Order Number 9325098

The Weil transform and ambiguity functions

Geshwind, Frank B., Ph.D.

City University of New York, 1993

U·M·I
300 N. Zeeb Rd.
Ann Arbor, MI 48106

The Weil Transform And Ambiguity Functions

by

Frank B. Geshwind

A dissertation submitted to the Graduate Faculty in Mathematics in partial fulfillment of the requirements for the degree of Doctor of Philosophy, The City University of New York

1993

This manuscript has been read and accepted for the Graduate Faculty in Mathematics in satisfaction of the dissertation requirements for the degree of Doctor of Philosophy.

04/22/93
Date

Chair of Examining Committee

04/22/93
Date

Executive Officer

Jozef Dodziuk

Alphonse Vasquez

Richard Sacksteder

Supervisory Committee

THE CITY UNIVERSITY OF NEW YORK

Abstract

The Weil Transform And Ambiguity Functions

by

Frank B. Geshwind

Advisor: Professor Louis Auslander

Recall that the Weil transform, Θ, is an intertwining operator between the $\mathbf{L}^2$ Dirac representation D, of the real, 3-dimensional Heisenberg group, $\mathcal{N}$, and the regular representation of $\mathcal{N}$ on $\mathbf{L}^2(\Gamma\backslash\mathcal{N})$, where Γ is the integer Heisenberg group. As such, it provides a unitary isomorphism between $\mathbf{L}^2(\mathbf{R})$ and $\mathbf{L}^2(\Gamma\backslash\mathcal{N})$.

In this work, the definition of Θ is extended so that its domain includes various types of distributions. Results in the literature relating smoothness and moments of functions on $\mathbf{R}$ to smoothness and moments of the corresponding functions on $\Gamma\backslash\mathcal{N}$, are extended to distributions.

A study of the local structure of smooth functions in $\Theta(\mathbf{L}^2(\mathbf{R}))$ is carried out. A product theorem is given for certain functions which we describe as having smooth growth. Theorems are given about the local homology of these functions. These results are applied to the problem of dividing smooth functions in $\Theta(\mathbf{L}^2(\mathbf{R}))$ by each other, which is related to questions of stable Weyl-Heisenberg wavelet expansions for such functions.

The Weil transform and the above insights into smooth functions in $\Theta(\mathbf{L}^2(\mathbf{R}))$ are applied to the thumbtack synthesis problem for radar ambiguity functions. We study known constructions for this problem, in the present context, and then provide new constructions using the insights gained.

Acknowledgments

I would like to express my deep appreciation to Professor Louis Auslander for providing stimulus, guidance and support throughout the latter part of my graduate studies, and during the genesis and preparation of this work.

I wish to thank Fred Warner and Marina Saadia-Otero for their encouragement and support.

I would also like to thank my wife, Eileen McCarthy, for her love and support during the writing of this dissertation: It would not have been possible without her.

Contents

List of Tables

List of Figures

1 The Weil Transform, Moments and Distributions on $\Gamma\backslash\mathcal{N}$

1.1 The Heisenberg Group

The Heisenberg group $\mathcal{N}$ is defined as follows. As a set $\mathcal{N} = \mathbf{R}^3$. The group operation is defined by

$$(x_1, y_1, z_1)(x_2, y_2, z_2) = (x_1 + x_2, y_1 + y_2, z_1 + z_2 + x_1 y_2).$$

Let Γ be the subgroup of $\mathcal{N}$ consisting of all integer lattice points (i.e., points for which all coordinates are integers) and form the coset space $\Gamma\backslash\mathcal{N}$. It turns out that $\Gamma\backslash\mathcal{N}$ is the unit cube $0 \leq x, y, z \leq 1$ with certain identifications on its boundary. See [1] for details. Since Haar measure on $\mathcal{N}$ is $dx\,dy\,dz$, the $\mathbf{L}^p$ space on $\Gamma\backslash\mathcal{N}$ with norm given by

$$\|f\|_p = \left(\int_0^1 \int_0^1 \int_0^1 |f(x, y, z)|^p dx\,dy\,dz\right)^{\frac{1}{p}},$$

will be denoted by $\mathbf{L}^p(\Gamma\backslash\mathcal{N})$.

1.2 The Weil Transform

Let $\mathcal{S}$ be the class of Schwartz functions on $\mathbf{R}$, and let $f \in \mathcal{S}$. The Weil transform of f, $\Theta(f)$, is a function on $\mathcal{N}$ defined formally by

$$\Theta(f)(x, y, z) = e^{2\pi i z} \sum_{n \in \mathbf{Z}} f(x + n) e^{2\pi i n y}.$$

Since $f \in \mathcal{S}$, the sum converges absolutely and uniformly on compact sets. Further, $\Theta(f)$ is invariant under the left action of Γ, and hence $\Theta(f)$ can be thought of as a function on $\Gamma\backslash\mathcal{N}$.

Note that a transform that is essentially the same as the Weil transform, but which is ordinarily used in contexts that ignore the Heisenberg group theory, is the Zak transform. For a discussion of this transform, and some of the historical controversy over the names of these related transforms, see [24].

It will now be shown that the Weil transform extends to a transform on a wide class of spaces. The tools for this are the following:

Definition 1.1 *Let* $\mathbf{X}, \mathbf{Y}$ *be metric spaces. Then a map* $f : \mathbf{X} \to \mathbf{Y}$ *is called* non-expanding *if for all* $x_1, x_2 \in \mathbf{X}$

$$d_{\mathbf{X}}(x_1, x_2) \geq d_{\mathbf{Y}}(f(x_1), f(x_2))$$

The following theorem is standard. It is usually proved for f uniformly continuous (see, for example, [27], chapter 6, proposition 11), but a non-expanding map is clearly uniformly continuous (just take $\delta = \epsilon$).

Theorem 1.1 *Let* $\mathbf{X}, \mathbf{Y}$ *be metric spaces, and let* $\mathbf{Y}$ *be complete. Let* $\mathbf{S} \subset \mathbf{X}$ *be a dense subset of* $\mathbf{X}$*, and let* $f : \mathbf{S} \to \mathbf{Y}$ *be a non-expanding map. Then* f *extends uniquely to a non-expanding map* $\tilde{f} : \mathbf{X} \to \mathbf{Y}$.

Theorem 1.2 *The Weil transform is a non-expanding map of* $\mathcal{S}$ *with* $\mathbf{L}^1$ *norm, into* $\mathbf{L}^1(\Gamma\backslash\mathcal{N})$.

Proof: Let $f \in \mathcal{S}$. Then

$$\begin{aligned}
\|\Theta(f)\|_1 &= \int_0^1 \int_0^1 \int_0^1 \left| e^{2\pi iz} \sum_{n\in\mathbf{Z}} f(x+n) e^{2\pi iny} \right| dx\, dy\, dz \\
&= \int_0^1 \int_0^1 \left| \sum_{n\in\mathbf{Z}} f(x+n) e^{2\pi iny} \right| dx\, dy \\
&\leq \int_0^1 \int_0^1 \sum_{n\in\mathbf{Z}} \left| f(x+n) e^{2\pi iny} \right| dx\, dy \\
&= \sum_{n\in\mathbf{Z}} \int_0^1 |f(x+n)|\, dx \\
&= \int_{\mathbf{R}} |f(t)|\, dt \\
&= \|f\|_1.
\end{aligned}$$

The change of order of integral and summation is justified since $f \in \mathcal{S}$. □

It is pointed out in [17] that the above fact, together with what is known about Θ on $\mathbf{L}^2(\mathbf{R})$, yields the following:

Corollary 1.3 *For any $1 \leq p \leq 2$, Θ is a non-expanding map from $\mathcal{S}$ with $\mathbf{L}^p$ norm into $\mathbf{L}^p(\Gamma\backslash\mathcal{N})$.*

Corollary 1.4 *For any $1 \leq p \leq 2$, Θ extends uniquely to a non-expanding map, $\Theta : \mathbf{L}^p(\mathbf{R}) \to \mathbf{L}^p(\Gamma\backslash\mathcal{N})$.*

Proof: This follows immediately from 1.1 and 1.3, since $\mathcal{S}$ is dense in $\mathbf{L}^p(\mathbf{R})$. □

In the case $p = 2$ more can be said. Indeed it is a standard result (see [1]), that Θ is a unitary isomorphism between $\mathbf{L}^2(\mathbf{R})$ and a certain closed subspace

$H_1 \subset \mathbf{L}^2(\Gamma\backslash\mathcal{N})$. H_1 is defined as the set of all functions in $\mathbf{L}^2(\Gamma\backslash\mathcal{N})$ which satisfy:

$$f(x,y,z) = e^{2\pi iz} f(x,y,0).$$

For details, see [1].

Functions in H_1 are precisely those functions $F(x,y,z)$ which are in $\mathbf{L}^2$ of the unit cube and satisfy:

$$F(x,y,z) = F(x,y,z+1) \tag{1}$$

$$F(x,y,z) = F(x,y+1,z) \tag{2}$$

$$F(x,y,z) = F(x+1,y,z+y) \tag{3}$$

$$F(x,y,z+u) = e^{2\pi iu} F(x,y,z). \tag{4}$$

In order to specify functions in H_1, the following will be a useful simplification.

Continuous functions $F \in H_1$ are in one to one correspondence with continuous functions $\tilde{F}$ defined on the unit square, which satisfy the following:

$$\tilde{F}(x,0) = \tilde{F}(x,1) \tag{5}$$

$$\tilde{F}(0,y) = e^{2\pi iy} \tilde{F}(1,y) \tag{6}$$

This can be seen as follows. Certainly, for any function $F(x,y,z) \in H_1$, the function $\tilde{F}(x,y) = F(x,y,0)$ restricted to the unit square, must satisfy (5), since

this is a special case of equation (2). To see that it also satisfies (6), simply note that

$$\begin{aligned}\tilde{F}(0,y) &= F(0,y,0)\\ &= F(1,y,y)\\ &= e^{2\pi iy}F(1,y,0)\\ &= e^{2\pi iy}\tilde{F}(1,y)\end{aligned}$$

Conversely, given $\tilde{F}(x,y)$ which satisfies (5) and (6), define, for x,y,z in the unit cube,

$$F(x,y,z) = e^{2\pi iz}\tilde{F}(x,y).$$

Equations (1)–(3) together with equation (4) now give the extra conditions which this F must satisfy. Here a function has been defined on a set which is slightly bigger than a fundamental domain for the group action, so it must be shown that, under the action of the generators of the group Γ, the function is invariant.

Equation (1) is satisfied since

$$\begin{aligned}F(x,y,z) &= e^{2\pi iz}\tilde{F}(x,y)\\ &= e^{2\pi i(z+1)}\tilde{F}(x,y)\\ &= F(x,y,z+1)\end{aligned}$$

Equation (2) is satisfied since the only time when y and $y+1$ are both in

the unit cube is when $y = 0$, and then equation (2) is trivially equivalent to equation (5). Similarly, equation (3) is equivalent to equation (6), because the only time when both x and $x+1$ are in the unit cube is when $x = 0$, and in that case we have:

$$\begin{aligned} F(0,y,z) &= e^{2\pi iz}\tilde{F}(0,y) \\ &= e^{2\pi iz}e^{2\pi iy}\tilde{F}(1,y) \\ &= e^{2\pi i(z+y)}\tilde{F}(1,y) \\ &= F(1,y,z+y) \end{aligned}$$

Equation (4) is satisfied by definition.

From now on the 'tilde' will be dropped from the discussion, and functions of two variables defined on the unit square will be considered interchangeably with functions of three variables on the unit cube. In this spirit, an $\mathbf{L}^2$ function on the unit square which satisfies equations (5) and (6) will be said to be in H_1.

Note that for $F \in H_1$,

$$\left[\Theta^{-1}(F)\right](t) = \int_0^1 F(t,y,0)\,dy. \tag{7}$$

In particular, if $F \in H_1$ is continuous, then $\Theta^{-1}(F)$ is continuous. Again, see [1] for details.

Another useful fact about the Weil transform, which is discussed in [1], is that the Fourier transform takes on a simple form under Θ. If we define the map J of

$\mathcal{N}$ onto itself defined by

$$J(x,y,z) = (-y, x, z - xy), \tag{8}$$

then J defines a C^∞ mapping of $\Gamma\backslash\mathcal{N}$ onto itself which we also denote by J.

In [5] it was verified that if $\mathcal{F} : \mathbf{L}^2(\mathbf{R}) \to \mathbf{L}^2(\mathbf{R})$ is the Fourier transform, then

$$\Theta^{-1} J \Theta = \mathcal{F}. \tag{9}$$

1.3 Distributions on $\Gamma\backslash\mathcal{N}$

This section will explore extending the domain of definition of the Weil transform to include more general objects. The results here will be applied in the next section to settle some questions raised in [1].

Define $ß = C^\infty(\Gamma\backslash\mathcal{N}) \cap H_1$. Our first task will be to show that Θ is a bijection between $\mathcal{S}$ and ß.

Lemma 1.5 *Let $f \in \mathcal{S}$. Then $\Theta(f')(x,y,z) = \frac{\partial}{\partial x}\Theta(f)(x,y,z)$.*

Proof:

$$\begin{aligned}
\Theta(f')(x,y,z) &= e^{2\pi i z} \sum_{a \in \mathbf{Z}} f'(x+a) e^{2\pi i a y} \\
&= \frac{\partial}{\partial x} e^{2\pi i z} \sum_{a \in \mathbf{Z}} f(x+a) e^{2\pi i a y} \\
&= \frac{\partial}{\partial x} \Theta(f)(x,y,z).
\end{aligned}$$

Note that differentiation may be moved outside the sum because the sum converges uniformly on compact sets. □

Lemma 1.6 *Let* $F \in ß$. *Then* $\Theta^{-1}(\frac{\partial}{\partial x}F(x,y,z)) = \frac{d}{dt}\Theta^{-1}(F)(t)$.

Proof: First note that $F \in ß$ implies that $\frac{\partial F}{\partial x} \in ß$, and hence $\Theta^{-1}(\frac{\partial F}{\partial x})$ is well defined and continuous. Also,

$$F(x,y,z) = \int_{x_0}^{x} \frac{\partial F}{\partial x}(t,y,z)\,dt + F(x_0,y,z). \tag{10}$$

Let $g(t) = \Theta^{-1}(\frac{\partial F}{\partial x})(t)$. Let

$$\begin{aligned} f(t) &= \Theta^{-1}(F)(t) \\ &= \int_0^1 F(t,y,0)\,dy \qquad (11) \\ &= \int_0^1 \int_{t_0}^{t} \frac{\partial F}{\partial x}(x,y,0)\,dx + F(t_0,y,0)\,dy \\ &= \int_{t_0}^{t} \int_0^1 \frac{\partial F}{\partial x}(x,y,0)\,dy\,dx + f(t_0) \qquad (12) \\ &= \int_{t_0}^{t} g(x)\,dx + f(t_0). \end{aligned}$$

The exchange of order of integration in (12) is justified by Fubini's theorem because of the convergence of the integrals in (10) and (11). Note that equation (11) comes from equation (7).

It follows that $g(t) = \frac{df}{dt}$, and the result is proved. □

Lemma 1.7 *Let $f \in \mathcal{S}$. Then*

$$\Theta(tf(t)) = x\Theta(f) + \frac{1}{2\pi i}\frac{\partial}{\partial y}\Theta(f).$$

Proof: Again the formal calculation below may be justified for $f \in \mathcal{S}$.

$$\begin{aligned}
\Theta(tf(t)) &= e^{2\pi iz}\sum_{a\in\mathbf{Z}}(x+a)f(x+a)e^{2\pi iay} \\
&= (x+\frac{1}{2\pi i}\frac{\partial}{\partial y})e^{2\pi iz}\sum_{a\in\mathbf{Z}}f(x+a)e^{2\pi iay} \\
&= x\Theta(f)(x,y,z) + \frac{1}{2\pi i}\frac{\partial}{\partial y}\Theta(f)(x,y,z).
\end{aligned}$$

□

Lemma 1.8 *Let $F \in \beta$. Then*

$$\Theta^{-1}(xF(x,y,z) + \frac{1}{2\pi i}\frac{\partial}{\partial y}F(x,y,z)) = t\Theta^{-1}(F)(t).$$

Proof: By equation (7) we have that

$$\begin{aligned}
\Theta^{-1}(xF(x,y,z) &+ \frac{1}{2\pi i}\frac{\partial}{\partial y}F(x,y,z))(t) \\
&= \int_0^1 tF(t,y,z) + \frac{1}{2\pi i}\frac{\partial}{\partial y}F(t,y,z)\,dy \\
&= t\int_0^1 F(t,y,z)\,dy + \frac{1}{2\pi i}\left(F(t,1,z) - F(t,0,z)\right) \\
&= t\Theta^{-1}(F)(t).
\end{aligned}$$

□

Lemma 1.9 *If $F \in ß$, then $\Theta^{-1}(F) \in \mathcal{S}$.*

Proof: Let $F \in ß$. This means that F is a C^∞ Γ-invariant function on $\mathcal{N}$. First observe that this implies that $\frac{\partial F}{\partial x}$ and $\left(x + \frac{1}{2\pi i}\frac{\partial}{\partial y}\right) F(x,y,z)$ have this property, and hence are in ß. To see this let $\gamma = (n_1, n_2, n_3) \in \Gamma$. We compute

$$\begin{aligned}
\left.\frac{\partial F}{\partial x}\right|_{\gamma(x,y,z)} &= \lim_{x_0 \to 0} \left.\frac{F(x+x_0,y,z) - F(x,y,z)}{x_0}\right|_{\gamma(x,y,z)} \\
&= \lim_{x_0 \to 0} \left(\frac{F(x+n_1+x_0, y+n_2, z+n_3+n_1 y)}{x_0} \right. \\
&\quad \left. - \frac{F(x+n_1, y+n_2, z+n_3+n_1 y)}{x_0} \right) \\
&= \lim_{x_0 \to 0} \frac{F(x+x_0,y,z) - F(x,y,z)}{x_0} \\
&= \frac{\partial F}{\partial x}(x,y,z).
\end{aligned}$$

Next, observe that

$$\left((x + \frac{1}{2\pi i}\frac{\partial}{\partial y})F\right)(\gamma(x,y,z)) = (x+n_1)F(\gamma(x,y,z)) + \left(\frac{1}{2\pi i}\frac{\partial}{\partial y}F\right)(\gamma(x,y,z)).$$

But

$$\begin{aligned}
\left(\frac{\partial}{\partial y}F\right)(\gamma(x,y,z)) &= \lim_{y_0 \to 0} \left.\frac{F(x,y+y_0,z) - F(x,y,z)}{y_0}\right|_{\gamma(x,y,z)} \\
&= \lim_{y_0 \to 0} \frac{F(x+n_1, y+y_0+n_2, z+n_3+n_1 y) - F(x,y,z)}{y_0} \\
&= \lim_{y_0 \to 0} \frac{e^{-2\pi i n_1 y_0} F(x,y+y_0,z) - F(x,y,z)}{y_0} \\
&= \lim_{y_0 \to 0} \left(e^{-2\pi i n_1 y_0} - 1\right) \frac{F(x,y+y_0,z)}{y_0} + \frac{\partial F}{\partial y}(x,y,z) \\
&= \lim_{y_0 \to 0} -2\pi i n_1 e^{-2\pi i n_1 y_0} F(x,y+y_0,z)
\end{aligned}$$

$$+(e^{-2\pi i n_1 y_0}-1)\frac{\partial}{\partial y_0}F(x,y+y_0,z)+\frac{\partial F}{\partial y}(x,y,z)$$
$$= -2\pi i n_1 F(x,y,z)+\frac{\partial F}{\partial y}(x,y,z),$$

and it follows that

$$(x+\frac{1}{2\pi i}\frac{\partial}{\partial y})F(\gamma(x,y,z))=(x+\frac{1}{2\pi i}\frac{\partial}{\partial y})F(x,y,z).$$

The lemma follows by induction since we can then multiply F by polynomials in $\frac{\partial}{\partial x}$ and $\left(x+\frac{1}{2\pi i}\frac{\partial}{\partial y}\right)$ and get an $\mathbf{L}^2$ function. Hence we can multiply $\Theta^{-1}(F)$ by polynomials in t and $\frac{d}{dt}$ and get a function in $\mathbf{L}^2(\mathbf{R})$. But this is one of the ways of defining the fact that $\Theta^{-1}(F)\in\mathcal{S}$, (see, for example, [12], appendix A.2). □

Lemma 1.10 $f\in\mathcal{S}\Rightarrow\Theta(f)\in ß$.

Proof: Let $f\in\mathcal{S}$, and $F=\Theta(f)$. Then $tf(t)\in\mathcal{S}$, and $f'(t)\in\mathcal{S}$. Hence F is continuous, as are $\frac{\partial}{\partial x}F$ and $\left(x+\frac{1}{2\pi i}\frac{\partial}{\partial y}\right)F$. But, since F is continuous, $xF(x,y,z)$ is continuous. Hence $\frac{\partial}{\partial y}F$ is continuous.

We can now apply induction. Suppose that for any n_1,n_2, with $n_1+n_2\leq n-1$, $\frac{\partial^{n_1}}{\partial x^{n_1}}\frac{\partial^{n_2}}{\partial y^{n_2}}F$ is continuous. Now, since $f\in\mathcal{S}$, $t^{n_1}f^{(n_2+1)}(t)\in\mathcal{S}$, and $t^{n_1+1}f^{(n_2)}(t)\in\mathcal{S}$. Hence $\left(x+\frac{1}{2\pi i}\frac{\partial}{\partial y}\right)^{n_1}\left(\frac{\partial}{\partial x}\right)^{n_2+1}F$ is continuous. But $\left(x+\frac{1}{2\pi i}\frac{\partial}{\partial y}\right)^{n_1}\left(\frac{\partial}{\partial x}\right)^{n_2+1}F=\left(\frac{\partial}{\partial y}\right)^{n_1}\left(\frac{\partial}{\partial x}\right)^{n_2+1}F+$ terms which are products of x, F, and lower total degree powers of $\frac{\partial}{\partial y}$, and $\frac{\partial}{\partial x}$. But all of the latter terms are continuous by the induction hypothesis, so $\left(\frac{\partial}{\partial y}\right)^{n_1}\left(\frac{\partial}{\partial x}\right)^{n_2+1}F$ is continuous. Similarly, $\left(\frac{\partial}{\partial y}\right)^{n_1+1}\left(\frac{\partial}{\partial x}\right)^{n_2}F$ is continuous. □

Recall (see, for example, [12], appendix A.2), that the topology on $\mathcal{S}$ can be given by the semi-norms

$$\|f\|_{\alpha,\beta,2} = \left\| t^\alpha \left(\frac{d}{dt}\right)^\beta f(t) \right\|_2 .$$

We can define an analogous topology on ß via the semi-norms

$$\|F\|_{\alpha,\beta,2} = \left\| (x + \frac{1}{2\pi i}\frac{\partial}{\partial y})^\alpha \left(\frac{\partial}{\partial x}\right)^\beta F(x,y,z) \right\|_2 ,$$

for all $\alpha, \beta \in \mathbf{Z}_+$. Since Θ and Θ^{-1} preserve $\mathbf{L}^2$ norms, these two families of semi-norms are the same under the correspondence, and hence we have set things up to make the following theorem trivial.

Theorem 1.11 *Θ is a homeomorphism between the spaces $\mathcal{S}$ and ß.*

Let $\mathcal{D} = C_c^\infty(\mathbf{R})$: the space of C^∞ functions on $\mathbf{R}$ with compact support. Since $\mathcal{D} \subset \mathcal{S}$, Θ has all of the nice properties on $\mathcal{D}$ that it has on $\mathcal{S}$. Let $\mathcal{D}_H = \Theta(\mathcal{D})$. Let $\mathcal{S}'$ be the space of tempered distributions on $\mathcal{S}$, $\mathcal{D}'$ the space of distributions on $\mathcal{D}$, and ß′ and $\mathcal{D}'_H$ the space of continuous linear functionals on ß and $\mathcal{D}_H$ respectively. Note that $\mathcal{D} \subset \mathcal{S} \Rightarrow \mathcal{S}' \subset \mathcal{D}'$.

The correspondence above allows us to give a natural definition of the Weil transform of a distribution. Let $T \in \mathcal{D}'$. Define $\Theta(T) \in \mathcal{D}'_H$ by

$$\Theta(T)(\Theta(f)) = T(f).$$

Because of the preservation of semi-norm structure, this clearly sets up a structure-preserving bijective correspondence between $\mathcal{S}'$ and ß′. Since Θ is an isomorphism between $\mathcal{S}$ and ß, the space ß′ is isomorphic to $\mathcal{S}'$, with the above definition of Θ.

Note that the definition of Θ above extends the usual one on $\mathbf{L}^2(\mathbf{R})$. This is because, if $f \in \mathbf{L}^2(\mathbf{R})$, then, as a distribution, for any $g \in \mathcal{S}$,

$$\begin{aligned} f(g) &= \int f(t)g(t)\,dt \\ &= \left\langle g, \overline{f} \right\rangle \\ &= \left\langle \Theta(g), \overline{\Theta(f)} \right\rangle \\ &= \int\int\int \Theta(f)\Theta(g)\,dx\,dy\,dz \\ &= \Theta(f)\left(\Theta(g)\right). \end{aligned}$$

The spaces $\mathcal{D}'$ and $\mathcal{S}'$ are not only linear spaces, but also modules over the ring generated by C^∞ functions and polynomials in t and $\frac{d}{dt}$. Let us try to set up the same structures for $\mathcal{D}'_H$ and ß′. We can not differentiate functions in ß with respect to y and stay within H_1, so we can not succeed directly. However, in a module, multiplication by an element of the ring is an endomorphism, and we can compute explicitly what these endomorphisms of $\mathcal{D}'$ are on $\mathcal{D}'_H$, hence setting up the analogous structure.

The difficulty is that, over $\mathbf{R}^n$, we have the coincidence that the partial derivatives of a function can be thought of as globally defined functions. Over general manifolds, the derivatives only have pointwise values as a local object. These

values change under changes of coordinates.

If we have a $\mathbf{C}^\infty$ function on $\mathcal{N}$ we can take its partial derivatives, and since $\mathcal{N}$ in the usual coordinate system is just $\mathbf{R}^3$, we get functions corresponding to these partial derivatives. However, these functions will not necessarily be Γ invariant, even if the original function was Γ invariant. In fact, since $\Gamma\backslash\mathcal{N}$ is a compact manifold, we can not hope to cover it by one coordinate patch, so there is no single coordinate system in which to define a global derivative function.

Fortunately, $\Gamma\backslash\mathcal{N}$ has a coordinate system which almost covers the whole space. This allows us to, for example, evaluate integrals of functions just over the unit cube, without having to worry about partitions of unity and different coordinate patches. Also, we have seen that although $\frac{\partial}{\partial y}$ does not define an endomorphism of ß, $(x + \frac{1}{2\pi i}\frac{\partial}{\partial y})$ does, and it corresponds to multiplication by t over $\mathbf{R}$. There is no problem with $\frac{\partial}{\partial x}$ since the change of coordinates from one usual patch to the next does not involve x.

We can multiply functions in $C^\infty(\Gamma\backslash\mathcal{N})$ by each other and by polynomials in D^α for any multi-index $\alpha \in \mathbf{Z}^3_+$. Hence we define these operations for distributions over $\Gamma\backslash\mathcal{N}$ in the usual way, where derivatives are taken in the unit cube coordinate system. Perhaps the only non-obvious part of this definition is that for distributions that come from smooth functions, the differential operators correspond to

the classical ones. This is true for $\frac{\partial}{\partial y}$ because

$$\int_0^1\int_0^1\int_0^1 \frac{\partial F}{\partial y} G\,dx\,dy\,dz + \int_0^1\int_0^1\int_0^1 F\frac{\partial G}{\partial y}\,dx\,dy\,dz$$

$$\begin{aligned} &= \int_0^1\int_0^1\int_0^1 \frac{\partial}{\partial y}(FG)\,dx\,dy\,dz \\ &= \int_0^1\int_0^1 F(x,1,z)G(x,1,z) - F(x,0,z)G(x,0,z)\,dx\,dz \\ &= 0, \end{aligned}$$

and so

$$\int_0^1\int_0^1\int_0^1 \frac{\partial F}{\partial y} G\,dx\,dy\,dz = -\int_0^1\int_0^1\int_0^1 F\frac{\partial G}{\partial y}\,dx\,dy\,dz.$$

A similar argument works for $\frac{\partial}{\partial z}$.

The case for $\frac{\partial}{\partial x}$ is slightly different.

$$\int_0^1\int_0^1\int_0^1 \frac{\partial F}{\partial x} G\,dx\,dy\,dz + \int_0^1\int_0^1\int_0^1 F\frac{\partial G}{\partial x}\,dx\,dy\,dz$$

$$\begin{aligned} &= \int_0^1\int_0^1\int_0^1 \frac{\partial}{\partial x}(FG)\,dx\,dy\,dz \\ &= \int_0^1\int_0^1 F(1,y,z)G(1,y,z) - F(0,y,z)G(0,y,z)\,dy\,dz \\ &= \int_0^1\int_0^1 F(1,y,z)G(1,y,z) - F(1,y,z+y)G(1,y,z+y)\,dy\,dz \\ &= \int_0^1\int_0^1 F(1,y,z)G(1,y,z)\,dy\,dz \\ &\qquad - \int_0^1\int_0^1 F(1,y,z+y)G(1,y,z+y)\,dy\,dz \end{aligned}$$

Since the Jacobian determinant of the transformation

$$(y,z) \mapsto (y, z+y)$$

is 1, the two integrals above are equal and so

$$\int_0^1 \int_0^1 \int_0^1 \frac{\partial F}{\partial y} G \, dx \, dy \, dz = -\int_0^1 \int_0^1 \int_0^1 F \frac{\partial G}{\partial y} \, dx \, dy \, dz.$$

Hence we have the following theorem:

Theorem 1.12 *Distributional derivatives over $\Gamma\backslash\mathcal{N}$ correspond to classical derivatives over $\Gamma\backslash\mathcal{N}$ when the latter are defined.*

In summary, given a distribution in ß we can multiply by polynomials in $\frac{\partial}{\partial x}$, and $(x + \frac{1}{2\pi i}\frac{\partial}{\partial y})$. We can also multiply by doubly-periodic smooth functions on $\mathcal{N}$, which are special types of Γ invariant smooth functions on $\mathcal{N}$. When the distribution corresponds to a function, these operations will correspond to the usual ones.

1.4 Derivatives and Moments

The purpose of this section is to prove theorem 1.13 and its corollaries. These results improve and make more precise some previous statements in the literature [1], [14], and [24].

Theorem 1.13 *Given non-negative integers $m, n \geq 0$, and a distribution T,*

$$\Theta(t^n(\frac{d}{dt})^m T) = (x + \frac{1}{2\pi i}\frac{\partial}{\partial y})^n (\frac{\partial}{\partial x})^m \Theta(T),$$

where all derivatives are in the sense of distributions.

Proof: Let $F \in \text{ß}$, $T \in \mathcal{S}'$, and let $f = \Theta^{-1}(F)$. Then

$$\begin{aligned}
\Theta(t^n(\frac{d}{dt})^m T)(F) &= (t^n(\frac{d}{dt})^m T)(f) \\
&= T(t^n(-\frac{d}{dt})^m f) \\
&= \Theta(T)(\Theta(t^n(-\frac{d}{dt})^m f)) \\
&= \Theta(T)((x + \frac{1}{2\pi i}\frac{\partial}{\partial y})^n(-\frac{\partial}{\partial x})^m F) \\
&= ((x + \frac{1}{2\pi i}\frac{\partial}{\partial y})^n(\frac{\partial}{\partial x})^m \Theta(T))(F).
\end{aligned}$$

□

Definition: Let $s \in \mathbf{L}^2(\mathbf{R})$. The k^{th} time moment of s, in the sense of Gabor [18], is defined by

$$\int_{-\infty}^{\infty} t^k |s(t)|^2 \, dt.$$

The k^{th} frequency moment of s, in the sense of Gabor [18], is defined by

$$\int_{-\infty}^{\infty} f^k |\hat{s}(f)|^2 \, df,$$

where $\hat{s}$ denotes the Fourier transform of s. These quantities will be referred to as the time and frequency moments of s. □

Note that, by the rule for derivatives and Fourier transforms, the $2k^{\text{th}}$ moment of s is equal to

$$\frac{1}{2\pi i}^k \int_{-\infty}^{\infty} |\widehat{s^{(k)}}(f)|^2 \, df.$$

Hence, if $t^k f(t) \in \mathbf{L}^2(\mathbf{R})$, then the $2k^{\text{th}}$ time moment of f is finite. If $f^{(k)}(t) \in \mathbf{L}^2(\mathbf{R})$, then the $2k^{\text{th}}$ frequency moment of f is finite. We can now state a corollary of theorem 1.13.

Corollary 1.14 *If $\Theta(f) \in \mathbf{C}^k$, then the first $2k$ time and frequency moments of f are finite. If the first $2k$ time and frequency moments of f are finite, then for any non-negative integers $m, n \geq 0$, with $n+m \leq k$, $\frac{\partial^{n+m}}{\partial x^m \partial y^n}\Theta(f)$ is an $\mathbf{L}^2$ function on $\Gamma\backslash\mathcal{N}$.*

Proof: As a function on $\mathcal{N}$, x is C^∞, so multiplication by powers of x preserves any amount of smoothness. Hence, if F is C^k, then $\left(x + \frac{1}{2\pi i}\frac{\partial}{\partial y}\right)^j F$ is continuous for $j \leq k$. Therefore, if $\Theta(f) \in C^k$, then for $j \leq k$, the functions $\left(\frac{\partial}{\partial x}\right)^j F$, and $\left(x + \frac{1}{2\pi i}\frac{\partial}{\partial y}\right)^j F$ are certainly $\mathbf{L}^2$ functions on $\Gamma\backslash\mathcal{N}$, and consequently $t^j f(t)$ and $f^{(j)}(t)$ are in $\mathbf{L}^2(\mathbf{R})$. Hence $\int_{\mathbf{R}} t^{2j}|f(t)|^2\,dt < \infty$, and $\int_{\mathbf{R}} \nu^{2j}|\hat{f}(\nu)|^2\,d\nu < \infty$, for $j \leq k$. The result for odd moments follows by interpolation.

Theorem 1.13 shows that if the first $2k$ time and frequency moments of f are finite, then for $j \leq k$, the functions $\left(\frac{\partial}{\partial x}\right)^j F$, and $\left(x + \frac{1}{2\pi i}\frac{\partial}{\partial y}\right)^j F$ are $\mathbf{L}^2$ functions on $\Gamma\backslash\mathcal{N}$. By an induction argument analogous to the one in the proof of lemma 1.10, it follows that $\frac{\partial^{n+m}}{\partial x^m \partial y^n}\Theta(f)$ is an $\mathbf{L}^2$ function on $\Gamma\backslash\mathcal{N}$, for $n+m \leq k$.

□

2 Structure of C^k Functions On $\Gamma\backslash\mathcal{N}$

2.1 Weyl-Heisenberg Wavelets and the Need to Divide Functions in H_1

This section reviews how the problem of reconstructing a function from its Weyl-Heisenberg wavelet coefficients leads naturally to the problem of taking quotients of functions in H_1.

Given $f, g \in \mathbf{L}^2(\mathbf{R})$, define the cross-ambiguity function of f and g by

$$A_{(f,g)}(\tau,\nu) = \int_{-\infty}^{\infty} f(t)\,\overline{g(t-\tau)}\,e^{-2\pi i\nu t}\,dt, \qquad \tau,\nu \in \mathbf{R}. \tag{13}$$

The reader is referred to the references [9], [11] and [26] for discussions of radar ambiguity functions from an engineering point of view.

The doubly-indexed sequence of the values of the cross-ambiguity function restricted to integer τ and ν, is alternately known as the sliding windowed Fourier coefficients of f with window g, or the Weyl-Heisenberg wavelet coefficients of f with respect to the wavelet g.

Now

$$\begin{aligned} A_{(f,g)}(m,n) &= \int_{-\infty}^{\infty} f(t)\,\overline{g(t-m)}\,e^{-2\pi int}\,dt \\ &= \left\langle f(t), g(t-m)e^{2\pi int}\right\rangle \\ &= \left\langle \Theta(f(t)), \Theta(g(t-m)e^{2\pi int})\right\rangle, \end{aligned}$$

because Θ is a unitary operator from $\mathbf{L}^2(\mathbf{R})$ to H_1. But

$$\begin{aligned}\Theta(g(t-m)e^{2\pi int}) &= \sum_{a\in\mathbf{Z}} g(x+a-m)e^{2\pi in(x+a)}e^{2\pi iay} \\ &= e^{2\pi i(nx+my)}\sum_{a'\in\mathbf{Z}} g(x+a')e^{2\pi ia'y} \\ &= e^{2\pi i(nx+my)}\Theta(g(t)).\end{aligned}$$

It follows that

$$\begin{aligned}A_{(f,g)}(m,n) &= \left\langle \Theta(f), \Theta(g)e^{2\pi i(nx+my)} \right\rangle \\ &= \left\langle \Theta(f)\,\overline{\Theta(g)}, e^{2\pi i(nx+my)} \right\rangle.\end{aligned}$$

So the sliding windowed Fourier coefficients of f with respect to g are simply the Fourier series coefficients of the doubly-periodic function $\Theta(f)\,\overline{\Theta(g)}$. Hence, in order to reconstruct f from its sliding windowed Fourier coefficients, it ought to be sufficient to form the doubly-periodic function

$$p(x,y) = \sum_{m,n} A_{(f,g)}(m,n)e^{2\pi i(nx+my)},$$

then compute $\Theta(f)(x,y) = p(x,y)/\,\overline{\Theta(g)}$, and finally find f as $\Theta^{-1}(\Theta(f))$.

