Multi-purpose CAN monitor and single-pair ethernet protocol shift for ROV

Johannes Davidsson

Table of contents

1. Terminology and Acronyms

Term **Acronym** **Description**	**Term** **Acronym** **Description**	**Term** **Acronym** **Description**
Single Pair Ethernet	SPE	Pair of copper wire used for communication.
Communication Area Network	CAN	A communication bus commonly used in the automotive industry
Power over Ethernet	PoE	Ethernet cable that replaced one or more wire pairs to deliver power
Power over Dataline	PoDL	Deliver power with the same cables that deliver data
Transmission Control Protocol	TCP	Ethernet Protocol that has safeguards to make sure the data arrives as sent
User Datagram Protocol	Udp	Ethernet Protocol that sends data with no safeguards and expects no response
Remote Operated Vehicle	ROV	A vehicle that is operated from a remote location
Raspberry Pi	RPI	Small linux compatible computer picked for this project
General Purpose Input/Output	GPIO	An input/output with programmable ports
Physical layer device	PHY	A device that handles a transmission
Power Sourcing Equipment	PSE	Supplies power
Powered Device	PD	Requires power
Serial Peripheral Interface	SPI	serial communication interface
Research and Development	RnD	A department for research in a company
Common Mode Choke	CMC	A commonly used filter to reduce noise

2. Introduction

Ocean Robotics International AB develops underwater Remote Operated Vehicles (ROVs), tailored to the needs of the customer. These can be ROVs for exploration, cleaning, leisure or whatever task you need done below the surface. However due to the underwater nature of these vehicles makes it necessary for them to survive in a hostile environment. Such adaptations tend to be expensive, bulky and by necessity simple. As such any modifications that enable smaller/fewer cables and less hardware to be used in the depths can be an advantage. This paper explores the possibility of changing the communication between the nodes of the vehicle.

Currently these ROVs use a Controller Area Network (CAN) bus to communicate within the nodes of the vehicle. The CAN bus also uses two dedicated wires not unlike Single Pair Ethernet (SPE). There is however a big difference in protocols. Usually one of the benefits of the CAN bus is that you can connect devices in a daisy chain without a lot of effort. The same cannot be said for the SPE which connects only two devices directly or multiple using a switch. The nature of Ocean Robotics ROVs requires a central hub where all the nodes are directly connected, where a node is any external device such as a thruster or a camera. This setup makes a switch to SPE theoretically possible without much hardware overhaul. The actual sought benefit however is that there is a possibility to deliver power along with the data or more commonly called Power Over Data Line (PoDL). If the switch would enable Ocean Robotics to power devices with the same cables used as data transmission it could cut the number of required wires up to half between devices. This could save a lot of hardware costs to the pods which today require bulky ports to support every cable. A smaller benefit would also be the connectivity to PCs when the ROVs natively use ethernet.

In addition Ocean Robotics needs a platform that can be used for RnD with their current system. It will need to be able to communicate with a PC and connect to a CAN bus moreover be open to further modifications. In addition a paired program will be developed to work with the platform.

v.03

2.1. Scope and research questions

In this section we ask what and why we are examining this avenue so the goal is clear.

- How will SPE cope with a system that has many nodes like the ROVs?
    - As SPE communicates directly between two nodes unlike the CAN bus which communicates with all nodes at the same time, maybe there are some additional requirements?
- Which hardware is available today?

    - This is an emerging technology and not yet well developed for all purposes, what can currently be bought from companies working in this field?

- What are the limitations of Power over DataLine (PoDL)?
    - Are there some inherent problems with PoDL? What is possible and what is not?
- How are the currently used microcontrollers going to be impacted?
    - The current microcontrollers use the CAN bus, are they capable of both? Is there a need for additional hardware or maybe there is a better fitting one?
- How much software changes are necessary to accommodate SPE?
    - How big of a change is necessary? Will there only be a need for the communication code to be changed or a bigger overhaul?
- Would there be a performance difference?
    - SPE on paper does come with better performance, applied to a system with many nodes may however come with some complications.

In no particular order I shall aspire to answer these questions and maybe even some more.

2.2. Expected results

The device is going to be able to work with the current system to monitor and send messages over the CAN bus. It needs to be able to be tailored for whatever Ocean Robotics needs down the line as well. In addition Ocean Robotics wants to explore changing the communication system from CAN to SPE in order to potentially save costs and improve performance.

2.3. Boundaries (Avgränsningar)

First this device is intended for use in a lab environment

- The device will not be adapted for the humid or wet environments
- It will not be capable of using Single Pair Ethernet
- The pc software will be developed for windows only
- The prototype will be based on a Raspberry Pi

As for the SPE theory, No actual SPE circuit will be tested

2.4. Similar systems

While this paper will focus on SPE there are other systems that can be adapted that have similar functionalities and could be explored. This is not an exhaustive list but could potentially be adapted for use in Ocean Robotics ROVs.

2.4.1. PoCan

This system experiments using PoDL over a CAN bus. (Tatsuki Matsushita) It is as I understand it not fully developed and would require a lot of RnD to embrace.

2.4.2. Powerline Communications

Also called Homeplug as a family name when used in a building. It is a developed standard and on the surface very similar to SPE PoDL systems. It uses the copper cables already present for AC power in buildings using the power wall sockets. This system then adds a signal on top of the power which travels to another power socket and is filtered out. It does suffer from interference with devices that use Radio Frequencies and from similar devices. Even devices such as microwave ovens can impact the signal. This technology does seem with some modifications applicable according to this article (List). Where they use a modified plug in order to achieve a viable connection and power the plug using the DC power available. They also claim a 100MBit/s connection which would be sufficient for most applications.

2.4.3. Discussion

It is unlikely that there is a company that already has made a system that can be incorporated by Ocean Robotics into their vehicles. Furthermore finding any commercially available devices at all seems slightly premature as no vendors are currently selling switches for these kinds of systems.

What is being advertised is the standardization of the technology such as (SPE Industrial Partner Network). They are working with companies to develop the infrastructure with little information about upcoming releases of systems. Companies like (Single Pair Ethernet SPE – The Infrastructure for IIoT | HARTING) are partners of the previously mentioned network. They advertise the possibilities of the technology but little about the availability of the systems.

2.5. Ocean Robotics requirements

To use this system Ocean Robotics would like to see how much power could be delivered along with the data. Preferably 200W, which is not something that I could find any reference to. In addition, since the standard (IEEE Std 802.3bu) does only support 50W, finding a finished solution is unlikely.

3. Background

The underwater ROVs operate at great depths and to get to places such as the bottom of the ocean Ocean Robotics ROVs use a tether from the mothership. This tether provides the power necessary to operate the vehicle for extended dives. An optical cable runs along the tether which serves well for communication between the vehicle and operator. The tether would not get better performance by using any version of ethernet as the optical cable is much better suited for long distances.

