LASER INTERFEROMETER GRAVITATIONAL WAVE OBSERVATORY
- LIGO -
CALIFORNIA INSTITUTE OF TECHNOLOGY
MASSACHUSETTS INSTITUTE OF TECHNOLOGY

Presentation	**LIGO-G1601123-v2**	2017/01/02

Extending a Plotting Application and Finding Hardware Injections for the LIGO Open Science Center

Nicolas Rothbacher, Mentors: Eric Fries, Jonah Kanner, Alan Weinstein
University of Puget Sound

Distribution of this document:
LIGO Scientific Collaboration

California Institute of Technology
LIGO Project, MS 18-34
Pasadena, CA 91125
Phone (626) 395-2129
Fax (626) 304-9834
E-mail: info@ligo.caltech.edu

Massachusetts Institute of Technology
LIGO Project, Room NW17-161
Cambridge, MA 02139
Phone (617) 253-4824
Fax (617) 253-7014
E-mail: info@ligo.mit.edu

LIGO Hanford Observatory
Route 10, Mile Marker 2
Richland, WA 99352
Phone (509) 372-8106
Fax (509) 372-8137
E-mail: info@ligo.caltech.edu

LIGO Livingston Observatory
19100 LIGO Lane
Livingston, LA 70754
Phone (225) 686-3100
Fax (225) 686-7189
E-mail: info@ligo.caltech.edu

http://www.ligo.caltech.edu/

Abstract

During the first science run of the recently completed Advanced Laser Interferometer Gravitational-Wave Observatory (aLIGO) detectors, two gravitational waves were detected, the first such detections in history. When the rest of the data is released to the public, meaningful data representation will be necessary for communication to both informed viewers and the general public. In its current state, LIGO's data representation is largely static and clumsy in nature, relying on Python to generate images of plots or simply distributing numerical data. This project sought to upgrade one of LIGO's plotting programs, splot, to feature interactive zooming and software design compatible with more expansive use. During the course of the project, splot was upgraded to use the plotly.js JavaScript library to dynamically zoom and display information as well as Django templates to eliminate the need for hard drive space when generating the plots, and other minor improvements. This work makes splot much more informative to users of all kinds and establishes a framework for future features that is friendly to larger audiences. Also prior to the data release, all signals that were injected using control mechanisms for detector testing must be catalogued. To make this catalog, a match filter was constructed and run over many of the known hardware injections over the course of O1, the first data collection run for aLIGO. The results were compiled into a table and weaknesses in the search were identified.

1 Introduction to Gravitational Wave Research

Albert Einstein first theorized gravitational waves in 1916 as a consequence of his theory of general relativity. The predicted amplitudes of these waves were incredibly small, making observation of these waves difficult [1]. In the 1960s, Joseph Weber led the first serious efforts to detect gravitational waves [2]. Weber's original design used a large mass of aluminum and a piezoelectric detector to sense changes in length of the aluminum. In the late 1960s and 1970s, prototype interferometer detectors were constructed, and in 1980 the NSF funded the construction of a 40-meter prototype at Caltech and a 1.5-meter prototype at MIT. These prototypes demonstrated the feasibility of a large interferometer and led to the creation of the Laser Interferometer Gravitational-Wave Observatory (LIGO) [3]. On September 14th, 2015 Advanced LIGO (aLIGO) directly detected a gravitational wave [4].

aLIGO is pair of detectors constructed using a modified Michelson interferometer design: one in Hanford, Washington and the other in Livingston, Louisiana. The detectors are separated by 10 ms of light travel time, and are oriented slightly off-axis from one another. The interferometers use 1064 nm lasers and $L = 4$ km arms, along with a number of signal enhancements. A passing gravitational wave induces a change in the length of the arms $\Delta L(t) = \delta L_x - \delta L_y = h(t)L$, where h is the strain amplitude projected onto the detector and δL_x and δL_y are the changes in arm length. h is the primary output channel, and the sensitivity of this output is determined by two main factors: the antenna pattern and the frequency of the wave detected. The antenna pattern is dependent on the sky position of the binary merger that produced the gravitational wave, the inclination of the binary orbit, and the orientation of the arms of the detector [5].

