# Introduction

Autonomous Underwater Vehicles have been around since the 1950s, with the first AUV on record developed by Washington University in 1957 named SPURV[10] (Self Propelled Underwater Research Vehicle). The device was fitted with various temperature and pressure sending probes and could tra- verse to depths of 3600m and one of the last uses of the SPURV AUV was to study submarine wakes in the 70s. Since this first development, in the last 67 years AUVs have been refined and further developed by various govern- ments, oceanographic institutes, universities and private companies. Today, AUVs are often considered cheaper, better and safer to deploy than manned underwater vehicles (particularly in the light of the more recent Titan sub- marine implosion[22]), and safer than deploying divers. They are capable of obtaining more data than any single diver can obtain, and capable of div- ing longer than any manned vehicle as the requirement for breathable air is negated. An AUVs dive time is only limited by its available power which will only increase as power technology improves. AUVs are used across a mul- titude of different industrial sectors spanning scientific research, surveying, military, shipping and gas and oil[11].

The aim of this project is to design and develop a rudimentary AUV. This project does not comfortably fall into any of the predefined template project ideas but was born from a personal enjoyment of the Internet of Things module which allowed me to explore microcontroller programming and interacting with physical sensors and actuators, which blossomed into a passion for robotics. As one of the early module videos highlighted, our projects should be something we are passionate about as without passion we would lack the motivation to complete such a lengthy and isolated piece of work[24]. I found that the existing Internet of Things templates involved topics I’m not particularly passionate about. I did gain permission from the Coursera Tutor Forums prior to embarking on this journey[31].

This project was initially inspired by the work of a YouTuber (Brick Ex- periment Channel[3]) who created a small remote controlled submarine using a watertight container and Lego. The accompanying blog[4] depicts the de- sign and development of several versions of this device, and also explains some of the key concepts that need to be considered such as buoyancy. The original plan for this project was to do something similar to this (occupying

the hobby space of the AUV market) - a small device that provides basic functionality autonomously, using similar materials (a water tight container and Lego). However, following the initial research into the AUV market there is a trend of AUVs costing thousands[23], a cost which will increase with the additional custom payloads (sensor arrays) required for the specific research projects. There are a small handful of manufacturers that offer lower cost AUVs to customers such as the ecoSub[5] which states one of

their submarines are *≈*£15k, this is a price point that may still be out of

reach for many projects. In addition to the initial cost of AUVs being ex- pensive, the parts would be custom made by the manufacturer and so any repairs could be equally costly.

While the inspiration for this project started with a small remote con- trolled submarine, the initial research has expanded the scope to try and develop an extremely low cost AUV which could be made open source (al- lowing for easy customisations) and can be manufactured by the researchers that require them using readily available parts and 3D printing which is now common relatively common and would mitigate the requirement for 3*rd* party companies fabricating parts. This would also mitigate costly repairs as all the design files would be a part of the project. The target users of this AUV may be required to develop their own payloads, but even the cost of hiring a developer to do this may still be cheaper than buying AUVs from known sources.

By developing an open source AUV, this could seriously open up under- water research for projects that are underfunded. These projects may have previously relied on human divers to obtain data as the safer unmanned al- ternative is out of reach for them so this would make such endeavours safer too.

# Literature Review

## Key Literature

The definition for an AUV is best described in an article titled “Autonomous Underwater Vehicles (AUVs): Their past, present and future contributions to the advancement of marine geoscience”.

Autonomous Underwater Vehicles (AUVs) are unmanned, self- propelled vehicles that are typically deployed from a surface ves- sel, and can operate independently of that vessel for periods of a few h to several days.[36]

The authors go on to highlight that “In addition, recent economic drivers, such as rapidly increasing vessel fuel oil costs, are making autonomous sys- tems a potentially attractive proposition to organisations responsible for large-scale and cost-effective marine data collection programmes”[36]. This article addresses the commercial benefits of AUV production and deploy- ment but doesn’t address the human benefits such as safety.

The authors also discuss the applications of AUVs and point out that “the sensors deployed determine the vehicle altitude, as well as its speed and endurance.”[36] The endurance in this context is the available power to the AUV which would affect its overall range. Because different sensors and actuators have varying power requirements, some of these components may draw more power than others. The array of sensors required is defined by the AUVs use case: it would be inefficient mounting water quality sensors onto an AUV designed for ocean bed mapping survey.

While this article includes a lot of jargon relating to marine geosciences, it provides a good overview of the uses of AUVs and their previous appli- cations. More importantly, it provides some good starting points for some concepts I will need to explore during the development of this project. This includes references to works that look into the Dead Reckoning method of positioning, and a launch pad into some key concepts such as Sonar.

Because the sensors deployed on an AUV are specific to its function and is something that can be configured, it is worth considering a modular design. As a part of this there could be some level of connectivity between

the separate modules. Modern MCUs typically feature multiple options for communication protocols such as UART, USART, *I*2*C*, SPI, USB, Ethernet or could be as simple as pulling a pin high or low. For the purpose of this project, we will consider *I*2*C* as it uses minimal wires, and allows for two way communication. The MCU data sheet[20] is the manufacturers documentation and is a vital reference for development on any given MCU and it directly relates to the MCU specific library in code. It has all the details and requirements to setup any of the provided MCU features, and gives a brief overview of each feature.

