Some Thoughts on “Real-Time” SSTV Processing

How to improve our present SSTV programs.

By Lionel and Roland (F2DC) Cordesses

Picture transmission—SSTV and fax—has been a part of the Amateur Radio world for many decades. The receiving end changed dramatically with the availability of PC-based software at the beginning of the 1990s. However, the methods used to demodulate the SSTV signal, that is, to measure the frequency of the received tone carrying the luminance, are not so different from those used 10 years ago. Today, microprocessors are powerful enough to run efficient “real-time” algorithms that yesterday needed a digital signal processor (DSP). Modern available computing power allows better processing of raw SSTV signals and better picture quality.

In this paper, we present some unconventional approaches to process and use synchronization signals as well as to extract luminance information. These processing methods significantly improve picture quality when receiving conditions are poor—in the presence of noise, QRM and so forth. They also perform an on-line, accurate estimation of the sound-card sampling frequency, circumventing the calibration step needed in many—if not all—varieties of SSTV software. A dedicated program using the described algorithms has been designed and tested not only during simulations but also on real on-the-air SSTV signals.

An Overview of SSTV

While this paper focuses on the processing of the Martin M1 signal, specifications of which were kindly provided to us by its author G3OQD,[1] it is clear that ideas and algorithms presented here can be applied to other SSTV formats. We will first present some methods that are used to demodulate SSTV signals and then recall some technical specifications of the M1 mode.

SSTV Demodulation

The purpose of any color SSTV decoder is to recover the red, green and blue original information from the frequency-modulated signal and to accurately extract the video line with respect to the horizontal synchronization signal. This has long been done using analog approaches.[2]

Before the all-digital era, a hybrid approach was used. The demodulator was still relying upon an analog circuit based on filters, adders and so on. Once demodulated, this analog signal was digitized. Further processing, such as image processing, was then carried out on a computer. This was done back in 1970.[3] Later, digital processing of the demodulated signal became the basis of some computer-based SSTV solutions. See “Viewport VGA” and our own solution based upon the same hardware.[4, 5]

The all-digital approach first converts the analog signal from the receiver into a digital signal, using an analog-to-digital converter (ADC). Then, all the DSP demodulation and synchronization detection is performed on a computer. The two-level ADC, often a simple op amp, became one of the bases of much software—*JVFax*, for instance.[6]

Then came sound cards. Thanks to low-cost hardware, the sampling of

[1]Notes appear on page 19.

26 rue du Montant
F63540 Romagnat
France
roland.cordesses@free.fr

2 allee Dauphine
F78140 Velizy
France
cordesses@renagri.com

analog signals was possible. Most of the actual sound cards can reach a 44.1-kHz sampling frequency with 16 bits per sample. A new generation of software makes use of the sound card: *JVComm32* is one of them.

Our approach clearly belongs to the last one—the all-digital one. After 10 years of experiments with SSTV demodulation for one author and more than thirty for the other, we really think there is room for improvement. Perhaps a lot of room, we think.

When receiving conditions are poor and the signal-to-noise ratio (S/N) is low, much of the available software performs poorly. Sometimes it is not able to receive a faint picture you can hear in the background. Other times, it does receive something. Unfortunately, most software is unable to synchronize properly under these conditions. We decided to focus on these situations and attempt an answer to the underlying technical questions. So, if you are interested in SSTV, or just curious to see what DSP can bring to such an old standard, go on reading!

The Martin M1 SSTV Signal

Let's briefly recall the specifications of such a SSTV signal. The original picture is in color, described by its red, green and blue spectral components. The size of the picture is 256× 256 pixels. Each spectral component is transmitted in turn; that is, the 256 green pixels signal first, the blue and then the red. As in McDonald's original system (see Note 2), the modulation is a frequency modulation described by Eq 1.

$$f_l = f_{\text{black}} + \frac{(f_{\text{white}} - f_{\text{black}})lum}{max_{\text{lum}}} \qquad \text{(Eq 1)}$$

where: f_{black} is the frequency of a black pixel (1500 Hz), f_{white} is the frequency of a white pixel (2300 Hz), and max_{lum} is the maximum value of the luminance signal. A usual value is 255 for pictures coded with eight bits per component (and thus 8 × 3 = 24 bits for red, green and blue). *lum* is the value of the luminance and f_l [read "*f*" subscript lower-case "L"—*Ed.*] is the corresponding modulating frequency.

One line of the original picture is transmitted according to the following timing (see Fig 1). The horizontal line-synchronization signal (f_{sync} = 1200 Hz) lasts t_s = 4862 µs. During t_1 = 572 µs, a 1500-Hz tone is transmitted. The green component of the line is then transmitted. Each pixel lasts 572 µs, thus the green signal lasts t_G = 256 × 572 = 146,432 µs.

During t_1 = 572 µs, a 1500-Hz tone is transmitted. The blue component of the line is then transmitted. Each pixel lasts 572 µs, thus the blue signal lasts t_G = 256 × 572 = 146,432 µs.

During t_1 = 572 µs, a 1500-Hz tone is transmitted. Lastly, the red component of the line is transmitted. Each pixel lasts 572 µs, thus the red signal lasts t_G = 256 × 572 = 146,432 µs. During t_1 =572 µs, a 1500-Hz tone is transmitted. This is the end of the line.

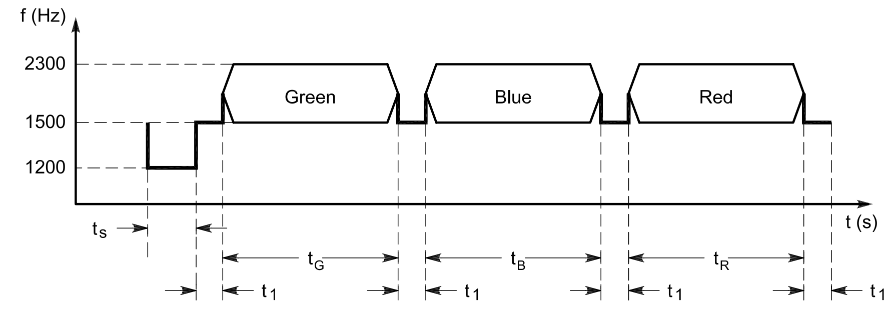


Fig 1—Martin M1 line timing.

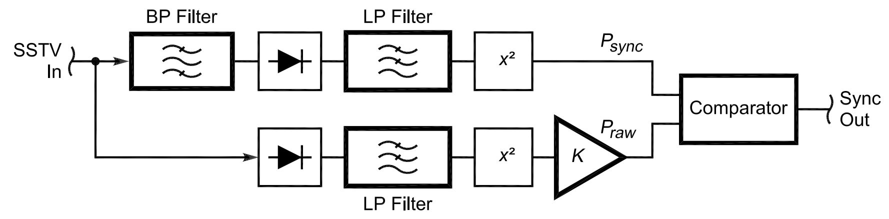


Fig 2—Synchronization detector overview.

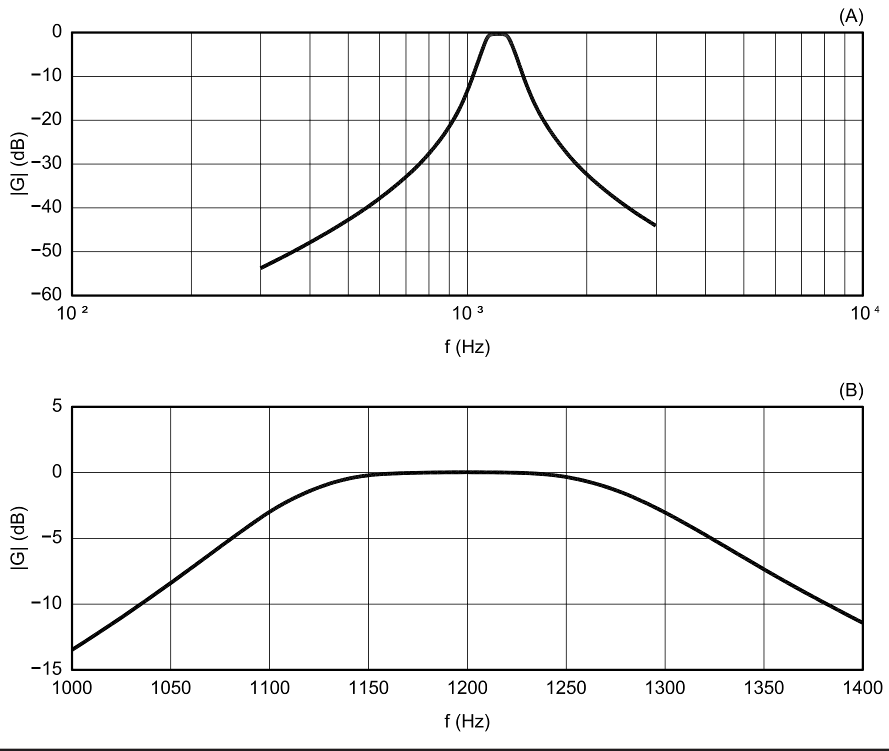


Fig 3—Frequency response of the synchronization filter. (A) shows the whole response curve, while (B) is a close-up of the peak region.

Horizontal Synchronization Method

This section is devoted to the horizontal synchronization algorithm we have developed. First, we consider the need for such a software module. We then describe the synchronization detector. It is based upon a band-pass filter followed by a linear Hough transform. The linear Hough transform is explained in a step-by-step approach.

The Need for Horizontal Synchronization

The reader may wonder why we need such a synchronization detector. Thanks to digital headers transmitted before the image signal (referred to as VIS code), software is able to detect the beginning of a new image. It also extracts from the digital header the mode used by the transmitter. Then, as the receiving software relies upon a calibrated time base, it *asynchronously* decodes the incoming signal, and it does not use the horizontal-synchronization signal any more.

Remember that we have decided to focus on realistic receiving conditions. The above mentioned method fails, for instance:

- When fading occurs during the VIS,
- When QRM prevents the software from detecting the start of image signal or
- When the operator misses the beginning of the transmission.

We will see that the forgotten horizontal-synchronization signal can drastically improve SSTV demodulation.

The Synchronization Filter: The analog SSTV signal is first digitized—thanks to the computer sound card—at $f_s = 44{,}100$ Hz. The resulting data are then processed by the synchronization detector presented in Fig 2. This incoming SSTV signal goes through a digital recursive band-pass filter (also known as an infinite-impulse response filter or IIR).[7] We have chosen a four-pole Butterworth filter for its burst time response.[8] The center frequency is $f_{\text{sync}} = 1200$ Hz and the bandwidth is 200 Hz. The coefficients of the IIR filter are given in Table 1. The output $y[n]$ of the IIR filter is computed using Eq 2.

$$\begin{aligned} a_0 y[n] &= b_0 x[n] + b_1 x[n-1] \\ &\quad + b_2 x[n-2] + b_3 x[n-3] \\ &\quad + b_4 x[n-4] - a_1 y[n-1] \\ &\quad - a_2 y[n-2] - a_3 y[n-3] \\ &\quad - a_4 y[n-4] \end{aligned} \qquad \text{(Eq 2)}$$

The frequency response of this filter is plotted in Fig 3. The upper curve displays the frequency response from 300 Hz to 3000 Hz, and the lower curve

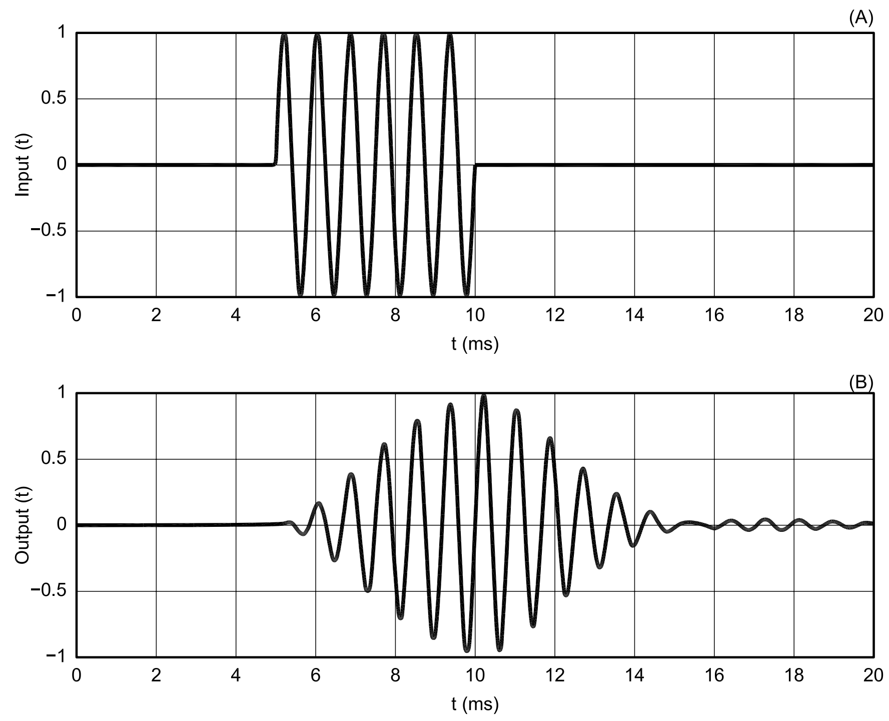


Fig 4—Burst response of the synchronization filter. (A) shows the 1200-Hz burst; (B) shows the burst as shaped by the filter response.

