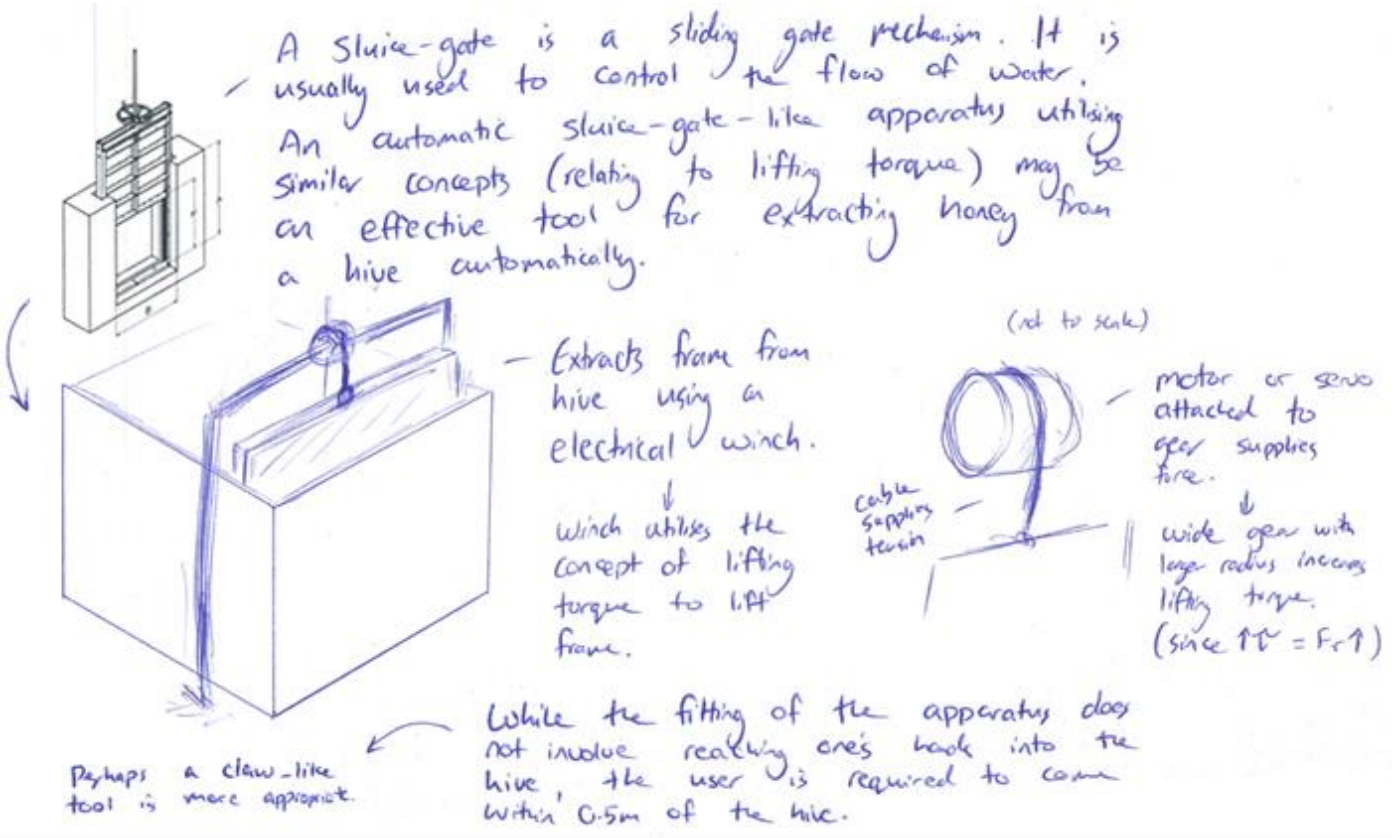
4. Sustainability

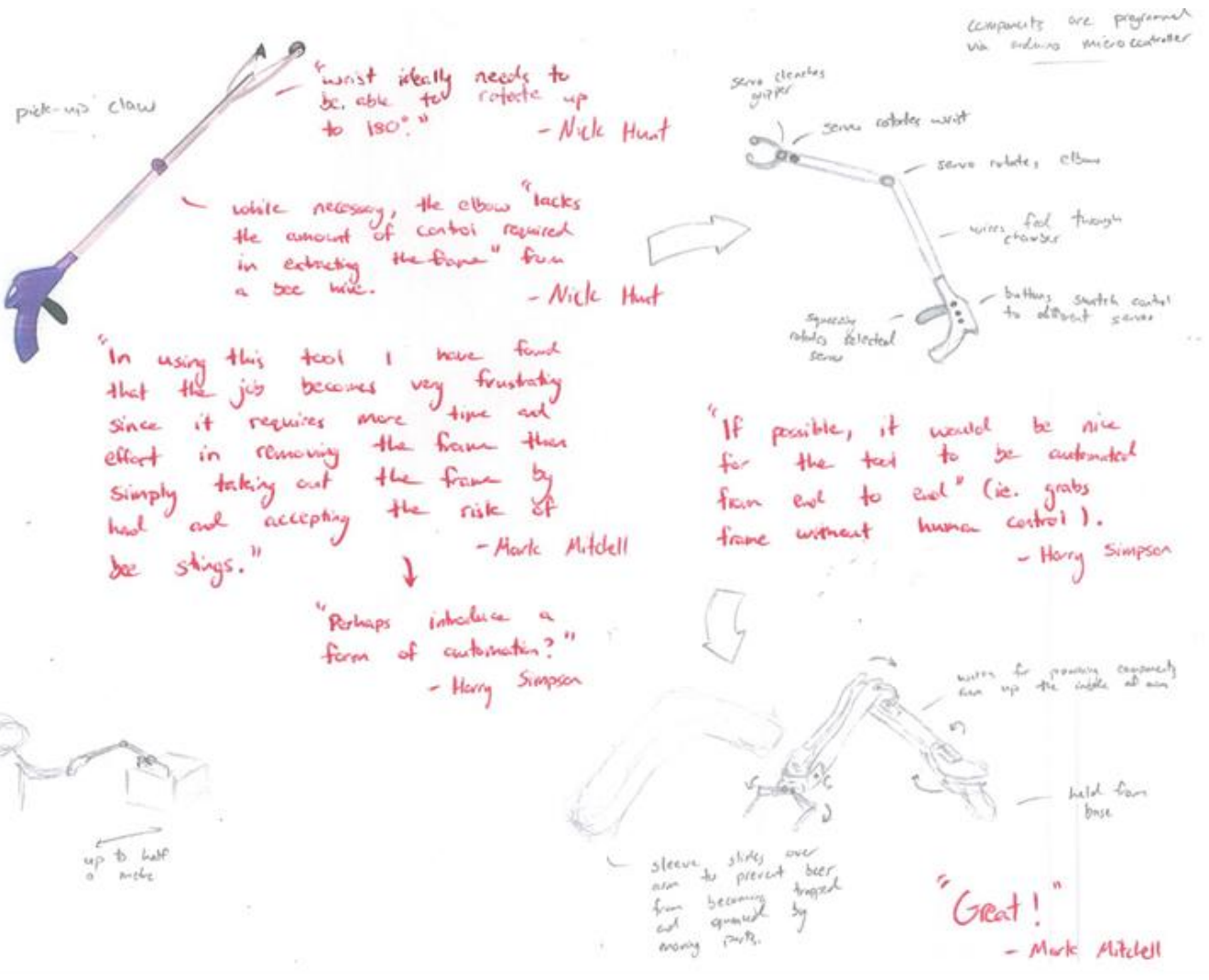
4.1 A replaceable cover to reduce wear of more central components will reduce **maintenance**.
4.2 The amount of material used in manufacturing can be **reduced** by maximising the amount of material cut.
4.3 The product will also be made of modular parts so can be **repaired**, extending its life.
4.4 I will try to **reuse** materials. E.g. electrical components can be extracted and used in other products.
4.5 I'll try designing the product as to be able to be **recycled** ethically after end of life.
4.6 The parts that are unable to be reused or recycled must be able to be (ultimately) disposed of safely.
4.7 These conditions are required so the solution has a long life and environmental impact is minimal.

Location Research

The tool will be used to extract honey frames from these hives. The hives are outdoors. It is unlikely however that Mark will be extracting frames in wet and rainy conditions. Frames may be wet after rain has passed, so the tool must work still work during these times. It must not run out of power until all the frames have been harvested. Hot temperatures may cause issues however bees hives are unlikely to be situated in colder climates. Therefore, electronic components must not overheat. A heat sink or another heat dissipating device may be necessary. Since there are so many hives to extract frames from, the tool must also be power efficient. Potential damage or deterioration due to stress or different environmental conditions must be reduced.

Concept Research





Product Research

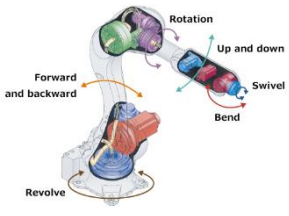
Crane vending machine. The crane vending machine utilises a claw-like manipulator to extract prizes from within the vending machine. The claw acts like a lever where the movement of the smaller shafts attached to the base of the device enables the movement of the larger shafts. However, these tricky machines come with a catch. The claw isn't designed to successfully grab the given prize. They use materials with low friction to ensure the claw is ineffective, so the user is driven to keep spending - much like gambling. The lever mechanism still works as a gripping mechanism in practice so I can employ similar techniques. I will have to avoid the slippery materials used by this design as unlike the crane vending machine, I intend my solution to work effectively.

Crane. The crane is a large machine. It is used to lift or lower large materials and to move them horizontally. They are commonly employed in large scale construction projects as the pully mechanism is incredibly effective at reducing the force needed to do work on the load (e.g. Lifting material for construction). W = Fd so by increasing the length of cable, force must decrease to maintain constant work done. These concepts will likely prove useful during the developing my solution.

3D printing device. The arm-like device used in some 3D printing mechanisms are incredibly flexible. It is highly automated and capable of movement along 3-axis. The flexibility granted by the device is necessary for the construction of 3D models. The device contains many servos to enable effective rotating of joints. A high degree of precision is needed so some joints have two servos to provide enough rotating and holding torque. This technique will prove essential for my solution where one servo won't provide enough torque – like at the base of the device.

Automatic side-loading apparatus. The side-loading apparatus is a device often used in rubbish collection. These devices must have incredibly high holding torque, so they don't collapse under the weight of a full rubbish bin. This is often achieved by using a ratchet-like mechanism. This way the apparatus has a failsafe and so is not over relying on the motors.

Industrial-grade manufacturing arm. An industrial grade manufacturing arm is an industrial robot used for manufacturing. Typical applications include welding and assembly. These tasks demand high endurance, speed and precision. The articulated robot uses electrically powered rotary joints ranging from two-jointed all the way to ten-jointed structures. The number of joints is proportional to the rotational complexity of the task. I will likely use the ergonomics of a human arm and to do so I will need to mimic the joint structures.



Prioritised Specifications

Safety (1)
The hazards imposed by the arm on the bees are the moving parts and the potential for bees to get trapped and squashed by these movements. Also, it will likely need to be cleaned after each use to prevent the build-up of gunk which would find its way into the honey if not dealt with. To minimise these, I will implement a sleeve which will slide over the tool. The sleeve will therefore not only protect the bees from going where they shouldn't, it will serve as protection for the tool to reduce the likelihood of damaged (since all the components will be enclosed by the sleeve). I will have to ensure that there are no sharp edges on the tool that may be of potential hazard to humans or lead to the tearing of the sleeve. I will try to minimise the risk of the user of the product by meeting the health and safety standards of the Consumer Protection Act (see https://www.consumerprotection.govt.nz/general-help/consumer-laws/consumer-guarantees-act/).

Properties (2)
To ensure the maximum ease of use and lowest risk of bee stings, the tool will be automated in its attempt to extract the frame from the beehive. It will have a combination of servos moving simultaneously like an arm to extend to the hive and grab the frame. My stakeholders also request that the arm would allow for rotation of a shoulder, elbow and wrist as well as a horizontal twist the rotates the rest of the arm parallel with the base.
The most stress will be put on the shoulder, so I'll use 2 servos for rotation of the shoulder. It'll need another servo at the elbow and yet another at the wrist. To grab the object, the gripper will also need a servo as well as a button to tell the servo to stop gripping and not crush the honeycomb once its squeezed hard enough. To enable rotation parallel to the ground I will have to use a stepper motor which provides the higher holding torque required in moving the whole arm. This means that a total of 5 servos, 1 button and 1 stepper motor will be used. To control these components, I will use an Arduino Uno micro-controller and develop the code required to solve the problem. It may also prove necessary to use a sensor (like an ultrasonic sensor) to measure distance so the servos can adjust according to how far the holder is standing from the hive.

Dimensions (3)
My stakeholders want the tool to provide the holder up to half a metre of distance from the hive. Realistically each limb can only be up to 200mm in length before the arm breaks under its own weight. This gives a maximum reach of 40cm.
The height of a servo is 30mm. This means the interior of the arm must be of a width greater than this in order to contain the servos. Ideally the tool will be 40mm in width.
The width of the lip of a honey frame can be up to 50mm. To allow for a moderate margin of error for when the 'arm' is attempting to grab the frame; the gripper should be able to widen more than twice this. Therefore, the gripper should open to at least 120mm.
The tool is held from a handle at the base. The 'arm' should be able to turn horizontally and so should not be attached directly to the handle but instead elevated above it with the output shaft of the stepper motor attached to the handle. The base of the arm should therefore be 80mm x 80mm with the top of the handle being the same before it curves down into the rod-like part of the handle where it is held.

