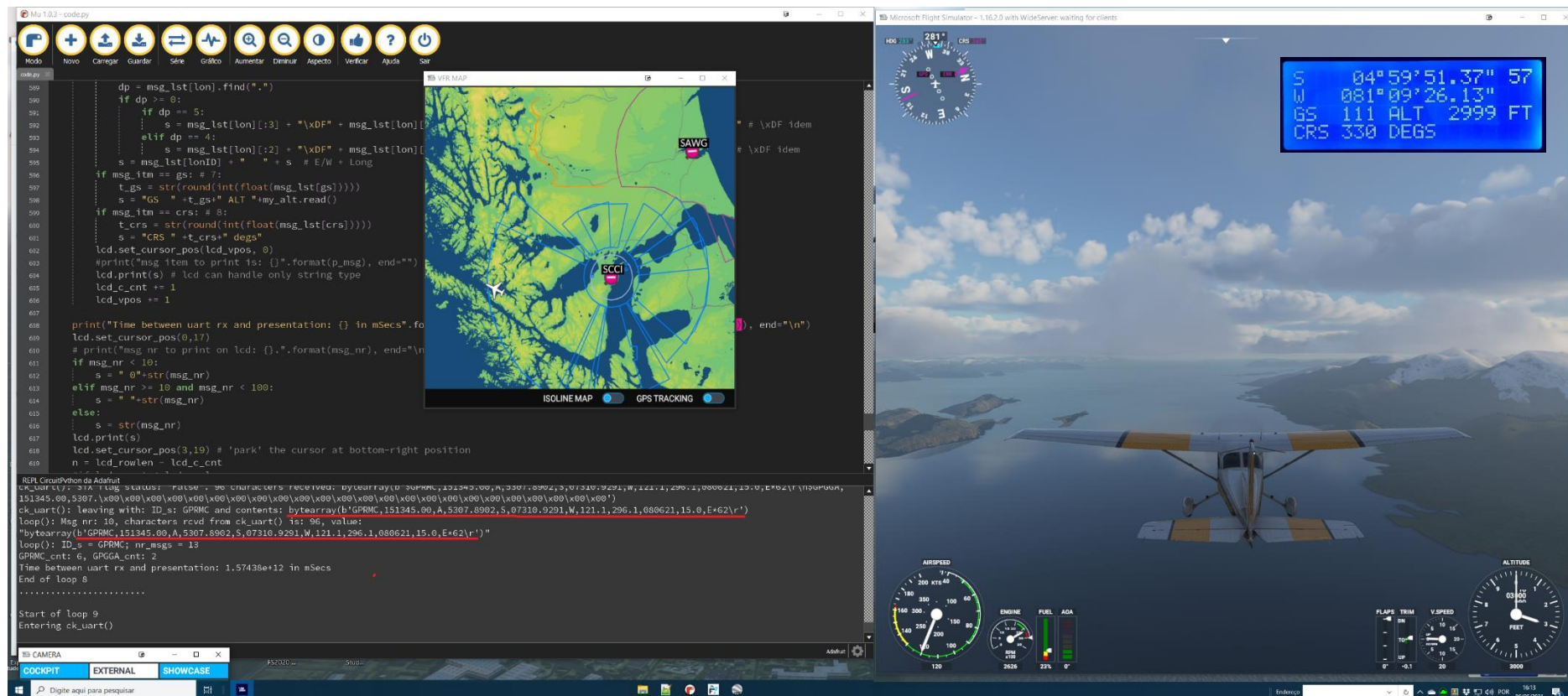


Project: Raspberry Pi Pico MS Flightsimulator GPS datagram receiver of GPRMC and GPGLGA types of GPS messages

This is a report relating my current project using a Raspberry Pi Pico with CircuitPython (CP) firmware to asynchronously receive, analyze, extract, save and present GPS datagram messages broadcasted during a Microsoft Flightsimulator 2020 (MSFS2020) ⁱ session using FSUIPC7's 'GPSout...' functionality. FSUIPC7 by Pete & John Dowson ^{viii}.