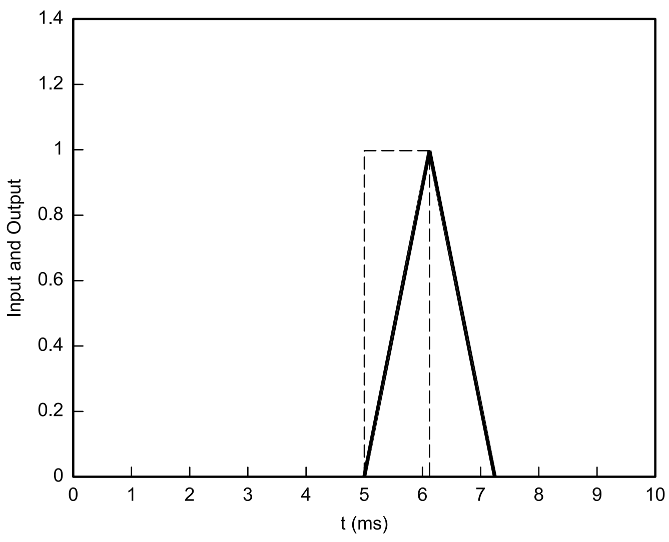


Fig 5—Time response (solid) of the FIR filter to a rectangular window dotted.

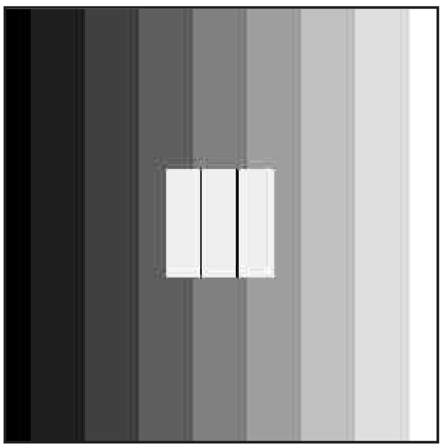


Fig 6—The reference picture.

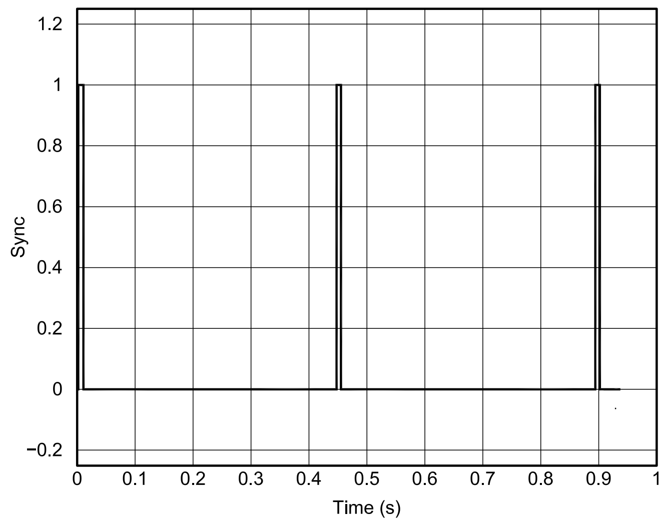


Fig 7—Output of the synchronization detector (M1 signal without noise).

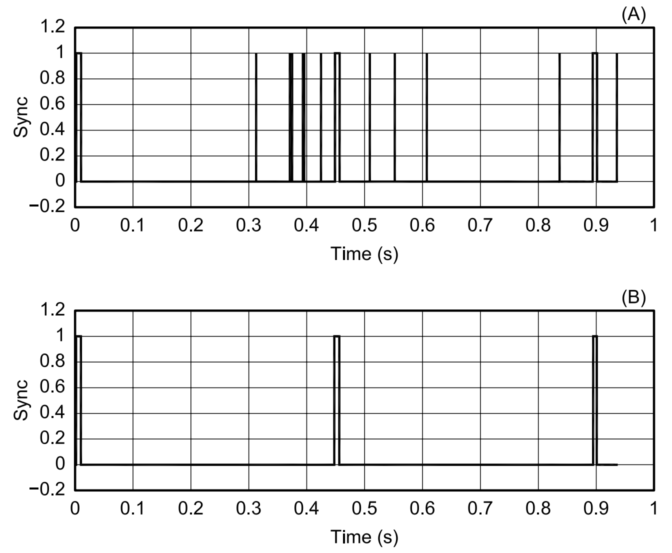


Fig 8—Output of the synchronization detector (M1 signal with noise, A, versus without noise, B).

is a closer look around the center frequency.

We are also highly interested in the behavior of this filter to a 1200-Hz sine-wave burst. We have therefore simulated a synchronization-like signal made of: 5 ms without signal, 5 ms with a 1200 Hz, unity-amplitude sine wave, and 10 ms without signal.

The choice of a 5-ms, 1200-Hz sine wave signal is realistic with respect to both the original standard (black and white) and the M1 (color) SSTV modes.[9] Both the input signal (Fig 4, upper curve) and the output of the IIR filter (lower curve) are plotted.

The 200-Hz bandwidth chosen during the design is the lower bound one can use. One can see in Fig 4 that the output signal reaches unity amplitude just before it starts decaying. This bandwidth is a trade-off between noise rejection and response time. It proved efficient during the many experiments carried out on-air with real SSTV M1 signals.

This filtered signal is detected thanks to an absolute-value function (the diode in Fig 2) along with the original signal. The remaining blocks are a poor-man's power-spectrum estimator. The 50-tap digital low-pass filter (a finite-impulse-response or FIR filter) is a moving-average type. It acts as a low-pass filter with a zero in its frequency response at $f_s / 50 = 882$ Hz.

Here's another way of explaining the behavior of this low-pass filter: It outputs the correlation of the detected signal with a rectangular window. This rectangular window, lasting $50 / f_s = 1.1$ ms, is related to the desired time resolution of the synchronization detector. The correlation gives an indication of how much two signals look alike, along with the time of this likelihood measurement. A simulation result is displayed in Fig 5. The dashed line is the input signal, made of 5 ms without signal, 1.1 ms with a unity-amplitude signal and 3.9 ms without signal. It is a rectangular window of 50 samples. The bold continuous line in Fig 5 is the output of the filter. The maximum value (1 on Fig 5) is reached when the likelihood is at its maximum.

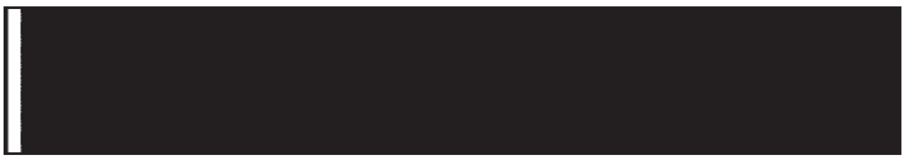


Fig 9—The synchronization signal—a 2D approach (no noise).

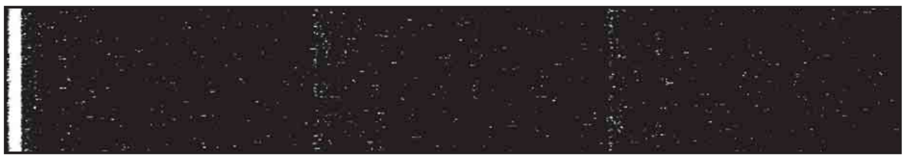


Fig 10—The synchronization signal—a 2D approach (with noise).

Now, look back to Fig 2: This FIR filter is applied to both raw and band-pass filtered detected signals. The detected value (the output of the diode in the figure) can be seen as a voltage. So squaring this value (the x^2 block in Fig 2) leads to a power, noted as P_{sync}. The raw power, P_{raw}, is normalized thanks to the gain K with respect to its assumed bandwidth (3000 Hz can be used as an upper bound for the average communication receiver). Now, both power channels can be compared against one another. The meaning of this comparison can be related to the following question: "Is there more power in the 1100 to 1300-Hz range than in the rest of the band?"

A simple threshold is set to decide whether the SSTV input signal is a synchronization signal or another type—video, for instance, or some QRM. During all the experiments, this threshold has been set to two: The incoming signal is said to be a synchronization signal when $P_{sync} > 2 \times P_{raw}$. Now, let us see how our synchronization detector performs.

Simulation Results: Why do we mention simulation results? A real, noisy, faded SSTV signal coming from the air would be nice; but it would be very difficult to analyze. Unless we have access to the original, clean signal, it is difficult to qualify the demodulator. Anyway, we will show some results with simulated data.

Simulation Signals: The original SSTV signal (the reference) has been created with a stand-alone program we have developed for this purpose. It reads a bitmap (see Fig 6) 24-bit color file and creates an M1-compliant monaural sound file (.wav). It also has the nice feature of adding noise to the pure M1 signal.

The noisy signal includes Gaussian noise. Its standard deviation from the mean is 1, and its bias (or mean value) is 0. This noise is then filtered by a low-pass filter, an 8th-order Butterworth IIR with a cutoff frequency of 2500 Hz. The value of 2500 Hz is a realistic one for SSTV signals.

This sound file is either used by our SSTV software or transferred to a CDROM for test purposes. This brand new audio CD, playing on a CD player, becomes the source of the SSTV signal that is digitized by the sound card. This solution has been extensively used when comparing the performance of our SSTV software against other programs.

Fig 7 plots the output of the synchronization detector for a clean SSTV M1 signal (without Gaussian noise). The horizontal-synchronization pulses are perfectly estimated. The period between two pulses, measured on Fig 7, is about 0.446 s. The theoretical value is 4862 + 572 + (572) (3) + (256) (572) (3) = 446,446 µs.

We have already mentioned that we were interested in the behavior of this synchronization detector when the S/N is low. We have therefore created a noisy M1 signal (S/N is almost 0 dB). The output of the same software is displayed on the upper curve of Fig 8 for this noisy signal. The reference signal is plotted on the lower curve to make comparison easier. At first sight, it is very difficult to accurately detect horizontal line synchronization out of this 1-D signal.

This first conclusion has led us to think about another representation of the *same* signal. The 1-D signal is indeed the output of the synchronization detector, so the 1-D representation may first come to mind. Yet this 1-D signal actually results from a 2-D signal; namely, the original picture. Let us plot the same output signal from the synchronization detector as a picture, thus as a 2-D signal. We use the following convention:

- A 0 in the 1-D signal (no synchronization signal) is displayed as a black pixel.
- A 1 in the 1-D signal (a synchronization signal) is displayed as a white pixel.
- The width of the resulting picture is X_{M1} = 1561 pixels.
- The height of the resulting picture is limited by the total amount of available synchronization samples.

The value X_{M1} = 1561 comes from the length of one M1 line, 446,446 µs, and the duration of one sample of 286 µs: 1561 = 446,446 / 286. The latter value of 286 µs has been chosen to get an integral number of samples for the synchronization signal (4862 / 286 = 17 samples) and for all other parts of the M1 line.

The resulting picture is displayed in Fig 9 for the M1 signal without noise (the same one used for Fig 7). A human being easily locates the synchronization signal on the left.

The case of the noisy M1 signal leads to the picture displayed in Fig 10. Even on this picture, coming from a very noisy M1 SSTV signal, the reader easily detects the synchronization signal.

The question is, "How can an algorithm accurately detect this vertical line?" Assuming it is possible, we would have built a robust synchronization detector based upon the whole synchronization signal.