Materials (4)
I require that the materials be durable, strong, long lasting, stable and able to support the weight of the rest of the arm as well as a honey frame. They need to be sustainable and not easily damaged or quickly deteriorated due to different environmental conditions. Simply, the materials need to provide the tool with longevity. Due to the nature of the arm's purpose however, it will need to be cleaned after each use to prevent the build-up of gunk which would find its way into the honey if not dealt with. The sleeve discussed above will have to be able to be removed and washed as the design requirements of the tool will not allow for easy maintenance of materials. Since my stakeholders request that the arm is aesthetically pleasing, I will try and be consistent in the use of a red and black colourway with these materials.
To meet these requirements, I have decided to use aluminium, MDF and acrylic. These are appropriate for the function however have a greater environmental impact than more socially acceptable materials such as wood, but my choices were limited to these three due to cost and availability restrictions (though a wooden handle may work). I will try to reuse sheets of aluminium, MDF and acrylic to improve sustainability and social acceptability.

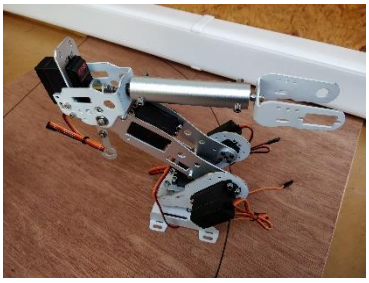
Sustainability (5)
It is important that I abide by the consumer protection act and meet health and safety standards. In testing I will use empty hives to minimise risk imposed by mechanical parts on the bees. Only when I'm confident that the model is suitable to be used in the presence of bees without harm will I move onto that stage. Materials will be sourced locally so environmental impact is almost negligible. I will still reduce waste material by maximising the amount of material used from the cut of material. The product will also be made of modular parts so can be maintained and repaired, extending the product life. I will try to optimise for reusability and recyclability, the components cannot be must be able to be (ultimately) disposed of safely. While I will aspire for longevity in my product, since the outcome will be a model, the determination of life cycle is unattainable since the outcome will merely be a proof of concept (rather than final product). *The tool is not threatening to replace workers because the concept is to improve their safety rather than automate their task.*

Revised Conceptual Statement

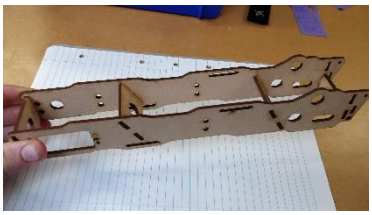
Bee stings are an issue. Regardless of the beekeeping suit, beekeepers still often get stung. My Grandad, Mark Mitchell, faces this issue. He owns an orchard in Queenstown where he cultivates bees which aid in the fertilisation of his fruit trees. These bees produce honey in which Mark harvests annually.
The issue arises when it comes time to harvest this honey. While Mark does use a beekeeping suit the main challenge comes when reaching his hand directly into the hive. The bees will often find a way to squirm their way into the suit where they panic, and Mark is inevitably stung. Mark would like a way to retrieve the frame from within the beehive without having to put his hands into the hive.
An ideal solution would involve Mark using a tool to fetch the frame from within the hive, so Mark can remain at a safe distance from the beehive and not get stung. As a stakeholder, Mark has asked for a solution to be found where he can stand no closer than half a metre from the hive when fetching the frame. The solution must be durable – but not too heavy so it doesn't collapse or snap under its own weight or that of the frame – and not be easily damaged from an abundance of bees crawling all over it.
My stakeholders request that the model solution be light and mobile. If hand-held the tool should be comfortable and easy to use. They would like to have a high degree of automation with minimal human action involved in working the tool. They'd like the tool to appear appropriate for the model's purpose (perhaps a red and black colourway?) and suggest that any electrical components or wiring should be screwed/strapped down tightly. The coding expert among my stakeholders, Mr O'Brien, imparted a great piece of wisdom upon myself. His coding expertise enables him to recognise the importance of the field. He proposes that I embrace the use of technologies in the development of the model, that digital technology is the "way of the future". As such, I will be sure to utilise the offerings of the technological domain in development.
In developing my model, I will also consider the safety of model and any potential hazards in may impose on the safety of the bees or the user. I must consider the properties and dimensions discussed to develop a model that fits with my stakeholders' requests. The outcome must promote environmental friendliness and sustainability. Potential damage or deterioration due to stress or different environmental conditions must be reduced. The amount of waste material must also be reduced while materials and components should be optimised for maintenance and repairability, reusability and recyclability. I hope to show my testing practices are ethical and that my trialling procedures are culturally appropriate and will strive for social acceptability.

Modelling

I am developing a model in attempt to verify whether the proposed solution to the conceptual statement is realistically achievable. The result will not be representative of a prototype, let alone a final working product.



This initial model was made of alumium. Its bearing construction works well, allowing for smooth rotation of the base. A servo is used at the base. I will replace this with a stepper motor which allows for greater precision and has a higher holding torque. The dual servo shoulder setup provides decent torque however the aluminium is too heavy – I will likely continue this design in future iterations however will replace the material. The design lacks vertical wrist movement and wires are unable to be threaded so may get caught while the arm is in use. It also lacks a gripper – I'll work on this for future iterations.

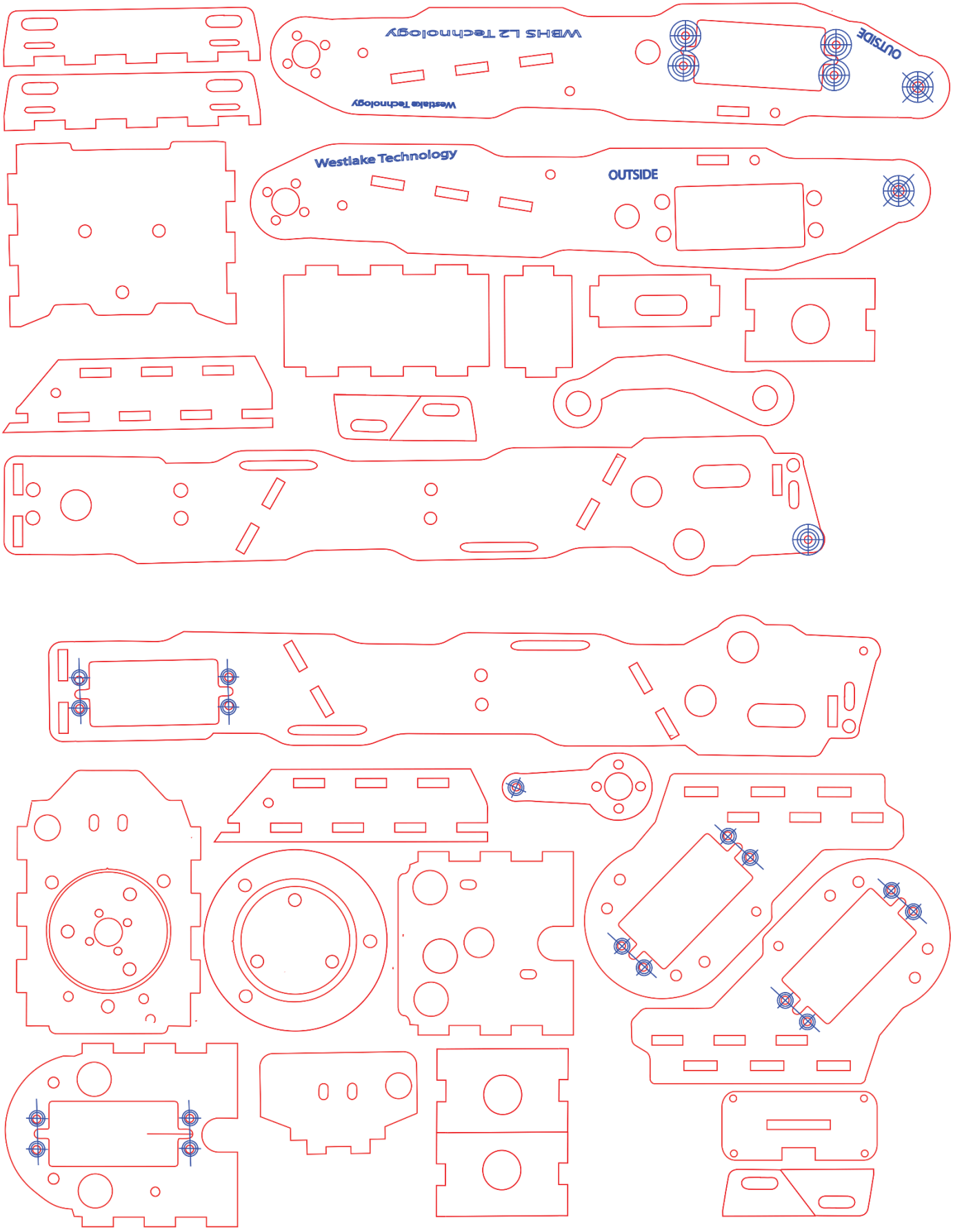


This segment will replace the forearm of the previous design. Unlike the previous, it enables a gripper to be rotated vertcially much like a human wrist. Its also has an elbow mechanism which works much like a human elbow. Supports are needed to prevent the two sides from buckling. The holes on either side will be where these get screwed into. Holes in the bridges between each side allow wire to be threaded to prevent them from getting caught while the arm is being used.

This design combines the upper arm of the first design and the improved forearm. I used a tap for the holes to create a threaded path in the acryilic for screws. A servo-bearing mechanism at the elbow enables effective rotation. However, the linkage put under a lot of force by the limbs during trialling and so it broke. I will have to re-cut the piece from aluminium (which is much stronger) using a plasma cutter. Update: Tapping the new piece was impossible using the available equipment so had to be ditched – I will have to thicken the acrylic piece instead.

Material considerations during modelling:

Material	**Advantages**	**Disadvantages**
Aluminium	Very strong.	Much too heavy – servos are often unable to lift each arm segment.
MDF	Low density so servos have no trouble lifting the load. Lightness granted by low density means the arm is easy to transport.	Water causes the MDF to swell so it is unlikely to endure the elements of the outdoors. MDF twists and buckles under loads – increasing thickness would prevent this, however it would nullify the advantage of MDF's low density.
Acrylic	Much denser than MDF so is much less likely to twist and buckle and is suitable in most outdoor environments.	Acrylic is more brittle than MDF so is likely to shatter when dropped onto a hard surface – if dropped on earth, the ground should be able to absorb the arm's impulse hence this shouldn't be much of an issue.



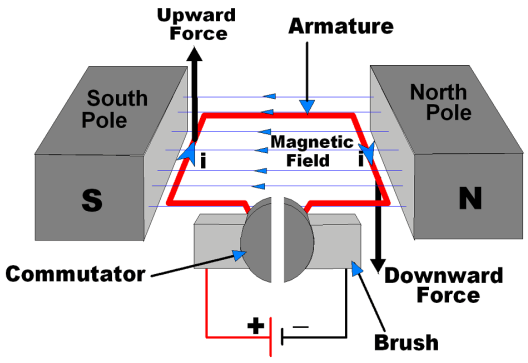
- Schematic is cut out of acrylic using a laser cutter.
- Radials indicates holes that require 3 mm and 4 mm tapping respectively.
- Holes which oppose screw holes allow a screwdriver to be threaded through the constructed chamber to tighten screws.
- Servo horns are screwed into the acrylic whereby servo heads are screwed into them.
- Pieces are glued together with acrylic glue.

*Schematic provided by technical advisor

Circuitry

A **Servo** is made up of 4 components: DC Motor, Control Circuit, Gearbox & Potentiometer.

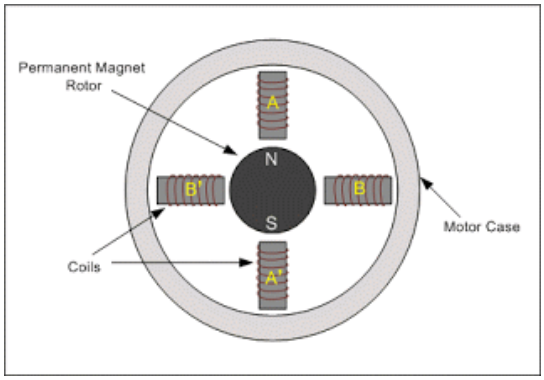
The DC Motor utilises the concept of electromagnetic force on a current carrying wire to create motion:



Current flows around the coil between the permanent magnet creating an electromagnetic force as it interacts with the magnet, causing the coil (Armature) to turn. A split commutator ensures that current always flows in the same direction around the coil. It reverses the connection to the power supply every 180° of rotation.

A digital signal is sent by the Arduino to the Control Circuit which tells the DC Motor to activate. The Gearbox and Potentiator then work together to ensure that the DC Motor is turned by the amount of degrees as specified by the Control Circuit.

