Identifying Possible Binary Brown Dwarf Systems

Using Point Spread Function Model Fitting

Loic Beus

A senior thesis submitted to the faculty of
Brigham Young University
in partial fulfillment of the requirements for the degree of

Bachelor of Science

Dr. Denise C. Stephens, Advisor

Department of Physics and Astronomy

Brigham Young University

ABSTRACT

Identifying Possible Binary Brown Dwarf Systems
Using Point Spread Function Model Fitting

Loic Beus
Department of Physics and Astronomy, BYU
Bachelor of Science

Observations of brown dwarf systems have shown that current atmospheric and evolutionary models do not always agree with observations. Unlike main sequence stars, brown dwarf's do not have a clear relationship between mass and luminosity which makes it difficult to determine parameters like mass and age which are important for developing models. To improve the models, more data on brown dwarfs with known mass, luminosity, and age are required. One of the best methods to determine a brown dwarf's mass is through the orbital interactions of a binary pair. However, brown dwarf systems are very dim and very far from earth, so binary systems are often imaged as a single point of light in a pattern known as a point spread function (PSF). The Hubble Space Telescope (HST) has observed a large number of brown dwarfs that could potentially be unresolved binary systems. I developed a model fitting code that improves upon the Python script it is based on. The program fits the HST images to PSF models to identify potential binary brown dwarf systems.

Keywords: brown dwarf, binary, Hubble Space Telescope, HST, point spread function, PSF, Python, C++

ACKNOWLEDGMENTS

I acknowledge the Brigham Young University Physics Department and the Office of Research and Creative Activities for providing the funding and support to make this research possible.

I would like to thank Dr. Denise Stephens for being an amazing research advisor and for giving me the opportunity to work on this project. Dr. Denise Stephens helped me to learn and grow throughout my undergraduate studies and gain a deeper appreciation for astronomy. I would also like to thank Dr. Tom Stephens for his advice and coding expertise throughout my research.

I would also like to thank my parents for teaching me to have a love for learning. Their support throughout my life and my studies has helped me get to where I am today, I couldn't have done it without them.

Contents

Chapter 1

Introduction

1.1 Introduction to Brown Dwarfs

A brown dwarf is a celestial object whose mass falls between that of the largest planets and the smallest of stars. Stars begin their lives as a large cloud of interstellar gas that begins to collapse on itself. As the gas continues to collapse, its energy and density increase until the temperature and pressure is high enough to ignite the fusion of hydrogen in its core. This creates pressure that balances out the force of gravity and prevents any further collapse, thus creating a star. The fusion at its core is what gives a star its energy, and its luminosity remains relatively constant through most of its main sequence life.

During the later stages of star formation, a disk of dust and gas surrounds the star and slowly begins to clump together and accumulate particles until they become large enough to create a planet. Brown dwarfs develop in the same way as a star does through the collapse of a cloud of gas, however, these gas clouds are less massive and never reach the conditions necessary to ignite hydrogen fusion in their core. A star requires about 80 Jupiter masses to ignite hydrogen fusion in its core, and brown dwarfs range between 10 and 75 Jupiter masses. Without the fusion in its core

to create energy, brown dwarfs slowly cool and become fainter over time.

1.2 Brown Dwarf Classification

Like stars, brown dwarfs are classified based on their spectra, known as spectral classifications. A star or brown dwarf's spectra is the measurement of intensity by wavelength of the light emitting from the object. Spectral features are determined primarily by the object's temperature. The four classes of brown dwarfs in order of highest to lowest temperature are M, L, T, and Y type. Since brown dwarfs do not fuse hydrogen in their cores, their spectral classification changes as they age and cool. The initial mass of the brown dwarf largely determines its initial spectral classification and cooling rate.

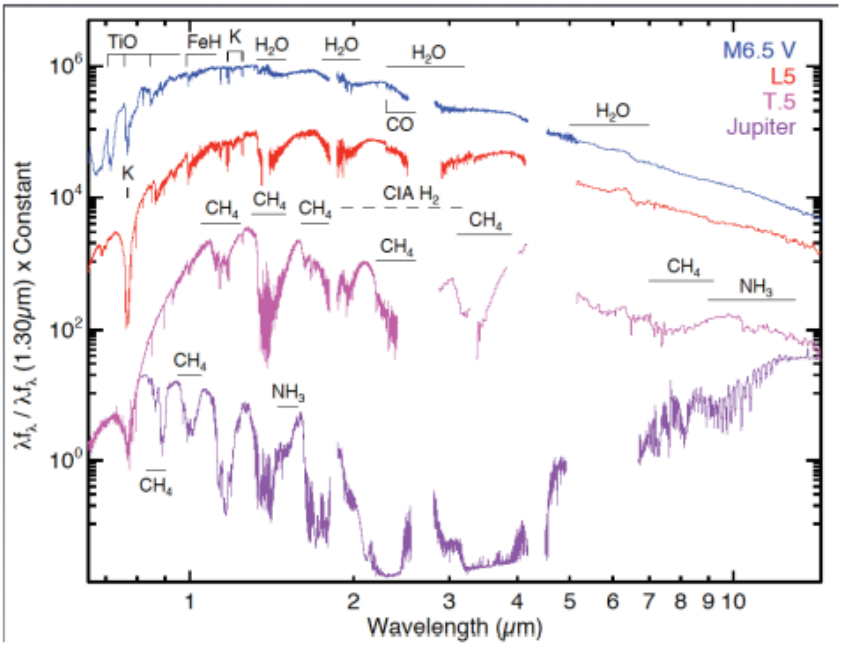

Figure 1.1 A comparison of spectra from various objects. From top to bottom: M star, L dwarf, T dwarf, Jupiter (planet). The gaps in the spectra are due to the earth's atmosphere absorbing specific wavelengths of light. This figure has been modified from (Marley & Leggett 2009).