With regards to the ATTiny 1627 MCU, the data sheet describes *I*2*C* as a TWI (Two Wire Interface): “The Two-Wire Interface (TWI) is a bidi- rectional, two-wire communication interface (bus) with a Serial Data Line (SDA) and a Serial Clock Line (SCL)”[20]. There are several advantages to using the TWI bus as a communication protocol as it allows the connec- tion of “one or several slave devices to one or several master devices”[20] allowing a network of devices to become interconnected. This would also allow for parallel computation as instead leaving one device to perform all of the sensor readings and calculations, some of which could be computation- ally expensive or using deliberate pauses of code execution, a master device could simply trigger the networked modules to each perform their actions and then do nothing while waiting for the responses.

In the article ’FEATURES, OPERATION PRINCIPLE AND LIMITS OF SPI AND *I*2*C* COMMUNICATION PROTOCOLS FOR SMART OB-

JECTS’ the authors share their opinion on how an IoT network should operate and how it should be secured, and then give an overview of both *I*2*C* and SPI protocols before suggesting a hybrid version of the two. For IoT networks the authors suggest that “different devices with different capa- bilities will take part in the creation of a network. A self-describing interface for each device is necessary in order to optimize the management of required tasks”[33]. This point backs up my idea of a modular design where sensor modules can be attached in a way to form a network where each module is responsible for a specific task. They then go on to highlight that “adding an object to the network should not cause the collapse of the network itself. The network should also properly handle the failure of a device”[33]. This point involves error handling within a network of discreet modules. If one module fails, the whole unit should not. If the module is deemed vital to the devices operation, such as obstacle avoidance sensors on an AUV, a graceful way to handle this could be to surface the device and broadcast a signal indicating the error on a predefined wavelength so it can be collected.

For *I*2*C* the authors give a detailed overview stating that:

The *Inter Integrated Circuit* (*I*2*C*) protocol, ..., was developed by Philips in 1982 and it is a serial, single-ended bus with multi- master support and typically used to connect low speed de-

vices.[33]

The bandwidth available over the standard *I*2*C* rail is slow at 100kbit/s, but there is a Fast mode and Fast Plus mode which allows up to 1Mbit/s. Additionally some devices also support a high speed mode which allows up to 3.4Mbit/s.[2]. This means the *I*2*C* rail may not be suitable for *all* types of sensors attached, particularly ones that gather a lot of data in real time, for example high definition cameras.

Since the bus is completely shared, it is possible, for devices, to receive not only unicast transmission but also broadcast mes- sages: for this reason, whenever master wants to write or read data from a particular slave, it will address it first. Addressing bits were originally seven, extended to 10 with the latest reviews to increase supported maximum number of connected devices. *I*2*C* bus supports multiple masters on the same bus too, but a proper conflict-solving algorithm has to be implemented. [33]

This addressing is how the devices know what devices they need to talk to on a network. With the system of addressing and the ability to broadcast transmissions we can begin to visualise the *I*2*C* protocol as akin to web sockets. The interesting point raised here is that conflict resolution needs to be handled where the network has multiple masters. One such proposal they put forward is Arbitration: “since *I*2*C* supports multiple masters, an arbitration rule is necessary. In this protocol, arbitration proceeds bit by bit and the rule is deterministic: the first master that produces a one when the other produces a zero loses the arbitration”[33].

Another resource that covers a lot of information about AVR Microcon- trollers and also providing an overview of the features as well as how to interact with actuators such as servos and stepper motors, is ’Make: AVR Programming’[35]. While this book does contain quite a broad overview of the features and protocols, it also has examples of code to interact with them, as well as an overview of the toolchain required to programme a mi- crocontroller (in this instance involving Make, avr-gcc[8] and AVRDude[1] as well as a physical hardware programmer). However, the example code is all written for a specific MCU that the book focuses on, being the AT- MEGA168. However, as the author notes, this should not be too difficult to port to my specifically chosen MCU as the book also tries to teach another skill being the ability to refer to relevant sections of the data sheet.

As well as internal device communication protocols, it would also be good if we could have some way of interacting with the device in flight: receiving data, and possibly some rudimentary manual over ride/remote control. As corroborated in an article exploring underwater wireless networks, “radio is seriously attenuated in water”[9]. Radio waves exist on the EM Spectrum

and the lower the frequency, the further it can penetrate water due to the electrical dissipation that occurs, which is accentuated by salt water. High frequencies that we are used to as standard data transfer frequencies (WiFi, Bluetooth) are measured in GHz, and these penetrate only a few cm of water. Even LoRa at 433MHz will only penetrate 30-50cm of water. The military use lower radio frequencies operating in the range of Hz to kHz to communicate with their submarines but the trade off is lower frequen- cies bandwidth (less data can be transferred as there are fewer cycles per second). Additionally, radio frequencies are controlled[25] and as a hobby enthusiast/student I only have access to a limited range of frequencies.

The article also states “acoustic communication is almost the only effec- tive way for underwater wireless transmission. Nevertheless, compared with the traditional radio, underwater acoustic communication is greatly affected by poor conditions, such as absorption, scattering, multipath interference and Doppler effect”[9]. While I think the authors are perhaps exaggerating this point to justify their experiments somewhat, it does offer an alternate avenue to wireless communication with a device while it is operating un- derwater, but it would require the development of an ultrasonic modem of some description. Alternatively we could just add a plug to the device for a physical wire for external control/data transmission. While this goes against the design of an AUV as being untethered, it would also act as a lifeline to retrieve the device while minimising risk, even when testing in controlled bodies of water.

