Optimization of the Frequency-Spin Down Hough Transform

Carl Sabottke
Louisiana State University

July 2011
University of Florida
International REU 2011

Sapienza
Mentors: Pia Astone and Cristiano Palomba

Abstract

Due to computational limitations, blind searches for continuous gravitational waves cannot be conducted using coherent methods. Less computationally expensive hierarchical schemes have, therefore, been developed, which combine coherent data analysis methods with incoherent data analysis methods, like the Hough transform. For the purposes of gravitational wave searches, the frequency-spin down Hough transform is an improvement on the sky Hough transform. Code and implementation methods for the frequency-spin down Hough transform are currently in development. This code is currently being tested using data from VSR2, and once optimized the code will be used to analyze the data from future science runs of the Virgo detector. Recent work has allowed some improvements to be made to this code, and comparisons will be made between the Hough transform code as of May 2011 and the Hough transform code as of July 2011. Work has also been done to help convert the code written in Matlab to code written in C that can be efficiently run in a grid computing environment.

Introduction

There are four basic categories of potential gravitational wave signals currently under investigation. These include burst signals, compact binary coalescences, continuous waves, and stochastic background signals, and each of these signal types then has a host of corresponding potential sources. For example, burst signals would be expected to come from supernovae or the merger of two of two compact stars. The source for a binary coalescence signal is self-evident. It is two coalescing black holes or neutron stars [1]. Then, for continuous gravitational waves, the most promising source is spinning neutron stars. If a pulsar contains some type of rotational asymmetry, like a bump or precession about the axis of rotation, then it is possible for this star to generate gravitational waves [2].

Searches for these continuous gravitational waves can be either targeted searches or blind searches. For known pulsars, like the Crab and the Vela, the source parameters are known. Therefore, it is possible to use targeted data search techniques for these pulsars. A targeted search involves a coherent or matched filtering analysis over a long period of observation time. Knowledge of the source parameters is essential for this type of search. Blind searches then investigate possible signals from unknown pulsars, which obviously means that the source parameters are not known too. For this type of search, an attempt is made to search over the space of all possible source parameters. Using solely coherent methods, this search method has a prohibitively high computational cost. Therefore, semi-coherent or incoherent search methods have been developed to be used as part of a hierarchical search in conjunction with coherent methods. [2]

Hierarchical Approach

Since blind searches cannot be carried out using coherent methods, a hierarchical approach is employed. Hierarchical approaches start with the short Fourier transforms and alternate between incoherent and coherent computational steps. The information in the short Fourier transforms is then pieced together with an incoherent method. This incoherent method then generates a list of candidates in the parameter space that meet certain criteria and have values above certain thresholds. Thus, these incoherent methods greatly reduce the size of the parameter space that needs to be searched. Then, since coherent methods have better sensitivity, a coherent analysis is performed on each of the candidates, but now it is possible to perform this coherent analysis over a much longer time period, which greatly increases the sensitivity [3, 4].

As a note, the data analysis group at Sapienza works with nondemodulated data. This means that the short Fourier transforms used in the very first coherent step do not have signals with detectable spin down effects or frequency modulation due to the Doppler Effect. The length of the short Fourier transforms used by the Sapienza group is 18 minutes. These short Fourier transforms can be up to 30 minutes in length without the Doppler Effect from the earth's motion or the star's spin down having a significant effect on the Fourier transforms. However, longer initial Fourier transforms lead to greater sensitivity. The downside is that this data must be demodulated to remove changes in the signal that are caused by the movement of the earth and the spin down parameter. This demodulation must be done differently for different sky regions and spin down values. These types of searches work with the F-statistic, and they have been used for searches with data from LIGO and GEO [3].

For the incoherent step, different data analysis techniques can be used. Examples of these methods include the stack-slide method, which is related to the Radon transform, and other techniques, which are based on the Hough transform [5]. The group at Sapienza uses the Hough transform as the incoherent search method for the group's data analysis. The Radon transform theoretically has a slightly higher sensitivity than the Hough transform. Nevertheless, the Hough transform is the preferred technique for data analysis at Sapienza because it has advantages over the Radon transform in regards to robustness and computational speed [6].

Hough Transform

The Hough transform was originally introduced by Paul Hough to aid the analysis of bubble chamber images at CERN. Now, it has obtained many more general uses and is a standard technique for feature detection. For example, depending on the implementation, the Hough transform can be used to detect the presence of lines, circles, or ellipses. The most basic version of the Hough transform is the linear Hough transform. As an extremely simple, yet still edifying example, we can have a line described by the equation

$$y = mx + b \tag{1}$$

Given this equation and an arbitrary image in the (x,y) plane, it is possible to perform a Hough transform from the two dimensional (x,y) plane to the two dimensional (m, b) plane. Performing this Hough transform will lead to the detection of the lines present in the image and yield results for the values of m and b for these lines. Computationally, we can define an array. Indices for rows will relate to m, the slope, and indices for columns will relate to b, the y-intercept. The dimensions will be determined by a resolution factor. Initially, this will be an array of zeros. Then, for each point in the (x, y) plane, we will make a list of all possible lines that this point could be contained within. These possibilities correspond to a curve in (m,b) space. Every position in the array that corresponds to a point on this curve will then

be updated. This process is repeated for every point in the (x, y) plane. The places where the curves in (m, b) space intersect correspond to the values of m and b for lines in the image that has been subjected to a Hough transform. These places of intersection also correspond to the positions in the array that have the highest values. This algorithm is obviously can be quite easily performed using a computer, which is part of the reason for its popularity, particularly in relation to computer vision. However, the technique has other uses, namely blind searches for continuous gravitational waves from unknown neutron star sources.

As a caveat, the linear Hough transform previously given as an example is quite simplistic in its formulation. In fact, it is so simplistic that it is actually a bit problematic. A vertical line has an infinite slope. There is obviously then no way to represent such a case in the two dimensional (m, b) plane. However, this problem can be remedied without introducing too much more complexity. A line can be described by more equations than just y= mx + b. A line can also be written as

$$y = (-\cos\theta/\sin\theta)\ x + (r/\sin\theta) \tag{2}$$

Here m has been replaced by $-\cos\theta/\sin\theta$, and b has been replaced by $r/\sin\theta$. r is shortest the distance from the line to the origin in the (x, y) plane, and θ is the angle between the x-axis and r if it is treated as a vector. θ has a range from 0 to π, and now the problem of a line with infinite slope has been resolved. This is just a small sampling of the flexibility of the Hough transform [3].