A **stepper motor** is a type of modern electric motor that allows for a high degree of accuracy. The concept of electromagnetic force on a current carrying wire is utilised to drive the motion. However, instead of rotating continuously like a DC motor, it rotates in the form of pulses or discrete steps (hence 'stepper'):



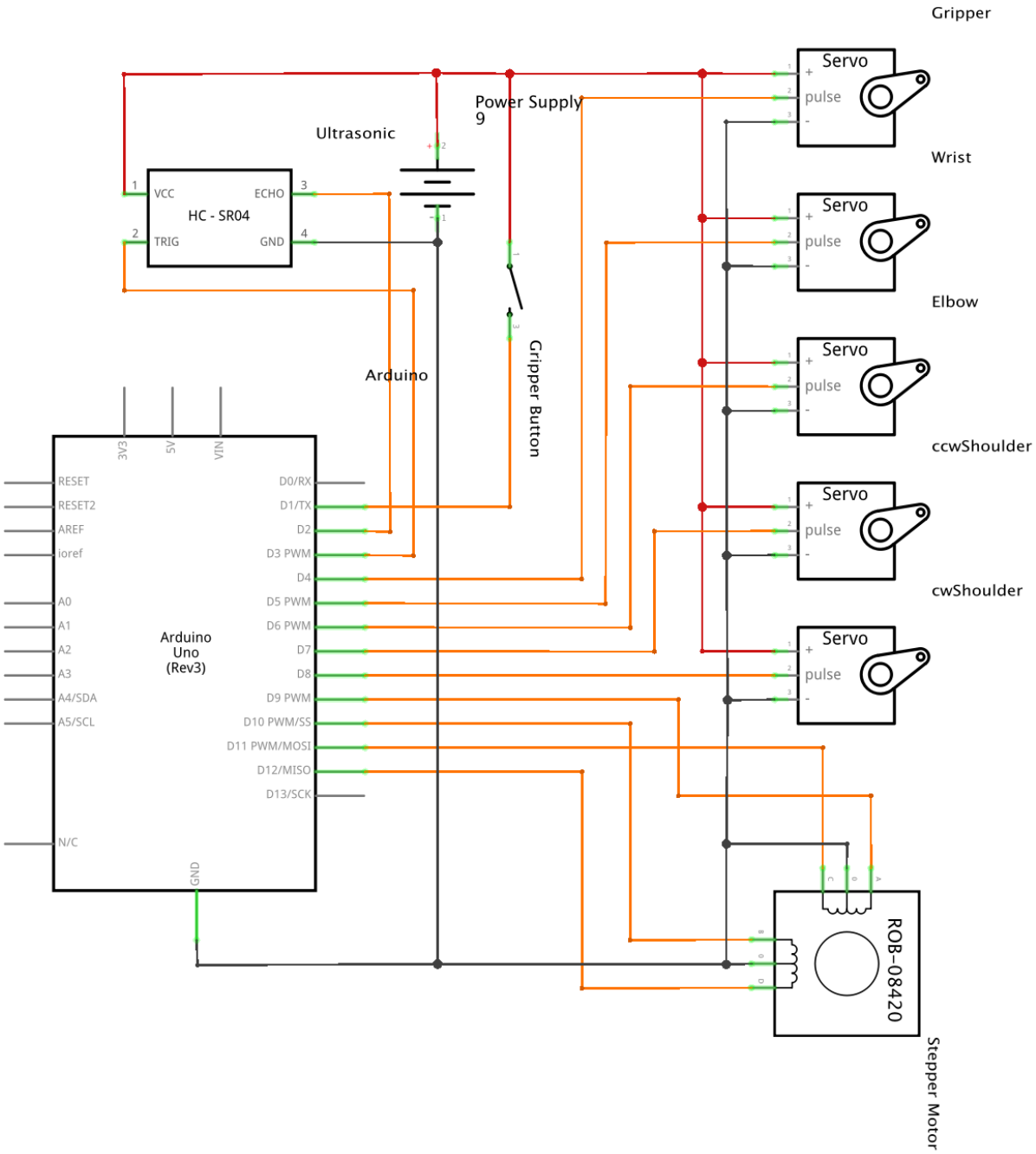
The rotor (permanent magnet) interacts with the magnetic field produced by the coils on the outside (the stator) which remain static, creating motion.

Each coil is activated by sending digital signals to let current through them in a particular order (A then B then A' then B' for clockwise) and so when the stator becomes magnetised the permanent magnet experiences an electromagnetic pulse causing propulsion to the motor through rotation of said magnet as the south pole is attracted to the activated coil.

Electrical signals sent by the Arduino Uno microcontroller are used to control both components. Standard metal geared servos allow 180° of rotation whereas stepper motors allow a full 360° of rotation. Research suggests that servo motors are more appropriate for rotating limbs since the joints do not require the greater holding torque and accuracy granted by the stepper motor and the servos are much cheaper. The stepper motor will need to be used at the base since these advantages are essential for rotation of this axis.

The **Arduino Uno** is a micro-controller which tells connected components how to behave according to the program being executed. It is equipped with sets of digital and analogue input/output (I/O) pins that are interfaced with to control connected components. These pins send discrete digital signals in one of two states: on or off (1 or 0). The device comes equipped with a compiler which converts source code into these binary signals called machine code which allows the controller to be interfaced with higher level languages. I will be using an Arduino derivative of C to interface which the device and connected components. It also has a set of 5V and GND pins to power these connected components.

The **ultrasonic sensor** is a device that utilises sonar technology to approximate distance. It transmits ultrasonic sound signals and records the time it takes for each pulse to return to the receiver. Using the speed of these waves (0.0343 cm/µs), distance can be calculated using the velocity formula (v = d/t) to calculate how far the ultrasonic sound waves must have travelled before they returned to the sensor. The value derived will have to be halved to account for the distance the sound waves have to travel to return to the sensor.



Power is transmitted directly (hence 'direct current' or DC) from the positive terminal (red) to the negative terminal (black) through the components in a parallel circuit. Voltage is therefore the same among components at the sacrifice of current, which is split among the components. The power supply is equalised with the ground to compensate for the current sent through the components by the digital pins.

Notes

12 signal pins
9 +ve, 9 -ve
1 resistor

Stepper 4× signal, 2× power

Servos (2× power, 1× signal) 5×

Sensor 2× power, 2× signal

Button 1× signal, 2× power, 10k resistor to GND

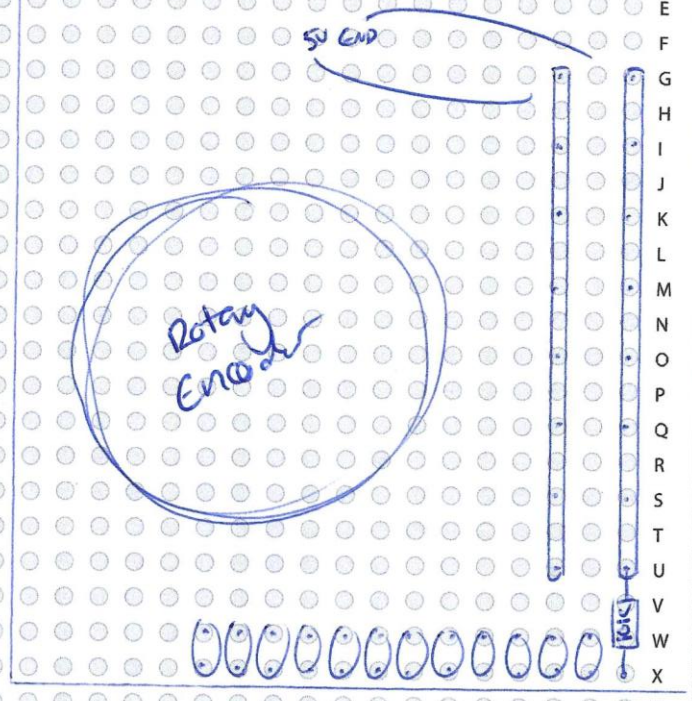
Perfboard Layout Planning Sheet

Project	
Designed by	
Version	
Date	

Data is transmitted along the signal wires (orange) from the Arduino board. This information tells the components how to behave according to code I will develop.

Powering the components in the circuit proved problematic. Initially I used the Arduino board's 5V power supply but quickly discovered that it could only power up to three components. Instead I used a 9V external power supply to power the components. This was effective for powering up to four components, however any more proved faulty. To discover any issues the circuit may have been encountering I probed it with a multi-meter. This showed that the current through the circuit was far too low. P = VI. Since the components were connected in parallel current is split among these components. Insufficient current was being supplied to the components which is why the circuit was failing. Using multiple 1.5V AA batteries enabled me to increase the current throughout the circuit while maintaining the same 9V. Hence, I connected the circuit to a 9V set of AA batteries (total of six).

Unfortunately, this solution didn't last for long. Further troubleshooting of the circuit showed that the voltage across the terminal was rapidly decreasing. The internal resistance of the terminal must have been increasing very quickly as batteries were drained. This is likely a consequence of the equalisation of the external power supply with the Arduino ground (necessary because of the additional current send by the Arduino along the signal pins to the components). I could either put a resistor along this connection or replace the batteries more often. Since I have already created the perfboard, the easiest solution is to simply replace the batteries more often.



Pins

button pin 1

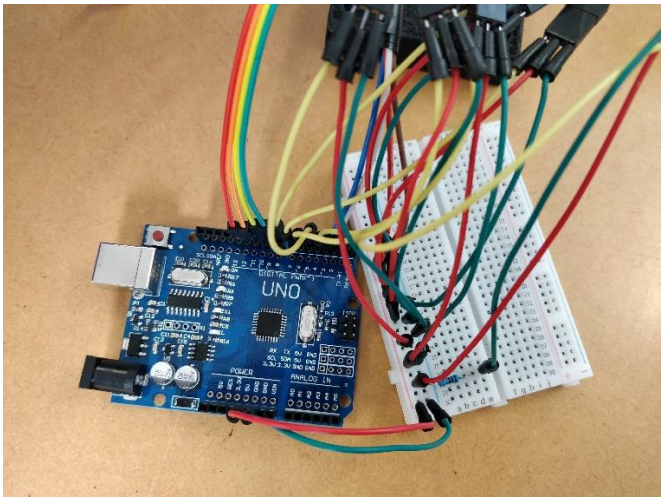
echo pin 2

trig pin 3

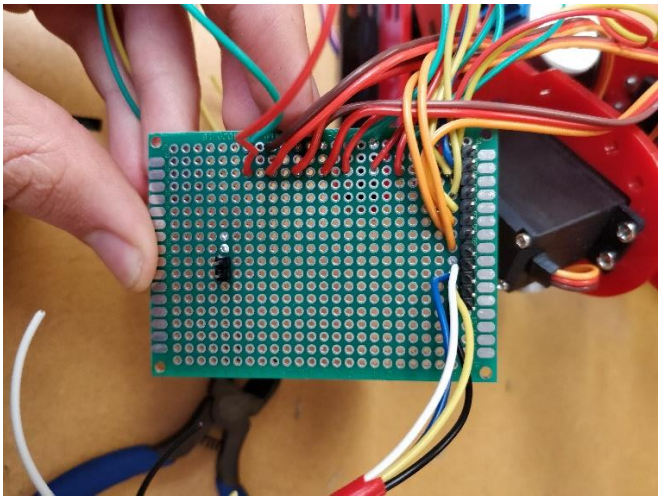
servos 4-8

stepper 9-12

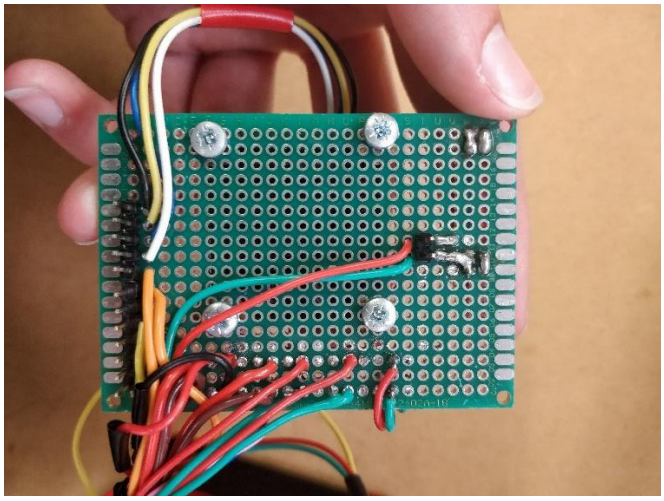
Perfboard



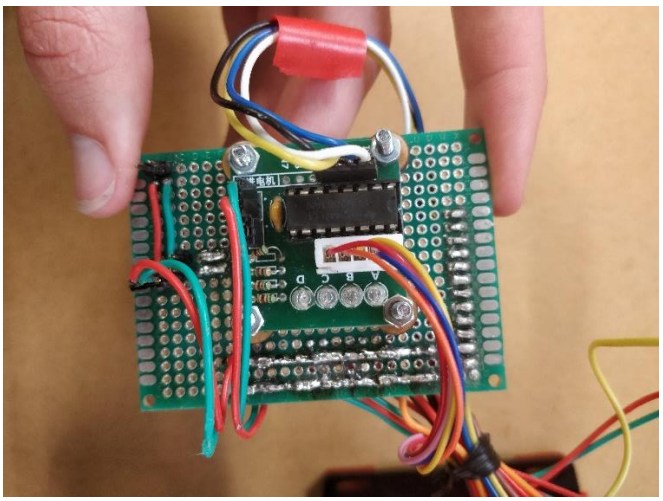
Up till now I have been using a breadboard to test components. They have been connected in parallel to the positive (red) and negative (green) terminal rails while signal (yellow) pins have been directly connected to the Arduino board. The breadboard is only temporary and must be replaced with a more permanent cicuit board before proper use. A perfboard serves as a more permanent circuit comared to the breadboard as components are hard wired which prevents them from falling out during use.



The first iteraion of the perfboard has all components soldered directly into the board. The male pins soldered onto the perfboard line up with the female pins of the Arduino board so that it behaves like a shield. This way components are centralised and are much better organised. Signal pins are soldered vertically on the right while voltage and ground pins are soldered horizontally along the top and are connected in parallel. An external power supply is used to power these components. The two pins on the left power the Arduino board.