## Previous works

### Amateur Level

The YouTuber Brick Experiment Channel[3] wrote a series of blog posts[4] outlining the build process for the 4*th* iteration of their remote controlled submarine.

The posts are detailed covering the steps of building this submarine with reasoning behind the design decisions along with a critical evaluation of each step. This version is using a large syringe to bring water onboard the vehicle to control buoyancy, and is driven by two motors: one for propulsion, and one for direction. The sensors incorporated measure pressure to accurately gauge depth, and a laser to measure distance from the bed of a body of water.

The blog also raised communication as a consideration: While the aim of this project is to build an autonomous vehicle, it would be nice to communi- cate with it in flight. The problem is this cannot be done easily with wireless frequencies as the higher frequencies that provide the required bandwidth do not penetrate water that well.

### Commercial Level

The company Advanced Navigation makes the Hydrus Drone[21], which claims to be one of the smallest AUVs on the market. The device itself con- trols depth and position with impellers, a 4K camera with AI integrations, and a forward facing sonar. The most important aspect of Advanced Navi- gation’s AUV is how they address the issues around wireless communication. This company has fitted their device with acoustic and optical modems so data can be transferred using audio frequencies and light (presumably sim- ilar to fibre octics).

### Research Level

The Woods Hole Oceanographic Institute[13] draws attention to the use of different AUVs depending on the area of research or environment. They use one of 6 AUV models, which operate at different depths or different functionality. The REMUS[14] can be equipped with varying sensors and is programmed for survey missions. It was also adapted to survey the Delaware river Aqueduct for leaks.

One of the more interesting AUVs this institute uses is the Spray Glider[15]. This description for this device is one that glides through the water without any external thrusters so can be travel for weeks at a time. The device uses internal bladders to control buoyancy, and is set to navigate preset paths equipped with varying sensors. What is interesting about this device is the lack of external propulsion seems to make this device more passive, and can therefore operate at lower voltage.

The UK National Oceanography Centre has details on the challenges around an designing an AUV[12]. For power and propulsion it highlights the lack of oxygen for internal combustion makes it necessary to use batter- ies for power and notes the difference in speed between AUVs and surface ships. For navigation it highlights that because GPS can’t penetrate the top few mm of water, the reliance on other techniques is required. It suggests an approach called Dead Reckoning which gets the speed of the AUV by measuring the Doppler shift from the sea bed for relative speed and using a gyro to measure the heading. It highlights that navigational accuracy is vital to survey missions.

**Design**

## Domain

AUVs have applications spanning multiple domains both in scientific ex- ploration as well as commercial enterprise. For science they are used for surveying coastlines and ocean beds, sampling in remote or hard to reach locations and exploring the depths to name a few applications. For com- merce they can de deployed to monitor gas and oil pipelines, drills, survey potential resource sites, and monitor ship hulls etc. The deployment of AUVs over physical diving or manned equipment offer more benefits when compared to the cost and safety. AUVs can potentially spend longer under- water depending on the design and payload they are equipped with and the size of the power source, and can gather far more data than a diver with far less risk to human life.

For this project, the aim was to develop a low cost, open source AUV that will be targeted towards the scientific exploration domain. Since this domain still covers a multitude of deployment scenarios (coastal surveys, lake/ocean floor mapping, deep sea exploration) spanning countless research applica- tions including water quality/temperature, imaging/mapping, ecosystem ex- ploration etc., this project will initially target scenarios that require an AUV to more likely to dive to shallower depths such as coastal surveys. Since the sensor payloads vary depending on the research requirements, the plan is to deliver the AUV shell that can perform the basics of autonomous traver- sal of a body of water relying on sonar technology for obstacle avoidance which can be easy to extend relying on the *I*2*C* bus for communication of connected modules. This can be extended by the researchers deploying the AUV either by integrating with the main electronics driving the device, or by making their own sensor payload independent to the main electronics.

## Technologies

### Microcontrollers

Because this project is something to be deployed to remote environments, the microcontrollers required should be efficient and low powered. To achieve

this I chose the ATTiny 1627 microcontroller mentioned previously in this report. This micro controller has been chosen due the fact it is feature rich with low power requirements as the device itself can operate from 1.8V to

5.5V and the efficiency can potentially be extended using the devices sleep settings. The full list of features for this microcontroller can be found in the appendices 1.2. Because this chip is extremely small, with the 24 pin version only being available in a QFN/VQFN (Quad Flat No leads) packaging, it was necessary to develop a simple breakout board to make this chip breadboard compatible. This required a small amount of learning how to design PCBs, but using KiCad[16], this proved relatively easy for such a simple design, and after manufacturing costs and MCU costs this proved economical as

each board cost *≈*£8 which is a price comparable to the popular Node 8266

MCU used during the Internet of Things module. The designs for this board can be seen in the appendices 1.3

While device packs such as the megaTinyCore[18] would allow the de- velopment of Arduino code on these microcontrollers, the chosen approach was to use “bare metal” C code and directly use the core AVR libraries to interact with the various registers. This is because, while Arduino as a level of abstraction has various API methods that make development a bit easier, it comes with a loss of being able to easily configure many of the device pe- ripherals to be used in specific scenarios, and some of the peripherals (such as Timer Counters) are already configured and used for some functions such as delay. By relying on peripherals and ISRs(Interrupt Service Routines), the code can be developed to be more event driven than running code in a main loop. ISRs work by raising an interrupt flag on the peripheral once a condition has been met. The flag triggers any code inside an ISR vector to be run. The design for the modules will be several modules interconnected using the *I*2*C*/TWI protocol in a star topology where only one device is the Host and the rest are Clients, and are designed in such as way that the pseudocode for each of the modules found in the appendices 1.5 can be used as a starting point ot develop the bare metal C code to run on the microcontrollers.