From a 2-D Picture to a Synchronization Detector

The very first approach one can test is a least-mean-square (LMS) approach. The problem can be stated this way: *Knowing the 2-D coordinates of some points, the algorithm has to estimate the abscissa of the vertical line while minimizing some criteria.*

It is a simple curve-fitting approach[10] with a line described by Eq 3:

$$x = b \qquad \text{(Eq 3)}$$

The parameter b can be estimated as the mean value of all the abscissas of the white pixels. It works perfectly when the M1 signal is clean (no noise),

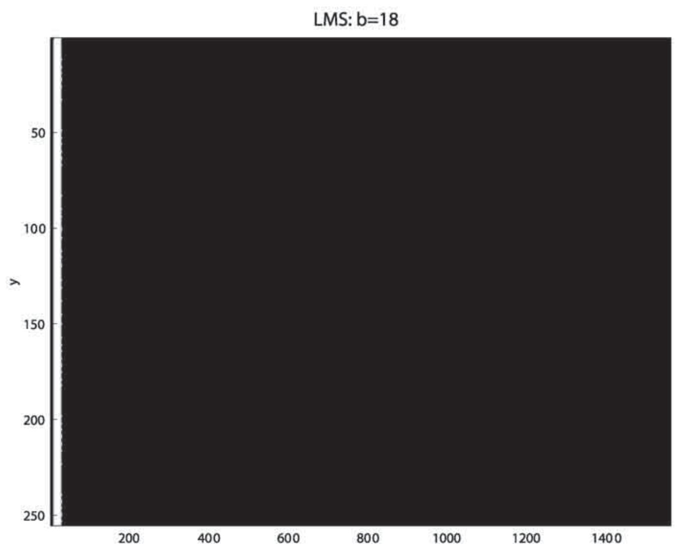


Fig 11—Synchronization signal estimated thanks to LMS (no noise, 250 lines).

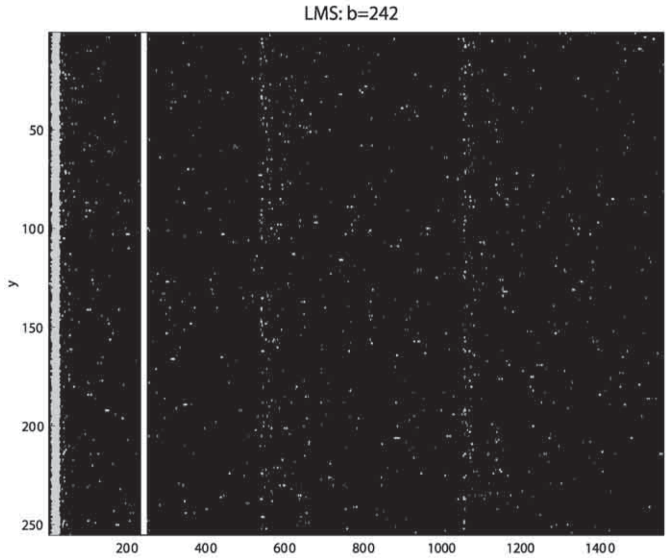


Fig 12—Synchronization signal estimated thanks to LMS (with noise, 250

and thus when the synchronization signal is perfectly recovered. The result from a LMS approach is displayed in Fig 11. The grey pixels are the output of the synchronization detector, and the white vertical line is the estimated one, thanks to the LMS. The estimated *b* is 18, which is correct.

This very same method totally fails when the signal is noisy (see Fig 12). The reader may wonder why such a popular method fails. The answer is simple: The parameter to be estimated, a constant, is corrupted by a noise with a nonzero mean. Here, the noise is produced by the synchronization detector. This is why we have used another approach that is robust against this kind of noise and is presented in the next section.

Using the Linear Hough Transform

The method we have used is robust against the kind of noise generated by the synchronization detector. It relies upon the linear Hough transform. We will first describe it using a step-by-step

```
// Input:
// pixel: an array of size (xMax+1,yMax+1)
// Output:
// bLine: the b parameter of the synchronization vertical line.
// maximum: the number of occurence of this b value.
//
// first step
for y=0 to yMax
   for x=0 to xMax
      if pixel(x,y) is white
         // estimate b parameter
         b=x
         // use the accumulator and take this new value
         // into account
         accumulator [b] =accumulator [b]+1
      end if
   end for
end for
// second step
maximum=0
for b=0 to xMax
   if (accumulator[b]>maximum)
     maximum=accumulator[b]
     bLine=b
   end if
end for
```

Fig 13—The Hough algorithm for a vertical line.

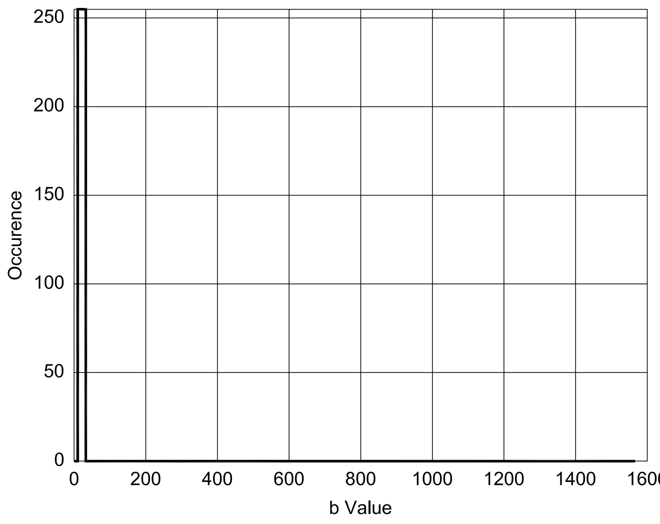


Fig 14—The accumulator corresponding to Fig 9 (M1 signal without noise) showing occurrences of *b* values (b_{line} = 9).

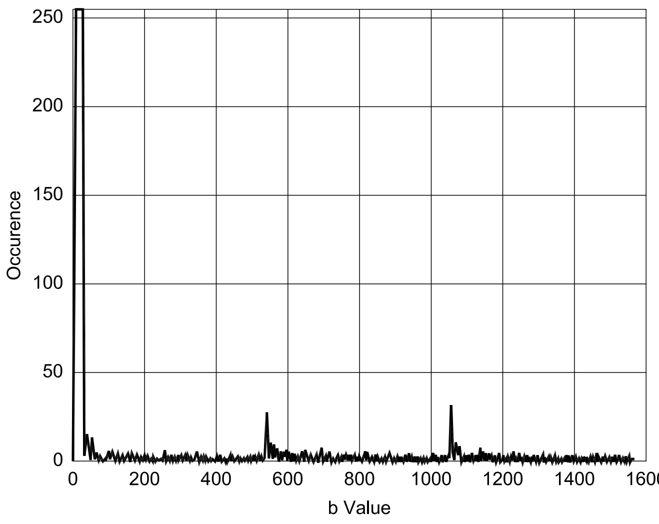


Fig 15—The accumulator corresponding to Fig 10 (M1 signal with noise) showing occurrences of *b* values (b_{line} = 11).

approach in a very simple situation. We will then introduce the linear Hough transform as presented in Reference 11. We will also illustrate it using another example in a step-by-step approach.

Estimating the b Parameter of a Vertical Line

This simple case deals with the same vertical line described by Eq 3. Let's have a closer look at either Fig 9 or 10. You can easily *see* where the vertical line is (on the left of both pictures). We could also write that *most of the pixels on the left belong to the* same *vertical line, whereas the remaining pixels do not belong to the same vertical line*. This idea can be translated into a two-step algorithm.

First step: For each white pixel, compute the line parameter *b* of the vertical line to which it belongs.

Second step: Among all the resulting *b* parameters, find the one with the highest occurrence.

The algorithm in Fig 13 performs these two steps. We have introduced an accumulator, which is an array. It is used to compute and store the number of occurrences of the *b* parameters.

This algorithm has been used on the very same data from the synchronization detector. Fig 14 plots the accumulator corresponding to the clean M1 signal (from Fig 9). Even in the presence of noise, the algorithm is still able to recover the synchronization. Fig 15 plots the accumulator corresponding to the noisy M1 signal of Fig 10. The *b* parameter is correctly estimated with this method: The error is 2/1561.

The Linear Hough Transform

This section could have been entitled "Estimating the Parameters of a Straight Line." We have seen in the previous section how to estimate the *b* parameter of a vertical line. It was a simple case of the linear Hough transform. We are now going to introduce the general linear Hough transform; we will illustrate its use through a step-by-step computation.

No More Slanted Pictures!

We have already seen that the results were convincing with this simple estimator, even in the presence of noise. The reader may wonder why we proceed further.

Have you ever seen a beautiful SSTV picture spoiled because it is slanted? How many times have you thrown away a rare DX picture just because of a badly calibrated time base?

A badly calibrated time base—at either the transmitter or receiver side—results in a slanted picture (see Fig 16, for instance). Much modern SSTV software allows the user to finely tune its time base to get perfect pictures. Actually, the received pictures are perfect as long as the transmitter time base is correctly calibrated.

Could we imagine an automatic system that relies only on the previously described "synch" detector? Such a system would receive a SSTV signal and output a vertical picture, even if the time base were not calibrated. We can design such a system. From a practical point of view, one can improve our SSTV synch detector—and thus the SSTV software—using the linear Hough transform.[11]

The Hough transform is named after its inventor (see the US patent).[12] It is widely used in the field of image processing.[13] We will see how to use for SSTV synch detection.

The slanted picture of Fig 16 is linked with the 2-D slanted synch signal (see Fig 17). The linear Hough transform allows estimation of the parameters describing a straight line. Eq 4, used to describe this line, is not the usual $y = ax + b$, but:

$$= -x \sin q + y \cos q \qquad \text{(Eq 4)}$$

Eq 4 is better for our application, as it can describe any straight line: vertical, horizontal or skew. The usual equation $y = ax + b$ fails to describe a vertical line (as long as a is finite). Moreover, Eq 4 closely relates the slope of the line and the parameter q, which is useful for the Hough transform.

The linear Hough transform relies upon a 2-D accumulator or array. The two dimensions of the accumulator are linked with the two parameters (d, q) of the line. The algorithm, also a two-step, is described in Fig 18.

A Step-by-Step Computation

An example illustrates the algorithm of Fig 18. Consider a very simple picture. Its size is 5×5. The round dots are the pixels of interest (see Fig 19).

During this computation, we are only looking for horizontal, vertical and diagonal lines. Thus, the corresponding values of q are 0°, 45°, 90° and 135°.

Step One

Let us start with the pixel located at coordinates (–1, –2). According to the algorithm, for each possible value of q we have to compute the corresponding d using Eq 4. The results are tabulated

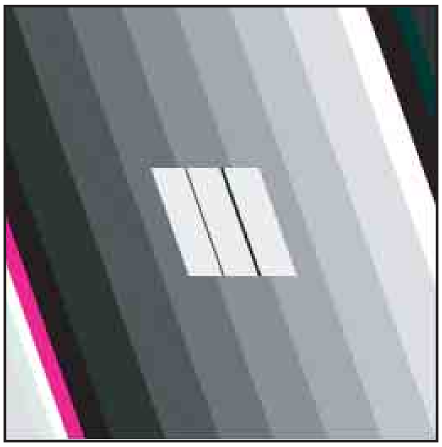


Fig 16—A slanted SSTV picture.

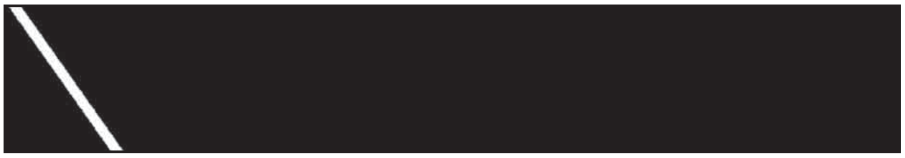


Fig 17—A slanted synchronization signal.

```
// first step
for each pixel (x,y) of interest
          for each possible q
             computes d = -x sin(q) + y cos(q)
             accumulate the set (d, q)
          end for
end for
// second step
find in the accumulator the set (d, q)
with the highest occurrence
```
Fig 18—The Hough algorithm for a straight line.

in Fig 20 (step 1). We must point out that *d,* computed using Eq 4, is a real number. It is represented by a floating-point number (see the third column). As this number is used as an index of an array (the accumulator), this floating-point number is rounded to the nearest integer (see the fourth column). The slight difference between the exact value of *d* and the value stored in the accumulator and used by the algorithm will cause an error in the estimated parameters. This error will be obvious when looking at the resulting estimated line. Nevertheless, it is not an important error, and it has no impact on our application.

