TESTING

VERSATILE ARDUINO NETWORK (VAN)

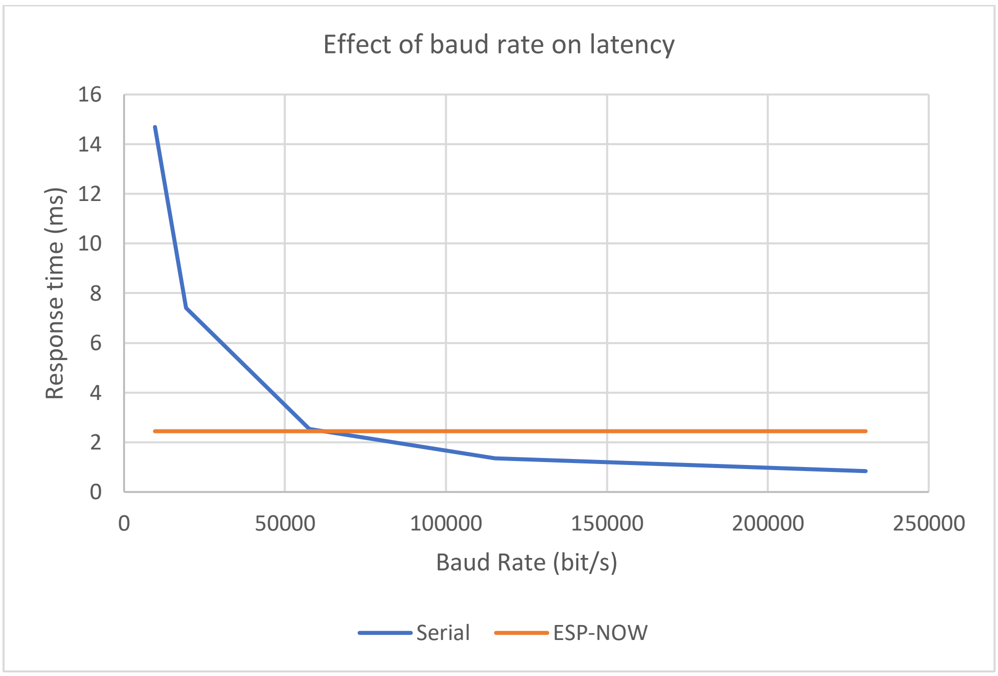
CONTENTS

Note: All tests and results are presented here, but for reference or reproduction of tests the testing code is available on the VAN2 GitHub repository in folder testingNodes.

EXPERIMENTAL TESTING

NETWORK LATENCY

Results were obtained using testingNodes/latencyTest mega.ino, uno.ino. A direct serial link was established between the two, with the mega sending an echo request, and the uno providing a response, using baud rates ranging from 9600bit/s to 230400bit/s. Additionally (using testingNodes/latencyTest espStatic.ino, espBuggy.ino), the response time using the ESP-NOW link was obtained for the same request. For all node's devices were not used, so that they do not interfere with timings. However, normal networking loops remained.



It is evident that slower baud rates can significantly increase network latency. Increasing past the default 115200bit/s provides diminishing returns, where it is likely that the execution time of networking code prevents much further improvement. It is also apparent that the ESP-NOW wireless link provides competitive communication speeds in comparison to the hardware serial links, although it is somewhat slower than the default hardware serial speed.

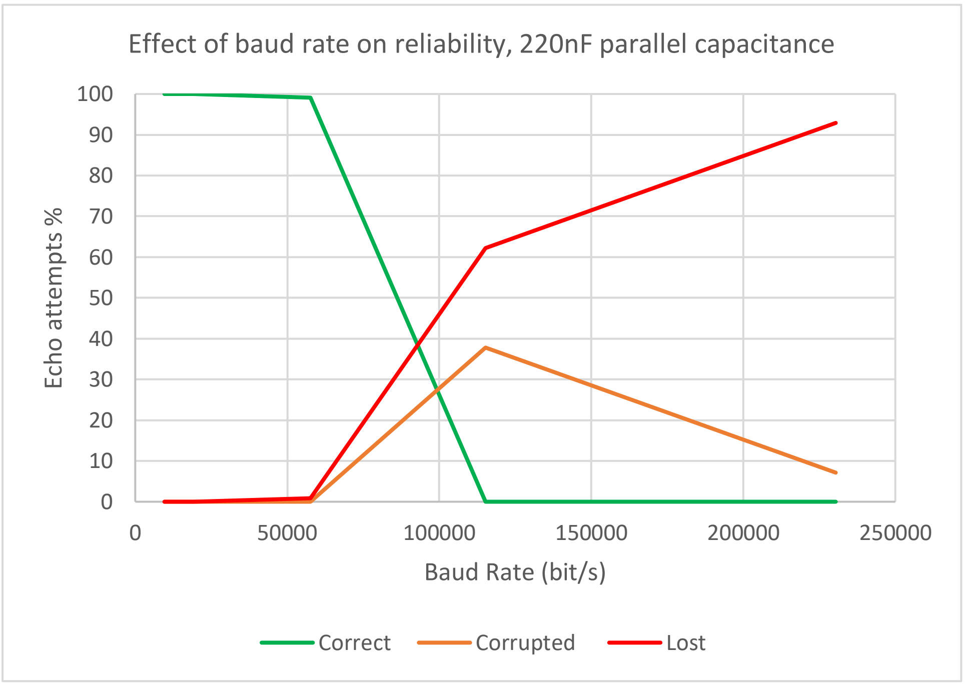
LINE QUALITY

For the purpose of assessing reliability over hardware serial, another echo testing routine was used (located in testingNodes/checksumTest) 1000 echo requests were sent from the mega: correct responses and corrupt responses (detected by the checksum) were tallied. Responses which were incorrect, but not caught by the checksum test were classified as “missed”. If no response was obtained 30ms after sending an echo request (which should be sufficient, even for the lowest baud rate of 9600bit/s) the message was classified as lost.