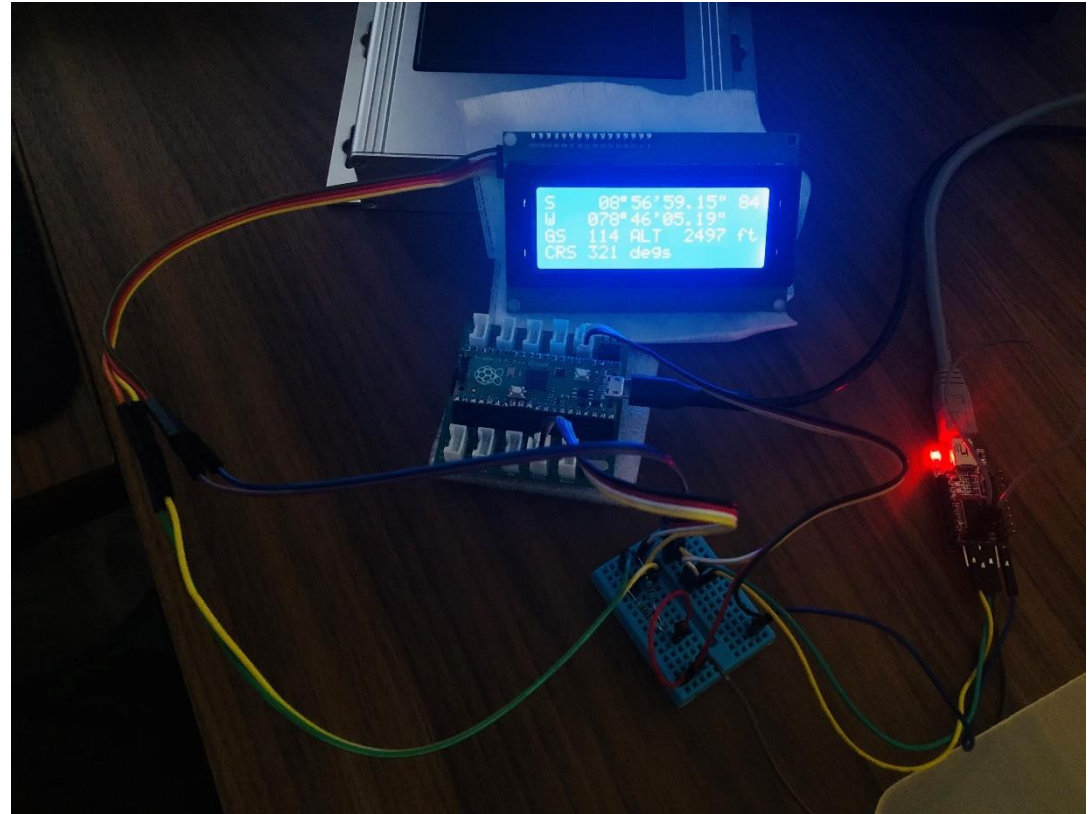
The image below: shows on the right: an ‘External’ view of a running session of MSFS2020. Inset, top right: output to the 4x20 character LCD (note: the text on the LCD image is not of moment as the details MSFS2020 are displaying). On the left the Integrated Development Environment (Mu IDE) with an inset of the map of MSFS2020). In the bottom part of the IDE, the output of the CP-script running.



Project: Raspberry Pi Pico MS Flightsimulator GPS datagram receiver of GPRMC and GPGLA types of GPS messages

The hardware used:

- A Raspberry Pi Pico RP2040 microcontroller board and its built-in LED ⁱⁱ;
- A Seeed Grove shield for Raspberry Pi Pico ⁱⁱⁱ;
- A 4x20 character LCD Hitachi model HD44780 with piggyback I2C expander ^{iv};
- An USB-to-RS232 converter (FTDI FT232RL) ^v;
- 2 x I2C pull-up resistors, each 5.1 kOhm (SDA, SCL lines);
- One USB-A to microUSB cable (connection PC – RPi Pico);
- One USB-A to mini-B cable (connection PC – RS232 converter).
- A breadboard to make the interconnections and install the two pull-up resistors;
- Various interconnection cables for breadboard use.