Figure 1 shows the amplitude spectral density of the primary output, demonstrating a variation in

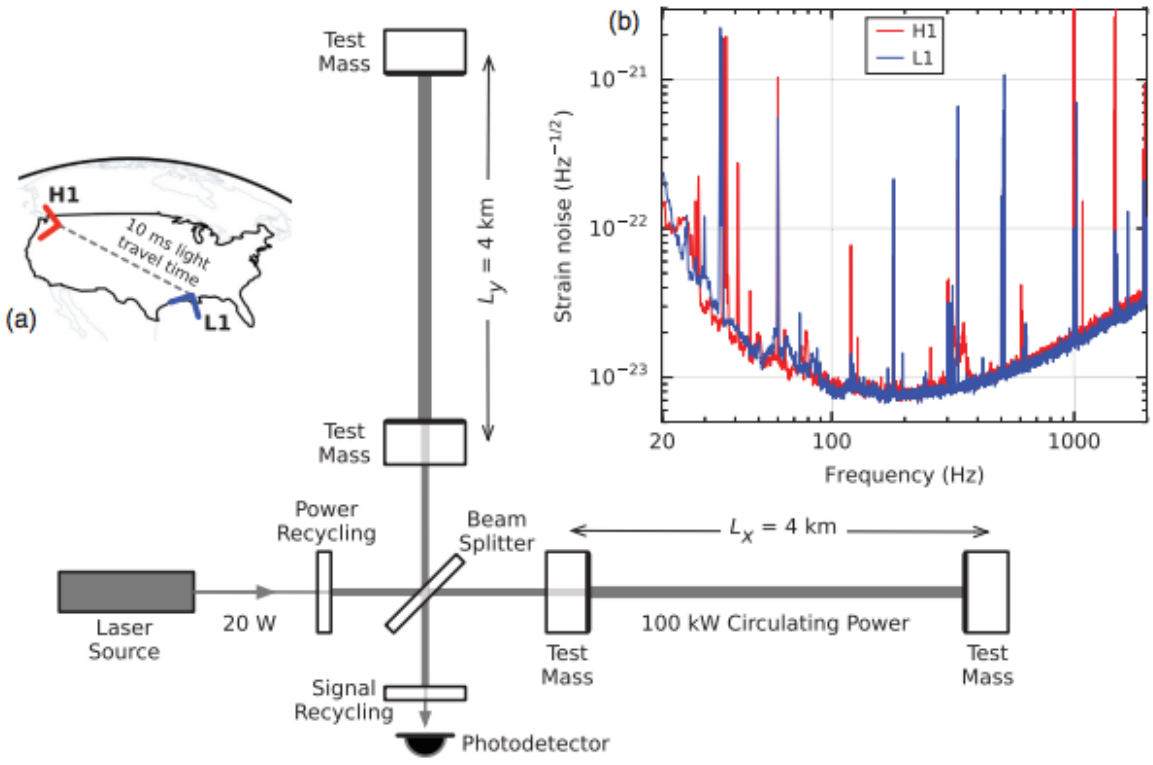

Figure 1: a) Arrangement of detectors and configuration of interferometer. b) Strain noise vs. frequency, showing the highest strain sensitivity in the middle part of the frequency spectrum.

sensitivity dependent on the signal frequency. This variation in strain sensitivity is due to different sources of noise. At low frequency, the limiting noise source is seismic vibration. At intermediate frequencies it is Brownian thermal noise originating from the mirror coating [6]. At high frequencies, the quantum shot noise produced by the Heisenberg uncertainty principle become the major noise source.

The gravitational wave event detected on September 14, 2015, GW150914, swept through frequencies from 35 to 250 Hz in 0.2 s with a maximum strain amplitude of 1.0 x 10^{-21} occurring at the highest frequency. GW150914 is the first direct detection of gravitational waves. This event has great historical value, and it is vital that the data and analysis be presented to the public in an effective and understandable format. The second gravitational wave event detected by LIGO, GW151226 was more faint, with a maximum strain of 3.4 x 10^{-22}, sweeping from 35 to 450 Hz in 1 s [7]. Having an understandable representation for this event will be even more important than for GW150914, since it is less apparent on first glance, making signal processing vital to identifying the signal.