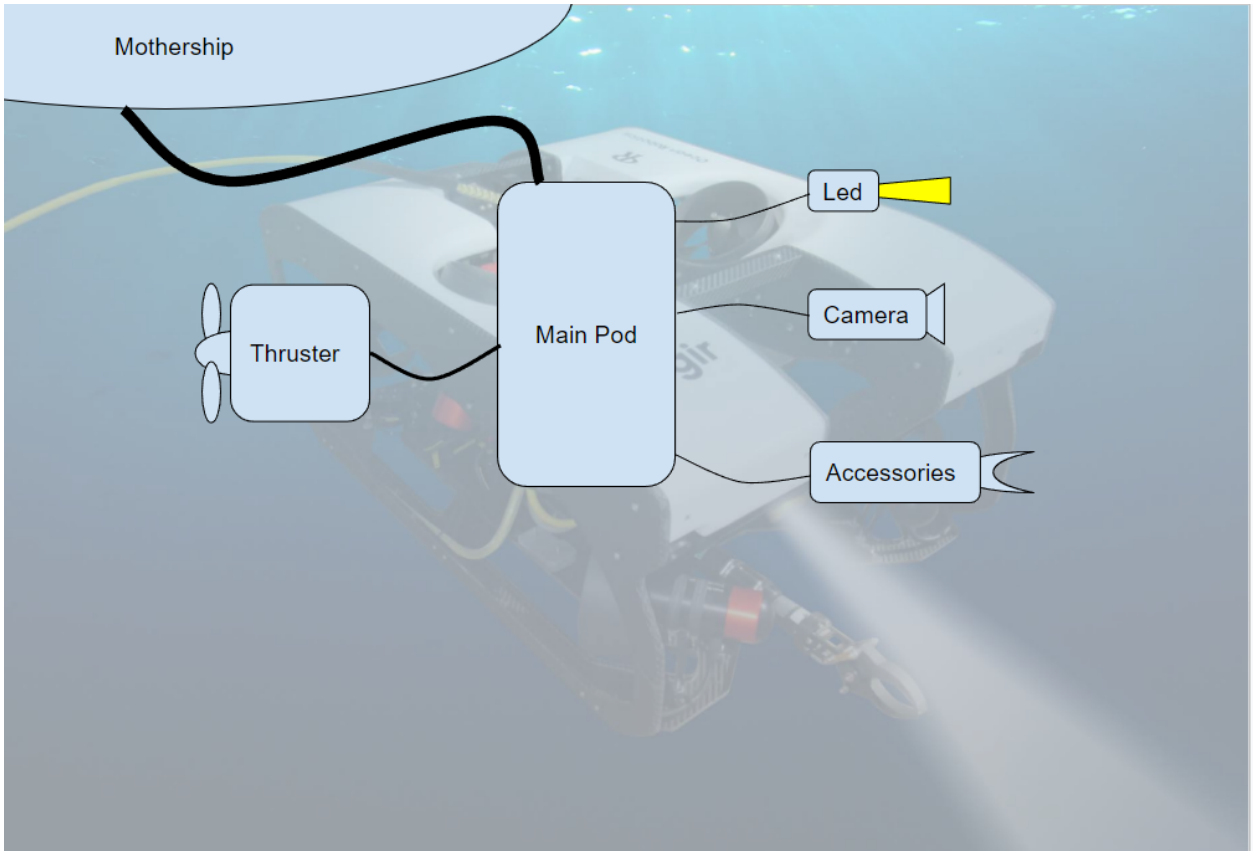


Figure 1 - Simplified ROV diagram

Ocean Robotics ROVs today use a CAN bus to connect its nodes and communicate between themselves. One big benefit of the CAN bus is the ability to daisy chain nodes to save space and simplify the design. In this case all of the cables are connected to the main pod of the vehicle, not unlike the idea of a switch used in ethernet communication. As such the daisy chain benefit of the CAN technology is not applicable. This means that a change between CAN and SPE would not need an overhaul of the current cabling of the system. However just because it is possible to use SPE there still needs to be incentive enough to warrant any changes to the system.

The tether connects to a specifically designed pod where most of the electronics is housed. This pod is also where all of the thrusters are connected, along with the lights, cameras, special

equipment etc. Currently the cables used to power the thrusters and other devices also carry the CAN wires in parallel inside of the same cable. Getting rid of the extra wires for CAN would make the entire cable thinner and make the required port smaller in turn.

Preparing a vehicle to travel very deep into the ocean is complex and Ocean Robotics hopes that being able to use smaller cables and ports would reduce costs and complexity. Provided that SPE in turn is a good replacement for their system.

4. Theory

The basics of separating a signal from a power source is pretty straightforward, at least for DC power. Just filter out the data with a capacitor and it is almost done. It mostly holds true for AC power as well, which has been a staple of home networking (Powerline Communication) for many years now. So in addition to looking at SPE there is also the potential of finding more ways to solve the same problem.

4.1. Single Pair Ethernet

Ethernet is at this point a very well established technology that most computers run natively and is not going anywhere anytime soon. In fact the standard has been around for more than 40 years (“Ethernet over twisted pair,”2022). The reason for that is that it is still being developed today, whether it is to add even more speed to the networking of your devices or adding more functionality. It does however always require a direct connection between nodes and as such require separate cables for each device.

Practically the cable that this is based on contains multiple pairs of copper wires twinned together. The number of pairs decides transfer speed but may also have other uses. In industry there has been a shift towards ethernet because of the versatility of the standard. We have as an example Power over Ethernet (PoE) that sacrifices one or more pairs to deliver power instead of data. This is a fairly simple approach that does something similar to what Ocean Robotics wants but requires a minimum of two pairs of wires.

Single Pair Ethernet by itself is exactly what is in its name, an ethernet cable with one single pair of wires. This imposes some limitations such as a data transfer rate of maximum 10 GBit/s (“Ethernet over twisted pair,”2022) but instead has the benefit of using a small and light cable. In the last few years there have been strides to develop power delivery over the same wires. Commonly called Power over Dataline (PoDL) has many applications as it incorporates both the connectivity and power requirements that devices need. Simplifying cabling in an industrial setting has big implications and could save a lot in expenses and maintenance.

4.2. Controller Area Network (CAN)

CAN is a well established technology that has its origin from 1991 and works quite differently from ethernet. Unlike ethernet it connects all nodes using a bus that lets all nodes talk to each other. This means that all devices share one single pair of wires instead of one pair each. This however means that all of the nodes share the bandwidth of the wires which is a maximum of 1 Mbit/s according to the standard. While this speed is significantly slower than ethernet it does require less hardware to use and many microcontrollers support it natively. In addition the CAN system still serves well in systems that do not require much data to be transferred such as

sensors. These systems are found in cars, access systems, production lines and similar systems and would get little benefit from a change between the methods.

4.3. Power over Data Line (PODL)

Cabling any distance can be quite difficult so halving the necessary wires is clearly useful so why isn't most systems using this? The idea is quite old as well, the telephone lines that still cover the country is one such system and it still works fine. Practically it is just an ac signal over an dc power after all.

With the phone analogy we could probably actually hear the problem, interference is a big issue to such a system. A bad connection over the phone could be somewhat tolerated as we are quite good at piecing together corrupted information, missing a word in a sentence will not stop you from having a conversation. The same is not true for data. While you can compensate by sending multiple copies for example, it just adds more complexity to the system

The basic concept of sending a signal along with the power is fairly straightforward. A device requires dc power so the signal needs to be encoded on top of it and then decoded when it arrives. The signal is then encoded and decoded using a capacitor and the power uses an inductor to filter out the signal.

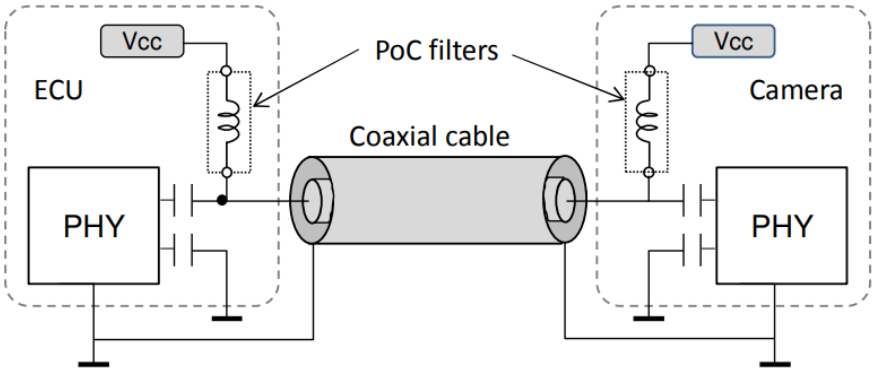


Figure 2 - (Felipe Jerez 2019, page 1) Typical POC implementation.

Above is a figure of a Power over Coaxial circuit, which better explains the basics of the concept. This is exactly what you would have on a phone line. PHY stands for Physical Layer Device and is the device that requires the data sent along the cable. The “PoC filters” that are pointed to, need to be specifically designed to handle the frequencies that the datastream is using in order to filter them out. Otherwise the signal might impact the performance of the device.

4.4. Typical Approach

When it comes to PoDL with SPE there is a standard already established (IEEE Std 802.3bu) which covers this application. It also covers how the data packets are encoded to work together with the Power Sourcing Equipment (PSE) and the Powered Device (PD). This among other things enables the PD to regulate the amount of current sent by the PSE.

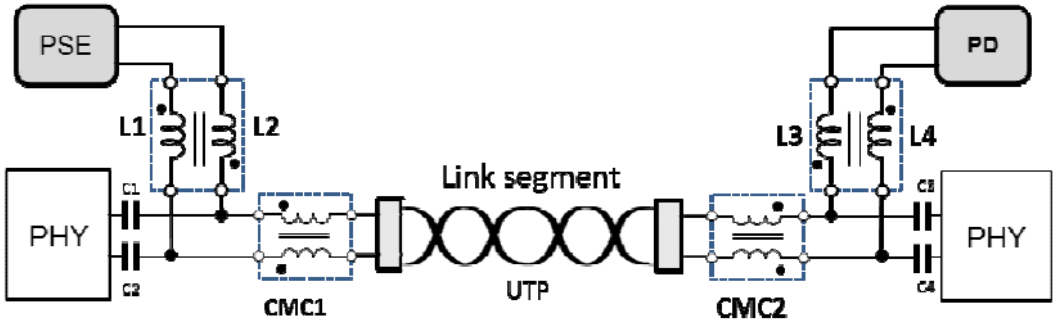


Figure 3 - (Felipe Jerez 2019, page 3) Typical PoDL with twinned cable.