Note: at the end of this document is presented a list of links, indicated by the endnote markings throughout the text of this document (expressed in Roman numerals).




Project: Raspberry Pi Pico MS Flightsimulator GPS datagram receiver of GPRMC and GPGGA types of GPS messages

The software:





> CIRCUITPY (K:)

<input type="checkbox"/> Nome	Data de modificação	Tipo	Tamanho
 .fseventsd	31/12/2019 23:00	Pasta de arquiv...	
 lib	31/12/2019 23:00	Pasta de arquiv...	
 .metadata_never_index	31/12/2019 23:00	Arquivo METAD...	0 KB
 .Trashes	31/12/2019 23:00	Arquivo TRASH...	0 KB
 boot_out.txt	31/12/2019 23:00	Documento de ...	1 KB
 code.py	09/06/2021 13:19	Arquivo Fonte ...	16 KB



> CIRCUITPY (K:) > lib

<input type="checkbox"/> Nome	Data de modificação	Tipo	Tamanho
 adafruit_bus_device	02/06/2021 19:55	Pasta de arquiv...	
 lcd	02/06/2021 19:46	Pasta de arquiv...	
 __init__.py	01/06/2021 05:58	Python File	0 KB

> CIRCUITPY (K:) > lib > lcd


<input type="checkbox"/> Nome	Data de modificação	Tipo	Tamanho
 __init__.py	23/07/2020 16:01	Python File	0 KB
 i2c_pcf8574_interface.py	02/06/2021 21:10	Python File	5 KB
 lcd.py	15/01/2021 13:00	Python File	10 KB
 README.rst	23/07/2020 16:01	Arquivo RST	2 KB

> CIRCUITPY (K:) > lib > adafruit_bus_device

<input type="checkbox"/> Nome	Data d
 __init__.py	06/01,
 i2c_device.py	01/06,

Project: Raspberry Pi Pico MS Flightsimulator GPS datagram receiver of GPRMC and GPGGA types of GPS messages

Good information about CircuitPython: ^{vi}. The CircuitPython firmware that I used for this project is:

<input checked="" type="checkbox"/> 	adafruit-circuitpython-raspberrypi_pico-en_GB-20210604-bc014ce.uf2	06/06/2021 19:42	Arquivo UF2	1 282 KB
---	--	------------------	-------------	----------

The hardware connections are as follows:

PC to RPi Pico through a USB-A to micro-USB cable;

PC to RS232 converter through a USB-A to mini-B connector;

Seeed Grove shield for Raspi Pico to 4x20 LCD via Grove connector "I2C1", via a 4-wire cable with a Grove connector on one end and 4 female maker connectors on the other end;

RS232 converter to RPi Pico UART0 via Seeed Grove shield Grove connector UART0 (TX GPIO0, RX GPIO1);

Note that the RS232 connections are crossed wires:

RS232 TX → to the Seeed Grove shield UART0 connector, yellow wire (connected to the Pico RX (GPIO1);

RS232 RX → to the Seeed Grove shield UART0 connector, white wire (connected to the Pico TX (GPIO0);

RS232 Vcc (5V) → to the Seeed Grove shield UART0 connector, red wire;

RS232 GND → to the Seeed Grove shield UART0 connector black wire;

The I2C connection from the Seeed Grove shield I2C1 connector SDA (and a 5.1 kOhm 'pull-up' resistor to +Vcc (5 Volt);

Also the I2C1 connector SCL (and a 5.1 kOhm 'pull-up' resistor to Vcc (5 Volt).