As applied to blind searches for gravitational waves, different Hough transforms can be implemented as part of the incoherent search methods. Previously, a popular method of the Hough transform was the all sky Hough. This Hough transform is not a linear Hough transform, and it involves mapping annuli of detection on the celestial sphere. To reduce the computation time, look up tables had to be used, and this added additional losses to sensitivity. This method of Hough transform is also more computationally expensive than the linear frequency-spin down Hough transform [7].

It is, perhaps, unsurprising then that presently a new Hough transform has superseded the old method all sky method. This new Hough transform involves a change from the (f, t) plane to the (f_0, d) plane. This is a linear Hough transform, and it is computationally less cumbersome than the old method. It is similar to the basic, general example of a linear Hough transform cited previously. The input parameters are f and t. Here, f is the observed frequency after a correction has been made for the Doppler effect. t is the time of the observation. f_0 is the source frequency. More specifically, this is the frequency of the gravitational wave initially emitted by the spinning neutron star. d is the spin down parameter. d can also be referred to as f'. The reason for this is that the spin down can be thought of as the rate of change for the source frequency. As neutron stars spin, they emit energy in the form of gravitational waves. The dissipation of this energy causes the speed of rotation for a neutron star to decrease over time. The spin down parameter, therefore, describes the rate of change for the neutron star's rotation. As the speed of rotation decreases, the frequency of the gravitational wave emitted by the source also decreases [8].

The frequency-spin down Hough has several advantages over the sky Hough. One main advantage is that increasing the frequency resolution does not carry a high computational cost. Also, the previous method had favored regions of the sky map, which would produce an excessive number of candidates. The frequency-spin down Hough transform does not share this problem, because it is constructed separately for each point in the sky. For the frequency-spin down Hough, the principal equation is

$$f = f_0 + d\,(t - t_0) \tag{3}$$

We can treat $t - t_0$ as one variable Δt. Now, we have an equation of the form $y = mx + b$, where $(\Delta t, f)$ equates to (x, y). Also, (d, f_0) equates to (m,b). We can also write the equation as

$$d = - f_0/\Delta t + f/\Delta t \quad (4)$$

where this rearrangement reflects more clearly the form of the line that is formed in the (d, f_0) space as a result of a point in the $(\Delta t, f)$ plane. The slope of this line is $-1/\Delta t$. Using this description, the problematic case of a vertical line will only arise when $\Delta t = 0$ and $t = t_0$. This situation can be avoided without too much difficulty. Therefore, this problem does not prevent the frequency-spin down Hough transform from being implemented effectively [8]. To demonstrate the result of this transform, Figure 1 shows the result of this Hough transform when performed on idealized data.

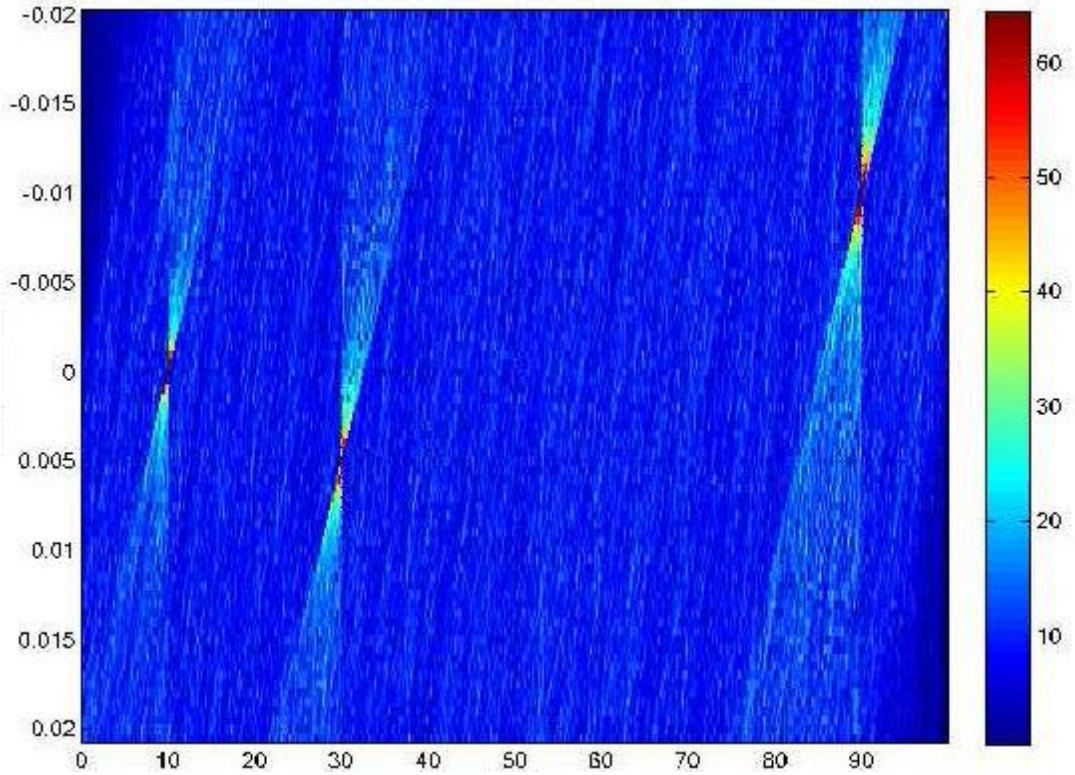

Figure 1- The x-axis displays frequency in Hz. The y-axis displays spin down values. The red areas show positions in the Hough map which have received many bin counts. These areas with high bin counts represent detections. This figure has been taken from Reference [8].