The meaning of the first line of Fig 20 is as follows: *the pixel (–1, –2) belongs to the line whose equation is, according to Eq 4,* $-2 = -x \sin(0) + y \cos(0)$. Alternatively, one can write that *the pixel (–1, –2) belongs to the line whose parameters (d, q) are (–2, 0)*. Now we could add one to the accumulator value located at (–2, 0). The accumulator would now reflect that there is one and only one line whose parameters are (–2, 0).

The resulting *d* values, along with their corresponding *q* values, are used to fill the accumulator. At the beginning, the accumulator is cleared—that is, filled with zeros. Using Fig 20, it is easy to build the accumulator at step 1:

$q = 0°$ leads to $d = -2$; $\text{ACC}[d, q] = \text{ACC}[d, q] + 1$

$q = 45°$ leads to $d = -1$; $\text{ACC}[d, q] = \text{ACC}[d, q] + 1$

$q = 90°$ leads to $d = 1$; $\text{ACC}[d, q] = \text{ACC}[d, q] + 1$

$q = 135°$ leads to $d = 2$; $\text{ACC}[d, q] = \text{ACC}[d, q] + 1$

The bold numbers in the accumulator represent the values different from the previous step. The picture on the left stands for the visual counterpart of the calculations. All the possible lines, corresponding to the tabulated values of *q*, are drawn on this figure.

Step Two

The same method is applied to the pixel (0, –1). The computed *d* for all the *q* values are tabulated in Fig 20.

Steps Three to Eight

The same method is applied to remaining pixels. The computed *d* for all the *q* values are tabulated in Figs 20-22.

The final values in the accumulator at (*d*, *q*) are the number of points belonging to the line (*d*, *q*). In our example, the highest value, 4 (see Fig 22) belongs to the line described by the set ($d = -1$; $q = 45°$). The line described by these parameters is plotted as a thick line in Fig 23, whereas the line calculated by LMS is the dashed line. Although not perfect (because of a rounding effect), the Hough estimation is better than the LMS.

The Case of the Slanted Picture

How to Estimate the Sampling Frequency of Your Sound Card

Unfortunately, this frequency is not as accurate as one might expect: the 44,100 Hz may be biased. A 40-Hz bias is common. This problem is well known in the SSTV community.

The following equations are based upon three hypotheses:

1. At time $t = 0$, the software is demodulating the very first video sample, located on the upper left corner (0 on Fig 24);
2. The M1 signal is perfect—that is, compliant with the timings defined by G3OQD;
3. The assumed sampling frequency, noted as f_S, is used to demodulate the M1 SSTV picture. This picture is available for the sampling frequency estimation (see Fig 24 for a model of such a slanted picture).

The coordinates of video sample B are (x_B, y_B); those of A are (x_A, y_A. Notice that one has $y_A = y_B$.

One video sample lasts t_1, that is 572 µs. During one second, the sound card digitizes f_S samples. Thus, one M1 video sample, lasting t_1, requires N_{M1} sound card samples, with N_{M1} defined by Eq 5:

(Eq 5)

Please note that N_{M1} is *not* an integer. The video sample B (see Fig 24) is received at time t_B. The total number of sound card samples corresponding to the video sample B is: $i_B \times N_{M1}$ with i_B the index of the video sample B, defined by $i_B = x_B + y_B \times X_{M1}$. Where X_{M1} is the number of video samples per M1 line. The video sample B is received at time t_B defined by Eq 6.

(Eq 6)

Where f_{unknown} is the unknown frequency of the sound card (if $B \neq A$, then $f_{\text{unknown}} \neq f_S$). From Eqs 5 and 6, we have:

$$t_B = i_B \times t_1 \times f_S \frac{1}{f_{\text{unknown}}} \quad \text{(Eq 7)}$$

As the received M1 SSTV signal is based upon the exact timings, we can infer that the video sample B should be located in A (see Fig 24).

The video sample A, which index $i_A = x_A + y_A \times X_{M1}$, occurs at time t_A:

$$t_A = i_A \times N_{M1} \frac{1}{f_S} \quad \text{(Eq 8)}$$
$$= i_A \times t_{M1}$$

As we know that the video sample B should be located in A, we can write Eq 9:

$$t_A = t_B \quad \text{(Eq 9)}$$

Substituting Eqs 6 and 8 into Eq 9 leads to:

(Eq 10)

Finally, we can compute the unknown frequency f_{unknown} with Eq 11:

$$f_{\text{unknown}} = \frac{i_B}{i_A} f_S \quad \text{(Eq 11)}$$

Such a simple equation easily allows us to accurately estimate the sampling frequency of the sound card. In fact, the accuracy of our approach is only limited by the quality of the slope estimation. That explains why we needed a very-high-performance synchronization detector.

Experiments carried out with real noisy SSTV signals allowed us to estimate the sampling frequency of our sound card. The overall quality of the estimator is easily checked: The picture demodulated with the estimated f_{unknown} is *perfectly* vertical.

Of course, a similar approach could be used when the sound-card sampling frequency is known, but the picture is nevertheless slanted. Such a situation occurs when demodulating a signal replayed on a CD player, whose frequency is not as accurate as one might think. The slope of the picture leads to an *ideal* value of f_S. When demodulating the same signal with this newly computed f_S, the formerly slanted picture now appears perfectly vertical. The very-high-performance

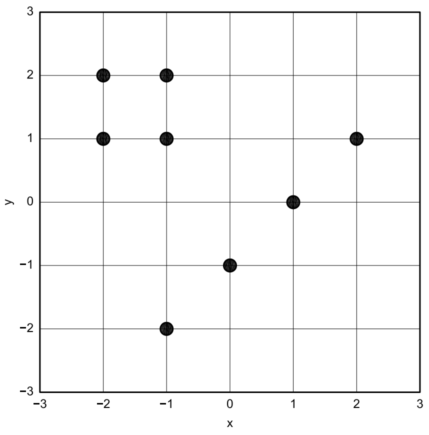
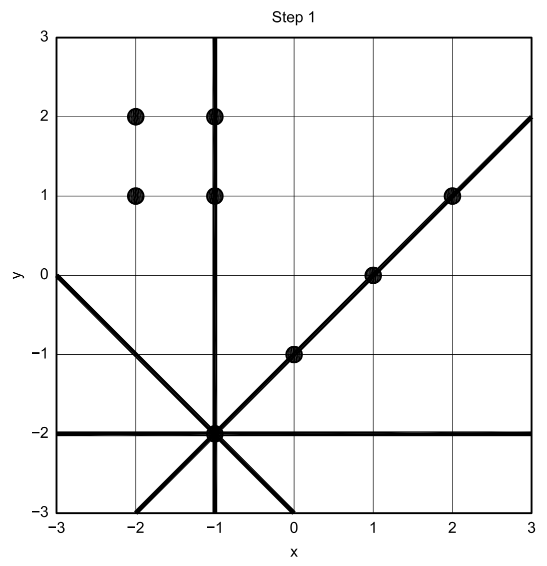


Fig 19—A simple 5×5 picture.

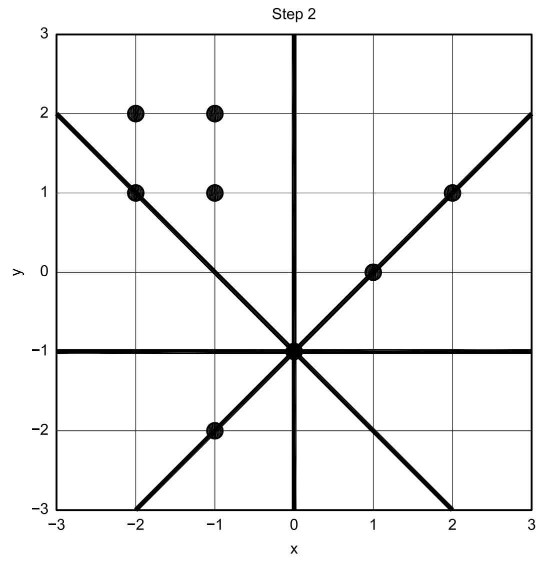


Step 1: Computing Parameter *d* for Pixel (–1, –2)

q	–x sin *q* + y cos *q*	*d*	*round(d)*
0°	1 × 0.00 –2 × 1.00	–2.00	–2
45°	1 × 0.85 –2 × 0.53	–0.71	–1
90°	1 × 0.89 –2 × –0.45	1.00	1
135°	1 × 0.09 –2 × –1.00	2.12	2

The Accumulator After Step 1

ACC	*0°*	*45°*	*90°*	*135°*
d = –2	***1***	0	0	0
d = –1	0	***1***	0	0
d = 0	0	0	0	0
d = 1	0	0	***1***	0
d = 2	0	0	0	***1***
d = 3	0	0	0	0

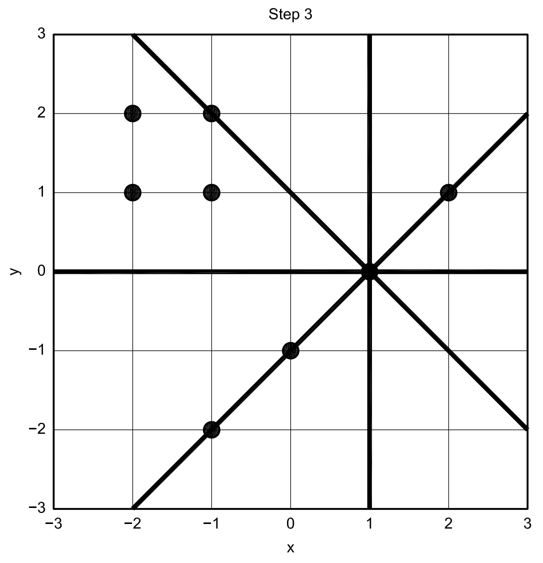


Step 2: Computing Parameter *d* for Pixel (0, –1)

q	–x sin *q* + y cos *q*	*d*	*round(d)*
0°	0 × 0.00 –1 × 1.00	–1.00	–1
45°	0 × 0.85 –1 × 0.53	–0.71	–1
90°	0 × 0.89 –1 × –0.45	–0.00	0
135°	0 × 0.09 –1 × –1.00	0.71	1

The Accumulator After Step 2

ACC	*0°*	*45°*	*90°*	*135°*
d = –2	1	0	0	0
d = –1	***1***	***2***	0	0
d = 0	0	0	***1***	0
d = 1	0	0	1	***1***
d = 2	0	0	0	1
d = 3	0	0	0	0



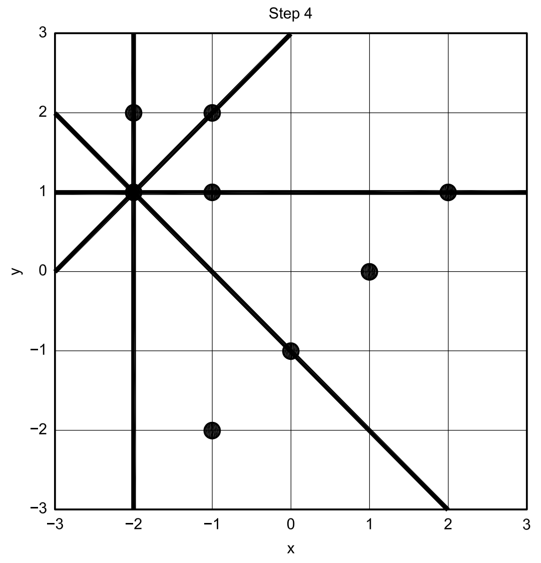
Step 3: Computing Parameter *d* for Pixel (1,0)

q	–x sin *q* + y cos *q*	*d*	*round(d)*
0°	–1 × 0.00 + 0 × 1.00	0.00	0
45°	–1 × 0.85 + 0 × 0.53	–0.71	–1
90°	–1 × 0.89 + 0 × –0.45	–1.00	–1
135°	–1 × 0.09 + 0 × –1.00	–0.71	–1

The Accumulator After Step 3

ACC	*0°*	*45°*	*90°*	*135°*
d = –2	1	0	0	0
d = –1	1	***3***	***1***	***1***
d = 0	***1***	0	1	0
d = 1	0	0	1	1
d = 2	0	0	0	1
d = 3	0	0	0	0

Fig 20—Hough transform—Steps 1 to 3.