Project: Raspberry Pi Pico MS Flightsimulator GPS datagram receiver of GPRMC and GPGGA types of GPS messages

(Note: the PC I used during this project's creation, testing and debugging is a custom desktop PC i7, 64GB RAM, 512GB SSD, MS Windows 10 Pro operating system).

Information about the various types of GPS messages: ^{vii}.

I use the Pete & John Dowson's FSUIPC7 application^{viii}, to broadcast two types of GPS datagrams to the Raspberry Pi Pico:

Type 1 GPS datagram: GPRMC;

Type 2 GPS datagram: GPGGA.

These two GPS datagram types consist of the following compositions:

Example of a GPS datagram of type GPRMC:

0	1	2	3	4	5	6	7	8	9	10	11	12
<code>b'GPRMC,195059.99,A,5501.6522,S,06542.9995,W,0.0,6.9,040621,10.7,E*65\r'</code>												

Example of a GPS datagram of type GPGGA:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
<code>b'GPGGA,192114.00,5314.4627,S,07424.3883,W,1,05,0.0,884.2,M,0.0,M,0.0,0000*71\r'</code>														

As we see, the items of the datagrams are delimited by comma (,) characters

Project: Raspberry Pi Pico MS Flightsimulator GPS datagram receiver of GPRMC and GPGLA types of GPS messages

From the GPRMC datagram we use the following items:

- ID;
- Latitude (in: degrees° minutes' and seconds")
- Latitude ID (N/S);
- Longitude;
- Longitude ID (E/W);
- Groundspeed (Kts);
- Course (Degrees).

From the GPGLA datagram we use only the altitude data:

- Altitude (MSL in meters. The python script converts the meters value to feet).
The script does not take into account when flying flightlevels, e.g.: 'FL100'.

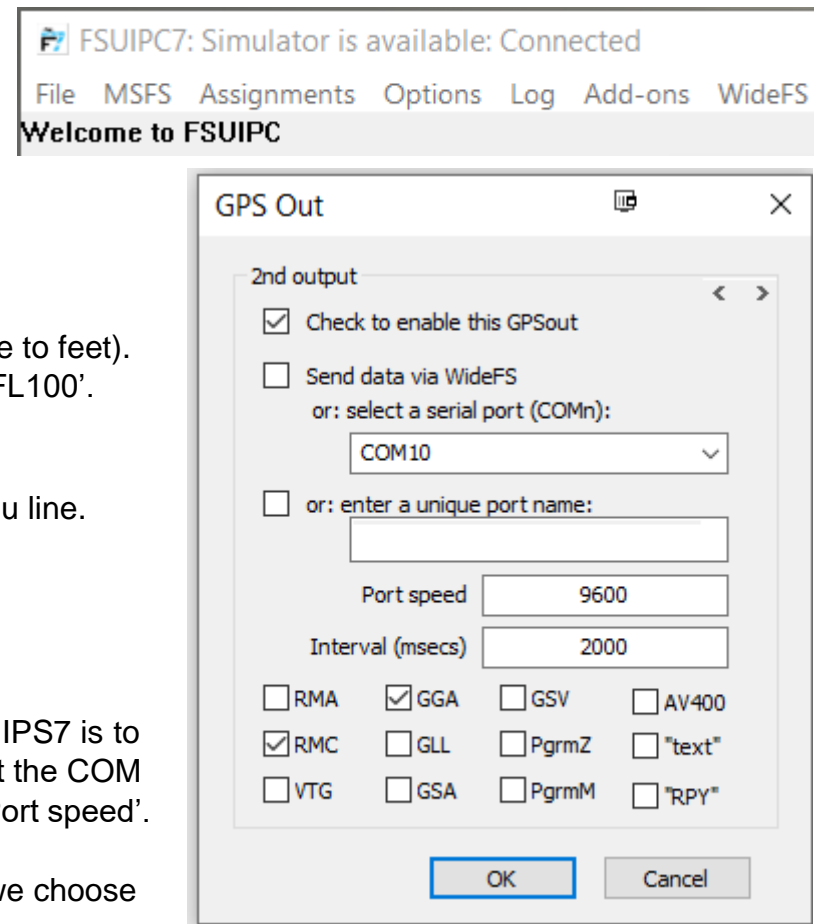
The screenshot to the right shows a screenshot of the FSUIPC7 main menu line. Below it a screenshot of the (2nd) GPSout interface page.