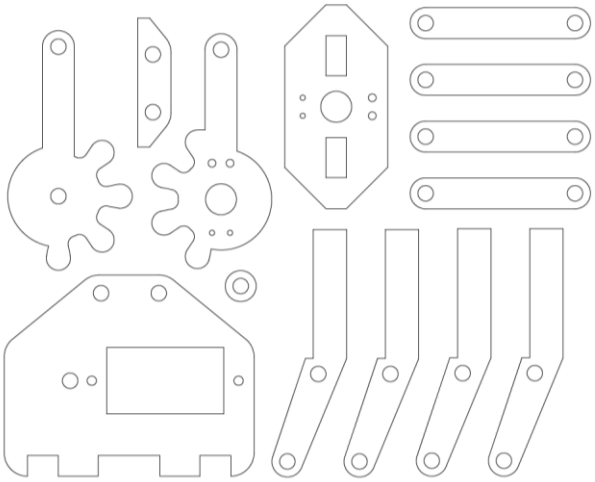


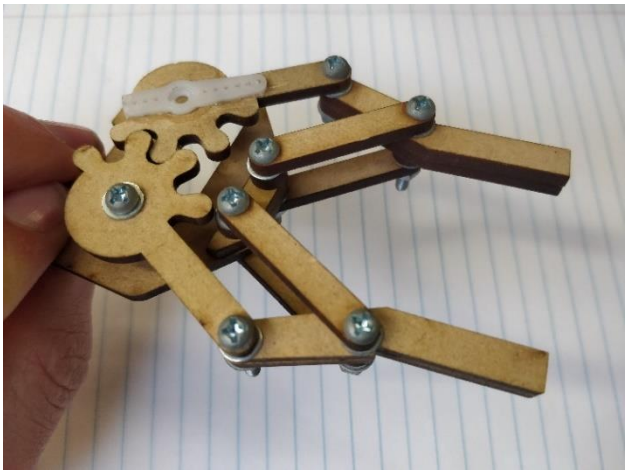
The additional solder near the terminal pins enables the negative terminal to be equalised with the Arduino board's ground if an external power supply is connected. I also discovered that the ultrasonic sensor can only operate at 5V while the external powers supply is 9V (to power all the servos). This means that the ultrasonic sensor must be powered directly from the Arduino board's 5V pins rather than the external power supply so the new design reflects this change.



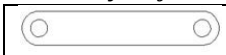
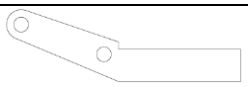
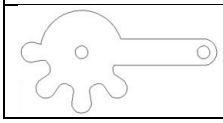
The completed circuit consists of a set of pins that allows alternation between the Arduio board's power supply and an external power supply. The button used in prior designs has since been removed. To furthur centralise components I drilled holes directly into the perfboard and screwed on the stepper motor rotary encoder (a device that enables full control of rotation and speed of the stepper motor). A twisty-tie holds wires together in a cable to prevent them from getting in the way of the pins ot getting caught by the moving parts.

Gripper

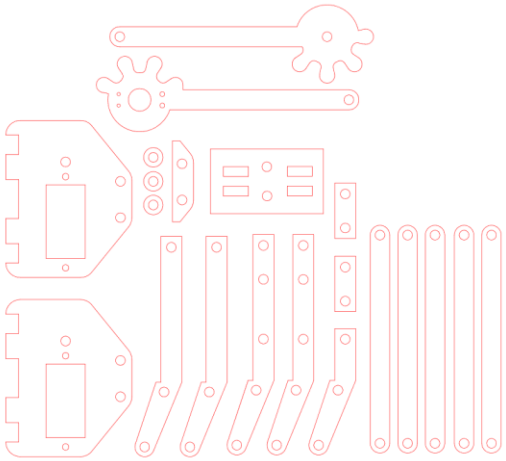


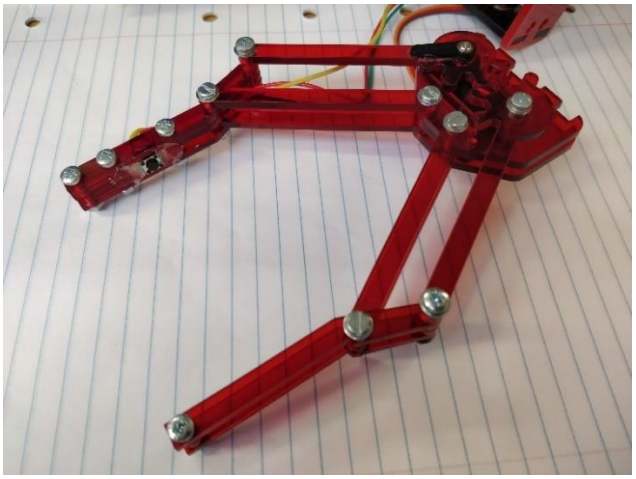


Necessary adjustments:

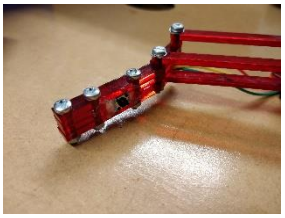
	Increase to 64mm in length.		Increase to 45mm in length.
	Increase to 64mm in length.		To fit SG90 servo, adjust width to 23mm

+ Modify one of the fingers to accommodate for the button



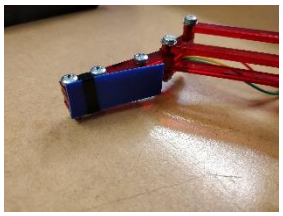


To house the button, I created an enclosure out of the existing finger design. The button required thicker fingers so to accommodate for this I used spacers between the top and bottom pieces. I also sandwiched the servo between two bodies to hold it more firmly in place and edited the attachment to accommodate for the additions.

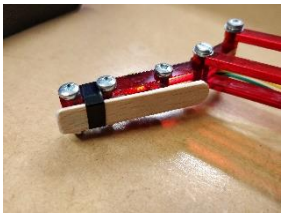


Button surface area is too small so button often doesn't get pressed

Ice-cream lid increases surface area however pivot proves ineffective



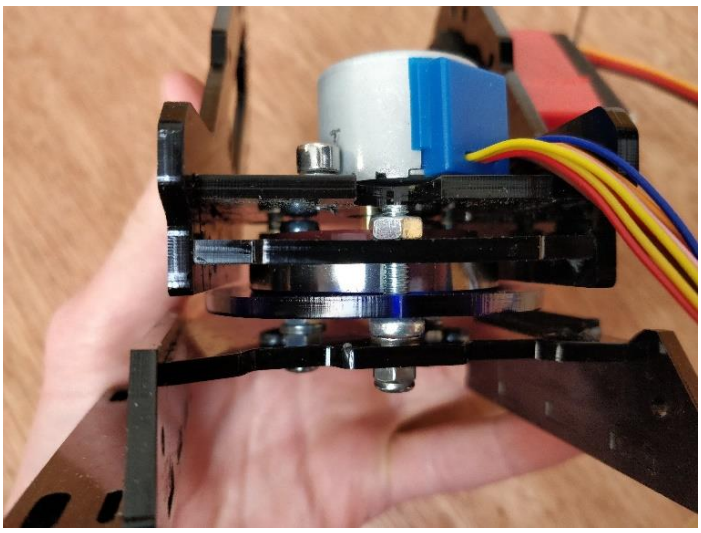
adjusting location of pivot proves effective however lid is too flimsy



Popsicle stick works!

Update: After extensive testing I found that the button resulted in jitter of the gripper and remained unreliable, so it has since been removed. Changes are minimal with the finger with the button simply being been replaced with a mirror image of the other.

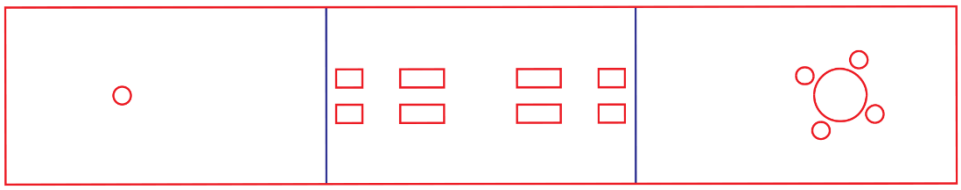
Base Module

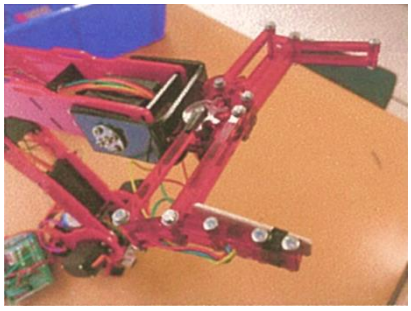


- Bearing (40mm)
- 2x acrylic spacers
- 60 mm acrylic disk – bearing mount
- Stepper mounting plate
- Stepper motor
- Acrylic chassis
- 3x 3 mm * 15 mm screws through chassis
- 3x 3 mm * 25 mm screws through bearing
- 2x 4 mm * 5 mm hex-head screws to attach stepper
- 9x 3 mm nuts
- 6x 3 mm spacers

*Screw heads were filed to prevent them from catching.
*Holes for hex-head screws were tapped.

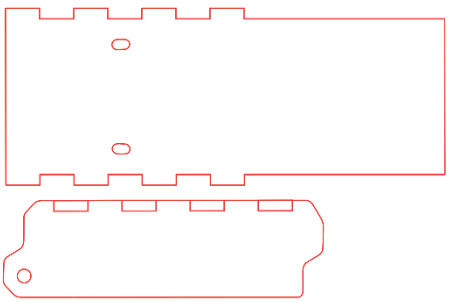
Gripper Attachment Attaches gripper to wrist:

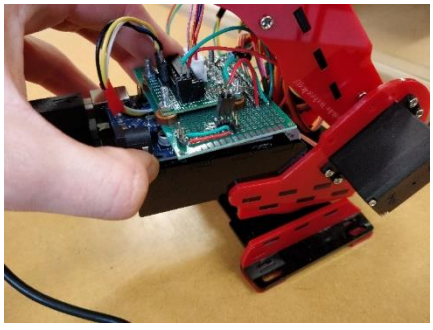




Base Extension

When rotating the base, the Arduino-perfboard shield is dragged across the table which results in jitter and has the potential of damaging the module. Elevating the Arduino-perfboard shield off the ground prevents this, hence the addition of this extension:

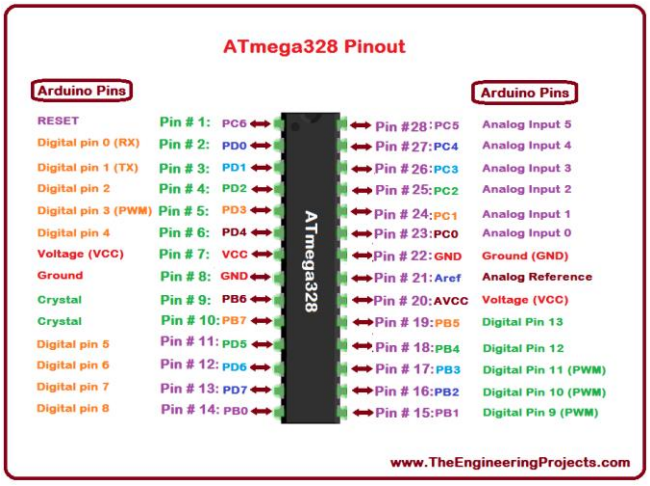




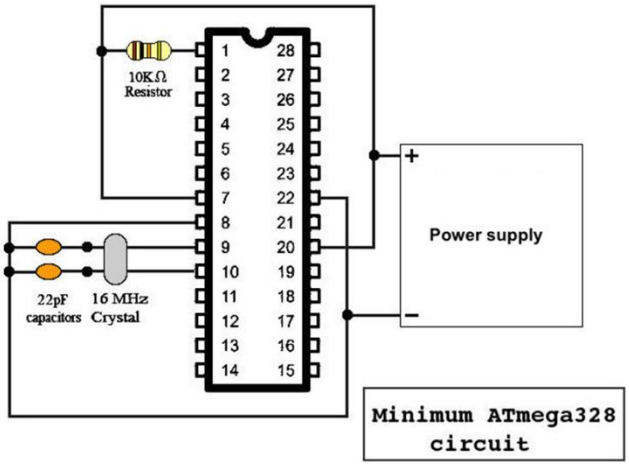
**Mark:* *"I'm worried about the durability of the circuit that you've made. A printed circuit might be more reliable."*
**Nick:* *"I like Mark's idea; a printed circuit could also be easily mass produced."*

- I'll see to implementing an embedded printed circuit in my model.

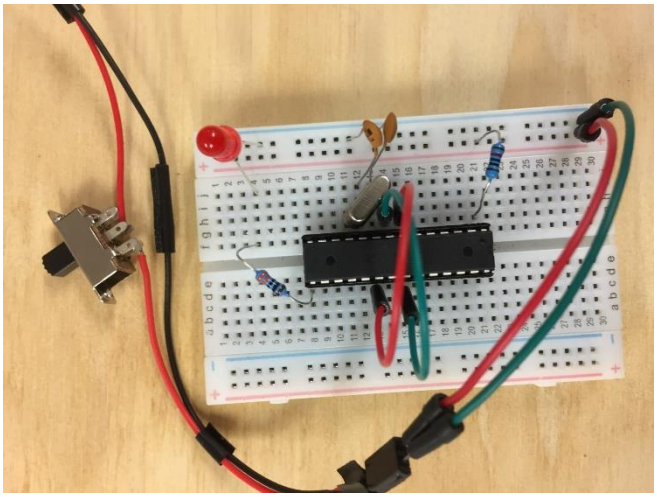
Constructing an Embedded System



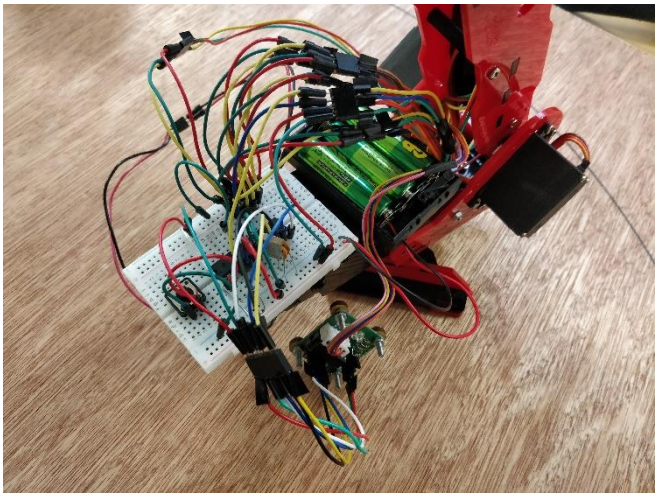
The Arduino Uno micro-controller is powered by the Atmel ATmega328 micro-processor. The processor itself can be removed from the Arduino Uno and used in a more versitile standalone circuit. This allows the microprocessor to be directly interfaced with, so hardware and software can be optimised for the given task. An embedded system is therefore more efficient. It is important that the pins are interfaced with correctly. Fortunately, the micro-processor has already been burnt with a custom Arduino bootloader. The above pinout is a pin map of the ATmega328 datasheet.