The resolution of the computation results in a width, Δf, for the lines in the (d, f_0) plane. In the Hough map array, each line increases the values in multiple bins across each row. The width determines the number of bins in each row updated by each line that results from a single point in the $(\Delta t, f)$ input plane. For example, a typical resolution factor used in calculations is 10. This means that Δf is also 10, and for each line added to the frequency-spin down array, 10 bins are updated in each row. If the direct differential method is used to compute the Hough, the resolution factor can be increased without greatly increasing the computational cost. This is because in the direct differential method it is acknowledged that for each line, all of the values that must be updated in the array are between two parallel lines. The parameters obey the inequality

$$f - d*\Delta t - \Delta f/2 < f_0 < f - d*\Delta t + \Delta f/2 \quad (5)$$

and with this inequality it is possible to initially only update the Hough map in two bins on each row for each line regardless of the resolution factor. The map can incremented at the position that corresponds to $f - d*\Delta t - \Delta f/2$. Then, the map should be decremented by an equal amount at the position that corresponds to $f - d*\Delta t + \Delta f/2$. After all the lines have been added to this derivative Hough map, an integration procedure can be performed to obtain the Hough map. This integration procedure simply computes the cumulative sum along the row for each position in the array. Computationally, this is much more efficient than updating every position along each band initially, particularly for the case of an adaptive Hough transform where incrementing the Hough map involves a float operation rather than just an integer operation [8].

This direct differential method is an extremely successful part of the May 2011 Hough transform code. However, it is actually possible to reduce the computation time even further. For the Hough transform itself, the most computationally expensive step involves updating the values in the array. The direct differential method is effective, because for a resolution factor of 10, it does not require 10 positions to be updated for each point in a line. Instead, it requires only two positions to be updated. However, the July 2011 Hough transform expands on this idea even further. To simplify the explanation of this improvement we have $a = f - d*\Delta t - \Delta f/2$. We then have $b = f - d*\Delta t + \Delta f/2$. The May 2011 Hough transform involves computations that act on two vectors, a and b. However, it is not necessary to work with both of these vectors. The end result is unaffected if the computation is only carried out using the a vector. The May 2011 Hough transform uses a and b to compute the values in an array binh_df0, which is the direct differential map before the cumulative sum operation is performed. In the July 2011 Hough transform, the a vector is used to compute values for an array binh_df0a. After, this array is completely constructed the values at each position in the array are copied into a new array binh_df0b where the values are shifted over by a number of positions equal to the resolution factor. This means all the values are shifted over to the right 10 positions in a typical construction. With this new array , binh_df0b, it is easy to obtain binh_df0 by subtracting binh_df0b from binh_df0a. This new computation technique nearly doubles the speed of the Hough transform computation.

For a standard Hough transform, the Hough map array or histogram is only updated by one at each position that corresponds to a detection in the image. However, for the detection of gravitational waves, this is not an ideal method. The images being subjected to the Hough transform in the study of gravitational waves are peak maps. These peak maps are obtained from short Fourier transforms. For each time period, they contain lists of the frequencies detected at that time. These peak maps also contain information about the noise in the region of the detected frequency for each time. The noise and the sensitivity in the detector are not constant for all times and all frequencies. Therefore, it is not ideal for all the points at all the detected frequencies and times to contribute equally to the Hough map. The adaptive Hough transform improves upon the standard Hough transform by allowing the value that the Hough map is incremented by to vary. This technique improves the sensitivity of the search without greatly increasing the computational cost. This is especially true when using the direct differential method of the May 2011 Hough transform code and also the streamlined direct differential approach of the July 2011 Hough transform code.

The Matlab programming environment is highly useful to the development of blind search methods. When developing novel algorithms and search techniques, initially the bulk of the code is written in the Matlab environment. This environment is used extensively to test the algorithm and the refine the code. Matlab is used in the early stages for multiple reasons. The Matlab language is quite simple and much more flexible than C. Additionally, the Matlab environment makes it quite easy to generate graphics and debug the code. However, it is not acceptable to use the Matlab environment for a real blind search. Matlab code is not maximally efficient, and the environment introduces overhead, which consumes time. Additionally, grid computing is essential for blind searches, and Matlab code is not easily run in parallel on the grid. Among other reasons, a main reason that Matlab cannot be run in the grid is because of licensing issues. Every machine in the grid would need to have a license to use Matlab, and the cost of this is prohibitive. Therefore, to lower both the cost and the speed of the analysis it is necessary to translate tested Matlab code in to C.

Translating code from Matlab in to C is not an easy process, and several difficulties can and do arise. For example, Matlab is a column major language, while C is a row major language. Matlab functions are optimized to work along columns, while C functions optimally work along rows. Most other complications arise, because Matlab code is written at a much higher level than C code. Unlike C, Matlab does not require specific declarations of data type. Additionally, Matlab specializes in being able to quickly perform vector operations, and it has a large suite of built-in functions that are not necessarily contained in convenient C libraries. A single line quickly written in Matlab can, therefore, take several lines of C code that include multiple processing loops and multiple function calls. Even though C code is typically more laborious to write and read through than Matlab code, the required explicitness of C code can be advantageous. A convenient single line of code in Matlab can hide a great computational burden. C code makes the time required to execute a given block of code much more intelligible and noticeable for the programmer.

For example, when the code for the adaptive frequency-spin down Hough transform was first being developed, it was believed that the adaptive Hough transform was much more computationally expensive compared to the standard Hough transform than it actually is in most cases. This is because for the May 2011 Hough transform the Matlab code deceptively made unnecessary function calls an excessive number of times. For the adaptive Hough transform, the increment is adjusted based on the noise. This adjustment is the same for all frequency maxima that appear at a given time for one of the short Fourier transforms. Therefore, this value only needs to be calculated once for each observation time. However, in the Matlab program, a function was called to calculate the increment value every time the Hough map histogram was in the process of being updated. This means that the function was called for every point in every line that was being added to the Hough map histogram. However, this function needed to only be called once for every two stripes that were added to the differential Hough map. Altering the code to avoid these unnecessary calculations significantly sped up the computation. The elimination of these types of function calls is one of the improvements of the July 2011 Hough transform.