Menu-item: > Options > GPS out... FSUIPC7 has 2 pages for GPS output

I used the '2nd output' page, but if you prefer you can use the '1st output'.

All that is needed to provide that GPS datagrams will be broadcast by FSUIPS7 is to choose the datagram type(s), in this project: 'GGA' and 'RMC'. Then select the COM port of the PC where the datastream will be directed to. Then we set the 'Port speed'. Initially I used 4800 baud, but I experienced that 9600 baud gives a better performance. At a baudrate of 4800 we will lose messages sent. Finally we choose the broadcast interval: I choose the default interval of 200 milliseconds.

After the settings for GPSout... have been set, confirm your settings by clicking on the OK button.



Project: Raspberry Pi Pico MS Flightsimulator GPS datagram receiver of GPRMC and GPGGA types of GPS messages

The CircuitPython script:

The script currently contains 385 lines of code. The script occupies 12 kilobytes on the RPi Pico 'disk'.

The script consists of eight functions (shown as function name and eventual result):

```
loop() # (void)
ck_uart() # (nr_bytes received)
bfr_fnd(int) # (int index to rx_buffer)
split_types() # (bool)
led_toggle() # (void)
empty_buffer() # (void)
lcd_pr_msgs() # (void)
main() # (void)
```

For the reception a bytearray buffer 'rx_buffer' of 160 bytes is used. Also a class object has been created to hold the received, analyzed and extracted GPS datagram data. This class object has the name: `gps_msgs`.

At the initialization an instance of this class is created. The name is: `my_msgs`. To access the GPRMC type of data inside this class we have to call the methods with a first integer parameter: 0 for GPRMC messages and 1 for GPGGA messages.

Project: Raspberry Pi Pico MS Flightsimulator GPS datagram receiver of GPRMC and GPGGA types of GPS messages

Below the `gps_msgs` class definition:

```
class gps_msgs:
    def __init__(self):
        # ID, Lat, N/S, Lon, E/W, GS, CRS, ALT
        self.gps = ["", "", "", "", "", "", "", ""]

    def write(self, s):
        tp = isinstance(s, list)
        if tp == True:
            self.gps[0] = s[0] # ID
            self.gps[1] = s[1] # Lat
            self.gps[2] = s[2] # LatID N/S
            self.gps[3] = s[3] # Lon
            self.gps[4] = s[4] # LonID E/W
            self.gps[5] = s[5] # GS
            self.gps[6] = s[6] # CRS
            self.gps[7] = s[7] # Alt

    def read(self, n):
        tp = isinstance(n, type(None))
        if tp == True:
            n = 0
        if n >= 0 and n <= 7:
            return self.gps[n]
        else:
            return self.gps

    def clean(self):
        self.gps[0] = ""
        self.gps[1] = ""
```


Project: Raspberry Pi Pico MS Flightsimulator GPS datagram receiver of GPRMC and GPGGA types of GPS messages

```
self.gps[2] = ""  
self.gps[3] = ""  
self.gps[4] = ""  
self.gps[5] = ""  
self.gps[6] = ""  
self.gps[7] = ""
```

The datagram datastream is received in the function `ck_uart()`.

Reception by the UART is put into the `rx_buffer`. See a part of the function `ck_uart()` on the next page.

