A ROTATING APERTURE MASK FOR SMALL TELESCOPES

A Thesis

presented to

the Faculty of California Polytechnic State University,

San Luis Obispo

In Partial Fulfillment

of the Requirements for the Degree

Master of Science in Mechanical Engineering

by

Edward Leo Foley

November 2019

COMMITTEE MEMBERSHIP

TITLE:	A Rotating Aperture Mask for Small Telescopes
AUTHOR:	Edward Leo Foley
DATE SUBMITTED:	November 2019
COMMITTEE CHAIR:	John Ridgely, Ph.D. Professor of Mechanical Engineering
COMMITTEE MEMBER:	Russell Genet, Ph.D. Research Scholar in Residence
COMMITTEE MEMBER:	William R. Murray, Ph.D. Professor of Mechanical Engineering
COMMITTEE MEMBER:	Jane Zhang, Ph.D. Professor of Electrical Engineering

ABSTRACT

A Rotating Aperture Mask for Small Telescopes

Edward Leo Foley

Observing the dynamic interaction between stars and their close stellar neighbors is key to establishing the stars' orbits, masses, and other properties. Our ability to visually discriminate nearby stars is limited by the power of our telescopes, posing a challenge to astronomers at small observatories that contribute to binary star surveys. Masks placed at the telescope aperture promise to augment the resolving power of telescopes of all sizes, but many of these masks must be manually and repetitively reoriented about the optical axis to achieve their full benefits. This paper introduces a design concept for a mask rotation mechanism that can be adapted to telescopes of different types and proportions, focusing on an implementation for a Celestron C11 Schmidt–Cassegrain optical tube assembly. Mask concepts were first evaluated using diffraction simulation programs, later manufactured, and finally tested on close double stars using a C11. An electronic rotation mechanism was designed, produced, and evaluated. Results show that applying a properly shaped and oriented mask to a C11 enhances contrast in images of double star systems relative to images captured with the unmasked telescope, and they show that the rotation mechanism accurately and repeatably places masks at target orientations with minimal manual effort. Detail drawings of the mask rotation mechanism and code for the software interface are included.

ACKNOWLEDGMENTS

This thesis is submitted as the product of hard work from many different contributors.

I would like to extend my sincere thanks to Dr. Russell Genet for reigniting my love of astronomy through his contagious mix of passion, curiosity, and geniality. He fully supported me throughout my odyssey to complete this project by offering vital direction and encouragement, not to mention connecting me with other astronomers and co-authoring an earlier paper published in the Journal of Double Star Observations. He graciously hosted me at the Orion Observatory and allowed me to use one of his telescopes for crucial optical and mechanical tests. I deeply appreciate the collaborative and positive research environment he fostered—not just for me and this project but also for others and their own scientific pursuits.

I am very grateful to thesis committee chair Dr. John Ridgely for the guidance he offered me and the patience he exhibited throughout this project. I also sincerely thank my committee members, Drs. William Murray and Jane Zhang, for attending my defense and sharing constructive feedback on a draft of this paper.

I am indebted to Dr. Neil Zimmerman of Princeton University for his counsel and for kindly contributing an optimized mask design for the Celestron C11 telescope. This paper and the conclusions drawn within it benefitted greatly from his efforts, just like the astronomers who might fruitfully use his bowtie mask in double star observations.

I am grateful to Jimmy Ray for running the first practical tests of the Gaussian donut mask on a telescope; and to Donald Loveland, who diligently evaluated and documented the performance of two mask designs on several pairs of double stars.

I thank Kevin Jantz for his machining assistance and feedback on early drawings. I also appreciate the efforts of Nick Bonafede, Marc Goupil, and Bodin Rojanachaichanin, who lent some of their time to review my engineering documentation.

I must recognize Keenan Reimer, who helped me get to and from campus in the home stretch of the project. I also thank Luke Williams, who delivered the Orion Observatory’s C11 to my house for an afternoon of important mechanical tests.

I extend gratitude to David Rowe, who kindly shared with me his Atmospheric Seeing Distortion simulator and adapted it to work with arbitrary aperture shapes in support of this project.

I thank the professors and administrators at Cal Poly for actively applying the University’s “Learn by Doing” mantra and for granting a mechanical engineering student the opportunity to enroll in courses within the fields of computer science, computer vision, image processing, digital control systems, and optics. The lessons from this unusual palette of subjects were fundamental in my acquiring an inside-out understanding of a unique project.

I give thanks to the many housemates, coworkers, grad lab occupants, and friends who cheered me on through this endeavor.

Finally, I thank my parents, Bill and Lori; my brother, Matt; and all my relatives for their love and encouragement. Hopefully their reading these sentences answers their long-standing and oft-repeated question of whether my thesis is done yet.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1. INTRODUCTION

1.1 Statement of problem

Stars that share a mutual gravitational bond provide valuable information to astronomers. Observing the evolution of a binary system's separation and position angle over a sufficiently long period of time allows us to chart orbits and establish the dynamical mass of the system using Kepler's Third Law (Bennett, Donahue, Schneider, & Voit, 2014). The dynamical mass can be combined with other information to deduce the masses of the system's individual components, which in turn help us place the star within an evolutionary timeline (Hillenbrand & White, 2004).

Approximately half of all stars are believed to reside in binary systems (Bennett, Donahue, Schneider, & Voit, 2014), but some of these stars are easy to miss because they are dim relative to their neighbor, close to their neighbor or both. Frequently, the light from the brighter star will overwhelm the light from the dimmer star, making the dim star difficult to isolate (Daley, 2007; Hecht, 2002, p. 514). A similar issue arises when stars of similar magnitude lie close together, in which case their light becomes muddled together by atmospheric interference or they fall beneath the resolution limit of the telescope observing them, rendering the stars effectively indistinguishable.

Many of the overlooked stars are late-M stars, which are cool, faint, and red relative to other main-sequence members (Figure 1). Type-M dwarfs are believed to represent about three-quarters of all main-sequence stars (Yang, Cowan, & Abbot, 2013; van Dokkum & Conroy, 2010), but their low luminosity compared to other main-sequence stars leads them to account for a disproportionately small number of observations (Malmquist, 1925). These stars are typically viewed with charge-coupled device (CCD) cameras, which tend to have low sensitivity to wavelengths in the infrared range where late-M stars emit much of their energy, further complicating their study (Figure 2). Improving our ability to detect these dim binary components will allow us to enhance the accuracy of our binary star records and strengthen the models that depend on them.

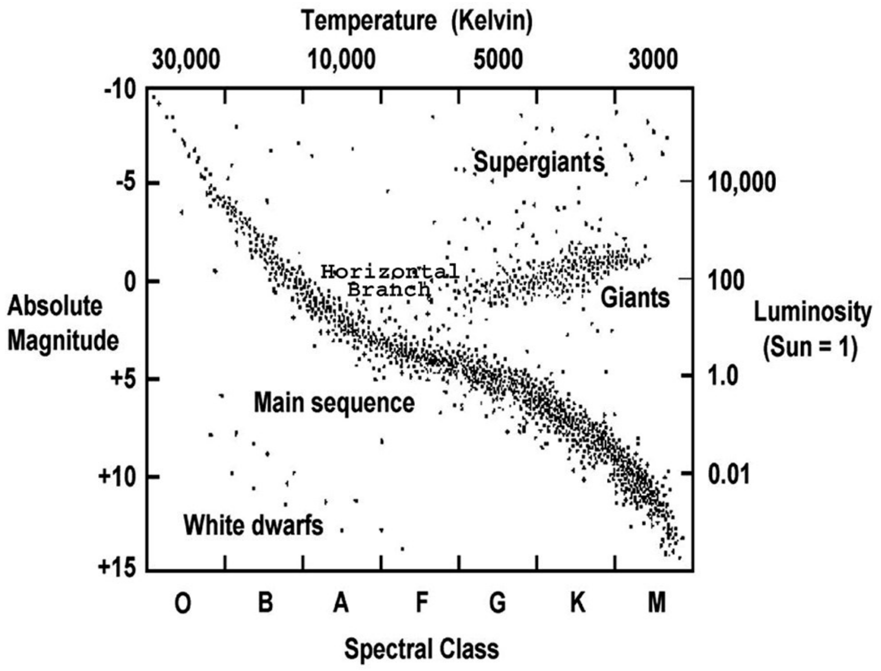
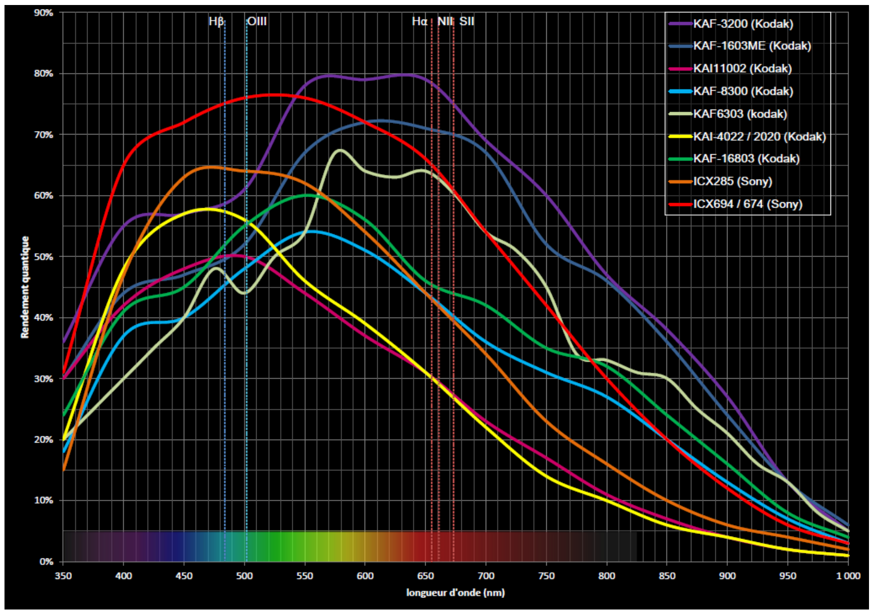


Figure 1. Hertzsprung–Russell diagram (NASA/CXC/SAO, 2015). Luminosity increases along the vertical axis and temperature decreases along the horizontal axis. Type-M stars appear toward the right side of the diagram.



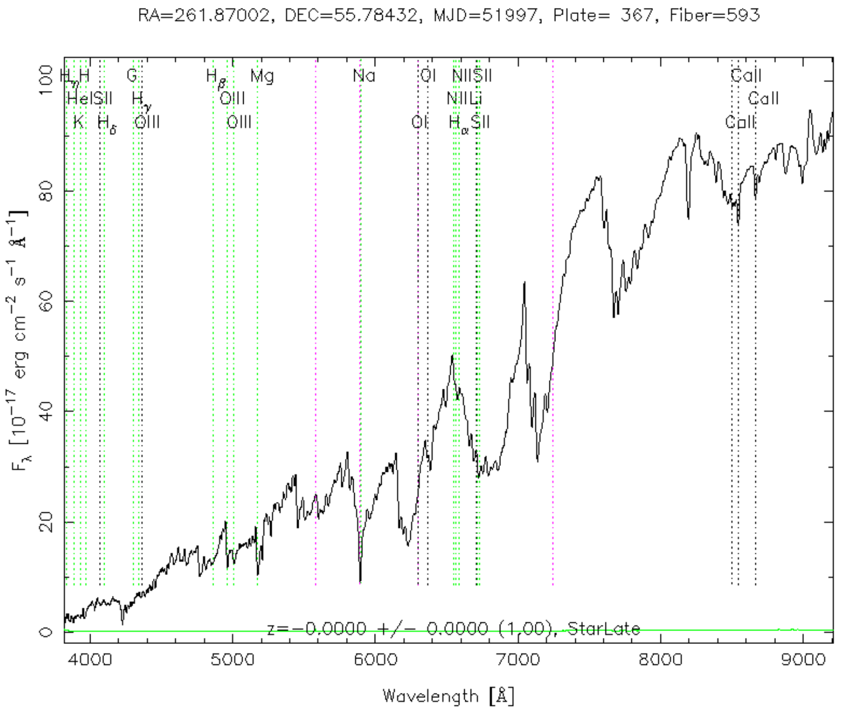


Figure 2. Top, a comparison of the quantum efficiency of common CCD models across a range of wavelengths (Bernhard, 2012). Bottom, spectral cross-correlation template for an M3 star (Spectral cross-correlation templates - SDSS DR7, 2005). Note that the two graphs cover similar domains despite using different wavelength units.

One method of improving resolution is to use a larger telescope: telescopes with larger apertures have greater light-collecting potential and produce concentrated diffraction patterns less prone to obscuring dim neighbors of bright stars (Figure 3). Naturally, because larger telescopes can be very expensive, they are not an appropriate option for all observatories. Instead, we would prefer to augment the abilities of existing telescopes to resolve close binary stars. If these smaller telescopes were better equipped to view binary stars, more observatories would be able to contribute to the study of known systems and the discovery of entirely new systems.

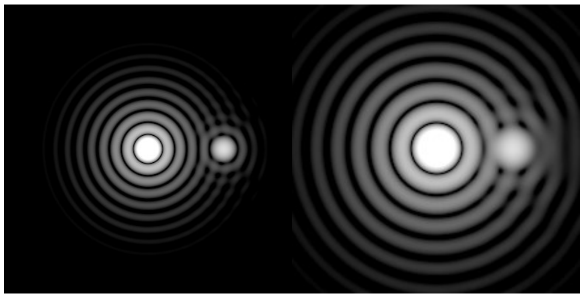


Figure 3. Brightness-normalized simulation of identical high-contrast star pairs viewed through circular apertures of two different sizes, assuming perfect seeing and monochromatic light. (Brightness is plotted on a nonlinear scale.) The telescope with the larger aperture can more easily resolve the dim secondary star.

The technology that resolves binary stars must have minimal impact on the astronomer and the telescope to have the highest adoption: it must be affordable and easy to use; its construction must be lightweight and cause no harm to the delicate and expensive optical components of the telescope. Our ideal solution is one that can be reproduced at home, using no exotic tools or processes; and is scalable to optical tube assemblies of different proportions. Working within these mechanical and usability considerations, our next task is to specify our desired resolution and contrast.

1.2 Resolution and contrast targets

When a single star is viewed through a telescope with a circular aperture, an image capturing light at a single wavelength shows not a pinpoint of light but rather a pattern of concentric rings encircling a central concentration of light. The bright rings represent constructive interference resulting from diffraction caused by the aperture's contour—regions where electromagnetic waves are reinforced by other waves at a similar phase. The dark rings represent destructive interference, representing the interaction of light out of phase.

The bright glow in the center of the pattern, which contains about 84 percent of the total power (Reidl, 2001), is known as the Airy disk (Figure 4). The size of this Airy disk forms the basis of the Rayleigh criterion, which is a common method of quantifying resolution in optical systems. Applied to astronomy, the Rayleigh criterion states that two stars are considered resolvable if their angular separation is greater than the radius of the Airy disk. Mathematically, this can be shown to be true when

$$D \sin \theta > 1.22\lambda/D \tag{1}$$

where D is the diameter of the aperture, θ is the angle of separation, and λ is the wavelength of light (Swinburne University of Technology, n.d.). Recognizing that the angle will be extremely small, we can apply a small-angle approximation to arrive at

$$\theta > 1.22\lambda/D \tag{2}$$

For a telescope with an 11-inch-diameter circular aperture observing 550-nm-wavelength light, this limit is 0.495 arcseconds (abbreviated *as*). Note that larger diameters and smaller wavelengths decrease the limit, improving the resolving power. Figure 3 demonstrates how the diffraction pattern changes when using two diameter values.

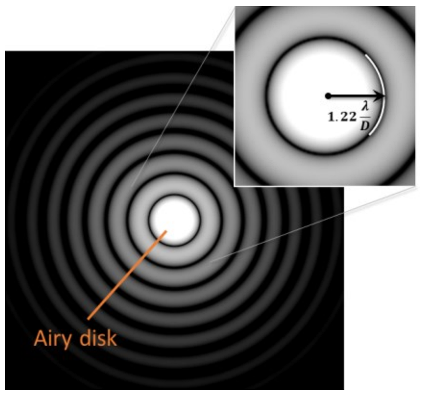


Figure 4. The Airy pattern and the Airy disk. The radius of the Airy disk is 1.22 wavelengths per diameter (λ/D).

In practice, the true resolving power of telescopes is worse than the Rayleigh limit implies due to atmospheric turbulence, imperfect focus, thermal effects, and miscellaneous optical aberrations in the telescope. Still, the criterion provides a simple, consistent, and convenient means of describing an important parameter.

Another important optical parameter is the contrast, which is especially relevant to the study of double stars with a large brightness difference. We use *contrast* in two different ways to refer to a ratio of electromagnetic powers.

When referring to diffraction patterns, contrast is the ratio of the power concentration at a point of interest relative to the pattern's maximum power concentration. Because these ratios can cover a large dynamic range, we will refer to the contrast by taking the base-10 logarithm of this ratio. Figure 5 shows the log-10 contrast along the radius of the Airy pattern. Note that darker regions correspond to log-10 values that are lower (more negative).

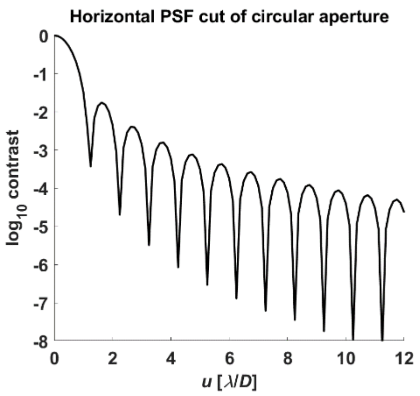


Figure 5. Contrast along the radius of the Airy pattern. Diffraction peaks of decreasing brightness alternate with diffraction nulls.

When referring to double stars, contrast refers to the brightness ratio between the components. For historical reasons, this is specified using the difference in their apparent visual magnitude, which, like our diffraction pattern metric, is logarithmic, but which uses a base of $\sqrt[5]{100}$ instead. Perhaps counterintuitively, higher apparent visual magnitudes describe dimmer stars, leading to a negative sign in the proportionality constant converting our diffraction-based contrast metric to the stellar contrast metric:

$$\Delta m_{\text{apparent}} = -2.5 \Delta m_{\log_{10}} \tag{3}$$

It is difficult to define a theoretical maximum contrast that can be measured because the result is dependent on a great number of variables including the magnification of the telescope, atmospheric conditions, camera sensitivity, and, if the telescope is aimed manually toward a dim subject, the astronomer's visual acuity and supply of patience.

According to Dr. Russell Genet, double-star studies of pairs with contrast ratios between 6 and 7 apparent visual magnitudes (log-10 values between −2.4 and −2.8) can be difficult to capture using unmodified telescopes, so we establish the more challenging end of this range, 7 apparent visual magnitudes or a log-10 contrast of −2.8, to be our contrast target. We will strive for a solution that achieves this contrast at the smallest possible inner working angle, since this is where diffraction fringes are brightest and thus where the greatest benefits theoretically lie. It is also important for this high contrast to be maintained outward to larger working angles so that we provide good contrast when studying double star systems with a range of separation values.

1.3 Existing approaches

Foley et al. (2015) summarize several ways in which an existing telescope can be modified to enhance resolution in double star observations. Some techniques include color filters, occulting bars, Lyot coronagraphs, and apodizing masks.

1.3.1 Color filters

Color filters are optical components that selectively admit or reject light based on the light's wavelength. The common Johnson/Cousins/Bessell filter set shown in Figure 6 contains five filters that admit light in bands from the infrared to the ultraviolet, but many other filters are possible. Where nearby stars have significantly different temperatures and thus different spectral profiles, color filters emphasize the light of one star by blocking some light from others (Figure 7).

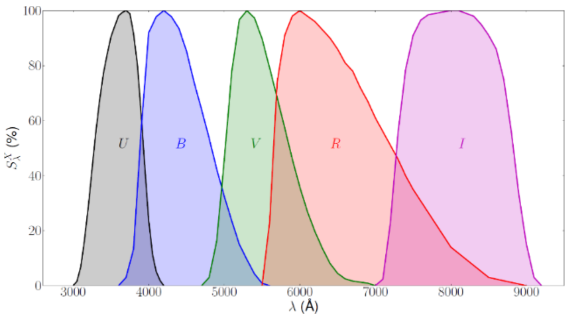


Figure 6. Transmission profiles of common photometry filters—ultraviolet (U), blue (B), visible (V), red (R), and infrared (I) (Coelho, Calvão, Reis, & Siffert, 2014). Each peak is normalized to 100 percent.

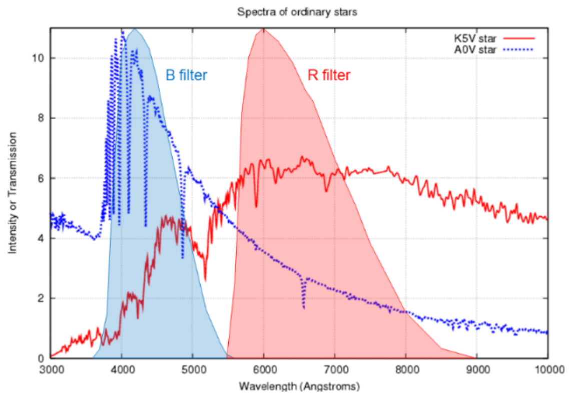


Figure 7. Spectra of blue A0 star and red K5 star (Richmond, 2007) with B and R filter transmission profiles from Figure 6 superimposed. The B filter favors light from the A0 star and the R filter favors light from the K5 star.

Color filters are attractive because of their intuitive operation, compatibility with other tools, and wide support in the astronomical community. They generally cannot be used to isolate stars with similar temperatures, because these stars will possess similar spectral signatures. While they can help reveal hidden stars by capturing evidence of their spectra, they do not enhance telescope resolution directly.

1.3.2 Occulting bars

Occulting bars are optical obstructions placed at the field stop of a telescope used to block or heavily attenuate light in a region of the image. These are especially useful when using a CCD to image binary stars with a large brightness difference, where light from the brighter star otherwise tends to saturate the sensors and cause a blooming effect that smothers the secondary star (Daley, A Method of Measuring High Delta m Doubles, 2007). Figure 8 demonstrates an occulting bar's operation.

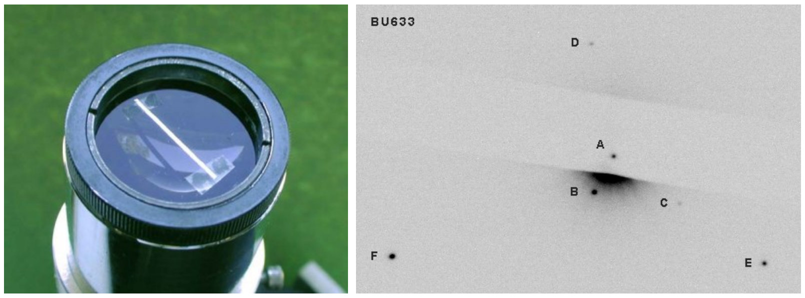


Figure 8. Left, an occulting strip affixed to a field lens (Daley, A Method of Measuring High Delta m Doubles, 2007); right, color-inverted image of γ Draconis captured with the occulting strip (Daley, A Method of Measuring High Delta m Doubles, 2007). Without attenuation, the light from star A would saturate the CCD sensors and hide other stars in the image.

The shape of occulting bars varies from narrow strips to larger semicircles that block half the field of view (Daley, A Method of Measuring High Delta m Doubles, 2007; Eagle, 2013). Most are constructed by astronomers out of common materials such as aluminum foil, electrical tape, or toothpicks. Whatever their shape or material, the bars must be placed exactly at the field stop for full effect. The closer to one another the stellar subjects are, the more precise the positioning of the bar and the telescope must be.

Occulting bars are useful where long exposures of bright stars dominate dimmer stars in the resulting image. They can also be used in conjunction with other telescope equipment. However, occulting bars do nothing to counteract diffraction effects. Airy rings from a bright primary star can still drown out light from a dim secondary, bar or not. Thus, despite their flexibility and low cost, occulting bars alone will not solve our problem.

1.3.3 Lyot coronagraphs

A Lyot coronagraph is a mask, placed at the pupil plane of an optical system, that blocks undesired light diffracted by an earlier obstruction. It is typically used to counteract the scattering of light around an occulting dot, placed at the focal plane, that blocks light traveling near the optical axis (Figure 9). This technique is an effective way to cancel light that would otherwise appear at the center of an image and possibly wash out features of interest around the periphery. The technique was first introduced by Bernard Lyot in 1939 to assist in studying the sun's corona without the benefit of a solar eclipse (Oppenheimer, 2003) but finds modern use in exoplanet discovery (Caldwell & Gray, 1997; Perryman, 2011, pp. 152–153) and binary star detection (Boccaletti, Moutou, Mouillet, Lagrange, & Augereau, 2001).