Yet, despite the occasional deceptiveness of the code, the advantages to initially programming in the Matlab environment are numerous. For instance, the Matlab Profiler is almost indispensible for optimizing code and improving performance. Activating the Profiler introduces additional computational overhead to the code, so it is only used as part of the debugging and optimization process. The Profiler runs selected code and then generates a report. This report gives the number of times all functions are called within all the functions. It also gives the time spent on each line of code. This tool is incredibly useful for optimization and for identifying oversights in the code. It is actually with this tool that the initial inefficiencies in the adaptive Hough code were first identified, even though this mistake would have also become more evident once the code was translated to C.

General Structure of the May 2011 Hough Transform

To fully understand the computational requirements of the Hough transform it is necessary to understand the structure of the program. This structure includes how it reads the data, how it uses this data to make computations, and then how it uses these computations to generate output. For Matlab, peak maps are contained in vb1 files. For C, peak maps are contained in p10 files. In Matlab, the computation is begun by calling a function called call_analysis. This function has several inputs. It receives a structure that contains information about the antenna the data comes from. It receives a verbosity argument, which controls the number of graphs the function generates. It receives information about the file to begin analyzing, and, since we are working with injected signals to do data analysis, it also receives information about the injected signal being analyzed as well as several Boolean arguments to modify whether or not the analysis will perform a standard Hough or an adaptive Hough or if it will account for the Doppler effect. It also receives a limit for the frequency band to be analyzed. For the May 2011 code, the frequency band is divided into sub-bands which have a span of 10 Hz. For the July 2011 code, this sub-band span has been added as another input.

The function call_analysis divides the frequency band into a number of sub-bands. Then, it for each sub-band, it calls a function analysis. The function analysis checks the arguments received from call_analysis and then calls crea_timefrequencyAllsky. The function crea_timefrequencyAllsky performs several tasks. This function first reads a file. For each Fourier transform in the file, it obtains arrays with all the frequency peaks and the associated noises and amplitudes. It sorts these files according to the sub-band limits to reduce the number of frequency peaks being dealt with. It then further reduces the amount of frequency peaks being dealt with by cutting down the arrays based on whether or not the amplitudes meet certain criteria. After this elimination process is completed, the code adds this array to an array that contains all the peak map data that meet the limit and threshold criteria. Constructing this array is an extremely time consuming operation, though, because in the May 2011 code the size of this array is dynamic. Changing this process will be an important improvement to the July 2011 code. Alternatives are currently under investigation. This step is so time consuming, because the array's size is dynamic, and it changes with each additional Fourier transform that is read. It is important for future versions of the code to not perform this operation. The function crea_timefrequencyAllsky also calculates the Doppler Effect at each time for each position in the sky.

The outputs of crea_timefrequencyAllsky are a structure that contains all the times in the files and the associated frequency peaks and noise levels. The Doppler Effect values for all the sky positions analyzed are also part of the output. This data is returned to the analysis function. The analysis function manipulates the time values and uses the time information to generate some spin down data. The maximum value of time, the observation time, is used to calculate the step size for the spin down parameter. The analysis function then initiates a for loop that goes through all the sky positions under analysis. Inside this for loop is a call to hfdf_adaptivehough2010. This function performs an adaptive Hough transform on the data that is given to it through multiple structures. This function then returns a structure that can be plotted or used to find maxima. A function called twod_peaks is called within the for loop to analyze the structures generated by hfdf_adaptivehough2010.

Much work has gone into enhancing the performance of hfdf_adaptive2010. However, work has also gone into to enhancing the performance of the program as a whole. At the moment, by far the most time consuming step in the program is the resizing of the arrays that have peak information for each time. However, this is not an essential step in the program, so there is currently a search for an elegant way to eliminate this step and not replace it with an equally time consuming step, while still maintaining flexibility in the code.

There is also another major problem with this structure of the program. The thresholds for selecting peaks from the larger peak maps are set before the Doppler Effect is taken into account and

before spin down is taken into account. Therefore, once these manipulations are performed, it is possible for the value of the frequency to exceed the bounds of the map being constructed. When this happens, it generates an error and causes the program to fail. For the May 2011 Hough transform, there is a workaround for this, but it is computationally expensive. The May 2011 Hough transform checks the bounds of each frequency in each vector. This is part of a triple nested for loop in hfdf_adaptive2010, which means it was part of a quadruple nested for loop in the entire program.

This is obviously a less than ideal solution. Therefore, the July 2011 Hough transform takes a different approach. Instead of performing limit checks, the July 2011 Hough transform simply increases the size of the frequency dimension of the Hough map array. This averts the crash caused by the array index being out of bounds. The array can then be trimmed back down to the original size after the Hough map has been computed. This is much faster than performing limit checks for every position in the array, and it achieves the same results.

Optimization of the May 2011 Hough Transform

A great deal of effort has been invested in optimizing the May 2011 Hough transform code to lead to the July 2011 Hough transform code. This has been a multi-faceted effort with the work being primarily concentrated in four areas. One of these areas and the one that in some cases consumes the least amount of time, but can reap the most rewards, involves simply cleaning up the existing Matlab code. This is work that does not fundamentally change the structure, function, or design of the program. Some of this work has already previously been discussed. An example of this work is making the change so that now the program only calculates the increment value for the adaptive Hough transform once for each line that is added to the map instead of at every position. This type of work does not introduce any new variables or increase the memory requirements of the program. It also does not prevent certain function from working properly.

Other examples of this first type of effort include elimination of other unnecessary computations. For example, to calculate the a and b vectors in the May 2011 code, several intermediate and unnecessary vectors were first computed. For the July 2011 code, these unnecessary computations have been eliminated without changing the flexibility of the program or its ability to efficiently perform other computations.

The second type of work is quite similar to the first. It still involves making changes to the Matlab code to increase efficiency and speed. However, these changes have more profound effects on the fundamental concepts of computation. An example of this type of work is the modification of the direct differential method. This modification eliminated the b vector. However, the b vector was used in other computations. Therefore, eliminating the computation of this vector forced other parts of the code to be modified as well. This modification was not as simple as most of the other modifications which involved making isolated changes to the program. The b vector was used in the May 2011 to perform limit checks. Eliminating the b vector then made this limit checking code completely ineffective and caused multiple problems in simulations. Fortunately, the alternative to limit checks, which was discussed previously is a much faster solution. Therefore, making this alteration to the code was actually beneficial. Nevertheless, this still represents a more significant alteration to the code.