Project: Raspberry Pi Pico MS Flightsimulator GPS datagram receiver of GPRMC and GPGGA types of GPS messages

The ck_uart() function:

```
def ck_uart():
    global rx_buffer, loop_time
    nr_bytes = i = 0
    delay_ms = 0.2
    while True:
        nr_bytes = uart.readinto(rx_buffer)
        #print("ck_uart(): rcvd data: {}".format(rx_buffer),end="\n")
        loop_time = monotonic_ns()
        if not nr_bytes:
            sleep(delay_ms)
            continue
        if nr_bytes > 1:
            return nr_bytes
        elif nr_bytes == 1:
            if rx_buffer[0] == b'\x00':
                sleep(delay_ms)
                i += 1
                if i % 1000 == 0:
                    print("Waiting for uart line to become ready")
            else:
                empty_buffer()
                sleep(delay_ms)
                continue
    #return nr_bytes
```

Project: Raspberry Pi Pico MS Flightsimulator GPS datagram receiver of GPRMC and GPGLA types of GPS messages

After `ck_uart()` has put the received datagram stream into the `rx_buffer`, program flow will pass back to the main 'loop()' function.

Next the contents of the `rx_buffer` is being analyzed in the function '`split_types()`'

This function first scans `rx_buffer` for the presence of a '\$' character, using the call: '`bfr_fnd(38)`'. If the search succeeds the function returns the index to the \$ character in `rx_buffer`. If the search failed the function returns -1. Both types of datagrams are identified in the first field by a \$ character followed by 5 characters, in our case: 'GPRMC' or 'GPGLA'. The fields are delimited by a comma (44 decimal or 2C hexadecimal). The end of the datagram is identified by a carriage return character '\r' (13 decimal or 0D hexadecimal). The carriage return character is searched using the call: '`bfr_fnd(13)`'. In Python language bytearrays can be sliced. We use the slice method ^{ix} to cut out the part of the `rx_buffer` stream that we need from the GPRMC datagram message. The same we do for the GPGLA datagram message.

An example: received datastream into `rx_buffer` is:

```
bytearray(b'$GPRMC,140714.00,A,4131.2257,S,07346.2442,W,117.1,307.8,080621,9.5,E*58\r\n$GPGLA,140714.00,4131.2257,S,07346.2442,W,1,05,0.0,915.5,M,0.0,M,0.0,0000*74\r\n\x00\x00\x00\x00\x00\x00\x00\x00')
```

From this datastream we extract as GPRMC datagram the part below shown in red color:

```
bytearray(b'$GPRMC,140714.00,A,4131.2257,S,07346.2442,W,117.1,307.8,080621,9.5,E*58\r\n$GPGLA,140714.00,4131.2257,S,07346.2442,W,1,05,0.0,915.5,M,0.0,M,0.0,0000*74\r\n\x00\x00\x00\x00\x00\x00\x00\x00')
```

The same we do to extract the GPGLA datagram, also shown in red color.

```
bytearray(b'$GPRMC,140714.00,A,4131.2257,S,07346.2442,W,117.1,307.8,080621,9.5,E*58\r\n$GPGLA,140714.00,4131.2257,S,07346.2442,W,1,05,0.0,915.5,M,0.0,M,0.0,0000*74\r\n\x00\x00\x00\x00\x00\x00\x00\x00')
```

After this operation, the bytearray extracted then is converted to a text string, using a command like : `s = bytearray.decode('utf-8')`. Only we defined as a global variable "encoding = 'utf-8' ".

The resulting text string is then split into a list object, using a command like: `list = s.split(",")`. The altitude data from the GPGLA message is as last item added to the list through a `list.append()` command.

Project: Raspberry Pi Pico MS Flightsimulator GPS datagram receiver of GPRMC and GPGGA types of GPS messages

The contents of the list is written to the `my_msgs` class, using the method `my_msgs.write()`.