The above process will not work in general since the series for $p(x,y)$ will not necessarily converge. In such cases we may conclude that g is not a suitable windowing function with which to analyze f, since in this case the series $A_{(f,g)}(m,n)$ will not be square summable.

When g is nice enough so that the series does converge, the above implies that $p(x,y)$ will be divisible by $\overline{\Theta(g)}$, at least in the sense that there will be an $\mathbf{L}^2$

function F such that $p = F\,\overline{\Theta(g)}$. However, in applied situations where there is noise, or computational error, we may have slight perturbations of the coefficients for $p(x,y)$. Unfortunately, these perturbations may cause a situation where the approximate $p(x,y)$ is no longer divisible by $\overline{\Theta(g)}$.

The condition that the set of functions $\{g(t-m)e^{2\pi int}\}_{(m,n)}$ be a frame for $\mathbf{L}^2(\mathbf{R})$, (see [14]), is the usual way to handle the problem of when a function can be stably reconstructed from coefficients arising from inner-products with a fixed family of functions. However, as discussed in [1], Balian's theorem [7], shows that all g which give rise to frames, in the Weyl-Heisenberg setting, have undesirable properties.

It is therefore interesting to study when division of functions in H_1 can be carried out. In the case where g does not give rise to a frame, one would like to know what conditions must be placed on f for division to be carried out stably.

Intuitively it is clear that functions can be divided to yield a smooth quotient if and only if they have 'the same' local behavior at their zeros. Actually we want, in some sense, that the numerator have 'the same or flatter' behavior at its zeros, as compared with the denominator. For these reasons, one of the areas of study in this work will be the local structure of C^k two dimensional functions.

2.2 Local Structure of Smooth Functions in H_1, and the Division Problem for Such Functions

Here we define functions of smooth growth and a type of non-degeneracy for such functions. This non-degeneracy generalizes the condition of non-zero Jacobian determinant. We show that the former is a natural condition for smoothness of quotients of functions.

A function F has smooth growth of order k at a point p in its domain if F is C^k at p, all partial derivatives of F of order less than k are zero at p, and some derivative of F of order k is non-zero at p. F has smooth growth at p if it has smooth growth of order k at p, for some k. In other words, a function of smooth growth at p is a function which is in some non-zero jet at p.

Let $F(x,y)$ and $G(x,y)$ be complex valued C^k functions, each with an isolated zero at $(0,0)$, and such that F and G have smooth growth of order k at $(0,0)$. This means that $F(x,y) = f(x,y) + R_f(x,y)$, where f is a non-zero homogeneous polynomial of degree k, and $\lim_{(x,y)\to(0,0)} \frac{R_f(x,y)}{|(x,y)|^k} = 0$. Similarly $G(x,y) = g(x,y) + R_g(x,y)$. Now a homogeneous polynomial of two variables is essentially a polynomial of one variable. Hence we can factor these homogeneous terms into products of linear factors, (ie. factors of the form $c_1x + c_2y$ for some complex c_1 and c_2).

For example, suppose that $f(x,y) = xy^2 + 2x^2y + x^3$. If we formally divide by

y^3, and write $z = \frac{x}{y}$, then we have $z^3+2z^2+z = z(z+1)^2$. Hence $xy^2+2x^2y+x^3 = x(x+y)^2$. So we may write

$$f(x,y) = (x+0y)(x+y)(x+y).$$

The linear factors of the form $c_1x + c_2y$ are thought of above as functions from $\mathbf{R}^2$ to $\mathbf{C}$. We may also think of them as functions from $\mathbf{R}^2$ to itself; as linear transformations. As such, they have determinants. If $c_1 = a + ci$, and $c_2 = b + di$, then the determinant of the linear factor $c_1x + c_2y$ is defined as $ad - bc$.

Notation: When F has smooth growth at some point p, we will write the above as $F = f_p + R_{f,p}$. Using this notation for F, G, f_p and g_p, and the following definition, we can state the main result of this section.

Definition: A function F of smooth growth at an isolated zero p will be said to be of *non-degenerate smooth growth at p* if each of the linear factors in f_p have non-zero determinant. Otherwise F will be said to be of *degenerate smooth growth at p.* □

Theorem 2.1 *If $F(x,y)$ and $G(x,y)$ have non-degenerate smooth growth at their zeros, all of which are isolated, then the pointwise quotient function $F(x,y)/G(x,y)$ is a continuous function if and only if for each p, if $G(p) = 0$, then $F(p) = 0$, and at each such p, either g_p has lower degree than f_p, or they have equal degree and are products of exactly the same linear factors, counted with multiplicity, and up to a multiple of each factor by a complex constant.*

Proof: If G has a zero where F does not, then the quotient is not defined, and hence not a continuous function. So suppose that F has a zero at each zero of G.

Now, let p be a common zero of F and G. Suppose, without loss of generality, that $p = 0$, and let $f = f_p = \prod_{k=1}^{d_f} (\alpha_{f,k}x + \beta_{f,k}y)$, and $g = g_p = \prod_{k=1}^{d_g} (\alpha_{g,k}x + \beta_{g,k}y)$ be the factorizations of f and g into linear factors described above.

Clearly F/G is continuous except possibly at the zeros of G. At such a zero, say at the point p, the continuity of the quotient is determined by limits.

By the non-degeneracy of F and G, the linear factors $\alpha x + \beta y$ in the above decomposition have non-zero determinant. Since the image of the unit circle under such a map is an ellipse that does not hit the origin, there are two positive real constants, $I_{\alpha,\beta}$, and $S_{\alpha,\beta}$, such that

$$I_a|(x,y)| \leq |\alpha x + \beta y| \leq S_a|(x,y)|.$$

Now, we can rewrite F/G, in a neighborhood of 0, as

$$\frac{f(x,y) + |(x,y)|^{d_f}\tilde{R}_f(x,y)}{g(x,y) + |(x,y)|^{d_g}\tilde{R}_g(x,y)},$$

where $\tilde{R}_f$ is continuous, $\lim_{(x,y)\to 0} \tilde{R}_f(x,y) = 0$, and similarly for $\tilde{R}_g$.

Using the two previous facts, there are real constants c_1, c_2, such that

$$\lim_{(x,y)\to 0} c_1|(x,y)|^{d_f - d_g} \leq \lim_{(x,y)\to 0} \frac{f(x,y) + |(x,y)|^{d_f}\tilde{R}_f(x,y)}{g(x,y) + |(x,y)|^{d_g}\tilde{R}_g(x,y)} \leq c_2|(x,y)|^{d_f - d_g}.$$

Therefore, if $d_f > d_g$, the limit exists and is zero. If $d_f < d_g$, the limit is infinite. Hence the only remaining case is when $d_f = d_g$.

The above point of view also makes it clear that the limit depends only on f and g, since $\tilde{R} \to 0$.

By multiplying by complex constants, which does not alter continuity, we may assume without loss of generality that $\alpha_{f,k} = \alpha_{g,k} = 1$, for all relevant k.

Consider the quotient

$$e(x,y) = \frac{f}{g} = \frac{\prod_{k=1}^{d_f}(x + \beta_{f,k}y)}{\prod_{k=1}^{d_g}(x + \beta_{g,k}y)}.$$

If each of the linear factors in g occurs as a linear factor in f, up to multiplication by a complex constant, and counting multiplicities, then the quotient is continuous, since we may just cancel the factors.

Conversely, suppose that $e(x,y)$ is continuous at $(0,0)$. Consider

$$e_\lambda(t) = e(\lambda t, t).$$

We have that

$$\begin{aligned} e_\lambda(t) &= \frac{\prod_{k=1}^{d_f}(\lambda t + \beta_{f,k}t)}{\prod_{k=1}^{d_g}(\lambda t + \beta_{g,k}t)} \\ &= \frac{t^{d_f}\prod_{k=1}^{d_f}(\lambda + \beta_{f,k})}{t^{d_g}\prod_{k=1}^{d_g}(\lambda + \beta_{g,k})} \\ &= \frac{\prod_{k=1}^{d_f}(\lambda + \beta_{f,k})}{\prod_{k=1}^{d_g}(\lambda + \beta_{g,k})}. \end{aligned}$$

This last equation shows that e_λ does not depend on t, and so it must be a constant. But, since e is continuous at $(0,0)$, e_λ is continuous at zero. Hence $\lim_{t\to 0} e_\lambda(t) = e(0,0)$. Therefore, all real λ satisfy the equation

$$\prod_{k=1}^{d_f}(\lambda + \beta_{f,k}) = e(0,0)\prod_{k=1}^{d_g}(\lambda + \beta_{g,k}).$$

Since the above gives a polynomial with a continuum of roots, it must be the zero polynomial, and the result follows. □

2.3 Winding Number Data

Here the winding number data is defined for smooth functions in H_1.

Let $X = \{(x,y)|0 \leq x,y \leq 1\}$. Given a smooth function $F \in H_1$, let $S = X - F^{-1}(0)$. Then we may think of F as a function from S to $\mathbf{C}^* = \mathbf{C} - \{0\}$. Applying the homology functor we get a group homomorphism

$$\mathbf{H}_1(F) : \mathbf{H}_1(S) \to \mathbf{H}_1(\mathbf{C}^*).$$

Now $\mathbf{H}_1(\mathbf{C}^*) \simeq \mathbf{Z}$, and in simple cases $\mathbf{H}_1(S)$ is determined by simple closed curves that wind around components of zeros of F. When there are a finite number of components of $F^{-1}(0)$, $\mathbf{H}_1(S)$ is the free Abelian group generated by these components (actually generated by simple closed curves that wind once around exactly one of these components, and don't wind around any of the others). In general, it is impled by theorem 7 of chapter 4 of [30], that $\mathbf{H}_1(S)$ is a direct limit of such simpler cases, but this is still a group generated by simple closed curves that surround sets of zeros of F (it just isn't possible to surround all components separately).

In fact, this group homomorphism corresponds to a co-homology class in $\mathbf{H}^1(S)$. To see this, partition the universal covering space of $\mathbf{C}^*$ into consecu-

tively numbered sheets (say by jumping to the next integer when passing a line in the inverse image of the line $0 \leq x < \infty,\ y = 0$). For any given singular simplex $\gamma : [0,1] \to S$, define

$$f(\gamma) = \text{sheet}\#(\tilde{F}_\gamma(1))) - \text{sheet}\#(\tilde{F}_\gamma(0))),$$

where $\tilde{F}_\gamma$ is the unique lifting of $F(\gamma(t))$ to the universal covering space. Then f is a co-cycle and hence corresponds to a unique co-homology class.

When there is an isolated, simply connected component, C, of the zero set of F, we will speak of the *winding number of F at C*, or the *winding number of F around C*. This is the unique integer N such that $F(\gamma_c)$ is homotopic to the curve $t \mapsto e^{2\pi i N t}$, where γ_c is a curve which winds once around C, and does not wind around any other zeros of F.

When F has a finite number of zeros, the winding number data of F consists of these finite number of points together with the winding number at each point. It is a fact that the total of these integers is -1. To see this, we may shift F with a diffeomorphic change of coordinates via translation in the Heisenberg group, so that none of the zeros are on the boundary of the unit square. This, being a diffeomorphism, will not change the topology of F, and hence will preserve winding numbers. Then, we simply compute the winding number of the F-image of the boundary of the unit square. It is -1, by the quasi-periodicity relations. See the proof of the zero theorem in [1] for a detailed proof of this.

2.4 Winding Numbers and Smooth Growth

For functions of non-degenerate smooth growth, we can easily say what the winding numbers of the function are, at its various zeros.

Lemma 2.2 *If F has winding number n_F about an isolated zero of F at p, and G has winding number n_G at an isolated zero of G at p, the pointwise product function $F(x,y)G(x,y)$ has winding number $n_F + n_G$, at its isolated zero at p.*

Proof: Let γ be a smooth curve that winds around p exactly once, and which winds around no other zeros of F or G. Let $f(t) = F(\gamma(t))$, and $g(t) = G(\gamma(t))$. Then the index of f is n_F, the index of g is n_G, and we must show that the index of the pointwise product $f(t)g(t)$ is $n_F + n_G$. This is clearly true for the functions $e^{2\pi int}$, and if it is true for f_1 and g_1, and if $f_1 \sim f_2$, and $g_1 \sim g_2$ are homotopies in the punctured complex plane, then the pointwise product of these two homotopies as functions on the unit square, gives a homotopy between the pointwise product curves $f_1(t)g_1(t)$ and $f_2(t)g_2(t)$ in the punctured complex plane. Hence, since all curves that avoid the origin are homotopic in the punctured complex plane to a unique curve of the form $e^{2\pi int}$, determined by its winding number, the result is proved. □

Now let $F(x,y)$ and $G(x,y)$ be complex valued C^k functions of smooth growth, as in section 2.2. This means that, using the same notation as in that section,

we can write F and G as sums of a non-zero smooth, polynomial part, and a remainder that has a zero of higher degree than the polynomial. Again, we will take the homogeneous parts of the polynomials to be factored into linear factors.

It is easy, using Jacobian determinants, to decide what the winding number is for each of these linear factors: it is $+1$, -1, or 0 according to the sign of the Jacobian determinant. Since winding numbers add under function multiplication, we can determine what the winding number of each of the homogeneous terms is.

If a smooth function has an isolated zero at some point p, and if its lowest degree non-zero terms sum to a function with a *non-isolated* zero at p, then these terms can not possibly describe the local topology of the function at p. However, if the lowest degree terms sum to a function with an isolated zero at p, then theorem 2.3 below shows that these terms determine the local homology of the function at p.

Theorem 2.3 *Using the notation of previous sections and above, if F has non-degenerate smooth growth at p, then the winding number of F is equal to the winding number of the homogeneous part of f_p of lowest degree. In other words, if F has non-degenerate smooth growth at an isolated zero, then we can read off the winding number of F from its partial Taylor expansion.*

Proof: Assume, without loss of generality, that $p = (0, 0)$. The term of lowest degree dominates near zero, and it also has an isolated zero, by the non-degeneracy

of F. Hence, a small circle about $(0,0)$ has an image under F which winds the same number of times about $(0,0)$ as does its image under f. □

2.5 A Product Theorem for Smooth Growth Functions

In this section we prove that a function in H_1 which has smooth growth at each of its zeros can be written as a product of certain fixed functions, and a smooth doubly periodic function with no zeros.

Let $\alpha = a + ic$, and $\beta = b + id$, be such that $ad - bc > 0$. Let $D_{\alpha,\beta}$ be a $\mathbf{C}^\infty$ diffeomorphism of the unit square that fixes the boundary and the point $(\frac{1}{2}, \frac{1}{2})$, and such that the derivative

$$dD_{\alpha,\beta}(\frac{1}{2}, \frac{1}{2}) = \begin{pmatrix} a & b \\ c & d \end{pmatrix}.$$

That such a diffeomorphism exists can be seen geometrically by looking at the map which is equal to the linear map, applied to $(x - \frac{1}{2}, y - \frac{1}{2})$, in a small square neighborhood of $(\frac{1}{2}, \frac{1}{2})$. The image of the boundary of this small square is an orientation preserving linear image of the boundary of the small square. Hence it can be smoothly deformed into the boundary of the unit square, in a way which exactly exhausts the remainder of the unit square.

Note that if $G \in H_1$, and if we set $(u, v) = D_{\alpha,\beta}(x, y)$, then $G(u, v, z) \in H_1$, since D fixes the boundary of the unit square, and hence does not disturb the Heisenberg periodicity relations.

Let $g(t) = e^{-\pi t^2}$. Let $G = \Theta(g)$. Then it is known, see [5], that $G(\frac{1}{2}, \frac{1}{2}, 0) = 0$, that this is the only zero of G in the unit square, and that $dG(\frac{1}{2}, \frac{1}{2}) = c \begin{pmatrix} -1 & 0 \\ 0 & 1 \end{pmatrix}$, where c is some real positive constant. Note that, in [5] the derivative of the function that is denoted by ψ, is given as $\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$, but a slightly different function is being considered here.

Define

$$G^{(\alpha,\beta)}(x, y, z) = \frac{1}{c} G(D_{(\alpha,\beta)}(x, y), z).$$

$$G^{(\alpha,\beta)}_{(x_0,y_0)}(x, y, z) = e^{2\pi i x_0(y_0 - \frac{1}{2})} G^{(\alpha,\beta)}(x - x_0 + \frac{1}{2}, y - y_0 + \frac{1}{2}, z - x(y_0 - \frac{1}{2})).$$

$$G^{(\alpha,\beta)}_{(x_0,y_0)}(x, y, z) = e^{2\pi i x_0(y_0 - \frac{1}{2})} G^{(\alpha,\beta)}(x - x_0 + \frac{1}{2}, y - y_0 + \frac{1}{2}, z - x(y_0 - \frac{1}{2})).$$

Lemma 2.4 $G^{(\alpha,\beta)}_{(x_0,y_0)}(x_0, y_0, 0) = 0$, *and*

$$dG^{(\alpha,\beta)}_{(x_0,y_0)}(x_0, y_0) = \begin{pmatrix} -a & -b \\ c & d \end{pmatrix}.$$

(x_0, y_0) *is the only zero of* $G^{(\alpha,\beta)}_{(x_0,y_0)}(x, y)$ *in the unit square.*

Proof: Since $G^{(\alpha,\beta)}_{(x_0,y_0)}$ is a Heisenberg shifted, diffeo-coordinate-changed version of G, the statements about the location and uniqueness of its zero are trivially true.

We will now switch to two variable notation for convenience.

The chain rule implies that

$$dG^{(\alpha,\beta)}(\frac{1}{2}, \frac{1}{2}) = \begin{pmatrix} -a & -b \\ c & d \end{pmatrix}.$$

But $G^{(\alpha,\beta)}_{(x_0,y_0)}(x,y) = e^{2\pi i x_0(y_0-\frac{1}{2})}e^{-2\pi i x(y_0-\frac{1}{2})}G^{(\alpha,\beta)}(x-x_0+\frac{1}{2}, y-y_0+\frac{1}{2})$. Therefore

$$dG^{(\alpha,\beta)}_{(x_0,y_0)}(x,y) =$$

$$e^{2\pi i x_0(y_0-\frac{1}{2})}\left(d\left[e^{-2\pi i x(y_0-\frac{1}{2})}\right]G^{(\alpha,\beta)}(x-x_0+\frac{1}{2}, y-y_0+\frac{1}{2})\right.$$

$$\left. + e^{-2\pi i x(y_0-\frac{1}{2})}dG^{(\alpha,\beta)}(x-x_0+\frac{1}{2}, y-y_0+\frac{1}{2})\right).$$

It follows, because $G^{(\alpha,\beta)}(\frac{1}{2},\frac{1}{2}) = 0$, that

$$\begin{aligned} dG^{(\alpha,\beta)}_{(x_0,y_0)}(x_0,y_0) &= e^{2\pi i x_0(y_0-\frac{1}{2})}e^{-2\pi i x_0(y_0-\frac{1}{2})}dG^{(\alpha,\beta)}(\frac{1}{2},\frac{1}{2}) \\ &= \begin{pmatrix} -a & -b \\ c & d \end{pmatrix}. \end{aligned}$$

□

For the proof of the next theorem, we will need the following definition and facts.

Definition: We define the space H_k, for $k \in \mathbf{Z}$, to be the set of all $F \in \mathbf{L}^2(\Gamma\backslash\mathcal{N})$ which satisfy the equation:

$$F(x,y,z) = e^{2\pi i k z}F(x,y,0). \tag{14}$$

□

In particular, H_0 is the space of $\mathbf{L}^2$ doubly periodic functions, and this definition of H_1 corresponds to the previous definition.

As in section 1.2, continuous functions in H_k correspond to continuous functions on the unit square which satisfy:

$$F(x,0) = F(x,1) \tag{15}$$

$$F(0,y) = e^{2\pi iky}F(1,y). \tag{16}$$

We can therefore switch between two variable and three variable notation, by equation (14).

It is easy to see that if $F \in H_k$, and $G \in H_l$, then the pointwise product function $F(x,y,z)G(x,y,z) \in H_{k+l}$. Also, if $F \in H_k$, then $\overline{F} \in H_{-k}$.

Finally, if we have a linear function $(x,y) \mapsto \alpha x + \beta y$ with negative determinant, then we can define

$$G^{(\alpha,\beta)}_{(x_0,y_0)}(x,y,z) = \overline{G^{(\overline{\alpha},\overline{\beta})}_{(x_0,y_0)}(x,y,z)}.$$

This function will be well defined, and in H_{-1}, since the map $(x,y) \mapsto \overline{\alpha}x + \overline{\beta}y$ will obviously have positive determinant. Also, lemma 2.4 will hold for this function as well.

Theorem 2.5 *Let $F \in H_1$ have non-degenerate smooth growth at each of its zeros. Then there exists a smooth function $G_F(x,y) \in H_1$, and a doubly-periodic function $P(x,y)$ with no zeros in the unit square, such that*
$F(x,y) = P(x,y)G_F(x,y)$. G_F can be taken to be a product of functions of the form $G^{(\alpha,\beta)}_{(x_0,y_0)}$.

Proof: Simply construct $G_F(x,y)$ as a product of functions of the form $G^{(\alpha,\beta)}_{(x_0,y_0)}$, to have the same lowest order finite Taylor expansion as F, up to multiplication by a complex constant, at each of the zeros of F.

To see that this can be done, first note that all of the zeros of a function of non-degenerate smooth growth are isolated, because the lowest degree term dominates in a neighborhood of each zero, and this term has an isolated zero. Hence F has a finite number of zeros, by compactness.

Now, we can find a G with any desired linear first order term, and hence we can find a product of G's with any desired homogeneous k-th order term, for any k.

So let F has its zeros at the points $p_1 = (x_1, y_1), \ldots, p_n = (x_n, y_n)$. Let $h_i = h_{f,p_i} = \prod_{k=1}^{d_i} (\alpha_{i,k}x + \beta_{i,k}y)$. Then we can take

$$G_F(x,y) = \prod_{i=1}^{n} \prod_{k=1}^{d_i} G^{(\alpha_{i,k},\beta_{i,k})}_{(x_i,y_i)}(x,y).$$

It follows from theorem 2.3, and the discussion at the end of section 2.3, that $G_F \in H_1$. This is because the winding number at p_i counts the number of k for which $G^{(\alpha_{i,k},\beta_{i,k})}_{(x_i,y_i)} \in H_{-1}$ minus the number of k for which $G^{(\alpha_{i,k},\beta_{i,k})}_{(x_i,y_i)} \in H_1$. This total must be -1, so the full product is in H_1.

Once G_F is constructed the result follows from theorem 2.1 and the fact that the zeros of $G^{(\alpha,\beta)}_{(x_0,y_0)}$'s are unique in the unit square. □

2.6 Homotopies and H_1

In this section we will show that a smooth function in H_1 is nothing more than a free homotopy between a given curve in $\mathbf{C}$ and a certain related curve. This will be applied to show how one can construct explicit continuous functions in H_1 with various desired properties, using topological and geometric intuition.

Consider the paths in the complex plane given by an $F \in H_1$ for fixed x, where again, as in section 1.2, $F \in H_1$ is taken to mean that F is a continuous function on the unit square which satisfies equations (5) and (6). write $\gamma_x(y) = F(x, y)$. Then equation (5) says that each γ_x is a closed curve, and (6) says that γ_1 is a specific curve related to γ_0.

Important Fact: The statement that F is continuous is precisely the definition of the statement that F is a free *homotopy* of close curves between γ_0 and γ_1.

One way of looking at this is that a way of specifying a function on $[0,1] \times [0,1]$ is to give a family of functions defined on $[0,1]$, with the family parameterized by the interval $[0,1]$. It is important to note that if such a family is given as a homotopy between (continuous) curves, then by definition the resulting function will be continuous as a function of two variables.

It was proved in [5] that a continuous function in H_1 must have a zero in the unit square. The above fact yields the following one-sentence proof of this

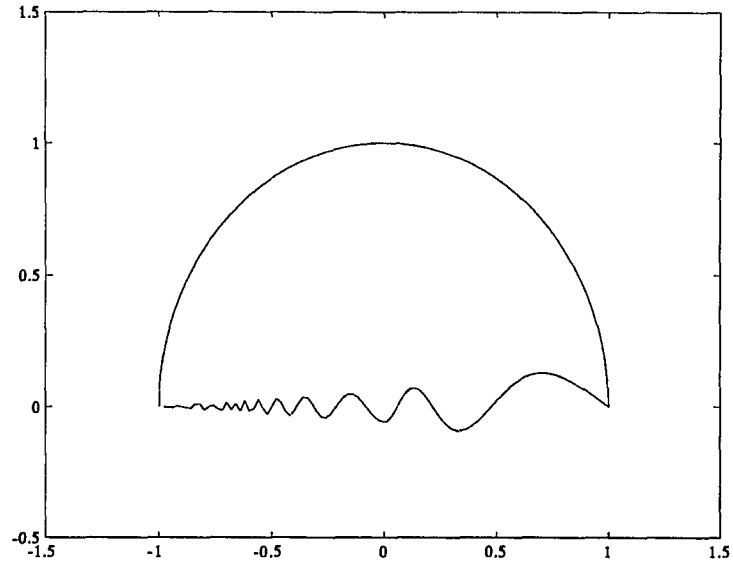
theorem: If F is continuous and never zero then it is a homotopy between a curve with winding number n and a curve with winding number $n-1$, but in the plane with the origin removed, curves with different winding numbers are, by definition, not homotopic, so any homotopy between them in the plane, must pass through the origin.

Generally, since we are free to perform the homotopy in any way we wish as long as the starting and ending curves are as required, quite general sets of zeros, with appropriate arithmetic conditions on winding numbers, can be obtained. An example which will be made precise shortly shows that it is possible to obtain an infinite set of zeros, with each having a non-zero winding number: put the graph of $x\sin(\frac{1}{x})$ into a piece of a circle, as in figure 1 and then 'glide it' through the origin, with the zero crossings corresponding to the points where the graph hits the x-axis.

Each of the countable number of zeros will introduce a winding number of absolute value 1, with alternating signs, except at the zero which is a limit point of other zeros. The winding number around this non-isolated zero is not well defined since there are no curves which surround this zero and no other zeros.

The detailed construction underlying this example uses two basic topological facts. The first is that the straight-line homotopy always exists between two curves in $\mathbf{R}^n$, and is a homotopy. The second is that homotopies form a psuedogroup. That is, if we are given a homotopy $H_s(t)$ from a curve $\gamma_0(t) = H_0(t)$ to a curve

Figure 1: Graph of g_1: $x\sin(\frac{1}{x})$ in a semi-circle



$\gamma_1 = H_1(t)$, and we are given a homotopy $G_s(t)$ from the curve $\gamma_1 = H_1(t) = G_0(t)$ to the curve $\gamma_2 = G_1(t)$, then the following will be a homotopy from γ_0 to γ_2.

$$F_s(t) = \begin{cases} H_{2s}(t) & \text{if } s \in [0, \frac{1}{2}] \\ G_{2s-1}(t) & \text{otherwise} \end{cases}$$

This result obviously generalizes to a result about gluing finitely many homotopies together, and is just a special case of the usual lemma about getting a continuous function by gluing together continuous functions which agree in the intersection of their domains. It is part of the proof that homotopy is an equivalence relation on curves.

The two basic facts will be used to see that the following defines a homotopy, and hence a continuous function on the unit square. Define $g_0(t) = e^{2\pi it}$. Define,

for $t \in [\frac{1}{2}, 1]$,

$$\tau = \begin{cases} \frac{16t-8}{22\pi} & \frac{1}{2} \leq t \leq \frac{3}{4} \\ \frac{28t-17}{22\pi} & \text{otherwise} \end{cases}$$

This is a monotonic function which takes $\frac{1}{2}$ to 0, 1 to $\frac{1}{2\pi}$, and $\frac{3}{4}$ to $\frac{1}{5.5\pi}$. Define

$$g_1(t) = \begin{cases} e^{2\pi i t} & \text{for } t \in [0, \frac{1}{2}] \\ 4(t - \frac{3}{4}) + i\tau \sin(\frac{1}{\tau}) & \text{otherwise} \end{cases} .$$

Define $g_2(t) = 2 + g_1(t)$. Finally define $g_3(t) = 1$. See figure 1 for a graph of the curve g_1. The graph is only approximate, and looses detail quickly at the bottom left, since it is impossible to sample the oscillations of $\sin(\frac{1}{x})$ fast enough, as $x \to 0$.

Define $F(s,t) = F_s(t)$ to be the homotopy which is the straight-line homotopy from g_0 to g_1 followed by the straight-line homotopy from g_1 to g_2 followed by the straight-line homotopy from g_2 to g_3. This defines a continuous function on the unit square because the g_i's are all continuous curves, and hence F is just a homotopy between continuous curves. Moreover, it follows by construction that $F \in H_1$.

To understand the zero set of F, one uses the mathematical fact that if two points lie outside some closed sector with angle greater than π, then the straight line-segment between them can not hit the origin.

Specifically, g_0 has no zeros, and g_1 has no zeros. $g_0(t) = g_1(t)$ for $t \in [0, \frac{1}{2}]$, so that there can be no zeros introduced for the 'lower half' of F here, and the other part can be divided into two quarters, each with g_0 and g_1 outside some definite

sector with angle greater than π, so that the straight-line deformation between them does not cross the origin.

g_2 is just a translate of g_1, and hence the straight-line homotopy between them is just a parameterized translation of the curves. The curves cross the x-axis exactly where the graph of $\tau \sin(\frac{1}{\tau})$ does, and hence there are a countable number of zeros of F which are introduced by this translation.

Finally, g_2 and g_3 are both well to the right of the y-axis so their straight-line can not meet the origin.

The only thing left to do is to understand the winding numbers at these 'zero-crossings'. The following calculation shows that the Jacobian of F is not zero at the isolated zeros of F, so that F is a diffeomorphism in a neighborhood of each of these zeros.

In a neighborhood of each of the isolated zeros,

$$F(x,y) = 4y - 3 + 3x - 1 + i\frac{16y-8}{22\pi}\sin(\frac{22\pi}{16y-8}).$$

Hence

$$DF|_{(x,y)} = \begin{pmatrix} 3 & 4 \\ 0 & \frac{16}{22\pi}\left(\sin(\frac{1}{\tau}) - \frac{1}{\tau}\cos(\frac{1}{\tau})\right) \end{pmatrix},$$

where

$$\tau = \frac{16y-8}{22\pi}.$$

Hence, at any point where the Jacobian of F is zero, we have

$$\tan(\frac{1}{\tau}) = \frac{1}{\tau}.$$

But at the isolated zeros of F we have $\sin(\frac{1}{\tau}) = 0$, and hence $\tan(\frac{1}{\tau}) = 0 \neq \frac{1}{\tau}$. Hence F is a diffeomorphism in a neighborhood of each of its isolated zeros.

Finally recall that the winding number around a zero or zero set comes from looking at the winding number of the F image of a path which winds around this set with winding number 1, so that around a zero where F is a local diffeomorphism, F can not have a winding number other than ± 1, because a diffeomorphism preserves or negates winding numbers. (One way to see this is that, 'in the small', a diffeomorphism sends parameterized circles centered at the zero to parameterized ellipses centered at zero, (i.e. its derivative has this property), so that it must send small parameterized circles to something very nearly a parameterized ellipse. But anything which is arbitrarily close to a curve with winding number ± 1 must have the same winding number, because something which varies continuously can not jump by an integer value. Winding number -1 comes from orientation reversal). In fact, because $\cos(\frac{1}{\tau})$ is alternately 1 and -1 at the zeros of $\sin(\frac{1}{\tau})$, DF is alternately orientation preserving and orientation reversing, and so the winding numbers are alternately 1 and -1.

3 Computations

3.1 Discrete Computation of Winding Number Data

Let $\tilde{\mathbf{C}}^*$ be the universal covering space of $\mathbf{C}^* = \mathbf{C} - \{0\}$. We will take $re^{i\theta}$ to be coordinates for $\tilde{\mathbf{C}}^*$, where we *do not* identify θ with $\theta + 2\pi n$.

By the discussion in section 2.3, winding numbers can be computed by finding a continuous phase-function for certain curves which wind around zeros. This is sometimes referred to as *unwinding the phase* of these curves. In other words, if we have $\gamma_0(t)$, a curve which winds once around an isolated zero of a function F, we can write $\gamma(t) = F(\gamma_0(t)) = r(t)e^{i\phi_0(t)}$. We want to find a lifting of γ to $\tilde{\mathbf{C}}^*$. In other words, we want a function $\phi(t)$ such that $\gamma(t) = r(t)e^{i\phi(t)}$ and such that $\phi(t)$ is continuous. An idea of how to do this comes from the following intuition, assuming that the functions involved are smooth enough.

First, we have that

$$\phi(t) = \int_{t_0}^{t} d\phi + \phi(t_0).$$

Now $\phi(t)$ can be any angle, so it is not clear in which interval of size 2π it lies. However, $d\phi$ is a differential, so it is very small, and must lie in the interval $[-\pi, \pi]$. Hence, we can 'automatically' compute $d\phi$ (we don't have any ambiguity up to integer multiples of 2π), and use this to compute ϕ.

In the discrete case, this idea goes as follows. We have a function $\gamma(\frac{n}{N})$: a discrete sampling of a curve whose winding number is sought. We can compute a

unique $\phi_0(\frac{n}{N})$; a function whose value at each n is an angle in $[-\pi, \pi)$ such that $\gamma(\frac{n}{N}) = |\gamma(\frac{n}{N})|e^{i\phi_0(\frac{n}{N})}$.

Now we need $\phi(\frac{n}{N}) = \phi_0(\frac{n}{N}) + 2\pi i l_n$, for some integers l_n, such that the resulting ϕ is 'continuous'. However, to say that a discrete function is continuous is not meaningful in the usual sense, so we need to look at the above idea.

We will take as our definition of continuity that $\Delta\phi$ is as small as possible. This is because $\Delta\phi$ is an angle, but $\Delta\phi \sim d\phi$, so we think of $\Delta\phi$ as being almost a differential angle, and hence it must be as small as possible.

This gives us a unique choice of $\phi(\frac{n}{N})$ given $\phi_0(\frac{n}{N})$.

Since $\phi(t)$ is a continuous function on a compact set, it is uniformly continuous. Hence, if Δt is small enough, it *will* be true that $\Delta\phi$ will be smaller than π in absolute value, and hence the above choice will be valid. Hence we can use the above to compute winding numbers, provided we know enough about $\phi(t)$ to insure that it is being sampled at a high enough rate.

3.1.1 Computing Discrete Phase Changes

If we define $\Delta\phi_0(n) = \phi_0(\frac{n+1}{N}) - \phi_0(\frac{n}{N})$, then

$$0 = \phi_0(1) - \phi_0(0) = \sum_{n=0}^{N-1} \Delta\phi_0(n). \tag{17}$$

The total change in phase, as one goes around γ, is given by $\phi(1) - \phi(0)$, and this is really the quantity that we wish to compute. Let $\Delta\phi(n) = \phi(\frac{n+1}{N}) - \phi(\frac{n}{N})$.

Then

$$\phi(1) - \phi(0) = \sum_{n=0}^{N-1} \Delta\phi(n). \tag{18}$$

Now, since $\phi_0(t) \in [-\pi, \pi]$, we know that $\Delta\phi_0 \in [-2\pi, 2\pi]$. Hence $\Delta\phi(n)$ differs from $\Delta\phi_0(n)$ by -2π, 0, or 2π, since $|\Delta\phi_0(n)| \leq \pi$, and the difference is an integer multiple of 2π.

Therefore the sums in equations (17) and (18) differ from each other by an integer multiple of 2π, and hence the sum (18) will be an integer multiple of 2π. Also, all winding number calculations can be determined by a data structure that contains either a 0, 1, or -1 on each oriented edge of the discrete sampling graph. This can be seen as follows:

Again, let $\tilde{\mathbf{C}}^*$ be the universal covering space of $\mathbf{C}^* = \mathbf{C} - \{0\}$. We will take $re^{i\theta}$ to be coordinates for $\tilde{\mathbf{C}}^*$, where we *do not* identify θ with $\theta + 2\pi n$. Partition $\tilde{\mathbf{C}}^*$ into a covering by disjoint *branches*, by sending $re^{i\theta}$ into branch number $\lfloor \frac{\theta}{2\pi} \rfloor$, where $\lfloor x \rfloor$ denotes the greatest integer less than of equal to x.