2 LIGO Data Processing and splot's Place in the Picture

To process these signals and parse out the noise, LIGO performs a range of calculations and creates a number of plots. The first of these is calculated from the power spectral density of the data. A power spectral density (PSD) characterizes the stationary noise of the data by averaging the square of the fast Fourier transform (FFT) over all time. An amplitude spectral density (ASD) is then calculated as the square root of the power spectral density; the resulting plot is shown in Figure 1 b. This set of amplitude data in the frequency domain is then used to "whiten" the strain data of the signal by using FFT to convert the strain to the frequency domain, dividing by the ASD and using an inverse FFT to return to the time domain. The resulting time series is shown in the top row of Figure 2. A spectogram can be calculated to show the frequency and amplitude behavior of the data. An optimised versions of this graph is shown on the bottom row of Figure 2. Bandpass filtering and matching to theoretical models the data are also techniques employed to further confirm and clarify the gravitational wave signal [8].

This project sought to expand the existing plotting software used by LIGO to represent the data in an interactive and publicly useful manner. Prior to the project, splot generated static plots of a time series of the strain, as well as a spectrogram and ASD of the strain. splot could also apply various types of filters to the data, adjust the sampling rate, and perform similar data manipulations. The size, scale and parameters of the plots were set prior to submitting a request, making changing these parameters a slow and awkward process. The plots were also stored on the server as image files referenced in an HTML document, a very hard drive space intensive solution. The goal of this project was to modify this functionality to include interactivity and increase efficiency by moving plotting to the client. At the conclusion of this project all of the representations mentioned above and interactive elements were incorporated into splot, the web-based plotting interface.

3 Improvements Made to splot

This project made two main changes to splot: it introduced interactive features and reduced the load on the server by implementing Django template rendering. The primary objective was to expand splot to allow the user to interactively zoom in on different parts of the plots and dynamically zoom multiple plots to better illustrate the nature of the data. Both of these features allow real time manipulation and do not require the user to reload the page. The secondary objective was to reprogram much of the data management to utilize the LIGO Open Science Center's Django web framework. This improvement eliminated the need to store any plots on the server and simplified the code-base.

Several JavaScript libraries were investigated to implement interactive elements, including mpld3 [10], bokeh [11] and plotly.js [12]. mpld3 and bokeh were both not suited to the task as they take an already generated Python plot and generate a JavaScript version. This does not work as well with the Django template tool, so plotly.js was determined to be more applicable as it simply receives data and creates a plot on the users' computer. Importantly, plotly.js also has functionality for all the representations that LIGO outputs: heatmaps for the spectrograms and Omega-scan, log-log