The cooling rate of a brown dwarf determines how it progresses through the spectral types. However, unlike stars, brown dwarfs do not have a clear relationship between mass and luminosity which make it difficult to determine parameters like its mass and age. (Dupuy et al. 2014) showed that theoretical cooling models predicted incorrect mass, age, and luminosity values for brown dwarfs he studied in binary systems with main sequence stars of known distance where he was able to directly measure these parameters for the brown dwarf. To create a more accurate model more data on brown dwarfs with known age, mass, and luminosity is required. To get this data we need to find more binary brown dwarf systems as we can only directly measure the mass from the gravitational influence of the two bodies interacting with each other.

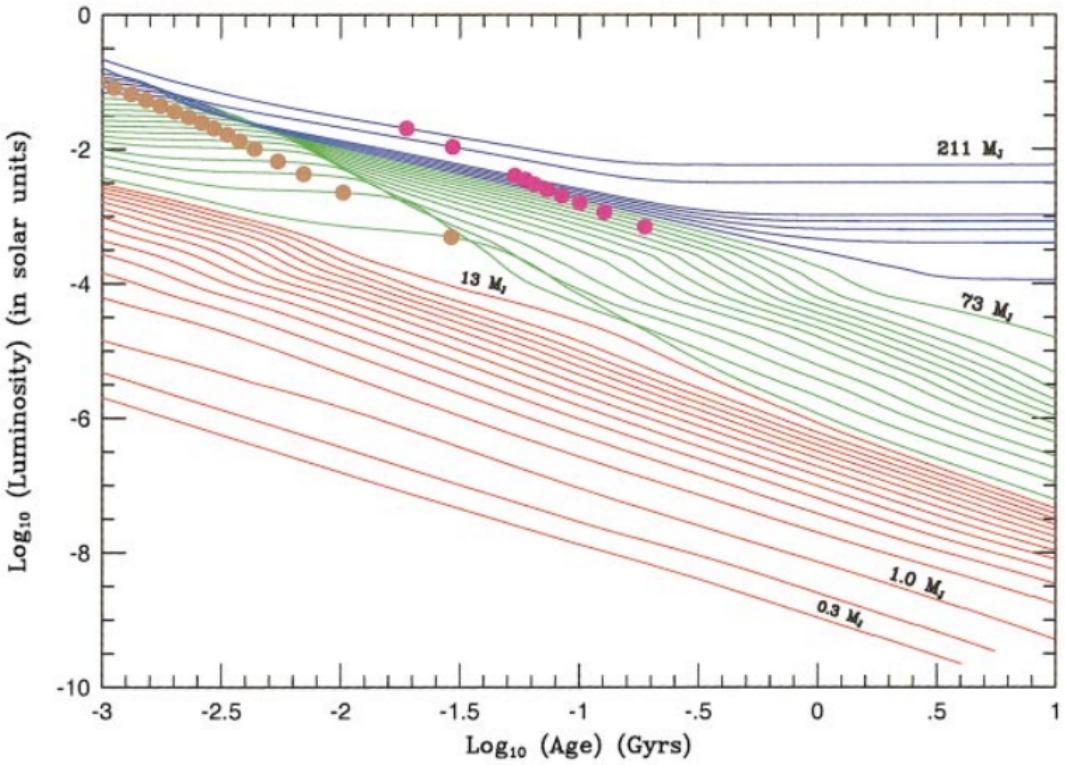

Figure 1.2 The evolution of luminosity of main sequence stars (blue), brown dwarfs (green), and planets (red). Main sequence stars have a constant luminosity because hydrogen fusion provides a stable source of energy, whereas brown dwarfs and planets lack an internal energy source and continue to cool over time. This figure has been modified from (Burrows et al. 2001).

1.3 Importance of Mass and Binary Systems

Mass is the most important parameter to determine how a stellar object will evolve, and for brown dwarfs it is the most difficult to measure. Since brown dwarfs lack a clear relationship between their mass and luminosity, we cannot determine their mass from an observation. The easiest way to determine a brown dwarf's mass is from a binary system. The relationship between the masses, period, and semi-major axis of a binary system provides a way to calculate the masses of the two objects in the binary system. Other parameters like age, radius, and temperature can be determined from a binary system where one star is a main sequence star (Konopacky 2013). Information can also be found from binary brown dwarf systems, where each object is a brown dwarf. These objects must have the same age and metallicity, and the difference in luminosity and temperature should be correlated. So we can also use binary brown dwarf systems to better improve the evolutionary models."