Given a point at which the function is sampled, consider moving from that point to the point directly above it on the lattice of sampled points. In doing this, under the assumption of minimum phase, we may either stay in the same branch on $\tilde{\mathbf{C}}^*$, or we may go up one branch, or go down one branch. Hence we have a 0, 1, or -1 for this point, representing what happens when we move in the sampling lattice *up* from this point to the next one.

Similarly, we can consider what happens when we move from one point to the

point just to the right of that point. Again we get a 0, 1, or -1.

3.2 Description of Software

In order to carry out the computations in the sections that follow, software was written using the Matlab programming language. The programs written are described below. Listings are given in appendix A.

af.m Computes the discrete cross-ambiguity function of two given sampled functions. Parameters allow the specification of time and frequency intervals and step size used to determine the points at which the discrete cross-ambiguity function is sampled. The program insures that the point $(0,0)$ is contained in the set of points at which the ambiguity function is sampled.

Consider the following sequence of approximations to the cross-ambiguity function of f and g:

$$\begin{aligned} A_{f,g}(\tau,\nu) &= \int_{-\infty}^{\infty} f(t)\overline{g(t-\tau)}e^{-2\pi i\nu t}\,dt \\ &\sim \int_{-N}^{N} f(t)\overline{g(t-\tau)}e^{-2\pi i\nu t}\,dt \\ &\sim \frac{1}{2MN}\sum_{k=-MN}^{MN} f(\frac{k}{M})\overline{g(\frac{k}{M}-\tau)}e^{-2\pi i\nu\frac{k}{M}}. \end{aligned}$$

If we restrict τ and ν to rational values with fixed denominator, and we therefore only consider a finite number of sample points for f and g, then the above approximation consists of a discrete shift of g, multiplied by f,

and followed by a discrete Fourier transform. The resulting formula is the definition of the discrete cross-ambiguity function.

If f and g are one dimensional arrays of length MN, then the discrete cross-ambiguity function of f and g is given by:

$$A^{\text{disc}}_{f,g}\left(\frac{n_t}{N}, \frac{m_f}{M}\right) = \mathbf{fft}\left(f. * \overline{\text{APSHIFT}(g, n_t)}\right)(m_f),$$

where **fft** is the fast Fourier transform, .* represents pointwise multiplication, and APSHIFT(g, n), the aperiodic-shift of g by n, returns g, zero padded, shifted by n, and re-truncated to its size before zero-padding.

gauss_fun.m Computes the function $e^{-\pi t^2}$, with an optional parameter that allows the addition of simulated noise.

mod.m MOD(m, n) computes a representative of the congruence class of m modulo n. The representative computed is the one closest to zero. The function is given by the short piece of code: $m - n * \mathbf{fix}(m/n)$. **fix** is a Matlab function which rounds its argument to an integer, rounding up for negative numbers, and down for positive numbers. Hence this function will work equally well for real number valued parameters, as in the call MOD(x, y). In this case it returns the smallest remainder that can result when subtracting an integer multiple of y from x. Smallest remainder means the remainder with smallest absolute value.

saturate.m Based on a program written by Steven Benno, at the Department of Electrical Engineering, Carnegie Mellon University. This function is used to turn a given Zak transform into an approximate orthonormal basis, as defined in [4]. SATURATE(x, n) is called with x equal to the absolute value of a discrete Zak transform. It then scales x so that its maximum value is 1. Then it multiplies x by n, and truncates all resulting values greater than 1, to 1.

shift.m SHIFT(x, n) periodically shifts the one dimensional array x, by n units. By a periodic shift we mean the application of a cyclic permutation to the index of the array.

apshift.m APSHIFT(g, n) returns the aperiodic shift of the one dimensional array g, by n. This is defined as follows: Let g have length k. Then, if $n \geq 0$, APSHIFT$(g, n) = [0, \ldots, 0, g_1, \ldots, g_{k-n}]$; n zeros followed by the first $k - n$ elements of g. APSHIFT$(g, -n) = [g_{n+1}, \ldots, g_k, 0, \ldots, 0]$; the last $k - n$ elements of g followed by n zeros.

sig_code.m Computes a discrete sampled function corresponding to the *waveform* of a given phase-shift code. Consider the function call: SIG_CODE$(code, len, freq, n)$. *code* will be a one-dimensional array of integers, $freq$, len, and n will be positive integers. Let l_c denote the length of the one dimensional array *code*. The function computed is defined as

follows:

The phase shift code given by *code* is represented by n^{th}-roots of unity, hence the waveform corresponding to the code is given by:

$$s(t) = \sum_{k=1}^{l_c} c_k p(t-k), \tag{19}$$

where $c_k = e^{2\pi i \frac{code_k}{n}}$, $code_k$ is the k^{th} element in the one dimensional array *code*, and we take $p(t)$ to be a square wave of duration 1. However, for convenience we allow the specification of a frequency with which to modulate the output, hence, in this program, $p(t)$ is a complex wave of frequency $freq$. When called with $freq = 0$, the function therefore returns the waveform corresponding to the code.

SIG_CODE$(code, len, freq, n)$ returns a sampled function, of length $len \times l_c$. The function first computes the *basic array*; a sampled version of a complex wave of frequency $freq$, given by $e^{2\pi i freq t}$, with t going from 0 to 1 in steps of size $\frac{1}{len}$. It then returns a concatenation of l_c copies of this basic array, corresponding to the terms in the sum (19), where the k^{th} copy is multiplied by c_k.

zak_code.m Computes a discrete sampled function corresponding to the discrete Zak transform of the waveform of a given phase-shift code. Consider the function call: ZAK_CODE$(code, len, n)$. *code* will be a one-dimensional array of integers, len, and n will be positive integers. Let l_c denote the length

of the one dimensional array *code*. The function computed is defined as follows:

The phase shift code given by *code* is represented by n^{th}-roots of unity, hence the Weil transform corresponding to the code is given, in the unit square, by:

$$\Theta(code)(x, y, 0) = \sum_{k=1}^{l_c} c_k e^{2\pi i k y}. \tag{20}$$

where again $c_k = e^{2\pi i \frac{code_k}{n}}$, and $code_k$ is the k^{th} element in the one dimensional array *code*. Note that the function $\Theta(code)$ is independent of x.

ZAK_CODE$(code, len, n)$ returns a sampled function, of length *len*. The function computes $\Theta(code)(y)$, with y going from 0 to 1 in steps of size $\frac{1}{len}$. Since this has the form of a discrete Fourier transform, the program is just a one line call to the built-in **fft** function, with arguments c_k and *len*.

dsig_code.m Computes a discrete sampled function corresponding to the *doubled waveform* of a given phase-shift code. Consider the function call: DSIG_CODE$(code, len, n)$. *code* will be a one-dimensional array of integers, *len*, and n positive integers. Let l_c denote the length of the one dimensional array *code*. The function computed is defined as follows:

The Weil transform of the phase shift code given by *code*, $\Theta(code)$, is defined as in equation (20), where again $c_k = e^{2\pi i \frac{code_k}{n}}$.

We define the Weil transform corresponding to the *doubled* waveform of the code, in the unit square, by the following doubly-periodic function:

$$\begin{aligned}\Theta(doubled\_code)(x,y,0) &= \Theta(code)(x,y,0)\Theta(code)(y,x,0) \\ &= \left(\sum_{k=1}^{l_c} c_k e^{2\pi iky}\right)\left(\sum_{l=1}^{l_c} c_l e^{2\pi ilx}\right).\end{aligned}$$

Rather than numerically calculating the inverse Weil transform of the above, we compute it formally, and then use these formulas to compute DSIG_CODE.

One readily computes that the waveform corresponding to the double of the code is given by:

$$\sum_{k=1}^{l_c} c_k p(t-k), \tag{21}$$

where

$$p(t) = \sum_{k=1}^{l_c} c_k e^{2\pi ikt}. \tag{22}$$

DSIG_CODE$(code, len, n)$ returns a sampled function, of length $len \times l_c$. The function first computes the *basic array*; a sampled version of $p(t)$, as in equation (22), with t going from 0 to 1 in steps of size $\frac{1}{len}$. It then returns a concatenation of l_c copies of this basic array, corresponding to the terms in the sum (21), where the k^{th} copy is multiplied by c_k.

sig_fbgbarker.m Computes a discrete sampled function corresponding to the new generalization of Barker codes described in section 6.1.

SIG_FBGBARKER$(num, len, freq)$ first computes a *basic array*; a sampled

version of a complex wave of frequency $freq$, given by $e^{2\pi i freq t}$, with t going from 0 to 1 in steps of size $\frac{1}{len}$, followed by len zeros. If num is odd, it is replaced by $num + 1$. The function then returns a concatenation of num copies of this basic array, where the k^{th} copy is multiplied by the k^{th} element in the sequence $\frac{1}{\pi i(-num+1)}, \frac{1}{\pi i(-num+3)}, \cdots, \frac{1}{\pi i(num-3)}, \frac{1}{\pi i(num-1)}$.

Hence this function is completely analogous to the function SIG_CODE, but it uses a modified form of the Fourier coefficients of a square wave, rather than the phase factors given by a phase-shift code.

zak_fbgbarker.m This function is analogous to the function ZAK_CODE, but it uses a modified form of the Fourier coefficients of a square wave, rather than the phase factors given by a phase-shift code. Also, the function was not written to use an FFT, to avoid complications over negative indices.

ZAK_FBGBARKER(num, len) returns a sampled function, of length len. The function computes

$$\sum_{n=-(num/2)+1}^{num/2} \frac{1}{\pi i(2n-1)} e^{2\pi i(2n-1)y},$$

with y going from 0 to 1 in steps of size $\frac{1}{len}$.

dsig_fbgbarker.m This function is analogous to the function DSIG_CODE, but again, it uses a modified form of the Fourier coefficients of a square wave, rather than the phase factors given by a phase-shift code.

DSIG_FBGBARKER(num, len) returns a sampled function, of length $len \times num$. The function first forces num to be even, and then computes the *basic array*. The call ZAK_CODE(num, len) produces the basic array in this case. It then returns a concatenation of num copies of this basic array, with the k^{th} copy multiplied by the k^{th} element in the sequence $\frac{1}{\pi i(-num+1)}, \frac{1}{\pi i(-num+3)}, \ldots, \frac{1}{\pi i(num-3)}, \frac{1}{\pi i(num-1)}$.

welch10.m Computes the discrete Zak transform, and the discrete waveform corresponding to the Welch-10 Costas array code. In this code, the permutation given by

$$\theta = \{2,4,8,5,10,9,7,3,6,1\},$$

is used as a frequency hop code. The waveform corresponding to this code is a concatenation of 10 waveforms, the k^{th} one being a 50 point sample of a complex wave of frequency $\theta(k)$. The discrete Zak transform corresponding to this code is computed as a 50×50 sampling of the function

$$\sum_{n=1}^{10} e^{2\pi i(nx+\theta(n)y)}.$$

welch30.m Computes the discrete Zak transform, and the discrete waveform corresponding to the Welch-10 Costas array code. The permutation

$$\theta = \{3,9,27,19,26,16,17,20,29,25,13,8,24,10,30,28,22,4,12,5,15,14,11,2,6,18,23,7,21,1\},$$

is used as a frequency hop code. The waveform corresponding to this code is a concatenation of 30 waveforms, the k^{th} one being a 50 point sample of a complex wave of frequency $\theta(k)$. The discrete Zak transform corresponding to this code is computed as a 50×50 sampling of the function

$$\sum_{n=1}^{30} e^{2\pi i(nx+\theta(n)y)}.$$

wdata2.m A function to compute the winding numbers for a given Zak transform, on sub-squares of the unit square. WDATA2(Z, n) partitions the unit square into a set of equal $n \times n$ sub-squares, and returns an $n \times n$ array of winding numbers around these squares. Z will be a discrete Zak transform.

The function calls WIND_STRUCT to compute the underlying winding number data of Z, and adds up these data around the sub-square paths, using WINDNUM, to get a total winding around each sub-square.

wind_struct.m A function which computes the basic underlying winding number data for a given discrete Zak transform. WIND_STRUCT(Z) computes the basic horizontal and vertical edge data for the discrete Zak transform Z. See subsections 3.1 and 3.1.1 for a discussion of this data.

windnum.m WINDNUM$(hwind, vwind, l_row, l_col, u_row, r_col)$ takes the output of WIND_STRUCT, as *hwind* and *vwind*, and uses it to compute the winding number around the rectangular path given by the lower left index

(l_row, l_col), and upper right index (u_row, r_col). It calls the functions HWINDPART and VWINDPART, (see the listings in the appendix).

zak.m, izak.m Functions written by Steven Benno, at the Department of Electrical Engineering, Carnegie Mellon University, to compute the discrete Zak transform and inverse discrete Zak transform of given data. These transforms are defined and computed in precisely the same way as the discrete Weil transform defined in [1].

4 Ambiguity Functions and Synthesis of Thumbtacks

Recall that given $f, g \in \mathbf{L}^2(\mathbf{R})$, the cross-ambiguity function of f and g is defined, as in equation (13), by

$$A_{(f,g)}(\tau, \nu) = \int_{-\infty}^{\infty} f(t)\,\overline{g(t-\tau)}\,e^{-2\pi i \nu t}\,dt, \qquad \tau, \nu \in \mathbf{R}.$$

Define the ambiguity function of f by $A_f = A_{f,f}$. This function is also sometimes called the auto-ambiguity of f. The engineering references on auto-ambiguity functions are the same as those given for cross-ambiguity functions, after equation (13), in section 2.1.

Facts: For $s \in \mathbf{L}^2(\mathbf{R})$,

- $A_s(0,0) = \|s\|^2$

- $|A_s(\tau,\nu)| < A_s(0,0), \quad (\tau,\nu) \neq (0,0)$
- $\int\int |A_s(\tau,\nu)|^2 \, d\tau \, d\nu = \|s\|^4$
- Let $s_1(t) = s(t-x)e^{2\pi i y t}$. Then $A_{s_1}(\tau,\nu) = A_s(\tau,\nu)e^{-2\pi i(x\nu+y\tau)}$.

In applications, the ambiguity function acts as a convolution kernel, and one wants to perform de-convolution. Because of the above equations, we can not get an ambiguity function which is a delta function, or an approximate identity, and hence this de-convolution becomes non-trivial. A solution to this problem, for purposes of certain applications discussed in the engineering literature, is to find functions whose ambiguity functions are 'thumbtacks'. An ambiguity function is a thumbtack if it can be expressed as a sum of two functions, one a function which has the property that its absolute value squared is an approximate identity, and the other a function which is uniformly bounded by some small $\epsilon > 0$.

In the rest of this work, we will apply the machinery developed above to the problem of finding waveforms (ie. reasonably well behaved functions on $\mathbf{R}$) which have thumbtack or related ambiguity functions.

4.1 Background on the Synthesis Problem

The synthesis problem for ambiguity functions can be stated as follows. Given a function $G(x,y) \in \mathbf{L}^2(\mathbf{R}^2)$, find a function $f(t)$ in some desired class of functions, such that $\|A_f - G\|_2$ is minimized over all f in the class.

Auto-ambiguity functions have been characterized by Siebert [29], Sussman [32], Wilcox [34], and Auslander and Tolimieri [6]. They showed how to get an $\mathbf{L}^2$ best ambiguity function approximating a given, *complex valued*, function.

In applications, one wants to synthesize $|A_f(\tau,\nu)|$, and not $A_f(\tau,\nu)$. That is, one wants to minimize, for example, $\||A_f(x,y)| - |G(x,y)|\|_2$. This is an unsolved problem. Engineers use trial, experience, intuition and ad-hoc constructions to get waveforms for a given problem.

In the discrete case, for any particular synthesis problem, we get a system of polynomial equations which can, in principle, be solved. However, the complexity of this solution is prohibitive.

One can also use the technique of following the gradient to local solutions, perhaps combined with the known techniques for the synthesis problem for A_f, to get numerical algorithms for synthesizing $|A_f(\tau,\nu)|$. This is still a relatively complex algorithm, and only gives numerical solutions. Hence it is still of interest to study specific synthesis questions from a mathematical point of view.

In what follows we will apply the formalism of the Heisenberg group and the Weil transform, and the observations of the previous sections, to provide insight into techniques for synthesizing thumbtacks. We will study known techniques and provide new ones.

4.2 Controlling Ambiguity on the Integer Lattice

Here the theory of approximate orthonormal bases and frames will be reviewed, and it will be proved that approximate orthonormal bases lead to thumbtacks on the integer lattice.

Definition: An ambiguity function A_g, with $A_g(0,0) = 1$, is a *lattice-thumbtack with parameter* ϵ, if $|A_g(m,n)| < \epsilon$, for all $(m,n) \neq (0,0)$. □

To understand which g might have the above property, recall that the auto-ambiguity function of g, restricted to integer values of τ and ν, gives the sliding windowed Fourier coefficients of g with window g. Hence control of A_g on the integer lattice can be related to controlling the basis-like properties of the family

$$\left\{g_{m,n} = g(t-m)e^{2\pi int}\right\} m,n \in \mathbf{Z}.$$

Let the family above be called the g-basis.

Consider the map from $\mathbf{L}^2(\mathbf{R})$ to $l^2(\mathbf{Z}^2)$ (for nice enough g), given by

$$A_g : f(t) \mapsto \{A_{f,g}(m,n) = \langle f, g_{m,n}\rangle\}_{\mathbf{Z}^2}$$

Because

$$A_{f,g}(m,n) = \left\langle \Theta(f)\overline{\Theta(g)}, e^{2\pi i(mx+ny)} \right\rangle,$$

A_g is a bounded operator if and only if there exists a $B < \infty$ such that $|\Theta(g)(x,y)| < B$ a.e.. A_g is injective if and only if $|\Theta(g)(x,y)| > 0$ a.e.. ${A_g}^{-1}$ is

bounded if and only if there exists an $A > 0$ such that $|\Theta(g)(x,y)| > A$ a.e.. The family $\{g_{m,n}\}_{m,n\in\mathbf{Z}}$ is an orthonormal basis for $\mathbf{L}^2(\mathbf{R})$ if and only if $A = B = 1$ above.

But the zero theorem, and Balian's theorem show that for 'nice' g, the g-basis is never a frame.

The theory of approximate frames, as developed in [3] and [4], is designed to get around the above circumstances, to the extent that it is possible. The idea is that we *can* have approximating sequences of smooth functions in H_1 that give rise to frames or orthonormal bases, except on shrinking nested sets. These sequences are called *approximate frames* and *approximate orthonormal bases* respectively. In the two cited papers, it was demonstrated that these families can substitute for frames under certain conditions, in applications. We now add one more result to the discussions in those papers.

Theorem 4.1 *Let $g_i \to g$ be an approximate orthonormal basis. Then g_i has an ambiguity functions which is a thumbtack on the integer lattice, with parameter ϵ_i, such that $\lim_{i\to\infty} \epsilon_i = 0$.*

Proof: A function g for which the g-basis is an orthonormal basis is one for which $A_g(0,0) = 1$ and $A_g(m,n) = 0$, $(m,n) \neq (0,0)$. Hence, such a g is a lattice-thumbtack with parameter $\epsilon = 0$.

Therefore, $g_i = g + (g_i - g)$ is a sum of a lattice-thumbtack with parameter 0,

and a function with total energy $\epsilon_i \to 0$. But this energy is a bound on any non-zero sliding windowed coefficients. Hence g_i is a lattice-thumbtack with parameter ϵ_i. □

4.3 Waveform Shaping and the Weil Transform

We will now see that the Weil transform puts many current techniques of waveform construction into a simple form of multiplication of functions. If we replace $p(t)$ by

$$s(t) = \sum_{m,n} a_{m,n} p(t-m) e^{2\pi i n t}, \tag{23}$$

we call this waveform shaping. Many techniques of radar waveform construction fall under this general scheme.

Notice that

$$\Theta(s)(x,y) = f(x,y)\Theta(p)(x,y), \tag{24}$$

where $f(x,y) = \sum_{m,n} a_{m,n} e^{2\pi i(nx+my)}$ is a doubly-periodic function. So waveform shaping is multiplication by a doubly-periodic function under the Weil transform.

In equation (24), f is doubly-periodic, and $\Theta(p)$ is "Heisenberg group periodic". By exchanging the underlying groups, we can study waveform shaping techniques. Specifically, letting $\lfloor x \rfloor$ denote the greatest integer less than or equal to x, we can write $\Theta(p) = P(x,y)e^{2\pi i \lfloor x \rfloor y}$, where $P(x,y)$ is a doubly periodic

function. Hence

$$\begin{aligned}\Theta(s)(x,y) &= f(x,y)\Theta(p)(x,y) \\ &= f(x,y)\left(P(x,y)e^{2\pi i\lfloor x\rfloor y}\right) \\ &= \left(f(x,y)e^{2\pi i\lfloor x\rfloor y}\right)P(x,y).\end{aligned}$$

If we take p to be the characteristic function of the interval $[0,1]$, then $P(x,y)=1$ everywhere, and hence s becomes a waveform which represents f.

We will use the above idea to investigate waveform shaping. It is essentially the idea of taking a doubly periodic function, making it a Heisenberg periodic function, and taking its inverse Weil transform, which is then well defined.

4.4 The Weil Transform and Ambiguity Bounds

An easy computation shows that

$$A_g(\tau,\nu) = \left\langle \Theta(g)(x,y), \Theta(g)(x-\tau, y+\nu)e^{2\pi i x\nu}\right\rangle.$$

This is correlation over the Heisenberg group, and it is just ordinary two dimensional affine correlation, with an extra phase factor.

4.4.1 Small Support in the Coset Space

One way to use the above observation is to note that if $\Theta(g)$ is supported in a small neighborhood of some point p, then its ambiguity surface will be sharp.

This is because $\Theta(g)(x,y)$ and $\Theta(g)(x-\tau, y+\nu)$ will have disjoint support for τ, ν away from integers. This allows us to give arbitrarily high-resolution detection waveforms, with resolution defined as in Blahut's paper [9]. For example, if F_n is a family of smooth functions in H_1, such that $\|F_n\|_2 = 1$, and with F_n restricted to the unit square, supported in a disk of radius $1/n$ about the point $(\frac{1}{2}, \frac{1}{2})$, then the main lobe of the ambiguity surface of $\Theta^{-1}(F_n)$ obviously is contained in a disk of radius $2/n$, by the above observation.

By continuity, the above observation can be extended to the situation where $\Theta(g)$ can be written as a sum of a function with small support, and a function with small norm. In other words, if $\Theta(g)$ has most of its mass on a small set, then most of the energy of the main lobe of its ambiguity function will be contained in a small set.

4.4.2 Time-frequency Staggering

We will now present another way to use the above observation that correlation over the Heisenberg group is somewhat like affine correlation.

By time-frequency staggering we mean the process of defining a function which has thumbtack-like auto-correlation over a given space, as follows: Choose some 'basic feature' to define the function on a patch of the space. Place copies of the basic feature in a pattern over the space. Choose the pattern so that only a few copies of the basic feature overlap for any given non-zero shift. This will work

because as the function is shifted around in the space and compared with the unshifted function, we expect that except for a zero shift, only a few coincidences occur for any given shift. This is caused by the nature of the placement of the patches above. However, because these few coincidences *do* occur, they will "use up" the total correlation, in a way which produces correlation values that are only some definite fraction of the correlation at zero shift. Of course, we must be lucky enough to have a situation where no correlation occurs by accident, say because the 'seams' between the patches 'look like' other parts of the function. While this may be hard to guarantee, it is nevertheless true that some examples of thumbtack ambiguity surfaces can be explained using this idea.

Several examples in the literature use similar ideas [8], [25], but not in the context of the Heisenberg group or the Weil transform.

4.5 Correlation Over the Torus

Let $f \in \mathbf{L}^2(\mathbf{R})$. Let $\Theta(f)(x,y,0) = F(x,y) = \sum_{m,n} a_{m,n} e^{2\pi i(mx+ny)}$. Such an expansion always exist because $F \in \mathbf{L}^2(\Gamma\backslash\mathcal{N})$ (see [5] for details). Recall that the ambiguity function of f is given by correlation over the Heisenberg group. By changing τ to $-\tau$ in the definition of the ambiguity function, we may write:

$$A_f(\tau,\nu) = \int_0^1 \int_0^1 F(x,y)\overline{F(x+\tau,y+\nu)e^{2\pi i x\nu}}\,dx\,dy.$$

Suppose we ignore the phase factor above. Then we are considering correlation

over the torus,

$$\begin{aligned}
&\mathcal{A}_{\mathbf{T},f}(\tau,\nu) = \int_0^1 \int_0^1 F(x,y)\overline{F(x+\tau,y+\nu)}\,dx\,dy \\
&= \int_0^1 \int_0^1 \sum_{m,n} a_{m,n} e^{2\pi i(mx+ny)} \sum_{m',n'} \overline{a_{m',n'}} e^{-2\pi i(m'(x+\tau)+n'(y+\nu))}\,dx\,dy \\
&= \sum_{m,n} |a_{m,n}|^2 e^{-2\pi i(m\tau+n\nu)}.
\end{aligned}$$

For functions such as those corresponding to Costas arrays [13], to be defined later, which have the property that their Fourier coefficients are all 0 or 1, we have that $A_{\mathbf{T},f}(\tau,\nu) = F(-\tau,-\nu)$. Also, for small ν, $A_{\mathbf{T},f}(\tau,\nu) \sim A_f(\tau,\nu)$, and hence the ambiguity function for small τ looks like the function coming from modifying the Fourier coefficients of the Weil transform of the function, by taking their absolute value squared. In particular, for Costas arrays $A_{\mathbf{T},f}(\tau,\nu) \sim \Theta(f)(-\tau,-\nu,0)$, for small ν.

5 Exploration of Known Waveforms Using the Weil Transform

In this section, many of the existing techniques of radar waveform construction will be explored in the context of waveform shaping with the Weil transform. It will be seen that the success of these techniques can often be understood in terms of the concepts presented in the previous sections. It is the theme of this survey that the thumbtack ambiguity properties of these waveforms can be explained in

terms of geometric properties of their Weil transforms. The observed geometric properties come from time-frequency staggering over the Heisenberg group, small support of functions in the coset space, and other observations about the shape of the Weil transforms of the waveforms. Hence we will give a catalog of examples to demonstrate this.

The known techniques that we will study can be found in the references [26], [11], and [15].

In each of the following subsections, a given type of waveform will be studied. For one or several waveforms of each type, the waveform, and the absolute value and phase of the Weil transform will be graphed, together with a discrete winding number plot, where appropriate.

Often, given waveforms $f(t)$ are designed to have good auto-correlation properties. That is, they have good ambiguity functions for time shifts, and frequency shifts are ignored. In these cases the auto-correlation properties can often be explained in terms of how the Weil transform of the waveform, $\Theta(f)$, depends on one of its variables. By taking the product $\Theta(f)(x,y)\Theta(f)(y,x)$, it is possible to construct a waveform which has the desired properties in both time and frequency. Note that, in the context of waveform shaping, the product will be doubly periodic, since it is a product of doubly periodic functions. This constitutes a genuine improvement over known techniques, and will be depicted in the relevant examples.

5.1 Uniform Pulse Trains

Early radar improved range-doppler resolution with the use of regular pulse trains (see [26], [9], [11]).

$$s(t) = \sum_{n=0}^{N} p(t - nT_p)$$

where $p(t)$ is a waveform of duration T_p.

The ambiguity surface of $s(t)$ is sharper than that of $p(t)$. By taking the Weil transform of the above equation, it is easy to see why.

$$\Theta(s)(x,y) = \left(\sum_{n=0}^{N} e^{-2\pi i n y}\right) \Theta(p)(x,y) = f(y)\Theta(p)(x,y).$$

But $f(y)$ is, up to a multiple of the modulating factor $e^{-2\pi i \frac{N}{2} t}$, a Dirichlet Kernel; an approximate identity. So this example illustrates the concept of small support in the coset space, from section 4.4.1.

Figure 2 shows a rectangular pulse. Figure 3 shows its ambiguity function. Figure 4 shows a train of rectangular pulses. Figure 5 shows its ambiguity function. Figures 6 and 7 depict the corresponding Weil transforms, and show how the Weil transform and the concept of small support explains what is going on.

Figure 2: A Rectangular Pulse

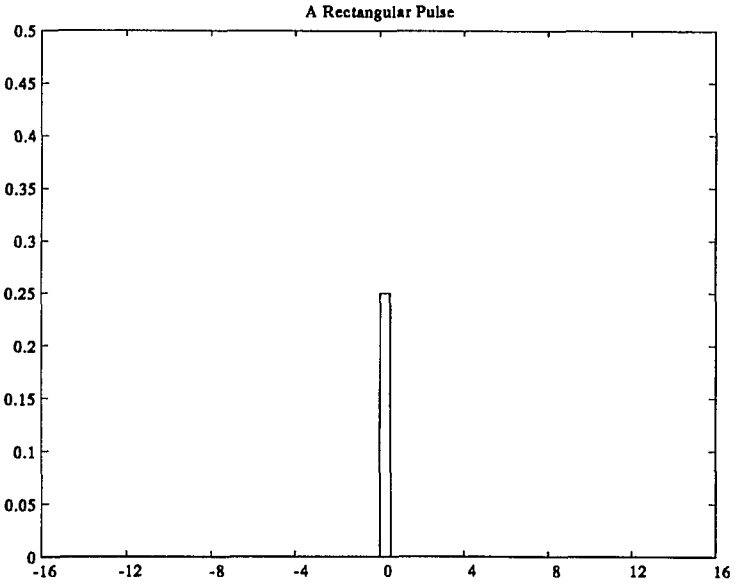


Figure 3: The Ambiguity Surface of a Rectangular Pulse

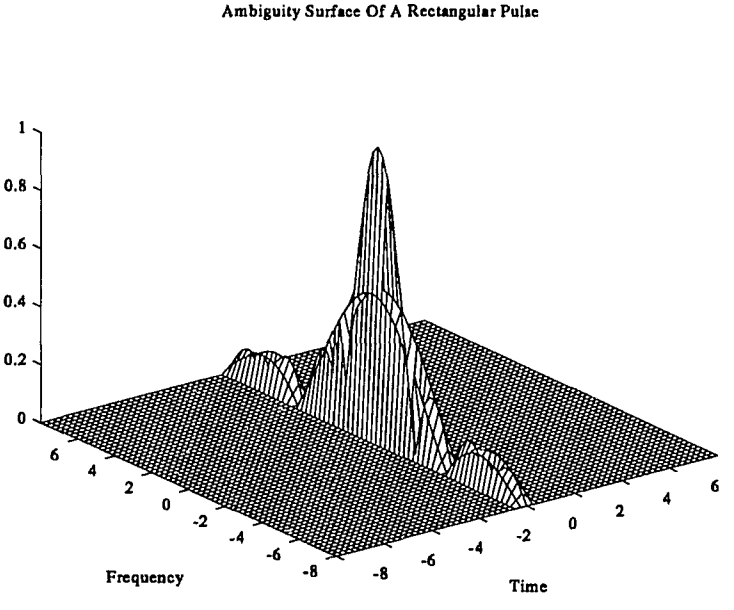


Figure 4: A Rectangular Pulse Train

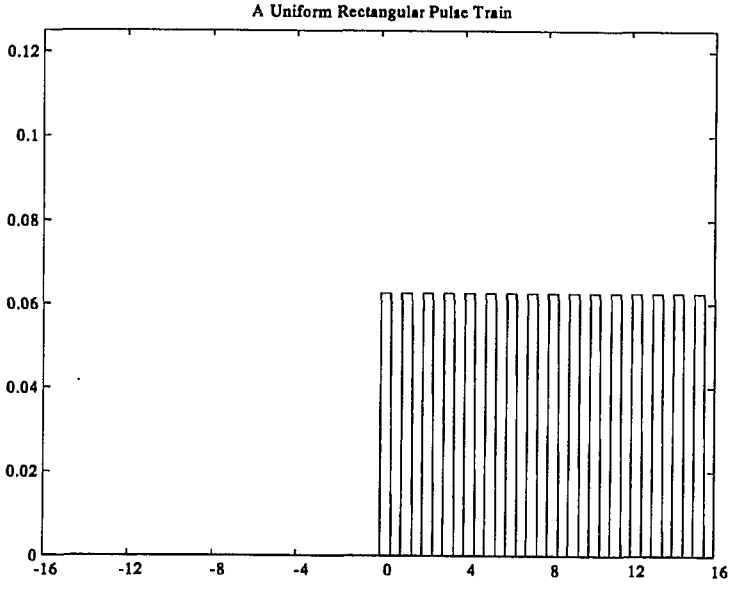


Figure 5: The Ambiguity Surface of a Rectangular Pulse Train

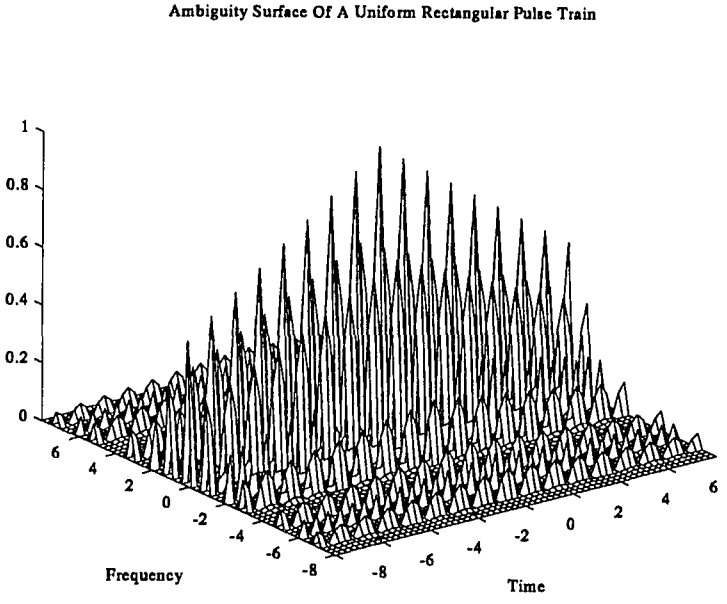


Figure 6: The Weil Transform of a Rectangular Pulse

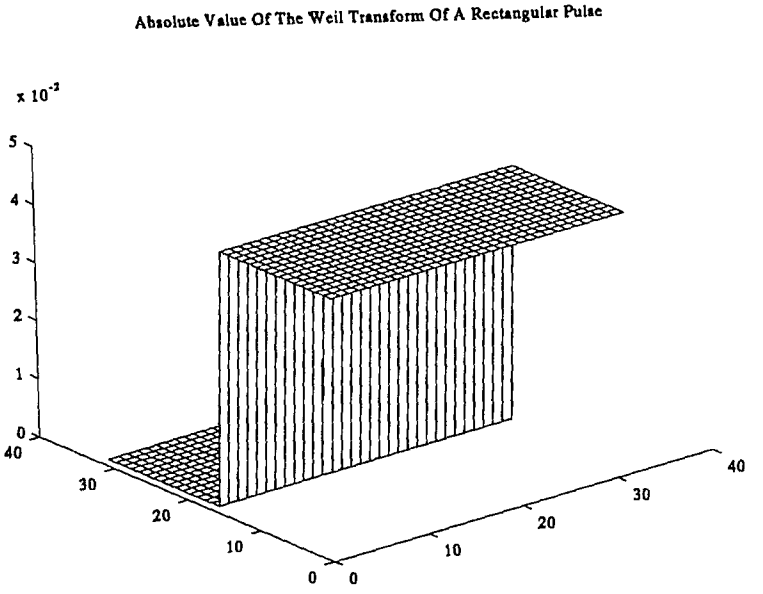
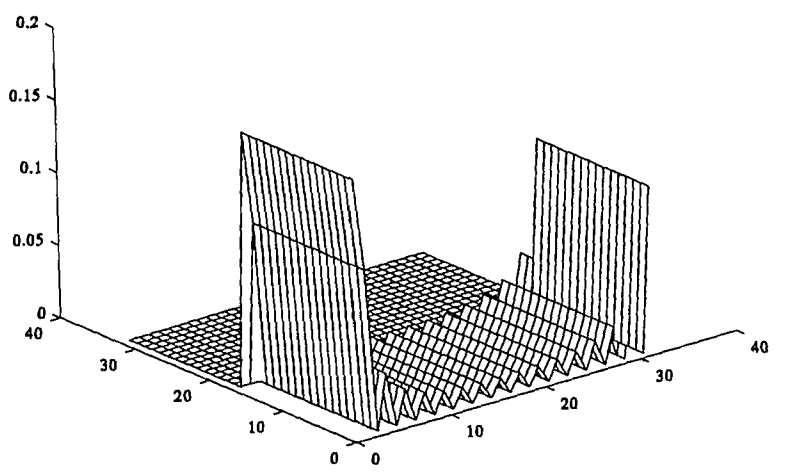


Figure 7: The Weil Transform of a Rectangular Pulse Train



5.2 Phase Shift Coding

Engineers use a kind of waveform construction called phase shift coding. This is a type of waveform shaping (equation (23)), where $a_{m,n} = 0$, for $n \neq 0$, and all non-zero $a_{m,n}$ are taken from the k^{th}-roots of unity, for some fixed k. Frequently $k = 2$.