### AUV Body

In order to keep the AUV as accessible as possible to any institutes or re- searchers looking to develop one, the AUV hull and internal fixtures has been designed to be 3D printed. This negates the requirement for users to need to get the parts fabricated in materials that require special tools or specialised knowledge on how to work with them, or that are costly such as metal or fibreglass or carbon fibre. Once the initial investment of a 3D

printer is overcome if not already available (*≈*£400 for one that prints a

variety of materials), the cost in filament is extremely low in comparison. This approach does require further research and experimentation to deter-

mine the best combination of filaments or materials to print the components in, along with any potential post-processing to increase the suitability and durability of the parts. Since it is common knowledge that often electronics and water do not mix, the design used magnets to connect moving parts on the inside to the outside of the vessel (propellers), as this removes the need to try and have a hole with enough clearance to freely rotate a shaft, while keeping the vessel waterproof.

### Navigation

For navigation and obstacle avoidance this project employs the use of Sonar. Sonar specifically for underwater applications on the consumer level is ex- tremely hard to come by but there are some options available such as the Ping2[29]. while these are feature rich (using multiple frequencies for accu-

rate sonar imaging), they are quite expensive (*≈*£400), and future availabil-

ity could be questionable. To this end, the original plan starting this project was to try to develop a custom sonar module using transducers rated for underwater use[27] that would operate by providing a distance in cm upon request. The requesting trigger would be by command via the *I*2*C* rail. The distance would then be available on the next *I*2*C* read request. This could have also been developed further to having an additional pin to no- tify the host device that the distance is ready to be requested, but really would be unnecessary given how fast the speed of sound travels, particularly underwater ( 1500m/s).

This however proved to be a fruitless endeavour which I can only surmise was down to a combination of inexperience with analogue/audio electronics and not meeting the voltage requirements of the transducers (160Vpp). This ended up using significantly more of the planned time than expected and in the end depended on the use of prefabricated ultrasonic rangefinders rated as “waterproof” (JSN-SR04T[32]). These devices also works in a similar way to the HC-SR04[7] where the trigger pin is pulled high for a minimum of 10µs then released which triggers the transmitting transducer to pulse, and pulls the echo pin high. Once the receiving transducer receives a signal, the echo pin is pulled low and the time between its high and low state can be used to calculate the distance using the formula:

*distance* =

(*time ∗ speedOf Sound*) 2

The division by 2 in this formula is because we only need to calculate the distance to the object, and the pulse measured is the time it takes the ultrasonic waves to reach the object and then return.

### Mechanical

The responses of the sonar modules incorporated into the design of this AUV are used to drive actuators to respond to the physical environment. This includes having the ability to dive and rise, move forward, backwards, and rotate to a new heading to avoid obstacles. With the aim of keeping this device low cost, the goal was to attempt to use inexpensive, readily available parts (stepper and toy motors). The Stepper motor was chosen as it provides precise control over some parts (the depth controller) and can also provide enough torque to physically operate them (depth controller is a large syringe connected to a threaded shaft to convert the rotary force into a linear motion). The main propeller was also quite large and so benefited from this additional torque. The toy motors chosen to turn the smaller side motors as these would benefit from the raw speed they can provide.

# Implementation

*I*2*C*

The main lynch pin of this project is the *I*2*C* protocol used to connect all the individual modules of this vehicle. There are a few different registers on the MCU that require writing to in order to set up the protocol correctly depending on if you want to configure it as a host, a client, or both and numerous considerations when developing a library to handle this protocol effectively as the bus used for communication can enter into a variety of different states.

First, the device data sheet[20] provides a very good overview of what is required to initialise the protocol and use it for effective communication. The initialisation procedures and further instructions are many pages long as it tries to be as thorough as possible to cover all the different scenarios (covering read, write operations, bus idle and error states and arbitration rules and how to handle these along with various interrupt flags that can be raised). This was the original point of reference for this protocol which was used to develop my own library from scratch (appendices 1.6) with the library code split into code specifically for the host device and client device depending on the specific use case of each MCU. For the host (appendix 1.6.1 and 1.6.2), the initialisation process involved setting the baud rate, setting the bus status to ‘IDLE’, and because I decided to use the peripherals “Smart mode” to save myself from having to manually send ACK and NACK bits after each byte is transmitted/received, the MCTRLB register has to have the relevant ACKACT configuration set, and then the protocol is enabled by writing to the MCTRLA register. Using the protocol was theoretically simple, with commands initiated by the host. The host issues a command to a connected client device by first sending the address by writing to the MADDR register (7 bits) which is shifted left by 1 bit as the first bit is used to indicate if the operation is read(0) or write(1). Once the client has responded with ACK (acknowledging it has received the request and is ready to receive more data), either the host transmits data until the client sends a NACK bit (indicating it can’t receive or won’t accept any more data), or the client sends data till the host sends an ACK or NACK bit.