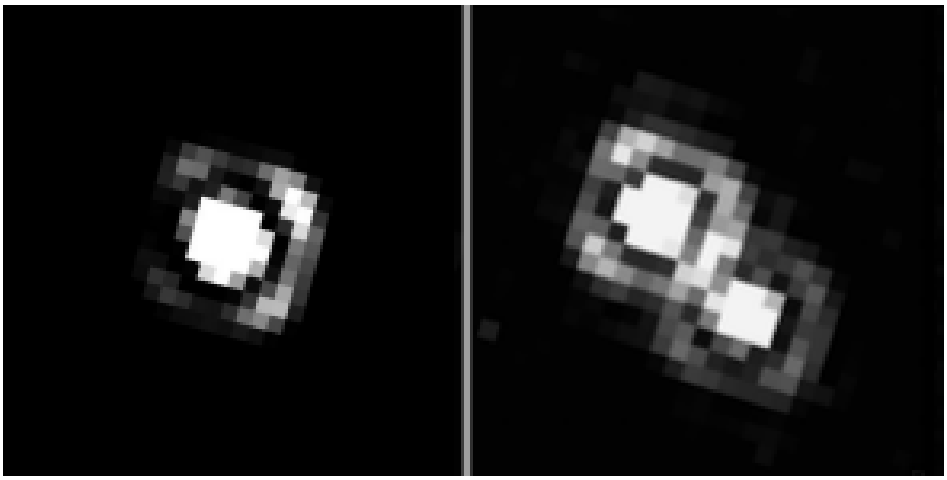

Figure 1.3 A set of images of the binary brown dwarf system Kelu-1 taken by the hubble telescopes at different dates. The left is an unresolved image that looks like a singular object while the right is a resolved image that clearly depicts it as a binary system. This figure is extracted from (NASA et al. 2009).

Brown dwarfs are very dim objects and thus are hard to observe. To discover binary systems

by eye, the two objects must have a large enough angular separation to appear individually on an image. An example of this is the Kelu-1 binary brown dwarf system as seen in Figure 1.3. The system was first detected in 1998 as a singular system, but in 2005 it was imaged again and revealed its companion brown dwarf. This was due to the elliptical nature of its orbit; in 1998 the two objects were too close together and could not be resolved, but in 2005 their separation was large enough to appear individually. Unresolved images of binary systems are easier to obtain, and by modeling an unresolved system we can discover binary systems without the need for a resolved image.

1.4 Point Spread Functions

Any stellar object, including a brown dwarf, appears to the naked eye as a point source of light when observed from earth. Through a powerful space-based telescope like the Hubble Space Telescope (HST), the light is diffracted through the lens in a predictable pattern known as a Point Spread Function (PSF). Each object will create a PSF when observed through a telescope, and in binary systems their PSF's will overlap if their angular separation is small enough (Figure 1.4). The minimum angular separation required for two object's PSF's to be resolved is known as the Rayleigh criterion. The Rayleigh criterion is calculated by $\frac{1.22\lambda}{D}$, where λ is the wavelength of the light emitted from the object, and D is the diameter of the telescopes primary mirror.

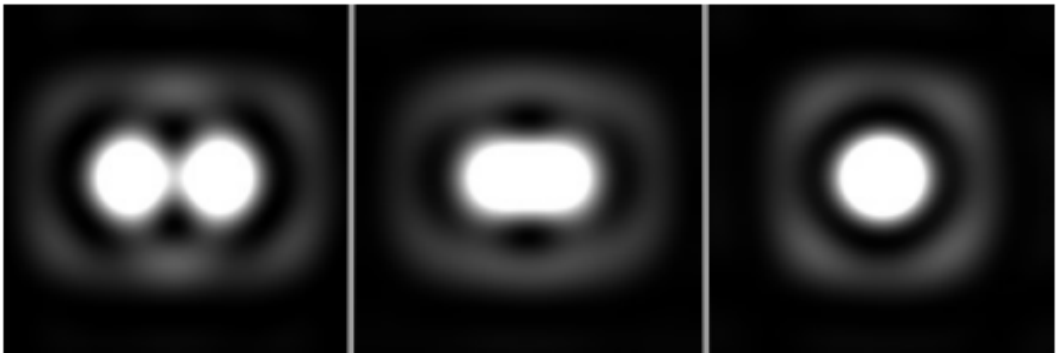

Figure 1.4 Varying levels of overlapping PSF's, from resolved on the left to unresolved on the right. This figure is extractd from (Matt 2017).

The HST has imaged many brown dwarf systems over its lifetime, and since we know all the properties of its various cameras and filters, we are able to create models of a brown dwarf's PSF observed through the HST's cameras. Tiny Tim is a program developed for creating these PSF models for the HST and is able to create models for different cameras, filters, and objects as seen in Figure 1.5. By creating models of individual and paired/overlapped PSF's we can compare them to actual brown dwarf images from the HST achive.

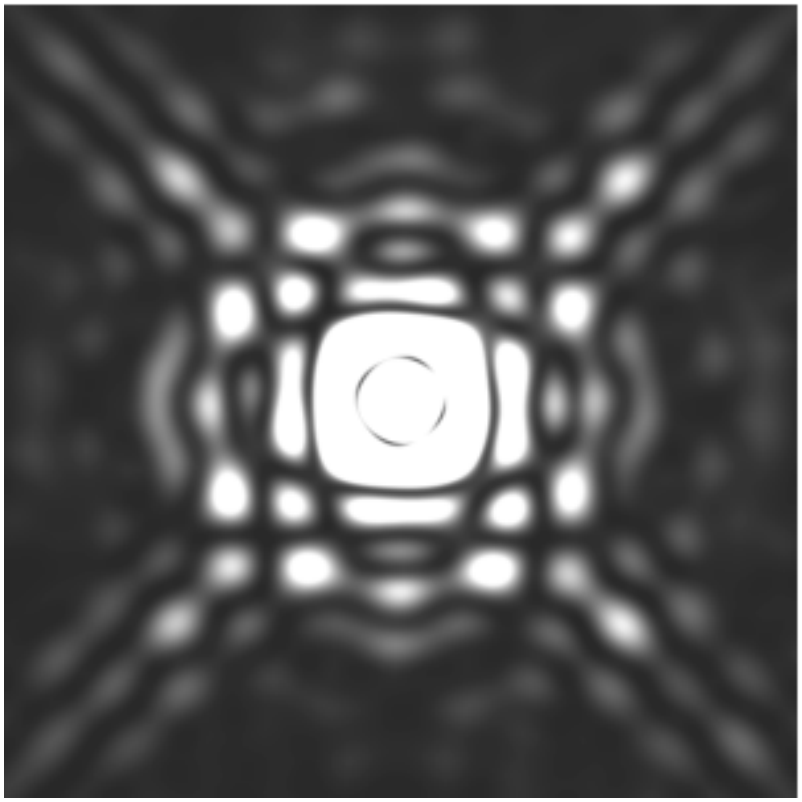

Figure 1.5 A model PSF image generated by Tiny Tim.