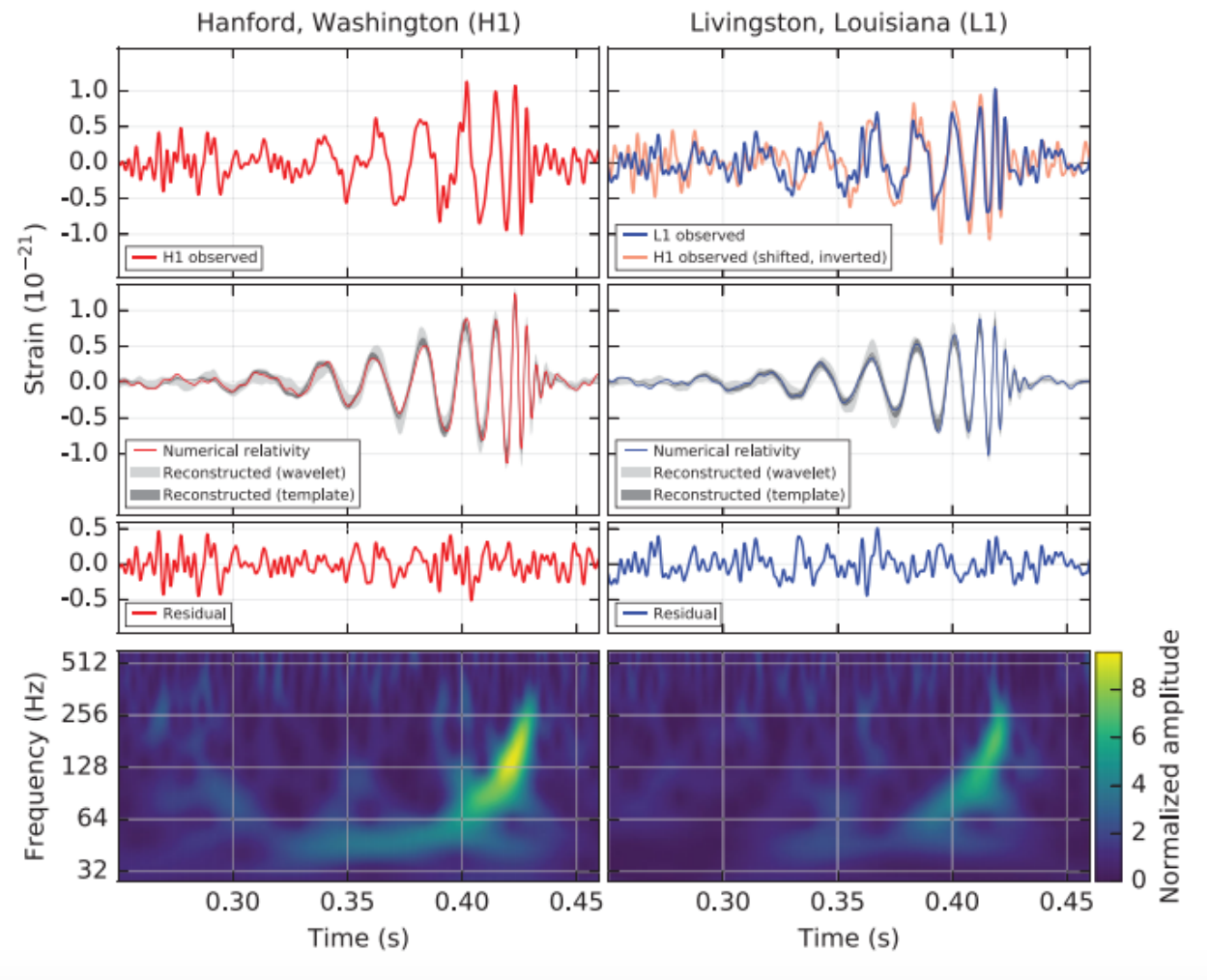

Figure 2: Two sets of figures from GW150914. The left column is from the Hanford detector and the right from the Livingston detector. The top row shows whitened gravitational wave strain over time. The second row shows a combination of different predictive techniques for theoretical wave models. The third row shows the residuals of the signal when the theoretical model is subtracted. The bottom row is a spectrogram of the data.

plots for the ASD and standard linked scatter plots for time series. plotly.js also has a feature that allows multiple plots to be linked. Using this, code was written to scale the frequency and time axis of the spectrogram and the corresponding axes on the time-series and ASD to the same values so each plot displays the same data.

The second objective was achieved using Django's template rendering language. It allows server requests to fill data from a Python back end into an HTML template file to be sent to the user [13]. This framework plays very well with the way that splot is already implemented, as this template rendering language simply takes Python objects and inserts the values into the HTML document and the data calculated by splot is already stored in Python objects. With this tool, the process for generating plots with splot became: the user submits a request, the server does calculations and sends a dictionary with the calculated values to the places where Django template tags are inserted

into an HTML document. Finally plotly.js takes this information and uses it to generate a plot in the user's browser.

Some auxiliary improvements for splot were made to the form for the input parameters by adding help boxes coded in CSS and particular events easily selectable by the user for plotting. An additional small improvement was combining the originally separate pages for the different data collection runs into one page with a parameter for the data run allowing easier access to all possible plots. An animated loading message was also added to remove any confusion about load times.