The above circuit is similar to the previous PoC example with the addition of a parallel wire. The addition of a Common Mode Choke (CMC) helps to avoid the noise introduced by the environment. This also helps when multiple cables run in parallel.

4.5 Single Pair Ethernet (SPE) signal

The SPE signal, depending on the transfer speed, needs a cable that supports the frequency that is used. According to (IEEE Std 802.3bp) using the 1000BASE-T1 standard you can reach 1Gbit/s under a length of 15m using a signal that is 600Mhz. Lower frequencies and transfer rates allow a longer cable. A frequency like 600Mhz can be susceptible to interference from signals like tv transmissions or civilian mobile service which can operate in that range. Therefore a CMC is included in a typical setup in addition to any shielding on the cable. In a PODL setup there is in addition to the SPE signal an DC element which is eliminated by a pair of capacitors.

4.6. Power Delivery

The direct current traveling along the spe signal needs additional filters to the CMC unless the CMC is placed after the PODL filters (L1-L4) like the figure below.

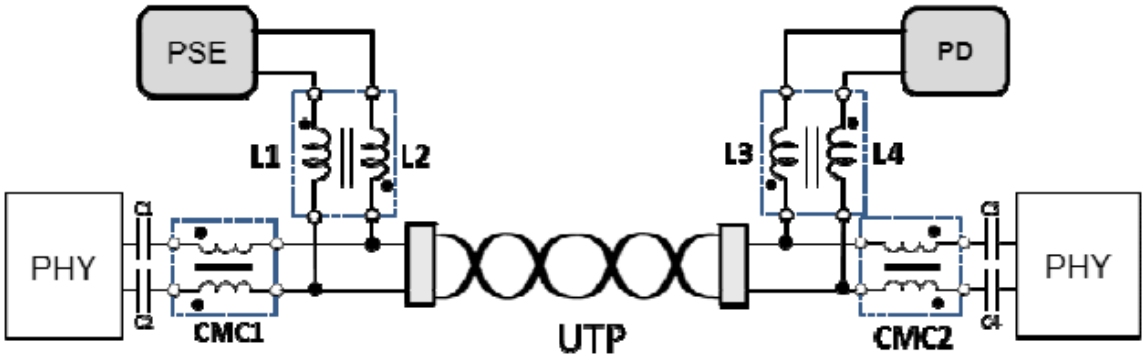


Figure 4 - (Felipe Jerez 2019, page 3) CMC variation

This lowers the requirements on the CMC as not as much current travels through it. The inductors intended to remove the AC elements from the DC power.

The standard (IEEE Std 802.3bu) covers multiple options of PSE implementation if certification after the standard is necessary.

	12 V unregulated PSE		12 V regulated PSE		24 V unregulated PSE		24 V regulated PSE		48 V regulated PSE	
Class	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
$V_{PSE(max)}$ (V)[a]	18	18	18	18	36	36	36	36	60	60
$V_{PSE\_OC(min)}$ (V)[b]	6	6	14.4	14.4	12	12	26	26	48	48
$V_{PSE(min)}$ (V)	5.6	5.77	14.4	14.4	11.7	11.7	26	26	48	48
$I_{PI(max)}$ (mA)[c]	101	227	249	471	97	339	215	461	735	1 360
$P_{Class(min)}$ (W)[d]	0.566	1.31	3.59	6.79	1.14	3.97	5.59	12	35.3	65.3
$V_{PD(min)}$ (V)	4.94	4.41	12	10.6	10.3	8.86	23.3	21.7	40.8	36.7
$P_{PD(max)}$ (W)[e]	0.5	1	3	5	1	3	5	10	30	50

[a] $V_{PSE(max)}$ is the maximum allowed voltage at the PSE PI over the full range of operating conditions.
[b] $V_{PSE\_OC(min)}$ is the minimum allowed open circuit voltage measured at the PSE PI.
[c] $I_{PI(max)}$ is the maximum current flowing at the PSE and PD PIs except during inrush or an overload condition. $I_{PI(max)}$ may be exceeded during inrush or an overload (see 104.4.6.2). Users are cautioned to be aware of the ampacity of cabling, as installed, and local codes and regulations (see 104.8.1).
[d] $P_{Class(min)}$ is the minimum average available output power at the PSE PI.
[e] $P_{PD(max)}$ is the maximum average available power at the PD PI.

Figure 5 - (IEEE Std 802.3bu # 37) Class power Requirements matrix

These are a good indication of what commercial product performance will be like when they are launched. This also suggests that delivering more than 50W requires a more tailored solution.

The wire itself also needs to be able to deliver the current without overheating. This should not pose a problem for Ocean Robotics if they use similar cable gauges as they do today for power delivery.

4.7. SPE Performance

Ethernet uses node to node connections unlike the CAN bus which networks all nodes. Currently the entire CAN bus has a maximum transfer rate of 1 MBit/s which works for the current system. Comparatively the IEEE standard for SPE PoDL does allow for speeds between 512 Kbit/s to 10 Gbit/s. A normal application of SPE would involve introducing a switch that is connected to all devices simultaneously. The switch would then enable traffic between two nodes at previously mentioned speed. However if multiple nodes are communicating with the same node, such as a processing node, this speed will be shared.

4.8. SPE PoDL Application

4.8.1. Discussion

The previous pages explored the theory of an implementation of the SPE technology but now it needs to be applied. The CAN bus uses very little hardware as of now and a change to SPE would definitely be adding hardware to the system not just replacing it. While you can include microcontrollers that do inherently communicate with ethernet there is none with enough ports. The necessity of a switch will add dedicated hardware to the system since it is something that CAN did not need. In addition there will need to be an addition to the pcb of the connected devices in order to make use of SPE. In exchange the cables connecting the devices can be made much thinner which use in turn smaller ports. As few commercial systems are available now it is very difficult to compare cost to savings on such a change.

4.8.1. Basics of designing a SPE system

Provided the power supply is dc makes the process simple, a capacitor is able to completely block such a signal. Otherwise a high pass filter that completely blocks the ac component is required.

The high pass filter that only lets the data transmit is mentioned in the IEEE 100BASE-T1 specifications where 0.1nF is mandated.

A passive low pass filter for the power delivery is made by inductors and needs to be picked after the supply. In addition depending on the device there could be interference there too, such as from a motor. An analogue camera would also require consideration when it comes to the filter, however with SPE the signal could be digitized and transferred using the superior speed.

4.8.2. Schematic of a basic SPE circuit

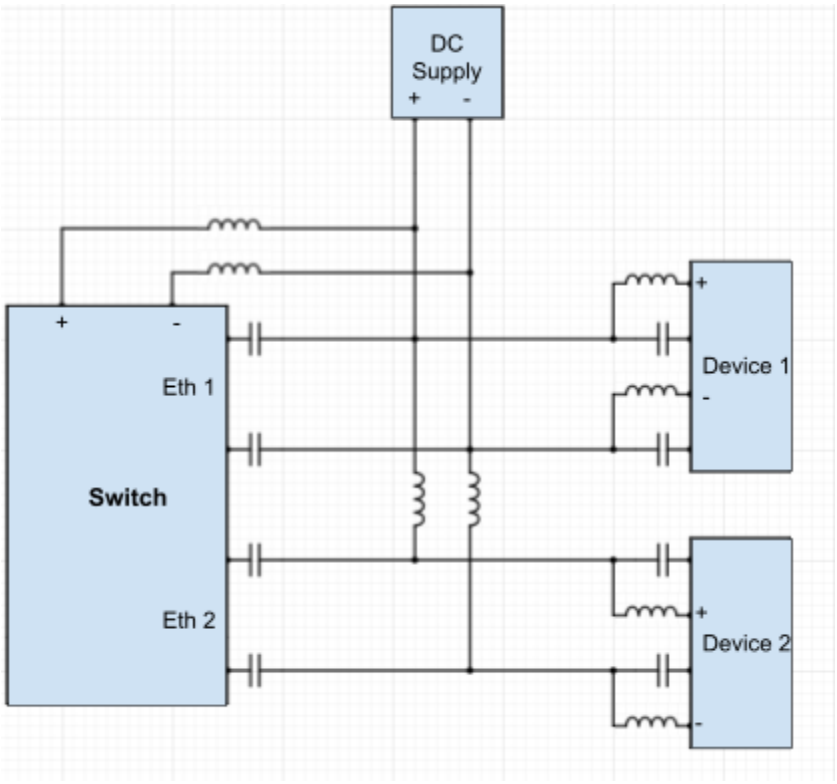


Figure 5 - Simple Ideal SPE circuit

The idea is that this schematic could be expanded to fit the system, as the ROV has differing amounts of attached devices. This would allow one supply to be used for multiple devices with an inductor between each pair in order to separate the signals. An inductor before the power input could be necessary, this is especially true if it is a low powered device. This is not a tested circuit and trying an active low pass filter might be better for low powered applications.