1.5 Overview

We lack the data necessary to better constrain and improve current brown dwarf evolutionary models. Discovering binary brown dwarf systems and obtaining parameters like mass, luminosity, and

temperature we can improve the evolutionary models. The purpose of this paper is to detail the PSF fitting process to identify potential binary brown dwarfs and detail the updates and improvements made on the model fitting process. Chapter 2 will review the development of the code including the changes and improvements made over the previous iterations of the program (Salway 2015), (Matt 2017), (Gardner 2015). Chapter 3 will detail the PSF fitting procedure from preparing the data to running the model fitting code. Chapter 4 will discuss the results and future work.

Chapter 2

Code Development

2.1 Previous Iterations of the Program

The original PSF fitting program was created to detect binary trans-Neptunian objects (Stephens & Noll 2006), which was subsequently modified to detect binary brown dwarf systems using data from the HST. The program was effective but time consuming as it required a lot of data preparation and user input. The next iteration was written in Python with the goal of reducing the user interaction of the program (Matt 2017). This code was written in Python 3 which allowed the code to directly process the images. The end-product was a modular program that required little preparation and user interaction and could be easily modified to work with various instruments and cameras. However, even though the program was easy to use, it was very slow. The FORTRAN program only took approximately 10 seconds to run after the initial preparation phase while the python code could take anywhere from 7 to 15 minutes. This is a significantly longer run time and made it extremely time consuming to run the code on large sets of data. In addition to the long run-time, the Python code was discovered to have a few errors that were leading to inaccurate fits. Most of my research was spent on correcting these errors, improving the run-time, and adding a Monte-Carlo simulation.

2.2 Code Explanation

The new program consists of one main script written in Python 3 that sequentially calls 4 separate C++ programs that each execute a major element of the model fitting process. The Python section begins by reading in the HST image and its corresponding Tiny Tim model that we created. The image and model are converted into a 2-D array of pixel values that are saved to separate files. The script calls the 'PSF_prep' C++ file that reads in the Tiny Tim model which is 10 times longer and wider than the actual image. It takes the oversized model and shifts it by one pixel before binning it back down to the image size. The position of the center pixel is varied at .1-pixel increments compared to the original image and subsequently 100 model PSF's are created with different center pixel locations.

Once the binning section is complete the script calls the 'single_fitting' C++ file that calculates the best single fit. The single fit section first iterates through each of the 100 models and subtracts them from the HST image and calculates which PSF model produces the lowest χ^2 value. That PSF model is then used to calculate a flux value that minimizes the corresponding χ^2 value. The single fit section ends by calculating the residual error from the resulting fit and saving the results into separate files. These results are read in by the Python script and written to the output file.

The Python script then calls the 'binary_fitting' C++ file that handles the binary fit. For the binary fit we make the assumption that the center pixel is the location of the primary, and the 8 surrounding pixels are the possible locations for the secondary. The program takes a second model image and centers it on the center pixel as well as the 8 surrounding pixels to represent the secondary. This is done to each of the 100 PSF models and like with the single fit we then take each of the resulting binary models and subtract it from the HST image to determine the lowest χ^2 value, then we calculate the primary and secondary flux that minimizes that χ^2 value. The residual error is calculated, and the results are saved and then are written into the output file by the Python script.