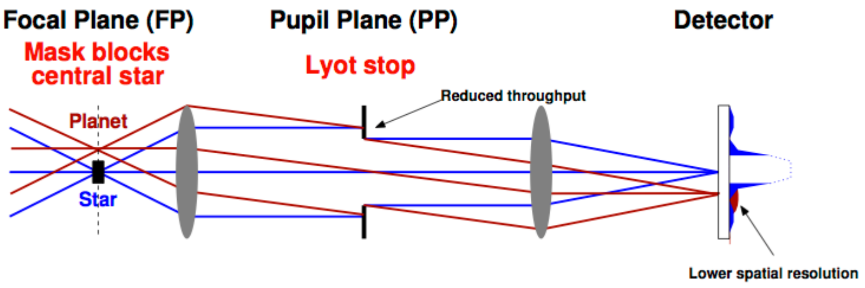


Figure 9. The function of a Lyot stop as applied to exoplanet discovery and observation (Kenworthy, 2018). An occulting mask (or "dot") at the focal plane blocks light but also casts a diffraction pattern that the Lyot stop, positioned at the pupil plane, reduces.

Unfortunately, Lyot stops are impractical for small- to mid-size telescopes because of the need to place additional hardware within the optical assembly. Thankfully, there is another way of using their masking principle to our advantage.

1.3.4 Apodizing masks

Apodizing masks are a category of optical filters that attempt to reduce or eliminate one or more Airy rings by changing the diffraction pattern. These masks can be categorized into two types: gradated, which have translucent profiles whose opacity varies as a function of the spatial coordinate; and binary, whose transmission at any point on the mask is either fully transparent or fully opaque. Apodizing masks are placed in the optical path at either at the pupil plane or near the entrance aperture of a telescope.

One implementation of a gradated apodizing mask is a mask whose opacity varies as a Gaussian function with respect to the radial coordinate (Park et al., 2002; Sacek, 2019a) (Figure 10, left). Such a mask, when applied to an aperture with no other obstructions, drastically reduces the Airy pattern surrounding a star's location (Figure 10, right; Figure 11).

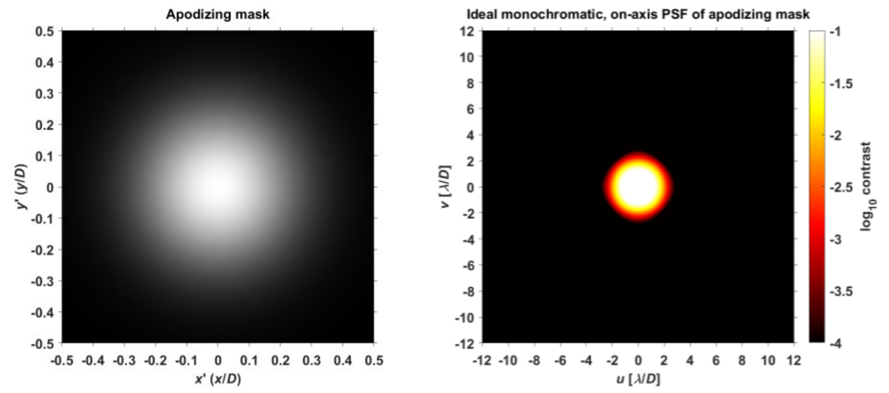


Figure 10. The transmission profile of an apodizing mask, left, and the mask's point spread function, right.

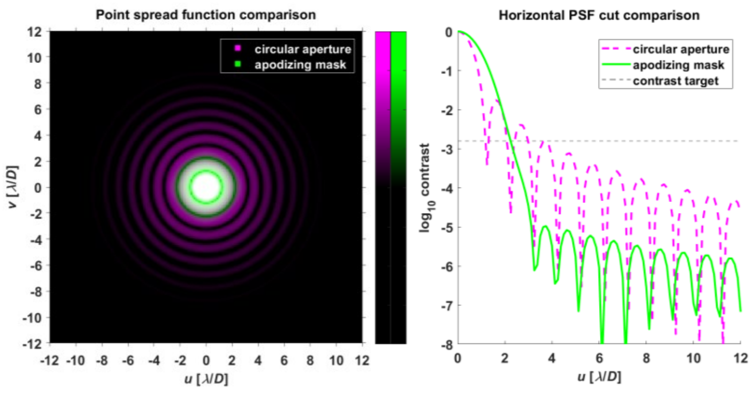


Figure 11. Comparison of the point spread function of a Gaussian-gradated apodizing mask to that of a circular aperture. Many Airy rings are effectively eliminated. (The fringes that remain, visible in the horizontal cut plot, are due to limiting the domain of the Gaussian transmission function to the shape of a circular aperture. This truncation causes an opacity discontinuity along the circumference.)

In addition to studying the Gaussian gradated apodizing mask of Figure 10, Park et al. (2002) studied masks with triangular and exponential opacity profiles and found that these gradients theoretically enhance resolution even further. Yet more options for gradated apodizing masks were included in Vanderbei et al. (2008) as solutions to a technique introduced in the paper that optimizes masks for arbitrary contrast and working angle targets.

Though gradated masks have many advantages on paper, their strict light transmission tolerances make them difficult to manufacture. For the translucent part of the mask, Park et al. (2002) proposed an etching process where the thickness of a chrome layer atop quartz is controlled to achieve a desired opacity; however, Martinez et al. (2009) wrote that varying the thickness of a metal layer introduces "wavefront phase errors" that would compromise the intended pattern.

As an alternative to using translucent materials, opaque elements can be patterned to produce regions whose light throughput is functionally equivalent to the desired translucency. To form their "microdot apodizer," Martinez et. al (2009) varied the density of tiny opaque dots of chrome rather than the chrome's thickness. An Internet search reveals that some crafty astronomers have also made their own apodizing masks by layering annular meshes (Suiter, 2001a–b; Florentino, 2009; Lovró, n.d.; Smith, n.d.). We can call these *apodizing screens*. The mask of Figure 12 (left) represents one such design with three layers of square mesh, each oriented at 30 degrees relative to the other two. The apodizing screen successfully diminishes the first Airy ring and somewhat improves the contrast of the telescope overall (Figure 13), but its contrast benefits fall well short of the smooth gradated apodizer seen in Figure 10. The screen's low fabrication cost partially offsets the underwhelming diffraction enhancement.

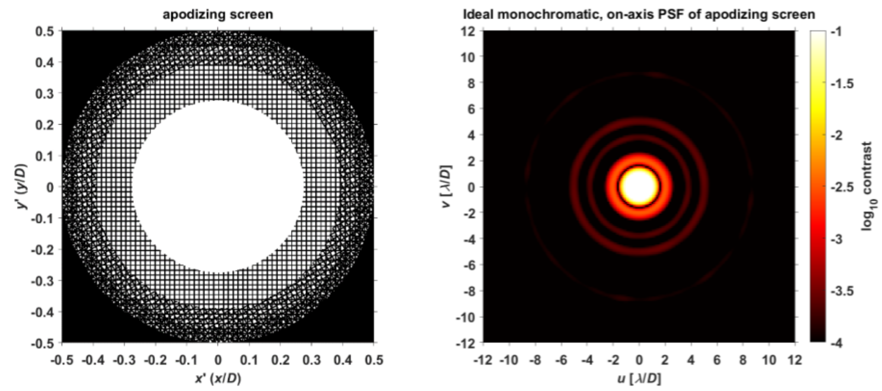


Figure 12. Apodizing screen imitating a Gaussian gradated apodizing mask, left, and its point spread function, right. The screen is arranged in layers of square mesh with circles cut to diameters 55 percent, 78 percent, and 90 percent of the full aperture's diameter (Lovró, n.d.; Smith, n.d.; Florentino, 2009).

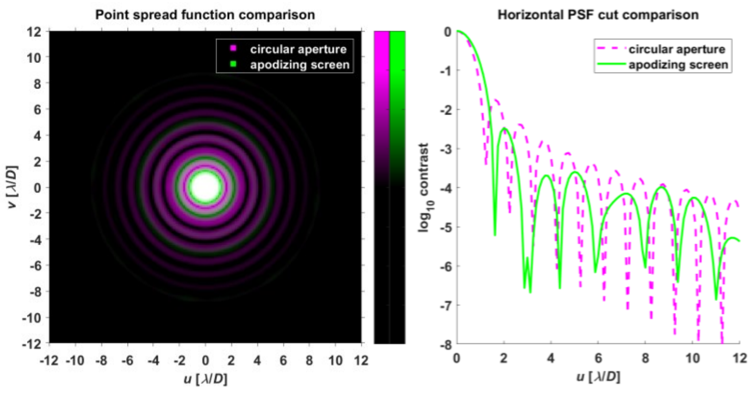


Figure 13. Comparison of the point spread function of an apodizing screen to that of a circular aperture. The screen offers only modest contrast benefits at working angles beyond about 2 λ/D.

These types of patterned masks are better called binary apodizing masks because their transmission at any point on the mask is either a maximized or minimized. At first, it might seem that binary masks' lack of intermediate opacity levels fundamentally compromises the optical performance that might be achieved using gradated masks, but Vanderbei et al. (2003) demonstrated[1] that a binary solution—specifically, a concentric ring pattern—provided better throughput than any circularly symmetric gradated mask that met prescribed contrast and working angle criteria. Later, Carlotti et al. (2011) found that binary apodizations performed better than gradated options in the presence of not just circular apertures but arbitrary aperture shapes. Kasdin et al.'s (2003) summary is perhaps the most efficient: "The best shaped pupils are as good as or better than the best graded apodizations."

The utility of binary masks was discovered far before our ability to optimize them. Functional and cosmetic diffraction effects of shaped apertures were well documented by John Herschel in his report of observations made at the Cape of Good

[1] Vanderbei et. al (2008) expands upon this.

Hope in the 1830s (Herschel, 1847). Herschel especially praised the impact of a triangular diaphragm on double-star observations, remarking that it "reduce[d] the discs to hardly more than a third of their size, and [gave] them a clearness and perfection incredible without trial." In the same report, he celebrated the striking beauty of the diffraction pattern's "perfectly straight, delicate, brilliant lines, like brightly illuminated threads."

Herschel's triangular mask is an example of a mask that is not circularly symmetric. All such asymmetric masks produce diffraction patterns whose light intensity varies across the azimuthal angle. As one example, the triangle mask creates a pattern with three thin, intersecting streaks (Figure 14). Between these streaks are six dark, triangular regions that provide good contrast for observing dim secondary stars. We call the regions of high contrast divided by the spikes *discovery zones* since they are the locations where faint neighboring stars would be most visible.

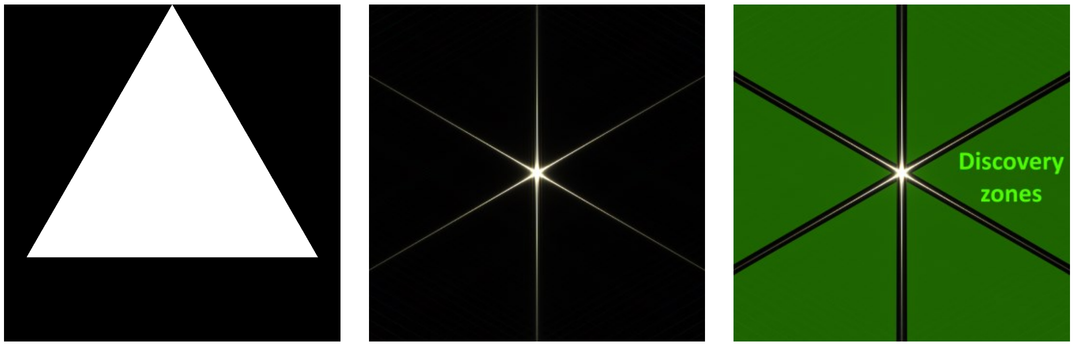


Figure 14. Simulation, center, of a star viewed through a triangular mask, left. Right, simulated image with high-contrast discovery zones highlighted in green. Simulated images were generated using Maskulator (Section 2.2.2). Discovery zone annotations were added by Foley.

Over the century and a half following Herschel's voyage, advances in optics theory, mathematical techniques, and computational capabilities would enable deliberate, bottom–up designs of alternate mask shapes. After Spergel (2000) introduced a Gaussian-shaped high-contrast pupil mask, Princeton University member Jeremy Kasdin found mathematical links between Spergel's design and prolate spheroidal wave functions described by Slepian & Pollack (1961) and Slepian (1965).[2] Leveraging these connections, Kasdin et al. (2003) and Kasdin et al. (2004) invented additional designs for contrast-enhancing pupil masks, a selection of which can be seen in Figures 15–17. Each shape strikes a different balance between contrast and discovery zone size. The authors were interested in the masks' use in exoplanet discovery, an application demanding exceptional contrast ratios on the order of 10^{-10} (Kasdin et al., 2003, p. 5).

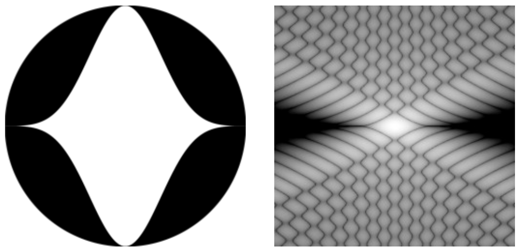


Figure 15. The "single Spergel–Kasdin prolate-spheroidal mask,"[3] left, and its point spread function, right, from Kasdin et al. (2003).

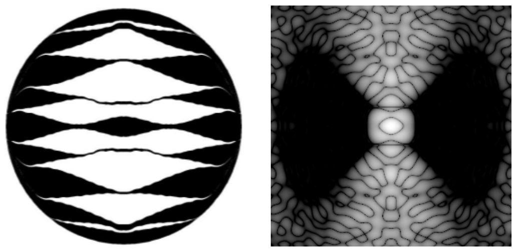


Figure 16. An "8-pupil circular eclipse-class mask," left, and its point spread function, right, from Kasdin et al. (2003).

[2] This history is summarized in Kasdin et al. (2005).
[3] This name appears in Kasdin et al. (2004).

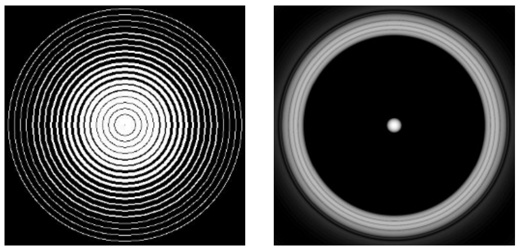


Figure 17. An "azimuthally symmetric mask," left, and its point spread function, right, from Kasdin et al. (2003).

Further research into shaped pupils, documented in a flurry of papers, produced additional variants of these masks along with some entirely new genera including Vanderbei et al.'s (2003) sharshape masks (e.g. Figure 18) and Vanderbei et al.'s (2004) checkerboard masks (e.g. Figure 19). These masks were designed to have the beneficial properties of azimuthally symmetric and one-dimensional apodizing masks while maintaining full structural continuity.

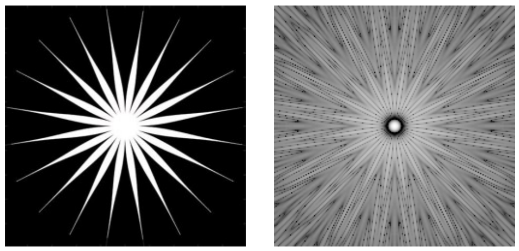


Figure 18. A 20-vane starshape mask, left, and its point spread function, right, from Vanderbei et al. (2003).

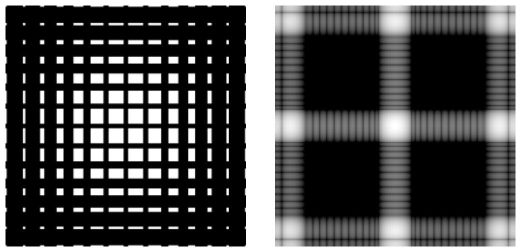


Figure 19. A "centrally-obstructed checkerboard mask," left, and its point spread function, right, from Vanderbei et al. (2004).

Many contrast-enhancing aperture masks have been manufactured and tested in astronomical applications.[4] Van Albada (1958) combined a wire grating with a “spindle shaped diaphragm” to study Procyon and I 1260 (Figure 20).[5] He concluded that “photographic observations of binaries with very large magnitude differences can be improved considerably by means of […] spindle shaped diaphragms” (van Albada, 1958). Daley (2014) reported successfully manufacturing and using a pupil mask resembling a Gaussian shape to resolve Sirius A and B with a 9-inch telescope (Figure 21).[6] Others have reported success using hexagonal apertures (Lindenblad, 1970), circular subapertures (Roberts Jr., 1998; Bernat, et al., 2010; Lacour, et al., 2011), and Gaussian subapertures (Debes et al., 2002; Ge et al., 2002; Debes et al., 2003; Debes & Ge, 2004). Figure 22 displays Debes & Ge’s (2004) subaperture approach.

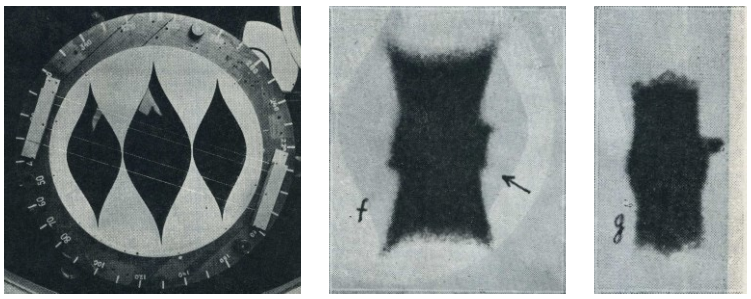


Figure 20. Mask and wire grating, left, used by van Albada to acquire images of Procyon, center, and I 1260, right (van Albada, 1958).

[4] Roberts Jr. (1998, pp. 61–78) contains a remarkably complete history of aperture masking up to the late 1990s with emphasis on aperture masking used in conjunction with interferometry techniques. The work of van Albada (1958) is one notable omission in this report.
[5] Though van Albada’s (1958) spindle shapes resembled Gaussian openings, they were formed using polynomials of even degree ≥ 4 that satisfied certain position and slope constraints. The Gaussian profile introduced by Slepian (2000) satisfied similar constraints.
[6] Sirius, being a star system whose components are separated in brightness by about 10 apparent visual magnitudes (Daley, 2014) or a log-10 contrast of -4, exceeds the contrast specification for our project; however, the angular separation of 9.66 arcseconds does not come close to challenging the resolving power of the telescope.

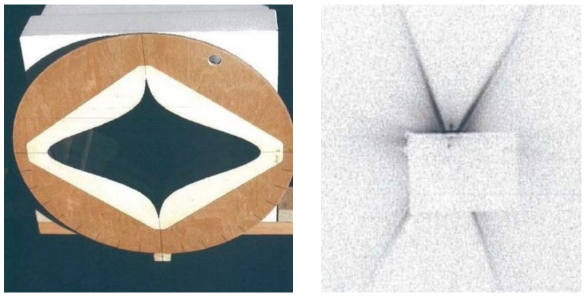


Figure 21. Left, the mask used by Daley to observe Sirius (Daley, 2014). Right, the corresponding CCD image recorded using this mask in tandem with a small "coronagraph focal mask foil" to reveal Sirius B (Daley, 2014).

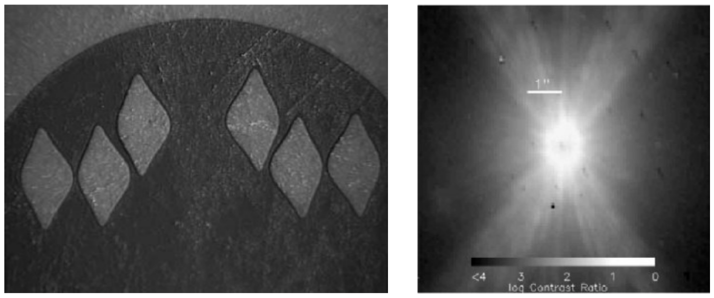


Figure 22. Gaussian subaperture mask (Debes & Ge, 2004), left, and resulting exposure of ε Eri (Debes et al., 2002), right.

Though they are used as telescope focusing aids rather than contrast-enhancing aperture masks, the Bahtinov mask (Figure 23) (Bahtinov, 2005) and Carey mask (Figure 24) (Carey, 2009) also represent successful realizations of aperture masks that operate on diffraction principles. Specifically, sets of coarse, parallel slots in the masks create diffraction spikes that communicate telescope focus. Both the Bahtinov and Carey masks are now well known and widely used in the astronomical community because of their utility, ease of manufacturing, and ease of use. This speaks well to the potential for astronomers to adopt other useful masks in their studies.

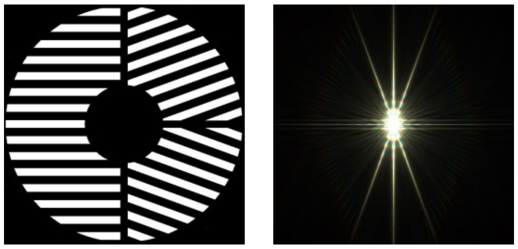


Figure 23. A Bahtinov mask, left, and a simulation of a star viewed through the mask using a well-focused telescope, right. Both images are from Niels Noordhoek's Maskulator utility (Section 2.2.2).

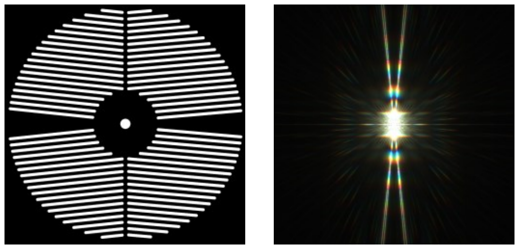


Figure 24. A Carey mask, left, and a simulation of a star viewed through the mask using a well-focused telescope, right. Both images are from Niels Noordhoek's Maskulator utility (Section 2.2.2).

The masks mentioned so far that have been successfully tested have full structural connectivity, leaving no freestanding opaque elements. Masks without mechanical continuity require some support structure. If the support structure is opaque, it will affect and potentially compromise the diffraction pattern. Using glass or a similar transparent substrate beneath the opaque layer can introduce phase distortion unless the smoothness and shape is controlled very carefully. In the context of support structures, Vanderbei et al. (2003) wrote, "It is felt that glass cannot be used because of the inevitable scatter that would result." Martinez et al. (2009) nonetheless successfully used a glass substrate polished to a roughness of $\lambda/20$ peak-to-valley for their pupil mask but noted that the substrate had "the highest quality requirement of all components used in [their] experiment." Difficulties related to the phase distortion caused by the glass substrate were also reported by Carlotti et al. (2011) along with an acknowledgement that this element is subject to "contamination" that could cause "additional amplitude and phase aberrations." Balasubramanian et al. (2015) replaced the transparent material with a very flat, highly reflective aluminum substrate, thus reducing phase errors but requiring a new optical configuration. To avoid these concerns, we will limit ourselves to structurally continuous masks from this point forward.

1.4 Selection of method

The contrast-enhancing behavior of apodizing masks, combined with the masks' convenient execution at the telescope aperture, makes them an ideal candidate for further study. Most existing astronomical apodization research concerns pupil masks inserted into the optical paths of large telescopes, leaving relatively few documented examples of aperture masks being integrated with small telescopes.[7] The deficit of published science in this area opens a natural opportunity for us to help advance binary star discovery and observation. Conveniently, aperture masks can be used in concert with other equipment used in this field such as color filters and occulting bars.

In the following chapters, we discuss simulating aperture masks, designing suitable masks for a Celestron C11 optical tube assembly, producing a mask rotation mechanism, and testing these elements.

[7] Herschel (1847), van Albada (1958), and Daley (2014) represent some of these examples.

2. SIMULATING SHAPED APERTURES

Modeling the optics of shaped aperture masks is a vital step toward efficiently simulating and optimizing different designs.

2.1 Optics summary

The point spread function of an aperture describes the image-plane flux density distribution created by an infinitesimally small, concentrated point of light viewed through the aperture (Hecht, 2002, p. 503).[8] Naturally, stars closely approximate concentrated sources, so the point spread function effectively describes the appearance of a star viewed through an aperture in the presence of diffraction effects. For example, the point spread function of a circular aperture is the Airy pattern (Figure 4). The more general term *power spectrum* also describes electromagnetic energy distributions, but we will use it to describe the compound pattern created by multiple sources. When viewing a single star in perfect optical conditions, the power spectrum is the same as the point spread function.

Under the conditions of Fraunhofer diffraction, which applies to the problem of viewing objects at large distances, the point spread function of an aperture is proportional to the square of the magnitude of the two-dimensional Fourier transform of the aperture's spatial transmissivity function (Weisstein, 2007; Cross, 2000):

$$P(u,v) = C \left\| \int_A \xi(x',y') \exp\left[-2\pi i \left(\frac{u}{\lambda}x' + \frac{v}{\lambda}y'\right)\right] dx'dy' \right\|^2 \tag{4}$$

[8] Mathematically, this function can be thought of as an impulse response.