The subsections below will detail several types of phase codes: Barker codes, generalized Barker codes and m-sequences, including Gold codes. For each of these codes, the waveforms are constructed as follows. We are given a sequence of n^{th}-roots of unity, $a_1, \ldots, a_m$. We take some basic pulse $p(t)$ of duration 1, say a rectangular pulse, and form

$$s(t) = \sum_{l=1}^{m} a_l p(t - l).$$

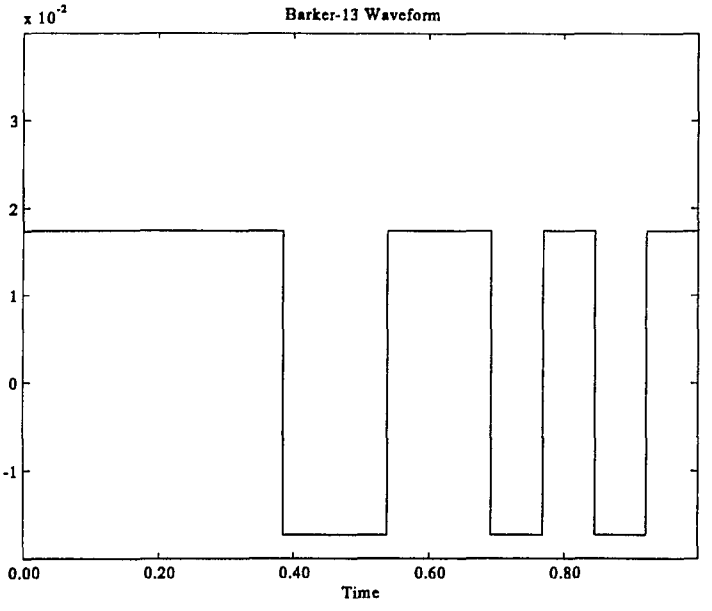
As pointed out in the introduction to section 5, when $p(t)$ is a rectangular pulse, the Weil transform of s does not depend on x. Hence the graphs in this section will all be one dimensional.

5.2.1 Barker Codes

For a code given by a finite sequence $a_1, \ldots, a_n$, its (discrete, non-periodic) auto-correlation function is defined, for $0 \leq k \leq n - 1$, by

$$A(k) = \sum_{l=1}^{n-k} a_l \, \overline{a_{l+k}}.$$

Figure 8: The Barker-13 Waveform



Note that this is an inner product between a_i and a non-periodic shifted version of a_i. Also note that for binary sequences, $(n = 2)$, the complex conjugation has no effect; it is included to make the definition extend to general settings below.

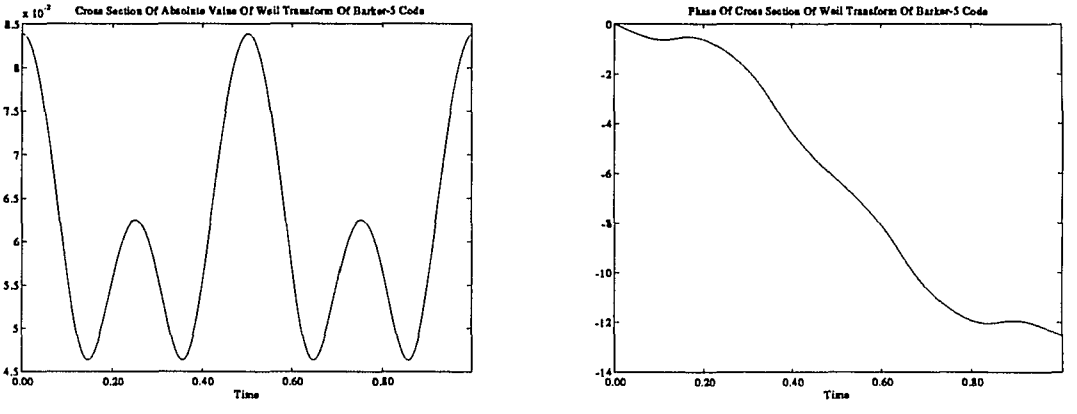
There are a few binary $(+1, -1)$ sequences, for which the auto-correlation function is optimal in the sense of Barker [26]. A code is optimal in the sense of Barker if its auto-correlation function satisfies $A(0) = n$, and $|A(k)| \leq 1, k \neq 0$. The Barker-optimal binary codes are known as Barker codes. Table 1 list all known Barker codes.

Figures 9–20 show the Weil transform of various Barker codes, the corresponding ambiguity surface, and the ambiguity surface for the doubled waveform. The Weil transforms for these examples only depend on y, and so are shown as func-

Table 1: The Known Barker Codes

Name	Length	Code
b_1	1	1
b_2	2	1,1
b_3	3	1,1,−1
b_4	4	1,1,1,−1
b_5	5	1,1,1,−1,1
b_7	7	1,−1,1,1,−1,−1,−1
b_{11}	11	1,−1,1,1,−1,1,1,1,−1,−1,−1
b_{13}	13	1,1,1,1,1,−1,−1,1,1,−1,1,−1,1

Figure 9: The Weil Transform of the Barker-5 Waveform



tions of one variable. The doubled waveform, as described earlier, is gotten by multiplying the Weil transform of the waveform, by the Weil transform with x and y exchanged with each other. This allows the geometric cancellation properties to act in both time and frequency.

There is good evidence that no Barker codes of length > 13 exist. See [10], [33], [31]. See also [23] for an additional study of Barker codes.

Figure 10: The Ambiguity Surface of the Barker-5 Waveform

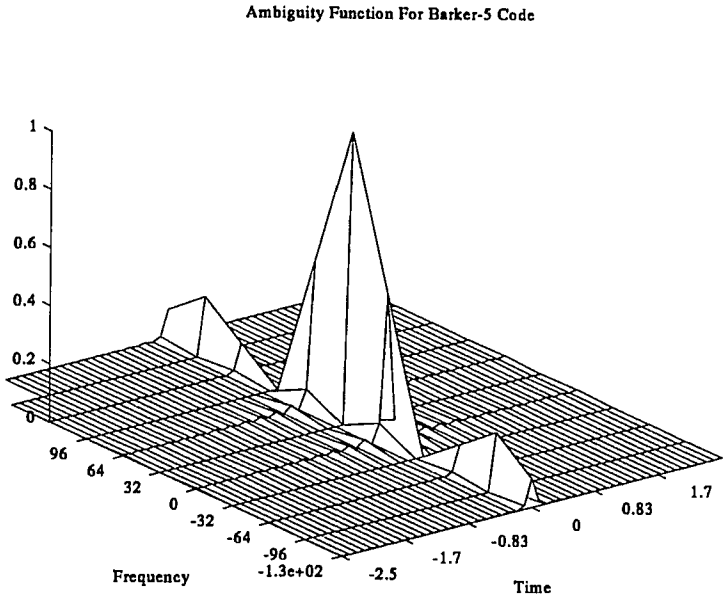


Figure 11: The Ambiguity Surface of the Doubled Barker-5 Waveform

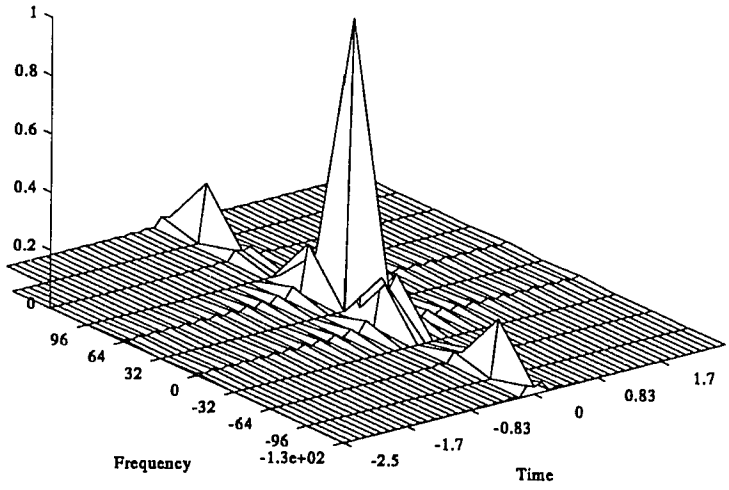


Figure 12: The Weil Transform of the Barker-7 Waveform

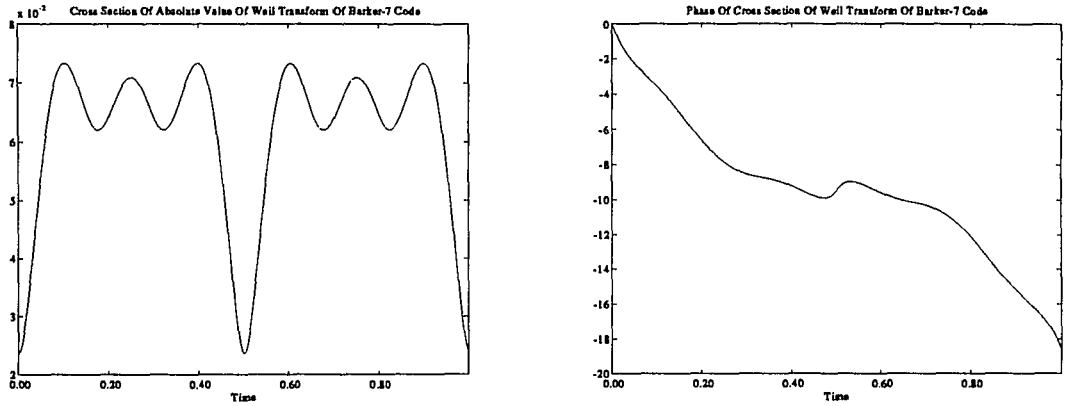


Figure 13: The Ambiguity Surface of the Barker-7 Waveform

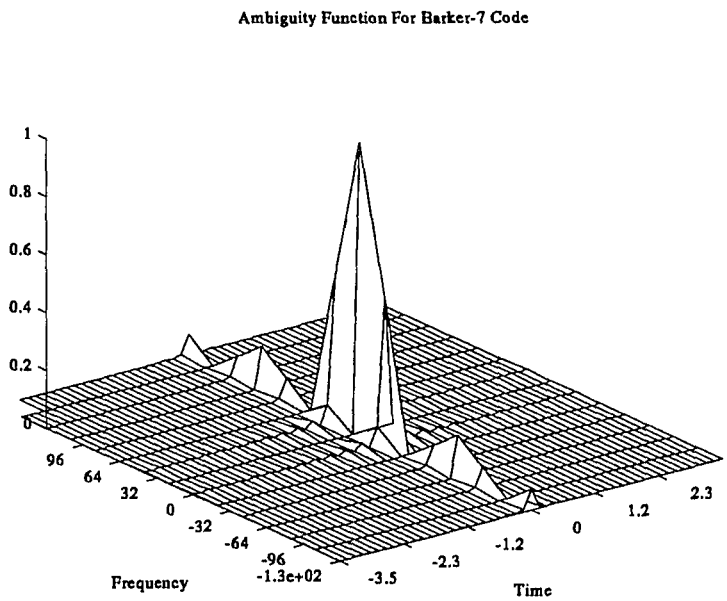


Figure 14: The Ambiguity Surface of the Doubled Barker-7 Waveform

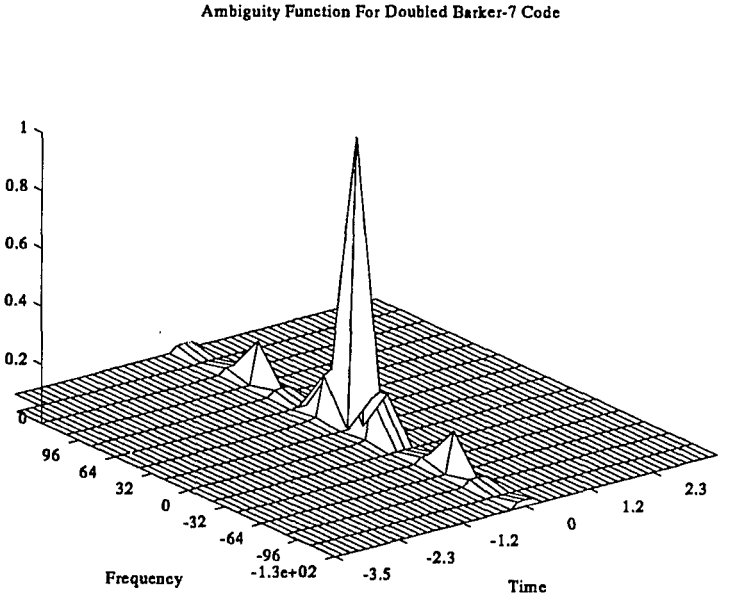


Figure 15: The Weil Transform of the Barker-11 Waveform

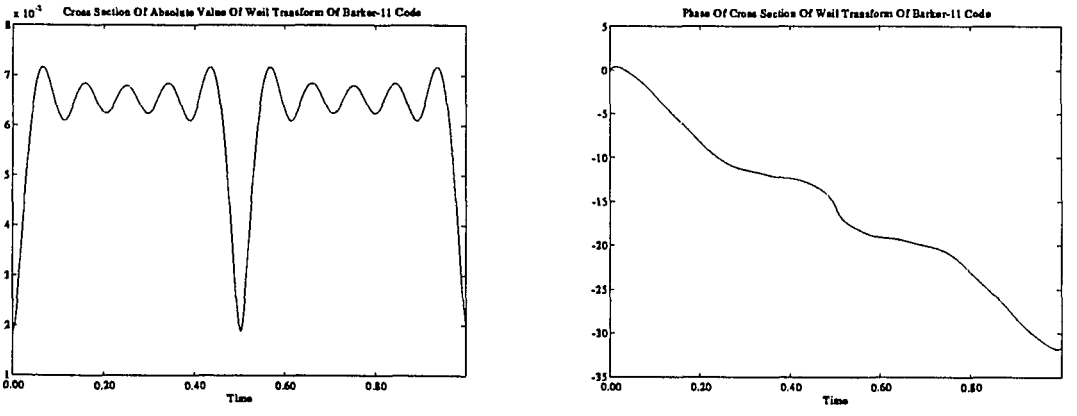


Figure 16: The Ambiguity Surface of the Barker-11 Waveform

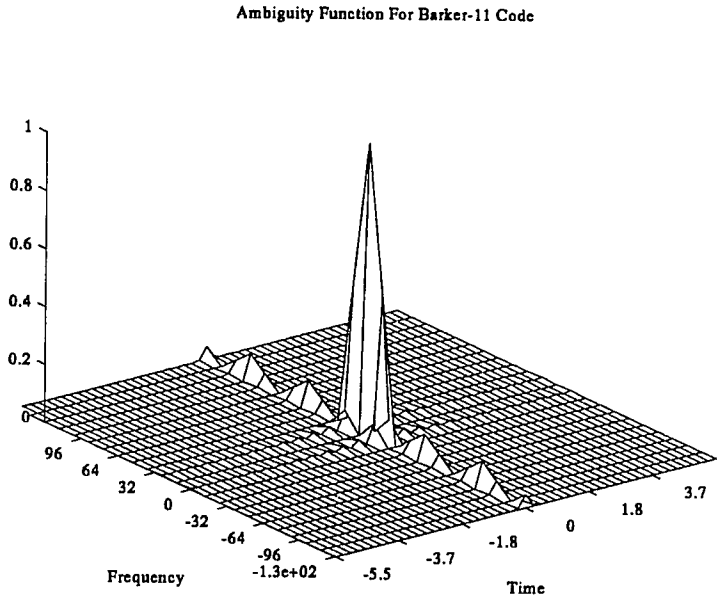


Figure 17: The Ambiguity Surface of the Doubled Barker-11 Waveform

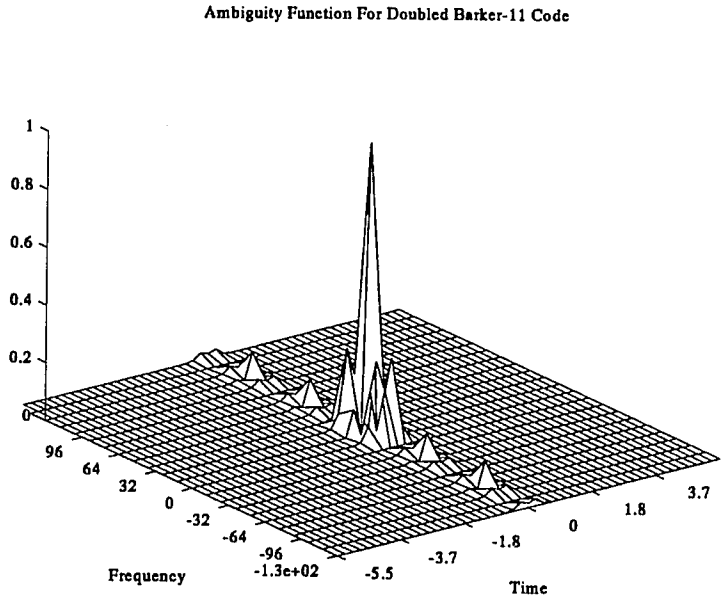


Figure 18: The Weil Transform of the Barker-13 Waveform

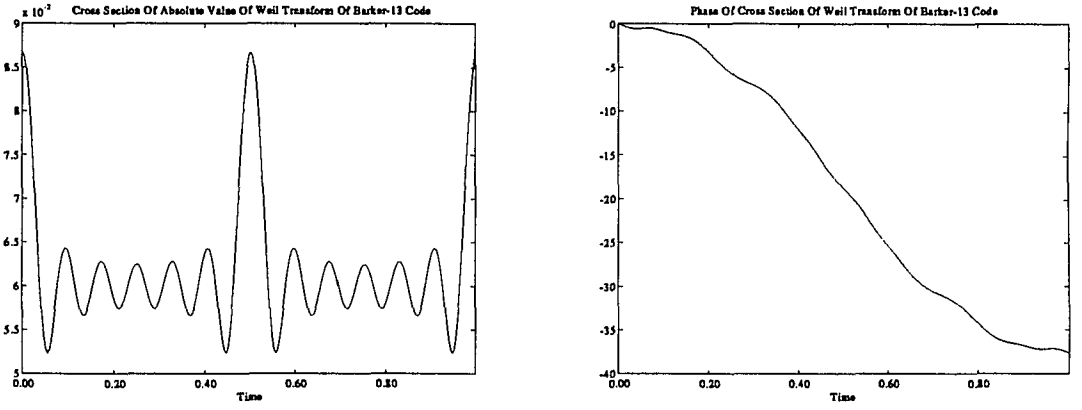


Figure 19: The Ambiguity Surface of the Barker-13 Waveform

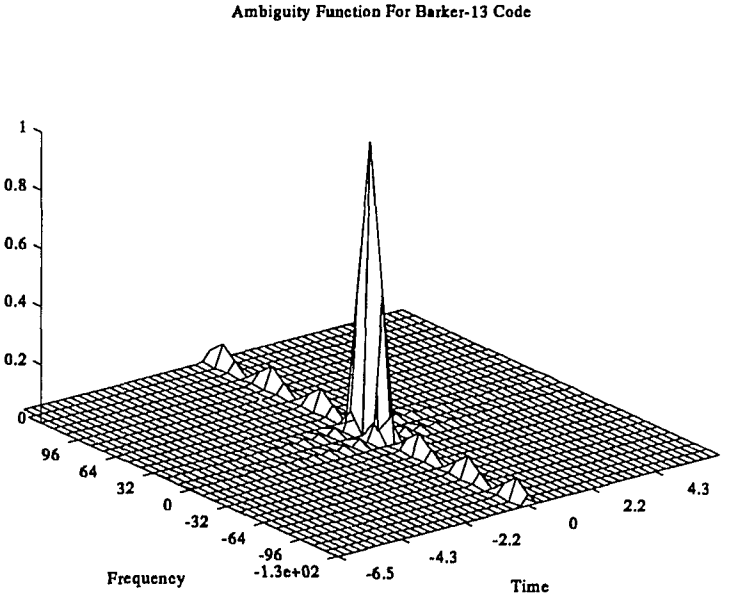


Figure 20: The Ambiguity Surface of the Doubled Barker-13 Waveform

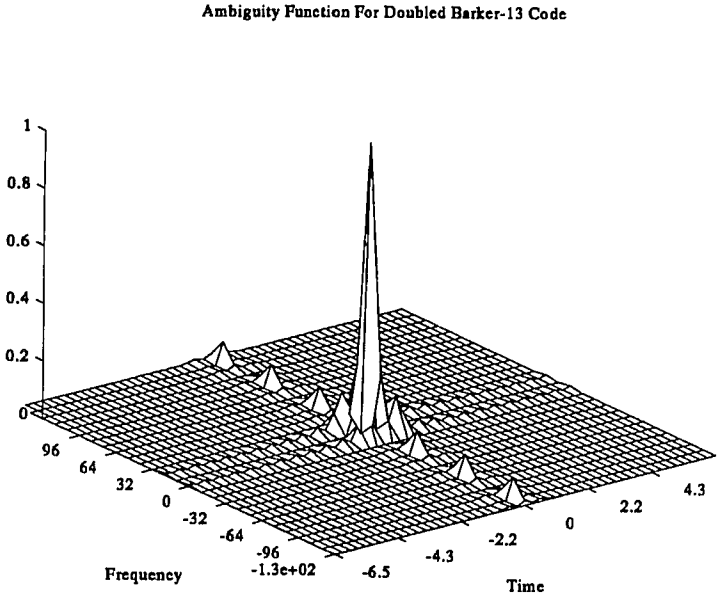


Table 2: The Generalized Barker Codes of Length 12

Name	Code
$b_{6,12,1}$	$0,0,1,0,5,2,2,4,0,4,1,3$
$b_{6,12,2}$	$0,0,1,1,5,5,2,3,0,3,0,4$
$b_{6,12,3}$	$0,0,1,2,3,1,1,5,0,3,0,4$

5.2.2 Generalized Barker Codes

Golumb and Scholtz [20] have studied a generalization of Barker codes. They studied phase codes using n^{th}-roots of unity, for $n > 2$, which are optimal in the sense of Barker. They concentrated on the case $n = 6$. They found codes for each length tried, and tentatively conjectured that sextet Barker codes exist for any length. The longest ones that they found were of length 12, and are listed in table 2.

In table 2, the codes are specified by integers which give the appropriate power of a primitive sixth root of unity. Figures 21–29 depict the Weil transform and ambiguity surfaces for the above waveforms, and the ambiguity surfaces of the doubled waveforms, as in the previous section.

5.2.3 MPS codes

Another sense in which Barker codes can be generalized is to define a code of length m using n^{th}-roots of unity to be optimal if $\max_{k\neq 0} |A(k)|$ is minimal over all possible codes of fixed m and n. Binary codes which are optimal in this sense

Figure 21: The Weil Transform of the $b_{6,12,1}$ Waveform

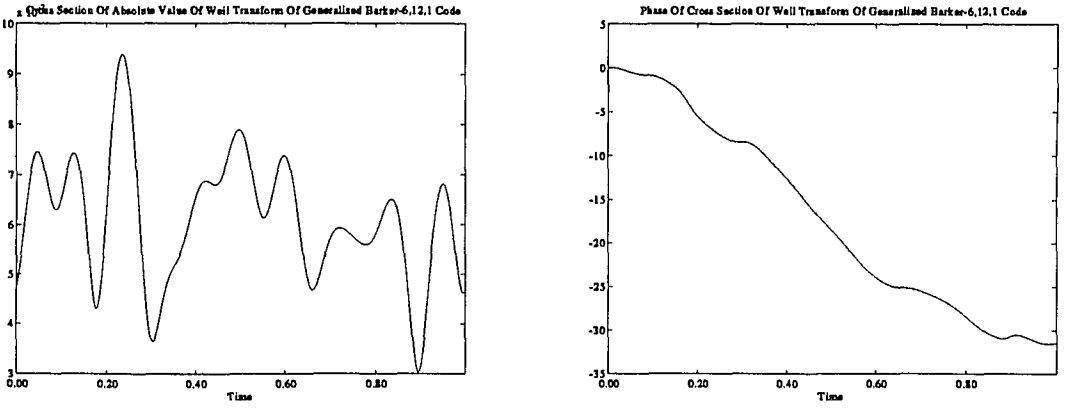


Figure 22: The Ambiguity Surface of the $b_{6,12,1}$ Waveform

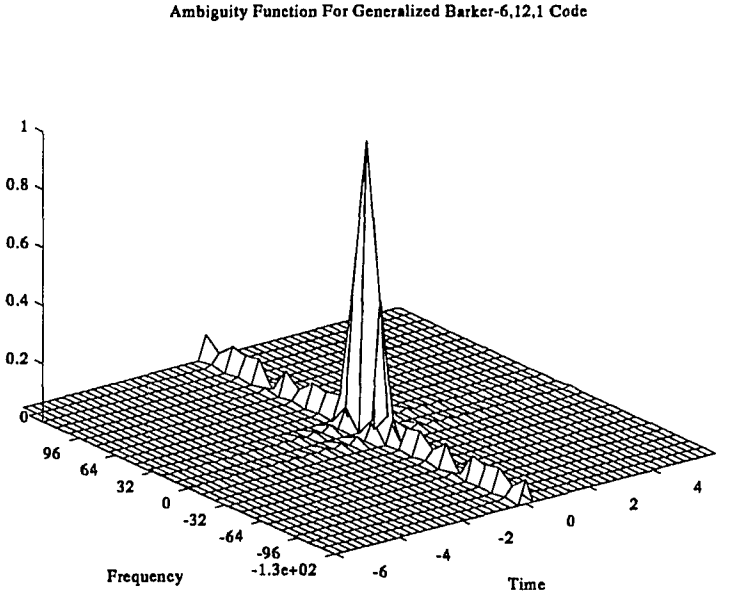


Figure 23: The Ambiguity Surface of the Doubled $b_{6,12,1}$ Waveform

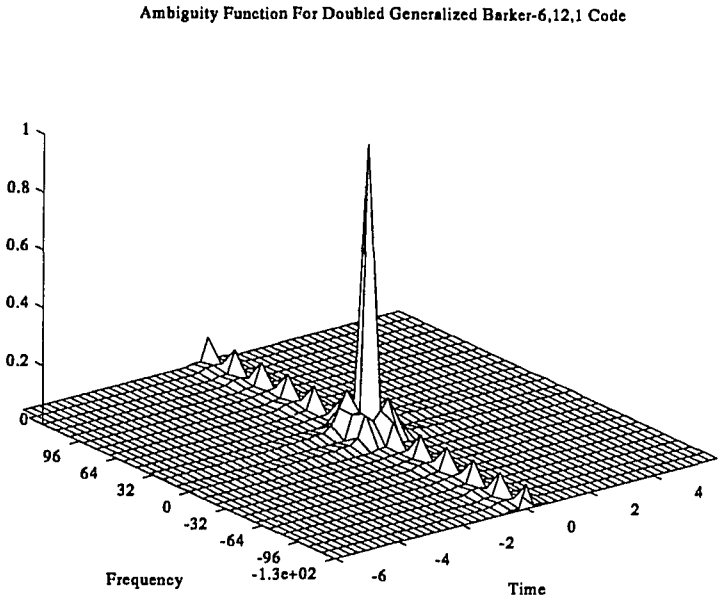


Figure 24: The Weil Transform of the $b_{6,12,2}$ Waveform

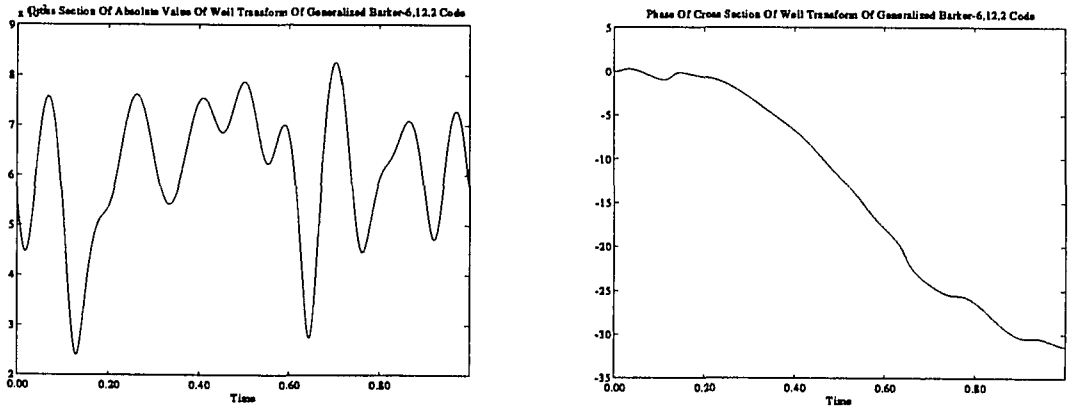


Figure 25: The Ambiguity Surface of the $b_{6,12,2}$ Waveform

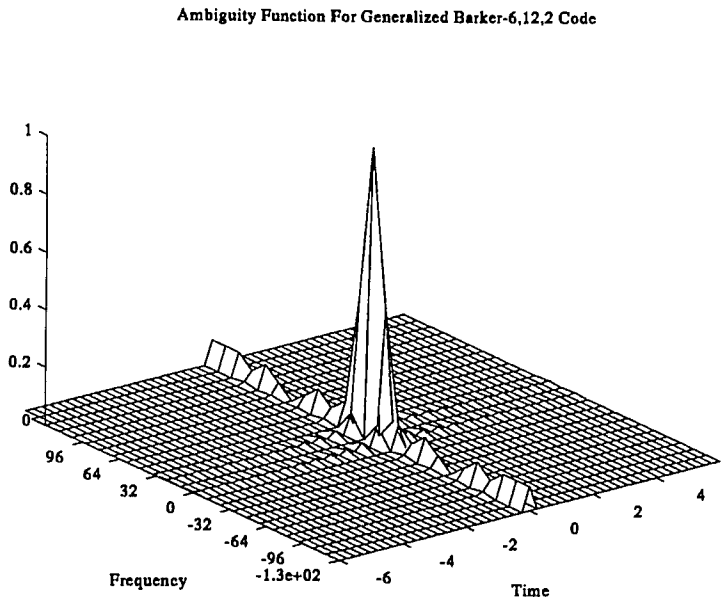


Figure 26: The Ambiguity Surface of the Doubled $b_{6,12,2}$ Waveform

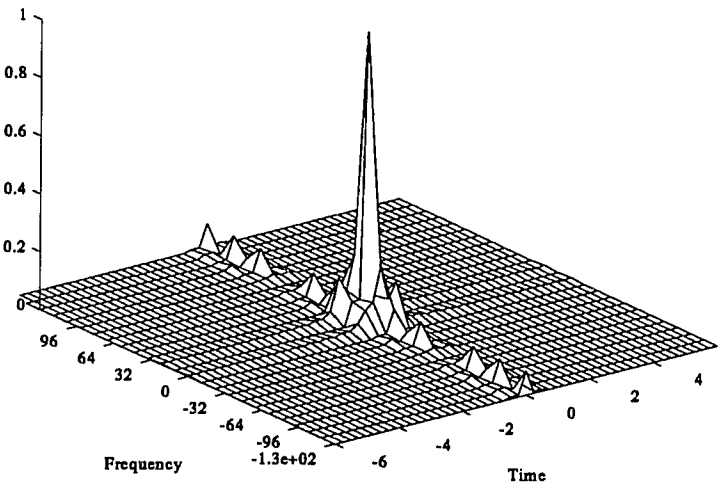


Figure 27: The Weil Transform of the $b_{6,12,3}$ Waveform

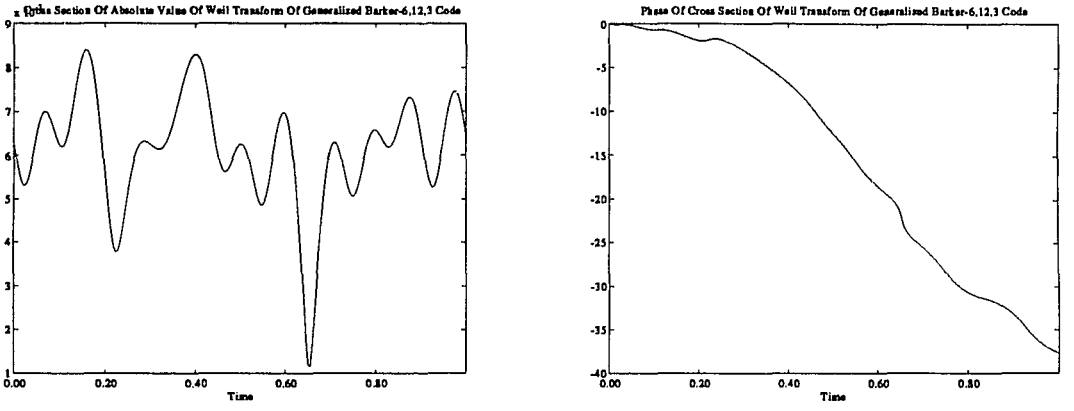


Figure 28: The Ambiguity Surface of the $b_{6,12,3}$ Waveform

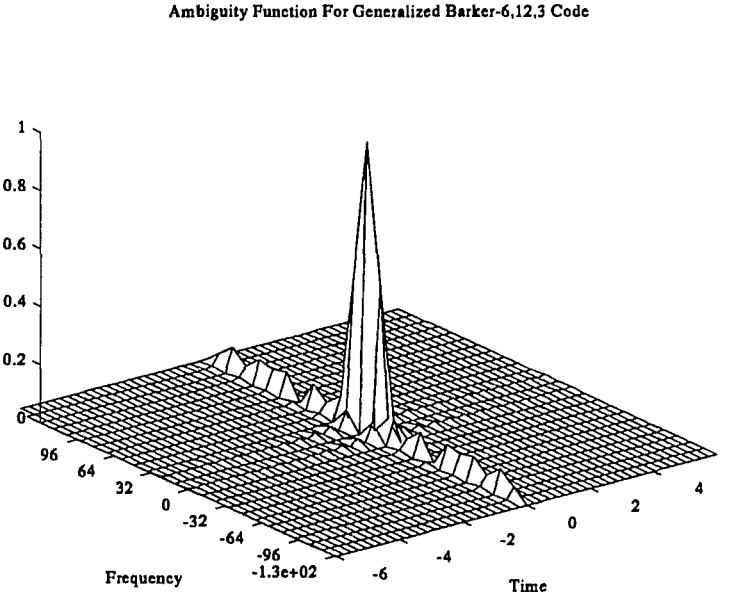
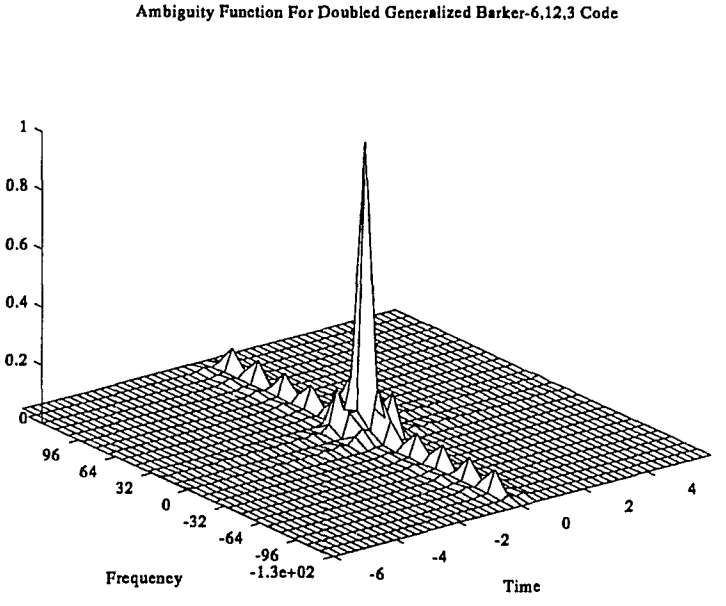


Figure 29: The Ambiguity Surface of the Doubled $b_{6,12,3}$ Waveform



have been enumerated for lengths up to 40, by Lindner (see [10]), and for lengths 41-48 by Cohn, Fox and Baden [10]. The latter call such codes minimum peak sidelobe codes (MPS codes). Table 3 list some samples of these codes of lengths up to 48. All MPS codes of lengths between 14 and 21 have a maximum sidelobe height of 2. All MPS codes of lengths between 22 and 48 have a maximum sidelobe size of 3, except for the codes of lengths 25 and 28 where the bound is 2.

Figures 30–47 show the graphs, as in the previous sections, for the waveforms corresponding to the MPS codes listed in table 3.