The third type of work involves making more general changes to the flow of information and structure of the code. This is the most complicated type of task. Part of the motivation for altering the overall structure and relationships of the functions involved is to increase the ease with which the code can be translated to C. Another reason is to generally increase the speed of the code. The general structure of the July 2011 Hough transform code is still similar to the May 2011 code. However, code is currently being tested that radically changes the structure of the program. This code still maintains the initial sub-band divisions. However, it builds up the differential Hough maps for each sky position every

time that a Fourier transform is read. This eliminates the very costly step of resizing arrays, and this code is much faster than the July 2011 Hough transform.

However, this prototype code still has issues to be resolved. The July 2011 code does not need to know how many files will be read in advance. It is a dynamic code that is flexible in regards to its input. While this flexibility is a nice feature, it slows down the code considerably. Optimizing the speed of the code requires eliminating flexibility. If the time for the final Fourier transform in the series of files can be known, then an observation time can be computed by also using the time from the first Fourier transform in the first file. This observation time is necessary to begin setting up parameters for building the Hough map. The current prototype code has a workaround for this, and the observation time is currently inputted directly by the user. However, this is less desirable than the ultimate goal of having a program that follows a procedure which quickly obtains the observation time.

It is likely that code developed in C will be more similar to this prototype code than the July 2011 code. The July 2011 code calls many convenient Matlab functions that are more difficult to code in C. The prototype code takes a more simplistic, rigid approach, which makes it easier to translate into C. Thus, this rigid code has two main advantages, because it is faster, and it is easier to write when using C, which is a low level language when compared to Matlab, even though it is not in itself a low level language.

The fourth facet of the work then involved making the Matlab code more easily translatable to C code. This concept had a heavy influence over attempts to change the general structure of the code. It also has had some influence over how the functions are written. However, Matlab code performs best when using vector operations. Therefore, instead of avoiding this highly useful of the feature of the Matlab language, an effort has been made to include comments in the code to aid in the translation.

Generous inclusion of comments in the code is truly the key to aiding the translation to C. C requires data types to be stated in advance, but Matlab does not. Therefore, code in Matlab can be very unclear about what data types are being manipulated. Including this vital information in comments greatly enhances the code's ability to be translated to another language.

Optimization Results

Much of the testing for the frequency-spin down Hough transform code has been conducted using the data from science run two of the Virgo detector. This detector data several injected signals for fake pulsars, and these fake signals are indispensible for running data analysis tests. To better understand the code, several analysis simulations have been conducted. Run time information for these simulations has been obtained both by using the Matlab Profiler and the tic and toc Matlab functions. These two techniques have greatly aided the process of understanding bottlenecks in the code.

For convenience, many tests involved looking at the signal from pulsar 8. Again, this is a fake signal that was injected into the data and does not represent a signal from a real pulsar. Pulsar 8 has a sky position marked by an ascension angle of 351.4 degrees and a declination angle of -33.4 degrees. The source initial frequency is 194.3 Hz, and the spin down parameter value for this fake pulsar is -8.65E-9.

The Matlab Profiler was mainly used to understand the computational cost of specific lines and sections of the code. The more accurate tic and toc function which do not introduce overhead to the code were used to compare the time needed to run different analyses with different codes. A typical test ran on a sub-band of 10 Hz. Pulsar 8 was examined, so the code searched the range from 190 Hz to 200 Hz. The call for the May 2011 code for this search is call_analysismay2011(virgo,0,'VSR2-V3','pulsar_8',10,1,1,1,pulsar_8,[190 200]). One time for this search was about 785.4 seconds. This search looked at three sky points, so the Hough transform was run 3 times. These Hough transforms each took about 153 seconds.

The July 2011 Hough transform code can perform the same search and produce the same results much more quickly. Using the call, call_analysis(virgo,0, 'VSR2-V3','pulsar_8',10,1,1,1,pulsar_8, 10 [190 200]), the search with the improved code takes only about 367.9 seconds. More revealing is that the Hough transforms each now only take about 3.4 seconds. The Hough transform code is now over forty times faster, because of some of the code optimization techniques discussed previously.

Admittedly, this factor of forty is a bit deceptive. One of the main obstacles to speed in the May 2011 is a complicated noise modification calculation for the adaptive Hough. This noise calculation was highly costly, and its effectiveness was not entirely clear. Removing this noise calculation and replacing it with a simpler calculation that simply uses the noise value given in the peak maps or the median of these values increases the speed by a factor of about 10. Running the May 2011 code without this noise calculation shows that the Hough transforms can only take about 15 seconds each.

Nevertheless, the other optimization techniques still represent significant computational improvements. At minimum, an attempt will be made to search about 3000 sky positions using grid computation. Therefore, additional optimizations are important. The factor of 4-5 increase in speed is still quite valuable. It also must be noted that in a realistic situation a Hough transform will not take only 3.4 seconds. For the purposes of tests, only one data decade file was analyzed. Each data decade file represents 10 days of detector data. For a realistic search, the program must perform analysis on all of the data decades that are part of the science run. Therefore, techniques like the enhanced direct differential Hough map construction technique will have much more appreciable effects when applied to a realistic search. The streamlined direct differential technique alone accounts for almost, but not quite a factor of two increase in the code speed. Calculating the adaptive increment outside of the innermost for loop also accounts for almost a factor of two increase in speed. Other code alterations that include the deletion of minor unnecessary intermediate variables and arrays account for why the increase in speed is slightly more than a factor of four.

As previously mentioned, the prototype code performs much better than even the July 2011 Hough code. This significant speed does not come from improvements to the Hough transform code, but to general improvements in the structure of the code. To some extent, these structural changes make the code less flexible. However, the speed advantages of this prototype code cannot easily be dismissed. For only a 10 Hz search, the speed increase is not completely appreciable. For example, it has already been stated that the time to do this for the July 2011 code is about 367.9 seconds. For the prototype code, the equivalent call is call_analysismodc(virgo,0, 'VSR2-V3','pulsar_8',10,1,1,1,pulsar_8, 10 [190 200]). This code achieves the exact same results as the July 2011 code, but it only takes 255.7 seconds. Thus, the prototype code is an improvement that takes only about 70% of the time as the July 2011 code in this case. However, there is still the issue of needing to know observation in advance.