In this equation, P is the point spread function, u and v are angular coordinates, C is a proportionality constant, A is the aperture, ξ is the transmission function of the aperture (1 for transparent regions, 0 for opaque), x' and y' are position coordinates in the aperture plane, i is the imaginary unit, and λ is the wavelength of light. Appendix A summarizes the optics theory that enables this conclusion.

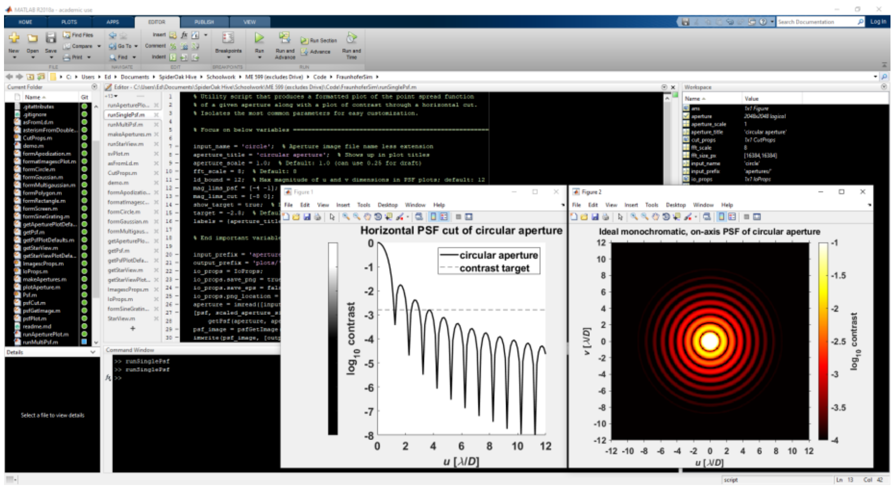
The link between Fraunhofer diffraction and the Fourier transform is serendipitous because it allows us to apply well-understood properties of the Fourier transform in our designs and unlocks the power of computing algorithms specialized for this operation—in our case, the two-dimensional fast Fourier transform (FFT). Wielding these tools, we can effectively and efficiently simulate point spread functions for arbitrary aperture shapes.

2.2 Diffraction simulation

There are many tools available for simulating Fraunhofer diffraction via fast Fourier transforms. We pursued a custom MATLAB solution for most of our modeling. In parallel, we used an existing diffraction visualization program called Maskulator that was written by Niels Noordhoek, an astronomer who studied a problem like ours (Noordhoek, 2009).

2.2.1 Diffraction simulation using MATLAB

MATLAB's digital signal processing and image processing capabilities are two faculties well suited to calculating and displaying power spectra in the digital domain. Appendix B fully describes the process we follow in MATLAB, beginning at an image representing the telescope aperture and ending in formatted figures of point spread functions or visualizations of star systems (Figure 25). These operations are supported by a network of functions written by Foley that are documented in Appendix K. Users have access to parameters that control simulation quality and various cosmetic attributes.



```
% Utility script that produces a formatted plot of the point spread function
% of a given aperture along with a plot of contrast through a horizontal cut.
% Isolates the most common parameters for easy customization.

% Focus on below variables ===============================================

input_name = 'circle';  % Aperture image file name less extension
aperture_title = 'circular aperture';  % Shows up in plot titles
aperture_scale = 1.0;  % Default: 1.0 (can use 0.25 for draft)
fft_scale = 8;  % Default: 8
ld_bound = 12;  % Max magnitude of u and v dimensions in PSF plots; default: 12
mag_lims_psf = [-4 -1];
mag_lims_cut = [-8 0];
show_target = true;  % 
target = -2.8;  % Defau
labels = {aperture_title

% End important variable

input_prefix = 'aperture
output_prefix = 'plots/
io_props = IoProps;
io_props.save_png = true
io_props.save_eps = fals
io_props.png_location = 
aperture = imread([input
[psf, scaled_aperture_si
    getPsf(aperture, ape
psf_image = psfGetImage(
imwrite(psf_image, [outp
```

```
>> runSinglePsf
>> runSinglePsf
>>
```

Horizontal PSF cut of circular aperture
circular aperture
contrast target
log10 contrast
u [λ/D]
Ideal monochromatic, on-axis PSF of circular aperture
v [λ/D]
log10 contrast

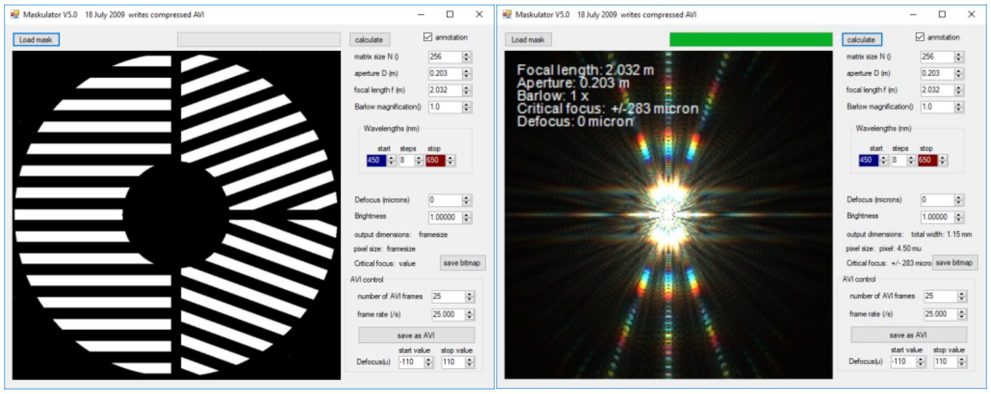
Figure 25. Diffraction simulation workspace in MATLAB.

The mask input image can be produced using any standard program that edits images; however, we provide MATLAB functions to generate many of the common shapes used in our analysis. These shapes include circles, polygons, gratings, and Gaussian functions (Appendix L). This fundamental geometry can easily be transformed and composed with other geometry to form subapertures, annuli, and other more complex shapes.

2.2.2 *Diffraction simulation using Maskulator*

Niels Noordhoek's Maskulator utility, shown in Figure 26, is another option for generating diffraction patterns of arbitrary masks (Noordhoek, 2009). Unlike our

MATLAB implementation, Maskulator can simulate imperfect telescope focus, making it an especially powerful tool for visualizing focus aids such as the Bahtinov and Carey masks. The program can also handle polychromatic light, which it does by dividing the spectrum into color slices and superimposing tinted instances of the monochromatic pattern that are scaled by the wavelength of the color at each slice.


Maskulator V5.0 18 July 2009 writes compressed AVI

matrix size N () 256
aperture D (m) 0.203
focal length f (m) 2.032
Barlow magnification() 1.0
Wavelengths (nm): start 450, steps 8, stop 650
Defocus (microns) 0
Brightness 1.00000
output dimensions: framesize
pixel size: framesize
Critical focus: value
AVI control
number of AVI frames 25
frame rate (/s) 25.000
Defocus(u) start value -110, stop value 110

Maskulator V5.0 18 July 2009 writes compressed AVI

Focal length: 2.032 m
Aperture: 0.203 m
Barlow: 1 x
Critical focus: +/-283 micron
Defocus: 0 micron

matrix size N () 256
aperture D (m) 0.203
focal length f (m) 2.032
Barlow magnification() 1.0
Wavelengths (nm): start 450, steps 8, stop 650
Defocus (microns) 0
Brightness 1.00000
output dimensions: total width: 1.15 mm
pixel size: pixel: 4.50 mu
Critical focus: +/- 283 micro
AVI control
number of AVI frames 25
frame rate (/s) 25.000
Defocus(u) start value -110, stop value 110

Figure 26. Niels Noordhoek's Maskulator utility operating with default settings on a Bahtinov mask.

Maskulator produces beautiful visualizations and is fast, powerful, and straightforward, but it lacks some customization features that would make it even more useful for this project. It offers no automatic means of padding the aperture image to enhance the resolution of the output. It also exposes no access to the values in its intermediate calculations, making it harder to adapt to applications the program is not explicitly designed for. A feature to simulate the viewing of multiple stars would be especially helpful for the purposes of this project.

Despite these shortcomings, Maskulator is a very useful tool. Appendix C contains information about acquiring and configuring this program.

2.3 Displaying power spectra

Power spectra have a very wide dynamic range, so we plot their output on a logarithmic scale to better visualize it. This approach parallels our eyes' nonlinear sensitivity (Portugal & Svaiter, 2011; Wilkes, 2015) and reflects the logarithmic basis of the log-10 contrast and apparent visual magnitude scales (Section 1.2). Without performing this adjustment, dim features and variations within them can easily be missed.

Because we are usually concerned with contrast within a point spread function rather than the pattern's absolute brightness, we normalize the output by assigning the brightest point—almost always the center—a contrast of 0. Dimmer areas, which provide better contrast, then have log-10 contrast ratios less than 0. This brightness normalization also helps keep the focus on the effect of the mask rather than the size or total light-collecting capacity of the telescope.

The angular coordinates of the point spread function also beg to be normalized. Under Fraunhofer diffraction, a point spread function will dilate in proportion to the wavelength of light and the reciprocal of the scale of the aperture producing it (Appendix A.1); thus, we can normalize our angular coordinates, writing them in terms of λ/D. Table 1 demonstrates how the conversion from λ/D to arcseconds varies by telescope diameter and wavelength, as calculated from Equation 5:

$$\begin{aligned} \theta\,[\text{as}] &= \left(\frac{\lambda\,[\text{nm}]}{D\,[\text{in}]}\right)\left(\frac{1\text{ m}}{1\times10^{9}\text{ nm}}\right)\left(\frac{1\text{ in}}{0.0254\text{ m}}\right)\left(\frac{360^\circ}{2\pi\text{ rad}}\right)\left(\frac{3600\text{ as}}{1^\circ}\right) \\ &\approx \left(\frac{\lambda\,[\text{nm}]}{D\,[\text{in}]}\right)\left(\frac{1\text{ as}}{123.14\text{ nm/in}}\right) \end{aligned} \tag{5}$$

Table 1. Equivalence of λ/D in arcseconds for different aperture diameters and observed wavelengths. Parenthesized letters denote the band. Wavelengths are the effective central wavelength of each band (Leibniz-Institut für Astrophysik Potsdam, 2014).

Telescope diam. [in]	**Infrared (I) (880 nm)**	**Red (R) (635 nm)**	**Yellow (V) (548 nm)**	**Blue (B) (435 nm)**	**Ultraviolet (U) (366 nm)**
6	1 $\lambda/D \approx$ 1.19 as	0.86 as	0.74 as	0.59 as	0.50 as
8	0.89 as	0.64 as	0.56 as	0.44 as	0.37 as
11	0.65 as	0.47 as	0.40 as	0.32 as	0.27 as
14	0.51 as	0.37 as	0.32 as	0.25 as	0.21 as

Our choice to normalize angular coordinates allows us to acquire a single characteristic point spread function for each aperture shape. Finding the actual angular extents of a diffraction pattern is as simple as multiplying the normalized angular coordinates by λ/D. For example, recalling from Section 1.2 that the radius of the Airy disk is about 1.22 λ/D, we can predict using Equation 5 that the angular extent of this radius for yellow light viewed through an 11-inch telescope will be about 0.494 seconds of arc (Equation 6).

$$\begin{aligned} \theta \approx & \quad (1.22)\left(\frac{548\text{ nm}}{11\text{ in}}\right)\left(\frac{1\text{ as}}{123.14\text{ nm/in}}\right) \\ = & \quad 0.494\text{ as} \end{aligned} \tag{6}$$

Armed with an arsenal of simulation tools and the knowledge of how to make use of our results, the next step is to finally apply our methods to a real telescope.

3. APERTURE MASKS FOR THE CELESTRON C11 OPTICAL TUBE ASSEMBLY

3.1 The Celestron C11 optical tube assembly

The C11 is an optical tube assembly produced by Celestron with an aperture diameter of 11 inches. As the C11 is a Schmidt–Cassegrain design, it has both a primary mirror and a secondary mirror. Light enters through the aperture, reflects off a primary mirror at the base of the telescope, reflects off a secondary mirror near the aperture of the telescope, and finally enters the eyepiece (Figure 27). The use of mirrors allows a more compact form than a refracting telescope of the same resolving power.

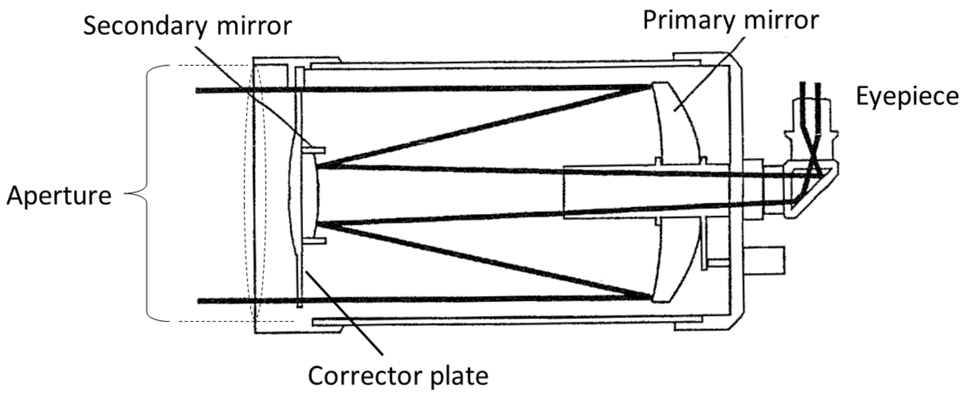


Figure 27. Key components of a Schmidt–Cassegrain telescope (United States of America Patent No. 7,595,942, 2009). Labels added by Foley.

A Schmidt–Cassegrain telescope's secondary mirror lies along the axis of the optical tube assembly, partially obstructing incoming light. This obstruction, approximately 3.9 inches in diameter on the C11, affects the diffraction pattern and the total light-collecting capacity of the telescope. Figure 28 shows the C11's annular aperture.

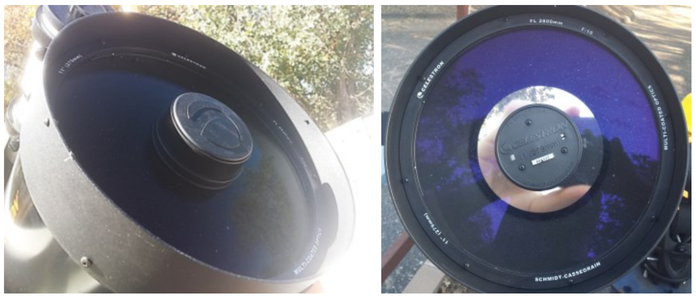


Figure 28. Two views of the aperture of the C11 captured by Foley. Note the optical obstruction caused by the corrector plate to which the secondary mirror is mounted internally. (In the image on the right, a cosmetic cap not affecting the aperture shape has been removed.)

A Celestron C11 was readily available for measurements and testing at the Orion Observatory in Santa Margarita, California. Already equipped for automated star surveys, the telescope had a motorized mount and a computer interface that could be programmed to operate various electrical peripherals. The C11 model is popular among smaller observatories, meaning any hardware designed for the Orion Observatory's telescope would be relevant to a broad portion of our intended audience.

Considering these attractive qualities, we selected the C11 for which to develop our masks and rotation mechanism. We began the process with a more thorough investigation into the telescope's optical properties.

3.2 Effect of central obstruction on diffraction pattern

The C11 aperture's 11-inch nominal diameter and 3.9-inch central obstruction combine to form a donut shape. Unmodified, this annulus creates a diffraction pattern that resembles, but is not identical to, the Airy pattern from a pure circular aperture. Figure 29 shows a comparison of the two apertures and their power spectra. If we overlay the two patterns, as in Figure 30, we see that the C11 concentrates energy in similar rings as the unobstructed aperture, but at different relative magnitudes. For example, the C11 distributes a greater proportion of its incoming light in bands near 1.6, 4.7, and 7.7 λ/D. These appear in Figure 30 (left) as green-tinted rings.

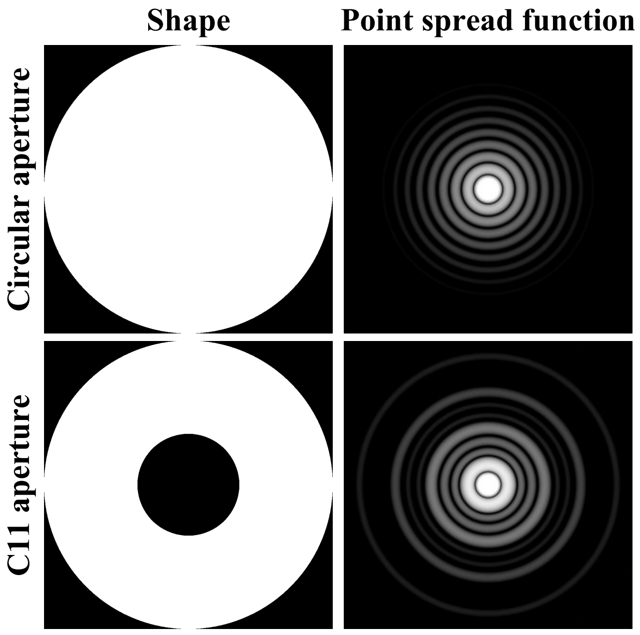


Figure 29. Comparison of shapes and characteristic diffraction patterns of a circular aperture and a C11 aperture. Brightnesses are plotted on a nonlinear brightness scale.

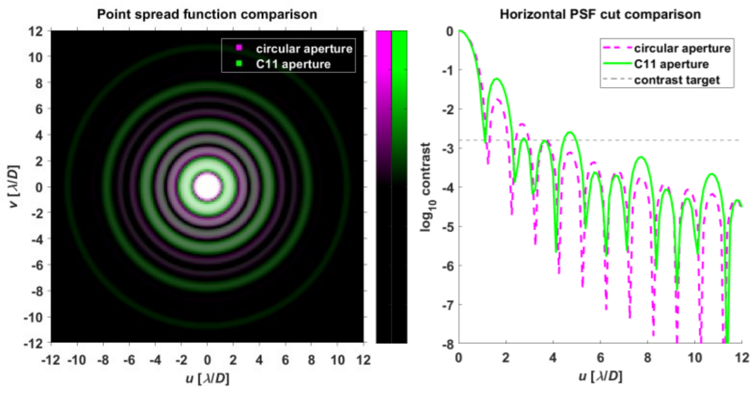


Figure 30. Comparison of power spectra of the C11 aperture and a circular aperture.

We see in Figure 30 that the contrast provided by the unmodified C11 satisfies our requirement of $10^{-2.8}$ when the working angle is greater than about $5.0\ \lambda/D$. Between 0 and $5.0\ \lambda/D$, it straddles the target, with some angles achieving the contrast specification and others not. For the sake of this paper, we will say that $2.3\ \lambda/D$ is the effective inner working angle of the C11, since the contrast exceptions beyond this point are contained within narrow angular spans. (The greatest offender is the peak of $10^{-2.6}$ that occurs near $u = 4.7\ \lambda/D$.) Given this decision, we should consider successful only those masks that, when added to the C11, reduce the inner working angle below $2.3\ \lambda/D$.

Applying a simple mask like those introduced in Section 1.3.4 to an annular aperture will not meaningfully improve the contrast at small working angles. The masks in Section 1.3.4 obscure an otherwise whole circular area, in effect transforming the shape of the telescope's aperture into that of the mask. However, when the mask fails to cover existing obstacles in the optical path, such as our secondary mirror, the point spread function of the composite aperture will exhibit diffraction artifacts from both the mask and the obstacles themselves. This is expected from the superposition property of Fourier transforms that model Fraunhofer diffraction (Appendix A.3). Figure 31 demonstrates a mask of this form that is unsuccessful because the discovery zones present in the Gaussian opening's point spread function are contaminated by diffraction effects from the central obstruction. The new artifacts appear as green-tinted bands in Figure 32.

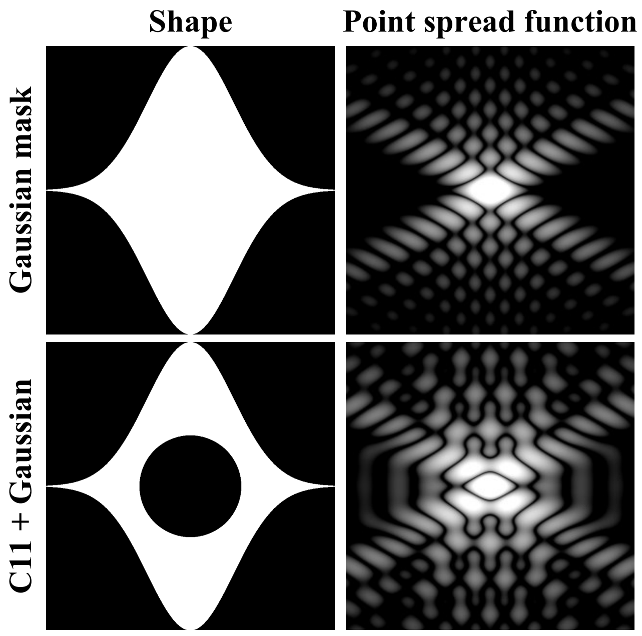


Figure 31. Comparison of shapes and characteristic diffraction patterns of a Gaussian mask and the compound pattern formed by a Gaussian mask placed on a C11 aperture.

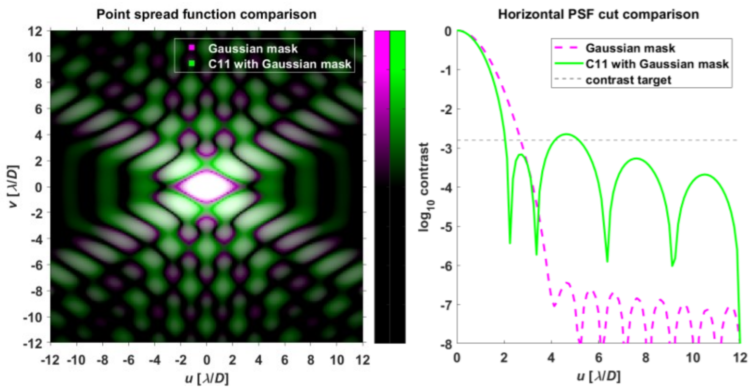


Figure 32. Comparison of point spread functions of a Gaussian mask alone and of a Gaussian mask combined with a C11 aperture. Vertical diffraction bands toward the left and right sides of the spectrum pollute a region that had high contrast in the original Gaussian design.

Clearly, the presence of the central obstruction requires extra attention because of its tendency to contribute unwanted diffraction artifacts. For clues on how to account for this, we look to published research.

3.3 Past attempts to mask a central obstruction

To circumvent an obstructed aperture on the much larger 100-inch telescope at Mt. Wilson, Debes et al. (2003) explored two designs derived from the Gaussian mask. In one, a Gaussian "secondary" obstruction is added to the aperture (Figure 33). In the other, the aperture is split into four smaller Gaussian openings (Figure 34). Both options completely eclipse the central obstruction so that it has no bearing on the final diffraction pattern.

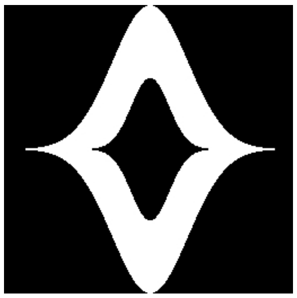


Figure 33. Gaussian with Gaussian secondary (Debes et al., 2003).

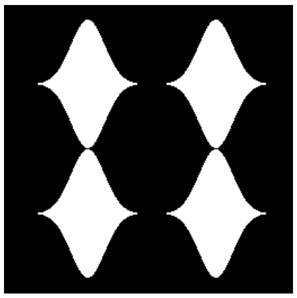


Figure 34. Multiple-Gaussian design (Debes et al., 2003).

Both masks had tradeoffs. In simulations, Debes et al. (2003) found that the Gaussian with a Gaussian secondary offered improved contrast relative to a circular aperture but only in regions not close to the central object. The multiple-Gaussian mask provided better contrast than the Gaussian secondary design but at the cost of a decrease in resolution due to the smaller openings. This group opted for the multi-Gaussian design.

In a separate study, Vanderbei et al. (2003) used a mathematical optimization routine to generate a concentric ring mask for an aperture with a 31-percent central obstruction (Figure 35). This mask theoretically attains a contrast of -10 log-10 magnitudes at working angles between 10 and 40 λ/D. Though the contrast provided by this mask far exceeds our specifications, the inner working angle is too large. It is also not clear how to manufacture such a mask without adding a support structure that might affect the diffraction pattern.