There is also the possibility of interference, but due to the short cables and the environment being far beneath the surface it would be a local source. A thruster could potentially give such interference as its power draw fluctuates when in use. A CMC could then potentially clear some of that interference or a specific band pass filter for the SPE signal.

5. Method

In order to get a baseline on the complicity of designing a power over dataline connection, a simple test that follows the basic theory was done. The test was done by comparing the speed and stability of a two meter long cable in a few cases. A few things needed to be established first before the test:

- Get a baseline of a double pair ethernet cable
- Attach a pair of capacitors at both ends and test without any power
- Run power over the cable and test performance

The experiment will use Iperf to test the speed of the ethernet connection between a laptop and a raspberry pi. To reduce complexity two pairs were cut and four capacitors were used on each side. No low pass filter added due to the load being only a variable resistor.

In this test both TCP and UDP will be tested as transfer protocols. TCP is expectedly more prone to interference as it requires both devices to mutually agree on what data is sent and whether it matches the data received. UDP however simply sends data along the connection and does not consider whether the receiver is ready and does not expect a response. As such I expect UDP to be able to be run at much higher speeds but may still be lower than expected.

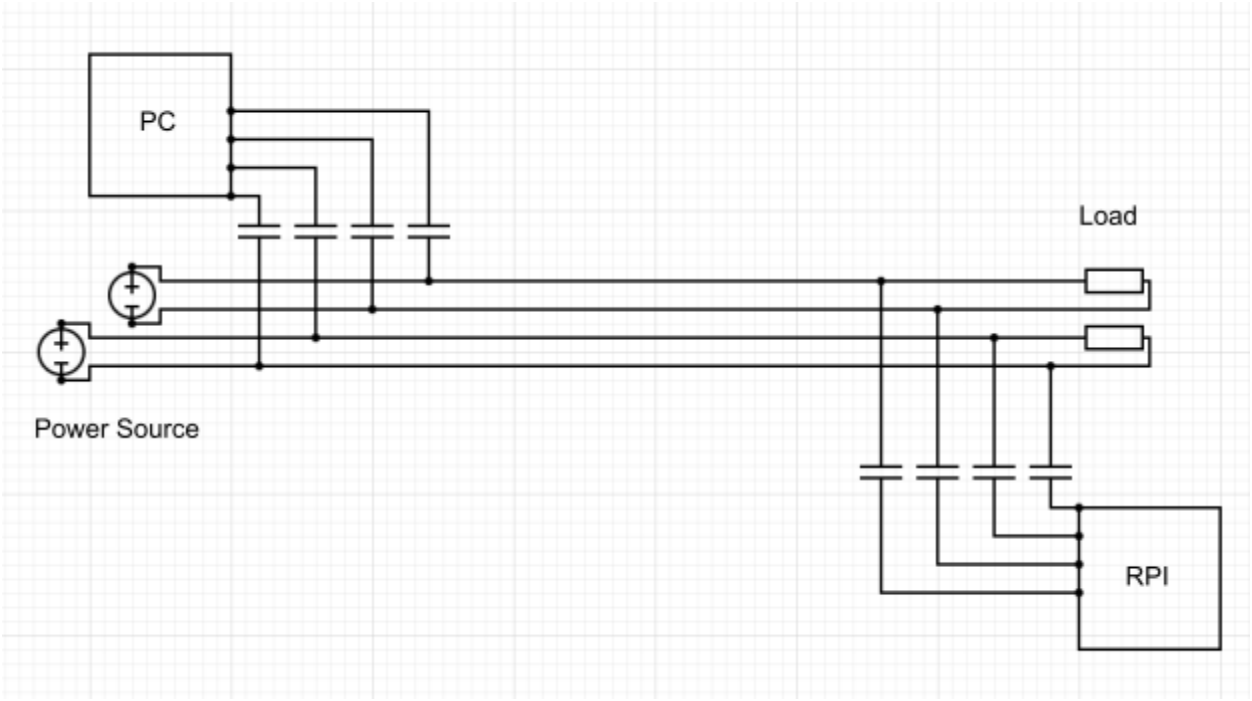
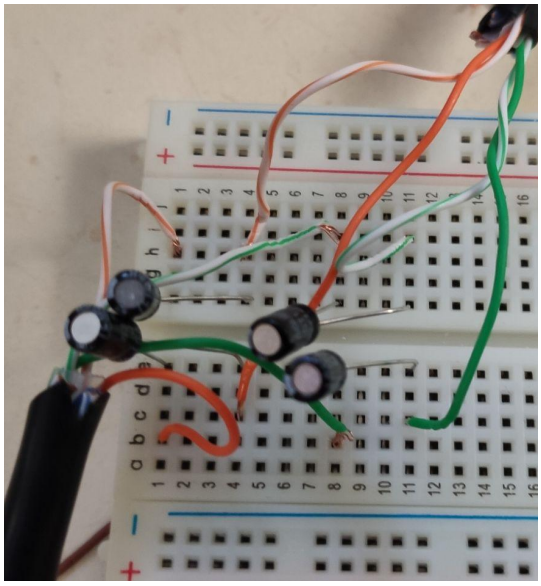


Figure 6 - Basic schematic for experiment

5.1. Cable experiment

First to establish that there was a working signal I just connected the capacitors on a breadboard with the two pairs. Iperf (Below picture to the right) could then report the transfer speed and check packets.



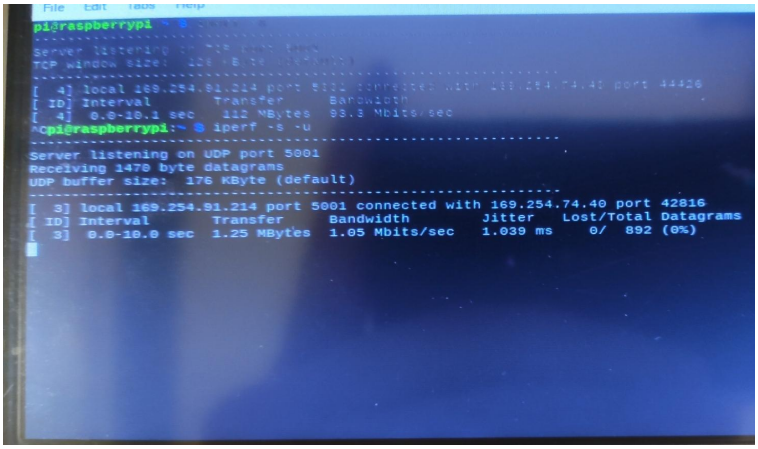


Figure 7 - Explores whether the capacitors have any impact on the connection.

The signal was not noticeably affected and both the TCP and UDP protocol worked. I then proceeded to connect everything and tried again. This time with two sets of capacitors on each ends.

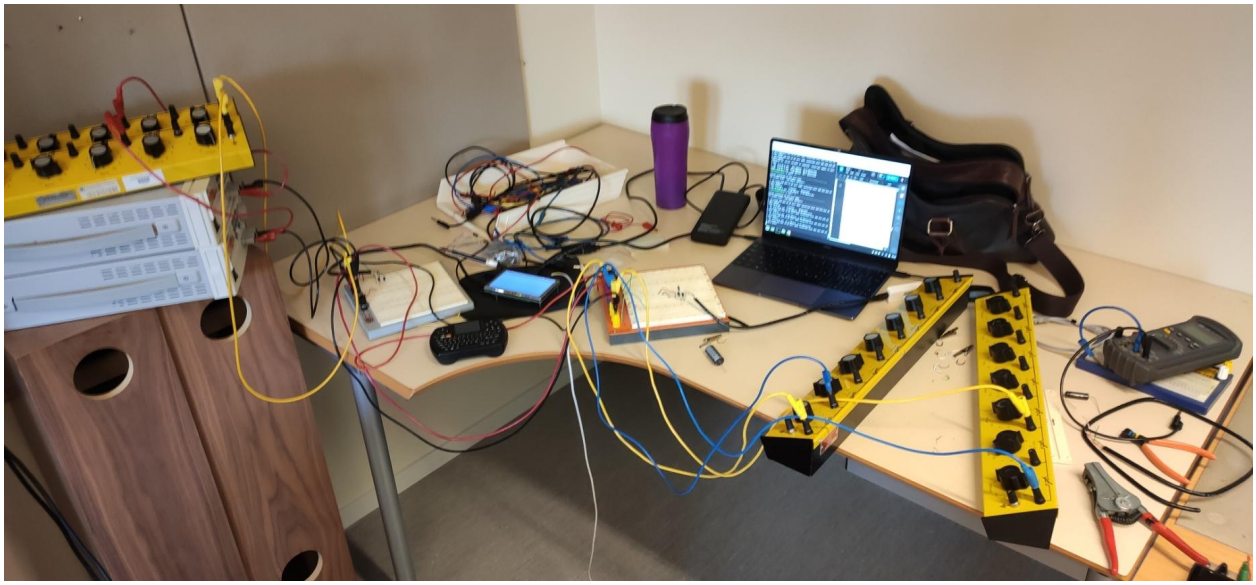


Figure 8 - The full cable testing setup

The load in this case is two variable resistors and I also used two Agilent U8002A power supplies. However the signal did not carry when connected according to the planned schematic. The power supplies even when idle interrupted the signal, adding resistance in series fixed the problem.