A direct serial link was tested first (~10cm wire length), and it proved very reliable. With baud rates ranging from 9600bit/s to 230400bit/s all tests returned 100% of responses correctly.

However, it is entirely likely that the communications link may not always be this simple. In an effort to simulate a poorer physical connection, or a much longer line length, a 220nf ceramic capacitor was placed across the TX and RX lines.

The test was run again, and its results are plotted below.



It is evident that lower baud rates can provide a reliable connection over a poor-quality line, where the default 115200bit/s, or higher baud rates fail completely. It is interesting to note that corrupted responses rise and then fall as the baud rate increases. The initial rise is likely due to the corruption occurring on the return transit, from the uno, which itself received a correct echo request. At still higher baud rates, more messages are simply lost because the echo requests are corrupted on the outgoing transit from the mega, and do not even prompt a response from the uno. No messages were missed, i.e. The checksum test caught 100% of corrupted responses. This is encouraging, as although it is not capable of detecting all possible corruptions, in practice it has proved very capable of detecting errors caused by poor line quality.

ESP-NOW RANGE

For the purpose of assessing reliability of the ESP-NOW wireless link, the previous test was also implemented on two ESP32 devices (code located in testingNodes/checksumTest). However, only the ratio of sent requests to correct responses were relevant, as the implementation off error checking via ESP-NOW differs from that of serial communications. Again, 1000 echo requests were sent for each condition.

Initially, the sending device remained inside, whilst the remote device was placed 60m away, outside with no direct line of sight. No messages were correctly received.

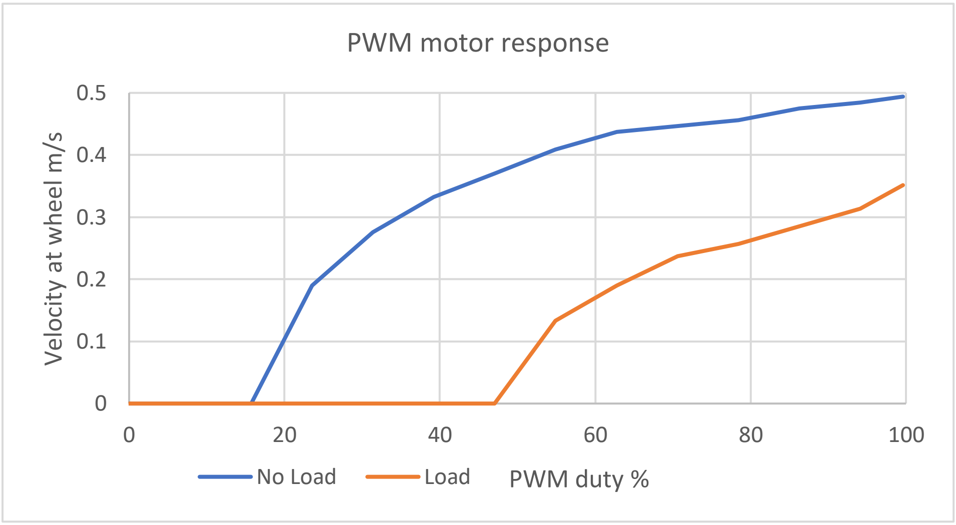
Range was then explored inside a typical residential building, and the number of correct responses were recorded in the table below. With the remote device in the same room (room 0) 100% of responses were received correctly.

Remote room	no closed doors	1 door	2 doors	3 doors	Other factors
Room 1 wifi off	579	1000	995		through fridge
Room 1	835	648	831		through fridge
Room 2	1000	326	715	927	2 walls
Room 3	1000	1000			1 wall
Room 4	1000	1000	1000	1000	2 walls
Room 5	0	0	0		3 walls
Room 6	1000	1000	1000	1000	through ceiling

The only reliable observations that could made is that ESP-NOW may not be capable of providing communication between all locations of a single building, but can perform reliably within a single room, or between two adjacent rooms.

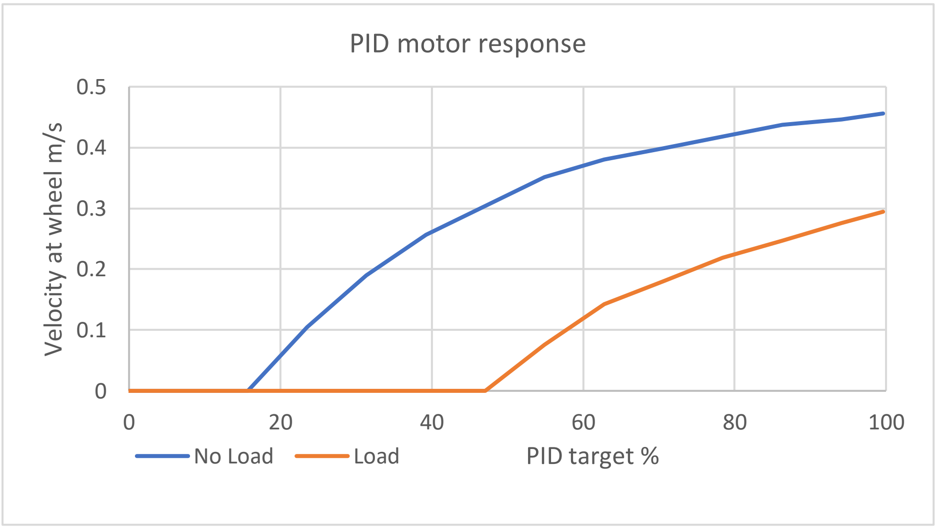
MOTOR RESPONSE

To assess how motor PWM affects the buggies motion, its motor response was recorded with and without load from the encoders. The VAN motors device was used to set duty, ranging from value 128, stop to 255, full speed forwards. Velocity was produced from encoder counts, where the wheel diameter, counts/ revolution, and counts/second were known. The applied load was driving the buggy on a level surface, with the other motor disabled. Results were taken once wheel velocity had stabilized, and are shown below.