The client code (appendices 1.6.3 and 1.6.4) initialisation is slightly sim-

pler. For this, we simply set the address of the client to the SADDR register (7 bits shifted left by 1 bit), with the first bit indicating if the protocol should respond to open broadcasts on the bus (messages addressed to 0x00), and also the SCTRLA register is where the protocol is enabled and the Interrupts are enabled. The client uses interrupts and callbacks to handle the opera- tions. The ISR code first checks the if the Address or Stop interrupt flag (APIF) is raised which if this is the case, it will check the address match flag (AP). If AP flag is raised this is the Address match and so it would send an ACK bit (by clearing the flag and setting the SCTRLB register to a RESPONSE configuration). If the AP flag is not raised, this indicates a stop command was received (after the host received a NACK) and so the SCTRLB register is set to a COMPTRANS configuration to end the trans- action and release the clock (which while controlled by the host could still be held by the client in a method known as ‘clock stretching’), forcing the bus state to enter an IDLE state. The next interrupt checked is the DIF (Data) which if raised it indicates that there is data ready to be read in the SDATA register (data received from the host).

This implementation of the *I*2*C* protocol is very basic but was still very difficult to develop. It does not handle issues with arbitration, and occa- sionally the code would get stuck on a blocking while loop that was waiting for either the bus to enter a specific state, or an interrupt flag to be raised. If the bus entered into an Error state, this would not occur and during the initial development of this library, this was indeed the case. While this suf- ficed for the midterm prototype, seeing this is the lynchpin of the software on each MCU (as without communication the modules would do nothing) this was deemed unfit for purpose for the final product. However, the more I tried to develop this library, the farther from working code I got.

At this point, since time was short due to other aspects of the project (the unsuccessful Sonar Module), I decided to run an experiment utilizing the megaTinyCore[18] Arduino device pack and its version of the popular Wire package[19]. Using Arduino code I developed a simple circuit of inter- connected microcontrollers and LEDs where the host device could set the colour of the RGB LEDs on the client and the clients could send a colour for the host (thereby testing both read and write operations). This worked flaw- lessly, so I decided this would a be good library to use as a basis. However the toolchains I was using (released by microchip) are not capable for com- piling C++ code for 8 bit MCUs, so I had to first of all remove references to other libraries (Wire extends Arduino’s ‘Stream’ class) which removed the ability to send characters over this protocol as this allowed for the interpre- tation of them, and also convert this library to C. I could have written the project in Arduino using this device pack, but as mentioned previously, this would remove the ability to easily configure the MCU peripherals which I needed the ability to do.

This new implementation (I named CWire since it was the Wire library

rewritten in C), while having *some* limitations (only able to send hex or integer values), worked flawlessly and there are a couple of differences be- tween this implementation and my own. The main difference is this version does not use Smart Mode to automatically send ACK or NACK bits, but controls this as necessary in the code by writing to the MCTRLB register. Second, it uses arrays to store data received and to be transmitted so as soon as the DIF flag is raised for example, it is cleared from the appropriate data register almost immediately and temporarily stored until read properly by the host code implementing this library. This aids in freeing up the bus state almost immediately. This library also combines the host and client code which while in this projects design is not necessary as only one device will be designated as host, it means I only have to maintain one library as opposed to two, and this can be exported as a stand alone entity to be used in other future projects using MCUs in the same family. Finally, this code has far more robust error handling for the different bus states, and it also handles arbitration by throwing an error, and cases where the bus is busy it will wait till it is idle before transmitting. The full code for this library can be seen in appendices 1.7

## Depth Controller

The depth controller in this AUV is an interesting area. Following similar design principles to The Brick Experiments remote controlled submarine, the depth is controlled using a large syringe drawing in water from outside the AUV to modify the overall mass of the vehicle affecting the bouyancy. The first design challenge was pushing and pulling the syringe. To accomplish this I designed and printed a housing for the syringe which are included in the AUV CAD drawings in appendices1.8. The housing connects the syringe plunger to a threaded shaft driven by a stepper motor. In order to prevent this from trying to over screw at either end (which could potentially cause physical damage) at either end I created a ‘buffer’. This is a small metal plate with voltage running through it (generated from one of the pins on the MCU to keep it at a safe current). The nut that traverses the thread has two wires attached connected to the MCU so when they touch each end of this ‘buffer’ we can determine that we have reached the end of travel and adjust the position variables in code accordingly. Because these detection wires are floating waiting to detect a voltage I enabled the internal pullup resistors on these pins as without this the pins would often not correctly detect the voltage and the pins have interrupts set to trigger on rising voltage changes. This buffer worked well, but because I did not want the detection pins to potentially sit at a position where it could be continuously drawing power, once detected the position is set to wind itself back to just before the buffer plate is touched. The complete code for the depth controller can be seen in

appendices 1.9.10.

The Depth Controller aspect of this project is directly related to the ballast calculations that are necessary to perform to ensure the AUV is correctly weighted. Bouyant force[34] is the upward force applied to an object against the gravitation downward pull. If the bouyant force is greater than the weight of the object in fluid, the object will float[17]. In order to calculate the required weight to be added to the AUV I have developed a simple python notebook to perform the calculations, which can be seen in the code repository[28].