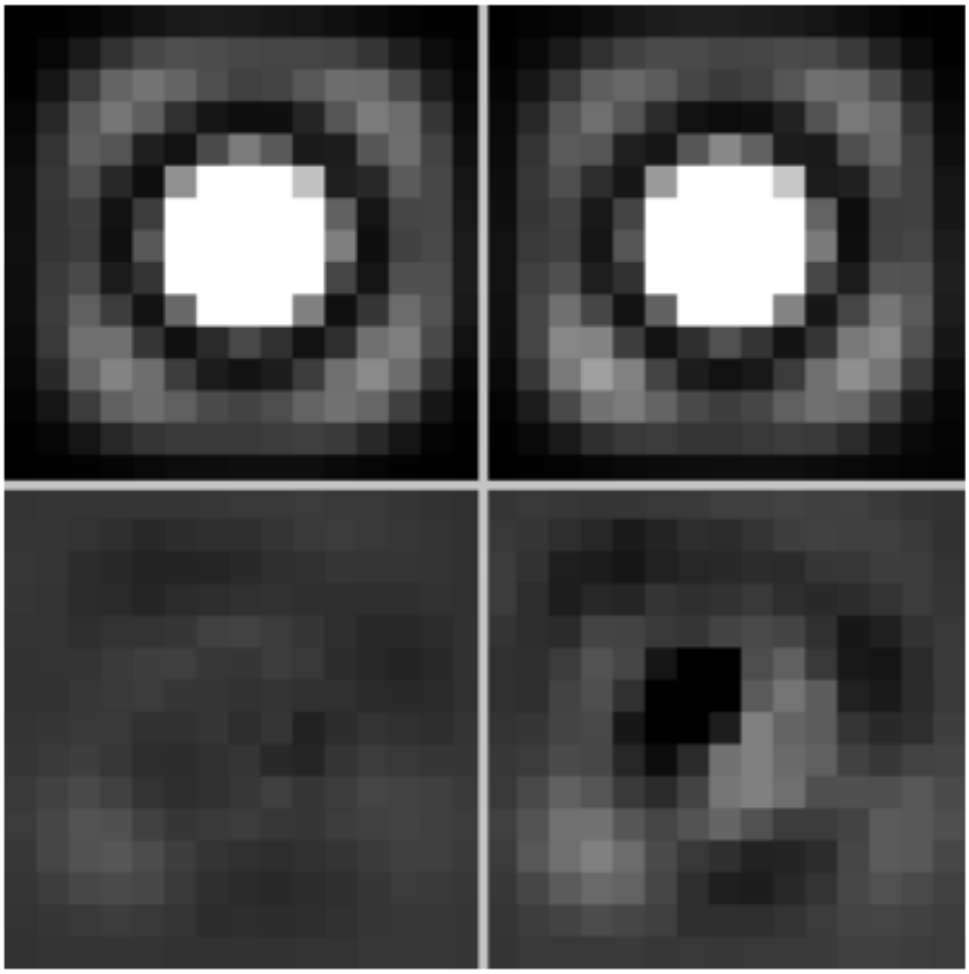

Figure 2.1 The residuals from both a single fit and the binary fit. The top left is the HST image, the top right is the best binary model PSF selected by the program, the bottom left is the residuals for the binary fit, and the bottom right is the residuals for the single fit.

With the single and binary fits complete, the next task is to calculate the position angle of the binary system and the angular separation of the two objects. More accurate center coordinates of each of the best PSF models is determined using the centroid function from the Photutils package and is used to calculate the position angle and separation. The Python script then converts the flux values that each fitting section calculated into magnitudes.

The Python script continues by taking the HST image and generating 1000 copies of it with

random noise added for conducting a Monte Carlo simulation on the fits. The script then calls the 'rn_single' and 'rn_binary' C++ scripts that run the same single and binary fits as the 'single_fitting' and 'binary_fitting' scripts but applies them to each of the 1000 HST images with random noise added. The Python script then takes the results for each image and plots the distribution of single fluxes, primary and secondary fluxes, position angles, angular separations, and errors for both the single and binary fits.

2.3 Changes From the Previous Program

While the program in Matt (2017) was effective, there were a few errors in it that resulted in inaccurate fits. Before running the fits, the code would prompt the user to input the center coordinates and would take the 3x3 square from the HST image based on that center pixel. The issue was that it did not take into account that the first element in a Python array is the 0'th element, so the image was always shifted to the right by one pixel. This resulted in the primary model being placed on a fainter neighboring pixel and missing the true center. This was an easy fix and only required changing the way it indexed the arrays.

In the actual fitting process, the program in Matt (2017), which we will call the "Python code", used a different method to calculate the errors in the single fit versus the binary fit. This made it difficult to compare the single fit to the binary fit and determine which was better. By making the error calculation the same across both fitting process we could effectively compare the two fits and determine whether the single or the binary fit was best.

In addition to the error calculations, the Python code also used different weighting methods between the single and binary fits. The weighting system helps to determine which pixels are the most important and is only used to position the model PSF's. HST's brown dwarf images are under-sampled, so most of the flux is found within one or two individual pixels. The primary object

is typically located within these pixels with higher flux; however, the secondary can be located in the neighboring pixels with relatively low flux counts. In the Python code, the single fitting process used a weight system where the center pixel and the 8 surrounding pixels were multiplied by a value of 1, and the rest of the pixels were set to 0. This made it so only the 3x3 box of pixels around the center pixel were candidates for the brown dwarf, which is acceptable for the single fit. The binary fit however, used the square root of each pixel's flux as the weight which gives the pixels with lower flux values more importance than without the weights. I wanted to keep the weighting system consistent across the two fits to get the most accurate and comparable results between the single and the binary fits. In the new code, both the single and the binary fit use the square root of the total flux of each pixel as the weight.

After the errors were corrected and the single and binary fitting methods were made more consistent, there was still the issue of speed. While Python is a very powerful and adaptable programming language, it is very slow compared to pre-compiled languages like C++ or FORTRAN. As such, the Python code took on average about 36 times longer to run compared to the original FORTRAN code. To remedy this, I re-wrote the computational sections of the python code in C++ in an effort to improve the runtime. The computational sections of the Python code uses NumPy, a scientific computing library for python built for working with arrays. NumPy's built in functions are many times faster than writing the same functions in base Python code, however, it is still significantly slower than C++. Translating from NumPy to C++ is not an easy task as C++ does not have the built-in functions that NumPy has. Instead of attempting to hard-code the NumPy functions into C++, I used a 3rd party package called Xtensor.

Xtensor is a C++ library designed for numerical analysis with multi-dimensional array expressions. Xtensor was designed to be the C++ version of NumPy; It follows the NumPy Application Programming Interface (API) very closely but utilizes the speed of pure C++. This allowed me to almost directly translate the NumPy code into C++ code. The new code utilizes Python's strength

in working directly with images and C++'s computational speed. The xtensor documentation can be found at:

https://xtensor.readthedocs.io/en/latest/index.html

Chapter 3

Method

3.1 The Hubble Space Telescope

The HST is equipped with a wide variety of cameras, however, the program is currently only able to run for the HST's Near Infrared Camera ad Multi-Object Spectrometer (NICMOS). Future work could include expanding the program to work with the Advanced Camera for Surveys (ACS) and Wide Field Camera 3 (WFC3). NICMOS processed images in the near infrared through its three different cameras, each with its own plate scale and filters. The program currently only works with the NICMOS 1 and NICMOS 2 camera, each with a plate scale of .043 and .075 arcseconds/pixel respectively. Figure 3.1 displays an image of the brown dwarf 2MASSJ203603+10 taken by the HST NICMOS 1 camera.