A standalone circuit requires some additional setup. To prevent the microprocessor from resetting during operation a 10k ohm resistor is attached from the RESET pin to +5V. A push button can be used here to reboot the microprocessor when pulled to ground. VCC and AVCC are connected to +5V whereas the two GND pins are sent to the negative terminal. A crystal oscillator regulates the clock speed of the microprocessor. I've used a 16 MHz crystal capacitor and two 22 pF ceramic capacitors which are attached between pins 9 and 10, running to ground.

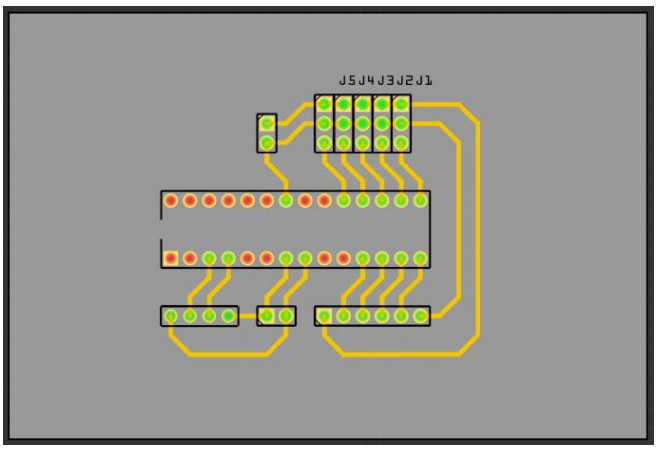


The above circuit is a standalone trial of a blinking led on a breadboard. The ATmega328 sits in a chip base (DIP socket). The external clock is attached to appropriate crystal pins and the 10k resistor to the RESET. Power is sent from the +5V to VCC pins and GND. I've decided to imlement a slide switch on the positive terminal of the microprocessor supply. This makes a push button on the reset pin redundant since two switch mechanisms are unnecessary. An led and 330 ohm resitor are attached to digital pin 9 with a delay of 1000 ms. Testing shows that the circuit indeed works, even after removing the supply from the Analogue VCC pin suggesting that the additional wiring is unnecessary.

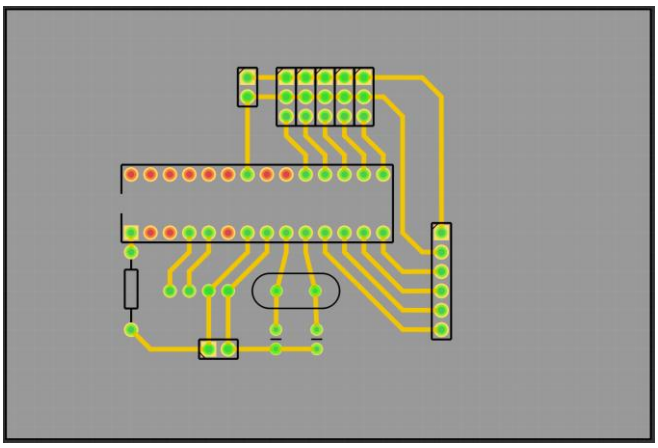


To ensure that an embedded system is compatible with my model, I trialled this more complex breadboard circuit. I removed the perfboard and instead soldered female header pins onto the actuators. I plan to solder male header pins to the PCB itself so I can plug in the actuators instead of soldering them directly into the PCB board – a more vertsitile approach that will likely prove essential during future trialling. For the breadboard above I used jumper wires to connect the components to it. I also tested a push button to see if its useful, though I have decided that it isn't so I will not be including it in future circuits. Though jittery – likely due to loose connections – the circuit appears to work.

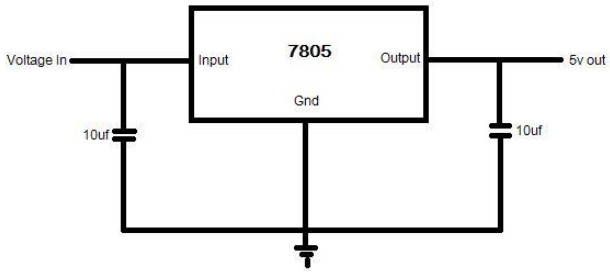
PCB Schematics



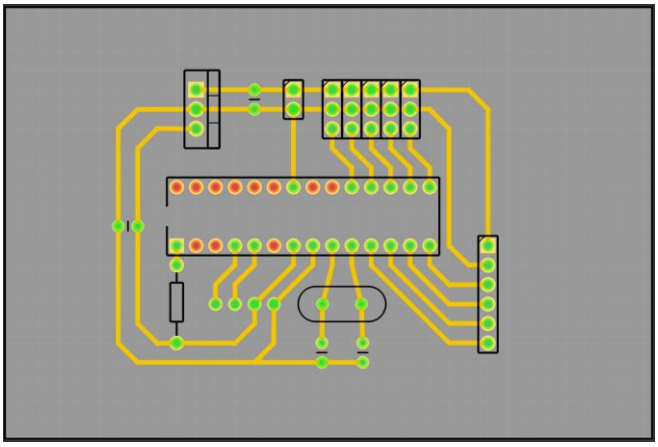
The above schematic was designed in fritzing – an open source CAD software for designing electronics hardware. The circuit above is a PCB schematic; it is not fully functioning (lacking an external clock), serving more as a software trial. In fritzing CAD files are saved with the .fzz file extension – a proprietary file type which cannot directly interface with PCB printers. Files must be exported as an 'Extended Gerber' for production. Due to limited resources and complexity, most of the exported files are unnecessary – only 'copperTop.gtl' and 'drill.txt' will be used. These files are copied (via a USB) to the Voltera V-One PCB printer I have access to.



The above schematic is a functioning prototype (in theory). A 10k resistor has been attached to the RESET and +5V. The crystal capacitor external clock is attached to GND. Power is sent through the VCC to GND from the +5V. The ultrasonic sensor is powered by the same supply and is connected to digital pins 2 and 3. Actuators are powered from an external +9V supply as before; the negative terminal is equalised with the other GND pin. Servos are attached to digital pins 9 through 13. The stepper motor is attached to digital pins 5-8. All components are rounted in parallel. I plan to solder PCB components directly through the vias, so hole sizes are 1.5mm (the barrel size of supplied rivets).

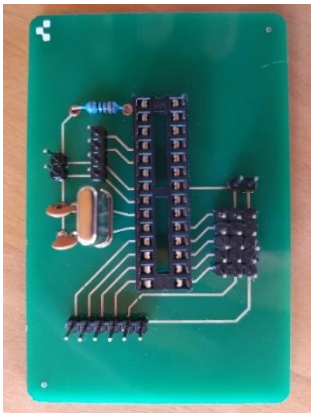


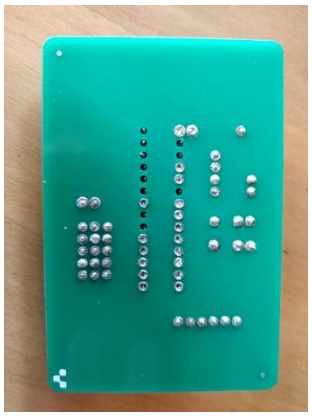
The perfboard prototype used two power supplies – one for the Arduino Uno and the other for some of the components. This was because the components needed more power than what was provided by a single power supply. Alternatively, a more powerful supply could be stepped down using a voltage regulator to power the microprocessor independently of the components. The above diagram details the setup required. The device works by emitting power as heat. A heat sink may be needed to dissipate this heat so the maximum internal die temperature (temperature tolerance) is not exceeded.



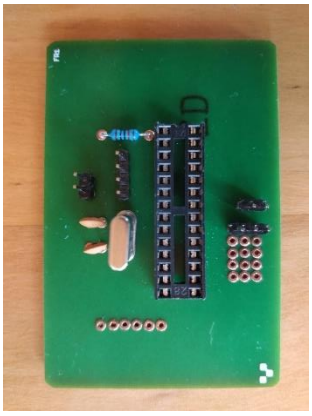
I've implemented a voltage regulator in the above circuit. The positive and negative terminals of the 9V supply run to the INPUT and GND of the regulator respectively. The 9V is stepped down to 5V, dissipating power as heat. The 5V runs through the OUTPUT pin. Two 10 μF capacitors bridge the positive and negative rails at either side of the regulator. These are used to reduce fluctuations in current through the device by reducing the equivalent series resistance (ESR). This is done by providing a low-impedance, high-frequency path at the input and output of the regulator to suppress high frequency noise from the load.

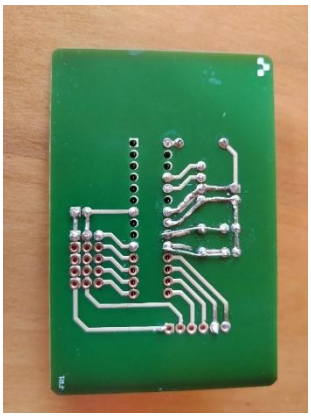
PCB Prototypes



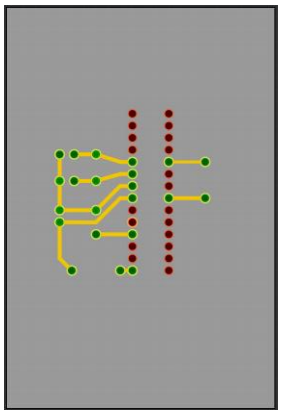


First prototype is non-functioning. I failed to consider a few potential issues which have manifested in the PCB above. The track is too thin and has scratched away in places causing the rails to become faulty. In fritzing the PCB view is top down so components must be attached to the top of the PCB, on the same side as the rails. This lends itself to additional track damage. Also, the conductive ink around the vias was too small so when the holes were drilled, little ink remained for rivets to contact with. Therefore the rivet head to track connections were faulty and unreliable. Rivets also began to lift since they were not installed with sufficient force. I will have to ensure that the rivets are all tight before moving on to soldering.

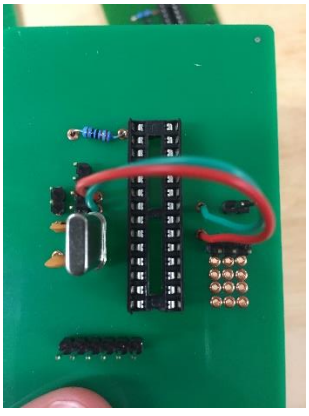


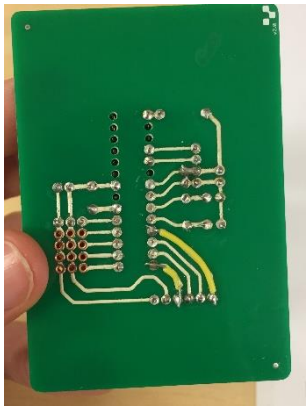


The second prototype is also non-functioning. Though I did increase the track width, there remains too little conductive ink around the vias. The thicker tracks tend to chip more often so during riveting the tracks sometimes split. I did a rough job of patching these breaks with solder and additional wire, however the solder doesn't stick well to the board or conductive ink. I may need to straighten the tracks to minimise potential weaknesses in the layout. I will also use an intermediate thickness between standard and extra thick so ink won't crack as much. I also discovered that the outer layer of conductive ink tends to oxidise during the baking process, preventing effective connection with rivets. I will have to burnish future circuits to avoid this.

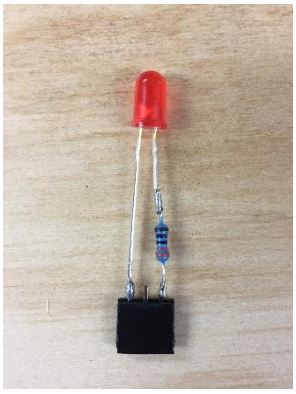


I decided to trial a simpler PCB circuit to more effectively troubleshoot problems and avoid wasting resources. The circuit above is a blinker circuit (similar to the breadboard). A pulse runs from digital pin 2 to a 330-ohm resitor, through the LED and to ground. For this circuit I increased the size of the ink around vias by replacing set components with single vias with more customization options for rivet size. The circuit works as hoped – likely because of these changes. I also added some track to the opposite side of the chip to perform additional tests, since I expected to encounter more issues. However, since the circuit worked first time I figured such tests were no longer necessary.