*Fb* = *pV g*

Using this calculation we can work out the bouyant force applied to the AUV. First, we require the density of the fluid to be displaced (*p*). While AUVs are typically deployed in the ocean, where salt affects the density we are testing in plain water where the density is known to be approximately 998kg*/*m3. Second we need the volume (*V* ) of the fluid that will be displaced (the volume of the AUV). This is easily obtained from the CAD software and is 0*.*00762mm3. Finally we need the gravitational acceleration(*g*) which is known to be 9.8. This calculates the Bouyant force to be 74.676N. In order to determine if the AUV will float, we also need to know its weight (in N) which can be simply calculated.

*N* = *mass × g*

The weight of the AUV is approximately 39.2N. Since this is less than the bouyant force, the AUV without ballast will float. For this AUV to be able to dive and raise effectivly without the use of propellers or fins/rudders, it would need to be slightly positively bouyant. This would cause additional water drawn in through the syringe on the depth controller to change the AUV weight so it would fall into negative bouyancy and therefore start to sink. Releasing the water should then cause it to go back to slight positive bouyancy causing the AUV to rise. This is acheived by adding ballast. Since the target bouyancy is 0.1N we need to perform the calculation *b* =

*Fb − W −* 0*.*1 which is *b* = 74*.*676 *−* 39*.*2 *−* 0*.*1. *b* is therefore the additional

weight we need to add as ballast. In order to get the required mass, we simply reverse the mass to newton calculation.

*mass* = *÷bg*

This equation would give us the overall target mass of the AUV which is calculated as 7.61(2d.p). To get the mass of ballast required we take off the AUVs current mass which is 4kg. This means we require 3.61kg of additional mass.

# Evaluation

The testing and evaluation of this project can be split into two parts: Soft- ware/hardware (electronics), and mechanical (meaning the testing of the 3D printed parts and mechanical solutions). While I am an advocate of test driven development, this approach is not the most appropriate for micro- controller programming as the code is very dependent on the hardware, or specifically the state of the bytes in the registers of the hardware. While it would be theoretically possible to mock these states in some kind of sim- ulator, there are some actions that also happen automatically by reading from or writing to certain registers such as interrupt flags being cleared by reading/writing from data registers. Test driven code using mocked simula- tions that don’t account for this could therefore be inaccurate when running the same code on the actual MCU. While unit testing *is* possible for MCU programming, there is no official methodology or libraries provided by Mi- crochip (AVR manufacturer), and forums[6] tend to be of split opinions as to whether this is practical or not owing to the code being tested for simulators not the actual hardware, but with some people stating they do use google test libraries and simulators.

Because of this I chose not to use a TDD approach developing this project. Instead code was manually tested on the devices manually with any unexpected behaviour debugged with the use of the hardware debugger available using the programmer (ATMEL-ICE[26]) programming the device via the Unified Programming and Debug Interface (UPDI) which allows stepping through the code line by line on the Microcontroller itself. This approach is not without its own issues, particularly when trying to debug issues related to data received from other devices over the *I*2*C* protocol as if a device is sent data while the receiver is in a frozen state due to debug- ging, this could (and did) throw false errors causing the *I*2*C* bus to enter error states. In most cases however this was a very useful tool as it allowed observation of the varying states of the registers at each step of the code execution. When combined with a static code analysis bugs could be traced with relative ease.

The development of the specific software features were manually tested with an attempt to cover as many edge cases as I could think of, but ig- noring scenarios that were deemed impossible to reach, such as the Depth

Controllers buffer hitting the IN and OUT buffers simultaneously is phys- ically impossible. This manual testing method did allow for an iterative approach to the code, where bugs were fixed as observed, but it was time consuming and could not be exhaustive. With more time available it would be worth exploring a TDD approach as there are some aspects of the code that are not reliant on the MCU itself, such as the distance calculations in the Sonar module. Admittedly the testing of the software could have bene- fited from some consideration prior to starting development by writing some manual test scripts but considering I was testing a debugging simultaneously alongside revising the code, this would have interrupted my workflow.

The mechanical aspects of the project (primarily the 3D printed parts seen in appendices 1.8) were also iteratively designed and tested in the same way as, and often alongside, the software. The depth controller, being a part that had moving parts and needed to operate smoothly was an exam- ple of successful iterative testing and designing. The first iteration of this was modelled to house the stepper motor and connect to the threaded shaft but immediately failed its test as this was modelled as a single piece and I did not account for clearance allowing the stepper motor to get inside the housing. Following this design flaw, the part was redesigned to be comprised of individual components that can be put together. This made future ad- justments to the model far easier, for example one issue the part had for a while during development was it would get stuck occasionally while turning while the software was trying to perform either a ‘raise’ or ‘dive’ operation. While it would move again by itself more often than not, the few seconds it was stuck at a position would throw the timings off on a software level (so where the software *thinks* the plunger is may not be accurate), and it put unnessary strain on the motor as it required torque to move past this point. The remedy for this was to allow for larger tollerances between the printed parts. By adding an additional 0.2mm clearance on the thread and nut driv- ing the syringe (0.4mm total), the travel was far smoother and mitigated any future requirements for additional components such as grease.