It is evident that PWM duties below a threshold needed to overcome static friction result in no motion, and this threshold is significantly increased with a reasonable load. It is also evident that the motor response is nonlinear, and greatly reduced by a reasonable load.

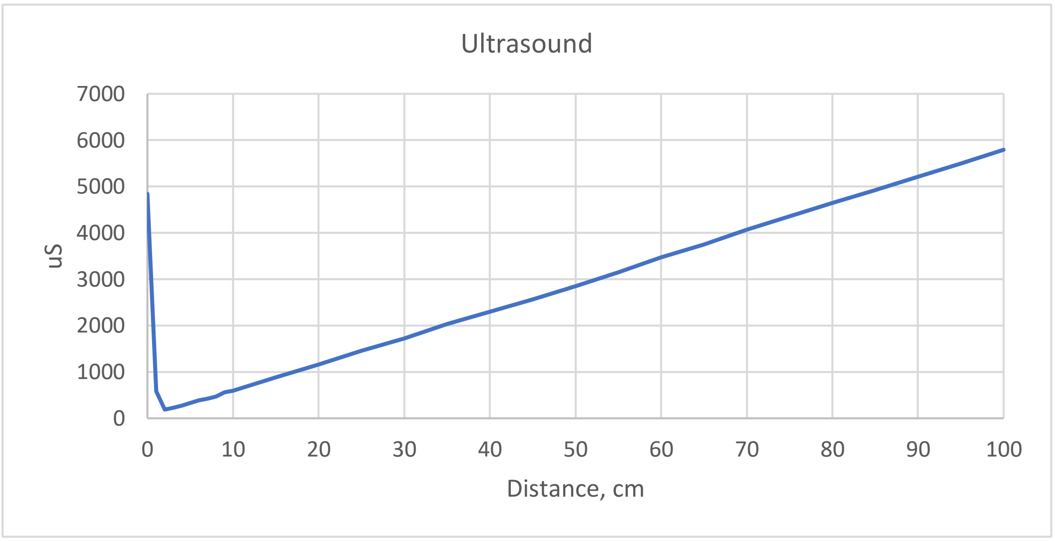
A PID controller was implemented (van_dev_pid), with its gains left at the default (encGain = 10, P = 15, Ig = 0, Dg = 2). Results were recorded across the same range of inputs, shown below.



The response shows marginally improved linearity. It is evident that PID control with these values does not compensate for a load effectively, where the wheel velocity has reached a steady state. The use of an integral component may be beneficial, but can cause other problems such as windup. Increasing both proportional and encoder feedback gain may also reduce the effect of load. It is worth noting that transient response was not recorded, so the derivative component shows no effect.

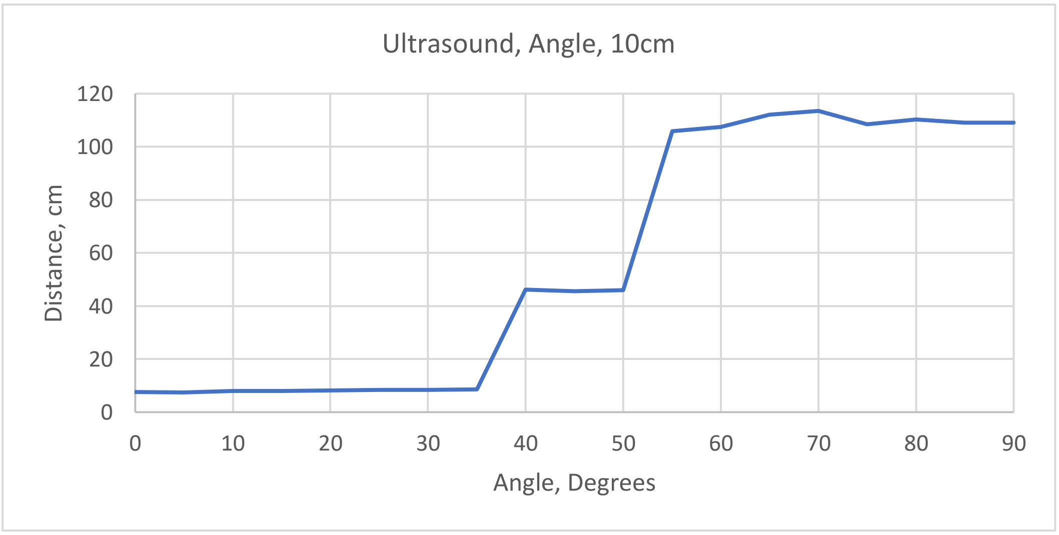
ULTRASOUND CALIBRATION

The HC-SR04 ultrasonic sensor was calibrated using the VAN device van_dev_ultrasound, with file \testingNodes\ultrasoundTest\uno.ino. Aimed directly at a flat face of a solid object, a reliable, linear relationship between measured distance and timed delay is apparent.



However, it is notable that for very close objects (<2cm) the sensor gives thoroughly inaccurate estimates of distance. Ignoring these erroneous readings, a good estimate of distance is given by delay(uS)/57.87.