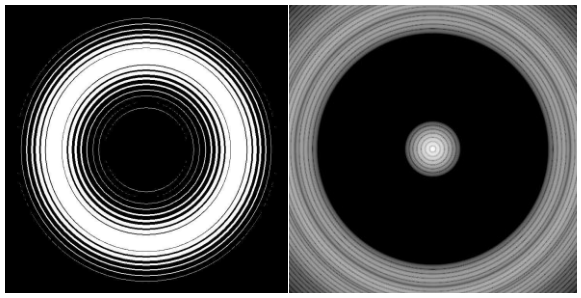


Figure 35. Left, a concentric ring mask designed for a circular aperture with a 31-percent central obstruction (Vanderbei et al., 2003). Right, the point spread function of the concentric ring mask (Vanderbei et al., 2003).

Tanaka et al. (2006) targeted an intermediate contrast ratio of 10^{-7} while developing a concentric ring mask for an early design of the Space Infrared Telescope for Cosmology and Astrophysics (SPICA) that had a four-armed spider structure in addition to a central obstruction.[9] The size of the secondary mirror had not yet been specified, so the group devised masks for a range of obstruction proportions between 10 and 24 percent. We focus on the 24-percent option because it most closely represents the C11. As shown in Figure 36 (right), the diffraction pattern of the composite aperture shape displays prominent artifacts caused by the support structure. The mask nonetheless achieved the team's target contrast at working angles between 6.0 and 12 λ/D along the diagonals. The fixed spider would seem to require reorienting the entire telescope to

[9] As of November 2019, the SPICA proposal specifies a three-legged spider and a 24-percent central obstruction (Roelfsema, 2019).

image dim stars that would otherwise appear on the cross-shaped diffraction concentration produced by a brighter star.

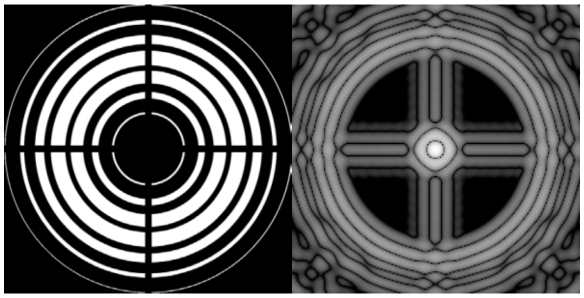


Figure 36. Left, a concentric ring mask placed on an aperture with a 24-percent central obstruction and a four-legged spider (Tanaka et al., 2006). Right, the corresponding point spread function (Tanaka et al., 2006). The point spread function clearly displays vertical and horizontal diffraction components produced by the horizontal and vertical beams of the support structure.

In the same paper, Tanaka et al. (2006) investigated a mask with a checkerboard design,[10] shown with its point spread function in Figure 37. This mask met the team's 10^{-7} contrast requirement at working angles between 5.4 and $11.4\ \lambda/D$.

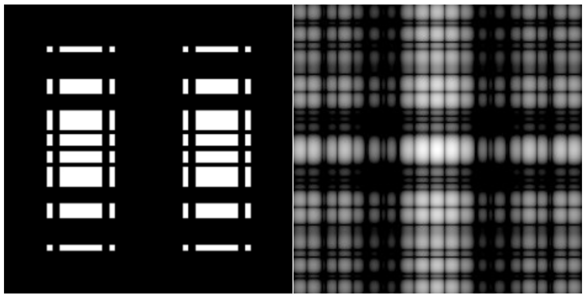


Figure 37. Left, an asymmetric checkerboard mask that accommodates a 24-percent central obstruction and a four-legged spider (Tanaka et al., 2006). Right, the corresponding point spread function (Tanaka et al., 2006).

[10] See Vanderbei et al. (2004) for information on checkerboard masks.

Years later, Enya & Abe (2011) adapted Kasdin et al.'s (2004) work on barcode masks to generate barcode mask solutions for SPICA assuming a 20% obstruction and a four-legged spider. These designs are shown with their point spread functions in Figure 38. The upper design achieved a contrast ratio of 10^{-6} at working angles between 3.4 and 15 λ/D; the lower achieved $10^{-5.3}$ between 3.3 and 10 λ/D.

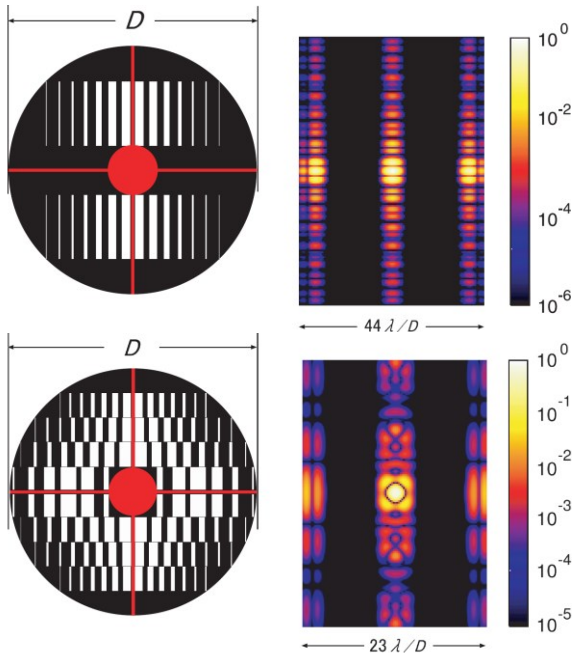


Figure 38. Left, barcode masks designed by Enya & Abe to accommodate a spider and a 20-percent central obstruction (Enya & Abe, 2011). The secondary and its support structure are shown in red; the masks themselves are in black. Right, the point spread functions of the composite apertures (Enya & Abe, 2011).

Carlotti et al. (2011) also volunteered an optimal solution for a 20-percent-obstructed, four-legged-spider SPICA design that provides a contrast ratio of 10^{-6} at working angles between 3.5 and 12 λ/D (Figure 39).[11] This mask, in contrast with

[11] The inner working angle for Carlotti et al.'s (2011) mask for SPICA is quoted in the paper as being both 3.5 λ/D and 3.3 λ/D. Seeing as all other working angles discussed in the paper are multiples of 0.5 λ/D, we believe 3.5 λ/D was the intended value.

Tanaka et al.'s (2006) and Enya & Abe's (2011) masks, does not exhibit full structural connectivity.

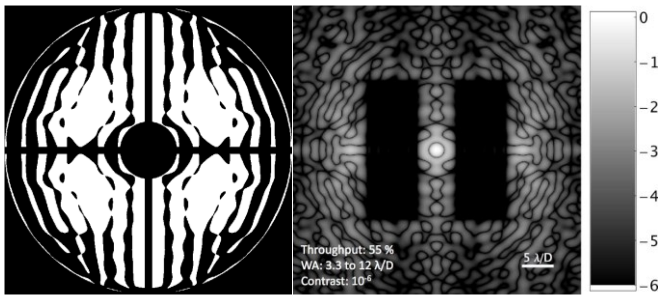


Figure 39. Left, an optimal pupil apodization for a version of SPICA with a four-legged spider and a 20-percent central obstruction. The design achieves a contrast ratio of 10^{-6} at working angles between 3.5 and 12 λ/D (Carlotti et al., 2011). Note the presence of free-standing islands. Right, the point spread function of this aperture (Carlotti et al., 2011).

Table 2 compares the masks introduced in this section. These solutions have three qualities in common:

1. Their contrast specifications are stricter than ours.
2. Their inner working angle requirements are looser than ours.
3. The circular obstructions the masks are designed for are smaller than ours.

Within this mask survey, Enya & Abe's (2011) second barcode mask achieves a contrast target closest to our own and offers the smallest inner working angle of $3.3\ \lambda/D$. Still, this working angle is significantly greater than the $2.3\ \lambda/D$ inner working angle of the unmodified C11 aperture. The mask is also not directly applicable because its central obstruction is much smaller than the C11's. Vanderbei et al.'s (2003) mask accommodates the largest obstruction but has a huge inner working angle and no structural connectivity.

Table 2. Summary of design criteria and performance metrics for masks in this section. The C11 aperture is included for reference.

Name	**Source**	**Fig.**	**Connected**[12]	**Obstr. diameter**	**Spider**	**Log-10 contrast**	**IWA $[\lambda/D]$**	**OWA $[\lambda/D]$**	**Notes**
Celestron C11	Celestron, LLC	29	N/A	35%	No	−2.8	2.3	∞	[13,14]
Gauss. w/second.	Debes et al. (2003)	33	No	≈ 25%	No	≈ −5.6	10	20	[15,16]
Multi-Gauss.	Debes et al. (2003)	34	Yes	≈ 25%	No	≈ −7.2	10	20	[15,16]
Rings A	Vanderbei et al. (2003)	35	No	31%	No	−10	10	40	–
Rings B	Tanaka et al. (2006)	36	Yes	24%	Yes	−7	6.0	12	–
Checkerboard	Tanaka et al. (2006)	37	Yes	24%	Yes	−7	5.4	11.4	–
Barcode A	Enya & Abe (2011)	38	Yes	20%	Yes	−6	3.4	15	–
Barcode B	Enya & Abe (2011)	38	Yes	20%	Yes	−5.3	3.3	10	–
Sym. dark hole	Carlotti et al. (2011)	39	No	20%	Yes	−6	3.5	12	–

[12] Masks are considered structurally connected if all opaque mask areas connect to a stationary point on the telescope, including the spider if applicable.
[13] For the Celestron C11, we quote the contrast ratio targeted by this project along with the corresponding working angles.
[14] Working angles shown here ignore a brief break in the high-contrast zone between 4.5 and 5.0 λ/D.
[15] Obstruction proportion estimated from Figure 3 in Debes et al. (2003).
[16] In their Figure 5, Debes et al. (2003) provided a chart displaying the average contrast ratios for several masks measured between select working angles. These working angles form the IWA and OWA values in our Table 2. The contrast ratios shown in our Table 2 were not specified directly by Debes et al. (2003) but have been interpreted from Figure 5 in their paper.

The survey of existing masks for obstructed apertures helps us understand the state of the art and gives us a chance to appreciate the wide diversity of shapes used to combat inconvenient diffraction properties of telescopes in their domain. It also informs us that our combination of a generous contrast requirement and a tight inner working angle requirement has little precedent in the astronomical field. This lack of direction simultaneously makes it more difficult and more satisfying to produce original qualifying mask designs.

3.4 New mask candidates

Armed with the conclusions of prior research and the tools to explore new mask variants, we at last propose our own masks for the Celestron C11 optical tube assembly.

3.4.1 Gaussian donut mask

Inspired by Spergel (2000) and the work of Debes et al. (2003), we design a "Gaussian donut" mask for the C11. We place within a nominal Gaussian aperture shape an opaque Gaussian secondary that covers the secondary mirror obstruction, like the mask of Figure 33.

The Gaussian donut shape presents several design choices, including the broadness of the outer Gaussian, the height of the secondary, and the broadness of the secondary. We must optimize these for our project objective of providing good contrast at small inner working angles.

Because we will be using Gaussian curves extensively, it benefits us to establish a convention for how to represent them. Gaussian profiles are traditionally defined in statistical applications using two parameters: a mean and a standard deviation. The distribution, which has an unbounded domain, is normalized such that its integral approaches one as the limits of integration extend toward positive and negative infinity. This ubiquitous bell-shaped curve is given by the following function.

$$f_g(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left[\frac{-(x-\mu)^2}{2\sigma^2}\right] \tag{7}$$

In this equation, f_g is the Gaussian distribution function, x is the independent variable, σ is the standard deviation (whose square is the variance), and μ is the mean of the distribution.

It is useful for us to stretch the bell curve vertically and horizontally to better understand how these transformations affect the diffraction pattern. To this end, we can ignore the restriction on the Gaussian distribution's integral and proceed to vary the peak height and standard deviation independently. For simplicity, we can say our mean is zero, establishing the center of the aperture as our datum. Letting h be the amplitude of our Gaussian peak, our equation becomes simpler:

$$f_g(x \mid \mu = 0) = h \exp\left(\frac{-x^2}{2\sigma^2}\right) \tag{8}$$

In most cases, when we change the height of the opening, we also want to change the width by the same factor so that we maintain the shape's proportions. An easy way to do this is to normalize the standard deviation by the peak height, defining $p = \frac{\sigma}{h}$:

$$f_g(x \mid \mu = 0) = h \exp\left(\frac{-x^2}{2p^2h^2}\right) \tag{9}$$

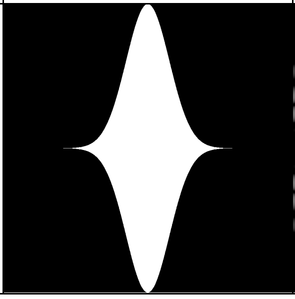
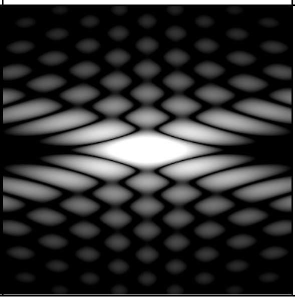
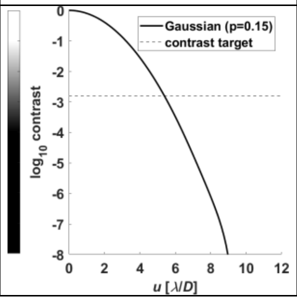
Because our designs are independent of any one aperture size, we can also normalize the height and width relative to the aperture diameter, D, by defining $a = \frac{h}{D}$ and $\hat{x} = \frac{x}{D}$. The term p is unaffected by this transformation because it is already dimensionless. We arrive at a simplified Gaussian profile equation fit for our use case:

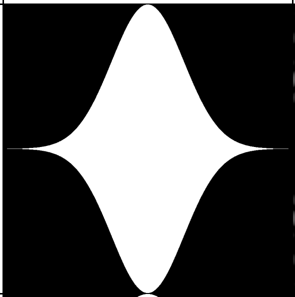
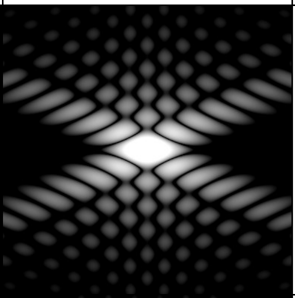
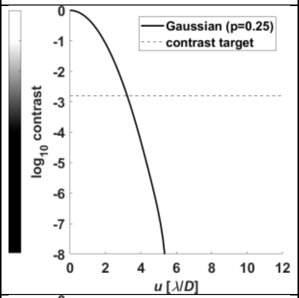
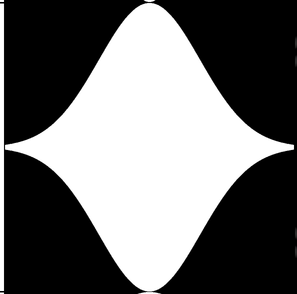
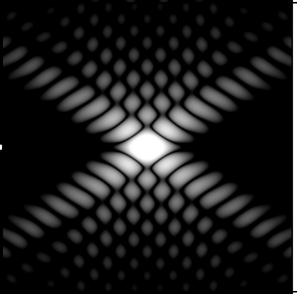
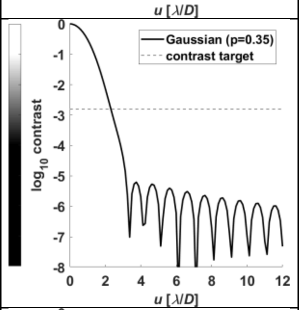
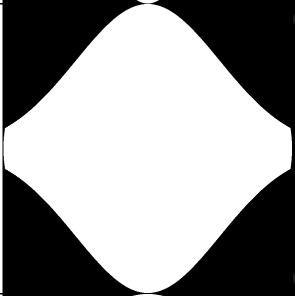
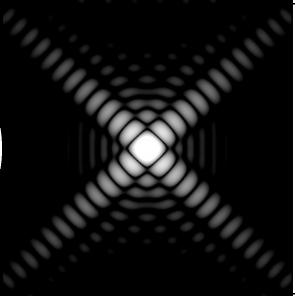
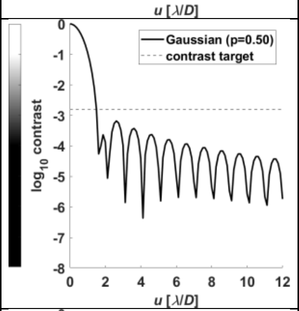
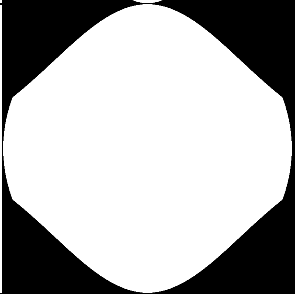
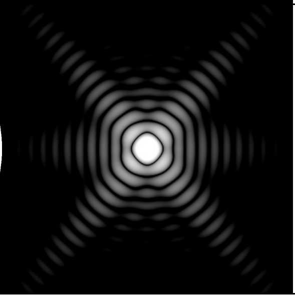
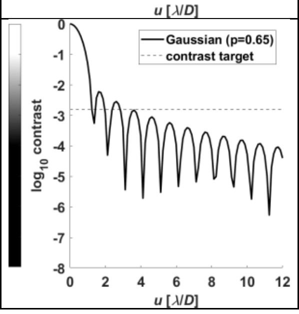
$$\boxed{g(\hat{x}) = a \exp\left(\frac{-\hat{x}^2}{2p^2a^2}\right)} \tag{10}$$

Adapting this normalized profile to a specific diameter is as simple as scaling the profile by D in both dimensions.

For our Gaussian donut mask, we set the outer profile height to the maximum value our aperture diameter accommodates, $a = 0.5$. From here, we vary p to explore its impact on the diffraction pattern. Table 3 compares the effect of different values of this term.

Table 3. Comparison of aperture shapes and power spectra for Gaussian profiles with different normalized standard deviation terms.

Std. dev. factor, p	**Aperture shape**	**Point spread function**	**Horizontal PSF cut**	**IWA $[\lambda/D]$**
0.15				5.4

Std. dev. factor, p	Aperture shape	Point spread function	Horizontal PSF cut	IWA $[\lambda/D]$
0.25				3.2
0.35				2.3
0.50				1.5
0.65				2.8

We see that as we begin to increase the value of p, we increase the angle of the discovery zones and decrease the minimum inner working angle. Both results are desirable for our application. This trend has its limits, however: as the broadness increases, the effect of truncating the tails of the Gaussian distribution becomes more significant. The truncated edges gradually resemble the contour of the original circular aperture, introducing deleterious diffraction artifacts along our discovery axis. We find that the Gaussian with $p = 0.50$ is the variant that meets the contrast target at the smallest working angle and maintains this contrast outward along the horizontal axis.

The shape of the Gaussian secondary is our next design consideration. This component must completely cover the secondary mirror cap lest the cap's circular shape add undesirable diffraction components. For best throughput, we decrease the amplitude of the secondary Gaussian for a given broadness to the point that the circular profile of the secondary mirror cap lies tangent to the Gaussian profile. This constraint on the amplitude reduces the two-dimensional problem of selecting a height and a breadth to a single dimension where only the broadness need be selected.

In this project, we choose to match the broadness factor between the inner and outer Gaussian curves. We know that a given broadness value will produce a discovery zone of a certain angle regardless of the peak height. We also know, due to superposition, that any mask formed by pairing Gaussian forms of mismatched broadness factors will produce in its diffraction pattern a complex superposition of the two hourglass shapes that each of its components would produce individually. In this situation, the maximum discovery zone angle is limited by the smaller broadness factor, so we suppose that picking a single factor for both Gaussian features will produce the widest possible search region.[17] Through some iteration, we find that $p = 0.50$ continues to work well in the donut form compared to other broadness factors. The shape formed by the outer and inner Gaussians is shown in Figure 40 along with its point spread function. This pattern is compared to the C11's in Figure 41.

[17] A promising variant with unequal broadness terms that challenges this reasoning is discussed briefly in Section 6.2.2.

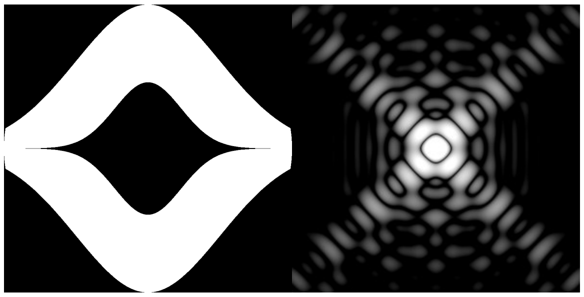


Figure 40. Gaussian donut shape (p=0.50) and its characteristic point spread function. Tails of the secondary Gaussian are so thin that they appear to vanish due to the limited resolution of the aperture image.

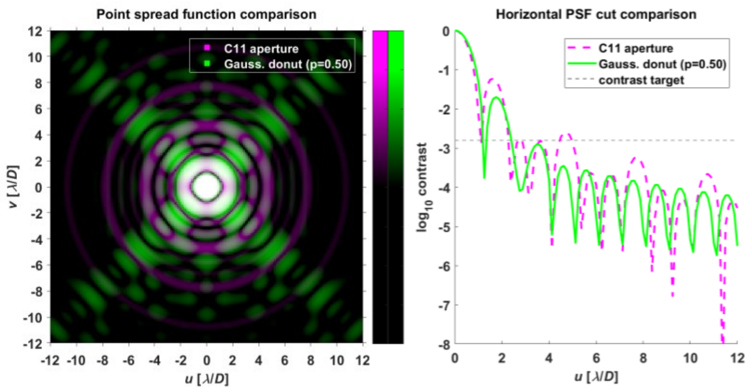


Figure 41. Point spread function comparison of the Gaussian donut shape (p=0.50) and the C11 aperture.

The addition of the secondary increases the inner working angle to 2.4 λ/D, well beyond the 1.5 λ/D offered by the singular Gaussian and even slightly higher than the unmasked C11's 2.3 λ/D. The combined shape does offer slightly better contrast at the prominent C11 diffraction bands (1.6, 4.7, 7.7 λ/D, etc.) but is otherwise unremarkable. The azimuthal restriction on the region in which even these middling benefits can be seen—with inferior results at other angles—makes the mask's use difficult to justify.

3.4.2 Multi-Gaussian mask

Another strategy for incorporating the Gaussian shape amid the C11's secondary mirror obstruction is to place multiple small openings around the perimeter of the mirror like Debes et al. (2003). Being careful to avoid locating an opening above the central obstruction, we array the Gaussian subapertures in a square. Through some trial and error, we find that a broadness factor $p = 0.65$ and amplitude $a = 0.18$ placed at horizontal centers $x' = \pm 0.225$ and vertical centers $y' = \pm 0.225$ produce a competitive working angle for this general arrangement. Figure 42 shows the resulting mask and its point spread function.

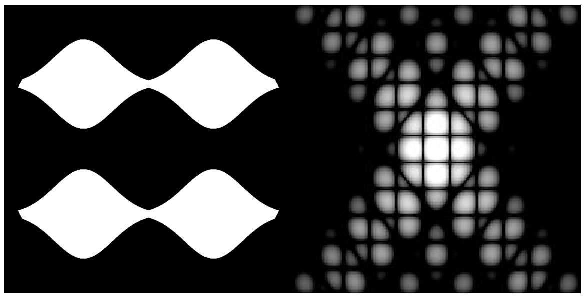


Figure 42. Multi-Gaussian profile and its point spread function.

It is clear from Figure 42 that the tails of the Gaussian subapertures are substantially truncated. This is true at the edges of the mask, where the tails collide with the limiting circular aperture. It is also true at the center of the mask, where pairs of tails overlap and effectively merge. One might expect the loss of tails to introduce noticeable artifacts into the point spread function, but it seems that either the magnitude of these effects is too small or the corresponding spikes appear at frequencies outside our working angle window. What *is* visible in the point spread function is an array of tiles that results from the regularly patterned subapertures.

Figure 43 compares the multi-Gaussian mask's power spectrum to that of the C11's default aperture. Unsurprisingly, the small Gaussian shapes create a broad diffraction pattern that results in somewhat low resolution. The inner working angle is 3.1 λ/D, greater than the Gaussian donut's 2.4 λ/D; the azimuthal discovery zone angle is about the same. These facts leave the multi-Gaussian design with few compelling features.

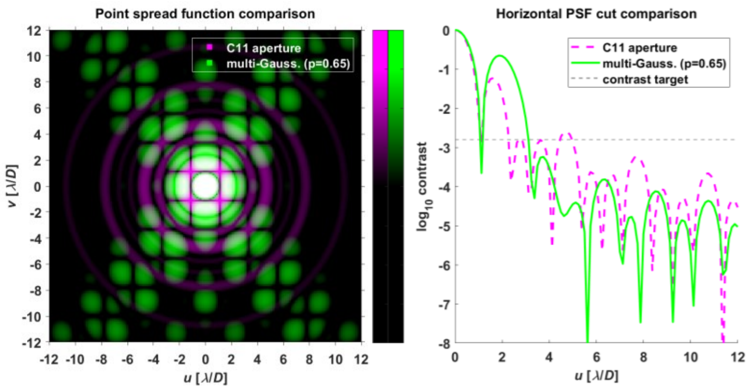


Figure 43. Comparison of the point spread functions of the multi-Gaussian mask and the unmasked C11 aperture.