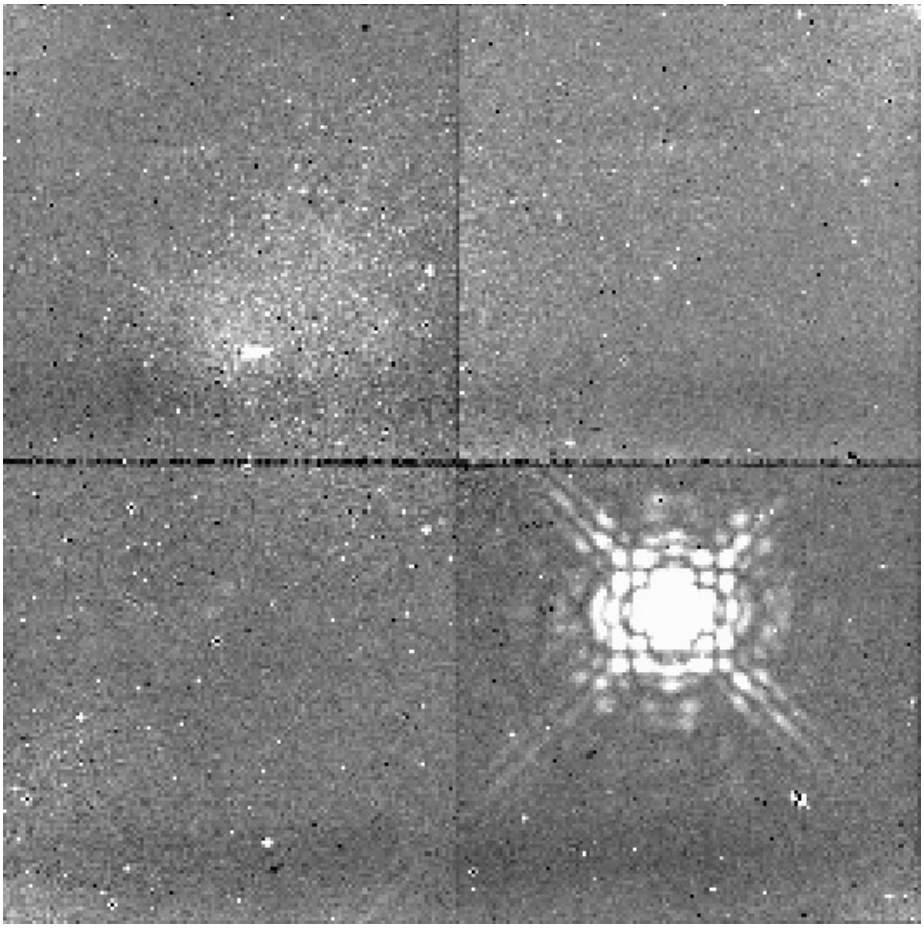

Figure 3.1 HST image of 2MASSJ203603+10 using the NICMOS 1 camera in the F170M filter.

3.2 Preparing the Data

Brown dwarf images are available for download through the HST archive which has NICMOS 1 and NICMOS 2 images from 1997-2008. The next step is to create the model PSF's using the Tiny Tim software. Tiny Tim allows us to create model PSF's for both NICMOS 1 and NICMOS 2 images in various filters and configurations. The Tiny Tim software and user manual can be found at:

https://www.stsci.edu/hst/instrumentation/focus-and-pointing/focus/tiny-tim-hst-psf-modeling

To create the PSF models the camera, instrument, filter, object coordinates, and spectral type of

the brown dwarf are needed. The following is a modified version of the step-by-step instructions to running Tiny Tim found in Matt (2017).

> tiny1 F170M.in Where F170M is the name of the file where Tiny Tim will save the parameters to. We use the filter name which for this example is F170M.

Tiny Tim will then prompt the user for the instrument and camera. Be aware that for the NICMOS1 camera there are two options, one for before 2002 and one for after 2002 which is when the new cryocooler was installed.

Next the user is asked to enter the x- and y- coordinates of the center of the object rounded to the nearest integer and each coordinate separated by a space.

: XXX YYY

For the NICMOS camera you will be asked if the image was taken after 2002. If true, enter Y, otherwise enter N.

: Y

The user is then prompted for the filter name in lower caps. In this example we are using F170M as the filter.

: f170m

Tiny Tim then gives a few options to build the spectrum, in this case we use an ASCII table of the brown dwarf's spectrum, which is option 5.

: 5

The description of the ASCII tables are given in Appendix C of the Tiny Tim manual (Krist & Hook 2004). In this example our spectrum is the file L4.txt

: L4.txt

The user is asked whether they want to subsample the PSF. For our model fitting program, we subsampled the PSF by 10 to get our .1 pixel resolutions as described in section 2.1.

: Y

: 10

Tiny Tim prompts for the secondary mirror despace which is 0 for our purposes

: 0

The user is then prompted for the filename of the output filename of the PSF model. Again, we use the filter name which in this case is F170M.

: F170M

Tiny Tim then saves this information to the parameter file which was named in the beginning. To generate the PSF model for the NICMOS camera we type in the terminal:

> tiny2 F170M.in

Tiny Tim then generates a PSF model with the name we entered with "00" tacked on at the end. For our example this would be F170M00.fits.