4 Further expansion for splot

In the course of the project, much of the existing error catching in the input parameters was disabled. A basic expansion could implement these in the new front end framework. This would be a minor upgrade that simply requires the adaptation of existing code. Another simple, related step would be to reexamine the parameters used and to eliminate or expand features as necessary to the utility of the user. Another easy upgrade would be to increase the amount of information provided to the user on the form page. Currently the list of hardware injections just states the type of injection, with no mention of parameters and this could be expanded to a very detailed list and include many more examples.

A more ambitious goal would be to allow the user to produce an Omega-scan, a correlogram, and an audio file of the strain. An Omega-scan, shown at the bottom of Figure 2, is a version of a spectrogram that tunes input parameters to maximize the power recovered by the spectrogram. A correlogram displays the correlation between the strains in the two detectors, after correcting for differences in phase and shifting in time. This will allow the user to see the importance of having two detectors and to visualize how the two detectors confirm the observation. An audio file is of interest as the signals are usually in the frequency band that is audible to the human ear. These features could be somewhat more difficult for two technical reasons. First, the servers may not be able to return an Omega-scan or a correlogram in a timely manner. In this case, the possibility the user submitting a request for the plot and alerting the user when the plot is ready should be investigated. Second, the integration may be difficult because it is plausible the existing analysis code cannot be easily merged with splot. Either of these technical issues may make the objective intractable.

Even further expansion of splot could include the ability to manipulate parameters in real time, or include some ability to filter the data in real time, going through the process step by step. These features will provide an resource to educate the public on the data processing that LIGO data undergoes and provide a more versatile experience for informed users. This particular objective may be very difficult as it would necessitate a change in the entirety of splot as all of the data processing is currently done in Python on the server and real time implementation would require a completely different implementation of that or perhaps a different programming language.

5 Hardware Injections and the Need for Documentation

Hardware injections are simulated signals that are injected into the real output of the detectors by moving the test masses as they might be expected to move when a real gravitational wave passes. These injections are used to test search pipelines under situations that might be seen in true gravitational wave events, with noise levels that are realistic, as well as to characterize the nature of the instrument.

A vital step to releasing the O1 data to the public is to catalog these hardware injections in the final data. This information must be recorded with as much detail as possible, particularly in regards to their signal to noise ratios (SNRs), to avoid any confusion in analyses that could be done after the data set release. Prior to this project, the level of documentation of the hardware injections for O1 was below what was needed. Records were compiled and a match filter was written to analyze the hardware injections and create a full picture of the injections performed in O1.

6 Matched Filtering

To understand how the hardware injections manifested in the data, each injection was analyzed using a matched filter. A matched filter is an algorithm that searches over a section of data for the times when the data matches a given template. This is very efficient for hardware injections since templates exactly like what would show up in the data are used to actually do the injection and matched filtering is a very effective search tool for signals buried in noise like those in the LIGO data.

The matched filter used in this project was the FINDCHIRP algorithm as described by Allen, et al [9]. This algorithm can be written as equation 1:

$$z(t) = 4\int_0^\infty \frac{\tilde{s}(f)\tilde{h}^*_{template}(f)}{S_n(f)} e^{2\pi i f t} df \tag{1}$$

where $z(t)$ is the complex valued output of the filter, $\tilde{s}(f)$ is the FFT of the detector signal, $\tilde{h}^*_{template}(f)$ is the complex conjugate of the FFT of a template and $S_n(f)$ is the stationary Gaussian noise of the output as a power spectral density. To find the SNR of a signal found with this filter, it must be normalized with the factor in equation 2:

$$\sigma_m^2 = 4\int_0^\infty \frac{|\tilde{h}_{D_{eff}Mpc,m}(f)|^2}{S_n(f)} df \tag{2}$$

The value σ_m is a measure of the detector sensitivity with a template, h_m of effective distance D_{eff}. The effective distance is a relationship of the true distance of a source to its orientation to the detector orientation. Then the amplitude SNR, ρ_m, of a signal with this template can be found using equation 3:

$$\rho_m(t) = \frac{|z_m(t)|}{\sigma_m} \tag{3}$$

This is the value of particular interest for this task, as the SNR of the hardware injection output should match the expected SNR of the template if an injection was successful and recovered fully.