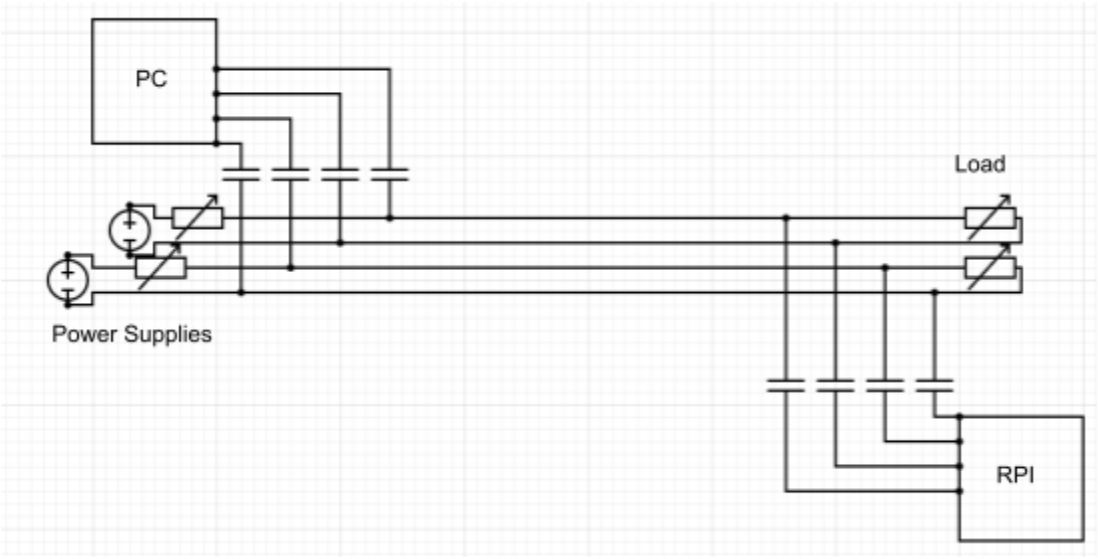


Figure 9 - The final Schematic of the experiment

The addition of the resistors severely limited the amount of power that could safely be conducted in the test as they were not built to withstand a lot of current.

6. Experiment results

The ethernet signal was found to be quite stable, especially when the power supplies was disconnected. In such a case the load could be as small as 1 ohm and there would still be no loss of speed or packets. With the power supplies connected the resistance was essential for stability.

Due to needing to use a resistor the circuit could not be run at very high wattage. Running the circuit at 2W had no impact on any measurement in this experiment.

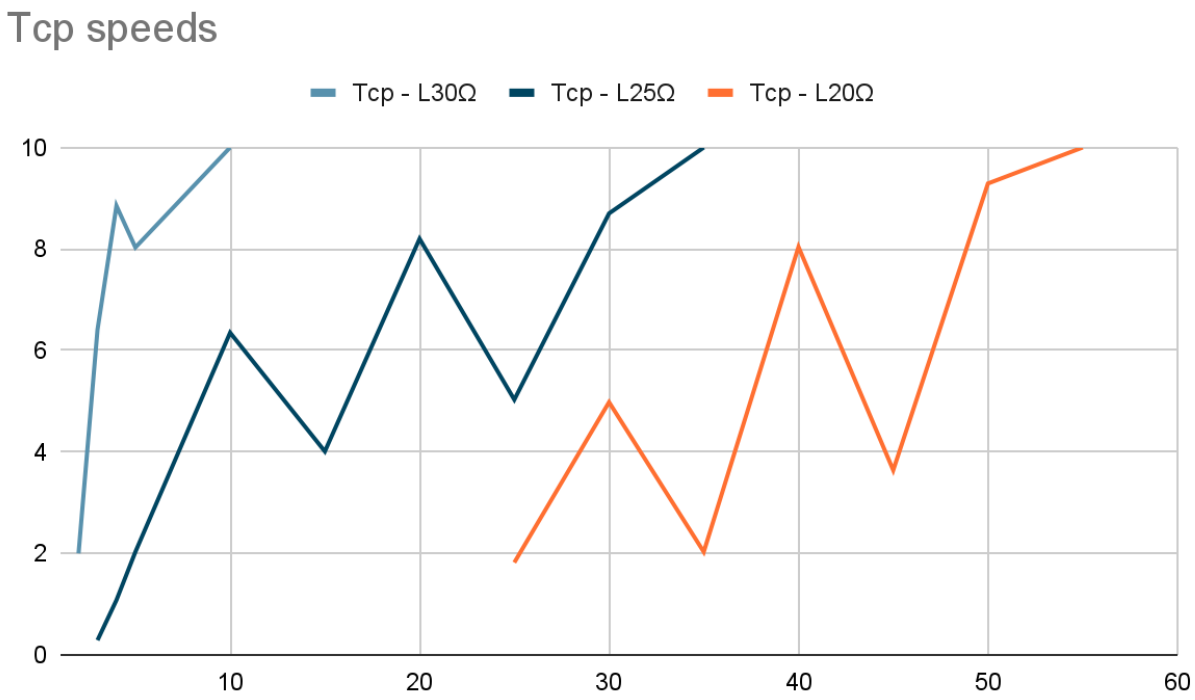


Figure 10 - Graph showing Tcp speeds with changing resistance

The Tcp results were more informative as you got closer to losing the connection. While the Udp could run at even lower resistance with more stability. However at those points the connection could be lost and the ip address could not be resolved. This could potentially be circumvented by not using windows or linux as there is quite a lot of overhead. Two SPE capable microcontrollers might not face that problem.

6.1. Discussion

The experiment was done to explore how the power impacted the signal but it would be more accurate that the supplies themselves impacted the system. Returning to the experiment with an oscilloscope to see what happens would be interesting. Then the problem could be found, whether it was some interference from the supply or maybe the signal was led to ground. The power itself did not interfere in this experiment but running more wattage could have an impact. In addition the load together with the resistor in series with the supply had a big impact. As to why, maybe an oscilloscope could tell how the signal is interfered with.

Using SPE for the experiment should have simplified the circuit as it would have halved the required amount of hardware. In addition the protocol is not dependent on another pair and should have higher stability.

A higher powered experiment could also show the importance of the low pass filter as the signal would be insignificant and would not impact the load. Especially since a high capacity inductor tends to be large and expensive.

The addition of an inductor between the pairs in this experiment would also have been instructive, as it could somewhat simulate what is required to separate the signals between devices.

6.2. Prototype: RaspiCan

6.2.1. Summary

The device is composed of a Raspberry pi 4 and CAN bus bridge and compatible PC software. The codebase developed for these should fit any Raspberry pi. The CAN bus bridge schematic I based mine on the one at Hackster.io. The Raspberry pi is running a script that will transmit data received on the CAN bus to a connected PC. It can also receive messages from the PC and send along those to the CAN bus.

6.2.2. CAN Bus Bridge

As the Raspberry Pi lacks the capability to connect to the CAN Bus, a dedicated circuit is necessary. The MCP2515 is an CAN Controller that can handle the extended frames used by Ocean Robotics systems. In addition it is necessary to use a CAN Transceiver to protect the Controller. The CAN Bus messages are then sent to the Raspberry Pi over SPI.

6.2.3.1. CAN Bus Schematic

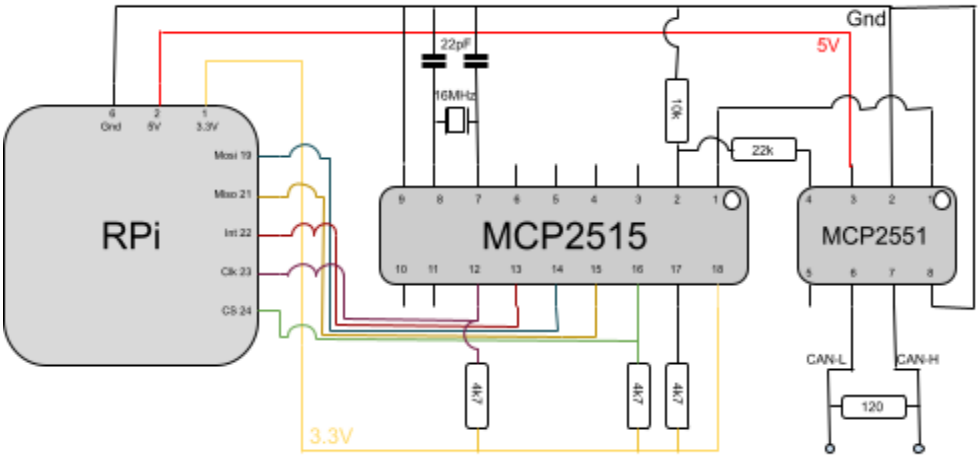


Figure 11 - CAN Bridge circuit

This simple bridge can be used to read and write to the CAN bus, the reference at Hackster.io had a few non-fatal mistakes that have been corrected here.