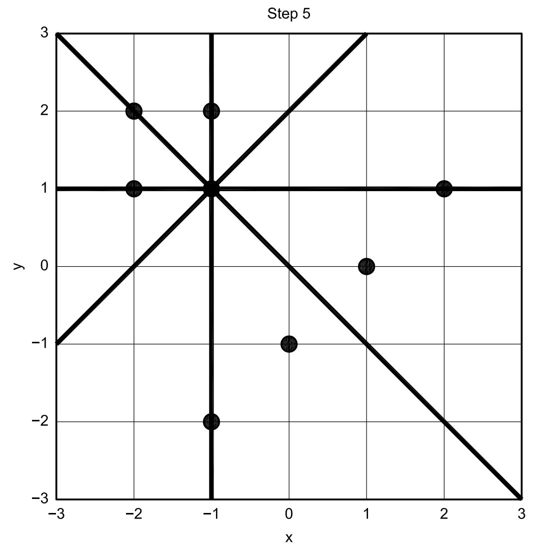


Step 4: Computing Parameter *d* for Pixel (–2, 1)

q	–x sin *q* + y cos *q*	*d*	*round(d)*
0°	2 × 0.00 + 1 × 1.00	1.00	1
45°	2 × 0.85 + 1 × 0.53	2.12	2
90°	2 × 0.89 + 1 × –0.45	2.00	2
135°	2 × 0.09 + 1 × –1.00	0.71	1

The Accumulator After Step 4

ACC	*0°*	*45°*	*90°*	*135°*
d = –2	1	0	0	0
d = –1	1	3	1	1
d = 0	1	0	1	0
d = 1	***1***	0	1	***2***
d = 2	0	***1***	***1***	1
d = 3	0	0	0	0

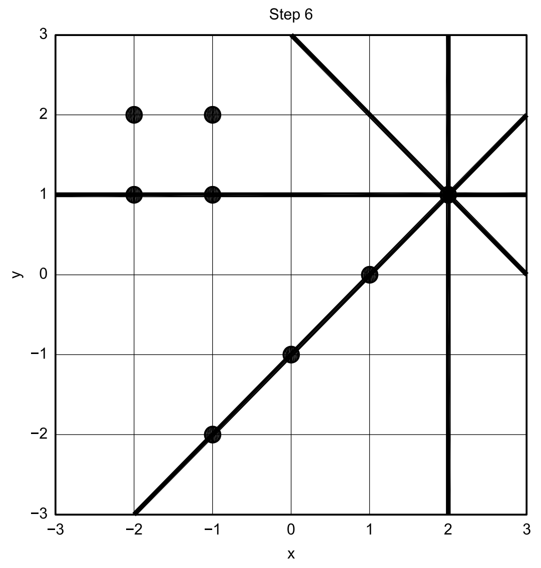


Step 5: Computing Parameter *d* for Pixel (–1, 1)

q	–x sin *q* + y cos *q*	*d*	*round(d)*
0°	1 × 0.00 + 1 × 1.00	1.00	1
45°	1 × 0.85 + 1 × 0.53	1.41	1
90°	1 × 0.89 + 1 × –0.45	1.00	1
135°	1 × 0.09 + 1 × –1.00	0.00	0

The Accumulator After Step 5

ACC	*0°*	*45°*	*90°*	*135°*
d = –2	1	0	0	0
d = –1	1	3	1	1
d = 0	1	0	1	***1***
d = 1	***2***	***1***	***2***	2
d = 2	0	1	1	1
d = 3	0	0	0	0



Step 6: Computing Parameter *d* for Pixel (2, 1)

q	–x sin *q* + y cos *q*	*d*	*round(d)*
0°	–2 × 0.00 + 1 × 1.00	1.00	1
45°	–2 × 0.85 + 1 × 0.53	–0.71	–1
90°	–2 × 0.89 + 1 × –0.45	–2.00	–2
135°	–2 × 0.09 + 1 × –1.00	–2.12	–2

The Accumulator After Step 6

ACC	*0°*	*45°*	*90°*	*135°*
d=–2	1	0	***1***	***1***
d=–1	1	***4***	1	1
d=0	1	0	1	1
d=1	***3***	1	2	2
d=2	0	1	1	1
d=3	0	0	0	0

Fig 21—Hough transform—Steps 4 to 6.

demodulator is at the heart of such magic.

No more slanted pictures, we wrote. Now it is done! From a practical point of view, however, it is not an excuse for transmitting non-M1-compliant signals!

The Video Demodulator

A lot of work and time has been devoted to the synchronization detector described in the first part of this article. SSTV receiving software must also demodulate the video signal. Lastly, it must display it. We will now focus on video demodulation.

From Frequency to Luminance

Can we improve frequency estimation? The reason behind this question is straightforward: Each video sample can be recovered using Eq 12, which is based upon Eq 1:

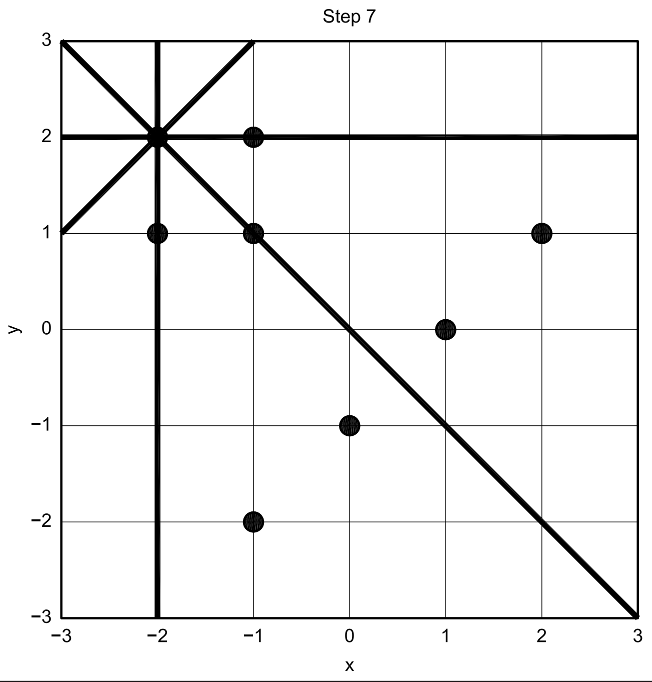
$$lum = \frac{f_1 - f_{\text{black}}}{f_{\text{white}} - f_{\text{black}}} max_{\text{lum}} \qquad \text{(Eq 12)}$$

Eq 12 will output the corresponding luminance, which will be stored in an array for further processing, such as for full-color display.

Unfortunately, such a beautiful equation leads to another question: How can we estimate this unknown frequency f_1? Many solutions have been proposed and used during the past decades. We have already summed up three important strategies at the beginning of this article:

- Analog filters can convert a frequency to a voltage. This voltage is used to drive either an SSTV monitor (see Note 2) or digitized and processed on a computer (see Note 4).
- The signal is crudely digitized thanks to a two-level (or 1-bit) A/D converter. The frequency is then estimated using, for instance, a period estimation (the method used in the Pasokon system).[14]
- The signal is digitized thanks to a decent A/D converter. The digital signal is then processed to estimate the frequency.

As we had already designed a robust synchronization estimator, the video demodulator performance had to

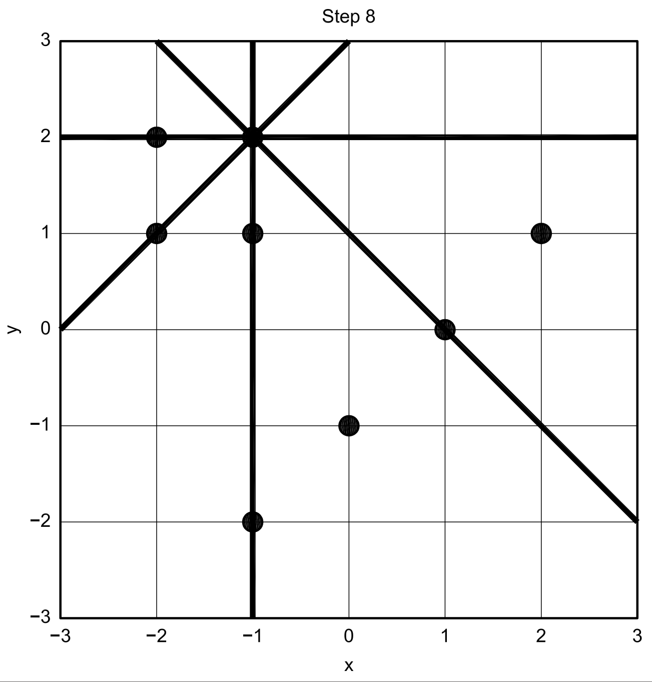


Step 7: Computing Parameter *d* for Pixel (–2, 2)

q	*–x sin q + y cos q*	*d*	*round(d)*
0°	2× 0.00+2× 1.00	2.00	2
45°	2× 0.85+2× 0.53	2.83	3
90°	2× 0.89+2× –0.45	2.00	2
135°	2× 0.09+2× –1.00	0.00	0

The Accumulator After Step 7

ACC	*0°*	*45°*	*90°*	*135°*
d = –2	1	0	1	1
d = –1	1	4	1	1
d = 0	1	0	1	***2***
d = 1	3	1	2	2
d = 2	***1***	1	***2***	1
d = 3	0	***1***	0	0



Step 8: Computing Parameter *d* for Pixel (–1, 2)

q	*–x sin q + y cos q*	*d*	*round(d)*
0°	1 × 0.00 + 2 × 1.00	2.00	2
45°	1 × 0.85 + 2 × 0.53	2.12	2
90°	1 × 0.89 + 2 × –0.45	1.00	1
135°	1 × 0.09 + 2 × –1.00	–0.71	–1

The Accumulator After Step 8

ACC	*0°*	*45°*	*90°*	*135°*
d = –2	1	0	1	1
d = –1	1	4	1	***2***
d = 0	1	0	1	2
d = 1	3	1	***3***	2
d = 2	***2***	***2***	2	1
d = 3	0	1	0	0

Fig 22—Hough transform—Steps 7 and 8.

match that high-quality level.

There are many well-known methods to estimate the frequency of a signal. One can use a period estimator, a phase-locked loop or a fast Fourier transform (FFT). There are many other methods, such as multiple-window spectrum estimation[15] and the wavelet transform. We have not thoroughly investigated the use of the last two approaches in the context of SSTV; it might be something worth looking at.

It is obvious that for high S/N many methods provide good results—and particularly, a small bias on the estimated frequency. When S/N becomes low, things are not so clear. Studies have been conducted by several authors to evaluate the behavior of estimators with a noisy signal.[16, 17] These papers show that the period estimator is among the worst methods, while the FFT is the least-biased approach.

The discrete Fourier transform (DFT), often implemented as the FFT, is an excellent frequency estimator:[18, 19, 20]

> "When the data consist of uniformly sampled time domain data containing some type of harmonic oscillations, the discrete Fourier transform is almost universally used as the frequency estimation technique. This is done for a number of reasons, but primarily because the technique is fast and experience has shown that the frequency estimates obtained from it are often very good."[21]

Even in the presence of noise, one can extract meaningful information from the power spectrum of the signal. Our frequency estimator relies upon both the discrete Fourier transform and the FFT (for performance issues only).

When the S/N is high, the user expects a high-quality picture. When the S/N is low, or when it suddenly decreases (due to fading, for instance), such quality cannot be maintained. A common thought is that the quantity of information transmitted per second decreases along with the S/N. This is perfectly true (see Shanon's famous paper,[22] for a mathematical explanation). Many everyday-life examples follow this rule: a CW signal is easier to recover using a narrow filter (200 Hz, for instance) than when using an SSB filter (3000 Hz, for instance). The signal can be recovered with the CW filter, whereas it could not with the standard SSB filter.

Keeping this idea in mind, we may wonder what a SSTV demodulator should do? Our answers are summarized below:

• It should provide a high-resolution picture when the S/N is high.

• It should gracefully degrade when the S/N decreases. The resulting video resolution would then be lower than in the theoretical case, as set by G3OQD.

• It should decide on its own which quality is the best during the SSTV demodulation. It should always provide the best trade-off between noise immunity and picture resolution.

Can a software demodulator do that? Fortunately, the answer is yes!

Remember that the frequency resolution of the DFT is closely related to the number of samples used to compute the power spectrum. When more samples are used the frequency estimation becomes more accurate.