Another mechanical aspect of this project that could have benefitted from earlier testing stages was the mechanism to drive the propellors. This aspect of the project was a bit of a challenge as a hole for the any shaft that provided clearances to allow free movement would also allow the ingress of water. To this end, I decided to attempt a solution using magnets to drive propellers seperated by a physical barrier. In this design, the motor with a shaft/gears to step up the speed are on the interior of the vessel, while the propeller itself is on the outside. Connecting these are 2 flat disks perpendicular to the physical wall seperating each other but parallel with each other, with equally spaced magnets with alternating polarities. The issue with this design is the motors do not have enough torque to physically turn these disks when they are magnetically attached directly to the wall seperating them. The only assumption for this is the surface area of the

disks offers too much friction for the parts to move freely. On the rear motor this was mitigated by adding a small washer in between reducing the contact area, while still keeping the magnets attracted to each other enough to drive the propeller. The side motors however relied on small toy motors that provide plenty of speed but next to no torque, and while the main propeller was supported by a prop guard, the side motors were designed to be detachable to reduce breakage during travel, and so could not be solved by inserting a washer between and so in this iteration of the final solution, the side propellers will not turn if attached. This is one instance where failure to properly test a solution prior to incorporating it fully into the project was a failure as this could have been identified earlier, and the designed revised before it was too late, while keeping the core concept (magnetically driven) and the inexpensive toy motor in place without having to look at more expensive (in terms of both economically and power consumption) motors that have more torque.

Finally, the waterproofing of this project was one aspect of the mechan- ical aspects of the project that did go through proper testing and rounds of iterative changes. The test scenario was to print the vehicle, assemble it and submerge it to check for water-tightness. The first test just the AUV sec- tions with only TPU printed gaskets in between and was largly unsucessful. Water seeped in from every joint, indicated by several streams of bubbles across the AUV. As the sections appeared to be the weak point, the next test was the same again but using silicone sealant to seal these sections. While this test resulted in no bubbles between the sections, water was still getting in to the interior. On closer inspection of the AUV body, there are small holes in the surface of the hull, particularly on the more complex areas such as the side motor pods. This is a negative aspect of using 3D printing to make the hull as with incorrect settings in the slicer, combined with the nature of how these parts are manufactured layer by layer can lead to such imperfections. To mitigate this and to save having to reprint everything at a higher resolution (a process with would have taken in excess of 10 days), each section of the hull was coated in epoxy resin and allowed to cure for over 24 hours. Finally, the vessel was reassembled (a process which was now much harder as the tolerances of the model did not include the thickness of the resin layer) and copious amounts of silicone was applied to every layer. This passed the water tightness test, but as it transpires it only really passed this particular ‘dry run’ if you’ll excuse the pun.

The final test for the AUV was testing the unit as a whole, in a body of water. The initial challenge here was finding a body of water large enough and suitable enough to adequately test. This was found by putting a post on Facebook, in the local community groups to see if anyone had a pool I could borrow for which one individual granted me access to his pool. This test failed. The after placing the AUV in the water, and turning it on, nothing appeared to happen, and eventually it seemed the bow of the AUV was

gradually sitting lower in the water than when it was first put in indicating it was taking on water, which was confirmed by the weight of the AUV when lifting it out. Unfortunately there are too many variables to determine what went wrong in this test. First, because the AUV is a sealed box, I had no visibility on the status LEDs soldered onto the modules to determine if it was operating correctly. It could be that connecting wires came unattached during the assembly process so nothing was working to start. The water ingress, while a problem on this test, highlights that the process to make the vessel water tight is too prone to error and needs further exploration, experimenting with different combinations of materials and printer settings. Another contributing factor is the prop guard at the rear of the AUV was broken in transit which could have provided an avenue for water to seep in. Unfortunately, this test only occured with a week to go before the final deadline and there is physically not enough time to refine, reprint, and resolder what is required to replace any potentially broken parts in that space. While there are lessons to be learnt from this test (and that is the point of tests), there is not enough time to enact on them in the scope of this module.