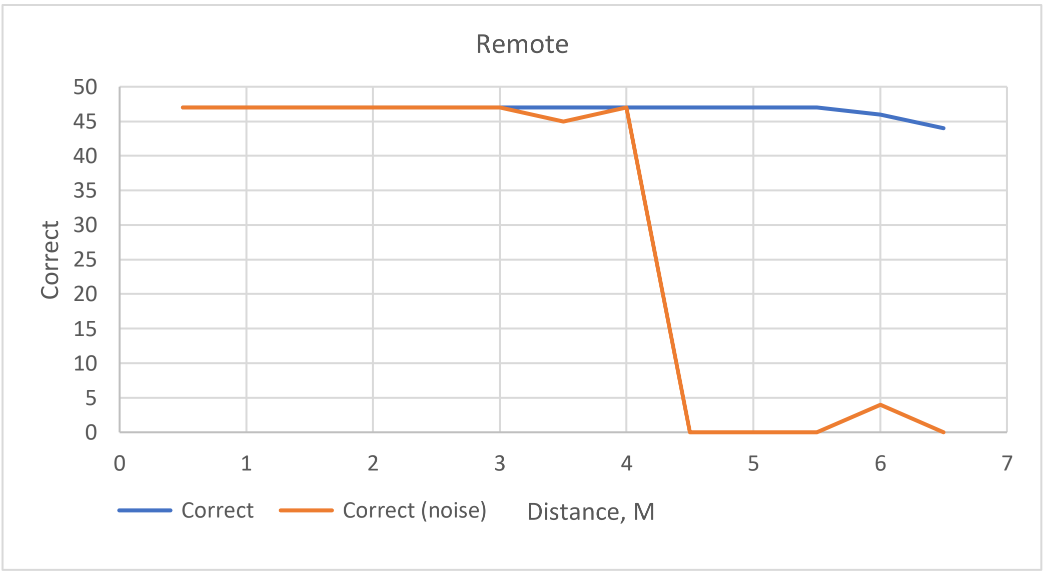
The sensors performance was also envaulted when the object is angled. This was done by rotating the same object around a stationary point 10cm from the ultrasound sensor.



It is evident that surfaces angled beyond 35 degrees away from the sensor can give thoroughly inaccurate estimates of distance.

IR REMOTE RELIABILITY

A measure of correct remote/receiver reliability was obtained over a 5 second period using testingNodes\remoteTest\espBuggy\espBuggy.ino. An individual 5 second trial is triggered by any remote code being received, subsequentially received 'repeat' codes are tallied as correct. A remote button held over a 5 second period in optimal conditions was found to reliably produce a total of 47 repeat codes. Results were obtained over a range of distances, the remote was aimed directly at the sensor, with and without IR noise. IR noise was provided by a 20W incandescent lamp aimed directly at the IR sensor, from a distance of 0.5m.



The sensor appears quite robust to constant IR noise, until a point is reached where the remotes signal becomes too weak by comparison. This does not give a full picture of the sensor's reliability. Although data was not recorded, it was found to be unreliable when the remote was presented at an angle, and sensitive to power supply noise.

CODE TESTING

HANDLEMESSAGE()

The purpose of these tests are to verify the function routes messages correctly, based on their content. Serial print out from Mega (using mega.ino file from testingNodes/ handleMessageTest)- output function sendMessage temporally changed to display action taken over serial0, rather than doing it. Command list changed to display over serial0. Error logging active, logging to this node for display. Comments added in blue

Make message flagged as invalid: dummy.set(STD, MEGA, ESP32BUGGY, REPORT, 12, 34, false);

Test 1, destination: ESP32BUGGY
no action taken, invalid messages ignored

Mark as valid
Test 2, destination: ESP32BUGGY
Message content:
7_1_2_1_12_34
Message checksum:
43
Sent via serial port:
1
esp32buggy node is connected to serial 1

Test 3, destination: JOYSTICK
Message content:
7_1_17_1_12_34
Message checksum:
56
Sent via serial port:
1
Joystick device resides on esp32static, connected to esp32buggy via esp-now
Therefore correct route is still serial1, but is only the first step

Test 4, destination: NANO
Message content:
7_1_4_1_12_34
Message checksum:
45
Sent via serial port:
2
nano node is connected to serial 2

Test 5, destination: NANO_ENC
Message content:
7_1_6_1_12_34
Message checksum:
47
Sent via serial port:
3
nano_enc node is connected to serial 3

Test 6, destination: PID
Message content:
7_1_31_1_12_34
Message checksum:
54
Sent via serial port:
3
PID device resides on nano_enc, so serial 3 is correct 1st step

Destination is now the node itself
Test 7, destination: MEGA
Message passed to local device: this node (MEGA)
7_1_1_1_12_34
Routed to correct function in commandList

Destination is now a device which resides on this node
Test 8, destination: MOTORS
Message passed to local device: motors
7_1_20_1_12_34
Routed to correct device in commandList

Destination is now an erroneous byte, 45, no device or node has been defined with this address
Test 9, destination: error
Message passed to local device: this node (MEGA)
Is still sent to mega node, as this has been set as the eLog destination
7_1_1_14_3_4
However, this is now an error message
src = 1, error triggered from mega
dest =1, error routed to mega
cmd = 14, identifies as error message
dat0 = 3, identifies file which produced error (vanMega)
dat1 = 4, error code from handleMessage function
//ERROR4- Message has destination unknown to this node

GETMESSAGE()

The purpose of these tests are to verify the function returns messages correctly, based on bytes in serial buffer. Serial print out from Mega (using mega.ino file from testingNodes/ getMessageTest)- Serial1, serial2 were directly connected with jumper wires, allowing bytes to be sent on serial1, and then received via getMessage on serial2. Comments added in blue

Test 1, bytes sent:7,2,3,4,5,6,1

Bytes compose message of correct length, with a correct start and checksum

Received message:

7_2_3_4_5_6

Received as valid message (checksum not shown)

Test 2, bytes sent:7,3,3,4,5,6,1

2nd byte has been corrupted, checksum not changed

Received message:

No valid message received; checksum has detected error

Test 3, bytes sent:7,3,3,5,5,6,1

4nd byte has been corrupted, checksum not changed

Received message:

7_3_3_5_5_6

checksum has failed to detect the error, since two corrupted bits in same byte position produce the same XOR, the checksum is still valid for this corrupted message

Test 4, bytes sent:7,3,3,5,6,5,1

last 2 bytes swapped, checksum not changed

Received message:

7_3_3_5_6_5

again, error not detected. Swapped bytes produce the same XOR

Test 5, bytes sent:8,3,3,5,6,5,1

Corrupt start byte to be invalid

Received message:

no message received, without start byte getMessage cannot know which byte corresponds to which message variable, so returns no valid message

Test 6, bytes sent:8,7,3,3,5,6,5,1

Add a junk byte before message

Received message:

7_3_3_5_6_5

Correct message extracted, junk byte ignored

Test 7, bytes sent:7,7,3,3,5,6,5,1
Add a junk byte which is a valid start (false start)
Received message:
false start has erroneously advanced all bytes with respect to the message variables, causing checksum to detect the error. No valid message returned.