The next step is to process the images and prepare them for the fitting process. NICMOS images are not linear and thus require a correction in order to accurately calculate magnitudes. This correction is done in PyRAF which is a Python-based version of IRAF (Image Reduction and Analysis Facility). IRAF, and subsequently PyRAF, is a software developed by the National Optical Astronomy Observatory (NOAO) to reduce and analyze astronomical data. Using the pedsky and rnlincor packages in PyRAF, we removed the estimated sky background and residual bias, and corrected the non-linearity in the image respectively. To get these packages we type in the commands:

> stsdas

> hst_calib

> nicmos

Now we can run the commands to apply pedsky and rnlincor to our images. For this example, our HST image file name is n8qt03mhq_cal.fits. To apply the correction to the images we run the pedsky function on the cal.fits image file followed by the output file name which ends with ped.fits

to indicate that pedsky has been run on it. Then we run the rnlincor command on the ped.fits file and output it to a rnl.fits file which is the file we will use for the actual fitting process.

> pedsky n8qt03mhq_cal.fits n8qt03mhq_ped.fits

> rnlincor n8qt03mhq_ped.fits n8qt03mhq_rnl.fits

3.3 Fitting the PSF Models

Now that the HST images are corrected for non-linearity the data can be run through the model fitting code. The _rnl.fits image file needs to be placed with its corresponding Tiny Tim generated PSF model in a single directory along with the PSFfitting.py, PSF_prep.cpp, single_fitting.cpp, binary_fitting.cpp, rn_single.cpp, rn_binary.cpp, and cameras.json files. Before the code can be run we need to make sure that the proper packages are installed. The code requires the user to have both Python 3 and a C++ compiler that supports C++14 or higher. For Python the numpy, astropy, and photutils packages are required. For C++ the external, third-party xtensor and xsimd packages needs to be installed. Instructions for installing xtensor and xsimd can be found at:

https://xtensor.readthedocs.io/en/latest/installation.html

https://xsimd.readthedocs.io/en/latest/installation.html

As discussed in section 2.3, xtensor is a C++ library designed for numerical analysis with multi-dimensional array expressions. xsimd is essentially a wrapper that can be used for xtensor to make reduction calculations faster (such as “sum” functions). To compile the .cpp files the paths to xtensor, xtl (comes installed with xtensor), and xsimd must be included in the compile command. In addition, -mavx2, -ffast-math, -DXTENSOR_USE_XSIMD -03, and -openmp must also be added to the compile line. For my machine my compile command for the PSF_prep.cpp file would be:

> g++ -mavx2 -ffast-math -DXTENSOR_USE_XSIMD -O3 -openmp -I

/home/loicbeus/anaconda3/pkgs/xtensor-0.21.10-h0efe328_0/include -I

/home/loicbeus/anaconda3/pkgs/xtl-0.6.21-h0efe328_1/include -I

/home/loicbeus/anaconda3/pkgs/xsimd-7.4.9-h0efe328_1/include/ PSF_prep.cpp -o PSF_prep

Once all five of the .cpp files are compiled the PSF fitting code can be called by running the following command in the terminal:

> Python3 PSFfitting.py F170M00.fits n8qt03mhq_rnl.fits

The results of the fit can be found in the file imagename_output.txt in the same directory, where "imagename" is the name of your image file. In my example the output file would be n8qt03mhq_output.txt.

Chapter 4

Results and Conclusions

4.1 Analysis

The model fitting program was run on three brown dwarf systems that we suspect may be binaries: 2MASSJ203603+10, 2MASSJ225518-57, and 2MASSJ055919-14. The results are summarized in the tables below.

Main Fit

Table 1: 2MASSJ203603+10 Results

Date (UT)	Camera	Filter	Separation (arcsecs)	Position Angle (degrees)	Primary Magnitude	Secondary Magnitude	Single Residual Error	Binary Residual Error
6/23/2006	NIC1	F170M	0.0347	31.08	13.455	14.123	0.72753	0.27175
6/23/2006	NIC1	F110W	0.0346	31.09	14.848	16.325	2.4136	0.98959
6/24/2006	NIC1	F170M	0.083	22.59	13.058	15.987	0.93903	0.17432
6/25/2006	NIC1	F110W	0.0551	40.01	14.747	16.81	3.207339	1.44667

Table 2: 2MASSJ225518-57 Results

Date (UT)	Camera	Filter	Separation (arcsecs)	Position Angle (degrees)	Primary Magnitude	Secondary Magnitude	Single Residual Error	Binary Residual Error
7/4/2006	NIC1	F170M	0.0491	8.17	13.692	14.689	0.814015	0.3411
7/4/2006	NIC1	F110W	0.0182	15.41	15.058	16.872	3.16008	0.83646
7/4/2006	NIC1	F170M	0.0387	60.3	13.441	15.755	0.4077916	0.37696
7/4/2006	NIC1	F110W	0.0155	93.8	15.405	15.87	1.41848	0.63798

Table 3: 2MASSJ055919-14 Results

Date (UT)	Camera	Filter	Separation (arcsecs)	Position Angle (degrees)	Primary Magnitude	Secondary Magnitude	Single Residual Error	Binary Residual Error
10/20/2004	NIC1	F090M	0.0345	145.26	16.909	18.099	0.239839	0.09727
10/20/2004	NIC1	F110M	0.0261	143.09	15.122	15.715	1.057003	0.3747
10/20/2004	NIC1	F145M	0.0347	92.44	14.908	16.877	0.4378065	0.18483
10/20/2004	NIC1	F165M	0.0178	138.66	14.284	14.331	0.6876878	0.29734