# Bibliography

1. avrdude. Avrdude documentation. 2023. url: https:// avrdudes. github.io/avrdude/7.2/avrdude.pdf.
2. I. Bus. Fast mode. url: [https://www.i2c-bus.org/fastmode/.](http://www.i2c-bus.org/fastmode/)
3. B. E. Channel. Brick experiment channel. url: https://www.youtube. com/c/BrickExperimentChannel.
4. B. E. Channel. Brick experiment channel. url: https@//brickexperimentchannel. wordpress.com/.
5. ecoSub. Ecosub. url: [https://www.ecosub.uk/.](http://www.ecosub.uk/)
6. A. Freaks. Unit test in microchip studio. url: https://www.avrfreaks. net/s/topic/a5C3l000000UcMhEAK/t161267.
7. E. Freaks. Ultrasonic ranging module hc - sr04. url: https://cdn. robotshop.com/media/s/spa/rb-spa-1388/pdf/hcsr04-ultrasonic- range-finder.pdf.
8. gcc. Avr-gcc documentation. 2024. url: https:/ / gcc . gnu . org / wiki/avr-gcc.
9. Z. Hong, X. Pan, P. Chen, X. Su, N. Wang, and W. Lu. A topology control with energy balance in underwater wireless sensor networks for iot-based applications. *Sensors*, 2018.
10. W. HR. Spurv - the first decade, 1973.
11. F. B. Insights. Autonomous underwater vehicle (auv) market size, share & covid-19 impact analysis, by type (small, medium, and large), by application (scientific, research , defense, and oil & gas industry), by propulsion system (electric system, mechanical system, and hy- brid system), by payoad (camera, sensors, intertial navigation sys- tem, and others), and regional forecase, 2021-2028. url: https:// www.fortunebusinessinsights.com/ segmentation/ autonomous- underwater-vehicle-market-105907.
12. N. O. Institute. Autosubs. url: https://noc.ac.uk/facilities/ marine-autonomous-robotic-systems/autosubs.
13. W. H. O. Institute. Auvs. url: https://[www.whoi.edu/what-](http://www.whoi.edu/what-)we- do/explore/underwater-vehicles/auvs/.
14. W. H. O. Institute. Remus. url: https://[www.whoi.edu/what-we-](http://www.whoi.edu/what-we-) do/explore/underwater-vehicles/auvs/remus/.
15. W. H. O. Institute. Spray glider. url: [https://www.whoi.edu/what-](http://www.whoi.edu/what-) we-do/explore/underwater-vehicles/auvs/spray-glider/.
16. KiCad. Kicad. url: [https://www.kicad.org/.](http://www.kicad.org/)
17. L. Learning. Archimedes principle and bouyancy. url: https : / / courses.lumenlearning.com/suny-osuniversityphysics/chapter/ 14-4-archimedes-principle-and-buoyancy/.
18. megaTinyCore. Spence konde. url: https://github.com/SpenceKonde/ megaTinyCore.
19. megaTinyCore TwoWire. Spence konde. url: https://github.com/ SpenceKonde / megaTinyCore / tree/ master/ megaavr / libraries/ Wire/src.
20. Microchip. Attiny1624/1626/1627 data sheet. url: https:/ / ww1 . microchip.com/downloads/aemDocuments/documents/MCU08/ProductDocuments/ DataSheets/ATtiny1624-26-27-DataSheet-DS40002234B.pdf.
21. A. Navigation. Hydrus. url: https:// www. advancednavigation. com/robotics/micro-auv/hydrus.
22. B. News. Remaining debris from destroyed titan sub found on atlantic seabed. Oct. 11, 2024. url: https://[www.bbc.co.uk/news/world-](http://www.bbc.co.uk/news/world-) us-canada-67073535.
23. P. Oceanography. Underwater robots: autonomous underwater vehi- cles (auvs), 1999.
24. U. of London. Module author interviews - project templates. url:

NEEDED.

1. Ofcom. Uk frequency allocation table. Aug. 2022. url: http : / / static.ofcom.org.uk/static/spectrum/fat.html.
2. RS-Online. Atmel-ice. url: https:// uk.rs- online.com/ web/ p/ chip- programmers/ 1306122 ? cm\_mmc=UK- PPC- DS3A- \_- google-

\_- 3\_UK\_EN\_Semiconductors\_Chip+Programmers- \_- Microchip+-

+1306122+ - +ATATMEL - ICE- \_- atatmel+ ice& matchtype = e & kwd - 66984139136&cq\_src=google\_ads&cq\_cmp=12689163037&cq\_term= atatmel%20ice&cq\_plac=&cq\_net=g&cq\_plt=gp&gad\_source=1& gclid=CjwKCAjwreW2BhBhEiwAavLwfFbf23eDzecxRUQwEWtZWzJqNBjkdBgyD1kSa7oO1rrOHBVb BwE&gclsrc=aw.ds.

1. M. Pro. Ultrasonic sensor. url: [https://www.farnell.com/datasheets/](http://www.farnell.com/datasheets/) 3771814.pdf.
2. RichardSefton. Auv repository. url: https://github.com/RichardSefton/ AUV.
3. B. Robotics. Ping2 sonar altimeter and echosounder. url: https :

//bluerobotics.com/store/sonars/echosounders/ping-sonar- r2-rp/.

1. RoS. Robot operating system. url: [https://www.ros.org/.](http://www.ros.org/)
2. R. Sefton. Project template. url: NEEDED.
3. P. Supplies. Jsn-sr04t v3.0 waterproof ultrasonic range finder. url: https://protosupplies.com/product/jsn-sr04t-v3-0-waterproof- ultrasonic-range-finder/.
4. P. Visconti, G. Giannotta, R. Brama, P. Primiceri, A. Malvasi, and

A. Centuori. Feature, operation principle and limits of spi and i2c communication protocols for smart objects: a novel spi-based hybrid protocol especially suitable for iot applications. *International Journal on Smart Sensing and Intelligent Systems*, 10:262–295, 2017.

1. P. Walker and K. Wordeman. How to calculate the bouyant force of a floating object. url: https:/ / study. com / skill/ learn/ how - to - calculate - the - buoyant - force - of - a - floating - object - explanation.html.
2. E. Williams. *Make: AVR Programming*. Maker Media Inc., 2015.
3. R. B. Wynn, V. A. Huvenne, T. P. L. Bas, B. J. Murton, D. P. Con- nelly, B. J. Bett, H. A. Ruhl, K. J. Morris, J. Peakall, D. R. Parsons,

E. J. Summer, S. E. Darby, R. M. Dorrell, and J. E. Hunt. Autonomous underwater vehicles (auvs): their past, present and future contribu- tions to the advancement of marine geoscience. *Elsevier*, 352:451–468, 2014