However, for a search of a greater band, the prototype code's become truly noticeable. For a search over 40 Hz from 160 to 200 Hz, the July 2011 code takes 730.8 seconds. As a note this is with a sub-band modification so that the sub-band is 40 Hz. The alternative would be to have a sub-band of 10 Hz and perform the search 10 Hz. This alternative constructs a Hough map for each sky point for each sub-band. Having a larger sub-band is obviously then the computationally cheaper alternative. The prototype code also used a 40 Hz sub-band to search from 160 to 200 Hz. The results were the same. However, this search took only 305.5 seconds. Now the prototype code only takes about 42% of the time it takes to run the July 2011 Hough transform.

Obviously, the results keep becoming more significant as the size of the search increases. For a 150 Hz search from 50 to 200 Hz, the July 2011 code takes 2251.2 seconds. However, for the same search with the prototype code we only need 514.7 seconds. Also, for a search of a 600 Hz band for the prototype code, it only requires 1673.5 seconds to perform a search from 0 to 600 Hz. This search was not able to be completed using the July 2011 code, because for a single machine the requirements of this search are prohibitive, and it was undesirable to devote significant resources to this type of test.

Conclusion

In summary, the frequency-spin down Hough transform is a recent improvement on the sky Hough transform for conducting incoherent data analysis and performing blind searches for continuous gravitational wave signals. Programs to perform this analysis have been built. However, they are currently undergoing optimization before being utilized in a grid environment with vast quantities of real detector data. Multiple improvements have been made to the May 2011 Hough code to lead to the July 2011 Hough code. Prototype Hough code is also in development that greatly outperforms this July 2011 Hough code.

Acknowledgements

Special thanks goes to the University of Florida International REU program for making this research possible through its aid and support. The NSF deserves thanks for its support of this research as well. Additionally, Fulvio Ricci, Cristiano Palomba, Sergio Frasca, and especially Pia Astone at Sapienza provided tremendous support and mentorship.

References

[1] Fairhurst S, et al 2011 Current status of gravitational-wave observations *Gen. Rel. Grav.* 43:388-408
[2] Palomba 2011 Searches for continuous gravitational wave signals and stochastic backgrounds in LIGO and VIRGO data
[3] Krishnan B, et al 2004 Hough transform search for continuous gravitational waves *Phys. Rev. D* 80 082001
[4] Astone P, Frasca S, Palomba C 2005 The short FFT Data Base and the peak map for the hierarchical search of periodic sources *Class. Quantum Grav.* S1198-S1210
[5] Frasca S 2001 Hierarchical Search for Periodic Sources *World Scientific*
[6] Frasca S, Astone P, Palomba C Evaluation of sensitivity and computing power for the Virgo hierarchical search for periodic sources
[7] Brocco L, Frasca S, Palomba C, Ricci F 2003 The search for continuous sources in the Virgo experiment. Full-sky incoherent step: 'local' and 'grid' tests *Class. Quantum Grav.* 20 S655-S664
[8] Antonucci F, et al 2008 Detection of periodic gravitational wave sources by Hough transform in the f versus f dot plane *Class. Quantum Grav.* 25 184015

Appendix

Selections of Matlab Code

Function hfdf_adaptivehough2010

Below is the Matlab code that performs an adaptive Hough transform on the peak map data. This code is part of the July 2011 Hough code.

```
function [hdf0]=hfdf_adaptivehough2010(antenna,sour,verb,hmap,peaks,ttdays,factdop)
%HFDF_HOUGH  creates a linear peakmap
%
%   hmap          hough map structure
%      .fr       [minf df enh nf] min fr, original step, enhancement
%                factor, number of fr
%      .d        [mind dd nd] min d, step, number of d
%   peaks(3,n)    peaks of the peakmap as [t,fr,pmean]
%   factdop(nt)   Doppler correction factor (if present)

%This program was developed initially by Pia Astone, and then edited by
%Carl Sabottke

inifr=hmap.fr(1); %int bottom of frequency subband
deltaf=hmap.fr(2); %
res=hmap.fr(3); %int resolution factor
ddf=deltaf/res;
nbin_f0=hmap.fr(4); %int number of bins in frequency dimension of hmap
dmin1=hmap.d(1); %
deltad=hmap.d(2);
nbin_d=hmap.d(3); %int number of bins in spin down dimension
n_of_ff=length(peaks);
ii=find(diff(peaks(1,:)));
ii=[ii n_of_ff];
nt=length(ii);
if ~exist('factdop','var')
   factdop=ones(1,nt);
end
ii0=1;
deltaf2=res/2; %int

%This is an alternative to the limit checks
%This is computationally much cheaper than the single vector bin checks
bump1=1000;
bump2=1000;
nbin_f0= nbin_f0 + bump1 + bump2;

%binh_df0=zeros(nbin_d,nbin_f0);
binh_df0a = zeros(nbin_d,nbin_f0);
binh_df0b = zeros(nbin_d,nbin_f0);


for it = 1:nt
```

```
   y = (peaks(2,ii0:ii(it))*factdop(it)-inifr)/ddf - deltaf2;  %One dimensional array of floats

   mm=peaks(3,ii0:ii(it));   %1-d array of floats

   t=peaks(1,ii0)/ddf;   %A float

   tdays=ttdays(ii(it));
   tsid=gmst(tdays)+antenna.long/15.0;  %local sidereal time
   sour.eps=0;   % non polarized sour
   radpat=radpat_interf(sour,antenna,tsid); %power radiation pattern

 inc = radpat/median(mm); %Float

   ii0=ii(it)+1;

 for id = 1:nbin_d
             d=dmin1+(id-1)*deltad;
             td=d*t;

     a = bump1 + round(y-td);

 binh_df0a(id,a)=binh_df0a(id,a) + inc;

 end

end


for br=1:(nbin_f0-res)
binh_df0b(:,br+res)=-(binh_df0a(:,br));
end

binh_df0 = binh_df0a + binh_df0b;

binh_df0=cumsum(binh_df0,2);

%Limit compensation
nbin_f0 = nbin_f0 - bump2;
binh_df0=binh_df0(:,bump1:nbin_f0);

hdf0=gd2(binh_df0');
hdf0=edit_gd2(hdf0,'dx',ddf,'ini',inifr,'dx2',deltad,'ini2',dmin1,'capt','Histogram of spin-f0');

end
```