The only requirement is to design a *magic box* that chooses the best length for the time series the DFT will process. Such a box must estimate the S/N of the SSTV signal. It then converts this S/N to the appropriate length thanks to a function $f(S/N)$. This function can be stored as a predetermined look-up table. Fig 25 shows the idea behind this adaptive scheme.

The S/N Estimator

This part has been very difficult to design. Some methods make use of silence to estimate the power of noise.[23] Our magic box cannot rely upon this approach because a typical SSTV signal lasts about two minutes and is continuous. We have therefore designed a S/N estimator that runs in real-time; that is, during actual SSTV reception.

A simplified SSTV power spectrum is depicted in Fig 26. It comprises the video signal plus some noise (the synchronization signal is not considered here). The purpose of a S/N estimator is to estimate: (1) the power of the SSTV-only signal for a given bandwidth; and (2) the noise power for a given bandwidth.

Let us split the spectrum into three bands, according to Fig 27:

1. A video-plus-noise band, ranging from 1500-2300 Hz.
2. A low-frequency noise-only band, ranging from 300-1100 Hz. The lower bound, 300 Hz, is realistic with an SSB receiver. The upper bound, 1100 Hz, has been chosen to avoid the synchronization signal (centered on 1200 Hz).
3. A high-frequency noise-only band, ranging from 2500-2700 Hz.

The total power in each band is computed thanks to a 2048-sample FFT (f_s = 44,100 Hz) after windowing the incoming signal with a Hanning window (see Reference 19,

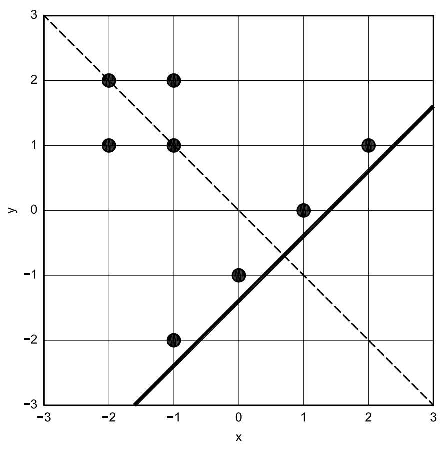


Fig 23—Hough versus LMS approximations.

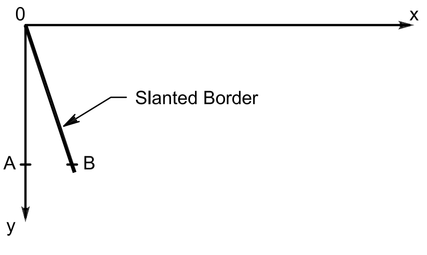


Fig 24—Model of a slanted picture.

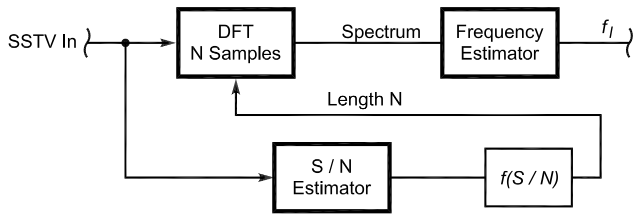


Fig 25—The adaptive scheme for frequency estimation.

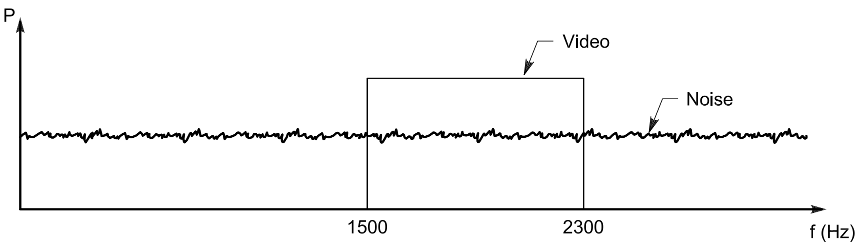


Fig 26—A simplified spectrum of a noisy SSTV signal.

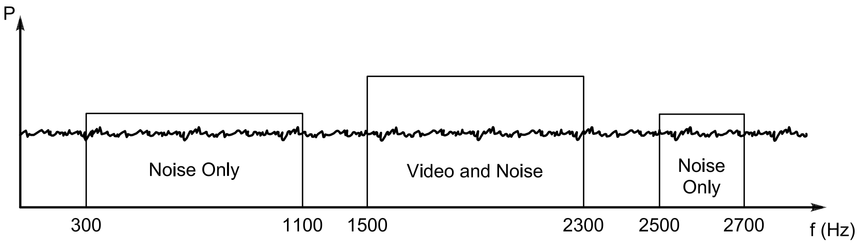


Fig 27—Splitting the spectrum into three bands.

p 20 for some window functions). We then introduce two new variables, namely $P_{\text{video\_noise}}$ and $P_{\text{noise\_only}}$.

- $P_{\text{video\_noise}}$ is the total signal power in the video band (1500-2300 Hz).
- $P_{\text{noise\_only}}$ is the total signal power in the lower band (300-1100 Hz) plus the total signal power in the higher band (2500-2700 Hz).

Under the following assumptions:

- The noise power is constant across all frequencies. If not, one can measure the global information filter response of the receiver and make up for it.
- The SSTV video signal lies from 1500 to 2300-Hz.

One can write:

$$P_{\text{video\_noise}} = P_{\text{signal}} + P_{\text{noise}} \frac{BW(P_{\text{video\_}}}{BW(recei} \qquad \text{(Eq 13)}$$

$$P_{\text{noise\_only}} = P_{\text{noise}} \frac{BW(P_{\text{noise\_only}})}{BW(receiver)} \qquad \text{(Eq 14)}$$

where:

$BW(receiver)$ = receiver bandwidth (2700 – 300 = 2400 Hz)

$BW(P_{\text{video\_noise}})$ = bandwidth of the video signal (800 Hz for SSTV)

$BW(P_{\text{noise\_only}})$ = total bandwidth used for noise-power estimation. According to Fig 27, it is (1100 – 300) + (2700 – 2500) = 1000 Hz

P_{signal} = SSTV video signal power for a video bandwidth $BW(P_{\text{video\_noise}})$

P_{noise} = noise power for the receiver bandwidth $BW(receiver)$.

From Eq 14, one can write:

$$P_{\text{noise}} = P_{\text{noise\_only}} \frac{BW(receiver)}{BW(P_{\text{noise\_only}})} \qquad \text{(Eq 15)}$$

From Eqs 13 and 15, we get:

$$P_{\text{noise}} = P_{\text{noise\_only}} \frac{BW(receiver)}{BW(P_{\text{noise\_only}})}$$

$$P_{\text{signal}} = P_{\text{video\_noise}} - P_{\text{noise}} \frac{BW(P_{\text{video\_}}}{BW(recei} \qquad \text{(Eq 16)}$$

And expanding Eq 16, one has:

$$P_{\text{noise}} = P_{\text{noise\_only}} \frac{BW(receiver)}{BW(P_{\text{noise\_only}})}$$

$$P_{\text{signal}} = P_{\text{video\_noise}} - P_{\text{noise\_only}} \frac{BW(P_{\text{vi}}}{BW(P_{\text{n}}} \qquad \text{(Eq 17)}$$

The S/N is then computed by:

$$S/N = 10\log\left(\frac{P_{\text{signal}}}{P_{\text{noise}}}\right) \qquad \text{(Eq 18)}$$

Notice that we can bound the minimum ratio $P_{\text{signal}} / P_{\text{noise}}$ to 0.01 (–20 dB), a value far below realistic conditions for SSTV reception. This value will be used in the sequel.

Results

A Martin M1 signal is generated. Gaussian noise is then created and filtered by the same low-pass filter used for synchronization detector tests, an 8th-order Butterworth IIR with a cutoff frequency of 2500 Hz. It simulates the filters of the receiver.

Nine different amounts of v_{noise} were added to the pure M1 signal. We have arbitrarily chosen: 3, 2, 1.5, 1, 1/1.5, 1/2, 1/3, 1/6 and 1/10 times the original amplitude of the noise. Please notice that such a simulation only focuses on noisy SSTV signals and that it does not consider QRM. The estimated S/N is plotted for the nine experiments, along with the input noise power, in Fig 28.

Here are a few comments about that figure. The reference noise power, denoted as "o = input," is computed by 10 log$((v_{\text{noise}})^{-2})$. The amplitude of the signal is one. Notice there is an offset between the input noise power and the estimated S/N. That is caused by the way the noise power is computed; that is, without appropriate scaling. It is not important for our application, as a gain between these values is converted into an offset in the logarithmic scale.

We already know the estimator is biased. Another interesting simulation gives an idea about the standard deviation from the mean $s_{S/N}$ of this estimator.

Using the same algorithms 100 times for each experiment, one can compute the standard deviation $s_{S/N}$ from the mean value $\mu_{S/N}$ of the S/N. We have displayed the results in Fig 29: the mean value $\mu_{S/N}$ as "×," along with the upper "Δ" and lower "∇" bounds. These bounds are related to the mean value thanks to: $\mu_{S/N} + \sigma_{S/N}$ and $\mu_{S/N} - \sigma_{S/N}$.

We may conclude that the estimator performs well for S/N > 0 dB. The standard deviation from the mean increases for lower values—the results will be less accurate. Moreover, we can see in Fig 29 that the method tends to underestimate the S/N for very low values. Anyway, the dynamic range of the estimator extends to limits beyond realistic receiving conditions. It is indeed very difficult to recognize a SSTV signal at a very low S/Ns.

Estimating the Luminance

As explained above, the luminance demodulation is based upon a DFT of the incoming signal. The number of samples used to compute the DFT is driven by the estimated S/N, as a function $f(S/N)$. We are now going to describe the luminance demodulation system and the trade-off we had to face.

High-Quality SSTV

How many samples do we have to use to get an accurate estimate of the SSTV signal? We have already explained the idea behind our adaptive scheme. We now focus on the highest-quality demodulator. It is the upper bound of the luminance demodulator. Using a trial-and-error process, we finally decided to use a 37-point Chebyshev window (see Fig 30).

The windowed signal is then fed into a DFT. This DFT only computes the relevant part of the spectrum, thus saving many microprocessor cycles. The output power spectrum is then followed by a parabolic interpolation—a common practice in frequency estimation.

This choice leads to excellent frequency estimations for high S/Ns. Moreover, the horizontal resolution allowed by such a number of points is compatible with G3OQD specifications.

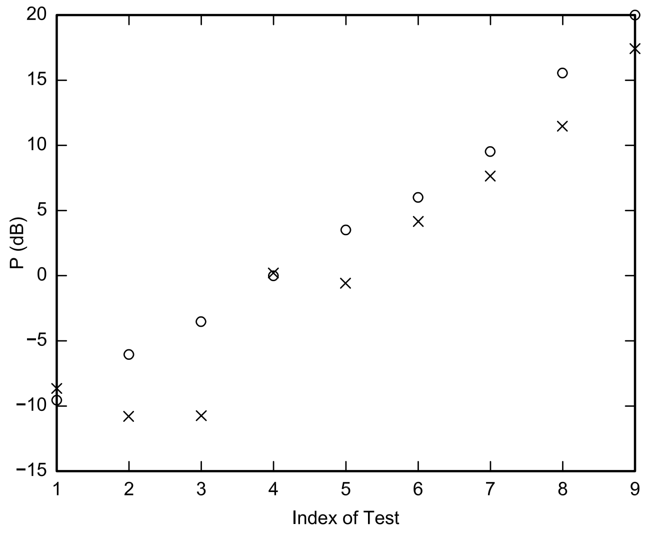


Fig 28—Original noise power (o) and estimated S/N (x).

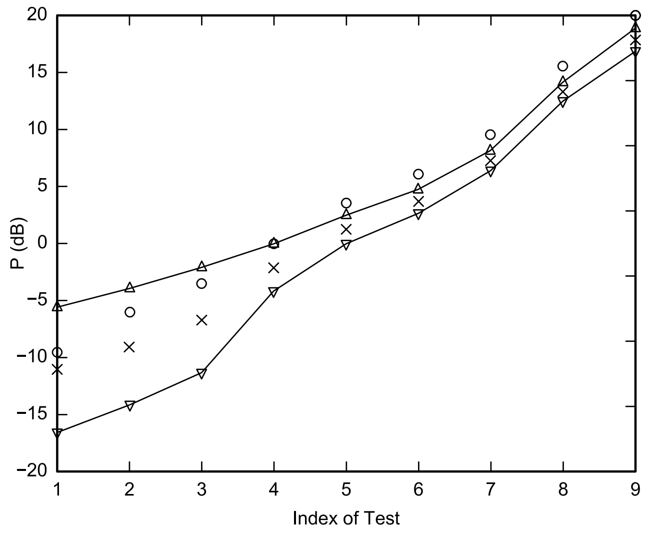


Fig 29—Standard deviation from the mean of the estimated S/N (o = input noise; x = estimated S/N).