6.2.3. Raspberry Pi

6.2.3.1. Development environment

This prototype used a Raspberry pi 4 Model B 4GB however since the GPIO pins are the same on all editions (that have them) it should be compatible. Performance wise it also should work

fine with either the earlier models or a Raspberry pi Zero. The prototype also used full Raspberry pi OS but using a headless environment would improve performance.

6.2.3.2. Can-utils package

Developed by Kleine-Budde and Harkopp this package is essential to the script and provides all the tools to communicate between the Raspberry pi and the CAN bus. Can-Utils (Linux-Can) makes it possible to read the messages over SPI provided there is a CAN bridge such as the one provided in this paper or possibly something similar. The messages has the format Id#Data in hexadecimal. Id can either have a short or long description (3 or 7 bits). In addition there is a timestamp with the message provided by the tools which is saved locally. This stamp will be used in case another message arrives with the same Id, in that case it will be compared to the previous one to establish the frequency of the message.

6.2.3.3. Python Script

The Raspberry pi uses a python script which can run on startup and should run indefinitely. It also should run on any network without any additional tweaking provided you can find the ip address given by the network. As soon as a connection is established it will start reading the CAN bus and sending that data as a stream to be sorted by the accompanying pc software.

As soon as a client is connected the script runs a console in the background and makes use of the information received by the Can-Utils tools. The information is then processed locally so that we can calculate the frequency of a message by comparing it with the last one. Since the timestamps are made locally there is no communication delay over the network. This timestamp can also be used to calculate the response time of a signal if requested by the client.

There is also an additional thread that checks if the client is sending any messages. If any are sent it will first decode whether the client specifies how to send it to the CAN bus. The client can send specific messages with a set number of repetitions along with a desired frequency, to the users specifications. There is also an additional function to time the expected response from the bus at a specific Id. If it successfully received a message with that id, it will send it along with the time it took since the client sent the message.

6.2.4. PC Software

6.2.4.1. Receiving Messages

The program was built in visual studio and communicates with the Raspberry Pi using TCP/IP communication. It requires an ip address of the Raspberry pi to make a connection. By default it uses port 10000, to change that the code needs to be edited in both the Python script and the pc software. Once a connection is established the messages will immediately be sent if there are any. In case of no messages but a successful connection the Data field will show a message.

The message is then displayed on the right sides along with the frequency of the message and the total number of received messages. The messages are sorted by the time of arrival and repeating messages will only show the most recent data content.

6.2.4.2. Sending Messages

To send a message over the CAN bus you need to create a message and direct the Raspberry pi on how to deliver it. You can manually add the ID whether it is 3 or 7 bits long and add the Data. Can-utils does not support sending odd messages such as 123 or ABCDE as contents need to be paired. In addition you can specify how many times a message should be repeated and at what frequency. If unfilled it will be sent once. There is also a feature to test the response time of a message. Supply the expected ID the response message should have and in addition to receiving the message along with the rest there is now also Data of the response with the time it took.

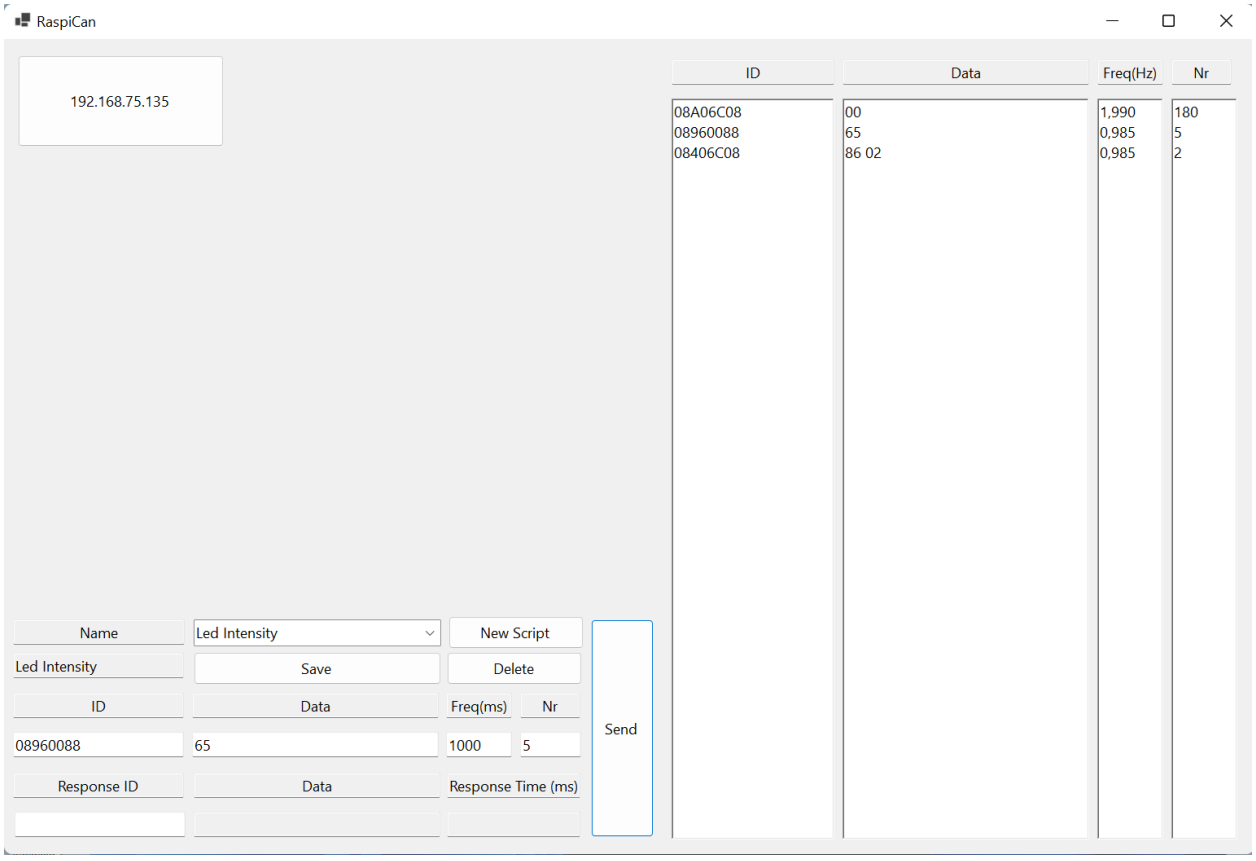


Figure 12 - PC software screenshot

7. Discussions

This prototype has no direct relation to the SPE system integration however unlike the SPE it is available now and works with the current system. There are commercially available products that do work in a similar fashion, in fact you can buy a CAN bridge hat for the raspberry pi and use the code, it should be compatible. However this prototype is easy and cheap to modify for any purpose that Ocean Robotics needs. Such as developing hardware to control the vehicles. It could also be assembled fairly simply by a client with locally sourced parts to allow control of the system over the internet if necessary. There is no security whatsoever in the software.

With some modification the Raspberry Pi could also be used to perform specific tasks or be a middle hand between systems. In fact one variation has already been used in order to allow communication between a network based device and the overlay system. That product modified the script to make use of information found on the network and send it to the CAN bus.

8. Conclusions

While the commercial standard for SPE is not going to cover anything over 50W, theoretically such a system can work. Especially when considering the cable length is limited. This would require designing a dedicated switch and thoroughly testing. The short and simple low powered test that was performed used no shielding and therefore is not accurate to any real scenario. It only confirmed that low power had no obvious impact on the signal for software.

It is likely that the cable will also be important for the stability of the signal, even when using short cables. It should also be noted that the test did not use a single pair connection but two paired normal ethernet. It also did not test multiple devices on the same supply, which could have worked as a bus if no proper low pass filter was used.

When it comes to the hardware developed as part of this project it is only tangentially related as it lets you see traffic on a CAN bus over ethernet. A Raspberry pi is locally connected over a CAN bridge, in addition a program was developed in Visual Studio for PC. The bridge can be built according to the schematic or it has been tested to work with a raspberry pi can bridge hat. The raspberry pi can be used directly for simple commands such as listen and sending messages with the linux tools. It otherwise boots with a script to work with the accompanying PC software to make use of the device. The software will then display messages and offer the ability to send messages and listen to potential responses and the time it takes. The time is counted locally on the Raspberry pi, as such this setup could be used over the internet and still be accurate. There is however no security at all and the information is sent as text with no encryption.