Test 8, bytes sent:7,3,3,5,6,5,7,1
Add a junk byte just before checksum (within message)
Received message:
No valid message returned, as the error has been detected, but the intended message cannot be extracted

ACKNOWLEDGEMENT

1st test:

The purpose of this test is to verify that the ack protocol works between directly connected nodes

Ultrasound connected to Arduino Nano sending forward motion enable command to Mega with motors connected. Serial print out from Mega- comments added in blue

```
17:52:20.861 -> Message received
17:52:20.861 -> 7_16_20_8_255_255
        7 bytes sent every 100ms when U/S is safe distance from object:
        7_16_20_8_255_255 (+checksum/not shown)
        7: std message start byte (does not require acknowledgement)
        16: message source is ultrasound
        20: message destination is motors
        8: command is change parameter 0 (forwards motion enable/disable)
        255_225: Boolean true
17:52:20.964 -> Message received
17:52:20.964 -> 7_16_20_8_255_255
17:52:21.067 -> Message received
17:52:21.067 -> 7_16_20_8_255_255
17:52:21.171 -> Message received
17:52:21.171 -> 7_16_20_8_255_255
17:52:21.309 -> Message received
17:52:21.309 -> 90_16_20_8_0_0
        When U/S has detected obstacle, message is added once to “waiting for acknowledgement buffer”,
        which sends the following every 100ms, until cleared from buffer:
        90_16_20_8_0_0
        90: ACK start byte now indicates a response is required. This is changed by the buffer, not US
        0_0: Boolean false
17:52:21.309 -> This message requires acknowledgement <- Mega has detected ACK start
17:52:21.309 -> Response returned to sender:
17:52:21.309 -> 110_20_16_8_0_0
        110_20_16_8_0_0
        110: RESP start byte now indicates that this message is a response
        20: message source is motors
        16: message destination is ultrasound
17:52:21.309 -> And normal message to command destination device:
17:52:21.309 -> 7_16_20_8_0_0
        7_16_20_8_0_0
        A standard message disabling forwards motion has been “split off”, and sent to the motors device.
        Obstacle was left in front of ultrasound sensor for a while.
        Messages from US stop, transaction complete (motor disable message has been cleared from ack
        waiting send buffer by response)
        After removal of object, motor enable messages resume:
17:52:34.946 -> Message received
17:52:34.946 -> 7_16_20_8_255_255
17:52:35.050 -> Message received
17:52:35.050 -> 7_16_20_8_255_255
```

2nd test

The purpose of this test is to verify that the ack protocol chains over intermediary nodes
Ultrasound now addressed to device IMU, located on ESP32 buggy. This requires it to route messages through the Mega: **NANO(ultrasound)->MEGA->ESP32(imu)**

Serial print out from Mega:

```
23:40:03.800 -> PORT:  2 <- message received on serial port 2 (attached to Nano with U/S)
23:40:03.800 -> 7_16_18_8_255_255
23:40:03.903 -> PORT:  2
23:40:03.903 -> 7_16_18_8_255_255
23:40:04.042 -> PORT:  2
23:40:04.042 -> 7_16_18_8_255_255
23:40:04.145 -> PORT:  2
23:40:04.145 -> 7_16_18_8_255_255
23:40:04.249 -> PORT:  2
23:40:04.249 -> 7_16_18_8_255_255
            U/S has detected obstacle
23:40:04.387 -> PORT:  2
23:40:04.387 -> 90_16_18_8_0_0
            90_16_18_8_0_0
            18: Destination is IMU
            The IMU is not local to the Mega, so no response is generated and message is passed on unchanged
23:40:04.387 -> PORT:  1 <- message received on serial port 1 (attached to ESP32 with IMU)
23:40:04.387 -> 110_18_16_8_0_0
            110_18_16_8_0_0
            110: message is a response
            18: message source is IMU
            16: message destination is ultrasound
            Messages cease
```

Simultaneous serial printout from ESP32:

```
23:40:03.800 -> NORM2 <- message received on serial port 2 (attached to Mega)
23:40:03.800 -> 7_16_18_8_255_255
23:40:03.905 -> NORM2
23:40:03.939 -> 7_16_18_8_255_255
23:40:04.043 -> NORM2
23:40:04.043 -> 7_16_18_8_255_255
23:40:04.147 -> NORM2
23:40:04.147 -> 7_16_18_8_255_255
23:40:04.251 -> NORM2
23:40:04.251 -> 7_16_18_8_255_255
            U/S has detected obstacle
23:40:04.388 -> NORM2    <- ESP has not yet checked if message requires response
23:40:04.388 -> 90_16_18_8_0_0
23:40:04.388 -> RESPONSE2<- ESP has detected need for a response
23:40:04.388 -> 110_18_16_8_0_0 <- generates response (RESP, src&dest swapped)
            Messages cease, transaction complete
```

3rd test

The purpose of this test is to verify that the ack protocol chains over intermediary nodes, with ESP-NOW as a wireless link in the chain.
Ultrasound is now addressed to device SKIDSTEER, located on ESP32 static. This requires it to route messages through the Mega, ESP32 buggy, and ESP-NOW:
NANO(ultrasound)->MEGA->ESP32buggy->(esp-now bridge)->ESP32static(skidsteer)