The `my_msgs` class has a method with the name “`read()`” which can be used to: a) extract all the datagram data saved inside this class; b) retrieve a specific field (lat, lon, gs, crs, alt) through the call: `my_msgs.read(n)`. If the received datagram data previously has been successfully written to the appropriate class, the call: `'my_msgs.read(0)'` will yield “`$GPRMC`”. If no data has been received previously or the data in the class has been wiped out as a preparation of a next reception loop, the call: `'my_msgs.read(0)'` will return an empty string. Note that to retrieve the complete datagram we have to call `'my_msgs.read()'` minimal with a integer value of greater than the object has data items (8). For example: `my_msgs.read(9)` will return list of eight items: `['$GPRMC', '0014.6015', 'N', '08021.6893', 'W', '124.5', '37.5', '2890']`.

Finally the function that writes the selected datagram data to the liquid crystal display is: `'lcd_pr_msgs()'`

In `lcd_pr_msgs()` are two calls to `led_toggle()`. This has the effect that the built-in led blinks while displaying the data. The lcd screen is in `lcd_pr_msgs()` only `clear()`'ed at startup, when the startup flag is -1. During the subsequent calls to `lcd_pr_msgs()` only the data on the lcd is overwritten. This has a nice easing effect.

Project: Raspberry Pi Pico MS Flightsimulator GPS datagram receiver of GPRMC and GPGGA types of GPS messages

The next screenshot contains a part of the `lcd_pr_msgs()` function:

```
if startup == -1:
    lcd.clear()
    lcd.set_cursor_pos(0,18)
    lcd.print("{:0>2d}".format(msg_nr))
    lcd_vpos = 0
    itms_lst = [lat_rmc, lon_rmc, gs, crs]
    led_toggle()
for msg_itm in itms_lst:
    if msg_itm == lat_rmc:
        lat_v = my_msgs.read(lat_rmc)
        dp = lat_v.find(".")
        if dp >= 0:
            if dp == 4:
                s = "{: >2s}\xDF{:0>2s}\'{:0>2s}.{:0>2s}\{}".format(lat_v[:2], lat_v[2:4], lat_v[5:7], lat_v[7:])
            elif dp == 3:
                s = "{: >2s}\xDF{:0>2s}\'{:0>2s}.{:0>2s}\{}".format(lat_v[:1], lat_v[1:3], lat_v[4:6], lat_v[6:])
            s = my_msgs.read(latID_rmc) + "    " + s
    if msg_itm == lon_rmc:
        lon_v = my_msgs.read(lon_rmc)
        dp = lon_v.find(".")
        if dp >= 0:
            if dp == 5:
                s = "{: >2s}\xDF{:0>2s}\'{:0>2s}.{:0>2s}\{}".format(lon_v[:3], lon_v[3:5], lon_v[6:8], lon_v[8:])
```

The average loop time between reception by the UART and the presentation on the LCD currently is: 265 milliseconds.

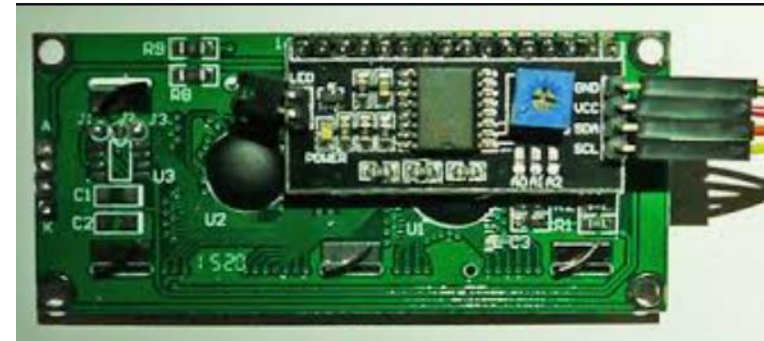
The ROM of the Hitachi LCD has the following character set:

Project: Raspberry Pi Pico MS Flight Simulator GPS datagram receiver of GPRMC and GPGLA types of GPS messages