Table 3: Some MPS Codes

Name	Code
c_{14}	−1,1,−1,1,−1,−1,1,−1,−1,−1,−1,−1,1,1
c_{21}	1,−1,1,1,−1,1,−1,1,1,1,−1,1,1,1,−1,−1,−1,−1,−1,1,1
c_{22}	−1,−1,1,1,1,−1,−1,1,1,−1,1,1,−1,1,−1,1,−1,1,1,1,1,1
c_{25}	1,−1,−1,1,−1,−1,1,−1,1,−1,1,−1,−1,−1,−1,−1,−1,1,1,1,1,−1,−1,1,1,1
c_{28}	1,−1,−1,−1,1,1,1,1,−1,−1,−1,1,−1,−1,−1,1,1,−1,−1,−1,1,1,−1,−1,1,−1,1,1,1,−1,1
c_{48}	−1,−1,−1,1,−1,1,−1,1,−1,1,1,1,−1,1,−1,1,1,1,−1,1,1,−1,−1,−1,−1,1,1,1,1,1,−1, −1,1,1,−1,−1,1,−1,−1,1,1,1,1,1,1,1,1,1,−1,−1,1,1

Figure 30: The Weil Transform of the c_{14} Waveform

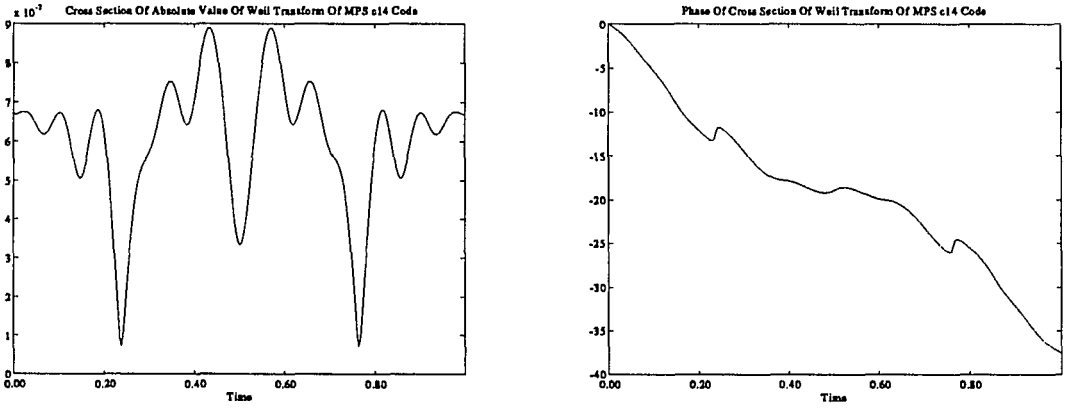


Figure 31: The Ambiguity Surface of the c_{14} Waveform

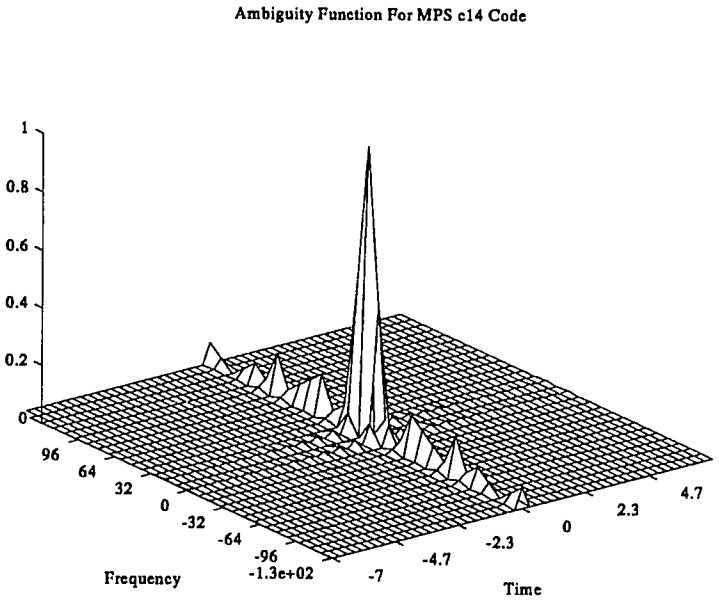


Figure 32: The Ambiguity Surface of the Doubled c_{14} Waveform

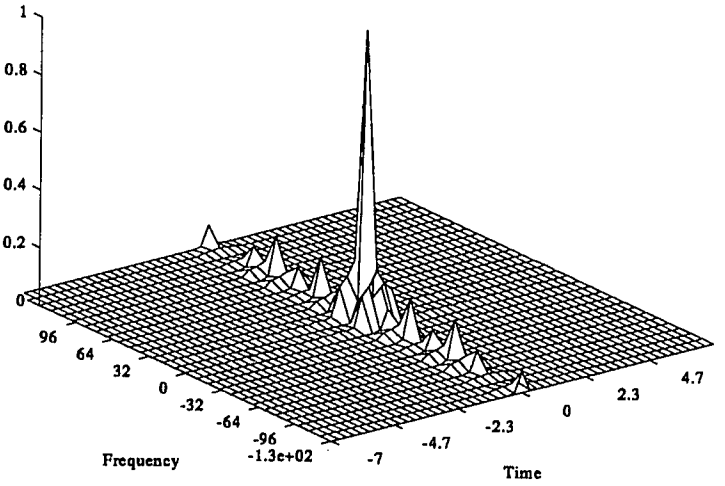


Figure 33: The Weil Transform of the c_{21} Waveform

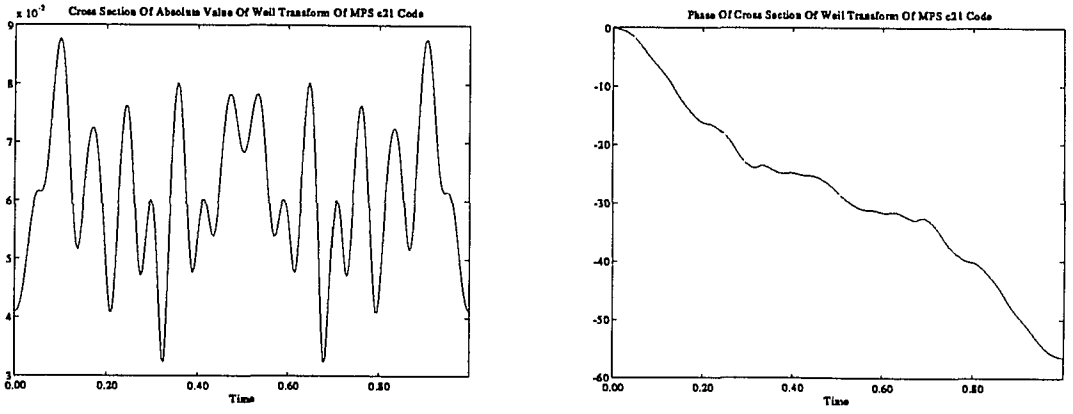


Figure 34: The Ambiguity Surface of the c_{21} Waveform

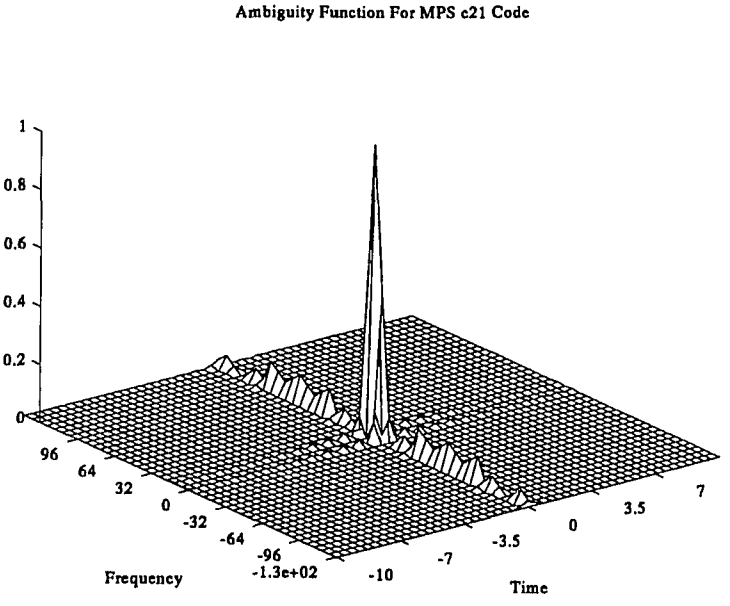


Figure 35: The Ambiguity Surface of the Doubled c_{21} Waveform

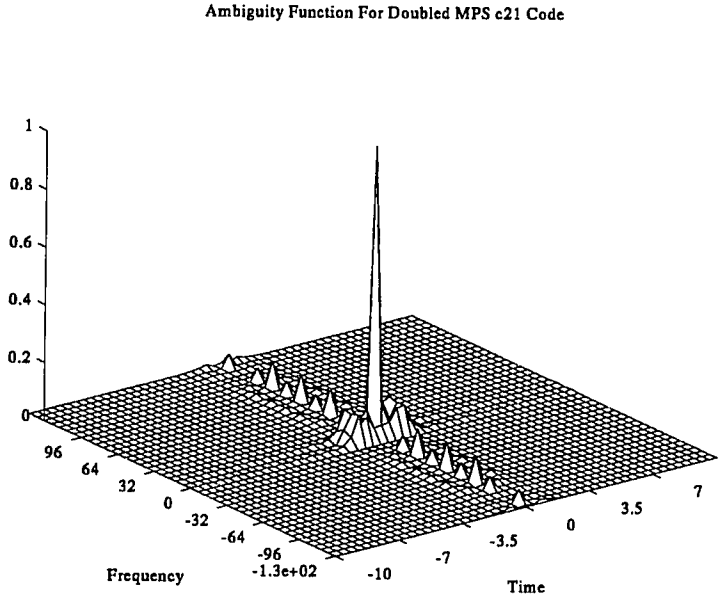


Figure 36: The Weil Transform of the c_{22} Waveform

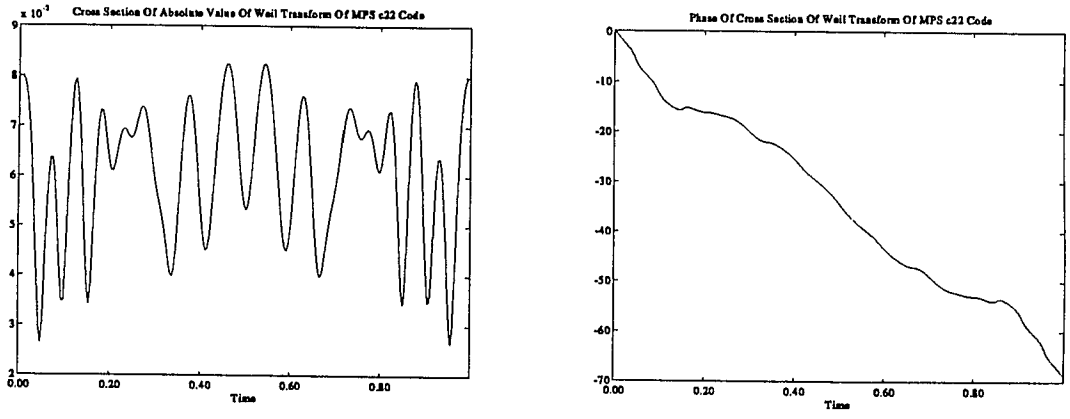


Figure 37: The Ambiguity Surface of the c_{22} Waveform

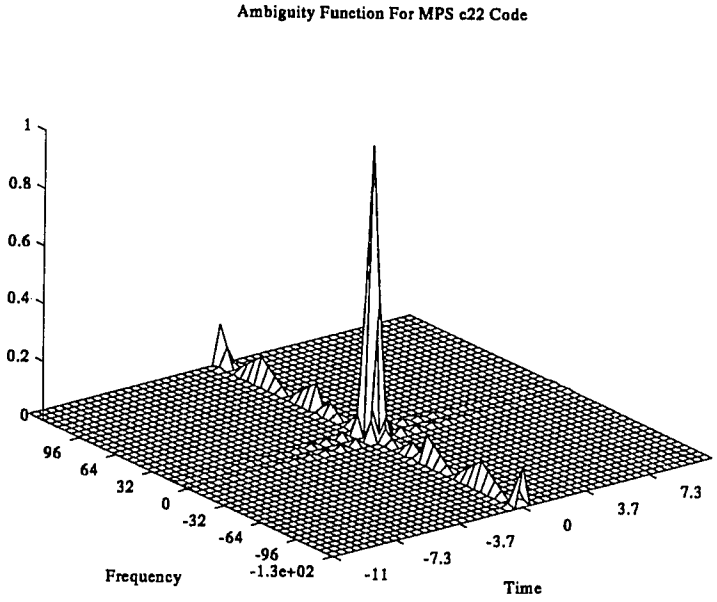


Figure 38: The Ambiguity Surface of the Doubled c_{22} Waveform

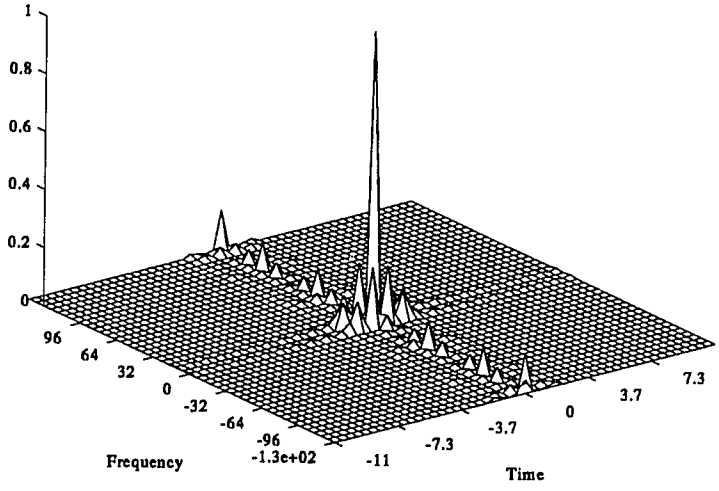


Figure 39: The Weil Transform of the c_{25} Waveform

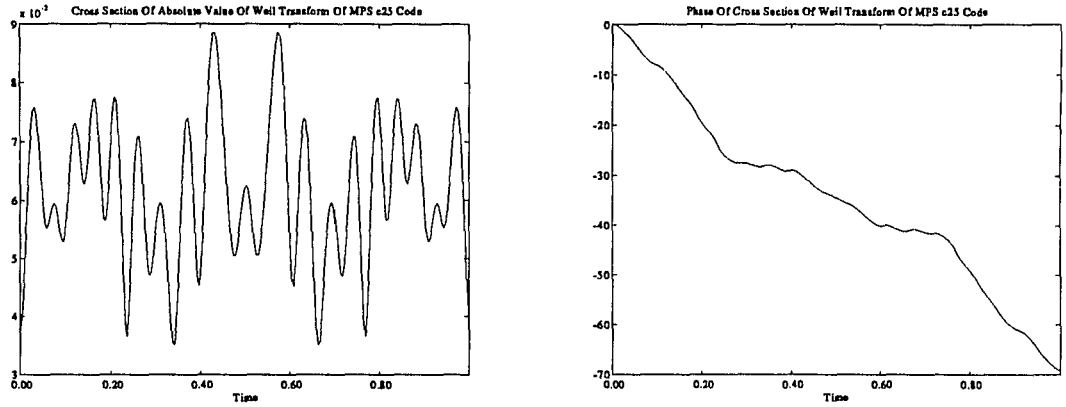


Figure 40: The Ambiguity Surface of the c_{25} Waveform

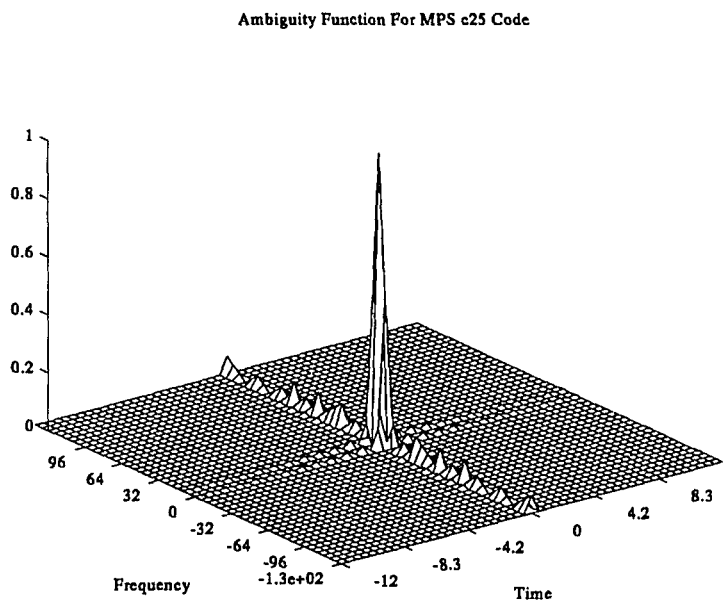


Figure 41: The Ambiguity Surface of the Doubled c_{25} Waveform

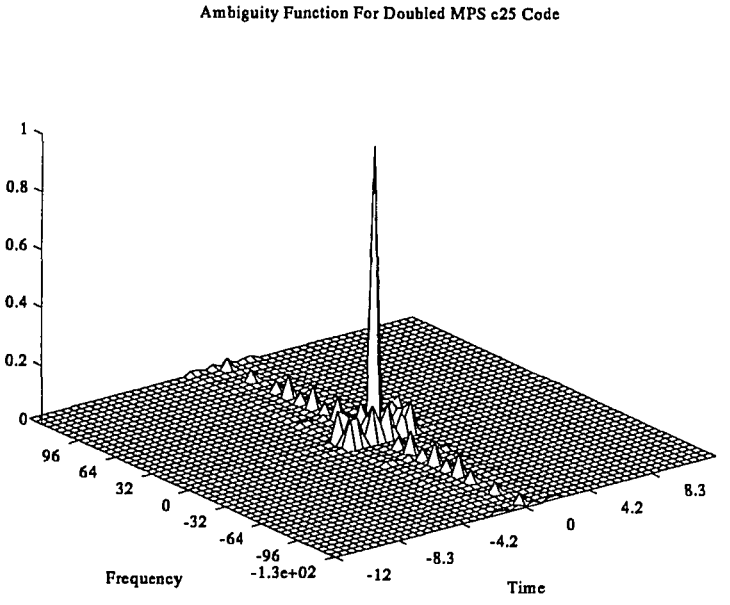
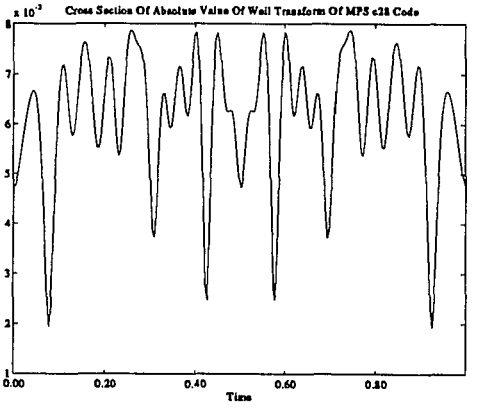


Figure 42: The Weil Transform of the c_{28} Waveform



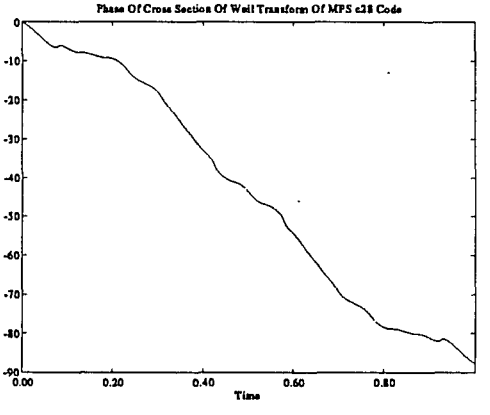


Figure 43: The Ambiguity Surface of the c_{28} Waveform

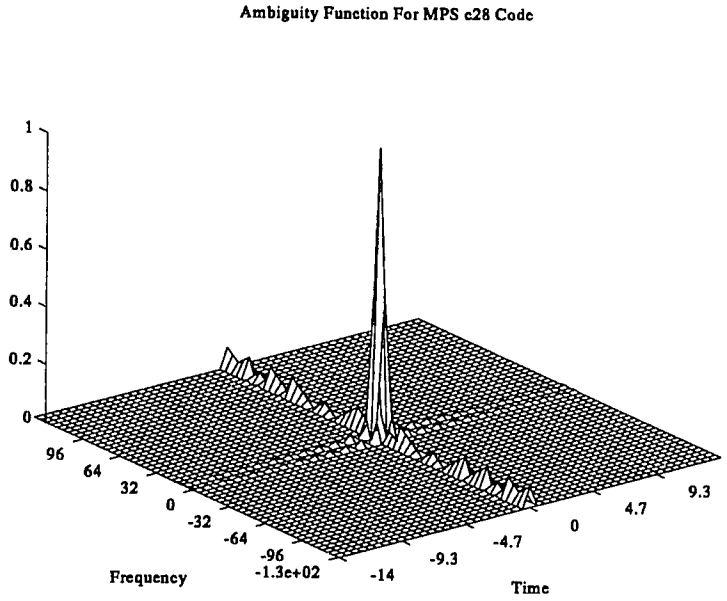


Figure 44: The Ambiguity Surface of the Doubled c_{28} Waveform

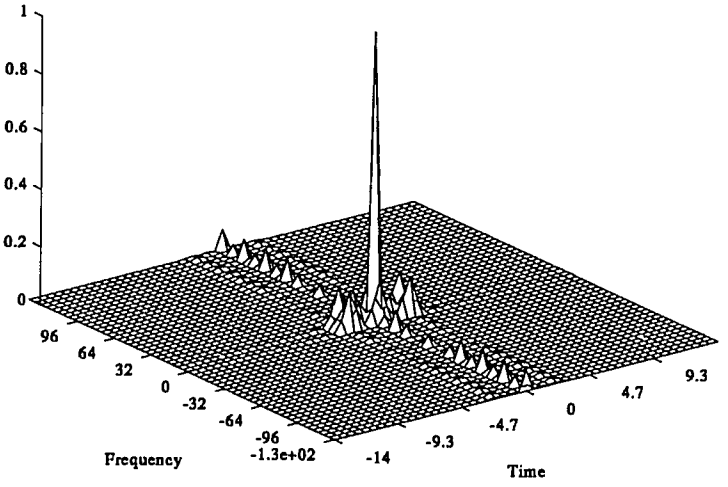


Figure 45: The Weil Transform of the c_{48} Waveform

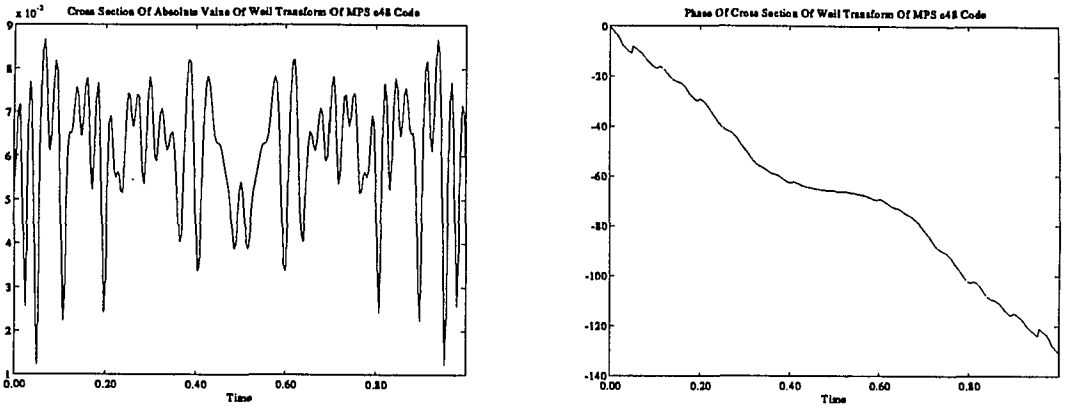


Figure 46: The Ambiguity Surface of the c_{48} Waveform

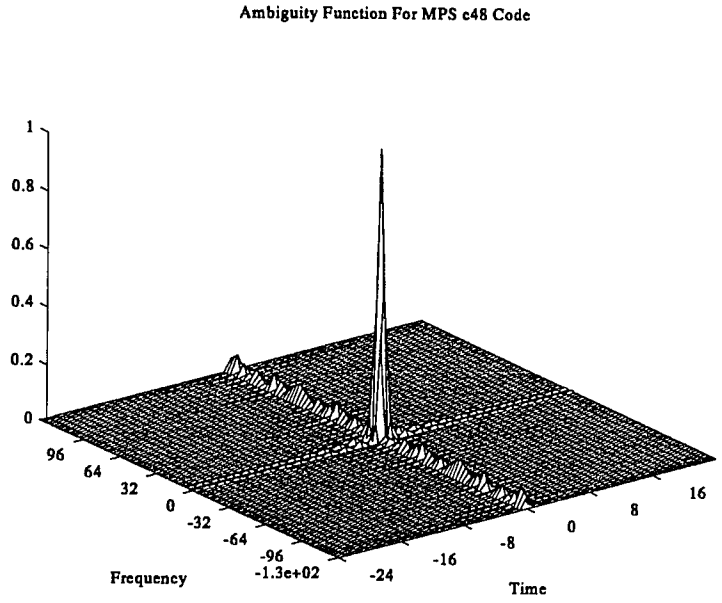
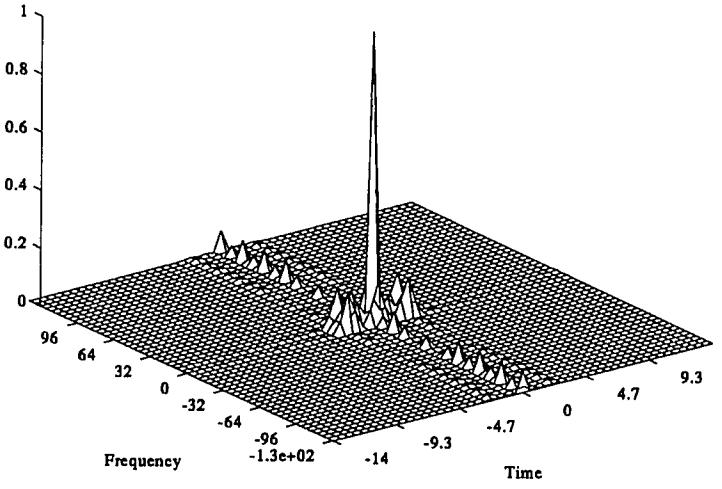


Figure 47: The Ambiguity Surface of the Doubled c_{48} Waveform



5.2.4 m-Sequences and Gold Codes

An m-sequence, or maximal length binary shift register sequence, is a type of binary phase code sequence that is often used in radar. m-sequences have desirable periodic correlation properties. A Gold code is just an m-sequence, thought of as a member of a family of related m-sequences, which have desirable cross-correlation properties. For details on m-sequences, the reader is referred to [19].

Let $h = \{h_0, h_1, h_2, \ldots, h_n\}$ be a vector in $\mathbf{Z}_2^{n+1} = (\mathbf{Z}/2\mathbf{Z})^{n+1}$, such that $h_n = h_0 = 1$. A binary sequence $a : \mathbf{Z} \to \mathbf{Z}_2$ is said to be a *sequence generated by* h if for all $i \in \mathbf{Z}$,

$$\langle h, \{a_j, a_{j-1}, \ldots, a_{j-n}\}\rangle \equiv 0 \pmod 2.$$

Since $h_0 = 1$, this says that

$$a_{j+n} = \sum_{k=0}^{n-1} h_{n-k} a_{j+k},$$

with arithmetic (mod 2).

The above formula is the reason for the terminology "binary shift register" since it corresponds to a device which generates terms in a sequence by shifting pervious terms of the sequence through the process of adding together a spatially fixed subset, mod 2.

It is a fact that any sequence generated by a vector in $\mathbf{Z}_2^{n+1}$ will be periodic, with period at most $2^n - 1$. A sequence with this maximal period is called a maximal length binary shift register sequence or m-sequence.

It is known that m-sequences have discrete periodic auto-correlation functions that are like Barker codes in the non-periodic case. Specifically, if a is an m-sequence of length N, and we define

$$\mathcal{A}_a(k) = \sum_{i=0}^{N-1} (-1)^{a_i}(-1)^{a_{i+k}},$$

then $\mathcal{A}_a(0) = N$, and $\mathcal{A}_a(k) = -1$, for $k = 1, \ldots, N-1$.

A preferred pair of m-sequences is a pair of m-sequences a, and b, of length $2^n - 1$, for which the periodic cross-correlation function

$$A_{a,b}(k) = \sum_{i=0}^{N-1} (-1)^{a_i}(-1)^{b_{i+k}},$$

takes on only the three values -1, $-(1 + 2^{\lfloor (n+2)/2 \rfloor})$ and $2^{\lfloor (n+2)/2 \rfloor} - 1$. Define the shift operator σ_j by $(\sigma_j(a))_i = a_{i+j}$. For any preferred pair of sequence a, b, a *Gold Code Family* is given by the family of m-sequences

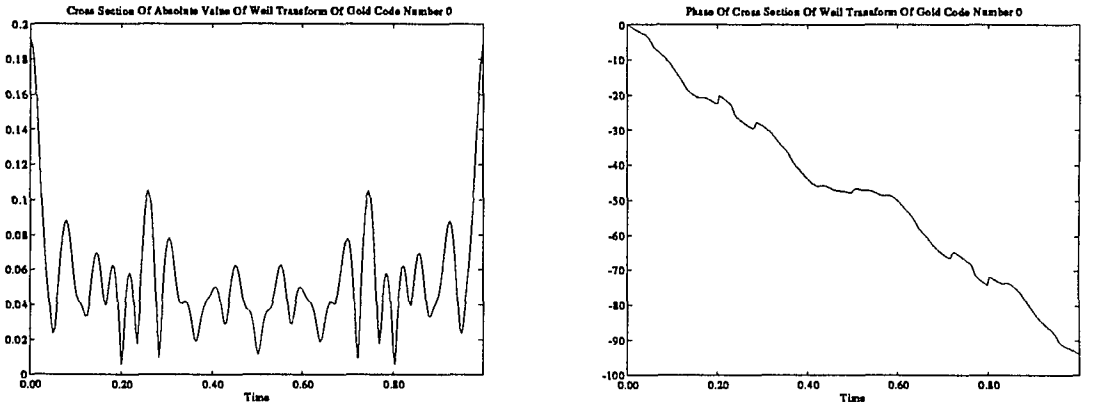
$$G(a,b) = \{a, b, a+b, a+\sigma_1(b), \ldots, a+\sigma_{2^n-1}(b)\},$$

where addition is in the group of binary sequences, Z_2^ω, which is a countable product of copies of $\mathbf{Z}_2$.

Figures 48–77 show, for a sequence of 10 Gold codes in a certain family, the Weil transform of the corresponding waveform, the ambiguity surface, and the doubled waveform ambiguity surface, as in the previous sections.

The particular family depicted comes from the Gold code sequence corresponding to $a = \{-1,1,1,-1,-1,-1,-1,1,1,1,-1,-1,1,1,-1,1,1,1,1,1,1,-1,1,-1,-1,-1,1,-1,-1,1,-1,1\}$, and b a decimation of a by 5. Hence a and b are a preferred pair, by a result of Gold, as cited

Figure 48: The Weil Transform of Gold Code 0



in [28]. Although this Gold code family contains 33 codes, only 10 are depicted, to give the flavor of the graphs without taking up more space.