9. References

1. IEEE. *IEEE Standard for Ethernet Amendment 8: Physical Layer and Management Parameters for Power over Data Lines (PoDL) of Single Balanced Twisted-Pair Ethernet*. 2017. *IEEE*, https://ieeexplore-ieee-org.e.bibl.liu.se/stamp/stamp.jsp?tp=&arnumber=7851124. Accessed 29 october 2021.
2. Jerez, Felipe. *Inductors for modern Power over Signal applications in automotive*. 2019, p. 4. *IEEE*, IEEE, https://ieeexplore-ieee-org.e.bibl.liu.se/stamp/stamp.jsp?tp=&arnumber=8998711&tag=1.
3. Jenny List. *Ethernet, Over DC Power. 2019* https://hackaday.com/2019/11/14/ethernet-over-dc-power/ Accessed 5 september 2022.
4. Kleine-Budde, Marc, and Oliver Harkopp. *Linux-can*. https://github.com/linux-can. Accessed 29 october 2021.
5. “Single Pair Ethernet SPE – The Infrastructure for IIoT | HARTING.” *Harting*, https://www.harting.com/DE/en-gb/single-pair-ethernet-spe. Accessed 25 November 2021.
6. “SPE Industrial Partner Network.” *Single Pair Ethernet*, https://www.single-pair-ethernet.com/en. Accessed 25 November 2021.
7. Younes. “How to Connect Raspberry Pi to CAN Bus.” *www.hackster.io*, 24 December 2018, https://www.hackster.io/youness/how-to-connect-raspberry-pi-to-can-bus-b60235. Accessed 29 October 2021.
8. Ethernet Over twisted pair(2022,August 28) in Wikipedia. https://en.wikipedia.org/wiki/Ethernet_over_twisted_pair#Single-pair
9. CAN Bus (2022, August 28) in Wikipedia. https://en.wikipedia.org/wiki/CAN_bus

10. Tatsuki Matsushita, Daisuke Umehara and Koichiro *Wakasugi Power over data lines for CAN using AMI code.* 2016 https://ieeexplore-ieee-org.e.bibl.liu.se/document/7835948 Accessed 5 september 2022.

10. Raspberry pi Script

```
import socket

import sys

import subprocess

import datetime

import time

import threading

import queue


class Error(Exception):


    pass

#This task runs on a thread continuously and checks for new messages
on the network connection.

#In Addition it converts the message into something we can use

#Then it sends the data along the CAN Bus

def cansend_task(connection, rqueue):

    while True:

        try:

            data = connection.recv(64)

            if data:

                print("Recieved " + data.decode())

                id, candata, sleeptime, nr, targetid =
data.decode().split("#")


                if sleeptime == "" or nr == "" or sleeptime == "0" or
nr == "0":

                    sleeptime = "0"
```

```
                    nr = "1"


                rqueue.put(targetid)
                if id != "":
                    for i in range(0,int(nr)):
                        c = subprocess.run("cansend can0
"+id+"#"+candata, shell=True)
                        st = (int(sleeptime)/1000)
                        time.sleep(st)
        except:
            print("failed to send")
            break
#Setup Network
def get_ip_address():
    ip_address = '';
    s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    s.connect(("8.8.8.8",80))
    ip_address = s.getsockname()[0]
    s.close()
    return ip_address
#CAN Bus Settings
p = subprocess.run("sudo ip link set can0 up type can bitrate
125000", shell=True)
#Start a Console that can be read from
p = subprocess.Popen("candump any -t A", shell=True, stdout
=subprocess.PIPE)
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
Server_address = (get_ip_address(),10000)
```

```
sock.bind(Server_address)
print('starting up on {} port {}'.format(*Server_address))
sock.listen(1)
#Define variables
newline=""
oldline=""
idlist = []
idlist.append("free")
frlist = []
frlist.append(datetime.datetime.now())
match = 0
responsequeue = queue.Queue()
compareid = ""
responseid = ""
responsetime = ""
#Wait for Connection
while True:
    print("Waiting for Connection")
    connection, client_address = sock.accept()
#Once connection is established parse the CAN Bus
    try:
        CanSend = threading.Thread(target=cansend_task,
args=(connection, responsequeue), daemon=True)
        CanSend.start()
        print('connection from', client_address)
        while True:
            newline = p.stdout.readline()
```

```
            match = 0


#If there is a new message decode it
            if oldline != newline:
                id, hexdata = newline.decode().split("]")
                date, t, can0, id , nrbyte = id.split()
                t = datetime.datetime.strptime(date.replace("(","") +
" " + t.replace(")",""), '%Y-%m-%d %H:%M:%S.%f')


                if not responsequeue.empty():
                    responseid = responsequeue.get()


                if responseid != "":
                    compareid = responseid
                    comparetime = t
                    responseid = ""
#If we tested the response time of a message and it's a match
                if compareid == id:
                    responsetime = t - comparetime
                    h,m,responsetime = str(responsetime).split(":")
                    print("Match Found")
                    compareid = ""


                for i, item in enumerate(idlist):
                    if id == idlist[i]:
                        fr = t - frlist[i]
                        h,m,fr = str(fr).split(":")
```

```
                        frlist[i] = t
                        match = 1


                if match == 0:
                    if idlist[0] == "free":
                        idlist[0] = id
                        frlist[0] = t
                        fr = "0"
                    else:
                        idlist.append(id)
                        frlist.append(t)
                        match = 0
                        fr = "0"




                hexdata = hexdata.strip()
                id = id.strip()
#Format the CAN message and then send to client
                can0 = "!" + id + "#" + hexdata + "#" + fr + "#" +
responsetime
                print('sending ' + can0 + ' to the client')
                connection.sendall(can0.encode('ascii'))
                oldline = newline
                responsetime = ""
    except socket.error:
```

```
        print("Connection Failed")
        connection.close()
        CanSend.join()

    finally:
        # Clean up the connection
        connection.close()
```

11. Raspberry pi Setup

This project used a Raspberry pi 4B which had more than enough capacity to make use of the script. It is very probable less powerful versions could handle it too like the Raspberry pi Zero. The code has been tested using the bridge described in this paper but it should be compatible with similar premade bridges.

11.1. Connecting the Hardware

Connect the bridge to the Raspberry pi GPIO pins.