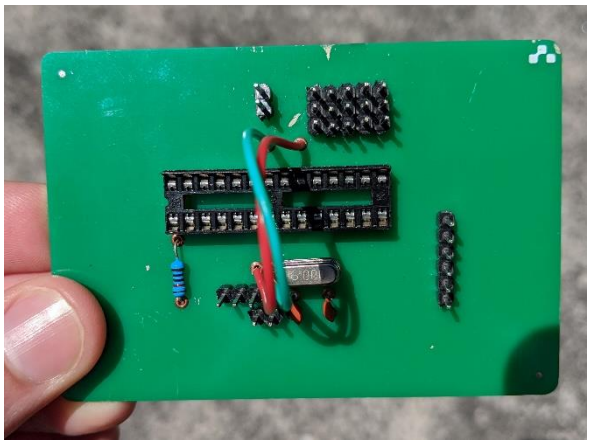


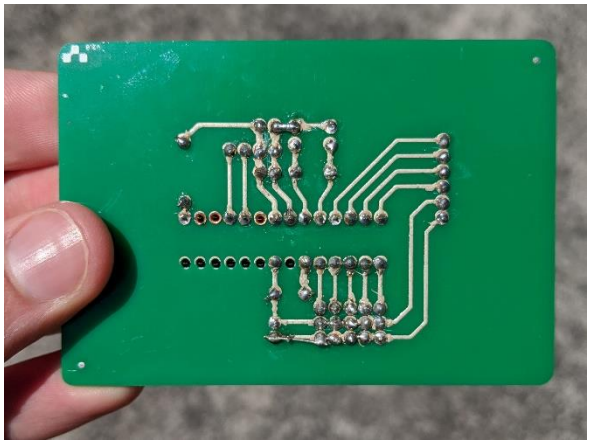
Further breadboard testing revealed that wiring through the AVCC and the second GND pin prevented issues with jitter by strengthening digital signals. So, in the above circuit I have run positive and negative wires over the chip to the respective pins. I straightened the tracks and reduced the track width slightly on the LED circuit, successfully preventing the tracks from cracking. Unfortunately, the same fix to the circuit above hasn't completely worked. It appears that the more complex circuit above does encounter similar cracking problems, though not as bad. Some repair jobs were necessary as a consequence. I may need to adjust the calibration setting of the PCB printer to reduce the amount of ink.



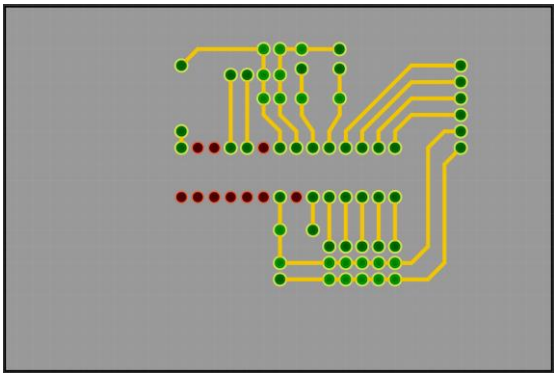
This LED-resistor device (right) has allowed me to effectively test the functionality of pin sets. The device is plugged into the male pins on the PCB. A digital signal is sent through the 330-ohm resistor from the ATmega chip, and then onto the LED causing it to light up, and finally the signal runs to ground. If the LED doesn't light up, I know there must be a connection issue. The 200-ohm resistor mode on a multimeter (left) allows me to determine exactly where that connection issue is. The terminals are connected across a track and give the resistance between two points. A null read means that there is no connection so the track must be faulty.

The Functioning PCB



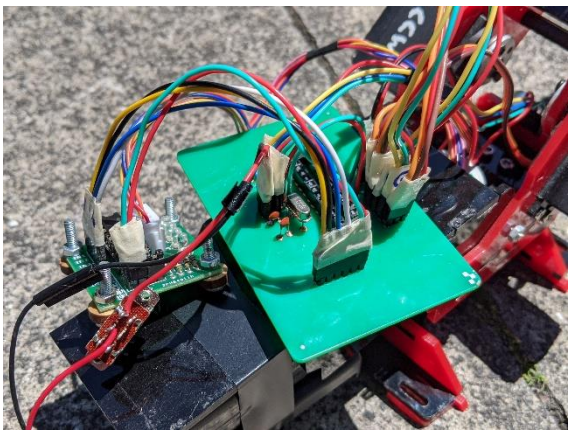


The necessary revisions (as discussed) were made in the preceding circuit. After installing the rivets, I added additional conductive ink by hand to the rivets to ensure there was a good connection between the rivets and the track. Adding more ink by hand post-baking also allowed me to strengthen tracks and fix damages made during the rivet installation process. After adding the ink, the PCB needed to be baked again. The circuit functions as needed. Unfortunately, the tracks are unable to carry a large current load. This means that I cannot have many actuators operating simultaneously. The model requires at least four servos operating at the same time (wrist, elbow and both shoulder servos). This need is met when the circuit is supplied from a consistent power source – for example when the circuit is supplied from a power adaptor plugged into a wall outlet. The constant large current draw drains the batteries very quickly, so a battery pack proves to be too inconsistent of a source. So, while this circuit serves as a proof of concept model, further revisions are necessary in a final product. Thicker tracks would be needed to carry a larger load; previous testing however revealed that thicker tracks tended to crack during the riveting process. Though this is somewhat fixable by applying ink post riveting, the connections aren't as good as they would be if the tracks weren't broken in the first place. The oxidised layer on top of the tracks interrupts the new connection (though burnishing helps deal to this). Voltera supplies flexible inks which allow for "higher conductivity" and "improved bending performance". Lead-acid batteries provide a higher output capacity than alkaline batteries. Changes such as these would likely provide the required improvements needed to make the circuit well-suited enough for a final product. However, due to limited time and resources, I am unable to make these changes.



The Functioning Model



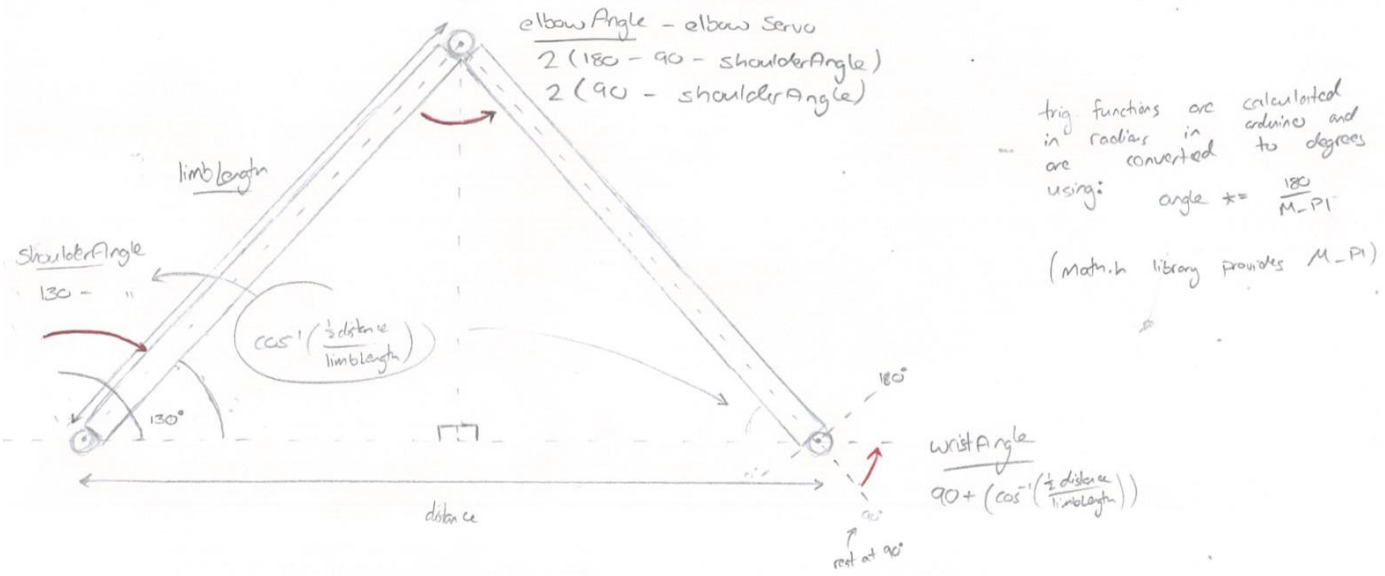



The tool's purpose is to fetch honey frames from within Mark's hives, so he can remain at a safe distance from the beehive and not get stung. The tool had to be durable – though not too heavy – and needed to tolerate many bees crawling over it while operating. The acrylic structure is brittle, yes, however the aspect of 'not collapsing under its own weight' outweighed its sturdiness. The acrylic structure probably wouldn't survive being dropped, though it is certainly durable enough to get the job done. The model is light and mobile, as requested, and I have satisfied their desire for a red and black colourway. Like Mr O'Brien suggested, I have employed modern digital technology methods in the operation of the device. Wiring is funnelled neatly down the chamber into a PCB board where digital signals are sent from an ATmega microcontroller to the connected components. Stakeholders seem content with the proposed outcome. Future refinements would need to be made before moving onto prototyping phases. However, the model has fulfilled its purpose: to verify whether the proposed solution to the conceptual statement is realistically achievable.

In developing my model, I have considered the safety and potential hazards. I have tried to optimise the usage of materials to promote sustainability. Potential damage or deterioration due to stress or different environmental conditions has been reduced by using long lasting materials and design techniques. Additional changes might involve shielding computer processing equipment in an acrylic guard. I hope to show my testing practices have been ethical and that my trialling procedures are culturally appropriate and strive for social acceptability.

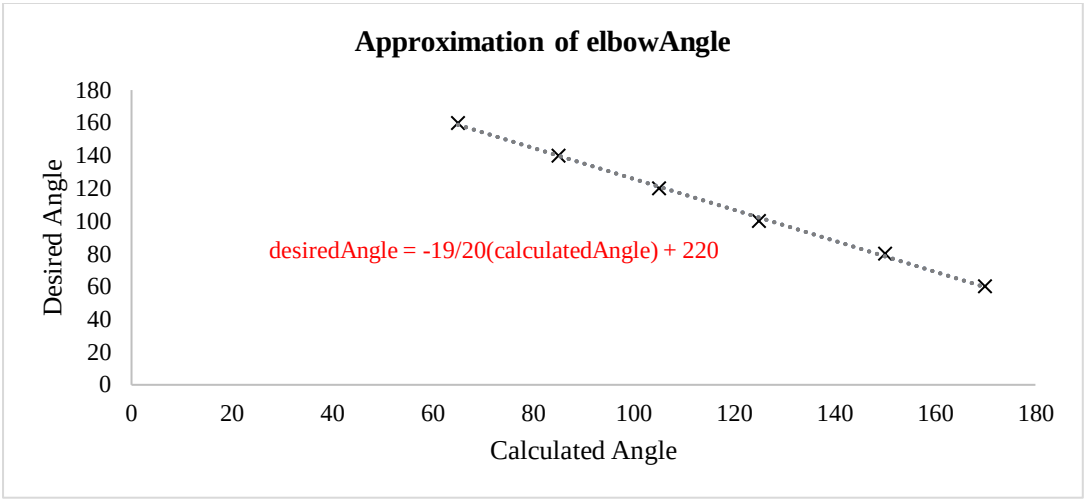
Though the servo hardware has the capacity to operate along two planes of rotation (x and z directions respectively), I only need the arm to operate along the x-plane (creating a script for additional control over the z-plane would not only be unnecessary, but exceedingly difficult). Movement along the x-plane is limited to between 22 cm and 40 cm from the base (origin) of the arm. Rotation of the entire arm around the stepper motor enables 360° exploitation of the y-plane. The process of grabbing the frame (though for testing purposes, a cube) will be automated. An ultrasonic sensor at the base pulses ultrasonic sound signals. These are scaled according to the wave speed (0.0343 cm/µs) to give the distance between the sensor and an obstruction in front. If the obstruction meets certain conditions (such as whether it is within the range stated), some arithmetic will be performed for determining the precise rotation of the servos for the given distance. These calculations will be based on trigonometric functions. The servos will be rotated to these values and the gripper will grab the frame. For greater versatility, the stepper motor will rotate 180° while the ultrasonic sensor sends sound pulses. This enables a wider range of movement to be exploited and the gathering of more data to determine the dimensions of the frame. The program will be based on picking up a cube in a controlled environment.

Algorithmic Considerations



The calculated elbow angle determines the angle between the upper arm and forearm. Inputting this value into the servo will not give this angle due to the lever mechanism. There is a linear relationship between the calculated angle and the desired angle however and can be found doing some simple investigation. First, I wrote the calculated angle to the elbow servo. I then measured the angle produced from the rotation and recorded this value in the table below. I repeated this procedure six times using 20° increments of the calculated angle and graphed the results to find the relationship:

Desired Angle	Calculated Angle
65	160
85	140
105	120
125	100
150	80
170	60



shoulderAngle = [$\cos^{-1}$(1/2distance / limbLength)] * (180 / π)

elbowAngle = -19/20 * [2(90 - shoulderAngle)] + 220

wristAngle = 90 + shoulderAngle

shoulderAngle = 130 – shoulderAngle

Part I – **Detecting and processing data**

Starting with the ping function to find distance:

```
void ping() { // Finds distance
  digitalWrite(trigPin, HIGH); // Sends signal through trigPin

  delayMicroseconds (10); // Pauses for 10 µs while signal is emitted

  digitalWrite(trigPin, LOW); // Stops sending signal through trigPin

  time = pulseIn(echoPin, HIGH); // Reads duration of pulse from echoPin

  distance = (time * 0.0343) / 2; // d = vt where v = 0.0343 cm/µs
}
```

This code utilises the ultrasonic sensor to approximate the distance to the closest surface directly in front of the device. A signal is sent through the transmitter attached to 'trigPin' for 10 microseconds, causing a pulse of ultrasound to be emitted. The propagation of these waves enables objects to be detected upon the waves returning to the receiver attached to 'echoPin'. The pulseIn() function reads the time it takes for the pulse to return to the receiver. This value is scaled according to the speed of these waves (0.0343 cm/µs) giving distance in centimetres (d = vt). The value derived is halved to account for the distance the sound waves travelled to return to the sensor.

```
void setup() { // Runs once on start
  pinMode(trigPin, OUTPUT); // trigPin emits signal
  pinMode(echoPin, INPUT); // echoPin recieves signal
}
```

The code requires some additional setup to configure the pins to the correct modes. 'trigPin' transmits the pulse so naturally must be configured as an OUPUT pin whereas 'echoPin' receives the pulse so is configured as an INPUT pin.

```
float time, distance; // Declares fraction variables
const int trigPin = 2, echoPin = 3; // Constant integers
```

Variables must also be defined. Here, time and distance are declared as 'float' variables, essentially fractions. The pin numbers must be constant integers – whole numbers that cannot be re-assigned later. They are therefore initialised as 'const int' variables to their respective pins.