7 The Hardware Injection Study

The FINDCHIRP algorithm was performed over a list of hardware injections found in a schedule file [14] that represented the most complete list of hardware injections performed in O1 (as well as one additional injection found in a LIGO wiki page [15]). The schedule was the reference for injection times and template and parameter files. This schedule file appears to record injections that were requested by the burst and compact binary coalescence detection groups, but not the detector characterization injections (the third largest group). The recovery of the 80% of injections that occurred while the detector they were injected into was collecting data are shown in Figure 3. The majority of these injections are distributed along a line of slope 1, indicating that these injections succeeded and were recovered. There is a distinct minority of injections (34%) that do not adhere to this problem though, lying along the horizontal axis instead. Such low recovered SNR, especially with the very high SNR templates, indicates that these injections were unsuccessful for some reason, probably not making it into the final LIGO strain data.

Parameters for each injection were then extracted from the parameter files referenced by name in the schedule file. Each injection was then paired with its parameter information and the recovered and expected SNRs. This information was compiled into a table to document injections recovered here for others to use as a reference for the O1 data set. This table is not complete in all of this information, since many of the parameter files for the injections were missing from the repository that held the rest of the parameter files.

8 Future Work for the Hardware Injection Search

The first improvement that should be made to this study of hardware injections should be the scanning of injections from the detector characterization group since documentation does exist for these injections. After this expansion of the study, the edge cases already identified by the study should be investigated. Many of the injections lie in the range of SNR that could be considered unsuccessful, but this should be confirmed and the details of these failures identified. The injections with missing parameter files should also be researched further to identify the parameters used to generate the templates and the schedule file should be checked against other resources to ensure completeness. One way to verify this schedule would be to check the control channels used to inject the signal to the detector. These channels represent the most complete resource for hardware injection logging since there is no other way to inject signal to the detector, but an entirely new methodology will need to be developed to scan these channels. One final improvement could be the generalization of the algorithm used to search for these injections to future data sets so that similar studies in the future will be easier.