Monte-Carlo Simulation

Table 4: 2MASSJ203603+10 Monte-Carlo Simulation Results

Date (UT)	Camera	Filter	Average Single Error	Average Binary Error	Average Separation (arcsec)	Average Position Angle (degrees)
6/23/2006	NIC1	F170M	0.901785	0.450896	0.034988	32.63879
6/23/2006	NIC1	F110W	2.45605	0.9456653	0.0371411	29.202388
6/24/2006	NIC1	F170M	0.973158048	0.423748829	0.073697404	23.80182029
6/25/2006	NIC1	F110W	3.256868646	1.507485348	0.054330994	42.4401892

Table 5: 2MASSJ225518-57 Monte-Carlo Simulation Results

Date (UT)	Camera	Filter	Average Single Error	Average Binary Error	Average Separation (arcsec)	Average Position Angle (degrees)
7/4/2006	NIC1	F170M	0.956075323	0.499246344	0.047639686	14.86218946
7/4/2006	NIC1	F110W	2.985244776	0.890004873	0.021579653	18.02078268
7/4/2006	NIC1	F170M	0.587519494	0.551784161	0.040243877	60.29769566
7/4/2006	NIC1	F110W	1.480292665	0.856459013	0.011798756	96.49048453

Table 6: 2MASSJ055919-14 Monte-Carlo Simulation Results

Date (UT)	Camera	Filter	Average Single Error	Average Binary Error	Average Separation (arcsec)	Average Position Angle (degrees)
10/20/2004	NIC1	F090M	0.436617757	0.302204006	0.032366026	111.5940711
10/20/2004	NIC1	F110M	1.115258666	0.54590363	0.024568388	143.7195852
10/20/2004	NIC1	F145M	0.469686574	0.412522388	0.03685032	88.19421434
10/20/2004	NIC1	F165M	0.726747454	0.450887493	0.018281657	109.8475804

Looking at 2MASSJ203603+10, the results are interesting. The first two images taken on June 23rd settled on almost exactly the same position angles and separations. However, the next two images taken on June 24th and 25th have very different results. The separations in these two are both larger than the first two and differ by about .03 arcsecs. The position angles in the latter two also differ from the first two images (see table 1). The Monte-Carlo simulation where we generated 100 copies of the HST image each with random noise confirm the results.The average values of the single error, binary error, separation, and position angle were all very close to the actual values from the main fit. While the results are inconclusive, this brown dwarf is promising, and we will continue to monitor it.

2MASSJ225518-57 is a little less promising, the separation and position angle values vary significantly from image to image (see table 2). This could be due the secondary being particularly faint in one of the filters or their separation is so small that it can't be accurately calculated. The Monte-Carlo Simulation for this data again has similar values to the main fit (see table 5). This brown dwarf is a less promising candidate, but we can't rule it out just yet.

2MASSJ055919-14 has has been closely examined by previous students (Salway 2015), (Gardner 2015). Their results on this brown dwarf showed that the separation and position angle were accurate to a probability of over 90 percent, with the Filters F145M and F165M being the least accurate. They also found that the separation values were fairly consistent across images, but the position angles varied a lot more. Our results mostly agree with that conclusion. Our separations are a little more varied but are still pretty close to each other, and for three out of the four images the position angle is almost exactly the same with one outlier (See table 3). The Monte-Carlo simulation produces average values that are reminiscent of the main fit results (see table 6).

4.2 Conclusions and Future Work

It is important to note that the main purpose of this research was to improve the existing model fitting program. The new program runs faster, requires less user interaction, and includes a modifiable Monte-Carlo simulation on the data. Some test fits on various brown dwarfs show some promise, but there are still some issues to work out. One issue is the error values; the binary fit can theoretically always have a lower residual error than the single fit because the binary fit has more degrees of freedom while the single fit only has the one position and flux that it tests. Future work will need to take into account the extra degrees of freedom in the binary fit and correct for that in order to compare it to the single fit and determine which fit is truly better. The Monte-Carlo simulation is useful, but incomplete. In order to better compare the images with random noise to the main fit the Monte-Carlo simulation needs to produce results on the magnitudes as well.

As mentioned in (Matt, 2017), the program uses a χ^2 to test for the best fits which may or may not be the best method. Testing the program with other error measurements like mean square error and absolute value of residuals should be the next step.

In addition, the HST has a lot of data on brown dwarfs that is not taken with the NICMOS cameras. There is data in the ACS and WFC3 instruments and expanding the code to work with those images should also be a priority. Further optimization and functionality can also speed up the model fitting process. Before fitting the images, we still have to create the Tiny Tim models and correct the HST images for non-linearity. A lot of the user interaction can be removed by automating the process through Python which would also increase the speed that we can run through the entire model fitting process.

Bibliography

Burrows, A., Hubbard, W. B., Lunine, J. I., & Liebert, J. 2001, Rev. Mod. Phys., 73, 719

Dupuy, T., Liu, M. C., & Ireland, M. J. 2014, The Astrophysical Journal, 790, 133

Gardner, D. 2015, Bachelors Thesis (Brigham Young University)

Konopacky, Q. M. 2013, Mem. S. A. It., 84, 1005

Krist, J., & Hook, R. 2004, https://www.stsci.edu/hst/instrumentation/focus-and-pointing/focus/tiny-tim-hst-psf-modeling

Marley, M. S., & Leggett, S. K. 2009, Astrophysics in the Next Decade, 101

Matt, K. 2017, Bachelors Thesis (Brigham Young University)

NASA, ESA, & Stumpf, M. 2009, https://esahubble.org/images/opo0901a/

Salway, E. 2015, Bachelors Thesis (Brigham Young University)

Stephens, D. C., & Noll, K. 2006, The Astronomical Journal, 131, 1142