3.4.3 Concentric ring mask

Applying a similar optimization routine as Vanderbei et al. (2003) but specifically to the C11 aperture and our modest contrast requirements, Princeton research astrophysicist Neil Zimmerman produced a concentric ring design that achieves a contrast ratio of $10^{-3.0}$ at a small inner working angle of 2.4 λ/D (Zimmerman, 2014) (Figure 44). This inner working angle matches that of the Gaussian donut, but the concentric ring mask excels by having no azimuthal limits to its discovery space.

Zimmerman noted that while the mask theoretically improves contrast relative to the unmodified aperture, the gains are small and may be difficult to leverage in practice. He also acknowledged the manufacturing challenge represented by the two freestanding rings and concluded that the concentric ring mask is "difficult to recommend for the project."

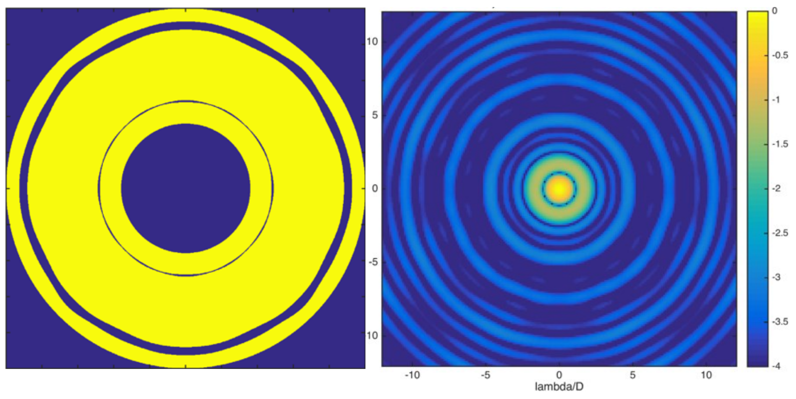


Figure 44. Left, a concentric ring mask optimized for our contrast requirements on the C11 (Zimmerman, 2014). Yellow regions represent transparency. Right, the mask's point spread function (Zimmerman, 2014).

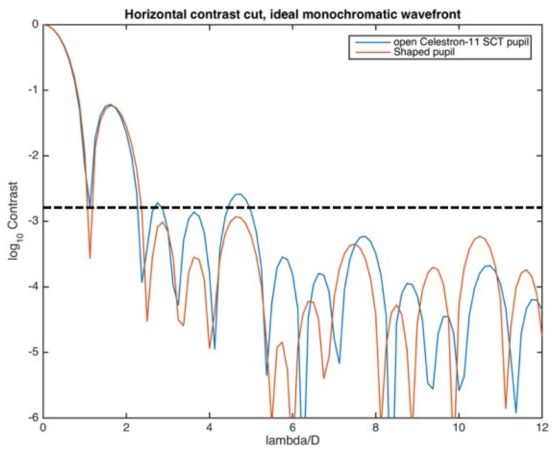


Figure 45. Plot showing the contrast of the concentric ring mask ("shaped pupil") relative to the C11 aperture (Zimmerman, 2014). The dotted line, superimposed by Foley, denotes the contrast target.

3.4.4 Bowtie mask

Neil Zimmerman proposed a second custom mask design for the C11, this one engineered to achieve a contrast ratio of $10^{-2.7}$ at the smallest possible inner working angle (Zimmerman, 2014) (Figure 46). This contrast requirement is slightly less strict[18] than our own of $10^{-2.8}$, but it is far closer than the targets seen in existing literature (e.g. Table 2). To produce the mask design, Zimmerman executed a numerical optimization process that maintained continuity at the outer and inner edges of the telescope aperture (Zimmerman, 2014). Owing to the mask's point spread function's "bowtie-shaped wedges," Zimmerman titled the creation a "bowtie mask" (Zimmerman, 2014).

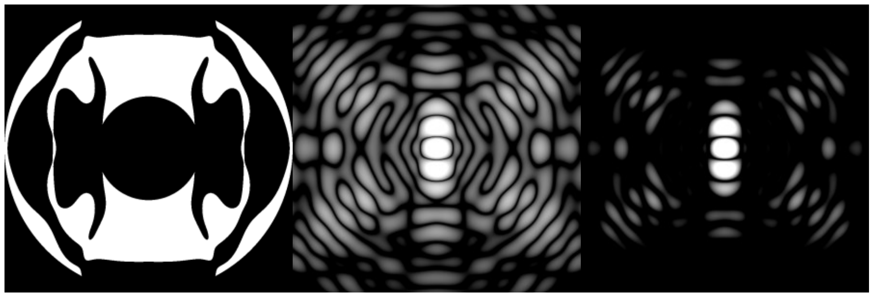

Bowtie mask
PSF, grays on log-10 $(-4, -1)$
PSF, grays on log-10 $(-2.7, -1)$

Figure 46. Left, Neil Zimmerman's bowtie mask (Zimmerman, 2014). Center and right, the bowtie mask's characteristic point spread function displayed at two different grayscale calibrations (Foley). In Zimmerman's view, the discovery zone shape resembles a bowtie, hence the mask's name.

The bowtie mask achieves its contrast requirement at working angles between 1.2 and 8.0 λ/D along the horizontal axis (Zimmerman, 2014)[19]. This inner working angle is

[18] Similar solutions for stricter contrast requirements produced unacceptable free-standing islands in the mask shape (Zimmerman, 2014).
[19] Discovery zone inner working angle quoted from Zimmerman (2014); outer working angle deduced by Foley from a plot in Zimmerman (2014).

smaller than the angles offered by all other masks we have seen so far. Figure 47 compares the performance of the bowtie mask to a C11 aperture and our original contrast target.

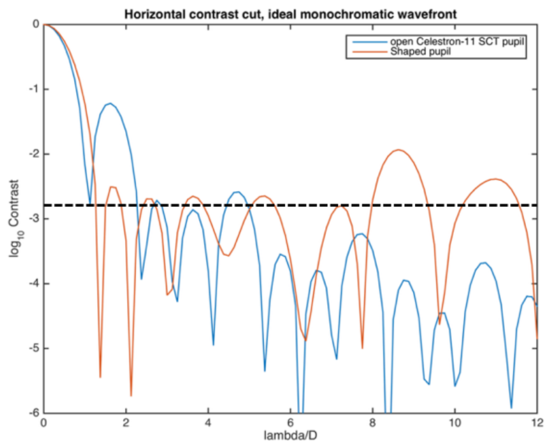
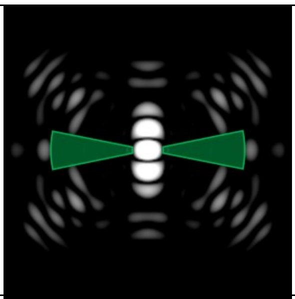
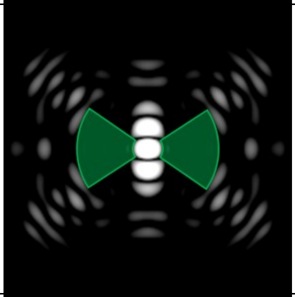
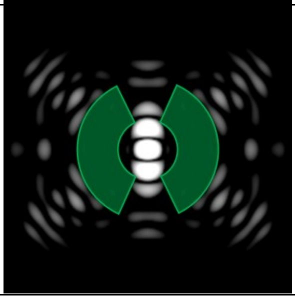


Figure 47. Plot showing the contrast of the bowtie mask ("shaped pupil") relative to the C11 aperture (Zimmerman, 2014). The dotted line, superimposed by Foley, denotes the contrast target we have used to this point. The contrast target the bowtie mask was designed to is slightly less strict, explaining some peaks that cross the dotted line.

Unlike the concentric ring mask, the bowtie mask does not have full rotational symmetry. Instead, the high-contrast areas take the form of wedges limited both by azimuthal angle and working angle. Zimmerman quotes the azimuthal span of each wedge as 90 degrees (Zimmerman, 2014), but the true breadth varies depending on the desired outer working angle. As conveyed by Table 4, this angle can vary between 24 and 132 degrees per wedge depending on the working angle window.

Table 4. Comparison of three different sets of parameters describing wedge-shaped discovery zones in the bowtie mask's point spread function (Foley). Where wide working angle limits are required, the azimuthal span of the search area is small. Inversely, where tighter working angle limits are acceptable, the azimuthal span of the search angle is large.

Wedge interpretation	**PSF and discovery zones**	**IWA $[\lambda/D]$**	**OWA $[\lambda/D]$**	**Arc angle (each)**
Narrow		1.2	8.0	24°
Medium		1.2	5.9	66°
Wide		2.4	5.9	132°

Regardless of the discovery zone's shape, we should not overlook the fact that the bowie mask is the first mask we have seen whose inner working angle is smaller than that of the default C11. This property more than makes up for the search area's limits in azimuth and outer working angle.

3.5 Mask selection

Table 5 summarizes the masks presented in this section. Of these masks, Zimmerman's bowtie mask is a clear winner because it achieves its contrast requirements at a narrow inner working angle and across a generous azimuthal span.

Table 5. Comparison of mask candidates for the Celestron C11. Unmasked Celestron C11 aperture included for reference.

Name	**Source**	**Fig.**	**Struct. connec.**	**Log-10 contrast**	**IWA $[\lambda/D]$**	**OWA $[\lambda/D]$**	**Az. cvg.**
Celestron C11	Celestron, LLC	29	N/A	−2.8	2.3	∞	360°
Gaussian donut	Foley, inspired by Debes et al. (2003)	40	No[20]	−2.8	2.4	∞	90°
Multi-Gaussian	Foley, inspired by Debes et al. (2003)	42	Yes	−2.8	3.1	∞	85°
Concentric ring	Zimmerman (2014)	44	No	−3.0	2.4	∞	360°
Bowtie[21]	Zimmerman (2014)	46	No	−2.7	1.2	5.9	132°

[20] Though structurally connected in a mathematical sense, the Gaussian donut mask has a bridge between the Gaussian secondary and the rest of the mask that is too thin for any practical realization.
[21] Values cited in this row are based on the "medium" wedge interpretation from Table 4.

For the sake of variety, we would like to select a second mask to accompany the bowtie mask. Given that the list in Table 5 offers no obvious runner-up, we choose to produce both the Gaussian donut and multi-Gaussian masks as alternatives. The fact that these masks exceed our contrast specification at large working angles but compromise on the inner working angle provides an interesting counterpoint to the bowtie mask, which improves upon the inner working angle specification of the C11 but achieves a less extreme contrast. In addition, the distinctive hourglass-shaped discovery zones of the Gaussian-family masks readily reveal the orientation of the masks in their resulting images—a useful feature during testing.

3.6 Enforcing structural connectivity

For all its benefits, Zimmerman's bowtie mask is not structurally connected by default. Similarly, the Gaussian donut mask requires reinforcement to be mechanically viable. The multi-Gaussian mask, being structurally sound by default, requires no special treatment.

3.6.1 Structural accommodations for bowtie mask

Zimmerman's original bowtie mask design (Section 3.4.4) exists in two separate components: one attaches to the secondary mirror cylinder; one attaches to the rim of the telescope. Because we prefer structural connectivity across the entire mask, we add four 1/8-inch beams between the inner and outer components of the mask at the regions of least separation, as seen in Figure 48.

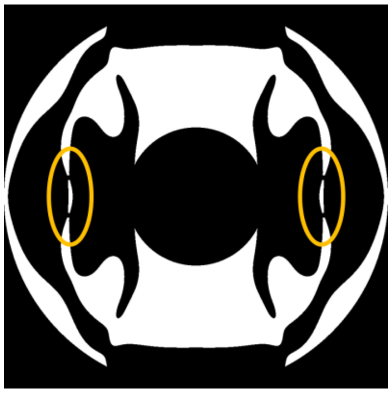


Figure 48. Form of 11-inch bowtie mask with 1/8-inch bars added to maintain structural continuity. Added bars are identified by orange ovals.

We choose these locations to add bridges because they are the locations where the least new material is required. We expect from our understanding of diffraction that small perturbations to the aperture shape correspond to high-frequency spatial components that will manifest themselves as diffraction spikes at working angles outside our range of interest. Indeed, we see in Figure 49 that the bridging between the interior and exterior of the pupil mask does not meaningfully affect the diffraction pattern that arises. The differences are negligible to the point of being almost invisible.

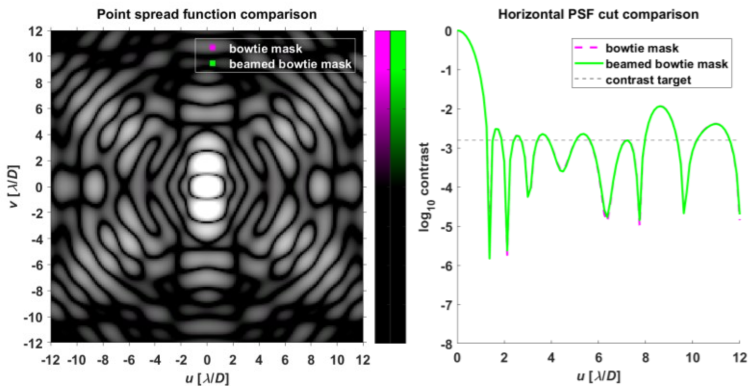


Figure 49. A comparison of the point spread functions of the beamed bowtie mask and the original bowtie mask. No significant difference is visible on this domain.

3.6.2 Structural accommodations for Gaussian donut mask

The Gaussian donut mask of Figure 40 maintains structural integrity in a mathematical sense because the thin tails of the secondary connect to the outside of the circular aperture; however, these tails are vanishingly thin. For practical purposes, a more robust support structure must be added between the Gaussian secondary and the outside of the mask.

Debes et al. identified the orientation of the support structure for their Mt. Wilson mask as "crucial," warning that "spider arms can completely ruin the high contrast axis if positioned perpendicular to the mask's horizontal axis" (Debes et al., 2003). The group oriented their four-armed support structure[22] at 45 degrees to the horizontal to place diffraction spikes "in the brighter regions of the pattern" (Debes et al., 2003).

Our support structure does not necessarily need to have four arms. We can readily avoid problematic vertical elements by placing a single 3/16-inch beam directly along the horizontal axis (Figure 50). Though this addition very slightly increases our inner working angle from 2.4 λ/D to 2.5 λ/D, the high-contrast regions remain intact (Figure 51). The differences are more pronounced along the vertical axis, where the beamed variant produces a strong and colorful diffraction spike (Figure 52).

[22] Debes et al. (2003) designed pupil masks for use with the Mt. Wilson 100-inch telescope. At 4 millimeters in diameter, these pupil masks were much smaller than our 11-inch aperture masks. The extremely thin support structure would appear to have made designs with two supports too fragile.

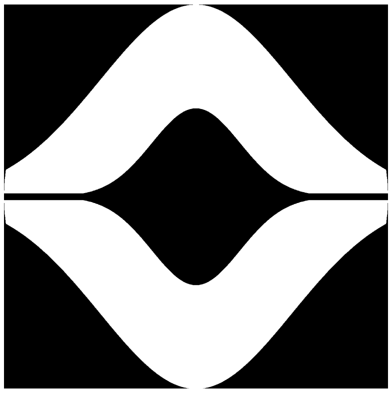


Figure 50. Gaussian donut with horizontal beam.

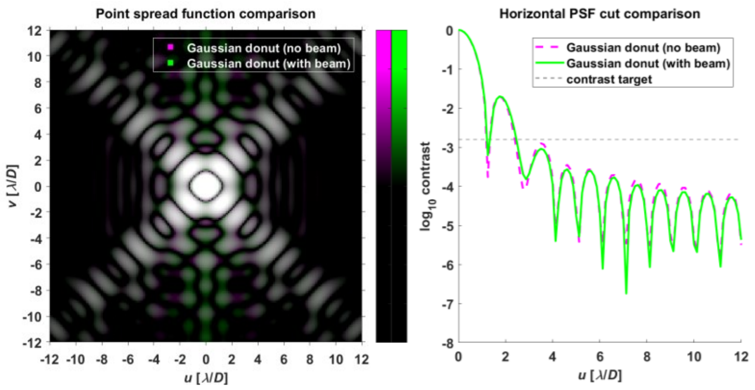


Figure 51. Point spread function comparison of the beamed and non-beamed Gaussian donut variants. The differences are very minor along the horizontal axis.

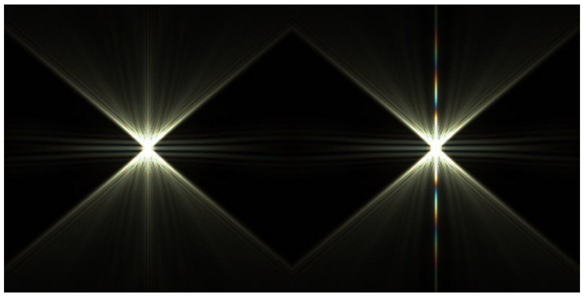


Figure 52. Comparison of Maskulator point spread function renders for Gaussian donuts without a beam and with a beam. The beam produces a vertical diffraction spike but has a negligible impact along the discovery axis.

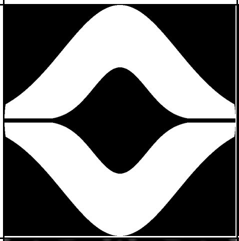
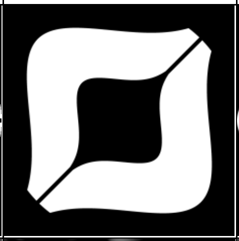
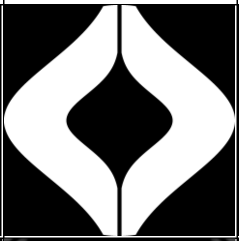
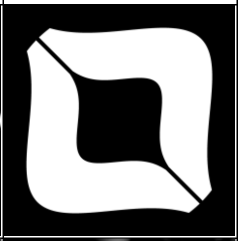
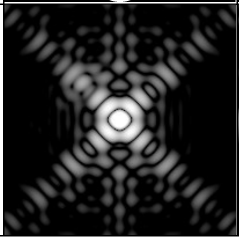
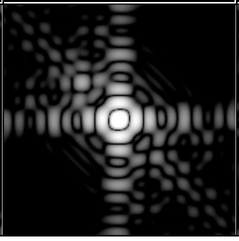
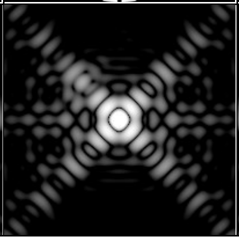
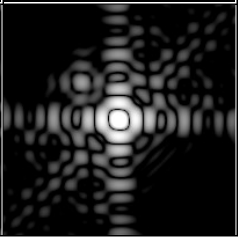
We feel some relief that our additions to the original bowtie and Gaussian donut masks show almost no impact on the theoretical diffraction patterns within our working angles of interest. At last, with the optical and mechanical properties of our selected masks on firm footing, we proceed to design a device that holds these masks.

4. MASK ROTATION MECHANISM FOR THE CELESTRON C11 OPTICAL TUBE ASSEMBLY

4.1 Need for rotation mechanism

None of our selected masks has complete axial symmetry, meaning their discovery zones cover only a fraction of the periphery of a star. To fully survey the neighborhood of a target, we must sweep high-contrast regions through the 360-degree arc by alternately rotating the mask and capturing an image. For example, a mask with two symmetric 45-degree discovery zones requires a minimum of four different positions (Table 6). That said, smaller angular intervals allow discovery zones in consecutive images to overlap, which guards against slop and miscellaneous mask placement errors. Smaller discovery zones require more discrete mask orientations.

Table 6. Effect of Gaussian donut mask rotation on a system with an apparent visual magnitude difference of 5 and a separation of 2.3 arcseconds, assuming an 11-inch telescope and 548-nanometer light. The secondary star (to the upper left of the primary) is arguably most visible when the mask is rotated at 135 degrees.

Mask angle	**0°**	**45°**	**90°**	**135°**
Mask				
Close double power spectrum				

Some advanced telescope mounts may be able to reorient masks by rotating the entire optical tube assembly about its axis, but for most telescopes and most common configurations of the C11, rotating the mask itself is the only feasible option. As for the method of rotation, one can either spin the mask manually or use a powered rotation device.

Rotating the mask by hand is tedious but adequate for some applications. For example, with prior knowledge of the placement of a secondary star relative to its primary, one can orient the mask by hand such that it emphasizes the secondary star. This works for general observation and monitoring. Manual mask rotation is financially inexpensive: the only extra equipment that must be added to the optical tube assembly are the masks themselves and a means of measuring the rotation angle. Of course, one must be careful not to impact the alignment of the telescope while placing the mask. Repeatedly rotating the mask and taking exposures also adds tedium to an astronomical observation process that already requires plenty of patience.

For the process of binary star discovery, automatic mask rotation is much preferred. Automated setups use computerized telescope mounts and triggered camera exposures, running on lists of star coordinates for many hours. The software to coordinate these actions is often versatile enough to be adapted to perform auxiliary operations in addition to its usual chores, offering us an opportunity to run mask rotation commands. In the end, the process of automated star observation and discovery looks almost the same as before, save the extra time required to rotate the mask and record additional exposures.

Electronic mask rotation requires some extra equipment. First, there must be some motorized mechanism to power the rotation, as well as some way of establishing an axis about which the mask rotates. This axis must be aligned with the telescope's optical axis through some mechanical link to the optical tube assembly. Electronics to interpret rotation commands and run the motor must also be present. All these must be implemented in a way that minimizes the impact on other behaviors of the telescope.

4.2 Mechanical design considerations

The design of a rotation mechanism for the C11 presents several challenges, the most significant of which are the potential for the mechanism to obstruct incoming light, the device's size and weight, the device's reliability and repeatability requirements, the need to avoid physically modifying the telescope to support the rotation mechanism, and the need to keep costs and manufacturing burdens small.

To preserve the delicate diffraction patterns created by our masks, we strive to avoid introducing any obstruction in front of the telescope aperture other than the mask itself. Even small additions, such as wires or structural spiders, will affect the diffraction pattern to some extent. We cannot expect these structural elements of the mechanism to rotate along with the mask, meaning some mask orientations will perform better than others in an unpredictable manner. While it may be possible to orient the optical tube assembly itself in ways that minimize the impact from any mask rotator supports, sidestepping the issue altogether is ideal. The optical isolation between mask and mechanism allows us to validate a mask's performance independently of the state of the rotator, which reduces the number of variables in our tests and makes it easier to arrive at meaningful conclusions.

In addition to our significant optics considerations, we must consider the mechanics of the device. First, the amount of weight suspended off the end of the telescope, which imparts a mechanical moment on the mount, must be minimized. Counterweights added to the opposite side of the mount typically balance the weight of apparatuses similar to ours, but this load nonetheless increases the moment of inertia, straining the motors responsible for aiming the optical tube assembly and lowering the frequencies of vibration. The effect is exacerbated if our mechanism is used in the presence of other astronomical devices, which the C11 at the Orion Observatory is.

We must keep slop reasonably small throughout the system so that we obtain consistent results at each rotation. A greater slop reduces trust in the reported mask angle. Users may respond to these ambiguous measurements by commanding additional orientations and recording additional exposures, which slows stellar observation and discovery. A loose mechanism may also center the mask off the optical axis, potentially introducing diffraction effects from exposed edges of the aperture that we intended to hide.

We must not damage the telescope we outfit. Telescopes are precise, fragile, and expensive pieces of equipment. No part of the telescope should be drilled into. The lens of the telescope must not be scratched. We should also be wary of making permanent cosmetic changes such as scuffs that might discourage an astronomer from using our solution. We would prefer not to mount to any sensitive optical elements, including the cylinder above the secondary mirror.

Finally, in the spirit of maximizing accessibility to this technology, we want to incorporate materials that are reasonably obtainable into an assembly that requires no exotic manufacturing techniques. These two objectives also tend to reduce the final cost of the system, which promises easier adoption.

4.3 Early designs

4.3.1 Virtual axis design

At first, we designed a rotation mechanism that would attach to the telescope only on its outside. To accomplish this, we would modify an off-the-shelf dew shield, which is a telescope extension normally used to reduce condensation at the surface of the lens (Astrozap, 2019). The dew shield comes with preinstalled thumb screws that give it a snug fit around the telescope's exterior. The mask would be placed inside the dew shield and suspended above the telescope objective by lipped gears added to the dew shield. Each of these gears would rotate about bearings mounted externally to the shield. These gears would center the mask—establishing a "virtual axis"—and one of the gears would be powered to drive the mask's rotation. This concept is shown in Figures 53 and 54.