Table 4 Correspondence between Character Codes and Character Patterns (ROM Code: A00)

Lower 4 Bits	Upper 4 Bits	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111	
xxxx0000	CG RAM (1)				0	a	P	`	P				-	タ	ミ	α	p	
xxxx0001	(2)			!	1	A	Q	a	q				。	ア	チ	△	ä	q
xxxx0010	(3)			"	2	B	R	b	r				「	イ	ツ	×	β	θ
xxxx0011	(4)			#	3	C	S	c	s				」	ウ	テ	ε	ε	ω
xxxx0100	(5)			\$	4	D	T	d	t				、	エ	ト	ヲ	μ	Ω
xxxx0101	(6)			%	5	E	U	e	u				・	オ	ナ	1	σ	Ü
xxxx0110	(7)			&	6	F	V	f	v				ヲ	カ	ニ	ヨ	ρ	Σ
xxxx0111	(8)			'	7	G	W	g	w				ア	キ	ヌ	ラ	g	π
xxxx1000	(1)			<	8	H	X	h	x				イ	ク	ネ	リ	フ	×
xxxx1001	(2)			>	9	I	Y	i	y				ッ	ケ	ル	ル	°	Y
xxxx1010	(3)			*	:	J	Z	j	z				エ	コ	ハ	レ	j	〒
xxxx1011	(4)			+	;	K	[k	{				オ	サ	ヒ	ロ	*	斤
xxxx1100	(5)			,	<	L	¥	l	l				ハ	シ	フ	ワ	Φ	円
xxxx1101	(6)			-	=	M]	m	}				ユ	ズ	ヘ	ン	も	÷
xxxx1110	(7)			.	>	N	^	n	→				ヨ	セ	ホ	°	ん	
xxxx1111	(8)			/	?	O	_	o	←				ッ	ソ	マ	°	ö	■

Note: The user can specify any pattern for character-generator RAM.



Hitachi HD44780 LCD device with piggyback I2C expander interface board

From this character set we use character addressed with the hexadecimal value \xDF to show a degrees character for the Latitude or Longitude position value.

Project: Raspberry Pi Pico MS Flightsimulator GPS datagram receiver of GPRMC and GPGLA types of GPS messages

To the right an image of datagram data being display on the LCD

The latest version, that uses print format commands that fills in spaces dynamically for Message number (msg_nr), GS, ALT and CRS. With msg_nr and CRS the filling is '0's.



To edit and test the Circuitpython script I used the Mu IDE (v. 1.0.3) ^x.

This completes this project report.
Thank you for taking time to read it.
Feedback is welcome.

Lisbon, June 10, 2021
@paulsk (Discord, Circuitpython group – deepdiver)

I created also a short .mp4 video to give an impression.

The CircuitPython script:

For the CircuitPython script listing see my repository on GitHub: https://github.com/PaulskPt/msfs2020_gps_rx

Project: Raspberry Pi Pico MS Flightsimulator GPS datagram receiver of GPRMC and GPGGA types of GPS messages

-
- i [Microsoft Flight Simulator - The next generation of one of the most beloved simulation franchises;](#)
 - ii [https://www.raspberrypi.org/products/raspberry-pi-pico/;](https://www.raspberrypi.org/products/raspberry-pi-pico/)
 - iii [RP2040,Raspberry Pi Pico - Seeed Studio;](#)
 - iv [IIC/I2C/TWI 2004 20x4 LCD Display Module for Arduino | eBay;](#)
 - v <https://www.ebay.com /itm/381374421597?hash=item58cbafda5d:g:k8AAOSwrklVMjlp;>
 - vi <https://learn.adafruit.com/welcome-to-circuitpython;>
 - vii <http://aprs.gids.nl/nmea/#rma;>
 - viii [Pete & John Dowson's Software \(petedowson.info\);](#)
 - ix [Python Bytes, Bytearray - w3resource;](#)
 - x [Code With Mu .](#)