Serial print out from ESP32static:

```
10:48:11.543 -> NORM3<- message received on 'serial port 3'
            'serial port 3' is ESP-NOW
10:48:11.543 -> 7_16_30_8_255_255
10:48:11.646 -> NORM3
10:48:11.646 -> 7_16_30_8_255_255
10:48:11.646 -> NORM3
10:48:11.646 -> 7_16_30_8_255_255
10:48:11.751 -> NORM3
10:48:11.751 -> 7_16_30_8_255_255
10:48:11.855 -> NORM3
10:48:11.855 -> 7_16_30_8_255_255
            U/S has detected obstacle
10:48:12.027 -> NORM3    <- ESPstatic has not yet checked if message requires response
10:48:12.027 -> 90_16_30_8_0_0
10:48:12.027 -> RESPONSE3
10:48:12.027 -> 110_30_16_8_0_0 <- generates response (RESP, src&dest swapped)
            Messages cease, transaction complete
```

RINGBUFFER

The purpose of this test is to verify that the FIFO ringBuffer object correctly stores and retrieves messages. To do this sequential dummy messages were generated, incrementing all variables other than .valid by 1 from one message to the next. This is so message order can be observed by looking at the first variable (ringBuffer does not act on message content other than bool .valid)

The code used to implement this test can be found in: testingNodes\ringbuff_U1

To test the buffer fully, multiple conditions were used:

1. Sequential stores, without reading, overflowing buffer
2. Sequential stores and sequential reads, keeping buffer at capacity
3. Sequential reads, draining buffer until empty

Serial printout, comments in blue:

START STORING

```
Another message stored <-1st message stored (values of 2 for all variables)
Status before reading:<-status of all 10 ringbuffer slots without reading any out (not destructive)
1:2_0:0_0:0_0:0_0:0_0:0_0:0_0:0_0:0_0:0_
        1:2 Only one message present-
        1 Bool, indicates valid message in slot
        2 Value of first variable of message
Another message stored
Status before reading:
1:2_1:3_0:0_0:0_0:0_0:0_0:0_0:0_0:0_0:0_
        1:3 Subsequent messages stored in next available slot
        This continues…
Another message stored
Status before reading:
1:2_1:3_1:4_0:0_0:0_0:0_0:0_0:0_0:0_0:0_
Another message stored
Status before reading:
1:2_1:3_1:4_1:5_0:0_0:0_0:0_0:0_0:0_0:0_
Another message stored
Status before reading:
1:2_1:3_1:4_1:5_1:6_0:0_0:0_0:0_0:0_0:0_
Another message stored
Status before reading:
1:2_1:3_1:4_1:5_1:6_1:7_0:0_0:0_0:0_0:0_
Another message stored
Status before reading:
1:2_1:3_1:4_1:5_1:6_1:7_1:8_0:0_0:0_0:0_
Another message stored
Status before reading:
1:2_1:3_1:4_1:5_1:6_1:7_1:8_1:9_0:0_0:0_
Another message stored
Status before reading:
1:2_1:3_1:4_1:5_1:6_1:7_1:8_1:9_1:10_0:0_
Another message stored
Status before reading:
1:2_1:3_1:4_1:5_1:6_1:7_1:8_1:9_1:10_1:11_
```

```
          All slots hove now been filled with messages
ESP-NOW ring buffer over capacity<- Next message has overloaded buffer
Another message stored <- has not- prints when storage attempt is made
Status before reading:
1:2_1:3_1:4_1:5_1:6_1:7_1:8_1:9_1:10_1:11_
          No change to buffer content, it cannot accept new messages
          This continues…
ESP-NOW ring buffer over capacity
Another message stored
Status before reading:
1:2_1:3_1:4_1:5_1:6_1:7_1:8_1:9_1:10_1:11_
ESP-NOW ring buffer over capacity
Another message stored
Status before reading:
1:2_1:3_1:4_1:5_1:6_1:7_1:8_1:9_1:10_1:11_
ESP-NOW ring buffer over capacity
Another message stored
Status before reading:
1:2_1:3_1:4_1:5_1:6_1:7_1:8_1:9_1:10_1:11_
ESP-NOW ring buffer over capacity
Another message stored
Status before reading:
1:2_1:3_1:4_1:5_1:6_1:7_1:8_1:9_1:10_1:11_
ESP-NOW ring buffer over capacity
Another message stored
Status before reading:
1:2_1:3_1:4_1:5_1:6_1:7_1:8_1:9_1:10_1:11_
ESP-NOW ring buffer over capacity
Another message stored
Status before reading:
1:2_1:3_1:4_1:5_1:6_1:7_1:8_1:9_1:10_1:11_
ESP-NOW ring buffer over capacity
Another message stored
Status before reading:
1:2_1:3_1:4_1:5_1:6_1:7_1:8_1:9_1:10_1:11_
ESP-NOW ring buffer over capacity
Another message stored
Status before reading:
1:2_1:3_1:4_1:5_1:6_1:7_1:8_1:9_1:10_1:11_
ESP-NOW ring buffer over capacity
Another message stored
Status before reading:
1:2_1:3_1:4_1:5_1:6_1:7_1:8_1:9_1:10_1:11_
ESP-NOW ring buffer over capacity
Another message stored
Status before reading:
1:2_1:3_1:4_1:5_1:6_1:7_1:8_1:9_1:10_1:11_
ESP-NOW ring buffer over capacity
Another message stored
Status before reading:
1:2_1:3_1:4_1:5_1:6_1:7_1:8_1:9_1:10_1:11_
          START READING
Read message: <-A message has been read out from the buffer (destructive)
```

```
1:2<- Returns oldest message
Status after reading:
0:2_1:3_1:4_1:5_1:6_1:7_1:8_1:9_1:10_1:11_
        0:2 Its slot is now empty
Another message stored <-No overcapacity error since there is space
Status before reading:
1:24_1:3_1:4_1:5_1:6_1:7_1:8_1:9_1:10_1:11_
        1:24 Latest message has filled empty slot
        Messages 12 to 23 have been lost- there was no space for them
Read message:
1:3 <- Returns oldest valid message, not first slot
Status after reading:
1:24_0:3_1:4_1:5_1:6_1:7_1:8_1:9_1:10_1:11_ <-another empty slot
Another message stored
Status before reading:
1:24_1:25_1:4_1:5_1:6_1:7_1:8_1:9_1:10_1:11_ <-1:25 slot filled by latest message
Read message:
1:4<- Returns oldest valid message, sequence continues…
Status after reading:
1:24_1:25_0:4_1:5_1:6_1:7_1:8_1:9_1:10_1:11_
Another message stored
Status before reading:
1:24_1:25_1:26_1:5_1:6_1:7_1:8_1:9_1:10_1:11_
Read message:
1:5
Status after reading:
1:24_1:25_1:26_0:5_1:6_1:7_1:8_1:9_1:10_1:11_
Another message stored
Status before reading:
1:24_1:25_1:26_1:27_1:6_1:7_1:8_1:9_1:10_1:11_
Read message:
1:6
Status after reading:
1:24_1:25_1:26_1:27_0:6_1:7_1:8_1:9_1:10_1:11_
Another message stored
Status before reading:
1:24_1:25_1:26_1:27_1:28_1:7_1:8_1:9_1:10_1:11_
Read message:
1:7
Status after reading:
1:24_1:25_1:26_1:27_1:28_0:7_1:8_1:9_1:10_1:11_
Another message stored
Status before reading:
1:24_1:25_1:26_1:27_1:28_1:29_1:8_1:9_1:10_1:11_
Read message:
1:8
Status after reading:
1:24_1:25_1:26_1:27_1:28_1:29_0:8_1:9_1:10_1:11_
Another message stored
Status before reading:
1:24_1:25_1:26_1:27_1:28_1:29_1:30_1:9_1:10_1:11_
Read message:
1:9
```

```
Status after reading:
1:24_1:25_1:26_1:27_1:28_1:29_1:30_0:9_1:10_1:11_
Another message stored
Status before reading:
1:24_1:25_1:26_1:27_1:28_1:29_1:30_1:31_1:10_1:11_
Read message:
1:10
Status after reading:
1:24_1:25_1:26_1:27_1:28_1:29_1:30_1:31_0:10_1:11_
Another message stored
Status before reading:
1:24_1:25_1:26_1:27_1:28_1:29_1:30_1:31_1:32_1:11_
Read message:
1:11
Status after reading:
1:24_1:25_1:26_1:27_1:28_1:29_1:30_1:31_1:32_0:11_
Another message stored
Status before reading:
1:24_1:25_1:26_1:27_1:28_1:29_1:30_1:31_1:32_1:33_
Read message:
Note: reading from slot zero again <-Messages read until end of storage, oldest messages now at first slot again
1:24<- Returns oldest valid message, not first slot. This sequence continues…
Status after reading:
0:24_1:25_1:26_1:27_1:28_1:29_1:30_1:31_1:32_1:33_
Another message stored
Status before reading:
1:34_1:25_1:26_1:27_1:28_1:29_1:30_1:31_1:32_1:33_
Read message:
1:25
Status after reading:
1:34_0:25_1:26_1:27_1:28_1:29_1:30_1:31_1:32_1:33_
Another message stored
Status before reading:
1:34_1:35_1:26_1:27_1:28_1:29_1:30_1:31_1:32_1:33_
Read message:
1:26
Status after reading:
1:34_1:35_0:26_1:27_1:28_1:29_1:30_1:31_1:32_1:33_
Another message stored
Status before reading:
1:34_1:35_1:36_1:27_1:28_1:29_1:30_1:31_1:32_1:33_
Read message:
1:27
Status after reading:
1:34_1:35_1:36_0:27_1:28_1:29_1:30_1:31_1:32_1:33_
Another message stored
Status before reading:
1:34_1:35_1:36_1:37_1:28_1:29_1:30_1:31_1:32_1:33_
Read message:
1:28
Status after reading:
1:34_1:35_1:36_1:37_0:28_1:29_1:30_1:31_1:32_1:33_
```

```
Another message stored
Status before reading:
1:34_1:35_1:36_1:37_1:38_1:29_1:30_1:31_1:32_1:33_
Read message:
1:29
Status after reading:
1:34_1:35_1:36_1:37_1:38_0:29_1:30_1:31_1:32_1:33_
Another message stored
Status before reading:
1:34_1:35_1:36_1:37_1:38_1:39_1:30_1:31_1:32_1:33_
Read message:
1:30
Status after reading:
1:34_1:35_1:36_1:37_1:38_1:39_0:30_1:31_1:32_1:33_
Another message stored
Status before reading:
1:34_1:35_1:36_1:37_1:38_1:39_1:40_1:31_1:32_1:33_
Read message:
1:31
Status after reading:
1:34_1:35_1:36_1:37_1:38_1:39_1:40_0:31_1:32_1:33_
Another message stored
Status before reading:
1:34_1:35_1:36_1:37_1:38_1:39_1:40_1:41_1:32_1:33_
Read message:
1:32
        STOP STORING
Status after reading:
1:34_1:35_1:36_1:37_1:38_1:39_1:40_1:41_0:32_1:33_
Status before reading:
1:34_1:35_1:36_1:37_1:38_1:39_1:40_1:41_0:32_1:33_<--No change, nothing stored
Read message:
1:33    <- Reading continues to return oldest message…
Status after reading:
1:34_1:35_1:36_1:37_1:38_1:39_1:40_1:41_0:32_0:33_
Status before reading:
1:34_1:35_1:36_1:37_1:38_1:39_1:40_1:41_0:32_0:33_
Read message:
Note: reading from slot zero again
1:34
Status after reading:
0:34_1:35_1:36_1:37_1:38_1:39_1:40_1:41_0:32_0:33_
Status before reading:
0:34_1:35_1:36_1:37_1:38_1:39_1:40_1:41_0:32_0:33_
Read message:
1:35
Status after reading:
0:34_0:35_1:36_1:37_1:38_1:39_1:40_1:41_0:32_0:33_
Status before reading:
0:34_0:35_1:36_1:37_1:38_1:39_1:40_1:41_0:32_0:33_
Read message:
1:36
Status after reading:
```

```
0:34_0:35_0:36_1:37_1:38_1:39_1:40_1:41_0:32_0:33_
Status before reading:
0:34_0:35_0:36_1:37_1:38_1:39_1:40_1:41_0:32_0:33_
Read message:
1:37
Status after reading:
0:34_0:35_0:36_0:37_1:38_1:39_1:40_1:41_0:32_0:33_
Status before reading:
0:34_0:35_0:36_0:37_1:38_1:39_1:40_1:41_0:32_0:33_
Read message:
1:38
Status after reading:
0:34_0:35_0:36_0:37_0:38_1:39_1:40_1:41_0:32_0:33_
Status before reading:
0:34_0:35_0:36_0:37_0:38_1:39_1:40_1:41_0:32_0:33_
Read message:
1:39
Status after reading:
0:34_0:35_0:36_0:37_0:38_0:39_1:40_1:41_0:32_0:33_
Status before reading:
0:34_0:35_0:36_0:37_0:38_0:39_1:40_1:41_0:32_0:33_
Read message:
1:40
Status after reading:
0:34_0:35_0:36_0:37_0:38_0:39_0:40_1:41_0:32_0:33_
Status before reading:
0:34_0:35_0:36_0:37_0:38_0:39_0:40_1:41_0:32_0:33_
Read message:
1:41 <-Returns last message left in buffer
Status after reading:
0:34_0:35_0:36_0:37_0:38_0:39_0:40_0:41_0:32_0:33_
        Buffer now empty (no valid messages)
Status before reading:
0:34_0:35_0:36_0:37_0:38_0:39_0:40_0:41_0:32_0:33_
Read message:
Note: reading from slot zero again <- No valid messages upwards of previous slot, resets to first slot
Error- Nothing in ringBuffer!!!!!!!!! <- None from first slot either, not really an error just an empty buffer
0:34<- read function must return something, so returns first invalid message to indicate null
Status after reading:
0:34_0:35_0:36_0:37_0:38_0:39_0:40_0:41_0:32_0:33_
Status before reading:
0:34_0:35_0:36_0:37_0:38_0:39_0:40_0:41_0:32_0:33_
Read message:
Note: reading from slot zero again
Error- Nothing in ringBuffer!!!!!!!!!<-- this continues
0:34
```