Figure 49: The Ambiguity Surface of Gold Code 0

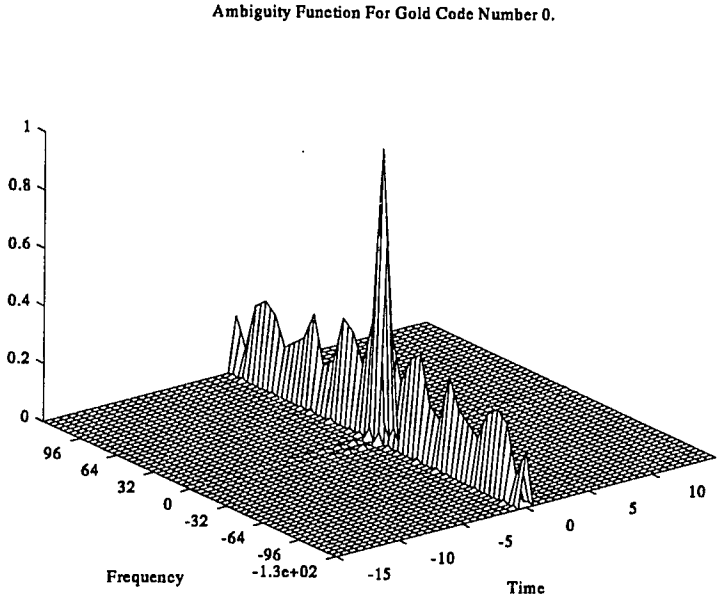


Figure 50: The Ambiguity Surface of Doubled Gold Code 0

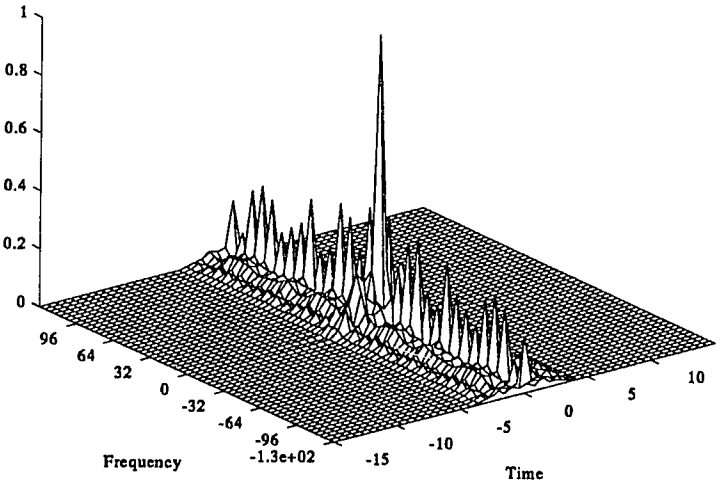


Figure 51: The Weil Transform of Gold Code 1

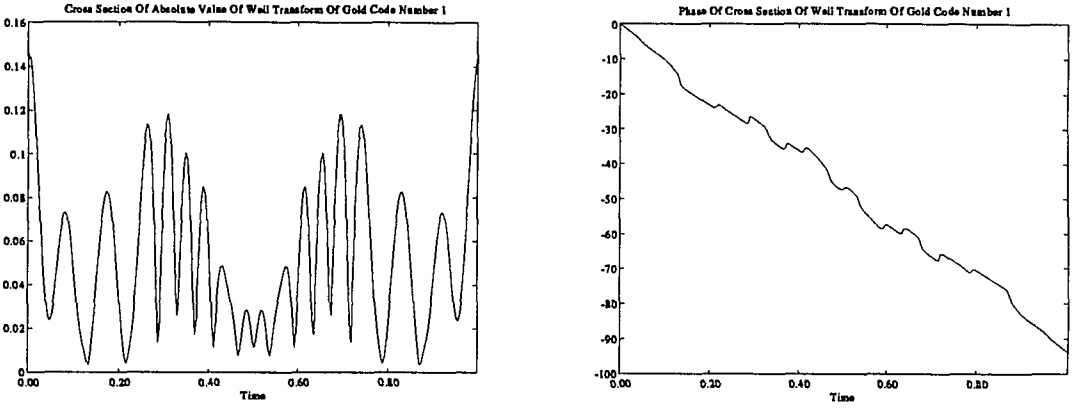


Figure 52: The Ambiguity Surface of Gold Code 1

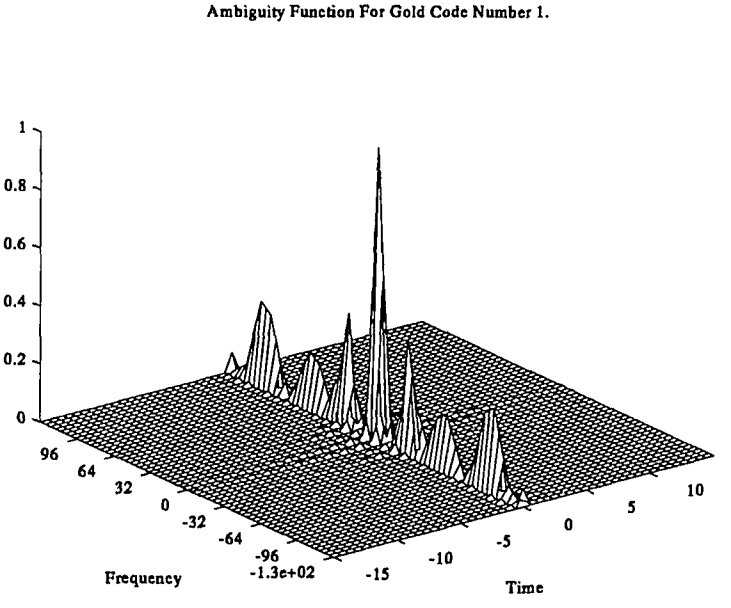


Figure 53: The Ambiguity Surface of Doubled Gold Code 1

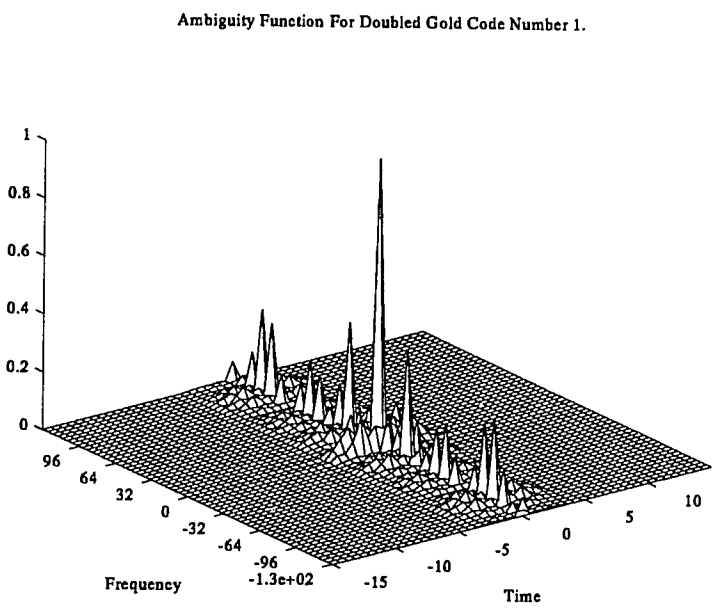


Figure 54: The Weil Transform of Gold Code 2

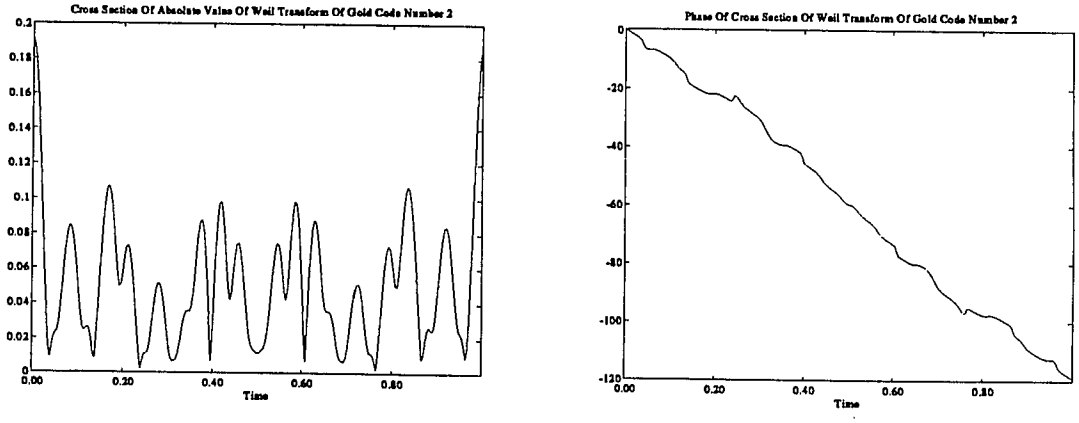


Figure 55: The Ambiguity Surface of Gold Code 2

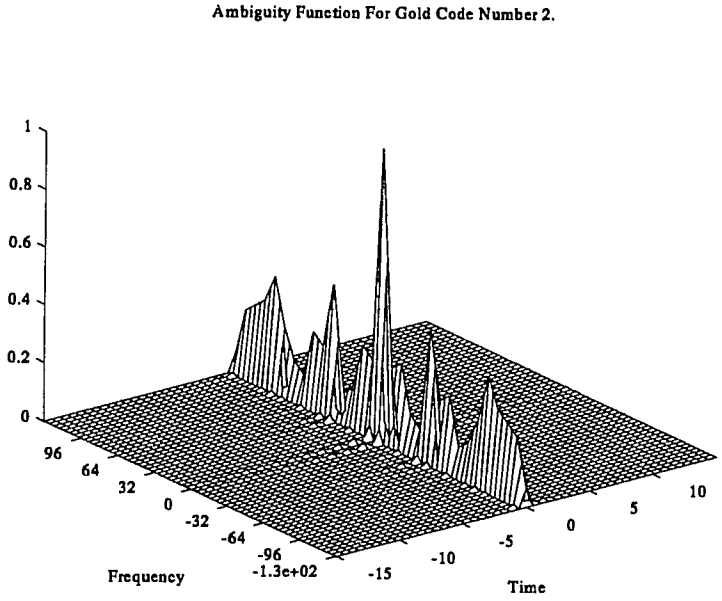


Figure 56: The Ambiguity Surface of Doubled Gold Code 2

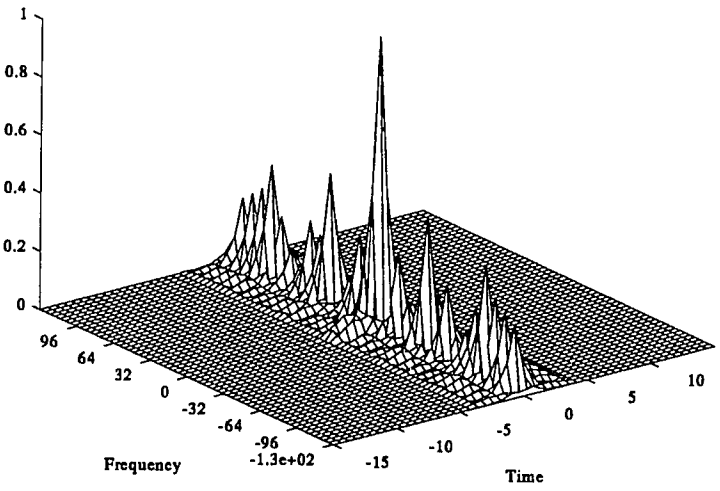


Figure 57: The Weil Transform of Gold Code 3

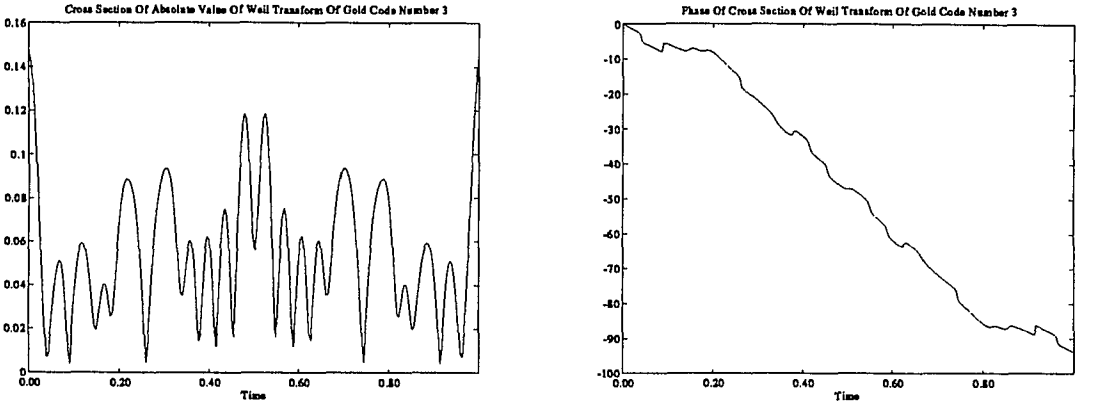


Figure 58: The Ambiguity Surface of Gold Code 3

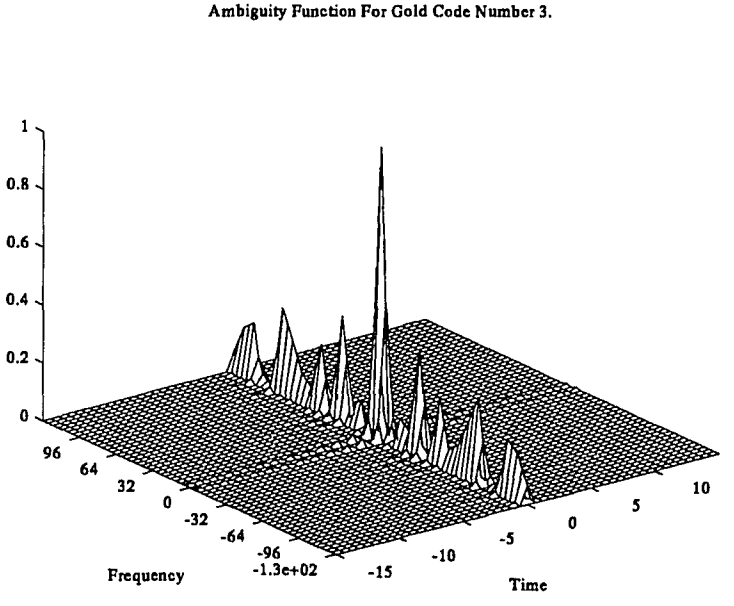


Figure 59: The Ambiguity Surface of Doubled Gold Code 3

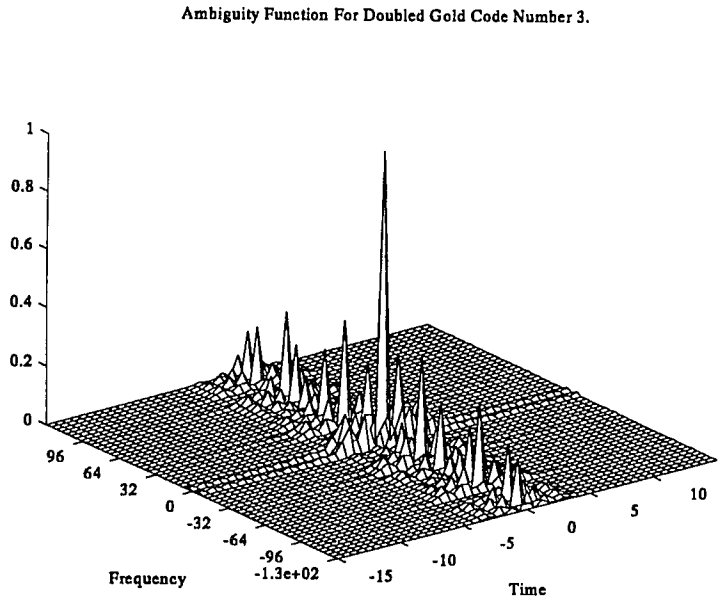


Figure 60: The Weil Transform of Gold Code 4

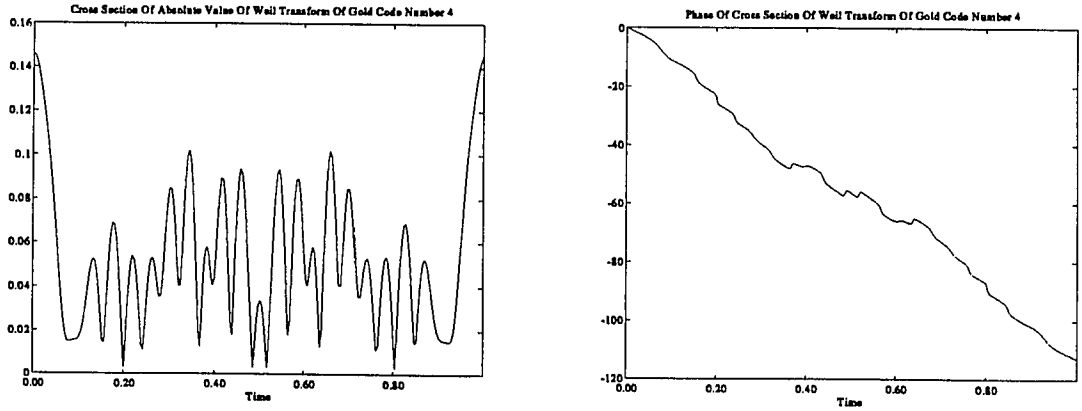


Figure 61: The Ambiguity Surface of Gold Code 4

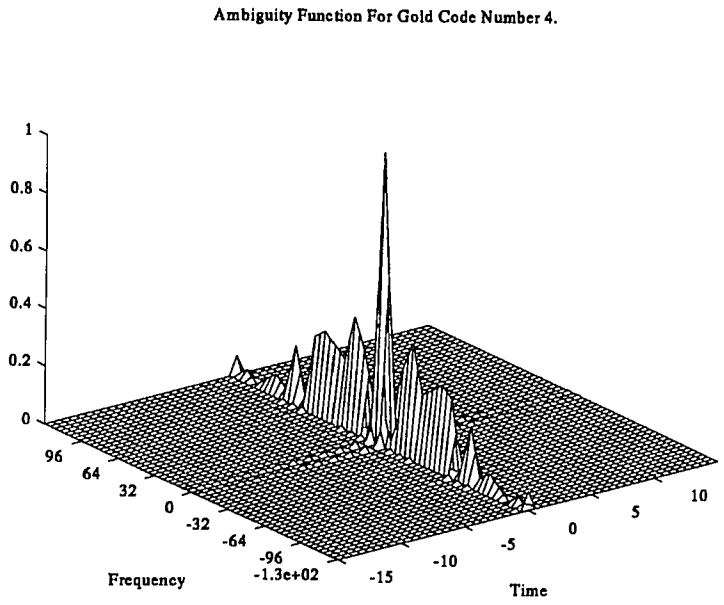


Figure 62: The Ambiguity Surface of Doubled Gold Code 4

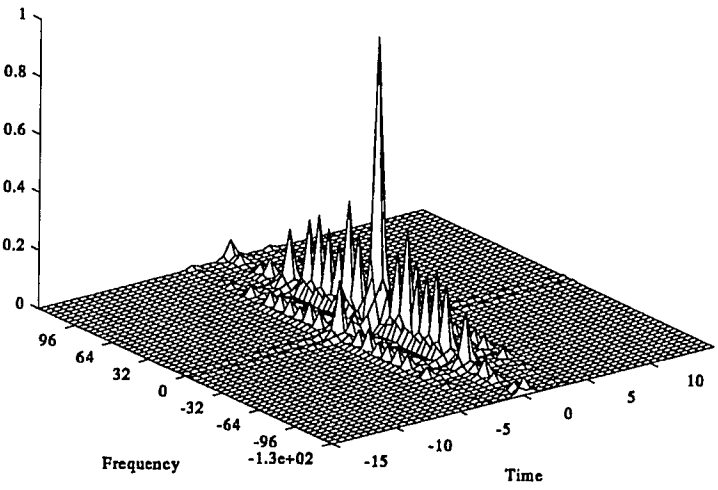


Figure 63: The Weil Transform of Gold Code 5

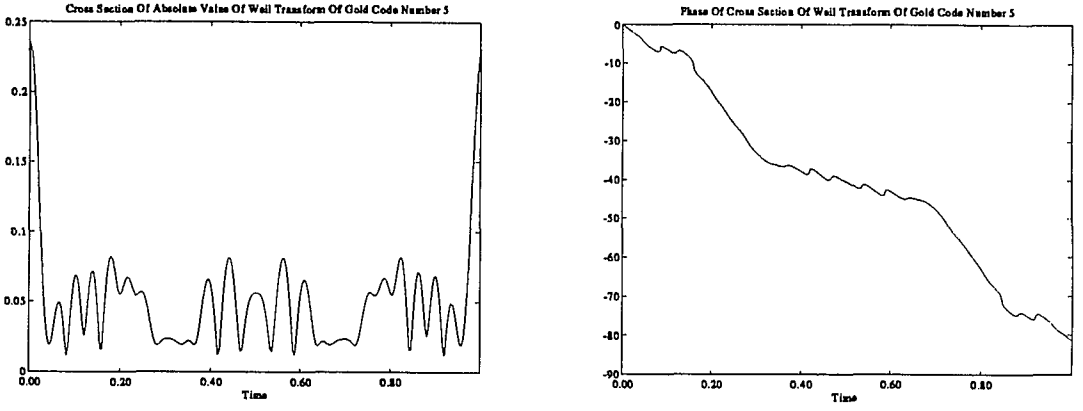


Figure 64: The Ambiguity Surface of Gold Code 5

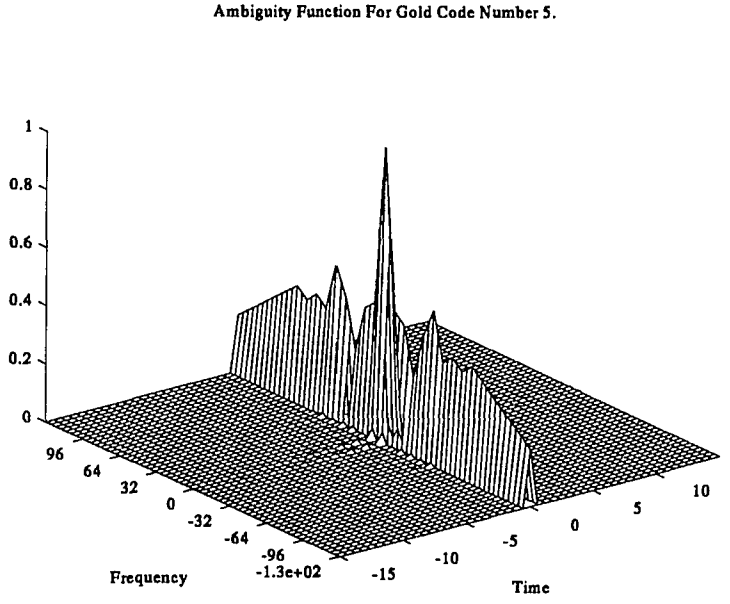


Figure 65: The Ambiguity Surface of Doubled Gold Code 5

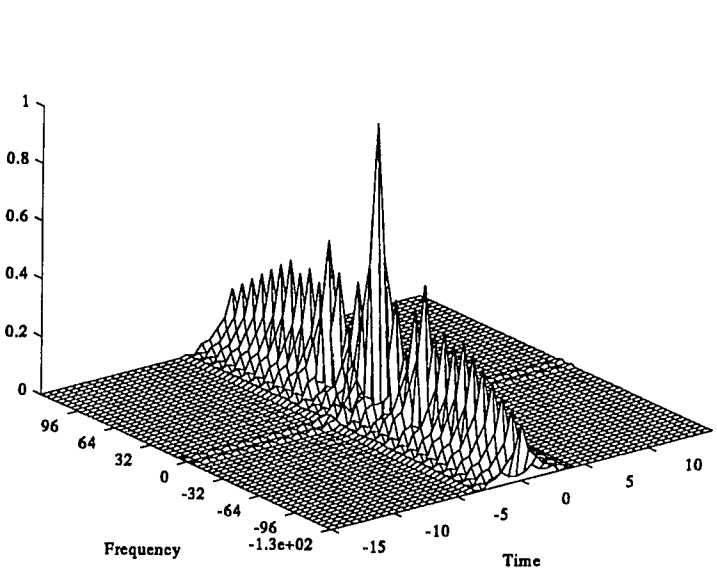


Figure 66: The Weil Transform of Gold Code 6

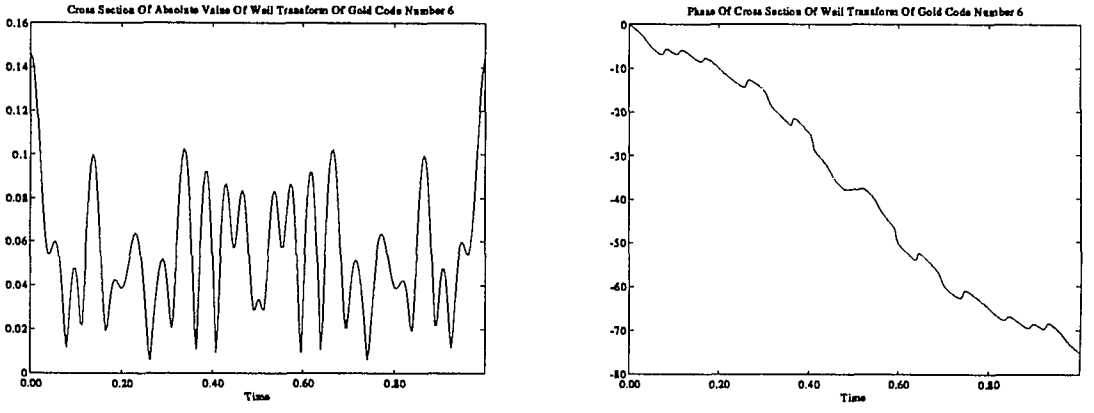


Figure 67: The Ambiguity Surface of Gold Code 6

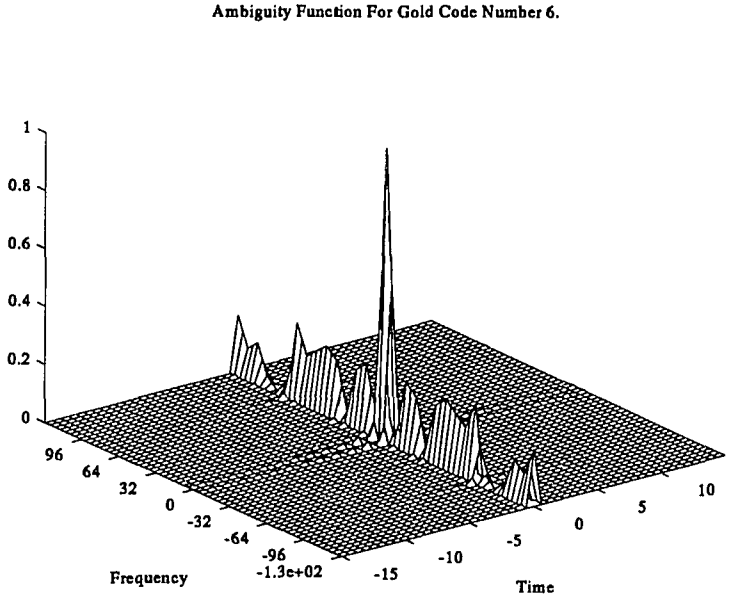


Figure 68: The Ambiguity Surface of Doubled Gold Code 6

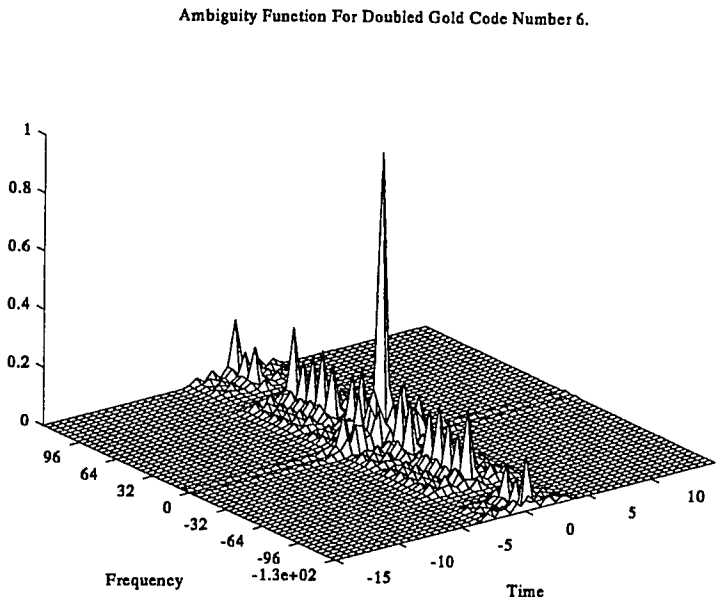


Figure 69: The Weil Transform of Gold Code 7

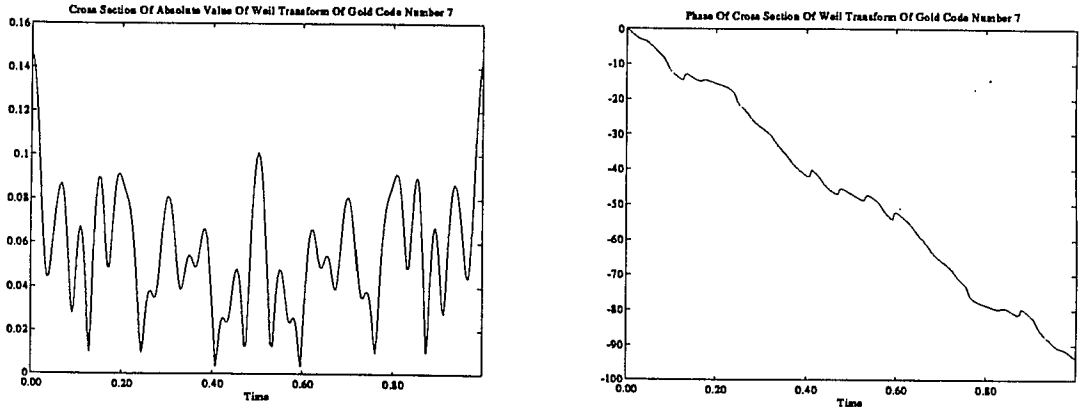


Figure 70: The Ambiguity Surface of Gold Code 7

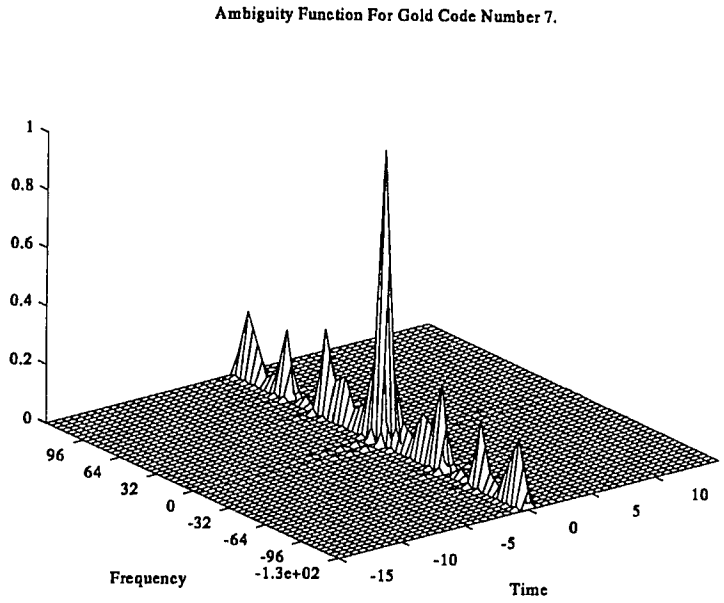


Figure 71: The Ambiguity Surface of Doubled Gold Code 7

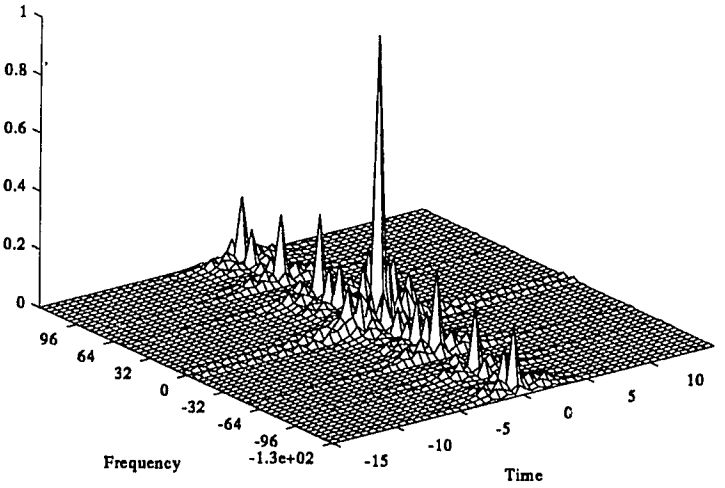


Figure 72: The Weil Transform of Gold Code 8

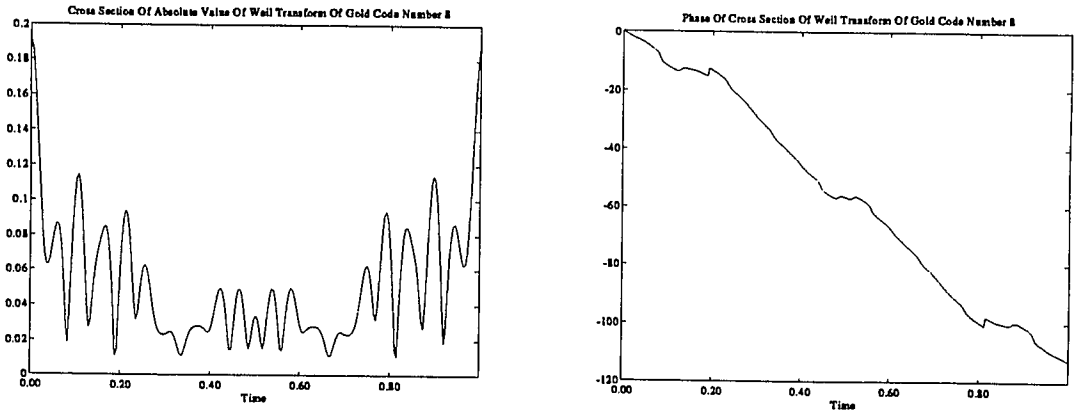


Figure 73: The Ambiguity Surface of Gold Code 8

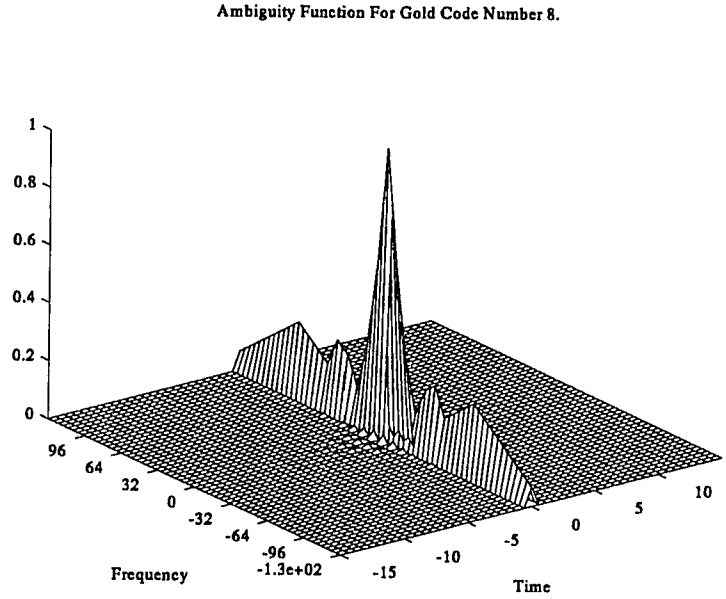


Figure 74: The Ambiguity Surface of Doubled Gold Code 8

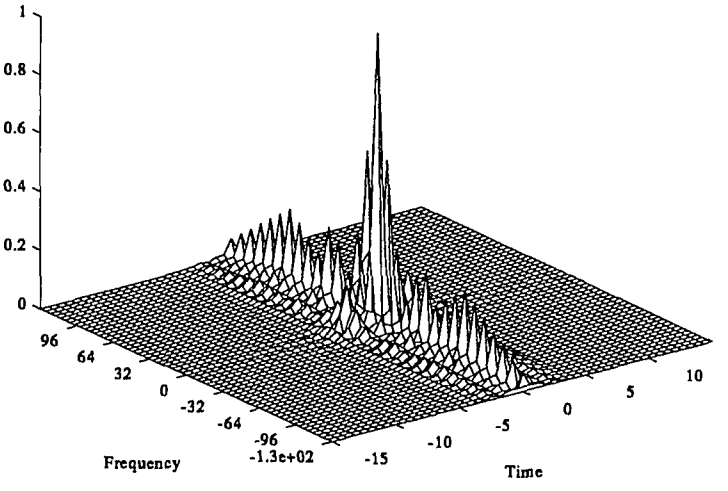
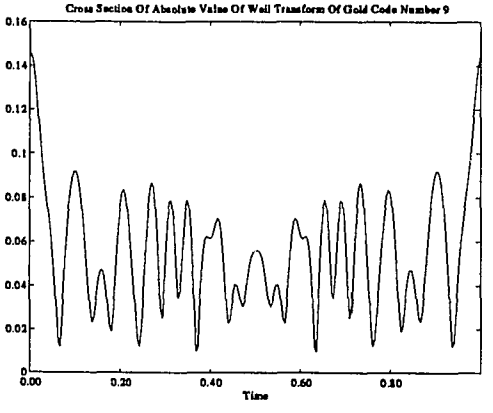


Figure 75: The Weil Transform of Gold Code 9



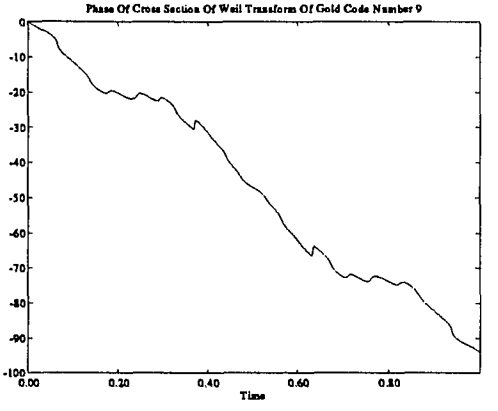


Figure 76: The Ambiguity Surface of Gold Code 9

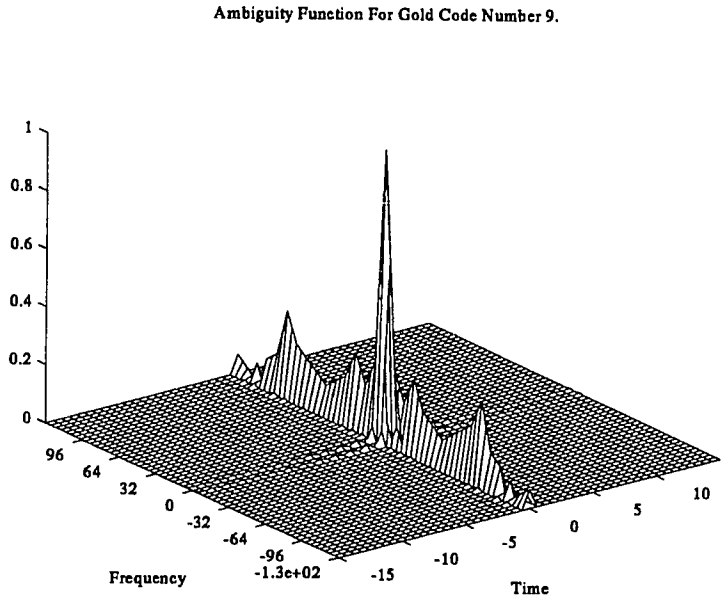
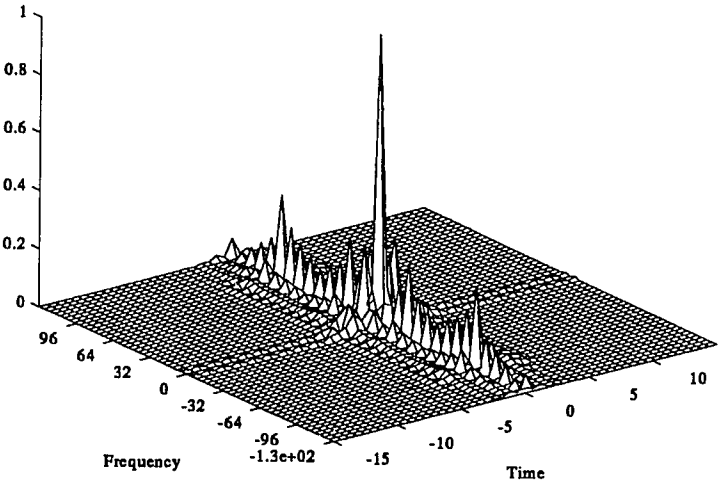


Figure 77: The Ambiguity Surface of Doubled Gold Code 9



5.3 Frequency Hop Coding

Frequency shift, or frequency hop coding corresponds to waveform shaping of the following kind.

$$s(t) = \sum_{n=0}^{N} p(t-n)e^{2\pi i \theta_n t},$$

where θ_n is an integer depending on n. Frequently, one takes $\theta_1, \theta_2, \ldots, \theta_n$ to be a permutation of the integers $1, 2, \ldots, n$. Frequency hop coding is waveform shaping with $a_{m,n} = 1$ for at most one n for each m, and $a_{m,n} = 0$ otherwise.

5.3.1 Costas Arrays

Costas [13] suggested that, under the scheme of frequency hop coding, if we choose the permutation θ cleverly, we can get a waveform with good thumbtack-like ambiguity properties. Discussions beyond the original paper on Costas arrays can be found in [9], [21], [22].

To design a Costas array, θ is chosen so that no two pairs of terms in the sum will have the same shift in both time and frequency. If we take the representation of the permutation group on n symbols, S_n, on $\mathbf{R}^n$ by permutations of the basis vectors, then any permutation corresponds to a matrix of ones and zeros, with exactly one 1 in each row and column. The definition of a Costas array can be stated in terms of the two dimensional, discrete, non-periodic correlation function of the matrix for the permutation θ, and it become completely analogous to the

definition of a Barker code. It is because Costas made his original definition in terms of these matrices that the term Costas *array* is used.

Let $(\theta_{i,j})$ be the permutation matrix of θ. If we define

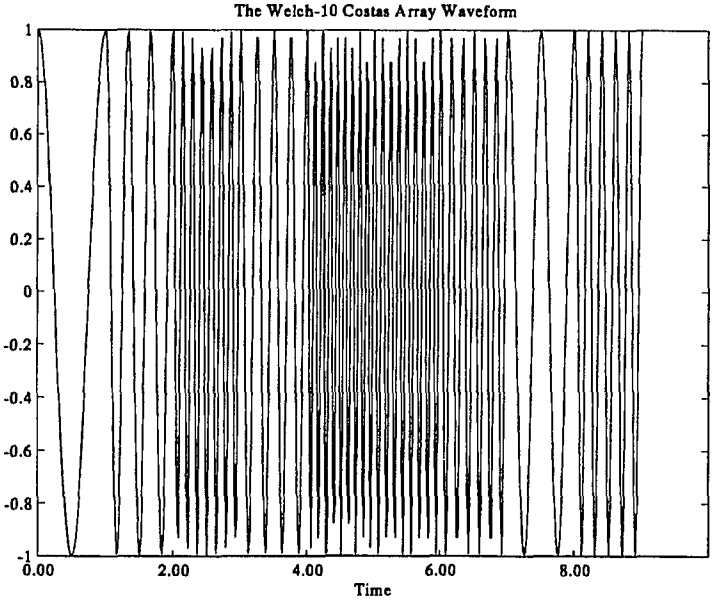
$$A_\theta(a,b) = \sum_{c=1}^{n-a} \sum_{d=1}^{n-b} \theta_{c,d}\theta_{c+a,d+b},$$

then the condition that θ be a Costas array is that $A_\theta(a,b) \leq 1$ for all $0 \leq a, b \leq n$, $(a,b) \neq (0,0)$.