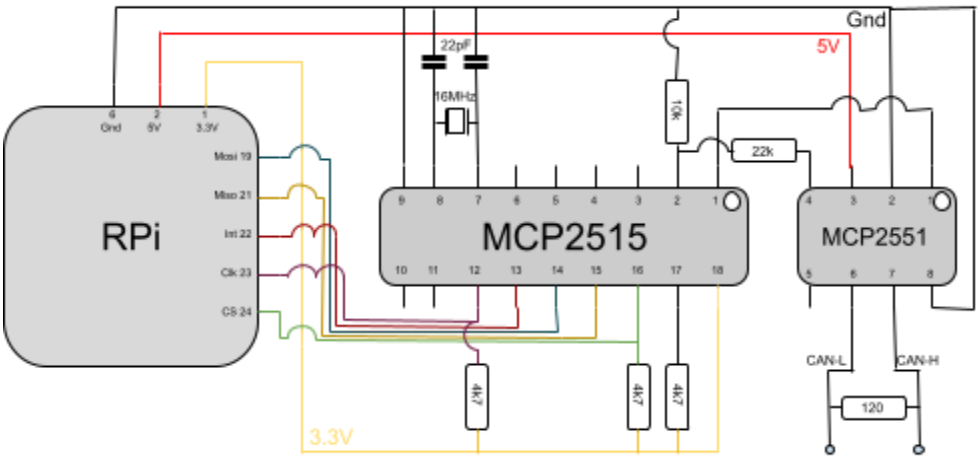


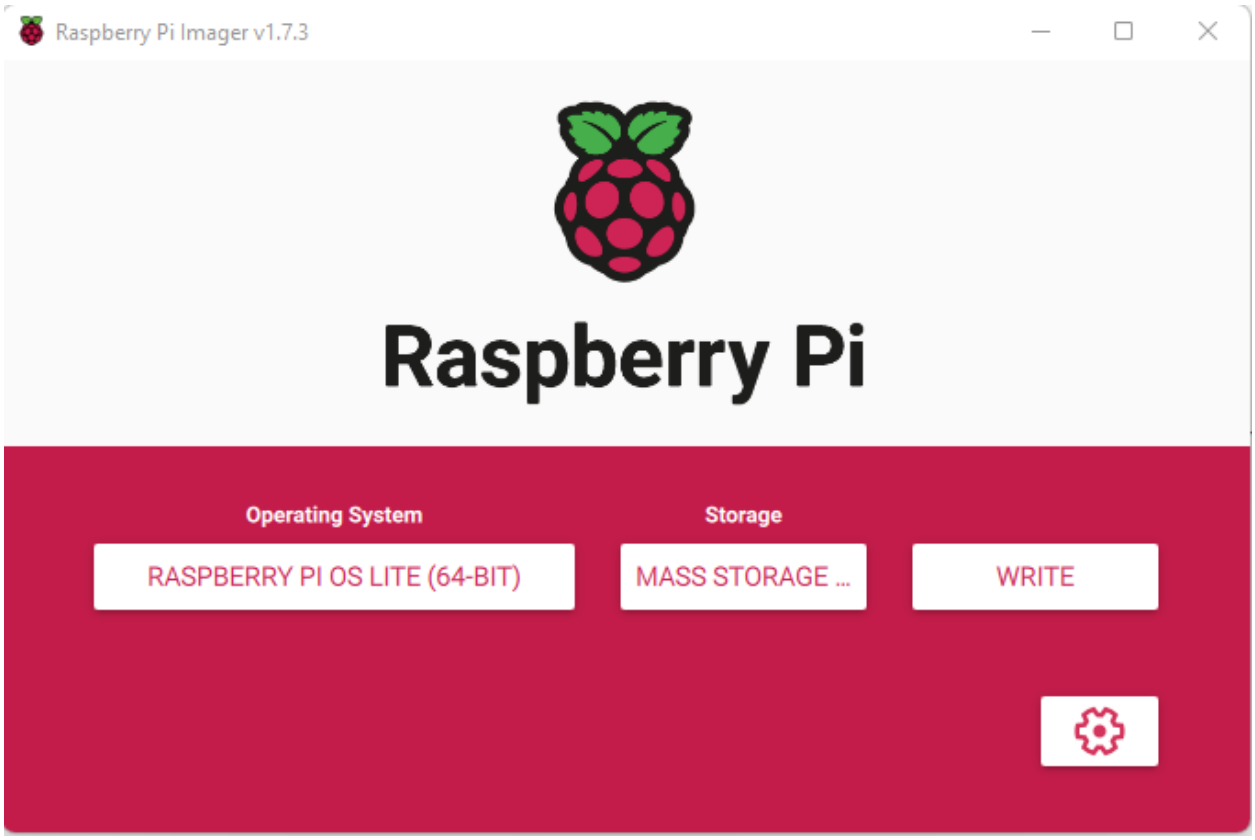
Figure 13 - Rpi GPIO connections

11.2. Setting up the operating system

This prototype was developed on Raspberry pi OS however instructions will be for the Headless Raspberry pi OS to improve performance. This also should allow older raspberry pi boards or less powerful ones like the Zero to run the program.

1. The easiest way to install the OS is to download and use the Raspberry pi Imager. Which is found here: https://www.raspberrypi.com/software/
2. Using the Imager you select an OS that works with your Pi under
   Choose OS -> Raspberry Pi Os (other)
   For Pi 3 or later pick the Raspberry Pi OS Lite (64-bit)
   For older Pi's select Raspberry Pi OS Lite (32-bit)
3. Connect the micro SD card that is going to house the OS and select the correct drive letter. Any SD card that has at least 512 mb should work, make sure to backup anything on the card.

4. On the Cogwheel you can set up your username and password preemptively. But more importantly if you are using a wireless connection you can enter that connection here under Configure wireless LAN. In addition if you do not have a screen connected to the Raspberry pi you can enable SSH for the next step.
5. Click Write and wait until it is finished.



11.3. Raspberry Pi Configuration

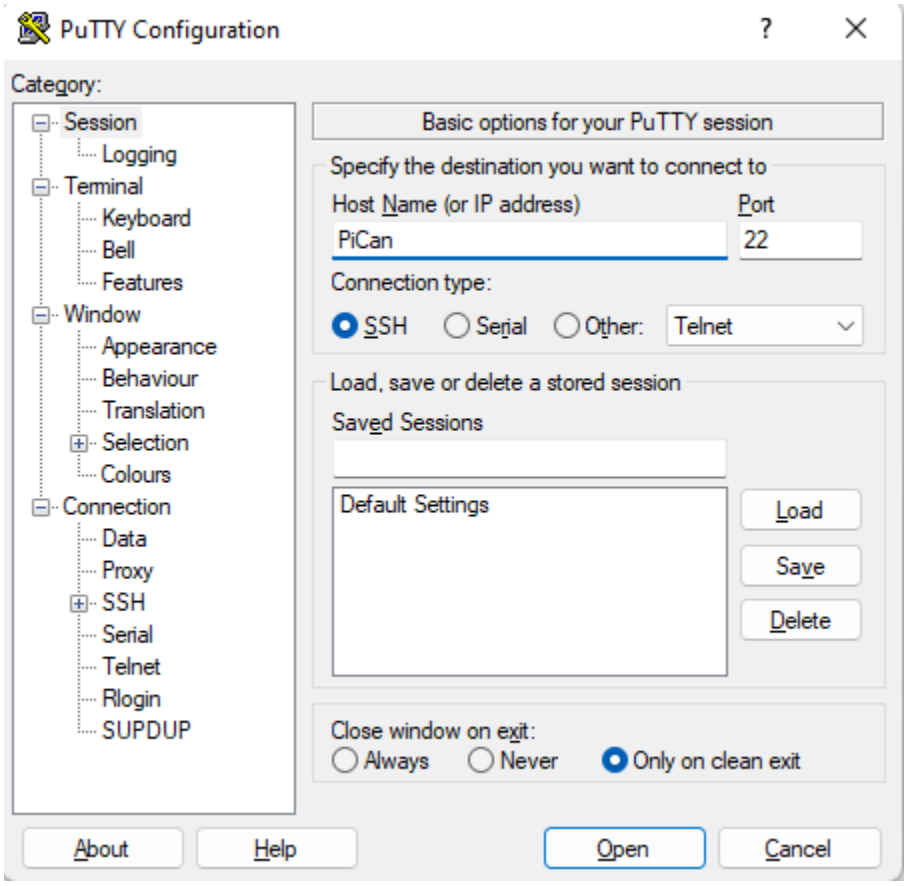
After slotting the SD card in the Raspberry pi, set up your user and password as you wish. You also need to have a network connection and a way to interact with the console on the Pi. This can be done using a connected monitor and a keyboard or using SSH. As the intention is for the Raspberry pi to run autonomously SSH is recommended.

11.4. Headless Setup

You can skip this step if you are not using SSH:

1. Download and install Putty
   https://www.chiark.greenend.org.uk/~sgtatham/putty/latest.html
2. If you set a hostname for your Raspberry Pi you can insert it otherwise you will need the ip address of the Pi. This can either be found by scanning your network for your Pi or looking at the router.

3. Click open and login or create the user credentials.



11.5. Installing prerequisites

The script is written in python and uses the package can-utils to communicate with the CAN Bus. It cannot be run without these packages and while the script does not require an internet connection it is required to install these packages.

1. Write the following commands for python3:
   sudo apt update
   sudo apt install python3
2. Write for can-utils
   sudo apt-get install can-utils

11.6. Transfer the script

There are multiple ways to transfer the script to the pi, this guide will use an usb drive.

1. Copy the script from the previous chapter to a text file.
2. Rename the script file type the .py. Such as: PiCanScript.py
3. Copy it to a usb drive and insert it to the Pi.
4. Write *sudo mkdir -p /mnt/usb1* in the Console.
5. Navigate to it using *cd /mnt/usb1*
6. Copy the script *cp PiCanScript.py /users/user* where user is the name you have set up previously and PiCanScript.py is the name of the file.

At this point the script can be run by using the command: *python3 PoCanScript.py* if everything was set up correctly.

11.7. Start the script after boot

In order to make the Pi work with no direct input we will need to run the script as soon as the Pi has started. This can also be done in many ways but an example follows.

1. Write *sudo nano /etc/rc.local* this will open up a text editor.
2. Add a new line before the *exit 0* line and add *python3 /home/user/PiCanScript.py &* where user is the previously setup user and PiCanScript is the name of the script.
3. Reboot

Hopefully the Raspberry pi will now run the script on boot and you can use the PC software.

11.8. RaspiCan software

After setting up the Raspberry Pi you can now connect the software developed in visual studio. RaspiCan requires that you know the ip address of the Raspberry pi. If you used SSH to connect using a hostname, you can use the Raspberry pi console to write *ipconfig*. This will provide the ip addresses of all the networks it is connected to. Pick the shared network ip address and copy it, in RaspiCan you click Connect and insert the ip address.

If successful you can tell that it is connected if a “connection successful” text shows up in the messages window. Or if the window is filled by CAN messages due to establishing connection.