Function analysismodc1

Below is a significant portion of the prototype Matlab code. This is the code that performs analysis on a sub-band, which is generated by the function call_analysismodc. Currently a function called analysismodc2 is in development, which will be called by call_analysismodc2. This prototype code makes even further improvements upon the code cited here and is more adjustable with regards to searches of large numbers of sky positions than analysismodc1 and call_analysismodc.

```
function
[hdf1,firstfile]=analysismodc1(antenna,verb,decadir,sourstring,res,doppleryes,adaptive,ibandina,inifr,ifif
r,file,sour,skymap)
% analysis.m  analyzes the vbl peakmaps data
% In a vbl-file the first channel are the parameters, the second the bins,
% the third the amplitudes; other channels can contain short spectra and other.
% Input parameters: see call_analysis.m
% Output parameters:
% hdf1: a gd2 with the Hough map
% firstfile is the name of the first file to be analyzed for each band. At the beginning it is asked the name.
% A log file with results
%The Hough transform code was originally written by Pia Astone and has been
%extensively modified by Carl Sabottke

%It builds up partial maps for each sky point and keeps them in memory
%while it reads through the FTs.  It should, therefore, not be set to
%search too many points with the same machine.
format long;
snag_local_symbols;
thr=0;   %%means use all peaks in the peakmap
time=0;  %%means use all the data in time
Nsky=length(skymap);

logfilename= strcat('Adaptivehough_',sourstring,'.txt');

firstfile=file;

if (ibandina==1)
   %%Creates  the log file for all the time decades examined. Written in the directory under analysis
   %%iprimo =0 means analysis of the first band
    logfile=fopen(logfilename,'w');
else  %%analysis of the next bands
   %%Re-open in append mode the log file for the time decade examined
    logfile=fopen(logfilename,'a');
end

frband=[inifr ififr];

%Begin the crea_timefrequency stuff
```

```
%Find a way to know how much will be read

len=0;  %to easily go through all the ffts.


 if(strcmp(file,'tobea')==1)
   [file,pname,fname]=selfile('  ');
end


 if(strcmp(sourstring,'instrument')==1)
display(Nsky);
 end

if thr == 0
   thr=[0 1e100];
end
if time == 0
   time=[0 1000000];
end

vbl_=vbl_open(file);
ictyp=1;
if vbl_.nch == 3
   disp('Type 1 peakmap file')
   ich=2;
elseif vbl_.nch == 5
   disp('Type 2 peakmap file')
   ich=4;
   ictyp=2;
elseif vbl_.nch == 6
   disp('Type 3 peakmap file')
   ich=4;
   ictyp=3;
else
   disp('*** Attention ! strange vbl file')
end
dfr=vbl_.ch(ich).dx; %Float or double
lfft=vbl_.ch(ich).lenx*2;
if dfr == 0
   disp('*** errors in fft parameters ! Use a default set')
   lfft=4194304;
   dfr=4096/lfft;
end
binmin=floor(frband(1)/dfr); %Ints
binmax=ceil(frband(2)/dfr);  %Because sorting is done prior to the application of the Doppler Effect and
spin down, issues arise with indices
vbl_.nextblock=0;
```

```
if len == 0
   disp('*** No length limit - cut at 1000000')
   len=1000000;
end

%Need a way to find observation time
%This can be made an input or code can be written to find this quickly
observation_time = 8.106499999998137e+05; %Float or double

incr=1;  % For the spin down bins width: one bin (if 1) or  half bin (if 2)
ddf=dfr/res; %Float or double

deltad=dfr/(incr*observation_time);%Float or double
nbin_d= 10;%Int

%dmax is 0 in this configuration
  dmin=-deltad*nbin_d;%Float or double

dmax=dmin+deltad*nbin_d;%Float or double if not 0
bump1=1000; %These help avoid indexing errors, ints
bump2=1000;
nbin_f0=bump1+bump2+ceil((frband(2)-frband(1))/ddf);

deltaf=res/2;

%We want to preallocate arrays of zeroes
%In C, we will use calloc.  Using calloc for a 2-d array and a 3-d array is
%not as simple as for a 3-d array.  We also need to preallocate certain
%arrays in C that we don't need to preallocate in Matlab, like binh_df0
binh_df0a = zeros(nbin_f0, nbin_d,Nsky);
binh_df0b = zeros(nbin_f0,nbin_d);

%Now we have Hough information and will begin reading the files

kbl=0;
%%It will be ideal to know how many files will be read in advance
%%This will allow observation time to be known ask
%Big read start
   while vbl_.eof == 0

   kbl=kbl+1; %Counter update

   if kbl > len
      break
   end
   vbl_=vbl_nextbl(vbl_);
%   to continue over files
   if vbl_.eof > 0
```

```
   fclose(vbl_.fid);
   file=[vbl_.filppost dirsep vbl_.filspost];
   vbl_=vbl_open(file);
   if vbl_.eof == 2
      break
   end
   vbl_=vbl_nextbl(vbl_);
end
%What is this? ask
kch=vbl_.block;
tim=vbl_.bltime; %Time, double
if tim > time(1) && tim < time(2)
   vbl_=vbl_headchr(vbl_);
   if vbl_.ch0.lenx == 0
      continue
   end
   vel=fread(vbl_.fid,vbl_.ch(1).lenx,'double'); % p09
   if (ictyp == 2 || ictyp == 3)
      vbl_=vbl_nextch(vbl_);
      lensp=vbl_.ch0.lenx;
      inisp=vbl_.ch0.inix;  %not used
      dfsp=vbl_.ch0.dx;
      sp=fread(vbl_.fid,lensp,'float32');