ACKBUFFER

Required changing buffer output from handleMessage to showMessage to observe output via serial.
The code used to implement this test can be found in: testingNodes\ackbuff_unit_1
Serial printout in bold

Test 1:
Create new buffer, set period to 1 to allow output
Show buffer status:
0:0_0:0_0:0_0:0_0:0_0:0_0:0_0:0_0:0_0:0_
Empty
Call output handleWaiting():
Nothing, it's empty

Test 2:
Add a message to the buffer: STD, REMOTE, PID, SET, 255, 255, true
Show buffer status:
1:19_0:0_0:0_0:0_0:0_0:0_0:0_0:0_0:0_0:0_
1:19 – 1 valid message (.valid = 1), with source byte remote (19)
Call output handleWaiting():
90_19_31_7_255_255
Outputs message stored earlier, with start flagged as ACK(90)

Test 3:
Check if message (STD, REMOTE, PID, SET, 255, 255, true) is still in buffer:
1
Yes
Add same message to the buffer: STD, REMOTE, PID, SET, 255, 255, true
Show buffer status:
1:19_0:0_0:0_0:0_0:0_0:0_0:0_0:0_0:0_0:0_
Has not added a duplicate message
Call output handleWaiting():
90_19_31_7_255_255
Sends message only once

Test 4:
Check if message (STD, IMU, PID, SET, 255, 254, true) is in buffer:
0
It is not, the source and DAT1 are different

Test 5:
Add this message (STD, IMU, PID, SET, 255, 255, true) to the buffer
1:19_1:18_0:0_0:0_0:0_0:0_0:0_0:0_0:0_0:0_
It's status now shows two messages, with sources 19, 18 (remote, IMU)
Call output handleWaiting():
90_19_31_7_255_255
90_18_31_7_255_254
Sends both messages flagged as ACK (90)

Test 6:
Clear message (STD, REMOTE, PID, SET, 255, 254, true), using .clear(message).
1:19_1:18_0:0_0:0_0:0_0:0_0:0_0:0_0:0_0:0_
No change, that exact message was never stored

90_19_31_7_255_255
90_18_31_7_255_254
Continues sending both messages

Test 7:
Clear message (STD, REMOTE, PID, SET, 255, 255, true), using .clear(message).
0:19_1:18_0:0_0:0_0:0_0:0_0:0_0:0_0:0_0:0_
That message cleared, (.valid = 0)
90_18_31_7_255_254
handleWaiting () sent only remaining message

Test 8:
Add that message again (STD, REMOTE, PID, SET, 255, 255, true)
1:19_1:18_0:0_0:0_0:0_0:0_0:0_0:0_0:0_0:0_
Its back in the buffer
Cancel message (STD, REMOTE, PID, SET, 47, 52, true), using .cancel(message).
0:19_1:18_0:0_0:0_0:0_0:0_0:0_0:0_0:0_0:0_
Message with same START, SRC, DEST, CMD has been removed, regardless of data values
90_18_31_7_255_254
handleWaiting () sends only remaining message

Test 9:
Set period to 0:
No output, sending from the buffer has been disabled