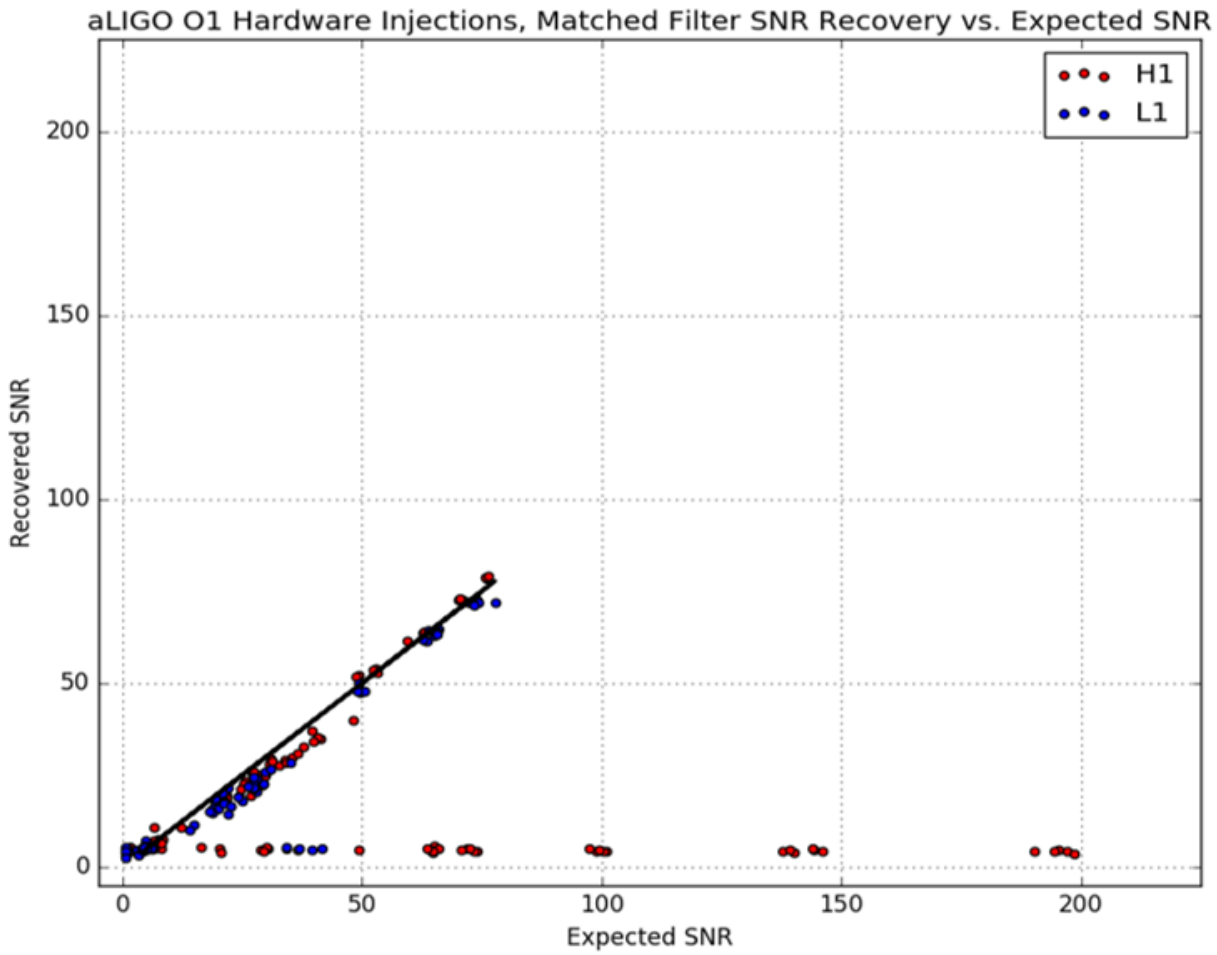

Figure 3: The injections that occurred during time when the detector was in science mode are shown plotted with the SNR recovered by the matched filter on the y-axis and the SNR expected from the template on the x-axis.

Acknowledgements

I would like to express my deepest gratitude toward my mentors, Eric Fries, Jonah Kanner and Alan Weinstein, without whom this study could never have been completed. Their expertise and direction was probably the most important factor to my success with both parts to my project.

In addition, I want to thank the other LIGO SURF students who worked in my office who contributed considerably over the course of the project. Their willingness to hear my thoughts and help me with understanding of basic topics was invaluable. I also want to thank Mykyta Hulko for contributing significant code to the hardware injections project.

The hardware injection project could not have succeeded without the help of the hardware injection team, so big thanks to them as well.

Finally, I want to thank the LIGO SURF Program for organizing such an amazing opportunity and

Caltech for graciously hosting me in the SURF program as well as the NSF for funding my REU grant.

References

[1] A. Einstein, Sitzungsber. K. Preuss. Akad. Wiss. 1, 688 (1916).

[2] LIGO, *A Brief History of LIGO*,
https://www.ligo.caltech.edu/system/media_files/binaries/313/original/LIGOHistory.pdf

[3] J. Weber, Phys. Rev. 117, 306 (1960).

[4] B. P. Abbott et al., PRL **116**, 061102 (2016).

[5] D. G. Keppel, Doctoral Thesis, 33 (2009).

[6] S. J. Waldman, *The Advanced LIGO Gravitational Wave Detector*,
https://arxiv.org/pdf/1103.2728.pdf

[7] B.P. Abbott et al., Phys. Rev. Lett. **116**, 241103 (2016).

[8] LIGO Open Science Center, *Signal Processing with GW150914 Open Data*
https://losc.ligo.org/s/events/GW150914/GW150914_tutorial.html

[9] B. Allen et al., Phys. Rev. D **85**, 122006 (2012).

[10] mpld3, http://mpld3.github.io/

[11] bokeh, http://bokeh.pydata.org/

[12] Plotly.js, https://plot.ly/javascript

[13] Django templates, https://docs.djangoproject.com/en/1.9/ref/templates/

[14] Schedule of O1 Hardware Injections. https://daqsvn.ligo-la.caltech.edu/svn/injection/hwinj/Details/tinj/schedule

[15] O1 Injections, LIGO Wiki. https://wiki.ligo.org/Main/O1Injections