Now that we can find the distance to the object, we must use that to calculate the angles the servos must turn to so the apparatus can reach for it:

```
void arithmetic() { // Calculates angles for servos from given distance
  // Proof earlier in digest
  shoulderAngle = acos((distance / 2) / limbLength);

  shoulderAngle *= (180 / M_PI); // Radians to degrees

  elbowAngle = -19/20 * 2 * (90 - shoulderAngle) + 220;

  wristAngle = 90 + shoulderAngle;

  shoulderAngle = 130 - shoulderAngle;
}
```

Here variables are assigned according to the proof established earlier in the folio.

```
#include <math.h> // Accesses math library
```

Trigonometric functions and the value of Pi are not readily available without accessing a math library. This code accesses the standard math library allowing me to use these mathematic functions.

```
const int limbLength = 20; // Constant integer
int shoulderAngle, elbowAngle, wristAngle, distance = 20; // Integer variables
```

Finally, variables must be declared/initialised. The length of each limb (i.e. upper arm and forearm) is 20 cm and is initialised as such as a constant integer. Angles are declared as integers while distance is initialised with the given value of 20 cm.

Now that these two functions are established, we can merge them into a cohesive program. First however, the 'ping()' function must be redefined as an integer function to return range. This is done fairly simply by replacing 'void' with 'int' and using 'range' in the place of distance. 'range' must then be returned upon a call to the function hence 'return range' is added to the bottom of the function. By adjusting the arithmetic function to now accept a float value for distance we can execute the two functions in a single elegant line of code:

```
arithmetic(ping()) // Calls arithmetic function and inputs range
```

At this point the input and processing aspects of the code is complete. The addition of a Serial Monitor enables angles to be seen in real time:

```
#include <math.h> // Accesses math library

float time, range; // Declares fraction variables
const int trigPin = 2, echoPin = 3; // Constant integers

const int limbLength = 20; // Constant integer
int shoulderAngle, elbowAngle, wristAngle; // Integer variables

void setup() { // Runs once on start
  pinMode(trigPin, OUTPUT); // trigPin emits signal
  pinMode(echoPin, INPUT); // echoPin recieves signal
  Serial.begin(9600); // Initialises Serial Monitor
}

void loop() { // Runs repeatedly
  arithmetic(ping()) // Calls arithmetic function and inputs range

  // Prints angles to Serial Monitor
  Serial.println(shoulderAngle);
  Serial.println(elbowAngle);
  Serial.println(wristAngle);
}

float ping() { // Returns range
  digitalWrite(trigPin, HIGH); // Sends signal through trigPin

  delayMicroseconds (10); // Pauses for 10 µs while signal is emitted

  digitalWrite(trigPin, LOW); // Stops sending signal through trigPin

  time = pulseIn(echoPin, HIGH); // Reads duration of pulse from echoPin

  range = (time * 0.0343) / 2; // d = vt where v = 0.0343 cm/µs

  return range; // Returns range upon calling the function
}

void arithmetic(float distance) { // Calculates angles for servos from given distance
  // Proof earlier in digest
  shoulderAngle = acos((distance / 2) / limbLength);

  shoulderAngle *= (180 / M_PI); // Radians to degrees

  elbowAngle = -19/20 * 2 * (90 - shoulderAngle) + 220;

  wristAngle = 90 + shoulderAngle;

  shoulderAngle = 130 - shoulderAngle;
}
```

Part II – Using processed data to manipulate actuators

Starting with basic manipulation of actuators:

```
void setup() { // Runs one on start
  // Attaches servos to repective pins
  shoulder.attach(4);
  elbow.attach(5);
  wrist.attach(6);
}
// Accepts integer values for desired servo positions
void gesture(int sPos, int ePos, int wPos) {
  // Rotates servos to desired angles
  shoulder.write(sPos);
  elbow.write(ePos);
  wrist.write(wPos);
}
```

This code attaches servos to digital pins and causes them to rotate to given angles. Calling the gesture function requires three values to be entered for each of the servos. 'sPos',' ePos' and 'wPos' are temporary integer values which represent the desired positions of each servo.

```
Servo shoulder, elbow, wrist; // Declares Servo variables
```

Servo variables must be declared of type 'Servo'.

```
#include <Servo.h> // Accesses Servo library
```

Servos are not readily available so the Servo library must be accessed.

Inputting the previously calculated angles into a call to the gesture function should cause the apparatus to reach for the object:

```
gesture(shoulderAngle, elbowAngle, wristAngle); // Calls gesture function and inputs angles
```

During execution of this statement servos do not operate simultaneously which means that the forearm is not given enough time to rotate counter-clockwise, away from the table, before the upper arm has completed its rotation, causing the forearm to hit the table.

Introducing a delay between each rotation solves the problem:

```
// Accepts integer values for desired servo positions
void gesture(int sPos, int ePos, int wPos) {
  wrist.write(wPos); // Rotates wrist to desired angle
  delay(1000); // Pauses for 1000 ms while wrist rotates to desired angle

  elbow.write(ePos); // Elbow servo rotates only after 1000 ms
  delay(1000);

  shoulder.write(sPos);
  delay(1000);
}
```

With this code the servos turn one after the other, starting with the wrist, then the elbow, and finally the shoulder. Unfortunately, the movement is now too slow and appears cumbersome which must be resolved.

Turning the servos simultaneously, 1° at a time, is another potential solution:

```
// Accepts integer values for desired servo positions
void gesture(int sPos, int ePos, int wPos) {
  while (true) {
    // If the current servo position is less than the desired position, perform contents
    if (sCurrent < sPos) {
      // Write the current servo position to the servo and then add 1 to it
      shoulder.write(sCurrent++);
      delay(10); // Pause for 10 ms
    }

    if (eCurrent > ePos) { // *Greater than
      elbowServo.write(eCurrent--); // *-1
    }

    if (wCurrent < wPos) {
      wristServo.write(wCurrent++);
    }

    if (sCurrent == sPos) { // If current servo position equals desired postion, perform contents
      if (eCurrent == ePos) {
        if (wCurrent == wPos) {
          break; } } } // Terminates the function
  }
}
```

This program requires three additional variables to be initialised. These variables represent the rest positions of each of the servos:

```
int sCurrent = 10, eCurrent = 170, wCurrent = 90;
```

This program uses the rest values as a reference point for the current servo positions. It assumes that the servos start at the rest values and then the program increments them respectively. The program takes the current value of a given servo, say shoulder, and checks to see whether this is (in this case) less than the desired angle, shoudlerAngle. Since the code exists within the while function – which runs repeatedly since it is always true – while the current position is less than the desired angle, the servo rotates one degree in the desired direction each loop iteration. This occurs until all the servos have reached their desired positions wherein the function is terminated. The group of if statements at the bottom is responsible for this termination. It checks each iteration to see whether they have indeed reached them and upon all statements being true (when the servos have reached the desired positions), the while loop terminates (hence the 'break' statement). The 10 ms delay in the first if statement exists to slow down the movement to ensure it isn't moving so fast that the apparatus breaks. This code is limited in its use because it only works in one direction and is rather bloated.

For the code to work in both directions the function has to take the bearing (clockwise or counter-clockwise relative to the rotation of the shoulder) as an input:

```
// Accepts integer values for desired servo positions and bearing
void gesture(int sPos, int ePos, int wPos, int bearing) {
  while(true) {
    if (shoulder.read() != sPos) { If servo doesn't equal given position, perform contents
    // Writes value returned from call to increment function to the servo
      shoulder.write(increment(shoulder, bearing));
      delay(10); // Pauses for 10 ms
    }

    if (elbow.read() != ePos) {
      elbow.write(increment(elbow, bearing *= counterclockwise));
    }

   if (wrist.read() != wPos) {
      wrist.write(increment(wrist, bearing));
    }
```

```
    if (cwShoulder.read() == sPos) { // If servo position equals desired postion, perform contents
      if (elbow.read() == ePos) {
        if (wrist.read() == wPos) {
         break; } } } // Terminates the function
  }
}
```

This code does away with the current position variables. Instead, 'servo.read()' statements read the last angle written to the servo (so for all intents and purposes, the current servo angle) and increments these using an 'increment' function. The new function increments each angle 1° in the desired direction which can be seen below. In this code the greater-than and less-than signs have been replaced with not-equal-to signs. This increases the functions versatility as it is no longer limited to only one direction. The updated function also accepts the bearing of the rotation – either 1 or -1, defined as constant integers 'clockwise' and 'counter-clockwise' respectively. This tells the function whether to extend or retract the apparatus relative to the movement of the shoulder (clockwise shoulder rotation meaning extend, counter-clockwise shoulder rotation meaning retract). Since the elbow operates in the opposite direction as the shoulder and wrist servos (i.e. its rest position is 170° compared to the shoulder's 10° so unlike the shoulder it must rotate counter-clockwise when reaching for the object) it must have the bearing multiplied by the counter-clockwise variable in order to rotate in the right direction.

```
// Integer function which accepts a Servo and integer variables
int increment(Servo servo, int bearing) {
  int angle = servo.read(); // Initialises angle variable and assigns current servo position
  angle += bearing; // Adds bearing to angle
  return angle; // Returns angle upon calling the function
}
```

The increment function is an integer function which returns an angle plus or minus one in the desired direction of the current servo position. It first initialises the integer 'angle' and assigns the current angle of the inputted servo to it. It then adds the bearing to the angle (this could be 1 or -1 depending on whether clockwise or counter-clockwise is inputted as the bearing) and proceeds to return this value to the call to the function. Back in the gesture function the servo is written to the new angle plus or minus one in the desired direction.

It occurred to me that this code could be majorly streamlined by using the increment function to not only increment the angle but to also write to the given servo. This means each servo doesn't need its own write function and instead can be written to from a call to a single function:

```
// Integer function which accepts a Servo and two integer variables
void increment(Servo servo, int pos, int bearing) {
  int angle = servo.read(); // Initialises angle variable and assigns current servo position

  if (angle != pos) { // If angle doesn't equal given position, perform contents
    angle += bearing; // Adds bearing to angle
    servo.write(angle); // Returns angle upon calling the function
  }
}
```

Giving a much-simplified gesture function:

```
// Accepts integer values for desired servo positions and bearing
void gesture(int sPos, int ePos, int wPos, int bearing) {
  while (true) {
    increment(shoulder, sPos, bearing); // Calls increment function and inputs servo, position and bearing data

    increment(elbow, ePos, bearing * counterclockwise);

    increment(wrist, wPos, bearing);

    if (shoulder.read() == sPos) { // If servo position equals desired postion, perform contents
      if (elbow.read() == ePos) {
        if (wrist.read() == wPos) {
          break; } } } // Terminates the function
  }
}
```

```
void handle(int gPos, int bearing) { // Function for gripper control
  increment(gripper, gPos, bearing);
  delay(5); // Pauses for 5 ms
}
```

The gripper requires additional control. Hence this function isolates the gripper. It accepts two integer variables; desired position ‘gPos’ and the bearing of the rotation. The 5 ms delay is necessary to slow down the movement.

```
void revolve(int displacement, int bearing) { // Function for stepper morot control
  for (int s = 0; s < displacement; s++) { // For loop repeats until stepper reaches desired displacement
    stepper.step(bearing); // Steps once in the clockwise or counterclockwise direction (depending on input
  }
}
```

This revolve function accepts two integers; displacement and bearing. These specify for a stepper motor how many steps it should take and in what direction as executed by the for loop. The stepper takes one step each for loop iteration.

```
#include <Stepper.h> // Accesses Stepper library
Stepper stepper(4096, 9, 11, 10, 12); // Stepper motor using pins 9 - 12; 4096 SPR
```

The Stepper library must be accessed. Also, upon declaring the stepper motor, steps per revolution and pins must be specified. The model I’m using takes 4096 steps per revolution and is attached to pins 9 through 12.