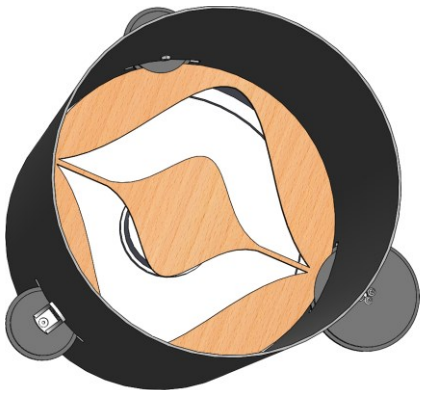


Figure 53. 3D model of the virtual axis design. Gear teeth are not included in the model for simplicity. Driving pinion (Figure 54) is at the lower right. Lower-left and top gears are unpowered guiders.

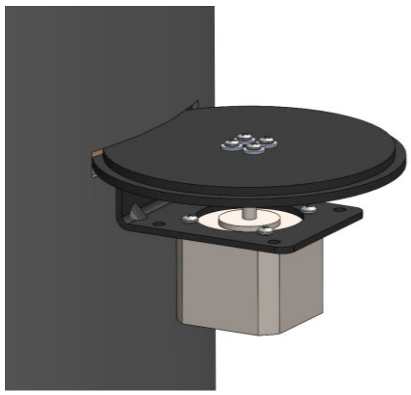


Figure 54. Detail of the stepper–pinion assembly.

The gears along the perimeter of the dew shield would be created by laser-cutting any compatible material. We selected newsboard, a thick, compressed paper-based material that was both inexpensive and readily available. The gear and the lip were cut individually and layered along the same axis, as seen in Figures 55 and 56. A plastic bushing hidden in these figures facilitates rotary motion.

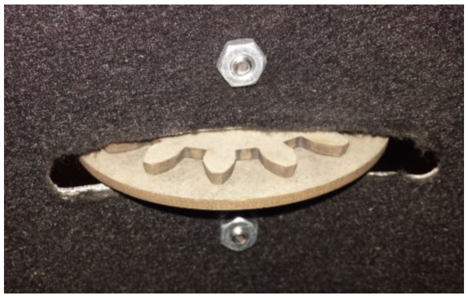


Figure 55. One of the mask-supporting lips of the virtual axis design, viewed from the interior of the dew shield.

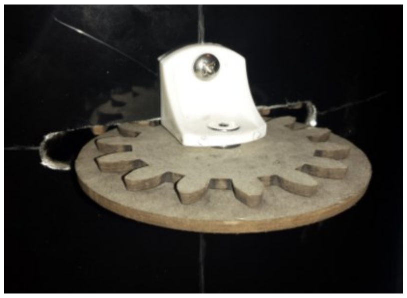


Figure 56. The method of mounting each gear from the outside. A bracket complementary to the white bracket visible here lies on the underside of the gear lip. The metal axle bridging them is surrounded by an unseen plastic bushing inside the gear–lip stack.

To achieve powered rotation, the design called for one of the outer gears to be motorized. Rotating this pinion would rotate the mask an amount determined by the gear ratio. Anticipating some friction in the system, we selected one of the stronger stepper motors that an Arduino Motor Shield could drive, which had a NEMA-17 form factor (SparkFun Electronics, Inc., 2019). An off-the-shelf motor bracket was used to attach the motor to the dew shield. The size of the motor required a pinion that was similar in design but somewhat larger than the two guiding gears, as seen in Figure 57.

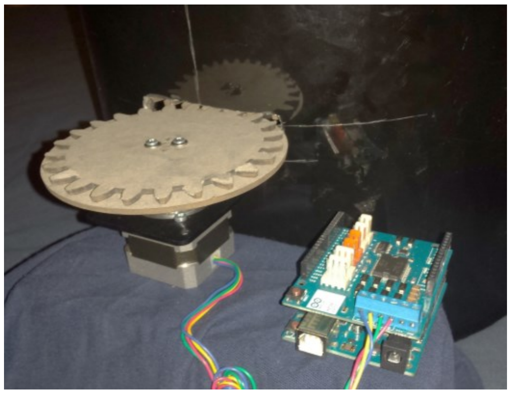


Figure 57. The size of the driving motor mandated a pinion larger than the guiding gears.

This design had many attractive properties. First, it required no attachment to the secondary mirror cylinder. This physically separated the rotation mechanism from fragile elements of the telescope, including the corrector plate. Second, it required few special parts: only the laser-cut gears, lips, and masks needed to be made custom; all other components were available off the shelf.

Sadly, the virtual axis design never worked.

The first issue with the design concerned the meshing between the teeth of the interior gear and the teeth of the exterior gears. The cross-section of the nominally cylindrical dew shield ended up being more elliptical than anticipated, causing the exterior gears mounted to it to squeeze the mask along the shield's minor axis but provide incomplete tooth engagement along the major axis. These effects would have likely contributed to mask misalignment, but this was not specifically tested. Some eccentricity in the shield was expected due to its thin, lightweight construction, but the extent of the eccentricity—visible to the naked eye—was not. This was likely caused by our using a wooden brace to keep the part sturdy while cutting gear slots with a rotary saw (Figure 58).

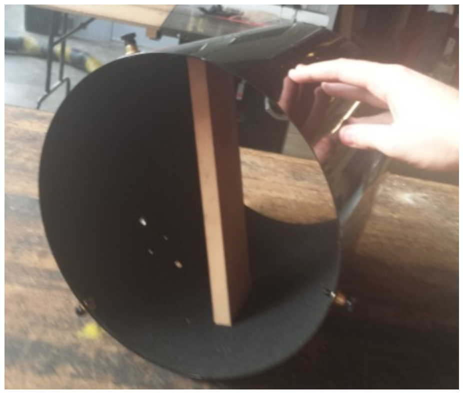


Figure 58. Our method of steadying the dew shield during manufacturing likely deformed the part.

As problematic as it was, the eccentricity of the dew shield was made moot by the more fundamental issue of using gears to position the mask. The force applied by the driven pinion tended to push the mask toward the gap formed by the pinion and the next guiding gear, causing the gear teeth to bottom out and gnash; meanwhile, the trailing guiding gear supported no lateral load. This was a major oversight that could have been solved by using a proper positioning element. One option would have been to add to the mask a newsboard layer with a smooth, circular edge and replace the guiding gears with smooth, circular lipped elements that would interface only with the new mask layer and not the gear teeth themselves. This change would use the gear teeth exclusively for power transmission, which is what they are designed for in the first place.

This flawed design was almost completely executed (Figure 59). The only major component not built was the holder for the mask rotator electronics.

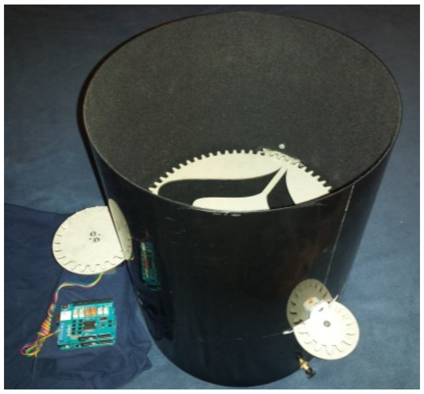


Figure 59. The virtual axis design seen with a mask. The design was abandoned before a holder for the electronics was created.

The failure of this design emphasized the importance of establishing a reliable rotation axis. It is difficult to center a mask when the positioning of the axis is subject to stacked tolerances. Of course, this rotation axis is only as useful as the device's ability to drive a mask about it: any friction or other forces that resist rotation are problematic and must be minimized. We proceeded to carry these lessons into our next design.

4.3.2 Lazy Susan design

An alternate means of rotating a mask from the outside without adding any hardware within the telescope aperture involves a large, sturdy ball bearing, the kind used in food turntables. We affectionately call this option the Lazy Susan design.

In this design, a mask would be placed such that it would rest on a circular lip attached to the interior of a large ball bearing, with the exterior of the ball bearing fixed to the exterior of the telescope. A secure mount between the bearing and the telescope would imply a well-established rotation axis, countering one of the shortcomings of the virtual axis design.

The lip at the bearing's interior would have several mounting pegs on it that would interface with holes in the mask. These pegs and holes would be arranged such that the mask could be installed in only one orientation. This would allow some indexing feature—perhaps a magnet, optical element or physical protrusion—to be a component of the holder, removing the need for such a feature to be incorporated in the mask itself. Similarly, gear teeth could be molded into the holder, reducing the complexity of the mask's geometry and making it easier to place and remove the mask.

Figure 60 shows a simple illustration of this concept to convey where the bearings would be placed on the telescope. Mounting hardware, indexing features, and electronics are not shown in this diagram.

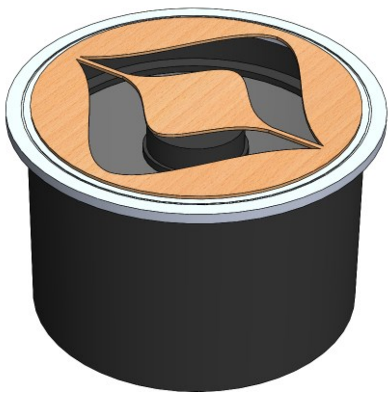


Figure 60. Solid model of Lazy Susan concept. The black body at the bottom represents the end of the C11 telescope. The white ring is the Lazy Susan bearing, which elevates the mask above the telescope aperture. Mounting hardware, indexing features, and electronics are not shown.

To help us evaluate the principle of the design, we acquired two off-the-shelf Lazy Susan bearings from an online retailer. Bearings in this family were difficult to locate, and the available sizes were limited. The two we purchased are shown in Figure

61 and described in Table 7. The smaller option, with inner diameter less than 10 inches, would have impeded the light collecting capacity of the telescope but had a more workable footprint. The other option—about two inches greater in diameter—would have avoided eclipsing the telescope optics but naturally came at the price of a larger package and greater weight.

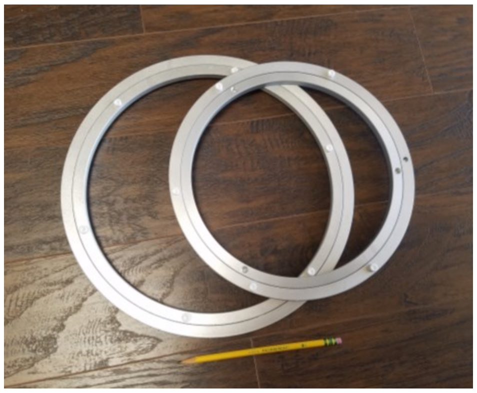


Figure 61. The bearings acquired in the process of evaluating the potential of the lazy Susan design. Pencil included for scale.

Table 7. Physical property comparison of lazy Susan bearing options.

Bearing size	**Outer diameter**	**Inner diameter**	**Weight [lbf]**
Large	13.81”	11.70”	1.59
Small	11.84”	9.72”	1.38

Immediately, we found the bearings’ friction and weight to be problematic. Though the amount of resistance when the bearing was rotating—the kinetic friction—was acceptable, the torque required to initiate motion—affected by static friction—was large and unpredictable. This friction seemed higher at some angles than others, and though it was never measured, it appeared that it would overpower the torque from the stepper motor, even accounting for the gear reduction. This friction would likely increase with time and wear, especially in an outdoor environment.

The only bearing option that would not have impeded the optics weighed 1.59 pounds, and this weight would fall at a large moment arm relative to the telescope's motors, magnifying its impact. The motor and electronics assemblies would further increase the load placed at the end of the telescope.

Though possessing some attractive qualities, the lazy Susan concept was not suitable to develop further due to these overarching concerns, not to mention unanswered questions about mounting. We abandoned the concept in favor of a centrally mounted axis option.

4.4 Final design

4.4.1 Establishing a fixed axis

Recognizing difficulties of mounting only to the outside of the telescope, we expanded our design window to include options where a component mounts to the secondary mirror cylinder that forms the central obstruction of a Schmidt–Cassegrain telescope's aperture (recall Figure 28). This mounting point is mechanically ideal because it is aligned with the telescope's optical axis—and thus our desired rotation axis—with a very high degree of precision. The tradeoff is that this component is also responsible for the telescope's delicate collimation calibration. As the cylinder represents a sensitive telescope component, mounting to it conflicts slightly with one of our design goals.

The secondary mirror cylinder is not the only path to a fixed axis: it would also be possible to install structural "spokes" extending from the circular perimeter of the telescope to the circle's center. While this would be a lightweight and inexpensive solution, the support structure would introduce a diffraction pattern that could reduce the effectiveness of the masks. Figures 62 and 63 show how spokes can affect the baseline diffraction pattern of the C11's aperture.

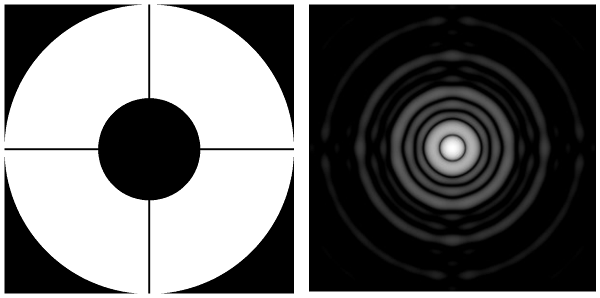


Figure 62. The aperture shape of a C11 outfitted with a four-armed 1/16-inch spider, left, and the shape's point spread function, right. Unsurprisingly, the pattern is not axisymmetric.

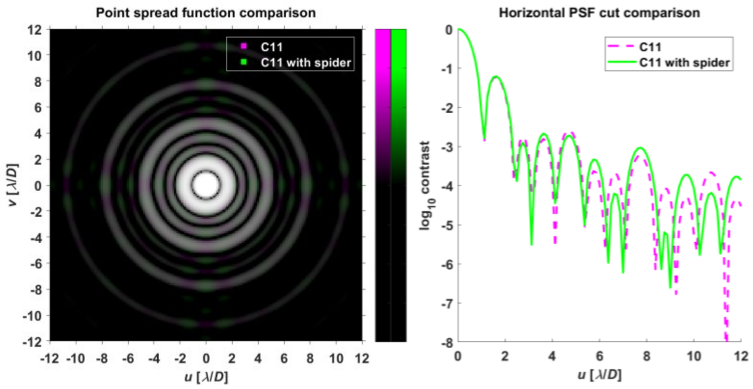


Figure 63. The spider's subtle effect on the C11's point spread function is visible in these overlay plots.

This structure would be difficult to rotate with the mask. If the spider were fixed to the telescope, then the convolution of the diffraction pattern of the spokes with the diffraction pattern of the mask would be different at different mask angles. We would much prefer the mask to cast a diffraction pattern clear of interference from the support structure.

Given these difficulties, we decide to mount an axis directly to the secondary mirror cylinder, even though this location is not as robust as we would like.

4.4.2 Strategies for powering rotation

Assuming we are able to create a central mount that supports the mask, the next order of business is the mechanism to induce rotation. One option would be to mount a motor directly atop the secondary mirror, operating the mask in a “direct drive” manner. This would be a simple solution and would minimize equipment. Unfortunately, energizing the motor is a concern. First, the motor can produce heat that might degrade the quality of the image. Second, power would either need to be delivered to the motor by batteries placed within a small footprint at the center of the aperture, or wires would need to be laid radially across the telescope aperture. Placing wires this way would lead to similar diffraction issues as a spider support structure. A clever approach would involve routing wires along the structurally connected underside of the mask itself such that they are never exposed to incoming light (Figure 64), but this could lead to unpredictable winding of the wire at the telescope’s exterior. Prioritizing simplicity, we instead decide to drive the mask from the outside of the telescope, where we have much greater flexibility to determine wire routing.

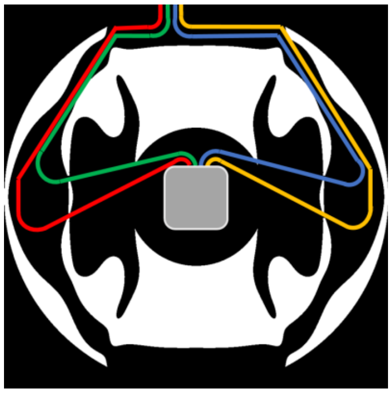


Figure 64. Routing wires along the underside of the mask is one way of delivering electricity to the center of the telescope without affecting diffraction patterns. This figure shows how the wires might be routed on the bowtie mask. The gray body at the center of the figure represents the stepper motor.

Pulleys or gears can be used to transmit rotation, but pulleys generally require a belt in tension that would place an undesirable lateral load on our axle. Thus, we opt for a geared solution. To minimize the amount of equipment, we make the mask itself a gear in much the same way we did in the virtual axis design. Gears are normally difficult to manufacture, but a laser cutter drastically reduces the effort relative to traditional machining techniques. Also like the virtual axis design, we can cut teeth into the aperture mask at the same time we cut the contours of the mask's openings. We can also laser-cut a pinion to mount to the motor shaft. Unlike the virtual axis design, however, the mask will be mounted to a physical axis rather than located by guiding gears, improving our alignment.

A stepper motor is a natural choice for the motor due to the small form factor and strong holding torque. Unless a stepper motor stalls or is forced past its holding torque, the shaft's relative angular position can be determined by the number of times the motor has been stepped forward electrically. Tracking motion by counting actuation steps in software effectively removes the need for an incremental encoder, though we still need some sense of absolute mask position to reestablish an orientation across power cycles or after a stall. This can be accomplished without an expensive absolute encoder by incorporating an indexing feature into our masks—some fiducial element that identifies a zero position. The fiducial can be a distinctive slit or mark captured by an optical sensor, or perhaps a magnet whose field triggers a Hall effect sensor, or maybe a physical protrusion that activates a limit switch. However it is implemented, the index provides our step counter context to determine the absolute position of the mask. We lean toward the magnet option for its forgiving alignment tolerances.

With our overall vision in place, we proceed to design the central mount, which we will call the axle cap assembly; the externally mounted motor holder, which we will call the motor bracket assembly; the gears; and the electronics assembly.

4.4.3 Axle cap assembly

The axle cap assembly, modeled in Figure 65, sits atop the cylinder that houses the C11's collimation screws. A countersink on the inside of the axle cap accommodates a central "axis bolt" whose flat head sits flush with the material around it. The axis bolt is secured using two consecutive fully threaded female standoffs with different diameters that act as a stepped shaft. A hole at the mask's center fits around the narrow part of the shaft, allowing the mask to sit on the upper edge of the larger standoff. A plastic thumb nut that screws to the top of the axis bolt prevents the mask from escaping the shaft (Figure 65). The thumb nut does not contact the rotating elements directly—it rests securely at the top of the stepped shaft with a small gap between it and the top of the mask. This makes unintentional disassembly during operation unlikely. The nut is plastic to minimize the impact of accidentally dropping it toward the telescope objective.

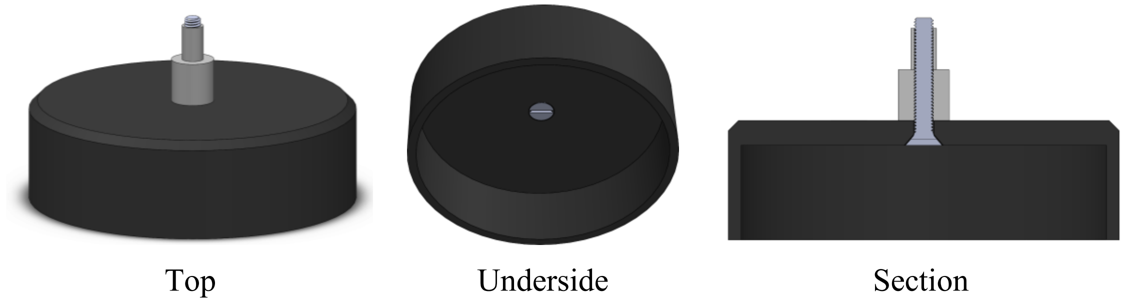


Figure 65. Solid model of the axle cap assembly, shown without the mask or thumb nut.

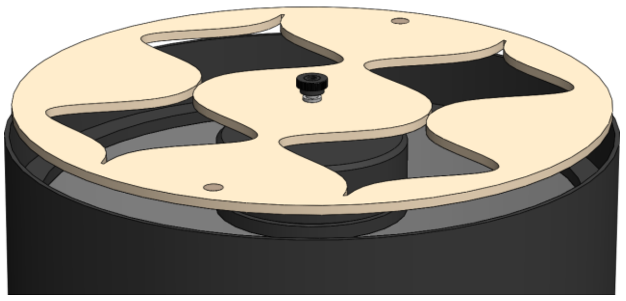


Figure 66. Solid model of the axle cap on a C11, shown with a simplified mask and the securing thumb nut.

To replace a mask, the user will unscrew the thumb nut, lift the old mask, place down the new mask, and reattach the thumb nut. In practice, gravity should secure the mask in a stable position and remove the need for the thumb nut, but unpredictable factors like wind in combination with use of the telescope at shallow altitude angles may challenge this assumption.

The axle cap itself is formed from acetal, a thermoplastic known for its excellent machinability. The axle cap is a fairly large component—approximately 3.8 inches in diameter—so the material selection has a significant effect on the cap's weight. Acetal's density is about 1.41 grams per cubic centimeter, which cuts 48 percent of the mass of an aluminum equivalent and 82 percent of a steel equivalent for a part of the same dimensions (MatWeb a–c). The reduced mass is especially relevant because of the cap's placement at the end of the telescope, where a large moment arm exists relative to the telescope mount's axis. Acetal is also recognized for its moisture resistance (W.S. Hampshire, Inc., 2019), so it should be suitable to leave on a telescope for multiple hours at a time—perfect for a fruitful night of stellar observation and discovery.[23]

The cap is designed with a slight clearance fit around the collimation cylinder that is large enough to facilitate easy installation and removal but tight enough to guard against mask misalignment and any substantial angling of the axle during use. The depth of the cap is just shallow enough to keep the cap's rim from contacting a step in the secondary mirror cylinder, providing good stability. The cap's outer diameter is less than the full width of the secondary mirror obstruction, and the difference in diameters is enough such that, even amid worst-case machining tolerances, the cap will not affect the optics. The depth clearance and diameter difference can be seen in Figure 67.

[23] Acetal is typically not recommended for use outdoors, but this stems from its susceptibility to degradation in the presence of ultraviolet light (Zeus Industrial Products, Inc., 2005). As our axle cap will not be used in sunlight, this is not a concern.

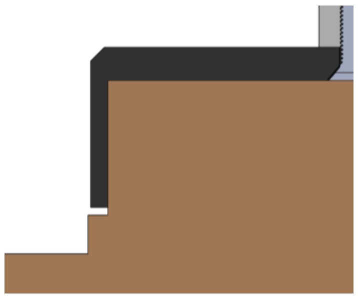


Figure 67. Section view of the axle cap's interface with the telescope. The axle cap is black; the telescope is brown. The depth of the cap is just less than the height of the upper part of the secondary mirror cylinder; meanwhile, the outer diameter of the cap is just shy of the diameter of the lower step of the secondary mirror cylinder.

The 10-32 axle bolt is significantly stronger than it needs to be given the minimal stresses in our system. We selected it based on the wide availability of standoffs with 10-32 female thread and because there were more length options for flat-head screws of this thread size relative to screws with smaller threads.

The bill of materials for the axle cap assembly is available in Appendix N. Appendix O contains engineering drawings of the axle cap and its related assembly.

4.4.4 Motor bracket and pinion assemblies

The motor bracket assembly, which sits on the outside of the telescope, requires three essential elements: a motor, a mount, and a pinion.

For the motor, we select the same motor that we used in the virtual axis design, a bipolar stepper motor with a NEMA-17 mounting pattern. The motor's 1.8-degree step angle implies 200 steps per revolution (SparkFun Electronics, Inc., 2019), which is plenty of granularity given that our masks' azimuthal discovery zones are no smaller than about 42 degrees each and given that we benefit from a gear reduction between the motor-mounted pinion and the mask itself. The motor's holding torque of 0.23 newton-meters

(SparkFun Electronics, Inc., 2019) is enough for the small loads we drive.[24] The electronics interface is a standard four-wire setup that is directly compatible with an Arduino Uno Motor Shield, as are the 12-volt voltage and 0.33-ampere current (SparkFun Electronics, Inc., 2019). At 0.44 pounds (SparkFun Electronics, Inc., 2019), the motor is not a light component, but we figure its weight is comparable to other stepper motors of similar specifications and should not give us pause. A DC gearmotor would be a lighter option, but it would likely demand an auxiliary encoder and a more sophisticated controller.

The mounting of the motor is a challenging design task due to the limited placement options. We dare not drill into the telescope to form a mounting point; instead, we must make use of the existing metal collar surrounding the C11's aperture. We envision a three-point mounting system with one contact point on the inside of the collar and two contact points spread apart on the outside of the collar (Figure 68). If the inner-side contact point is made to be a thumb screw, then tightening this screw will cause the mount to clamp. A platform connected to these mounting points that lies parallel to the telescope lens but free of the optics forms the plane the motor mounts to. This principle can be seen in the model of our motor bracket assembly, shown in Figures 69 and 70.