      vbl_=vbl_nextch(vbl_);
   end
   vbl_=vbl_nextch(vbl_); %%does not read INDEX
   npeak=vbl_.ch0.lenx; %Length of the list for each time, int



   bin=fread(vbl_.fid,npeak,'int32');   %bins of the peaks, at the time tim
   vbl_=vbl_nextch(vbl_);
   amp=fread(vbl_.fid,npeak,'float32'); %peak CR of the peaks at the time tim
   if (ictyp ==3)
   vbl_=vbl_nextch(vbl_);
    pmean=fread(vbl_.fid,npeak,'float32'); %peak mean of the peaks at the time tim
   end
   %This is time consuming but might be the best, in C this gets
   %trickier.  We will need to use if statements.
   ibin=find(bin>=binmin&bin<=binmax);

   if length(ibin) < 1
      kbl=kbl-1;
      continue
   end
   ibmin=ibin(1);
   ibmax=ibin(length(ibin));
   bin=bin(ibmin:ibmax);
```

```
      amp=amp(ibmin:ibmax);
      pmean=pmean(ibmin:ibmax);

       ibin=find(amp>=thr(1)&amp<=thr(2));
       ff=bin(ibin)*dfr;
       ff = ff';
       mm=pmean(ibin);


 if kbl==1
   epoch=tim;
 end

if(strcmp(sourstring,'instrument')==0)
 epochsource=sour.fepoch;
 Fexp=sour.f0+sour.df0*(epoch-epochsource)*86400.0;
end

   tdays=tim;

tim=(tim-epoch)*86400;

%Need a way to get this
%observation_time=max(tt);

t=tim/ddf;

   tsid=gmst(tdays)+antenna.long/15.0;  %local sidereal time
   sour.eps=0;   % non polarized sour
   radpat=radpat_interf(sour,antenna,tsid); %power radiation pattern

    inc = radpat/median(mm);%Float or double

         for iS=1:Nsky
            lambda=skymap(iS,1);
            beta=skymap(iS,2);

            factdopC=doppler_correctionfromvbl(lambda,beta,vel);

             y = (ff*factdopC-inifr)/ddf - deltaf;


 for id = 1:nbin_d
              d=dmin+(id-1)*deltad;
              td=d*t;
    a= bump1 + floor(y-td);
    %binh_df0a(id,a,iS)=binh_df0(id,a,iS) +inc;
binh_df0a(a,id,iS)=binh_df0a(a,id,iS) + inc;
```

```
 end

        end

  end

  end %End read
 display(kbl);
display(t);

%Now add the stuff from analysis
dt=1/(dfr*lfft);

if (ibandina==1)
   fprintf(logfile,'first file of the decade  %s  \n', file);
   fprintf(logfile,'original sampling time [s] %f  \n', dt);
   fprintf(logfile,'lenght of the FFTs  %d  \n', lfft);
   fprintf(logfile,'frequency step [Hz] %f  \n', dfr);
   fprintf(logfile,'Beginning mjd time  %f  \n', epoch);
   if(strcmp(sourstring,'instrument')==0)
      fprintf(logfile,'Source name           %s  \n', sourstring);
      fprintf(logfile,'Source reference mjd epoch  %f  \n', sour.fepoch);
      fprintf(logfile,'Source freq at epoch source [Hz] %f  \n', sour.f0);
      fprintf(logfile,'Source spin down times 10^8  [Hz/s] %f  \n', 10^8*sour.df0);
      fprintf(logfile,'Used mjd epoch  %f  \n', epoch);
      fprintf(logfile,'Expected source frequency at the analysis beginning time [Hz] %f  \n', Fexp);
   end
end

if (ibandina==1)
   fprintf(logfile,'spin down min,max,step  [Hz/s] %e %e %e \n', dmin,dmax,deltad);
   fprintf(logfile,'frequency over resolution factor %d\n', res);
   fprintf(logfile,'Total FFTs number %d \n',kbl);
end

fprintf(logfile,'FrBand  [Hz] %f %f \n', inifr,ififr);

   for iS=1:Nsky

          for br=1:(nbin_f0-res)
binh_df0b(br+res,:)=binh_df0a(br,:,iS);
       end
%        for br=1:(nbin_f0-res)
%binh_df0b(:,br+res)=binh_df0a(:,br,iS);
%        end

binh_df0 = binh_df0a(:,:,iS) - binh_df0b;
```

```
binh_df0=cumsum(binh_df0,1);
%binh_df0=cumsum(binh_df0,2);
nbin_f0f=nbin_f0 - bump2;
%binh_df0=binh_df0(:,bump1:nbin_f0f);
binh_df0=binh_df0(bump1:nbin_f0f,:);

hdf1=gd2(binh_df0);
hdf1=edit_gd2(hdf1,'dx',ddf,'ini',inifr,'dx2',deltad,'ini2',dmin,'capt','Histogram of spin-f0');

      if(iS<=2)  %2 is arbitrary. does only 2 figures.
         plot(hdf1)
         xlabel('f_0 [Hz]')
         ylabel('spin-down [Hz/s]')
         title('Frequency Hough map in the selected band')
      end

      selhough=y_gd2(hdf1);
      selhough=selhough(:);
      average=mean(selhough(:));
      mediana=median(selhough(:));
      sigma=std(selhough(:));
      % thres on the percentage of data, FA
       ys=sort(selhough(:));
       dims=length(ys);
       %is=ceil(dims-dims*1/100);
       is=ceil(dims-10); %only the top 10 . per provare pia.
       thres=ys(is);
      [xp,yp,zp]=twod_peaks(hdf1,thres);
      display('spin down times 10^8');
      display('iS,Nsky, percentage analyzed');
      display(iS);
      display(Nsky);
      display(iS/Nsky);
      show_twod_peaks(xp,yp*10^8,zp,2);
      fprintf(logfile,'Sky grid position num alfa delta;  %d   %f   %f \n',iS,skymap(iS,1),skymap(iS,2));
      fprintf(logfile,'Average, Median and std of the hough map;  %f   %f   %f \n',average, mediana,sigma);
      fprintf(logfile,'threshold on the percentage, FA percentage %f %f \n', thres,1-(dims-10)/dims);
      freqspinamp=[xp',10^8*yp',zp',(zp'-average)/sigma];
      fprintf(logfile,'frequency,  spin down times 10^8,  number of peaks, CR mean\n');
      fprintf(logfile,'%f   %f          %f       %f    \n', freqspinamp');

   end
save prova.mat
```