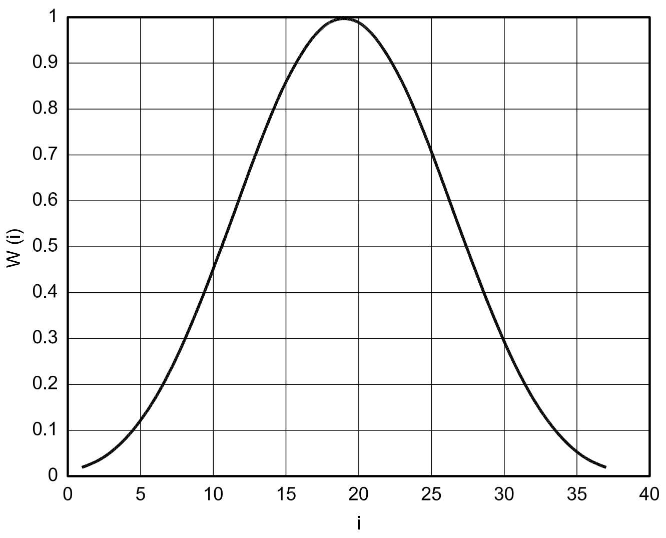


Fig 30—37-point Chebyshev window.

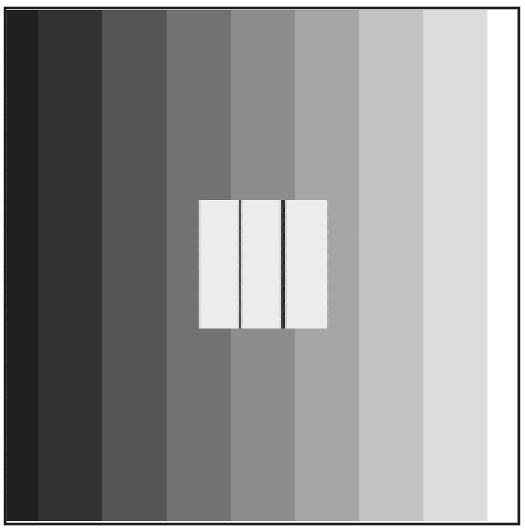


Fig 31—The reference picture

Some simulation results of this demodulator are presented below. We have used the same test picture as in Fig 6. The demodulated picture is presented in Fig 31. The luminance of the 128th line of the test picture is displayed in Fig 32. The luminance of the demodulated picture is displayed in Fig 33. One can check that the demodulated signal is very close to the reference. The only difference is a slight shift between the original and the demodulated pictures. It produces the black vertical line on the right. It will be fixed in a future release of the software.

This horizontal shift prevents us from using the usual peak S/N to estimate image quality. A far better quality estimate is to estimate bias and the standard deviation from the mean across a vertical line. The choice of a vertical line allows us to use samples that are not correlated. That topic is far beyond the scope of this paper.[24]

A Demodulator for a Low S/N

When the S/N is low, the software chooses a longer window—a Hanning window. The lengths used in the software range from 64-2048 points. The windowed signal is then followed by a FFT. Only the relevant part of the estimated spectrum is used to recover the frequency. Any maximum of the power spectrum outside of the video band (1500-2300 Hz) is excluded, thus helping to mitigate picture degradation induced by QRM.

The very same software has been used to demodulate a very noisy signal (more noise than SSTV signal). Two resulting pictures are presented: One was demodulated with a 37-sample DFT (Fig 34), and another demodulated with the adaptive, S/N-based demodulator (Fig 35).

We have also plotted the same 128th line using the following convention: The continuous line is for the adaptive scheme; the dots are for the 37-point DFT (see Fig 36). We can conclude the adaptive scheme, under these conditions, outperforms the basic demodulator.

Implementation Issues

The whole software has been coded in *C++*, some parts being *C* functions with a *C++* wrapper. It was developed using Borland *C++ Builder* version 3

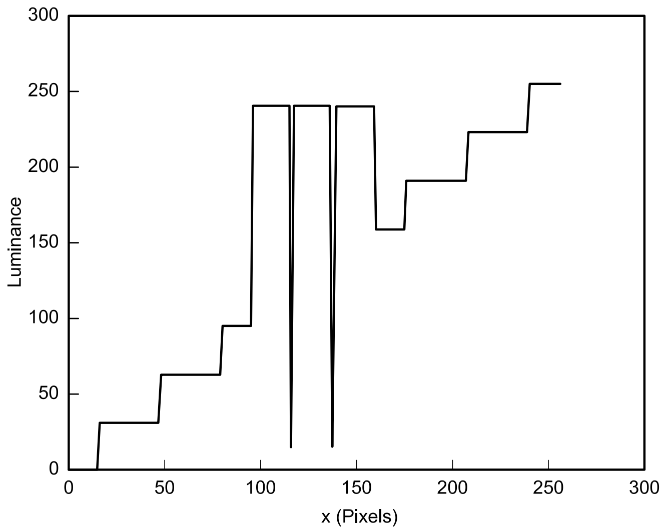


Fig 32—Picture demodulated with the 37-point DFT.

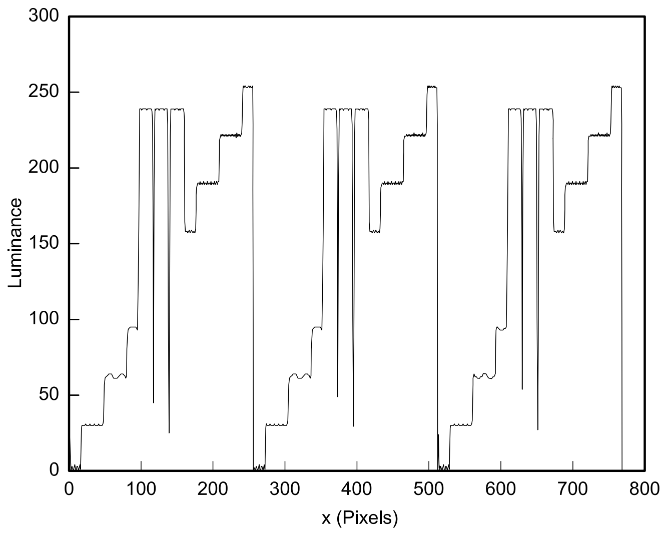


Fig 33—Demodulated signal, $N = 37$, line 128.

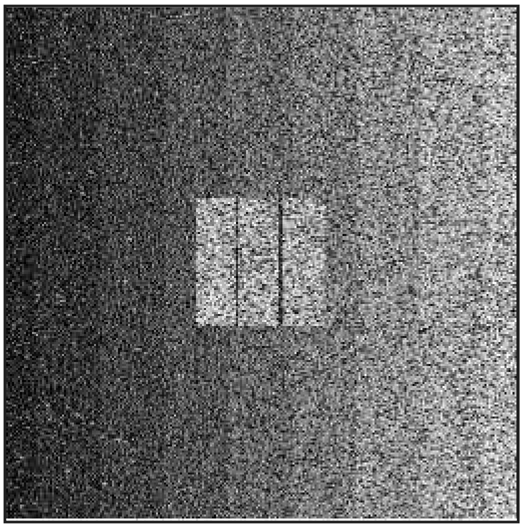


Fig 34—Noisy picture demodulated with a 37-point DFT.

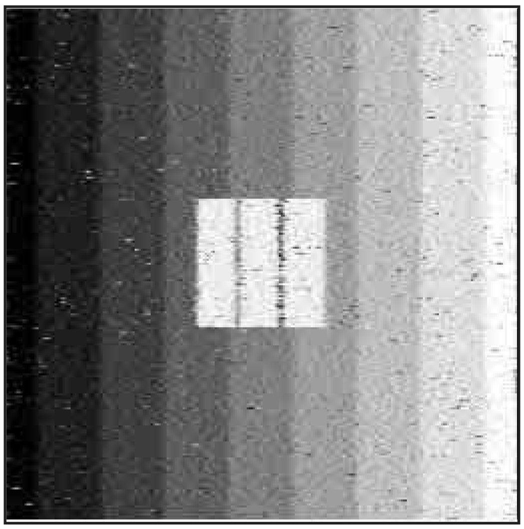


Fig 35—Noisy picture demodulated with the adaptive method.

(professional edition for students—the project started a while ago). More recently, it has also been compiled with version 5 of the same product without trouble.

The DFT relies upon two kinds of modules: a plain DFT and a FFT. The DFT is used for N = 37 points. Using some recent *C*++ coding techniques, the compiled code provides us with a very short execution time on a processor from the Pentium family. It relies upon a clever use of templates for the dot product of vectors.[26] As this topic is far from the scope of this paper, but of high interest to anyone "cooking" and coding DSP modules, we invite the interested reader to view references on the topic.[27, 28]

The FFT is used for both the video demodulator and the S/N estimator. We have used the "Fastest Fourier Transform in the West" library, also known as FFTW (see **www.fftw.org**).[29, 30] This software is licensed under the GNU General Public License (GPL, see **www.gnu.org**). The sound capture under the *Windows* operating system is a modified version of the code published in *Dr Dobbs Journal*.[31] As it uses

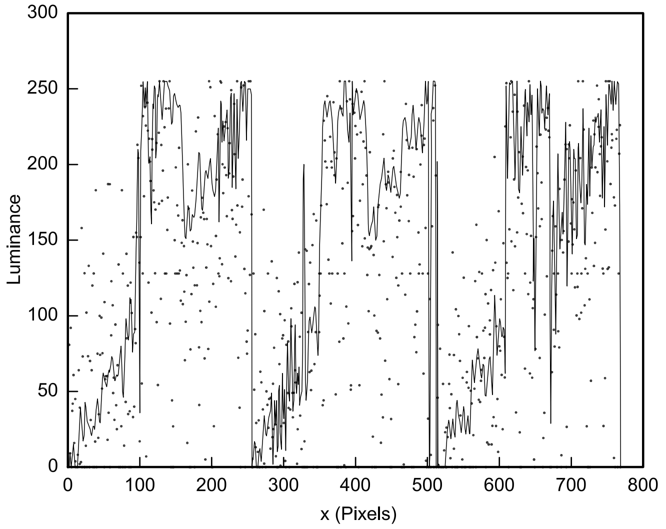


Fig 36—A noisy picture demodulated with both *N* = 37 and adaptive techniques, line 128.

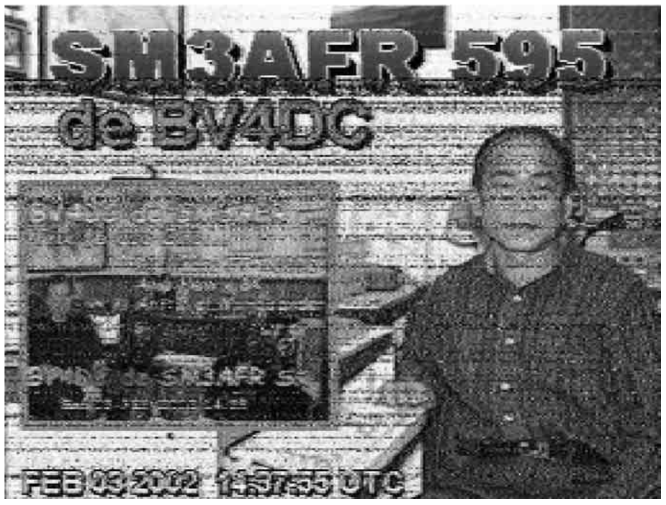


Fig 37—BV4DC's picture, *N* = 37.