[24] A conservative analysis in Appendix D shows that the required torque is no greater than about 0.21 newton-meters.

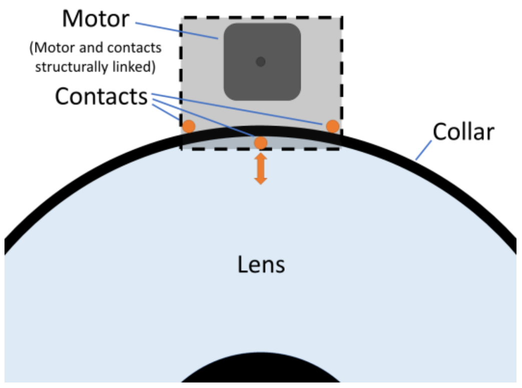


Figure 68. Illustration of the three-point mounting concept. The inner contact is adjustable to allow a tight fit on the collar of the telescope regardless of the telescope's diameter.

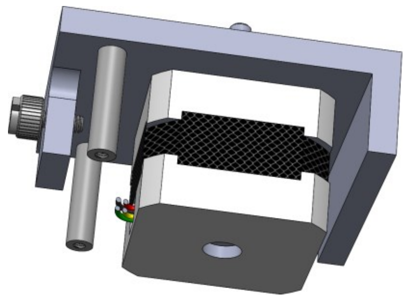


Figure 69. Model of the underside of the motor bracket assembly, showing at the left side the adjustable thumb screw and the two stationary mounting posts.

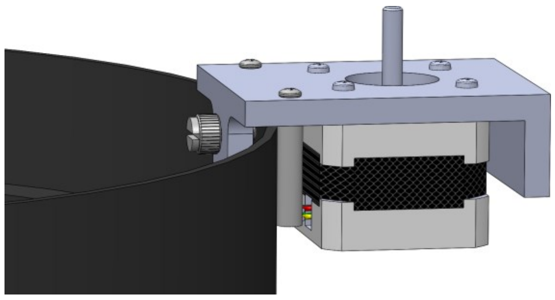


Figure 70. Model of the motor bracket assembly mounted to the collar of a C11 telescope.

The thumb screw allows the design to adapt to telescopes of different diameters easily. On a smaller telescope, the screw can be "tightened" further so it and the two stationary posts align to the tighter curvature of the rim. Inversely, on larger telescopes, the screw sits shallower in its threads. The thumb screw's length determines the smallest compatible telescope, though the motor bracket may impinge on the optical path for especially small cylinder diameters. There is no equivalent upper diameter limit.

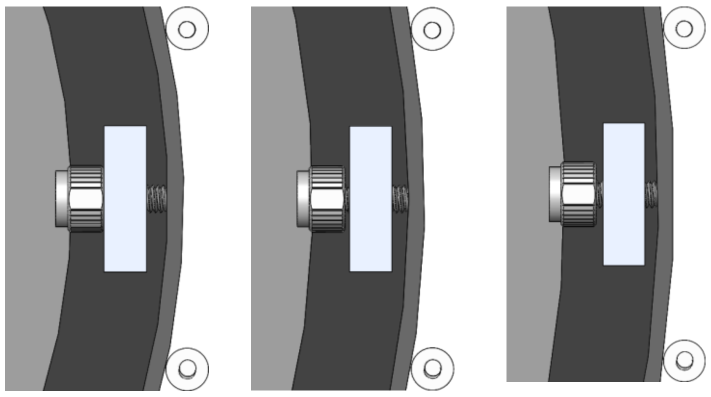


Figure 71. Cross-sections, looking toward the telescope aperture, of the motor bracket's interface with telescopes of different sizes. (Swapping the thumb screw for a longer alternative would be recommended for the 8-inch telescope so that it does not bottom out.)

Like the axle cap's thumb nut, the motor bracket's thumb screw is plastic to reduce potential damage to the telescope if it is accidentally dropped.

The shape of the motor bracket component is a mix of function and material choice. The part's essential purpose is to support a mounted motor and provide a right angle between the motor mount and the surface through which the thumb screw hole is tapped. We found that aluminum U-channel stock could be machined to support both these functions at a lower cost than some alternative methods. The vertical ridge placed away from the telescope at the right sides of Figures 69 and 70 is an artifact of the stock material. We make use of it in the electronics assembly (Section 4.4.8).

The stepper motor comes with a mounting hub, which is a natural place to attach a pinion. The pinion assembly design we arrived at, modeled in Figures 72 and 73, uses three laser-cut pieces of acrylic. Two 3/32-inch-thick, clear, circular pieces of different sizes sandwich a 3/16-inch-thick gear. To keep the pinion close to the elevation of the telescope edge, we attach the pinion beneath the hub, forming a mushroom-like shape. A small set screw pinches the hub to the stepper motor shaft. A model of the pinion assembly attached to the motor bracket assembly is shown in Figure 74.

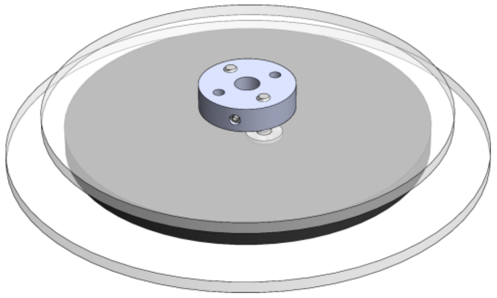


Figure 72. A model of the pinion assembly. The black cylinder represents a simplified gear.

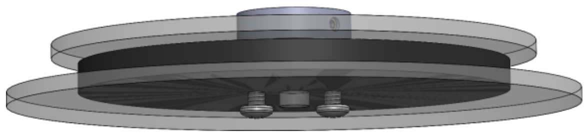


Figure 73. A view of the pinion assembly model from a lower angle, showing the screw heads.

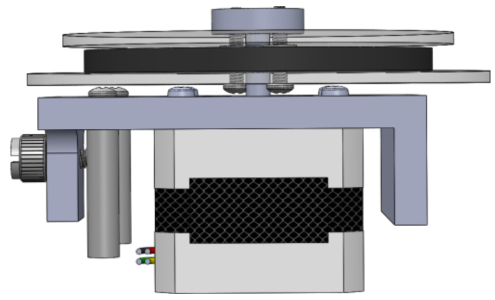


Figure 74. A model of the pinion assembly placed on the motor shaft. The narrow clearance between parts is apparent.

The two pinion lips are present to gently maintain the vertical position of the edge of the mask during operation. Even though the mask is supported at its center by the axle cap standoff, the mask is still liable to wobble slightly, and the lips help correct this motion. The lower lip is sized large enough to support parts of the mask within the mask gear's root circle. The upper lip is smaller than the lower one to facilitate mask removal and reinsertion, but it still prevents errant mask excursions because it enters the mask gear's addendum circle. These lips are made of transparent acrylic to provide better visual feedback when inserting the mask into the rotation mechanism.

We select the thickness of the pinion gear itself to be 3/16 of an inch because it is the largest that can be reasonably accommodated on the shaft and because materials thicker than this would make very heavy masks. To account for the case in which a material 3/16 of an inch thick is selected for a mask, we add two washers that act as spacers within the pinion assembly. These are visible in Figure 75.

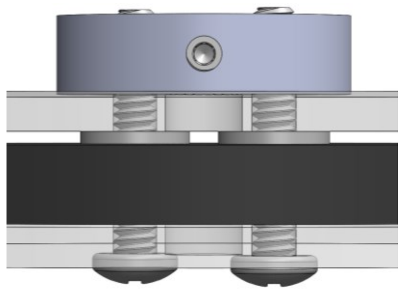


Figure 75. Detailed profile of the pinion assembly's central column, exhibiting the humble spacers and set screw.

Bills of materials for the motor bracket and pinion assemblies can be found in Appendix N. Engineering drawings related to these assemblies are in Appendix O.

4.4.5 Gear design

At this point, we have established the need for two gears—a pinion and the mask itself—but we have not fully defined their geometry. The most important remaining variables are the number of teeth on each gear and their shared circular pitch. These variables dictate the pitch diameter of each gear and the separation between them, among other dependent variables.

It helps us to begin with some basic guidelines and use them to narrow our design window. First, under no circumstance should the mask's root diameter be smaller than the telescope aperture's diameter, lest we leak light through the gaps between gear teeth. Second, the teeth must be large enough that they will engage amid clearance in the system that might result from manufacturing tolerances. Third, the gear axes should be close to each other, within reason, to minimize material and weight at the motor bracket.

After some iteration, we arrived at a design with 72 teeth at the mask and 17 teeth at the pinion. The circular pitch ended up at 0.5236 inches per tooth, which produces a mask pitch diameter of 12 inches, a pinion diameter of 2.83 inches, and a center distance of 7.42 inches. These dimensions and more are summarized in Table 8. Figure 76 displays the result. Importantly, the root diameter of the gear comfortably exceeds our 11-inch minimum (Figure 77).

Table 8. Parameters of the mask and pinion gears.

Parameter	**Gear**	**Pinion**	**Units**
Number of teeth	72	17	teeth
Outer diameter	12.333	3.166	in.
Pitch diameter	12.000	2.833	in.
Root diameter	11.583	2.416	in.
Circular pitch	0.5236		in./tooth
Pressure angle	20		deg.
Clearance	0.056		in.
Backlash	0.02		in.

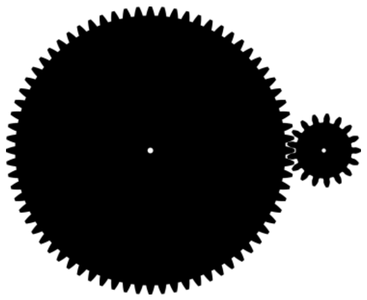


Figure 76. The 72-tooth gear and 17-tooth pinion displayed to scale. Figure was created using Rainer Hessmer's Involute Spur Gear Builder utility (Hessmer, 2014).

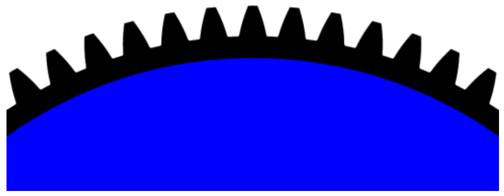


Figure 77. Mask gear image overlaid with an 11-inch-diameter circle, demonstrating the clearance between the C11 aperture and the gear's root diameter.

The choice of 72 teeth for the mask is particularly convenient. Each tooth represents exactly five degrees of rotation, allowing masks with two-way, three-way, four-way or six-way symmetry—and even those with some higher-order patterns—to have their symmetry reflected in the tooth pattern itself. The mask's teeth will mesh with the pinion the same way at every rotationally equivalent angle at which the mask's opening can be oriented. This implies that the backlash behavior at all such rotations is the same, which may lead to slightly more consistent imaging results than a configuration in which teeth do not reflect the same rotational symmetry as the mask opening.

Another small benefit of meshing a 72-tooth gear with a 17-tooth pinion is that wear patterns caused by a defect on a gear will be evenly distributed among all points on the opposite gear. This occurs because the greatest common factor between 72 and 17 is 1, as shown in Table 9. This property would be much more important to high-speed, high-power geared systems, but we nonetheless accept the modest benefit.

Table 9. Gear teeth values and factors.

Element	**Teeth**	**Factors**
Gear (mask)	72	1, 2, 3, 4, 6, 8, 9, 12, 18, 24, 36, 72
Pinion	17	1, 17

We included a small clearance of 0.056 inches between the gears to account for possible engineering tolerance stack-up within the axle cap and motor bracket assemblies. This dimension is derived in Appendix E. With less rigor, we set the backlash between the two gears to be 0.02 inches, amounting to an angular backlash of about 0.19 degrees at the gear—plenty precise for our masks, whose discovery zone sectors are orders of magnitude wider. The clearance and backlash are visible in Figure 78.

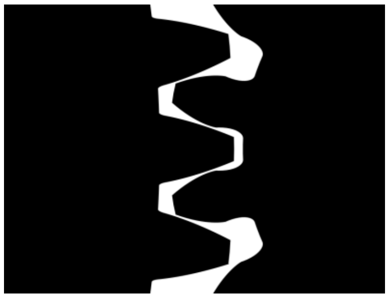


Figure 78. A close-up of the interface between the mask gear, left, and pinion gear, right, revealing the small clearance between them. Figure was created using Rainer Hessmer's Involute Spur Gear Builder utility (Hessmer, 2014).

The process of producing gears with these dimensions is described in Section 4.5.2.

4.4.6 Mask assembly

The hard part of our mask designs is over: we selected the shapes of our openings in Section 3.5 and defined gear teeth to add our masks in Section 4.4.5. Yet unspecified is the masks' material, for which we select 1/8-inch-thick birch. Wood is a good choice because it is light and sturdy. It can be used in a laser cutter—likely the only machine that is precise enough to cut our masks properly—and it is opaque to all light. Its matte finish prevents stray light from contaminating the image. Working against it is its tendency to absorb moisture, which can affect its dimensions (Ross, 2010). We opt not to use the same newsboard material that we used in the virtual axis design because of the excess of ash it creates during the laser cutting process. Naturally, wood also creates ash, but the amount is not as alarming.

Initially, we considered using black acrylic because of acrylic's excellent laser-cutting characteristics, but a mask of this material would not have performed well in the field. Even though the material is visually black, it does not block infrared light (Figure 79). This would cause light at only some wavelengths to diffract as designed, ruining any exposure recorded with an image sensor that detects wavelengths outside this range.

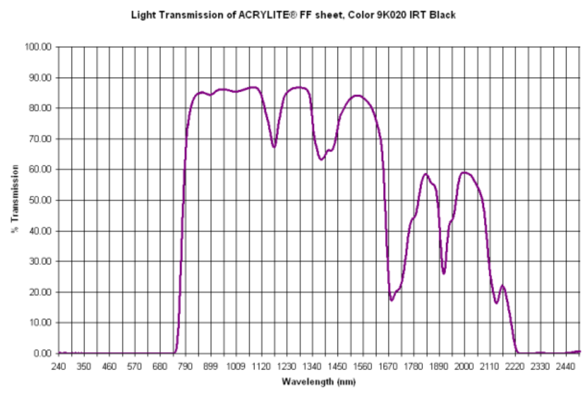


Figure 79. Light transmission of black acrylic sheet (Acrylite). Even though the sheet is visually black, it transmits considerable light at infrared wavelengths.

We establish index positions in each mask by adding magnets toward the outside of the mask at azimuthal angles where the mask opening is rotationally symmetric relative to some base orientation. All our masks have two-way rotational symmetry, so each mask requires two magnets in diametric opposition across the mask, as shown in Figure 80.

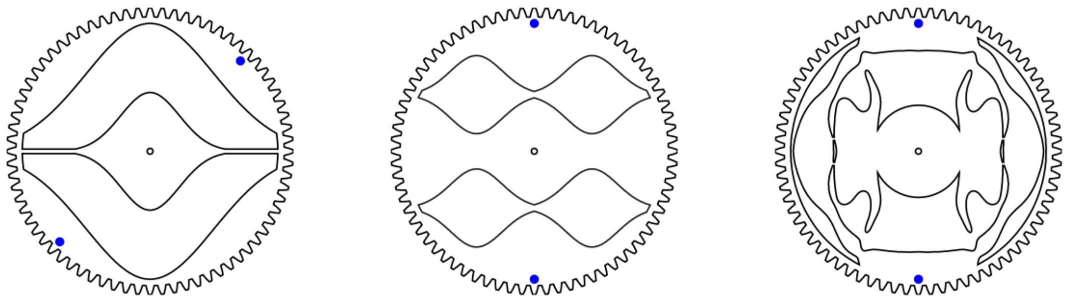


Figure 80. Magnet positions vary by mask based on available space. Magnets are shown as blue circles.

The exact placement of the magnets relative to the opening is unimportant so long as the magnets are evenly spaced across the full 360-degree span in a way that reflects the opening's symmetry. We use this flexibility to solve a geometrical challenge with the Gaussian donut mask: unlike the bowtie and multi-Gaussian masks, the Gaussian donut cannot accommodate magnets placed along the nominal vertical axis of the mask, so we instead place them at an angle 45 degrees to the vertical.

The selected magnets have a diameter of 3/8 of an inch, a thickness of 1/16 of an inch, and are neodymium. They sit in shallow magnet cubbies etched into the mask and are attached to the mask with super glue. The magnets are placed in a circle with a diameter of 11 inches, which is the widest possible dimension that avoids encroaching upon gear tooth geometry. At this location, the magnets slightly enter the aperture diameter, but because they would already sit atop an opaque part of the mask and because the opaque areas in our three masks are large, this is not an issue for us. Alternate mask designs that are especially open toward their outsides may not be as forgiving.

The poles of the magnets should be aligned the same way for the benefit of the Hall effect sensor. Doing this conveniently allows the self-aligning nature of magnets to keep stacks of masks together neatly when placed in storage, as seen in Figure 81.

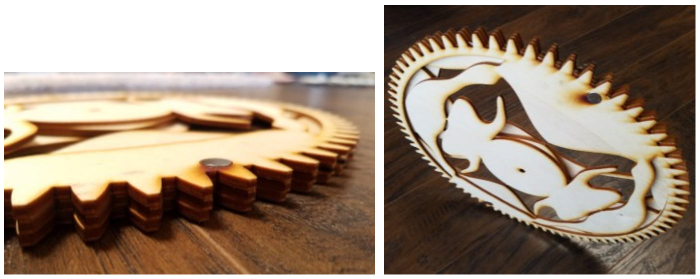


Figure 81. Magnets gently hold the masks in a stack when the masks are not in use.

Mask manufacturing is discussed in Section 4.5.2, with part and assembly drawings available in Appendix O. Appendix N contains a list of materials used in the mask assembly.

4.4.7 Microcontroller selection

We select a combination of an Arduino Uno and an Arduino Motor Shield to perform our rotation logic. Arduino is an open-source electronics prototyping platform that breaks out features of a microcontroller into convenient physical interfaces. Users write and upload firmware using the free Arduino integrated development environment (IDE). The Arduino series of devices enjoys wide support from hobbyists in part because it eliminates major hurdles in designing, building, and programming circuits. The availability of boards and support for them make the platform very accessible to those without a background in electronics or software development. An Arduino Uno and an Arduino Motor Shield are shown in Figures 82 and 83.

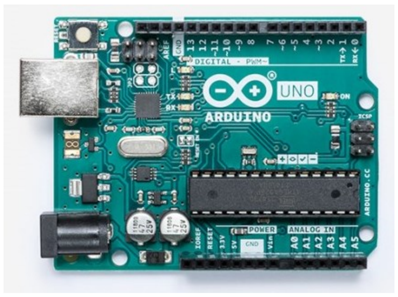


Figure 82. An Arduino Uno (Arduino AG, 2019).

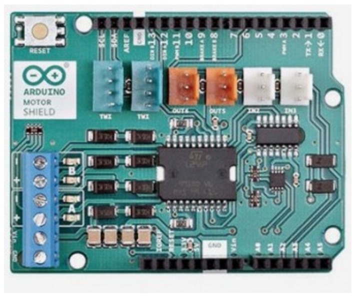


Figure 83. An Arduino Motor Shield (Arduino AG, 2019).

The Uno runs on the Atmel ATmega328P, an 8-bit processor which runs at 16 megahertz (Atmel Corporation, 2015). Given our meager input/output requirements, and in the absence of foreseeable demands in computational speed or complexity, this processor should be fine for our needs. The Uno exposes a USB-B connection that allows communication with a computer, allowing the chip to be programmed easily through the Arduino IDE.

The Motor Shield is based on the STMicroelectronics L298, a dual full-bridge driver that can be operated through Arduino electronics at up to 12 volts and 4 amperes (Arduino AG, 2019). The full capacity of the chip itself is much higher (STMicroelectronics, 2000), but the Arduino-limited capacity is plenty to drive a strong stepper motor like the 12-volt, 0.33-ampere bipolar stepper motor we identified in Section 4.4.4. The Motor Shield fits directly atop the Uno.

At roughly 3 inches by 2.5 inches by 1.25 inches, the physical footprint of the Uno and the Motor Shield stack is far from compact (Figure 84). This size comes in part from the amount of onboard electronics irrelevant to this project. Certainly, a proprietary embedded system solution would be more space-efficient than the bulky Arduino hardware, but we will leave this optimization for a follow-up project. In the meantime, we prioritize function over form.

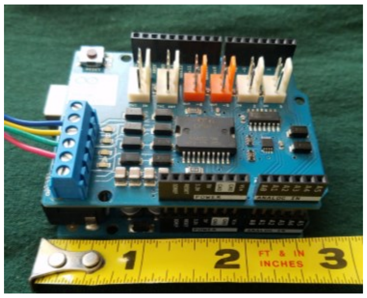


Figure 84. The Arduino boards are rather large.

4.4.8 Electronics assembly

The ridge of the motor bracket farther away from the telescope aperture (seen at the right sides of Figures 69 and 70) provides a convenient mounting surface for our electronics. First, we acquire a plastic case that holds the Arduino Uno and the Motor Shield; second, we affix adhesive-backed hook-and-loop to this case and the motor bracket lip (Figure 85); third, we press the two together to keep them in place. This arrangement is shown in Figures 86 and 87. Though the hook-and-loop design may not qualify as elegant, it is inexpensive and otherwise suits our needs.

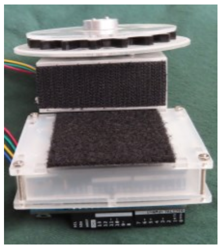


Figure 85. Detail of hook-and-loop placement.

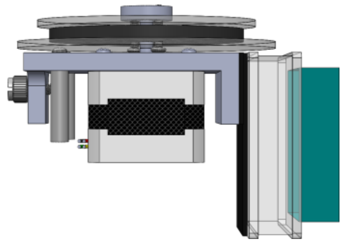


Figure 86. Solid model of the motor bracket assembly and an approximation of the electronics case joined via hook-and-loop.

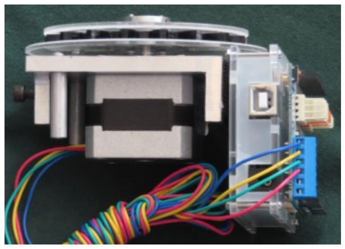


Figure 87. Realization of the attached motor bracket assembly and electronics case. The Arduino Uno lies within the case; the Motor Shield protrudes.

We must incorporate a Hall effect sensor to register the magnets placed in each mask. For convenience, we use an off-the-shelf Hall effect switch breakout board (Figure 88) and simply glue it along the side of the motor bracket beside the thumb screw (Figure 89). This breakout board position places the Hall effect sensor beneath the circular path of the indexing magnets, which is important for acquiring reliable sensor readings and thus reliable index positions (Figure 90). It is vital that soldered connections on the back side of the board are electrically insulated from the aluminum part to avoid a short circuit.

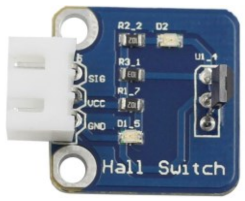


Figure 88. A Hall effect sensor breakout board (SunFounder, 2017). The sensor itself is the black component at the right side of the image.

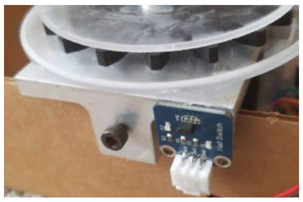


Figure 89. A Hall effect breakout board attached to the side of the motor bracket.

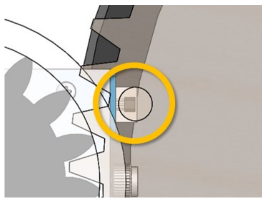


Figure 90. Top view of the full assembly with the mask made transparent to demonstrate the alignment between the indexing magnet and the Hall effect sensor. These components are highlighted by the yellow circle.

The Hall effect sensor establishes the zero position of the mask and is rigidly attached to the motor bracket, so the location of the motor bracket determines the mask datum relative to the telescope. Typically, an astronomer will want to maintain this position between observing sessions such that reported mask rotations are consistent night to night. The telescope offers few landmarks, so the astronomer might add a piece of masking tape to the outside of the telescope to mark a datum. While this design is not the most robust, it gets the job done.

Bills of materials for the electronics assembly can be found in Appendix N, with assembly drawings in Appendix O.

4.4.9 Cost analysis

A comprehensive bill of materials for the mask rotator with costs included can be found in Appendix N.

At quantity, the total cost of parts for the mask rotator is approximately US$111. Electric components account for nearly three-quarters of the total, with the Arduino Uno at $22, the Arduino Motor Shield $22, the stepper motor $16, the power supply $8, the Hall effect sensor breakout $7, and the USB cable another $6. One of the easiest ways to reduce the part cost at scale would be to replace the Arduino devices with a custom printed circuit board. Many of the peripherals present on Arduino boards are not used in this project and only run up the size and price of our solution.