Since the array for θ is, in some sense, a pattern in time-frequency space, the idea is really one of time-frequency staggering. But, since there is no time-frequency space in which the pattern really resides, this is only an intuitive view in this case. The time-frequency staggering point of view, in this case, goes as follows. As $s(t)$ is shifted around in time and frequency, and compared with an unshifted $s(t)$, we see that all terms overlap at $(0,0)$. This accounts for the main lobe. Also, one pair of terms overlap at many places. This "uses up" the side lobe energy in a way which produces a side lobe only $\frac{1}{N}$ as high as the main lobe. Finally, two pairs never overlap, so there are never bigger side lobes.

The Welch-10 and Welch-30 codes give examples of Costas arrays. The Weil transforms of these codes have interesting properties. They seem to have an approximate identity at $(0,0)$, which should account for a sharp main lobe, using the idea of small support in the coset space. This is also of interest in view of the fact that in this case the Weil transform approximates the ambiguity function, near

Figure 78: Signal Corresponding to the Welch-10 Costas Array



$(0,0)$ (see section 4.5). They also have a pedestal region, which somehow kills the side lobes. There are many zeros, with winding numbers $+1$, and -1, in a pattern that gives an example of time-frequency staggering. Finally, these functions have the appearance of an approximate frame, thus controlling ambiguity on the integer lattice.

The above reinforces the idea of looking for other Weil transformed functions with these properties.

Figure 79: The Weil Transform of the Welch-10 Waveform

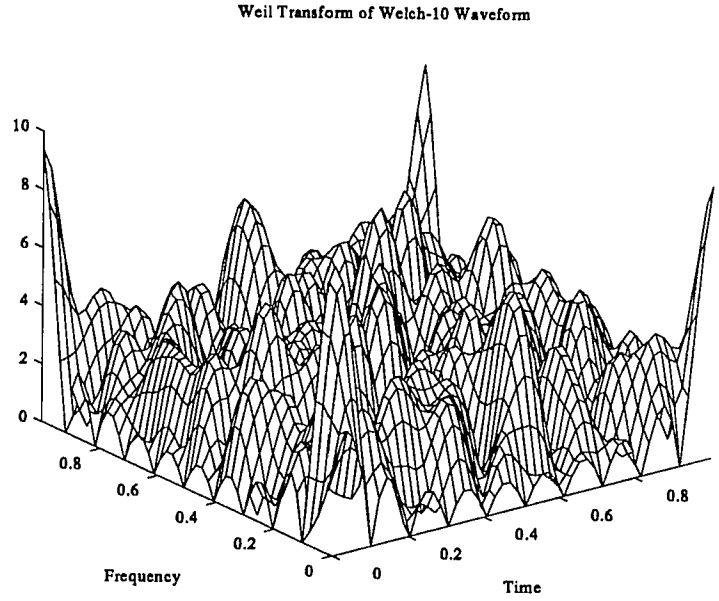


Figure 80: The Winding Number Data for the Welch-10 Waveform

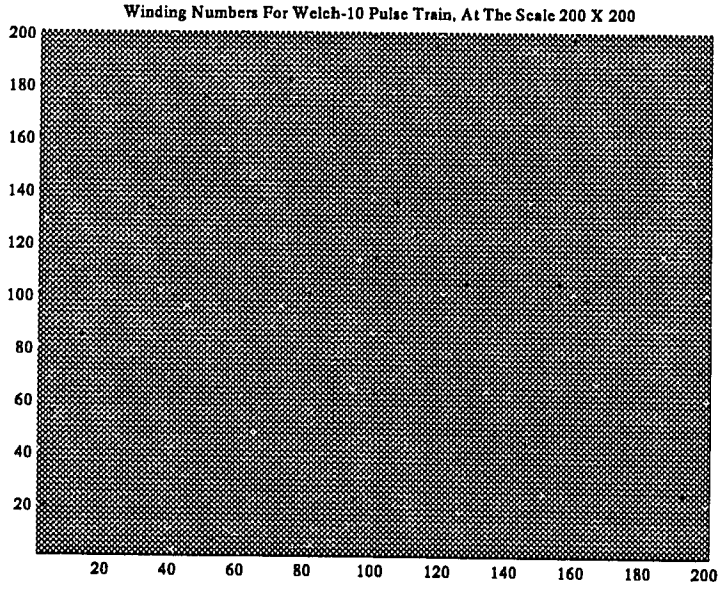


Figure 81: The Ambiguity Surface of the Welch-10 Waveform

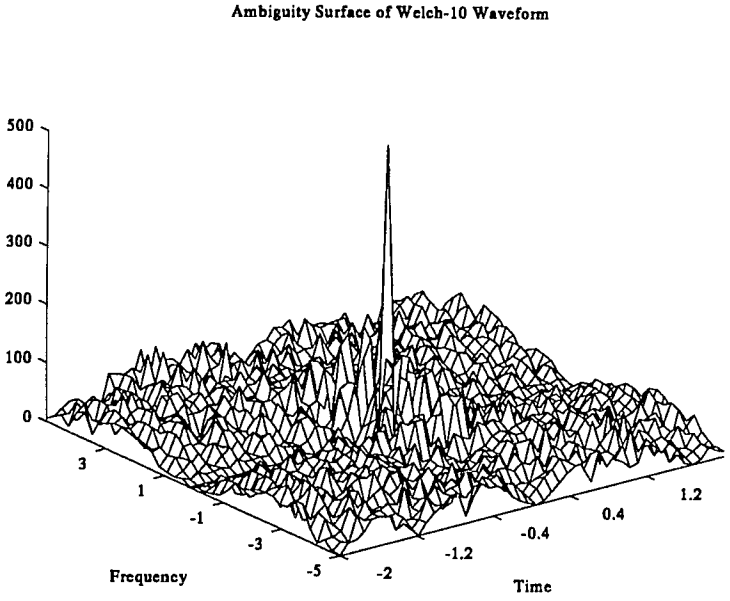


Figure 82: The Weil Transform of the Welch-30 Waveform

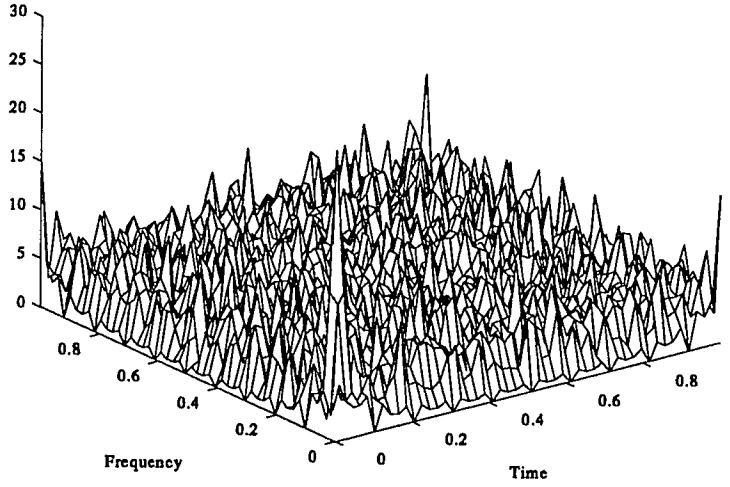


Figure 83: The Winding Number Data for the Welch-30 Waveform

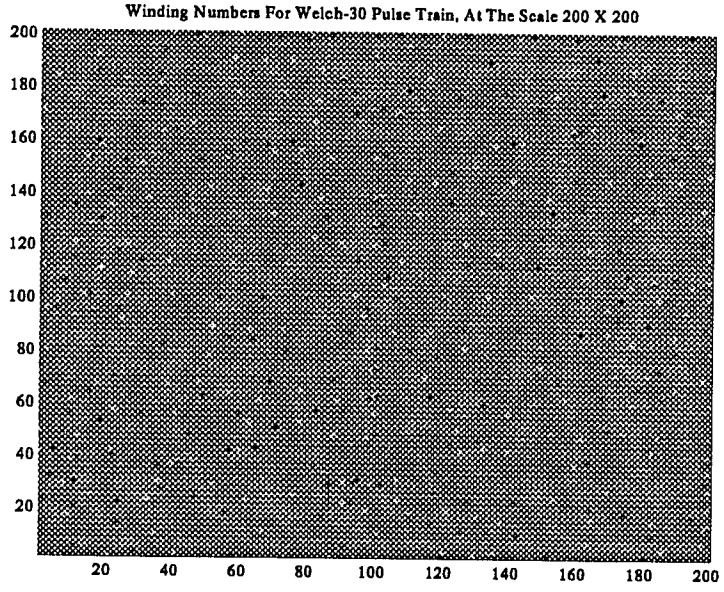
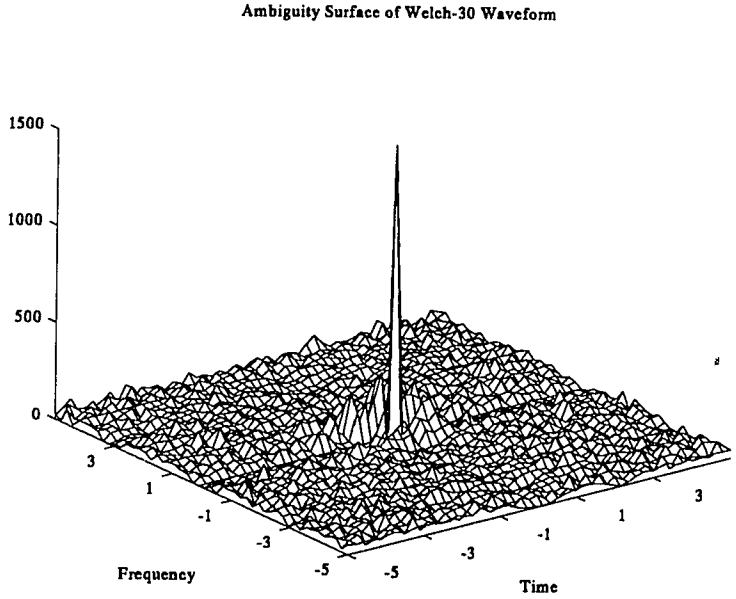


Figure 84: The Ambiguity Surface of the Welch-30 Waveform



6 New Thumbtack Constructions

The techniques for waveform construction described in section 5, were all based on combinatorial constructions, and therefore have the drawback that one must find solutions to hard combinatorial problems before waveforms can be constructed.

A theme of the survey above has been that the thumbtack nature of the waveforms in question can be accounted for by the structure of the Weil transform of the waveforms. The advantage of this is that the complication of finding combinatorial patterns disappears; one simply has to construct functions in H_1 with desired geometric properties.

In this section, we will look at two examples of how this can be done.

6.1 A New Generalization of Barker Code Waveforms

As a first example of the philosophy stated above, if one looks at table 1, it is not at all obvious what these codes have in common. In fact, the only known way to find them is by searching through all possible binary codes [10]. However, a quick glance at the graphs of the Weil transforms of these waveforms reveals a pattern, especially if the b_5 and b_{13} graphs are inverted. Consider the following observations about these graphs:

The phase of these Weil transforms are nearly linear. Hence, by demodulation, they can be made nearly flat. For this reason, we will ignore phase.

The left half of the absolute value of these Weil transforms looks like a band-limited approximation to a square wave of duration $\frac{1}{2}$. The right half is symmetrically identical. A function which has a band-limited approximation with this property is a square wave of height 1, and duration $\frac{1}{2}$, with the constant $\frac{1}{2}$ subtracted. That is, the function, defined for $t \in [0,1]$, by:

$$f(t) = \begin{cases} \frac{1}{2} & t \leq \frac{1}{2}, \\ -\frac{1}{2} & t > \frac{1}{2}. \end{cases}$$

The Fourier series coefficients of $f = \sum_{n=-\infty}^{\infty} a_n e^{2\pi i n t}$ are

$$a_n = \begin{cases} 0 & n \text{ even or } 0, \\ \frac{1}{\pi i n} & n \text{ odd}. \end{cases}$$

Hence, if we take the band-limited functions

$$F_N(y) = \sum_{n=-N}^{N} a_n e^{2\pi i n t},$$

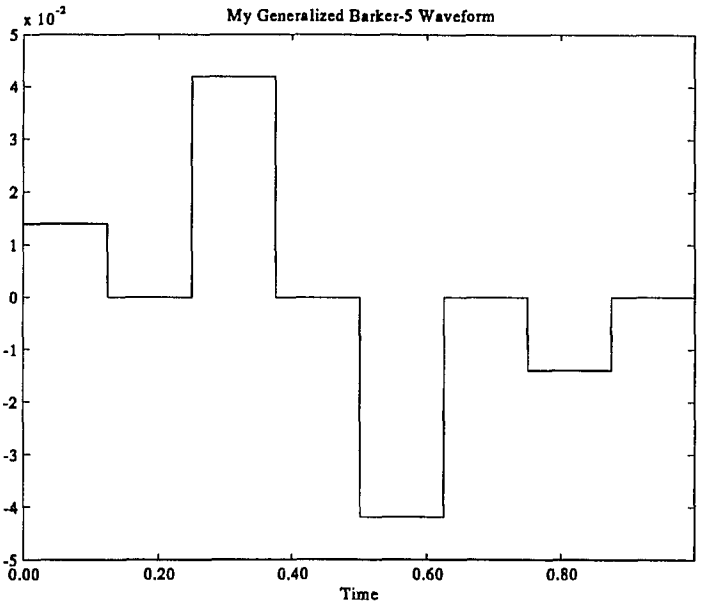
we arrive at functions which look a lot like the Weil transforms of Barker codes. Examples of waveforms corresponding to these constructed Weil transforms, are given for $N = 5, 10, 15$, and 20.

Note that the waveforms corresponding to these constructions are given by

$$w_N(t) = \sum_{n=-N}^{N} a_n p(t-n),$$

where $p(t)$ is a rectangular pulse of duration one, and a_n is as above. It is perhaps interesting to notice that when the functions are re-scaled so that their domains are all $[0,1]$, we get a sequence which converges to the distributional derivative of the delta function at the origin.

Figure 85: The Waveform w_5



By examining the ambiguity surfaces of these new waveforms, it is seen that we get an infinite family of waveforms which have the correlation properties of the Barker codes. Note that although the ambiguity surfaces of these new waveforms have a flare at the ends, it is still the case that the sidelobes are only a fraction of the height of the main lobe.

Figure 86: F_5, The Weil Transform of w_5

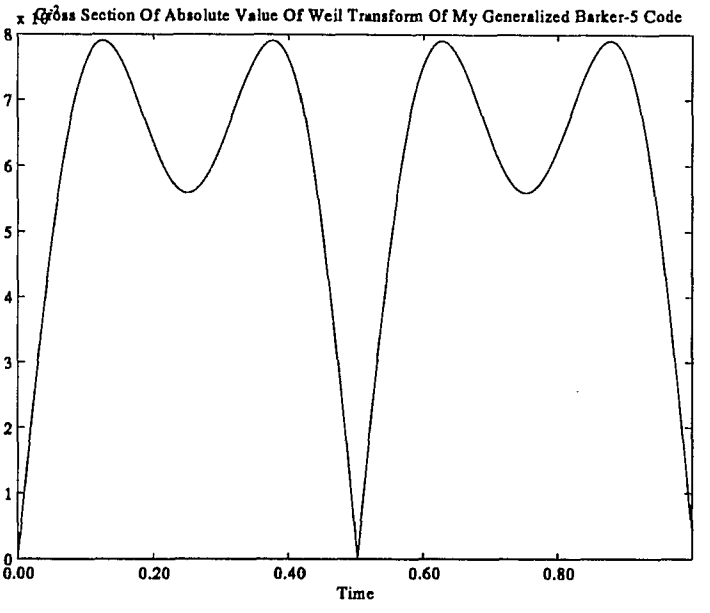


Figure 87: The Ambiguity Surface of w_5

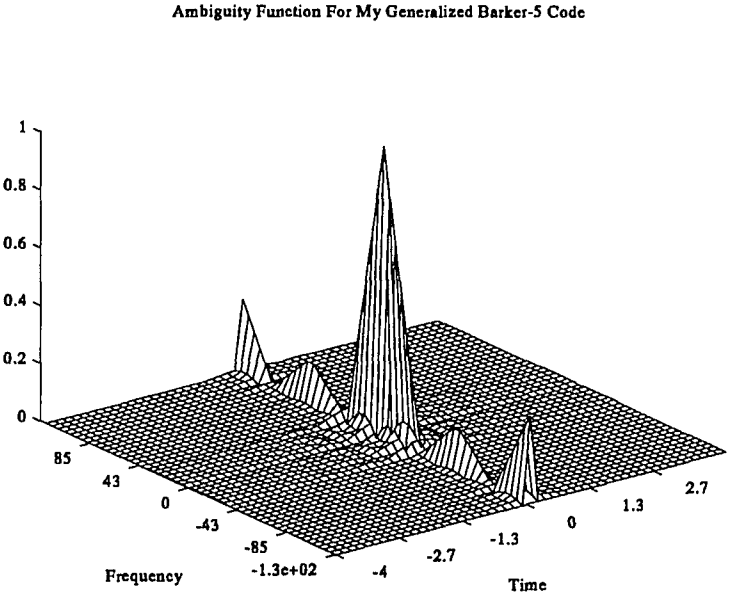


Figure 88: The Waveform w_{10}

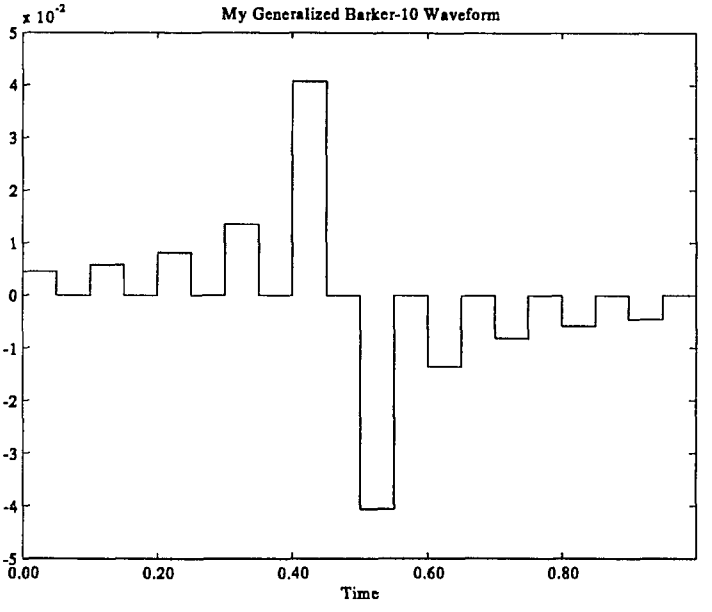


Figure 89: F_{10}, The Weil Transform of w_{10}

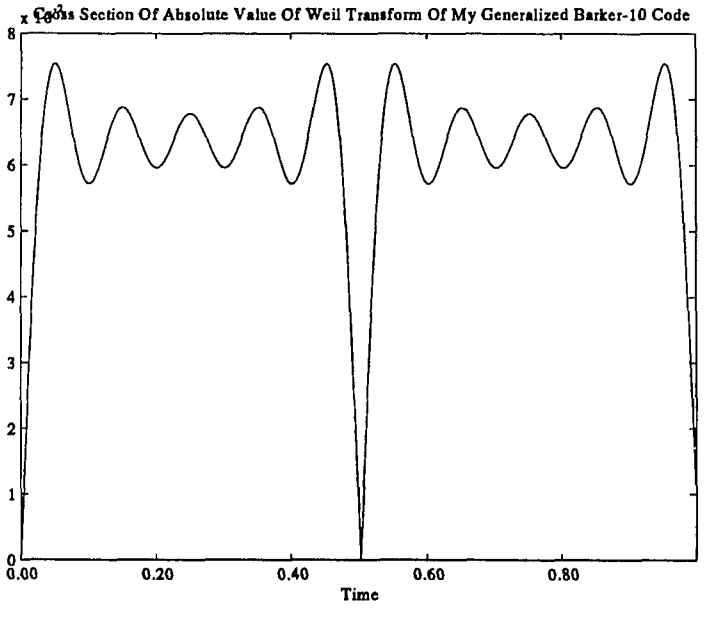


Figure 90: The Ambiguity Surface of w_{10}

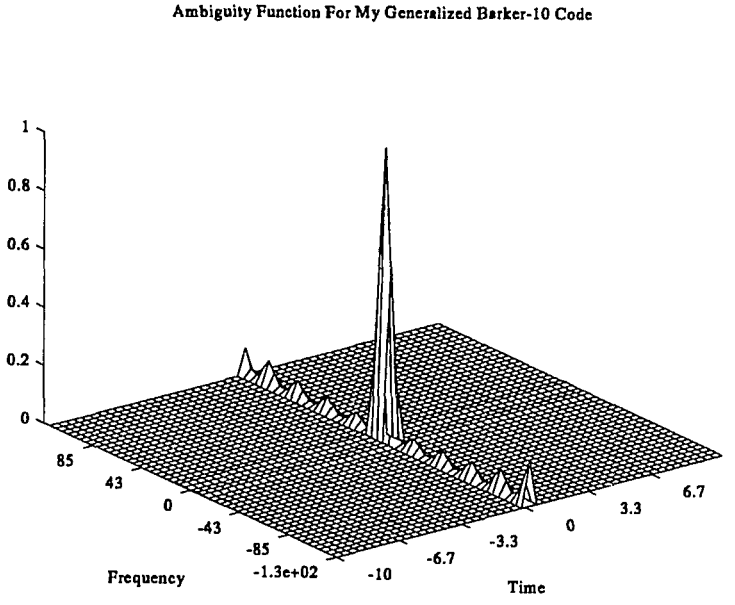


Figure 91: The Waveform w_{15}

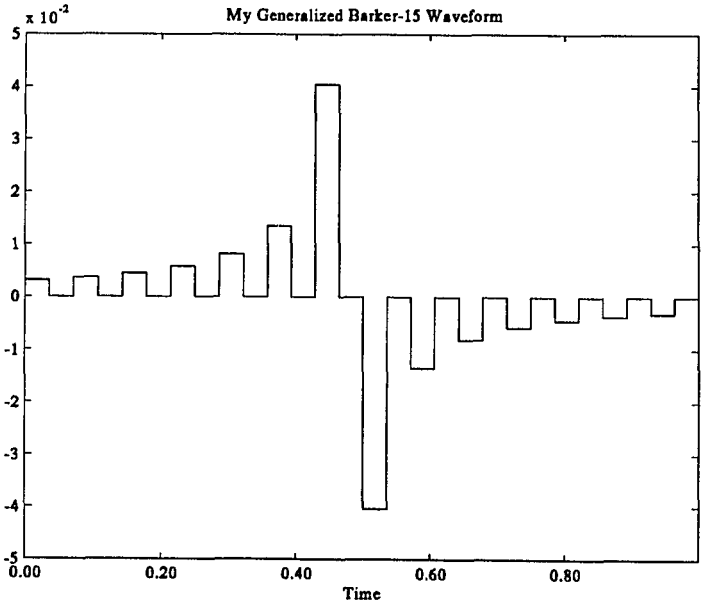


Figure 92: F_{15}, The Weil Transform of w_{15}

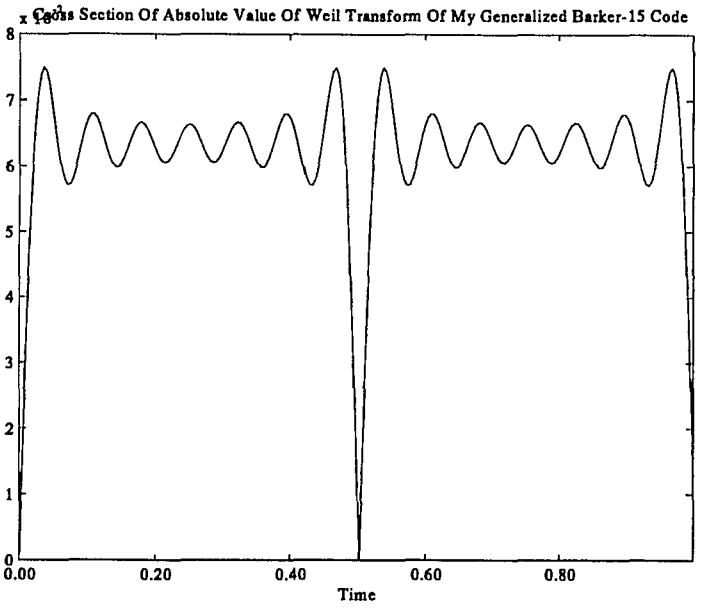


Figure 93: The Ambiguity Surface of w_{15}

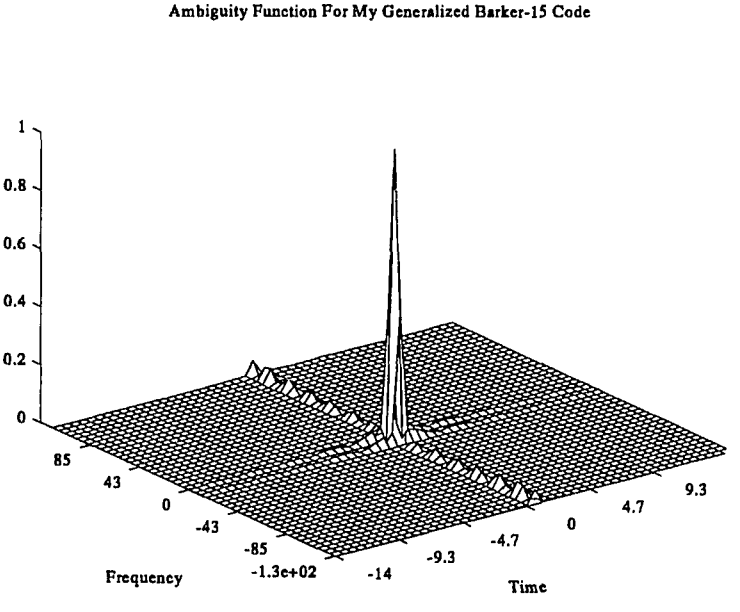


Figure 94: The Waveform w_{20}

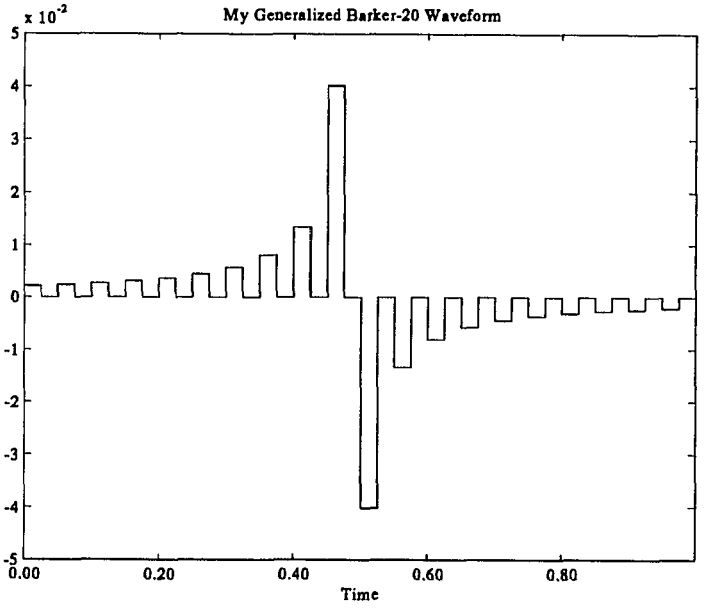


Figure 95: F_{20}, The Weil Transform of w_{20}

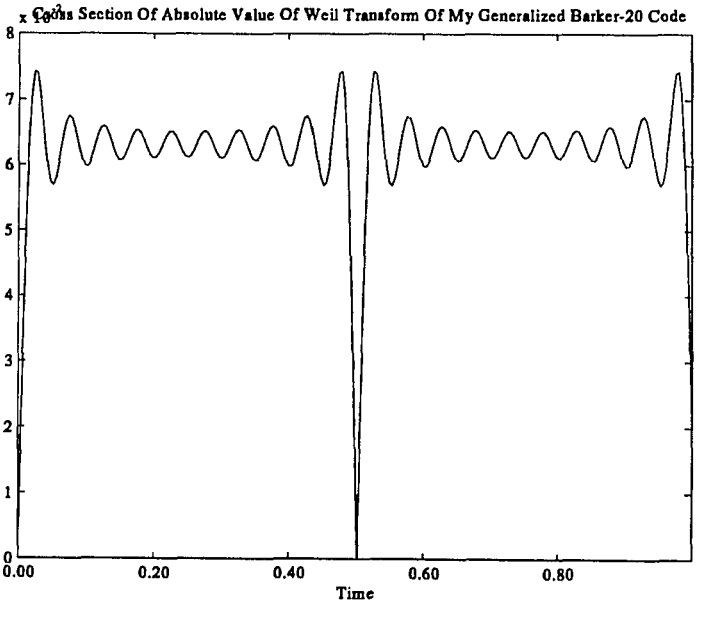
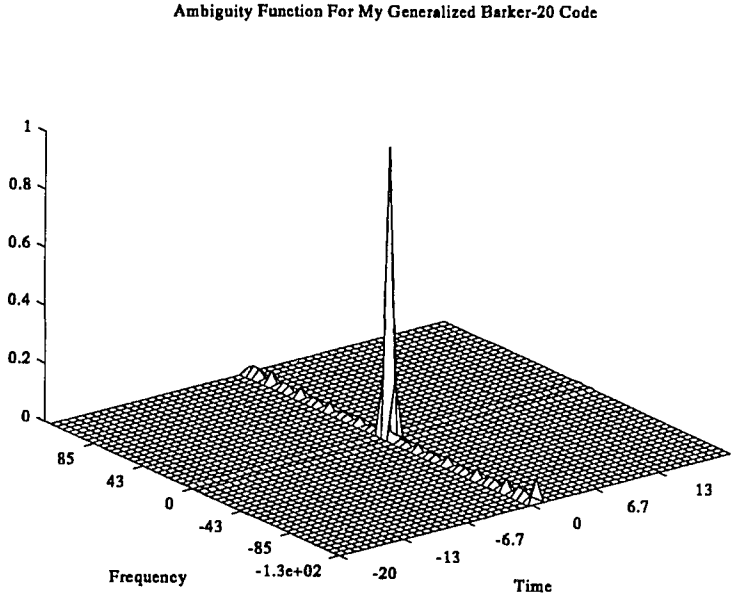


Figure 96: The Ambiguity Surface of w_{20}



6.2 Staggering of Winding Number Data

In this section, a new technique of radar waveform construction will be given, and it will be demonstrated by example that this new construction leads to thumbtack ambiguity functions.

The new technique is simply to place zeros with winding numbers 1 and -1, in a time-frequency staggered pattern in the unit square. This is done in the examples by placing the zeros randomly. The function is constructed using the product theorem 2.5.

In the first example, we place 21 zeros randomly on a 100×100 grid, 10 with winding number 1, and 11 with winding number -1, in a random pattern. In the second example, we place 41 zeros randomly on a 100×100 grid, 20 with winding number 1, and 21 with winding number -1, in a random pattern.

These examples demonstrate the correctness of the intuition that staggering of winding number data for functions in H_1 is a good way to construct functions with thumbtack-like ambiguity functions.