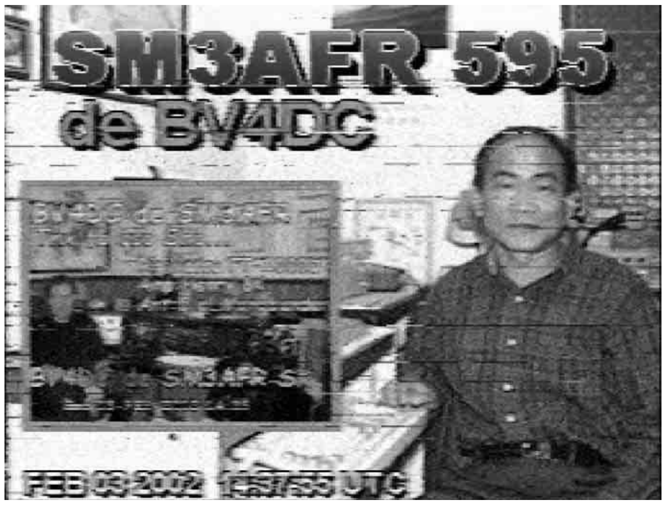


Fig 38—BV4DC's picture, adaptive.

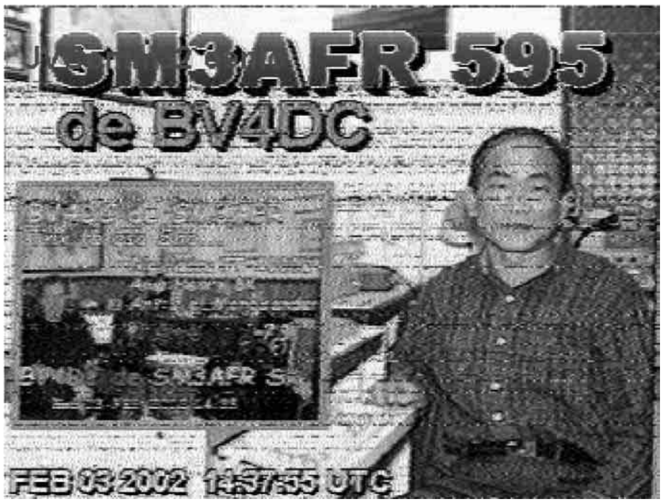


Fig 39—Same signal, demodulated with other common SSTV software.

Table 1—IIR Synchronization Filter Coefficients

b_0	$b1$	$b2$	$b3$	$b4$
1.9897E–004	0	–3.9794E–004	0	1.9897E–004
a_0	a_1	a_2	a_3	a_4
1	–3.9024E+000	5.7672E+000	–3.8245E+000	9.6050E–001

software released under the GPL, our SSTV software is also released under the GPL. Look for the source code at **roland.cordesses.free.fr/sstvrep** under "RealTime Processing."

Experimental Results

The program described here, running under the *Windows* operating system, can only receive M1 SSTV signals. While not having all the bells and whistles found in today's software, it has been designed to match various experimental conditions. The way the picture appears on the PC screen is close to any SSTV program, but we can point out some unusual features our software presents.

• The flux of SSTV lines is not constant and changes with the levels of noise and interference.

• When one tunes a SSTV transmission the beginning of which has been missed, some strange colors are first presented on the screen; but after a few seconds, the system resynchronizes itself even in presence of noise or QRM.

• At the end of a slanted picture, the program "unslants" it automatically.

• The result file presents some statistical information associated with a received picture and helps one understand how the program processed the SSTV signal.

Moreover, the user can store not only the received picture (BMP file) but also the associated audio signal (WAV file). It is therefore possible to replay the audio file as often as desired to experiment with different processing configurations. During our experiments, we produced many audio CD files. Thus we could test not only the various capabilities of our program, but also compare them to other popular SSTV software.

Figs 37 through 39 show examples of a picture transmitted by BV4DC on the 15-meter band. The propagation changed quickly and the signal, which was good at the beginning of the transmission, suddenly dropped with a slow and deep fading. Some interference appeared at the end of the picture.

Fig 37 presents the BV4DC picture as received with the 37-sample DFT, while Fig 38 shows the same signal demodulated with the adaptive, S/N based demodulation. It is clear that this last picture is far less noisy than the first, but it does not have the same sharpness. The adaptive algorithm is doing a nice job under those poor conditions, and we can accept easily the small loss of picture resolution.

Fig 39 is a picture from the same audio signal produced by one of the best available programs we have tested. The picture quality is slightly better than that of the 37-samples FFT of our software, but far behind that of the adaptive method shown in Fig 38.

Conclusion

We have presented some unconventional ways of processing SSTV synchronization signals and an adaptive approach to extract luminance information. The results obtained with our program making use of these algorithms are spectacular, especially when receiving conditions are poor.

As a final word, when we were reviewing the papers presented in the bibliography, we were very impressed by the conclusion in Reference 3: "We can be assured that picture and graphical processing by computer will shortly pervade many aspects of our lives." That sentence, written more than 30 years ago and before the introduction of the microprocessor, was really prophetic and has been subsequently verified!

Acknowledgements

First of all, I would like to thank Professor Jean Gallice. Back in 1996, he spent many hours casting a new light on frequency estimation. He is the man behind our DFT idea.

Thanks to G3OQD, the father of the popular Martin M1 mode. He kindly sent us the exact timings of the M1 mode (see Note 1).

And lastly, we must mention (in alphabetical order): Donovan, Bob Dylan, Tom Paxton; Peter, Paul and Mary. Thanks for the musical background!

Notes

[1]Martin Emerson G3OQD. M1 technical specifications in personal communication, February 1993.

[2]C. Macdonald, "A Compact Slow Scan TV Monitor," *QST*, Mar 1964, pp 43-48.

[3]Th. Cohen, H. L. Husted, and P. R. Lintz, "Computer Processing Slow-scan Television Pictures," *ham radio*, Jul 1970, pp 30-37.

[4]J. R. Montalbano, "The ViewPort VGA Color SSTV System," *73 Amateur Radio Today*, Aug 1992, pp 8-16.

[5]R. Cordesses and L.Cordesses, "SSTV couleur et Fax sur compatible IBM PC," (in French) *Radio REF*, juillet-aôut 1994, pp 23-27.

[6]*The ARRL Handbook for Radio Amateurs* (Newington, Connecticut: ARRL, 2000; ISBN 0-87259-183-2), Chapter 12, p 12.44.

[7]C. Marven and G. Ewers, *A Simple Approach to Digital Signal Processing* (Texas Instruments, 1994; ISBN 0-904 047-00-8), Chapter 4.

[8]An analog Bessel filter is a better choice than an analog Butterworth filter for this kind of application. Unfortunately, we are not able to design a digital Bessel filter.

[9]The so-called Scottie 1 or S1 color mode may use a longer horizontal synchronization signal. Unfortunately, we have never been able to get information from the creator of this mode. The author of *JVComm32*, DK8JV, has kindly sent us the timings he uses for his software for the S1 mode. Design and simulations should be done with a 10-ms burst for the S1 mode.

[10]W. Press, S. Teukolsky, W. Vetterling, and B. Flannery, *Numerical Recipes in C* (Cambridge University Press, 1994; ISBN 0-521 43108 5) Chapter 15.

[11]R. O. Duda and P. E. Hart, "Use of the Hough Transformation to Detect Lines and Curves in Pictures, *Communications of the ACM* (Association for Computing Machinery), 15:11-15, Jan 1972.

[12]P. V. C. Hough, *Method and Means for Recognizing Complex Patterns,* US Patent Specification US3069654, Dec 1967.

[13]J. Illingworth and J. Kittler, "A survey of the Hough transform," ACM *Computer Vision, Graphics, and Image Processing*, Vol 44, Issue 1, pp 87-116, Aug 1988.

[14]J. Langner, "Slow-Scan TV: It isn't Expensive Anymore!" *QST*, Jan 1993, pp 20-30.

[15]D. J. Thomson, "Highlights of Statistical Signal and Array Processing, Multiple Window Spectrum Estimate," *IEEE Signal Processing Magazine*, Sep 1998, 15(5): 30-32.

[16]D. Sirmans and B. Bumgarner, "Numerical Comparison of Five Frequency Estimators," *Journal of Applied Meteorology*, Sep 1973, 14: 991-1003.

[17]F. Baudin, "Estimateurs de Fréquence pour Mesurer la Dérive Doppler Sodar," (in French) Technical report: *Note Technique CRPE/CNET 38*, Jan 1977.

[18]C. Marven and G. Ewers, *A Simple Approach to Digital Signal Processing* (Texas Instrument, 1994; ISBN 0-904 047-00-8) Chapter 5.

[19]D. Smith, "Signals, Samples, and Stuff: A DSP Tutorial (Part 3)," *QEX*, July 1998, pp 13-27.

[20]W. Press, S. Teukolsky, W. Vetterling, and B. Flannery, *Numerical Recipes in C* (Cambridge University Press, 1994; ISBN 0-521 43108 5) Chapter 12.

[21]G. Bretthorst, *To Appear in Maximum Entropy and Bayesian Methods* (Netherlands: Kluwer, 2000), Chapter "Nonuniform Sampling: Bandwidth and Aliasing."

[22]C. E. Shannon, "Communication in the Presence of Noise," *Proceedings of the IRE*, 37: 10-21, 1949.

[23]A. Vizinho, P. Green, M. Cooke, and L. Josifovski, "Missing Data Theory, Spectral Subtraction and Signal-to-Noise Estimation for Robust ASR: an Integrated Study," in *Proceedings of EuroSpeech '99*, pp 2407-2410, Budapest, 1999.

[24]We have used the bias and the standard deviation from the mean to design and qualify the demodulator. Although it is a powerful tool, its results are hardly linked to the visual quality of the picture. The same problem applies to the aforementioned PS/N (see Note 25).

[25]M. J. Nadenau, S. Winkler, D. Alleysson and M. Kunt, "Human Vision Models for Perceptually Optimized Image Processing—A Review," submitted to *Proceedings of the IEEE*, Sep 2000.

[26]T. L. Veldhuizen, "C++ Templates as Partial Evaluation," In *ACM SIGPLAN Workshop on Partial Evaluation and Semantics-Based Program Manipulation* (PEPM'99), pp 13-18, San Antonio, Texas; Jan 1999.

[27]Todd L. Veldhuizen, "Expression Templates," *C++ Report*, 7(5): 26-31, Jun 1995. (Reprinted

in *C++ Gems*, Editor Stanley Lippman.)

[28]T. Veldhuizen and K. Ponnambalam, "Linear Algebra with *C++*-template metaprograms," *Dr. Dobb's Journal*, 21(8): 38-44, Aug 1996.

[29]M. Frigo and S. G. Johnson. "FFTW: An Adaptive Software Architecture for the FFT," in *ICASSP 1998*, volume 3, pages 1381-1384, 1998.

[30]M. Frigo, "A fast-Fourier-transform compiler," *Proceedings of the ACM SIGPLAN '99 Conference on Programming Language Design and Implementation*, Atlanta, Georgia, United States, 1999, pp 169-180.

[31]R. Cook, "Real-Time Sound Processing," *Dr. Dobb's Journal*, Oct 1998.

Lionel Cordesses first discovered SSTV on his father's (F2DC) P7-phosphor SSTV monitor when Lionel was 6 years old. Ten years and a few homemade receivers later, he designed his own PC-XT based SSTV demodulation software.

1994 saw the beginning of a bench program to test image-decoding algorithms. Soon limited by the performance of the processor, the project revived and finally ended in November 2001, thanks to more number crunching power, and a stronger theoretical knowledge.

He is now an Electronics Engineer (1997) with a PhD in electronics (2001). His research interests include GPS-based control of farm vehicles and analog electronics. He then joined the R&D electronics department of RENAULT Agriculture, a company manufacturing farm tractors. He now focuses on control systems and embedded controllers. As an engineer, he always tries to explain complex theories with simple examples, trying to bridge the theory-practice gap. He is an IEEE and ACM member.

His hobbies include building model aircraft and their related electronics and flying indoor-light models. He also enjoys drawing Donald Duck and other characters of the Duck family in the Carl Barks' way. He is also fond of French and American film noir of the 1940s and 50s.

Roland Cordesses, F2DC, was licensed in 1962, and he has been a member of ARRL since 1964. He graduated as an Electronics Engineer and is presently a Research Engineer at OPGC, an Observatory devoted to atmospheric and earth sciences he joined in 1973.

For nearly 30 years, he has been working in the design and development of radar systems for remote sensing of the atmosphere. Recently, he designed and field-tested the world's first Doppler radar specifically devoted to the monitoring of volcanic eruptions. During his career, he has presented or published many papers related to electronics and geophysics.

Roland's Amateur Radio interests mainly deal with homebrewing: Over the years, he has designed and built many receivers, transceivers and SSTV projects.

Aside from Amateur Radio, he enjoys building and flying model aircraft (homemade transmitters and recei-vers, of course) and hiking in the mountains.