An additional servo at the shoulder is included. It rotates opposite the initial shoulder servo. So, finally:

```
// Accesses respective libraries
#include <Servo.h>
#include <Stepper.h>

Servo cwShoulder, ccwShoulder, elbow, wrist, gripper; // Servo variables
Stepper stepper(4096, 9, 11, 10, 12); // Stepper motor using pins 9 - 12; 4096 SPR
const int clockwise = 1, counterclockwise = -1; // Constant integers

// Accepts integer values for desired servo positions and bearing
void gesture(int sPos, int ePos, int wPos, int bearing) {
  while (true) {
    increment(cwShoulder, sPos, bearing); // Calls increment function and inputs servo, position and bearing data

    increment(ccwShoulder, sPos, bearing * counterclockwise);

    increment(elbow, ePos, bearing * counterclockwise);

    increment(wrist, wPos, bearing);

    if (cwShoulder.read() == sPos) { // If servo position equals desired postion, perform contents
      if (elbow.read() == ePos) {
        if (wrist.read() == wPos) {
          break; } } } // Terminates the function
  }
}

// Integer function which accepts a Servo and two integer variables
void increment(Servo servo, int pos, int bearing) {
  int angle = servo.read(); // Initialises angle variable and assigns current servo position

  if (angle != pos) { // If angle doesn't equal given position, perform contents
    angle += bearing; // Adds bearing to angle
    servo.write(angle); // Returns angle upon calling the function
  }
}

void handle(int gPos, int bearing) { // Function for gripper control
  increment(gripper, gPos, bearing);
  delay(5); // Pauses for 5 ms
}

void revolve(int displacement, int bearing) { // Function for stepper morot control
  for (int s = 0; s < displacement; s++) { // For loop repeats until stepper reaches desired displacement
    stepper.step(bearing); // Steps once in the clockwise or counterclockwise direction (depending on input)
  }
}
```

**Servo attachment statements not included as these are subject to change.*

Part III – Developing a controller to effectively manoeuvre the mechanical muscle

The controller is the principal function of the program. It interprets sensory data and signals the actuators accordingly.

```
stepper.step(1); // Steps clockwise
distance = ping(); // Finds distance
```

The mechanical muscle is designed to find and move objects within a 40 cm radius. The ultrasonic sensor is attached to the base of the apparatus so turns with the stepper motor. This means for each step that the stepper motor takes, the ultrasonic sensor must emit a pulse. Hence the code above.

```
if (distance >= minDist && distance <= maxDist) { // Reduces noise
```

The program then needs to determine whether this distance fits within the range specified by the limits of the apparatus. As established the objects may be found in a 40 cm radius so the device must have a maximum reach of 40 cm. This program assumes that any obstructions within this radius will be designated objects and not unintentional obstructions such as a wall. The testing environment grants these assumptions. Thus, the device must have a maximum reach of 40 cm. The apparatus is limited by its flexibility so the minimum distance must be 22 cm. These two values are stored as constant integer ‘maxDist’ and ‘minDist’ respectively and the if statement compares these to the distance gathered by the pulse. If the distance falls outside this range, no obstruction is detected and so the program skips the statement.

Now it’s time to put some code inside the if statement:

```
if (!obstruction) { // If an obstruction has not yet been detected
  obstruction = true; // Set 'obstruction' to true
  initial = step; // Assigns the current step to the 'intial' edge of the object face
}
/* Continually re-assigns the current step to the 'final' integer variable until the
obstruction is no longer detected and this way finds the 'final' edge of the object face  */
final = step;
```

The program cannot assume that an obstruction has a mono-dimensional face. It must therefore determine the width of the obstruction, mapping the object. This is achieved by determining whether the program as already detected an obstruction in the last step. The obstruction variable is a Boolean that is initialised as false; no obstruction has been detected yet. Upon detecting an obstruction, the program sets this variable to true and stores the current step integer as the ‘initial’ edge of the obstructions face. The step variable is an integer which will be initialised to 0 and will be iterated upon at the end of the loop function. This means that the value of ‘initial’ edge will not be overwritten upon future loop iterations. It is necessary however for value of the ‘final’ edge to be overwritten each iteration. This way the ‘final’ edge is continually updated until the first if statement comparing distance no longer runs true, which will only occur once the ultrasonic sensor has entirely passed the object. With these two variables the program can some simple maths to determine the centre of the object to turn back to upon scanning the whole region.

Before that however, the program has to set ‘obstruction’ to false when an obstruction is no longer detected:

```
} else { // If the above statement comparing distance is not true

  if (obstruction) { // If an obstruction was detected on the last step
    obstruction = false; // Set 'obstruction' to false
  }
}
```

An else statement connected to the if statement comparing distance serves well in this case. This will run when the if statement no longer reads true. It is necessary to use an if statement to specify that only if there was an obstruction detected prior to the execution of the else statement to set obstruction to false. Otherwise the program would be needlessly wasting resources constantly setting obstruction to false if it was already.

Outside the if-else statements it is appropriate to do some simple arithmetic:

```
/* Calculate the step which correlates to the middle of the object face and assigns this to the 'i'
instance of the 'position' array */
position[i] = stepRange - (final - initial) / 2 + initial;
// Increments 'i' by 1 so the next instance of the 'position' array is assigned upon the next statment call
i++;
```

I've used an array here to store the step that represents the middle of the object. An array means I can store many positions of objects within different instances of the same variable. The letter 'i' represents each instance and is initialised as 0 (since counting in an array starts from 0). 'i' is then incremented so that the next assignment is for position is a different instance. The calculation is fairly simple; it takes the middle of the object 1/2(final – initial) and adds this to the initial edge to give the step from the starting point to the middle of the object. This is then subtracted from the total range of the region. This is because the arm must return to the object after rotating the full range so the position must describe how many steps to the object turning counter-clockwise.

Up till now the code has been designed to run repeatedly within the loop function to map the region. This code, while still existing within the loop function, decides what happens after the apparatus has reached the end of the region:

```
if (step == stepRange) { // If the stepper has completed the full range
  for (i; i >= 0; i--) { // Repeat for each object within the range
    muscle(); // Perform the 'muscle' function to grab the object
  }

  revolve(stepRange, 1); // Return to the starting point
  exit(0); // Terminate the program
}
```

The variable 'stepRange' is a constant integer which defines the edge of the region. A 180° regions consists of 1024 steps of the stepper motor so 'stepRange' will be defined as such. The if statement compares the current step (which will have been iterated upon each loop iteration) to this value. Upon running true, the apparatus has reached the end point of the region and so mapping is complete, and it is time to actually grab the objects. The for loop takes the value of 'i' and while it is greater than or equal to zero, the function muscle is executed. One is subtracted from 'i' at the end of each iteration of the for loop. The result is that the mechanical muscle grabs each object within the range. The muscle function consists of the steps which the program must follow to grab the object. To finish off the if statement the arm returns to the stating position and then the program is terminated.

The muscle function:

```
void muscle() { // Grabs object
  revolve(position[i], counterclockwise); // Turns to object

  arithmetic(ping()); // Calculates angle for given distance

  gesture(shoulderAngle, elbowAngle, wristAngle, clockwise); // Extends arm

  handle(gripperAngle, clockwise); // Grips object

  gesture(shoulderRest, elbowRest, wristRest, counterclockwise); // Retracts arm

  revolve(position[i], clockwise); // Turns to the dropping zone (end of range)

  handle(gripperRest, counterclockwise); // Releases object
}
```

This function combines the previously established functions in both data processing and actuator manipulation sections of the folio. First the stepper motor rotates counter-clockwise toward the object's position. Inputting the ping value into the arithmetic function finds the angles the servos must turn to. The servos are then turned to these angles, reaching for the object. Upon reaching the object the gripper handles it. The apparatus then returns to its rest position and proceeds to step towards the dropping point at the edge of the region. The object is dropped and the function repeats for each of the objects detected within the range. 500 ms delays may have to be added between each statement to ensure that each statement has enough time to be completed before moving on to the next stage.

Part IV – Bringing it all together

It would be more efficient to separate each module into a distinct class or method. Unfortunately, the Arduino IDE doesn't support these, so each function has been brought together into a single cohesive program. The object-oriented coding structure is more versatile and promotes better organisation of line by line execution. Optimising the program in this way saves much storage space and enables dynamic memory to work much more efficiently during runtime.
Some slight revisions have been made and are listed below.

- The 'pulseIn()' function reads a pulse (either high or low) on a pin. By default, it takes 1 second to wait for the pulse to start. Since the program pings once per step, for which there is 1024 within the range, the statement generates noise since the ultrasonic sensor is unable to ping fast enough. To reduce noise, the pulse duration ('timeout') must be specified. 10 ms (10000 µs) is enough to suffice, hence I have added '10000UL' to the end of the pulse statement.
- 'attach' function attaches servos and writes them to their rest values. Function is executed in the setup function; however, servos aren't needed until much later in the program. Upon attachment servos return to 0°. Due to the construction of the mechanical muscle some of the servos rest at 170° and rotate counter-clockwise from there. Thus, it is necessary to write them to their respective rest values immediately after attachment.
- Adjusted digital pins according to circuit revisions to ensure servos are attached correctly.

```
// Accesses respective libraries
#include <math.h>
#include <Servo.h>
#include <Stepper.h>

float time, distance; // Declares fraction variables

Servo cwShoulder, ccwShoulder, elbow, wrist, gripper; // Servo variables
Stepper stepper(4096, 5, 7, 6, 8); // Stepper motor using pins 9 - 12; 4096 SPR

// Constant integers
const int shoulderRest = 10, elbowRest = 170, wristRest = 90, gripperRest = 10;
const int minDist = 22, maxDist = 40, stepRange = 1024;
const int trigPin = 2, echoPin = 3, limbLength = 20;
const int clockwise = 1, counterclockwise = -1;

// Integer variables
int shoulderAngle, elbowAngle, wristAngle, gripperAngle = 90;
int step = 0, initial, final, position[5], i = 0;

bool obstruction; // True or false

void setup() { // Runs once on start
  pinMode(trigPin, OUTPUT); // trigPin emits signal
  pinMode(echoPin, INPUT); // echoPin recieves signal
  attach(); // Attaches servos
}

void loop() { // Runs repeatedly
  controller();
}

float ping() { // Returns range
  digitalWrite(trigPin, HIGH); // Sends signal through trigPin

  delayMicroseconds (10); // Pauses for 10 µs while signal is emitted

  digitalWrite(trigPin, LOW); // Stops sending signal through trigPin

  time = pulseIn(echoPin, HIGH); // Reads duration of pulse from echoPin

  range = (time * 0.0343) / 2; // d = vt where v = 0.0343 cm/µs

  return range; // Returns range upon calling the function
}

void arithmetic(float dist) { // Calculates angles for servos from given distance
  // Proof earlier in digest
  shoulderAngle = acos((dist / 2) / limbLength);

  shoulderAngle *= (180 / M_PI); // Radians to degrees

  elbowAngle = -19/20 * 2 * (90 - shoulderAngle) + 220;

  wristAngle = 90 + shoulderAngle;

  shoulderAngle = 130 - shoulderAngle;
}

void attach() { // Attaches servos and sets them to rest values
  cwShoulder.attach(12); // Attaches servos to respective pins
  cwShoulder.write(shoulderRest); // Writes servos to rest values

  ccwShoulder.attach(13);
  ccwShoulder.write(180 - shoulderRest);

  elbow.attach(11);
  elbow.write(elbowRest);

  wrist.attach(10);
  wrist.write(wristRest);

  gripper.attach(9);
  gripper.write(10);
}
```

```
// Accepts integer values for desired servo positions and bearing
void gesture(int sPos, int ePos, int wPos, int bearing) {
  while (true) {
    increment(cwShoulder, sPos, bearing); // Calls increment function and inputs servo, position and bearing data

    increment(ccwShoulder, sPos, bearing * counterclockwise);

    increment(elbow, ePos, bearing * counterclockwise);

    increment(wrist, wPos, bearing);

    if (cwShoulder.read() == sPos) { // If servo position equals desired postion, perform contents
      if (elbow.read() == ePos) {
        if (wrist.read() == wPos) {
          break; } } } // Terminates the function
  }
}

// Integer function which accepts a Servo and two integer variables
void increment(Servo servo, int pos, int bearing) {
  int angle = servo.read(); // Initialises angle variable and assigns current servo position

  if (angle != pos) { // If angle doesn't equal given position, perform contents
    angle += bearing; // Adds bearing to angle
    servo.write(angle); // Returns angle upon calling the function
  }
}

void handle(int gPos, int bearing) { // Function for gripper control
  increment(gripper, gPos, bearing);
  delay(5); // Pauses for 5 ms
}

void revolve(int displacement, int bearing) { // Function for stepper morot control
  for (int s = 0; s < displacement; s++) { // For loop repeats until stepper reaches desired displacement
    stepper.step(bearing); // Steps once in the clockwise or counterclockwise direction (depending on input)
  }
}

void muscle() { // Grabs object
  revolve(position[i], counterclockwise); // Turns to object
  delay(500); // Pauses for 500 ms while statement is executed

  arithmetic(ping()); // Calculates angle for given distance
  delay(500);

  gesture(shoulderAngle, elbowAngle, wristAngle, clockwise); // Extends arm
  delay(500);

  handle(gripperAngle, clockwise); // Grips object
  delay(500);

  gesture(shoulderRest, elbowRest, wristRest, counterclockwise); // Retracts arm
  delay(500);

  revolve(position[i], clockwise); // Turns to the dropping zone (end of range)
  delay(500);

  handle(gripperRest, counterclockwise); // Releases object
  delay(500);
}

void controller() {
  stepper.step(1); // Steps clockwise
  distance = ping(); // Finds distance

  if (distance >= minDist && distance <= maxDist) { // Reduces noise
    if (!obstruction) { // If an obstruction has not yet been detected
      obstruction = true; // Set 'obstruction' to true
      initial = step; // Assigns the current step to the 'intial' edge of the object face
    }
    /* Continually re-assigns the current step to the 'final' integer variable until the
    obstruction is no longer detected and this way finds the 'final' edge of the object face  */
    final = step;

  } else { // If the above statement comparing distance is not true

    if (obstruction) { // If an obstruction was detected on the last step
      obstruction = false; // Set 'obstruction' to false
      /* Calculate the step which correlates to the middle of the object face and assigns this to the 'i'
      instance of the 'position' array */
      position[i] = stepRange - (final - initial) / 2 + initial;
      // Increments 'i' by 1 so the next instance of the 'position' array is assigned upon the next statment call
      i++;
    }
  }

  if (step == stepRange) { // If the stepper has completed the full range
    for (i; i >= 0; i--) { // Repeat for each object within the range
      muscle(); // Perform the 'muscle' function to grab the object
    }

    revolve(stepRange, 1); // Return to the starting point
    exit(0); // Terminate the program
  }

  step++; // Increments step by 1
}
```