Figure 97: The Weil Transform of the First Staggered Data Function

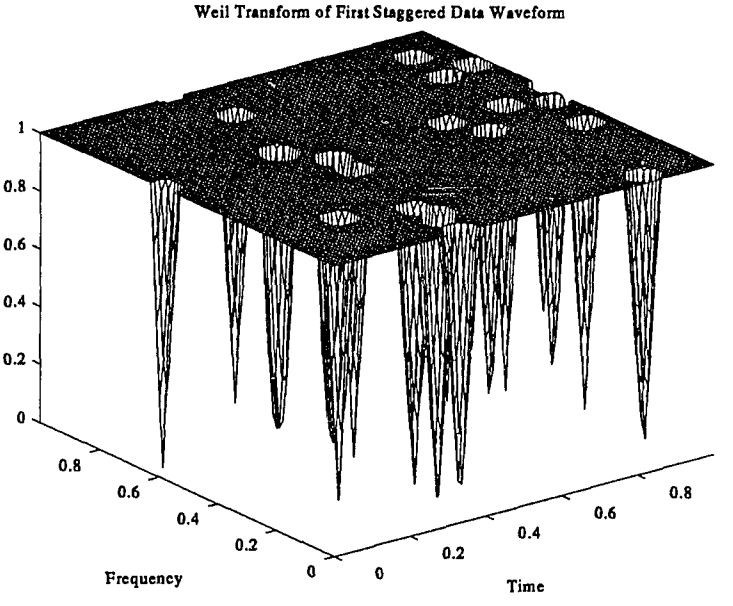


Figure 98: The Waveform of the First Staggered Data Function

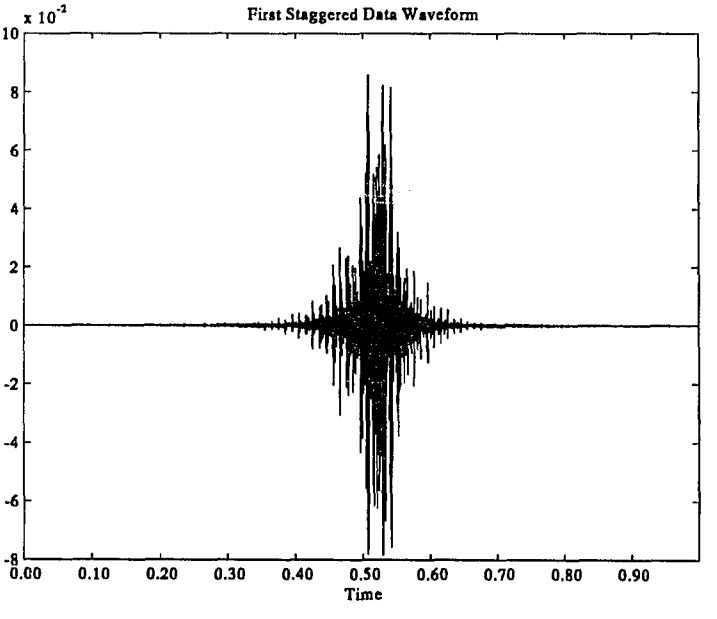


Figure 99: The Winding Number Data for the First Staggered Data Function

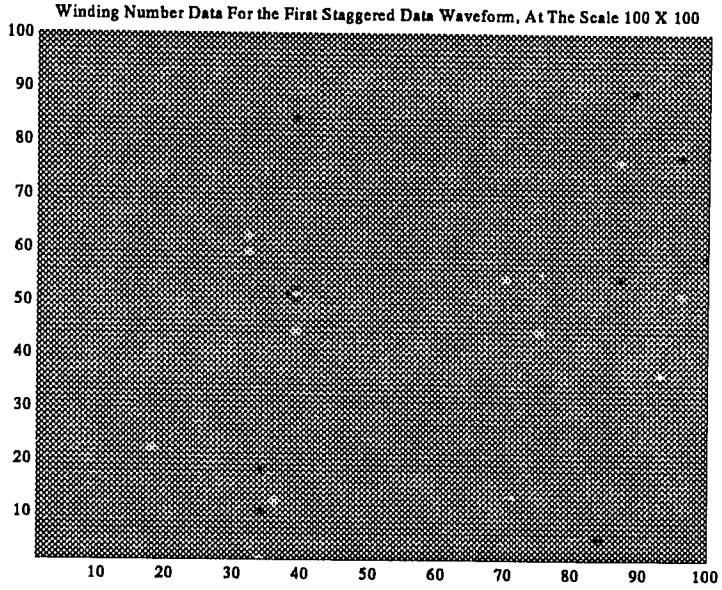


Figure 100: The Ambiguity Surface of the First Staggered Data Function

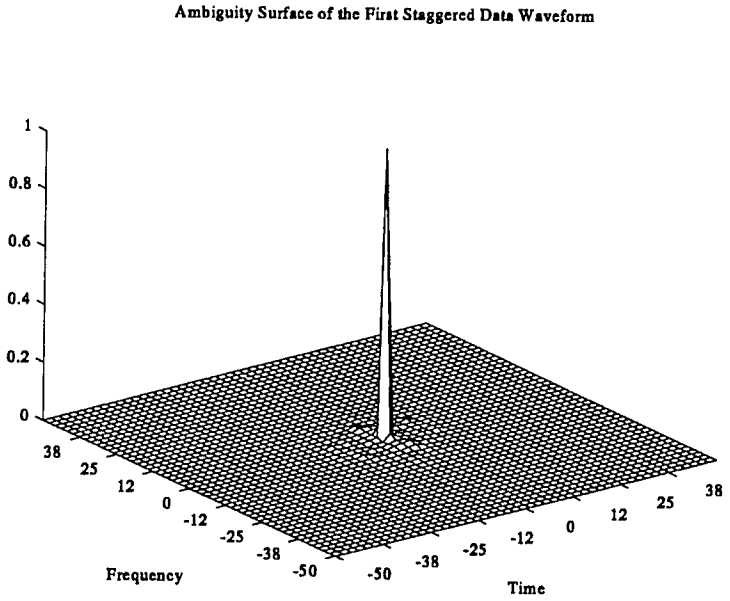


Figure 101: Detail of the Ambiguity Surface of the First Staggered Data Function

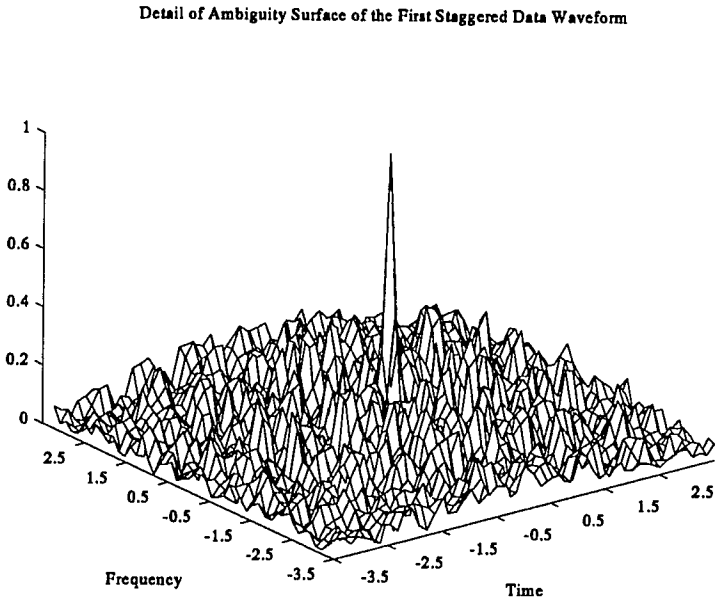


Figure 102: The Waveform of the Second Staggered Data Function

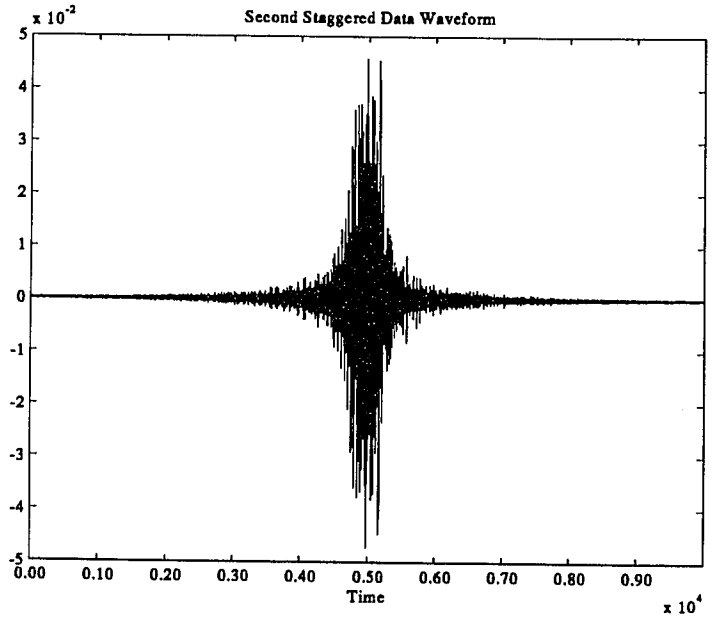


Figure 103: The Weil Transform of the Second Staggered Data Function

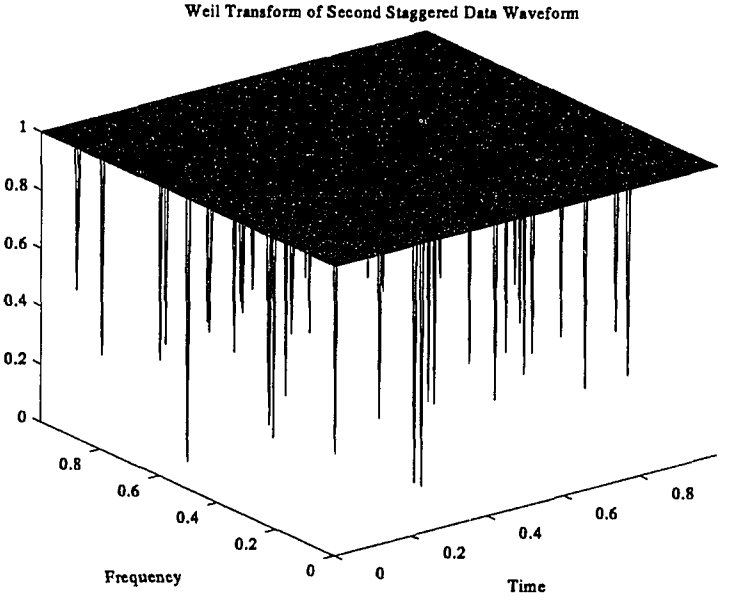


Figure 104: The Winding Number Data for the Second Staggered Data Function

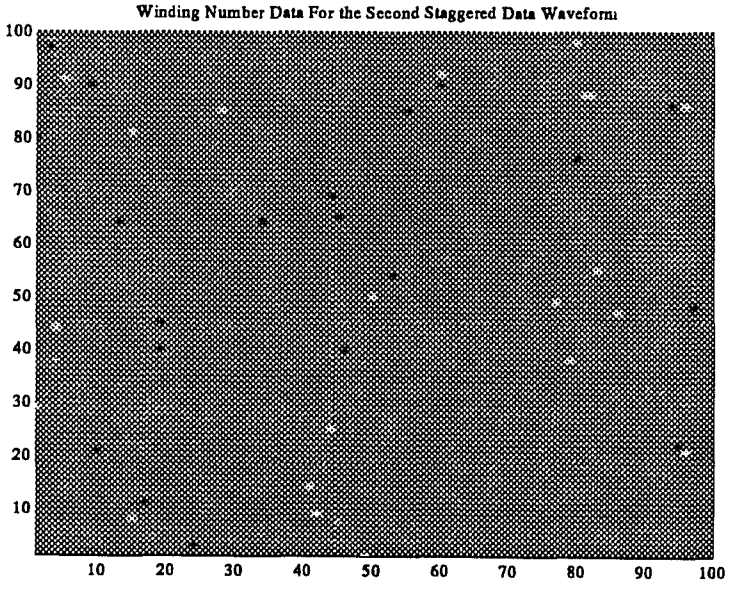


Figure 105: The Ambiguity Surface of the Second Staggered Data Function

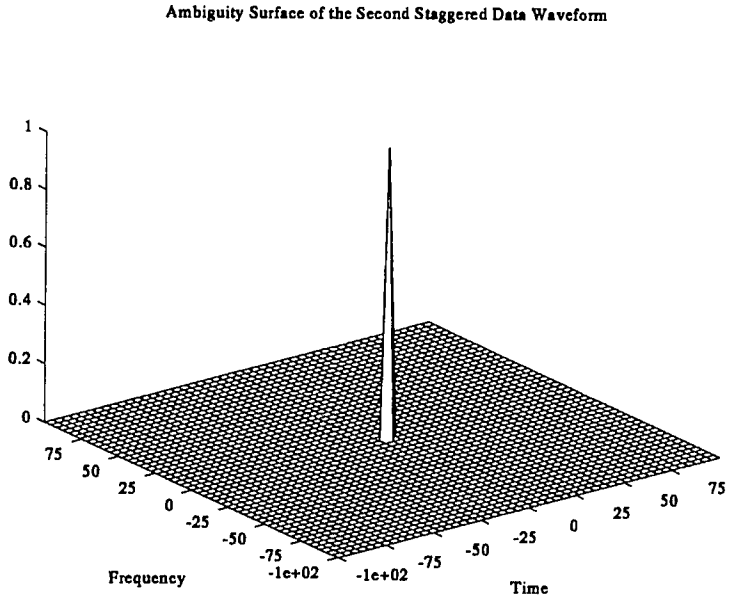
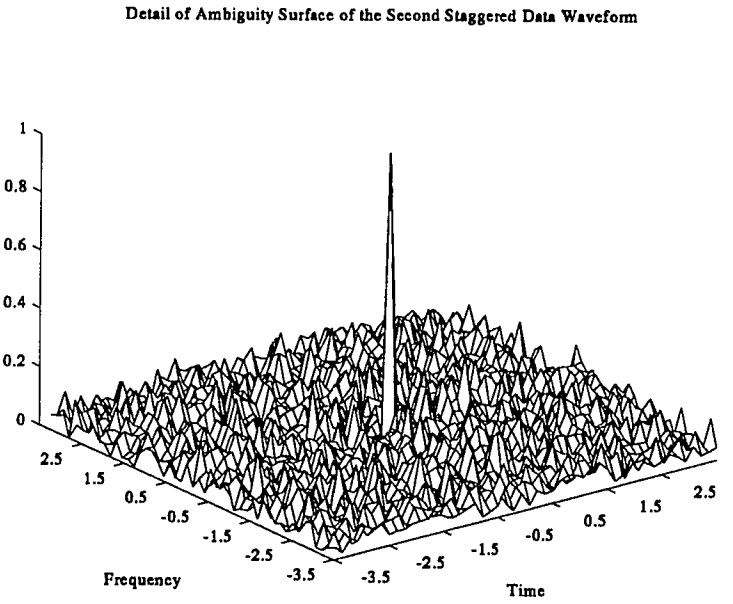


Figure 106: Detail of the Ambiguity Surface of the Second Staggered Data Function



A Program Listings

Notation: Listings of programs written in the Matlab computer language are given below. Note that a line beginning with a "%" is a comment. Built-in Matlab keywords and functions will be denoted in **this typeface**. Function names for functions written by me will be denoted in THIS TYPEFACE. Variables and mathematical symbols will be denoted like this: $variable1+variable2 = variable3$.

function $A = \text{AF}(x, g, tmin, tmax, dt, fmin, fmax, df, nsamp, samprate)$
%
% Copyright ©1993, Frank Geshwind. All rights reserved.
%
% Purpose:
% Returns the discrete cross-ambiguity function of x and g.
% Usage:
% $A = \text{AF}(x, g, tmin, tmax, dt, fmin, fmax, df, nsamp, samprate)$
% Parameters:

%	x, g:	Vectors of the same size. The discrete signals.
%	$tmin$, $tmax$, dt:	We will let time go from $tmin$ to $tmax$ with
%		an increment of dt, to get the time values
%		to shift by.
%	$fmin$, $fmax$, df:	We will go from $fmin$ to $fmax$ with an
%		increment of df, to get the frequency
%		values to shift by.
%	$nsamp$, $samprate$:	We assume that the vectors x and g are being
%		thought of as samples of a signal that
%		has a duration of $nsamp$ seconds, with
%		$samprate$ samples/second. These are used
%		to convert numerical values of time and
%		frequency into indexes into vectors and
%		their FFT's

% make sure that both x and g are row vectors
$x = x(:)'$;
$g = g(:)'$;

% initialize i for a loop below
$i = 1$;

% initialize j for the loop. j will essentially go from $tmin$ to $tmax$
% with a step size of dt, but we add a correction to insure that the

```
        % zero time shift is hit as one of the values for j, at least
        % if tmin < 0 < tmax.
j = tmin - MOD(tmin, dt);

        % convert fmin, fmax, and df into their discrete counterparts
deltaf = fix(nsamp * df);
ffmin = fix(fmin * nsamp + length(g)/2);
ffmax = fix(fmax * nsamp + length(g)/2);

        % these next two numbers guarantee that the central frequency
        % (0) is kept when we down-sample. The idea is the same as the
        % correction term for j.
center = fix(length(g)/2) + 1;
offset = MOD((center - ffmin), deltaf);

        % the actual loop
while(j <= tmax)

            % convert j into its discrete counterpart
    fj = fix(j * samprate);

            % shift x by j seconds
    sx = SHIFT(x', fj)';

            % compute the FFT of the dot-product, as in the definition
            % of the ambiguity function.
    Atmp = fftshift(fft(sx. * conj(g)));

            % throw out some of the answer by decimation and truncation
            % if the FFT was computed for more values than needed.
    A(i,:) = Atmp(ffmin + offset : deltaf : ffmax);

            % Update for next iteration
    i = i + 1;
    j = j + dt;
end

function F = GAUSS_FUN(t, noiselev)
%
% Copyright ©1993, Frank Geshwind. All rights reserved.
%
```

```
% Purpose:
%     Returns a discrete sampled gaussian function, with simulated white
%     noise added.
% Usage:
%     F = GAUSS_FUN(t, noiselev)
% Parameters:
%     t:          A vector containing the values of t at which to sample
%                 the function.
%     noiselev:   Multiplier for white noise term

F = exp(-pi * t. ^ 2) + noiselev * randn(size(t));


function y = MOD(x, N)
%
% Copyright ©1993, Frank Geshwind. All rights reserved.
%
% Purpose:
%     To compute remainder when x is divided by N.
% Usage:
%     y = MOD(x, N)
% Parameters:
%     x, N:  Real numbers

y = x - N * fix(x/N);

function y = SATURATE(x, m)
%
% Copyright ©1993, Steve Benno and Frank Geshwind. All rights reserved.
%
% Purpose:
%     Scales the input array x by m, and saturates at, or
%     truncates values greater than, m.
% Usage:
%     y = SATURATE(x, m)
% Parameters:
%     x:  An array of values to saturate
%     m:  The slope of the transform

[row col] = size(x);

    % Resize m so that it is in terms of a (possibly large)
```

```
        % fraction of the maximum of x
m = m/max(max(x));

        % Make an m-scaled copy of x,
y = m * x;

        % and truncate to 1, all values greater than 1.
for r = 1 : row
    for c = 1 : col
        if(y(r, c) > 1)
            y(r, c) = 1;
        end;
    end;
end;


function sx = SHIFT(x, amount)
%
% Copyright ©1993, Frank Geshwind. All rights reserved.
%
% Purpose:
%     returns the row vector x, shifted by n units, modulo
%     the size of x
% Usage:
%     sx = SHIFT(x, amount)
% Parameters:
%     x:        A one dimensional array to shift, assumed to be a row
%               vector
%     amount:   Amount to shift by

    % Get the length of the vector x, assumed to be a row vector
len = size(x);
len = len(1);

    % Convert amount to shift by, to an amount between 0 and len.
index = MOD(amount, len);
if index < 0
    index = index + len;
end;

    % If index is zero, there is nothing to do,
```

```
if (index == 0)
    sx = x;

    % otherwise build the answer as the last len - index members
    % of x, followed by the first index members.
else
    sx = [x(index + 1 : len); x(1 : index)];
end;

function sx = APSHIFT(x1, amount)
%
% Copyright ©1993, Frank Geshwind. All rights reserved.
%
% Purpose:
%     returns the row vector x1, aperiodically shifted
%     by amount units
% Usage:
%     sx = APSHIFT(x1, amount)
% Parameters:
%     x1:      A one dimensional array to shift
%     amount:  Amount to shift by

    % Convert to a column vector
x = x1(:)

    % Get the length of the vector x1, saving info about row or col
len1 = size(x1);
if len1(1) == 1
    len = len1(2);
else
    len = len1(1);
end;
    % If shifting by more than the length of x, in either direction,
    % then return all zeros
if (abs(amount) >= len)
    sx = zeros(len, 1);

    % Otherwise, for negative shifts return the last len - |amount|
    % elements of x followed by |amount| zeros. For positive
    % shifts, return amount zeros, followed by the first len - |amount|
    % elements of x
```

```
else
    if amount < 0
        sx = [x(1 − amount : len); zeros(−amount, 1)];
    else
        sx = [zeros(amount, 1); x(len − amount : len)];
    end;
end;
    % convert back to a row if necessary
if len1(1) == 1
    sx = sx';
end;

function sig = SIG_CODE(code, len, freq, n)
%
% Copyright ©1993, Frank Geshwind. All rights reserved.
%
% Purpose:
%     Produces a sampled function corresponding to the phase shift code
%     given by code. code is assumed to be given in terms of n^th-roots of
%     unity. code should be expressed in terms of integers (mod n)
%     which specify a root of unity. If code(j) = k, then the
%     corresponding phase is e^{2πik/n}.
%     The output has len samples per unit time, and has a total
%     duration of length(code) units of time. The function is a
%     complex wave of frequency freq, phase shifted by e^{2πicode(k)/n} in the
%     k^th unit of time.
% Usage:
%     sig = SIG_CODE(code, len, freq, n)
% Parameters:
%     code:  A phase shift code, given by integers representing n^th
%            roots of unity
%     len:   The length of the output per code symbol
%     freq:  The carrier frequency of the output waveform
%     n:     A positive integer; (see definition of code)

    % make sure that i is properly defined
i = sqrt(−1);

    % set up the time points at which to sample the basic array, and
    % compute that array
t = [0 : (1/len) : 1 − .9 * (1/len)];
```

```
basic = exp(2 * pi * i * freq * t);

    % initialize the output
sig = [];

    % loop through the elements of the code, and tack on a copy of the
    % basic array, multiplied by the phase corresponding to the code
for k = [1 : length(code)]
    sig = [sig  exp(2 * pi * i * code(k)/n) * basic];
end;

function func = ZAK_CODE(code, len, n)
%
% Copyright ©1993, Frank Geshwind. All rights reserved.
%
% Purpose:
%     This program produces the Weil transform corresponding to the
%     phase shift code given by code. It gives a signal of duration len.
%     The Weil transforms are given as functions of one dimension, since
%     in the phase shift coding context, the transform only depend on
%     one variable.
% Usage:
%     func = ZAK_CODE(code, len, n)
% Parameters:
%     code:  A phase shift code, given by integers representing n^th roots
%            of unity
%     len:   The length of the output
%     n:     A positive integer; (see definition of code)

i = sqrt(-1);
func = fft(exp(2 * pi * i * code/n), len);


function sig = DSIG_CODE(code, len, n)
%
% Copyright ©1993, Frank Geshwind. All rights reserved.
%
% Purpose:
%     Produces a sampled function corresponding to the doubled
%     waveform coming from the phase shift code given by code.
%     code is assumed to be given in terms of n^th roots of unity. The
%     output has len samples per unit time, and has a total duration
```

```
%       of n_c = length(code) units of time. The function is given by n_c
%       copies of a basic waveform. The basic waveform has discrete Fourier
%       series coefficients given by the phases in code. That is
%       basic(t) = sum_{k=1}^{n_c} c_k e^{2πikt},
%       where c_k = e^{2πicode(k)/n}.
%       The k^th copy of the basic waveform is phase shifted by
%       e^{2πicode(k)/n}.
% Usage:
%       sig = DSIG_CODE(code, len, n)
% Parameters:
%       code:  A phase shift code, given by integers representing n^th roots
%              of unity
%       len:   The length of the output per code symbol
%       n:     A positive integer; (see definition of code)

        % make sure that i is properly defined
i = sqrt(-1);

        % set up the time points at which to sample the basic array, and
        % compute that array
t = [0 : (1/len) : 1 - .9 * (1/len)];
basic = zak_code(code, len, n);

        % initialize the output
sig = [];

        % loop through the elements of the code, and tack on a copy of the
        % basic array, multiplied by the phase corresponding to the code
for k = [1 : length(code)]
        sig = [sig  exp(2 * pi * i * code(k)/n) * basic];
end;

function out = SIG_FBGBARKER(num, len, freq)
%
% Copyright ©1993, Frank Geshwind. All rights reserved.
%
% Purpose:
%       Produces a sampled function corresponding to a geometric
%       generalization of Barker codes. The output consists of 2num
%       concatenated copies of a basic waveform, each multiplied by an
%       appropriate weight. The basic waveform is a complex wave of
```

```
%       frequency freq, sampled at len values from 0 to 1. The weights
%       are alternately zero, for even copies, and 1/(pi*i*n) for n odd,
%       as n goes from -num to num.
% Usage:
%       out = SIG_FBGBARKER(num, len, freq)
% Parameters:
%       num:  The number of basic arrays to concatenate
%       len:  The size of the basic array
%       freq: The carrier frequency

        % initialize the output
out = [];

        % set up the points at which to sample
t = [0 : (1/len) : 1 - (1/(2 * len))];

        % make sure that num is even
num = num - MOD(num, 2);

        % form the basic array
basic = [exp(2 * pi * i * freq * t)  zeros(size(t))];

        % concatenate weighted copies of the basic array, as in the
        % definition of the waveform
for n = [-num + 1 : 2 : num - 1]
        out = [out  (1/(pi * i * n)) * basic];
end;


function out = ZAK_FBGBARKER(num, len)
%
% Copyright ©1993, Frank Geshwind. All rights reserved.
%
% Purpose:
%       Produces a sampled Weil transform of a function corresponding
%       to a geometric generalization of Barker codes. The output consists
%       of samples of a function given by the sum sum_{n=-num}^{num} a_n e^{2 pi i n t}.
%       The a_n are alternately zero, for even n and 1/(pi*i*n) for n odd.
% Usage:
%       out = ZAK_FBGBARKER(num, len)
% Parameters:
```

```
%       num:  The number of non-zero terms in the sum
%       len:  The length of the output

t = [0 : (1/len) : 1 - (1/(2 * len))];
out = zeros(size(t));

        % make sure that num is even
num = num - MOD(num, 2);
for n = [-num + 1 : 2 : num - 1]
        out = out + (1/(pi * i * n)) * exp(2 * pi * i * n * t);
end;

function out = DSIG_FBGBARKER(num, len)
%
% Copyright ©1993, Frank Geshwind. All rights reserved.
%
% Purpose:
%       Produces the waveform corresponding to the double of a
%       geometrically generalized Barker code.
% Usage:
%       out = DSIG_FBGBARKER(num, len)
% Parameters:
%       num:  The number of basic arrays to concatenate
%       len:  The size of each basic array

out = [];

        % make sure that num is even
num = num - mod(num, 2);

        % Form the basic waveform
basic = FBGBARKER(num, len);
basic = [basic  zeros(size(basic))];

        % Concatenate appropriately weighted copies
for n = [-num + 1 : 2 : num - 1]
        out = [out  (1/(pi * i * n)) * basic];
end;

% Program: WELCH10
%
% Copyright ©1993, Frank Geshwind. All rights reserved.
```

```
%
% Purpose:
%       Compute the waveform and Weil transform corresponding to the
%       Welch-10 Costas array
% Usage: WELCH10
% Parameters:
%       none.

        % The Welch-10 permutation
perm = [1 3 7 4 9 8 6 2 5 0];

        % Set X and Y to the grid on which to sample the Weil transform
[X, Y] = meshgrid(0 : .02 : .998, 0 : .02 : .998);

        % Zero out the storage for the Weil transform and the waveform
Z = zeros(size(X));
sig = zeros(1, 500);

        % Loop through the elements of the code, and add appropriate
        % components to the output arrays
for j = [1 : 10]
        Z = Z + exp(2 * pi * sqrt(-1) * (j * X + perm(j) * Y));
        sig((j - 1) * 50 + 1 : j * 50) =
                exp(2 * pi * sqrt(-1) * perm(j). * [0 : .02 : .998]);
end;

% Program: WELCH30
%
% Copyright ©1993, Frank Geshwind. All rights reserved.
%
% Purpose:
%       Compute the waveform and Weil transform corresponding to the
%       Welch-30 Costas array
%
% Usage: WELCH30
% Parameters:
%       none.

        % The Welch-30 permutation
perm = [2 8 26 18 25 15 16 19 28 24 12 7 23 9 29 27 21 3 11 4 14 13 10 1 5 17 22 6 20 0];
```

```
        % Set X and Y to the grid on which to sample the Weil transform
[X,Y] = meshgrid(0 : .02 : 1, 0 : .02 : 1);

        % Zero out the storage for the Weil transform and the waveform
Z = zeros(size(X));
sig = zeros(1, 1500);

        % Loop through the elements of the code, and add appropriate
        % components to the output arrays
for j = [1 : 30]
        Z = Z + exp(2 * pi * sqrt(-1) * (j * X + perm(j) * Y));
        sig((j - 1) * 50 + 1 : j * 50) =
                exp(2 * pi * sqrt(-1) * perm(j). * [0 : .02 : .998]);
end;

function W = WDATA2(z, N)
%
% Copyright ©1993, Frank Geshwind. All rights reserved.
%
% Purpose:
%       Computes the numerical winding numbers associated with given
%       discrete Zak transform data. The winding numbers are computed
%       over a regular array of sub-squares of the unit square.
% Usage:
%       W = WDATA2(z, N)
% Parameters:
%       z:  discrete Zak transform data
%       N:  The sub-square will be in an N × N grid

    % Get the size of z, and the underlying winding number data for z
[row, col] = size(z);
[wh, wv] = WIND_STRUCT(z);

    % Loop through the subsquares of interest, and compute the winding
    % numbers around each, by calling WINDNUM
for iw = 0 : N - 1
    for jw = 0 : N - 1
        W(iw + 1, jw + 1) =
            WINDNUM(wh, wv, iw * row/N + 1, jw * col/N + 1,
                (iw + 1) * row/N + 1, (jw + 1) * col/N + 1);
    end;
```

```
end;

function [WH, WV] = WIND_STRUCT(Z)
%
% Copyright ©1993, Frank Geshwind. All rights reserved.
%
% Purpose:
%     Computes the basic 1,0,-1 edge data for calculating winding
%     numbers of a given Zak transform. Outputs two arrays: the
%     horizontal and vertical data
% Usage:
%     [WH, WV] = WIND_STRUCT(Z)
% Parameters:
%     Z: a discrete Zak transform

     % Get the size of Z, and make sure i is correct
[row_ws, col_ws] = size(Z);
i = sqrt(-1);

     % The Zak transform is computed on [0,1) x [0,1). We need to
     % fill in the values on the missing edges of the unit square.
     % This is done using the quasi-periodicity relations that Z must
     % satisfy to be in H_1. We set the left edge to a modulated copy
     % of the right edge, and set the bottom edge equal to the top edge
modulate_ws = exp(2 * pi * i * [0 : col_ws - 1]/col_ws);
Z = [Z; Z(1,:). * modulate_ws];
Z = [Z  Z(:,1)];

     % compute the angles of Z
A = angle(Z);

     % For the horizontal data, compare each element to the one to its
     % right, divide by π, and we get a -2, -1, 0, 1 or 2.
     % the computation is analogous for the vertical data
WH = fix((A(:,2 : col_ws + 1) - A(:,1 : col_ws))/pi);
WV = fix((A(2 : row_ws + 1,:) - A(1 : row_ws,:))/pi);

     % take care of the cases above where we get 2 or -2
WH = WH - (WH == 2) + (WH == -2);
WV = WV - (WV == 2) + (WV == -2);
```

The functions HWINDPART and VWINDPART add up the winding number data over horizontal and vertical paths respectively, are self-explanatory, and are:

```
%
% Copyright ©1993, Frank Geshwind. All rights reserved.
%
function N = HWINDPART(wh, row, col1, col2)
N = sum(wh(row, col1 : col2));

function N = VWINDPART(wv, col, row1, row2)
N = sum(wv(row1 : row2, col));

function N = windnum(wh, wv, lrow, lcol, urow, rcol)
%
% Copyright ©1993, Frank Geshwind. All rights reserved.
%
% Purpose:
%     Computes the winding number around a rectangle with given
%     vertices
% Usage:
%     N = windnum(wh, wv, lrow, lcol, urow, rcol)
% Parameters:
%     wh, wv:      The horizontal and vertical winding number data
%     lrow, lcol   the lower row and left column of the rectangle
%     urow, rcol   the upper row and right column of the rectangle

N = HWINDPART(wh, lrow, lcol, rcol - 1);
N = N + VWINDPART(wv, rcol, lrow, urow - 1);
N = N - HWINDPART(wh, urow, lcol, rcol - 1);
N = N - VWINDPART(wv, lcol, lrow, urow - 1);
```

The following code fragments were used to generate the staggered winding number data examples

```
    % Parameters for the first staggered winding number data waveform:
    % the size of the grid, grid, and the saturation parameter alpha.
grid = 100;
alpha = 10;

    % Set the positions of the zeros. pos holds the winding number 1
    % locations, neg holds the winding number -1 locations.
pos = [4 94;87 36;52 90;10 65;94 96;12 21;73 47;26 5;63 71;95 100;1 17];
```

```
neg = [5 49;34 96;55 68;1 52;39 74;28 61;68 46;60 46;2 8;9 23];

    % Alternately, for the second example:
grid = 200;
alpha = 100;
pos = 2 * [85 28;50 50;98 80;55 83;81 15;21 96;38 79;44 4;8 15;29 1;88 82;25 44;91 5;92 60;14 41;88 81;49 77;
           86 96;1 49;9 42;47 86];
neg = 2 * [48 97;76 80;90 60;45 19;54 53;40 46;69 44;80 1;97 3;64 13;64 34;3 24;65 45;40 19;86 94;90 9;22 95;
           85 55;21 10;11 17];

    % make sure that i is correctly set
i = sqrt(-1);

    % Set up the time values to sample with, for a grid × grid Weil
    % transform
t = -1 * grid * grid/2 : grid * grid/2 - 1;
t = t/grid;

    % We will be forming a product of certain functions, so initialize the
    % answer to all 1
zgp = ones(grid, grid);

    % Put in all of the positive winding number factors.
    % for each one, we take a Gaussian and shift it in time and frequency,
    % to get its isolated zero shifted by the correct amount.
    % Note that the locations of zeros will be shifted by (1/2, 1/2)
fork = [1 : length(pos)]
    atmp = fftshift(GAUSS_FUN(t - (pos(k, 1) - 1)/grid, 0).*
        exp(i * t * (pos(k, 2) - 1)));
    ztmp = ZAK(atmp, grid, grid);

        % Modify our factor by the approximate orthonormal basis
        % construct, using saturate
    ztmp = exp(i * angle(ztmp)). * SATURATE(abs(ztmp), alpha);

        % multiply the answer by this factor
    zgp = zgp. * ztmp;
end;

    % Do the same for the negative winding number factors.
    % In this case we use the complex conjugate of the Weil transform of
```

```
    % an appropriately shifted Gaussian
for k = [1 : length(neg)]
    atmp = fftshift(GAUSS_FUN(t − (neg(k, 1) − 1)/grid, 0).*
        exp(i ∗ t ∗ (neg(k, 2) − 1)));
    ztmp = ZAK(atmp, grid, grid);
    ztmp = exp(−i ∗ angle(ztmp)). ∗ SATURATE(abs(ztmp), alpha);
    zgp = zgp. ∗ ztmp;
end;
```

This next code fragment was used to generate the Gold code sequence used in the examples.

```
    % Gold code study.
    % First take u to be an m-sequence of length 31
u = [1,0,0,1,1,1,1,0,0,0,1,1,0,0,1,0,0,0,0,0,1,0,1,1,1,0,1,1,0,1,0];
    % Then take v to be a decimation of the periodization of u, by 5.
    % By a theorem of Gold, these two codes will be a preferred pair.
v = u(1 + MOD(0 : 5 : 31 ∗ 5 − 1, 31));
    % Enumerate the family of the Gold code G(u, v)
for k = [0 : 30]
    code = u. ∗ SHIFT(v′, k)′;
```

References

[1] L. Auslander. Sliding windowed Fourier transforms and the Heisenberg group. to appear in Proceedings of the NATO ASI on Underwater Acoustics, 1992.

[2] L. Auslander, S. Benno, F. Geshwind, J. Moura, and F. Warner. Summary of ideas. unpublished internal note, 1992.

[3] L. Auslander and F. Geshwind. Approximate frames and the multi-target radar problem. to appear in Proceedings of the NATO ASI on Wavelets And Their Applications, 1992.

[4] L. Auslander and F. Geshwind. Multi-target ambiguity functions. to appear in Proceedings of the NATO ASI on Underwater Acoustics, 1992.

[5] L. Auslander and R. Tolimieri. *Abelian Harmonic Analysis, Theta Functions and Function Algebras on a Nilmanifold*, volume 436 of *Lecture Notes in Mathematics*. Springer-Verlag, Berlin, 1975.

[6] L. Auslander and R. Tolimieri. Ambiguity functions and group representations. In C. C. More, editor, *Group Representations, Ergodic Theory, Operator Algebras and Mathematical Physics*, volume 6, pages 1–10. Mathematical Sciences Research Institute Publications, 1986.

[7] R. Balian. Un principe d'incertitude fort en théorie du signal ou en mécanique quantique. *C.R. Acad. Sc. Paris*, 292, Sérre 2, 1981.

[8] Jerome R. Bellegarda and Edward L. Titlebaum. The hit array: An analysis formalism for multiple access frequency hop coding. *IEEE Transactions on Aerospace Electronic Systems*, AES-27(1):30–39, January 1991.

[9] R. Blahut. Theory of remote surveillance algorithms. In *Radar and Sonar, Part I*, volume 32 of *IMA Volumes in Mathematics And Its Applications*. Springer-Verlag, New York, 1991.

[10] M. Cohen, M. Fox, and J. Baden. Minimum peak sidelobe pulse compression codes. *IEEE International Radar Conference*, pages 633–638, 1990.

[11] Charles E. Cook and Marvin Bernfeld. *Radar Signals: An Introduction To Theory And Application*. Academic Press, New York, 1967.

[12] L. Corwin and F. P. Greenleaf. *Representations of Nilpotent Lie Groups and Their Applications, Part 1: Basic Theory and Examples*. Number 18 in Cambridge Studies in Advanced Mathematics. Cambridge University Press, Cambridge, 1990.

[13] J. P. Costas. A study of a class of detection waveforms having nearly ideal range-doppler ambiguity properties. *Proceedings of the IEEE*, 72(8):996–1009, August 1984.

[14] I. Daubechies. The wavelet transform, time frequency localization and signal analysis. *IEEE Transactions on Information Theory*, 5:961–1005, 1990.

[15] Gary W. Deley. Waveform design. In M. I. Skolnik, editor, *Radar Handbook*, chapter 3. McGraw Hill, 1970.

[16] William F. Donoghue, Jr. *Distributions and Fourier Transforms.* Academic Press, New York, 1969.

[17] G. Folland. *Harmonic Analysis in Phase Space.* Number 122 in Annals of Mathematics Studies. Princeton University Press, Princeton, New Jersey, 1989.

[18] D. Gabor. Theory of communications. *J. Inst. Elec. Engin. (London)*, 93(III):429–457, 1946.

[19] S. W. Golumb. *Shift Register Sequences.* Holden-Day, San Francisco, CA, 1967.

[20] S. W. Golumb and R. A. Scholtz. Generalized Barker sequences. *IEEE Trans. on Information Theory*, IT-11:533–537, 1965.

[21] Solomon W. Golumb. Algebraic constructions for Costas arrays. *Journal of Combinatorial Theory*, 37:13–21, 1984.

[22] Solomon W. Golumb and Herbert Taylor. Constructions and properties of Costas arrays. *Proceedings of the IEEE*, 72(9):1143–1163, September 1984.

[23] Ernest E. Hollis. Comparison of combined Barker codes for coded radar use. *IEEE Transactions on Aerospace and Electronic Systems*, pages 141–143, January 1967.

[24] A. J. E. M. Janssen. The Zak transform: a signal transform for sampled time-continuous signals. *Phillips J. Res.*, 43:23–69, 1988.

[25] Joel B. Resnick. High resolution waveforms suitable for a multiple target environment. Master's thesis, Massachusetts Institute of Technology, June 1962.

[26] August W. Rihaczek. *Principles of High-resolution Radar.* McGraw-Hill, New York, 1969.

[27] H. L. Royden. *Real Analysis, second edition.* Macmillan Publishing Co., Inc., New York, 1968.

[28] Dilip V. Sarwate and Michael B. Pursley. Crosscorrelation properties of psuedorandom and related sequences. *Proceedings of the IEEE*, 68(5):593–619, May 1980.

[29] W. M. Siebert. Studies of Woodward's uncertainty functions. In *Quarterly Progress Report.* MIT Radio Engineering Laboratory, Cambridge, MA, 1958.

[30] Edwin H. Spanier. *Algebraic Topology.* McGraw Hill, New York, 1966.

[31] J. E. Storer. Optimum finite code groups. *Proceedings of the IRE*, page 1649, September 1958.

[32] Steven M. Sussman. Least-squares synthesis of radar ambiguity functions. *IRE Transactions on Information Theory*, pages 246–254, April 1962.

[33] R. Turyn. On Barker codes of even length. *Proceedings of the IEEE*, page 1256, September, 1963.

[34] C. H. Wilcox. The synthesis problem for radar ambiguity functions. In *Radar and Sonar, Part I*, volume 32 of *IMA Volumes in Mathematics And Its Applications.* Springer-Verlag, New York, 1991. (Originally appeared as Army Research Center, MRC Technical Summary Report Number 157, 1960).