There are five different raw materials used to manufacture the mask rotator: stock acetal for the axle cap, stock aluminum U-bar for the motor bracket, black acrylic for the pinion, clear acrylic for the pinion lips, and birch for the masks themselves. The minimum quantities sold by vendors are excessive for one mask rotator unit, but when the materials are used for multiple units, the effective cost per unit drops significantly.

We estimate machining to take approximately five hours, of which two are spent on the axle cap, two are spent on the motor bracket, and one is spent laser-cutting the masks and pinion components. If a machinist is paid \$20/hour to perform these operations, the total cost of the rotation mechanism increases by \$100 to \$211. In the future, machining costs can be reduced somewhat by simplifying the motor bracket part and by using an online-based commercial laser-cutting service.

4.4.10 Mechanical design summary

Figure 91 displays a model of the entire mask rotation mechanism assembly.

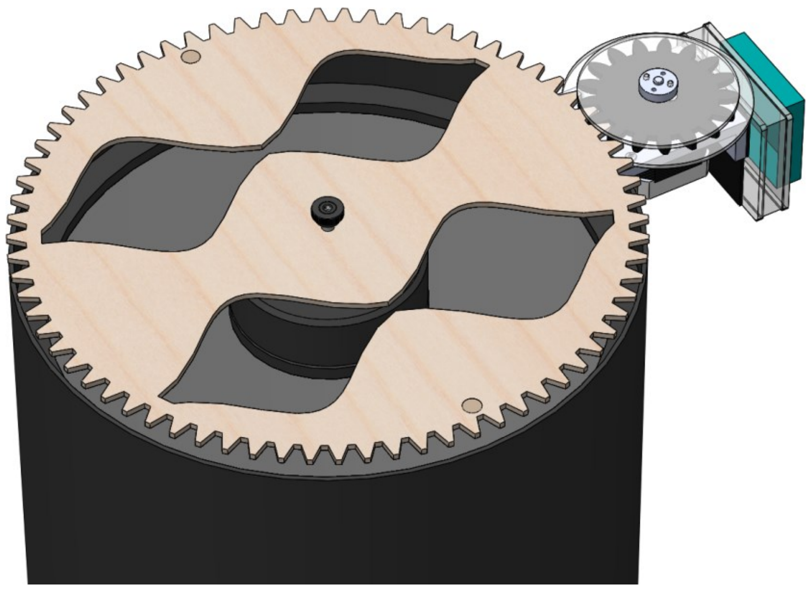


Figure 91. A solid model of the complete mask rotation mechanism, excluding wires for clarity.

We set several goals in Section 4.2 that are worthy of revisiting now that we have a complete mechanical design.

First, with respect to the purity of the optics, we succeeded at designing a rotation mechanism that does not enter the space forward of the C11's telescope aperture. To do this, we had to mount the axle cap to the secondary mirror cylinder and the motor bracket assembly to the collar of the telescope. Only the mask can intercept parallel light rays that would otherwise strike the telescope aperture, as intended.

Second, with respect to minimizing the size and weight of the device, we arrived at a mixed bag. The overall size of the device, in the author's opinion, is acceptable, though the Arduino devices are large and placed awkwardly on the outside of the motor bracket assembly. The weight of the device is inflated somewhat due to the use of metal components that were overkill for their function, one example being the steel standoffs on the axle cap. The form factor and weight can be improved somewhat by switching from Arduino to a more compact electronics solution and by substituting a lighter component for the makeshift stepped shaft at the axle cap assembly.

Third, with respect to designing a system that is reliable and repeatable, we made careful efforts to minimize engineering tolerances and provide a consistent method of achieving desired mask positions. The components of our system mate with existing telescope features and off-the-shelf parts that are machined to high levels of precision. As shown in Appendix E, the worst-case tolerance stack-up causes a gear mesh variance of 0.056 inches. This distance is much shorter than the length of the gear teeth. Meanwhile, our automatic indexing function provides users a convenient and repeatable method for recovering mask positions, even if the device stalls or loses power.

Fourth, with respect to avoiding interfacing with sensitive telescope components, we did not entirely succeed. Whereas we were hoping to avoid mounting to the secondary mirror cylinder, difficulties with other designs pushed us in this direction. We rejected the notion of using spiders in Section 4.4.1, but an engineer who continues this project may reconsider this decision.

Finally, with respect to the accessibly of reproducing the design, we largely succeeded. The only custom parts required are the axle cap, the motor bracket, and the laser-cut components (mask, pinion, and pinion lips), which can all be produced using common machine shop tools. As documented in Section 4.4.9, we estimate the total hardware cost at about $111 and expect it to take about five hours to manufacture. The part count is high due to fasteners at the motor bracket assembly. Future designs can surely simplify the device.

A complete list of materials for the mechanism can be found in Appendix N. High-level assembly drawings are featured in Appendix O.

4.5 Fabrication

4.5.1 Rotation mechanism fabrication

We turned to Cal Poly mechanical engineer Kevin Jantz to assist us with machining the motor bracket and axle cap parts.

The motor bracket begins as a piece of aluminum U-channel stock. We selected part 9001K1 from McMaster–Carr's catalog because the cross-section has square interior corners. These are important for us near the threaded tab in the finished part: a square corner lets the part lie flush with the top of the telescope collar rather than be propped up by a fillet.

Forming the threaded tab is the most difficult operation. It requires milling one flange of the stock such that the machined surface matches the existing surface on the underside of the channel as closely as possible. This operation creates an obvious contrast in finish between the machined and unmachined planes (Figure 92), but we can easily tolerate this imperfect aesthetic.[25] Other machining steps in the part are trivial, such as cutting the stock to length, drilling holes for the various mounts, tapping the thumb screw hole, and breaking the edges. Figure 93 shows the motor bracket just after being machined.

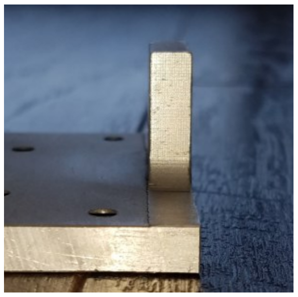


Figure 92. Profile of threaded lip, showing the square corner and change in surface finish.

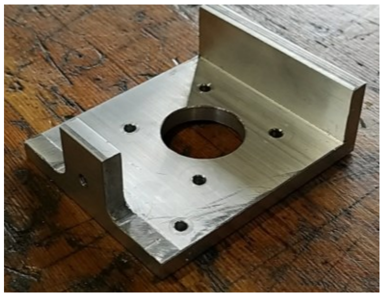


Figure 93. A motor bracket is born! (Photo by Kevin Jantz.)

The axle cap is a fairly simple part. It begins as 4-inch diameter acetal, which can be trimmed to size and hollowed out with a lathe. From here, drilling and countersinking are all that are needed to finish the part. A tailstock equipped with appropriate drill bits

[25] Let us not forget that this device is used at night.

can be used for these hole operations if available. Figure 94 shows the completed axle cap.

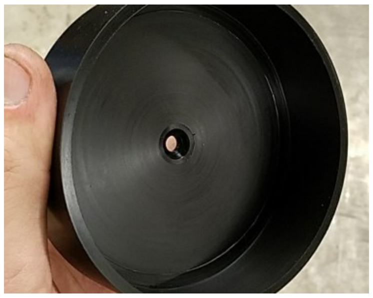


Figure 94. A newly machined axle cap. (Photo by Kevin Jantz.)

Technical drawings for these parts are available in Appendix O.

4.5.2 Mask and pinion fabrication

A laser cutter is the most logical manufacturing method for the mask and the pinion components because these parts are flat and have complicated profiles. It operates as a special type of plotter, interpreting thin lines in the graphics file as paths to cut and broader regions as areas to engrave.

Each mask has two important sources of cutting paths that need to be combined. The first is the gear tooth profile, which we generate using Rainer Hessmer's Involute Spur Gear Builder utility (Appendix F); the second is the profile of the mask's openings, which needs to be interpreted from a black-and-white image of the mask shape (Appendix G). With some effort, we combine these paths in Adobe Illustrator and set their stroke size small enough that they register to the laser cutter as elements to cut rather than raster. We add filled circles to our image to represent the magnet cubbies.

Unlike other profiles in the mask, these must be engraved rather than cut. Figure 95 shows an example of a completed image.

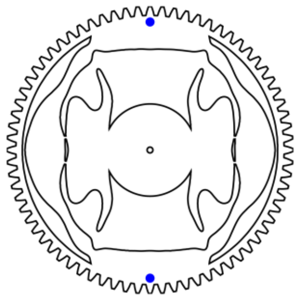


Figure 95. A complete bowtie mask representation for laser cutting. Black lines are paths to cut; blue circles are regions to engrave.

We cut the mask using a 75-watt Epilog Fusion laser cutter. Because all laser cutters and materials are different, we calibrated the machine first by running several test cuts and engravings to arrive at ideal settings (Table 10).

Table 10. Laser cutter settings used for our masks.

	Cut (for 1/8" birch)	**Raster (1/16" depth in birch)**
Speed	16%	22%
Power	100%	100%
Frequency	10%	N/A

We encountered a few small challenges in the cutting process. A faulty sensor in the laser cutter sometimes triggered spuriously mid-cut, causing the machine to seize—or even worse, continue a cut in the direction of the last command it received, sometimes clean through the remainder of the part. Obviously, this malfunction ruined the mask.

Figure 96 shows a heartbreaking example of a part that fell victim to the seizing behavior just seconds before the cut would have finished.

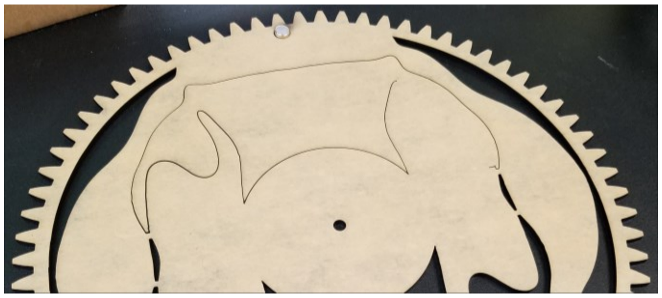


Figure 96. A mask left incomplete due to a malfunction in the laser cutter.

A more minor issue was the scorching of the wood during the etching process, which deposited ash along the mask in the direction of the airflow within the machine. For the mask's sake, the price of this was mostly aesthetic, though it is possible that the airborne debris slightly contaminated the laser lens.

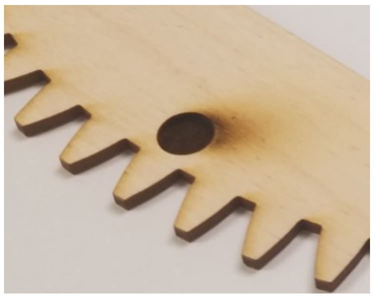


Figure 97. The etching process tended to stain the wood slightly.

Once our cuts finally succeeded, we ended up with three handsome masks, shown together in Figure 98.

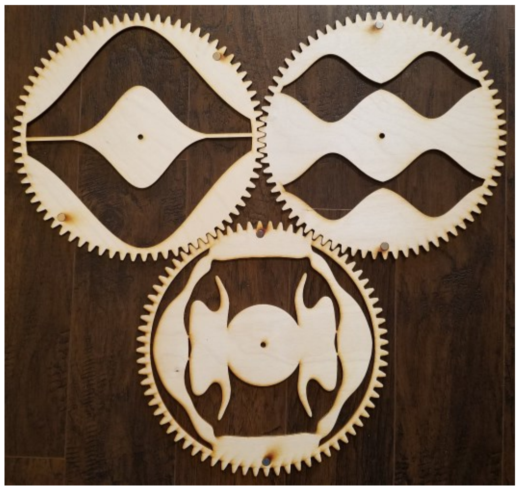


Figure 98. The result of cutting and assembling the masks. At the upper left is the Gaussian donut mask; at the upper right is the multi-Gaussian mask, and at the lower center is the bowtie mask.

Cutting the plastic pinion and pinion lips proceeded much the same as cutting the mask. Our settings for acrylic are shown in Table 11. The pinion cuts were performed on a different day than the mask cuts, so the settings may not follow the expected relationship relative to those used for the birch (Table 10).

Table 11. Laser cutter settings used for our pinion and pinion lips.

	Cut (3/16” acrylic for pinion)	**Cut (3/32” acrylic for pinion lips)**
Speed	8%	20%
Power	100%	100%
Frequency	100%	100%

True to the material’s reputation, the acrylic cut beautifully. However, we discovered an issue with the pinion lips that was not related to the cutting process. In the case the elevations of the mask and pinion did not match, the mask’s teeth tended to catch on a pinion lip at the point they met the lip's contour, causing unpleasant chattering and strain on the motor. To counteract this, we added a taper to the edges of the pinion lips in Adobe Illustrator (Figure 99). With the taper, the lips performed their function much better than before (Figure 100).

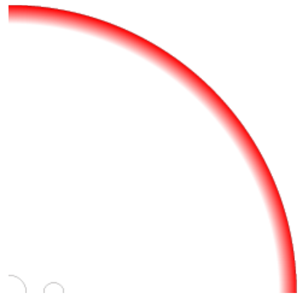


Figure 99. A radial gradient added to the pinion lip defines a taper.

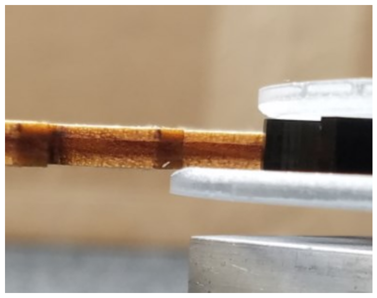


Figure 100. The tapering of the pinion lips, though subtle, effectively keeps the gear teeth from getting caught on the edge of the lips.

4.5.3 Assembly

Assembling the mechanism was mostly trivial: the axle cap slid nicely onto the secondary mirror cylinder (Figure 101); mounting the motor to the bracket and the pinion to the motor shaft was straightforward (Figure 102).

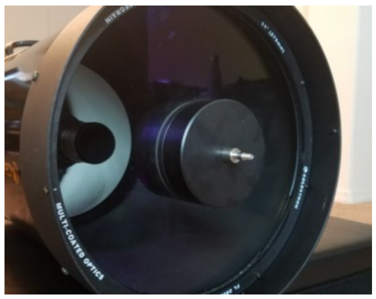


Figure 101. The axle cap placed on the C11's secondary mirror cylinder.

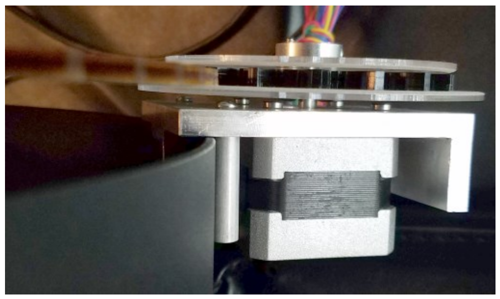


Figure 102. The motor bracket assembly attached to the telescope collar.

One unforeseen issue arose from the thread callout in the original drawing of the motor bracket, which errantly specified an 8-36 thread instead of the intended—and much more common—8-32 thread. (The part drawing in Appendix O is the corrected version.) Dutifully, Jantz tapped 8-36 thread. After the error in the drawing was discovered, we purchased a small set of 8-36 stainless steel screws to use in place of the plastic thumb screw. These new screws were a little longer than ideal, but we made the best of the limited available length options for fasteners in this uncommon thread size.

Figure 103 shows our first complete mechanical assembly of the rotation mechanism, excluding the electronics.

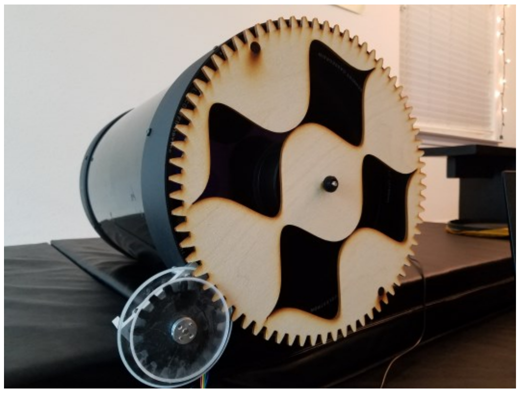


Figure 103. An assembled mask rotation mechanism less the electronics assembly. The mask shown is a prototype of the multi-Gaussian mask.

The gears meshed very nicely, leaving a slight clearance without too much slop. The region of gear meshing (which proved surprisingly difficult to photograph amid clouding in the upper pinion lip) is shown in Figure 104.

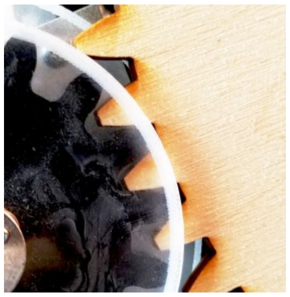


Figure 104. Region of meshing teeth in the rotation mechanism.

4.6 Mask rotator software

4.6.1 Design and interface considerations

The simplest way to communicate between a computer and our electronics is to establish a virtual serial connection between the two devices using the Arduino's UART-to-USB bridge. The Arduino IDE provides a serial terminal that operates this interface for debugging purposes. We communicate at the default rate of 19200 baud, which is slow compared to most electronics but adequate for our infrequent commands.

The selection of messages we transmit over the interface is much more interesting. To support a wide range of use cases, we establish the notion of an operating mode, which can either be absolute or relative. In absolute mode, position commands target an absolute position for the mask. To use this mode effectively, the zero position of the mask must first be established by an indexing operation. In relative mode, position commands refer to an angular position relative to the current position. This mode is convenient when indexing functions are not available or when a quick readjustment is needed. We also support continuous rotation commands, which are useful for debugging, as well as accessors that provide the current position angle and position target of the mask. A complete list of commands is included in Appendix H along with information about the electrical interface.

When a new position target is requested in absolute mode, the mechanism takes the shortest route to the new angle relative to the mask's current position, even if it has not finished executing its most recent command. For example, if the mask is currently headed from 0 to 180 degrees but receives a new command partway to travel to 270 degrees, software will decide either to continue the original rotation or to reverse course to reach 270 degrees depending on whether the mask has passed the halfway point.

Absolute positions are interpreted using modular arithmetic where values outside [0, 360) are wrapped onto this domain. For example, if the current absolute position of the mask is 1 degree and the user requests a target of 3600 degrees, software will recognize the new target as a multiple of 360 degrees, equivalent to 0, and command the motor to rotate the mask backward just one degree. In relative mode, the full angle is respected such that a target of 3600 degrees will rotate the mask ten full rotations to an identical position, which is useful for testing. All the while, the system accounts for the 72:17 gear ratio between the pinion and the mask, causing the motor to travel more than four times the mask's angular distance for any rotation operation.

The limited resolution of the stepper leads to small discrepancies between commanded positions and achievable positions. At all times, the motor's step count is maintained as an integer that never loses precision. Angles contained in mask position commands are rounded to the nearest number of motor steps. Position accessors convert the present number of motor steps to the mask angle equivalent. Using the motor step count as the definitive position gives the device immunity from floating-point and roundoff errors.

The full mask rotator code can be found in Appendix M.

4.6.2 Indexing behavior

Knowing the absolute rotation of the mask relative to the telescope is important so that we can make sense of our results as we survey the neighborhood of a star. The information can also be recorded and used to reproduce exposures if needed.

We selected a Hall effect sensor coupled with mask-mounted magnets as our index detection method as discussed in Section 4.4.2. The Hall effect sensor acts as a binary switch, where logical high indicates a magnetic field strength below the trigger threshold and logical low indicates a magnetic field strength above the trigger threshold.

As the mask rotates to an index position and the magnet approaches the sensor, we see a transition from logical high to logical low; then, as the magnet travels beyond the sensor, we see the opposite transition. At first glance, it seems that we can simply take the average of the two transition positions and declare this the true index point (reasoning that the sensor will read the most "low" when the magnet is directly above it), but this technique will not work as expected due to hysteresis in the sensor. The magnetic field strength thresholds for latching and unlatching the sensor are asymmetric, which means a two-point average will not provide the actual position of peak magnetic strength. Worse, the sense of the positional error will depend on the direction of approach. It makes more sense for us to do two passes instead—one forward and one in reverse—to eliminate this directional dependence and yield a more accurate result.

We implement the two-pass approach by first rotating the mask in one direction until we see the see the sensor latch and unlatch, then record the positions of these events. Next, we reverse the direction of rotation and again note the angles at which the sensor latches and unlatches. We figure that the average of these angles represents the point of maximum magnetic field, so we establish this point as the index position. Figure 105 illustrates this two-pass indexing algorithm using variables explained in Table 12.

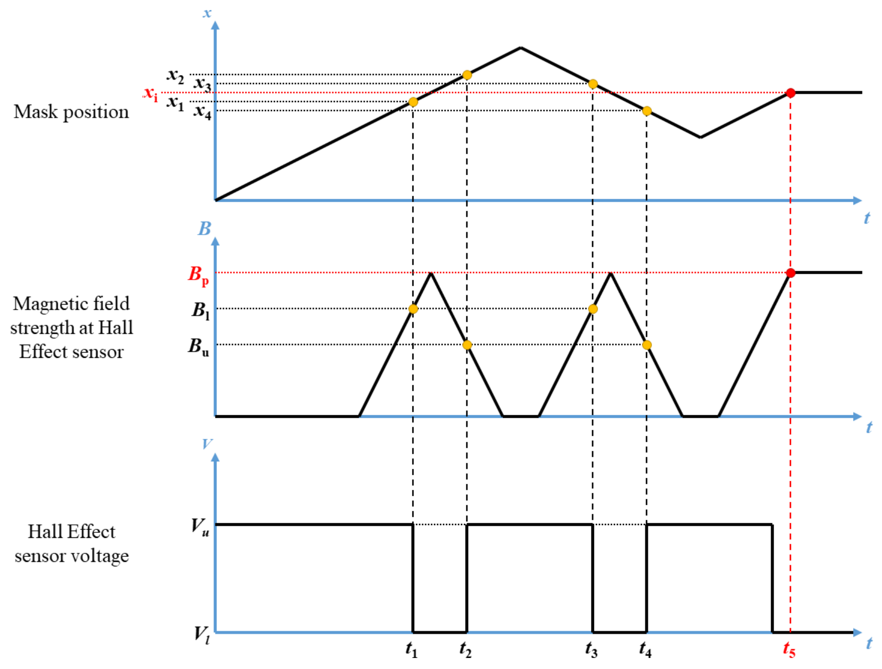


Figure 105. Depiction of the two-pass approach for indexing behavior.

Table 12. Variables used in illustration of two-pass indexing behavior.

Variable	**Significance**
x_i	Index position: expected point where $B = B_p$, calculated as $x_i = \frac{x_1+x_2+x_3+x_4}{4}$

Variable	**Significance**
t_1, x_1	Time and mask position at which Hall effect sensor latches low ($B > B_l$) during first pass
t_2, x_2	Time and mask position at which Hall effect sensor latches high ($B < B_u$) during first pass
t_3, x_3	Time and mask position at which Hall effect sensor latches low ($B > B_l$) during second pass
t_4, x_4	Time and mask position at which Hall effect sensor latches high ($B < B_u$) during second pass
t_5	Time at which mask has reached established index position ($x = x_i$)
B_p	Maximum magnetic field strength at Hall effect sensor. This is the strongest field possible given the system's physical configuration.
B_l	Latching threshold for Hall effect sensor. When B increases beyond B_l, the sensor's voltage goes from V_u to V_l.
B_u	Unlatching threshold for Hall effect sensor. When B decreases below B_u, the sensor's voltage goes from V_l to V_u.
V_l	Hall effect sensor voltage when latched (low)
V_u	Hall effect sensor voltage when unlatched (high)

The moment an indexing routine is commanded, and every time the Hall effect sensor changes state, we set a ten-second timer. If the timer expires before we reach the next step in the indexing process, we consider the routine a failure and no new index is recorded. The software object we use to coordinate indexing operations (IndexTask) can be configured to invoke a callback that communicates the routine's success or failure to the higher-level application. An indexing operation might fail if the motor stalls or if the sensor is too far away from the magnet to detect the magnetic field.

The Hall switch breakout board contains a blue LED indicating the board's power state and a yellow LED communicating when the sensor is triggered. These LEDs produce stray light that is unacceptable in an astronomical application, so we reserve a GPIO pin for the power state of the Hall switch breakout board and energize it only when an indexing operation is active. As soon as an index is acquired—or as soon as the index operation fails—we set the pin low to turn off the LEDs. This strategy also reduces power consumed by the Hall switch itself. One can desolder the offending indicators for additional insurance against stray light.

4.7 Extensions to telescopes of different sizes

The Celestron C11 is only one Schmidt–Cassegrain optical tube assembly model in a field populated with options from many different manufacturers in different sizes. To adapt the overall hardware design to other telescope configurations requires resizing the axle cap and the masks. It is likely that a new mask will have a new number of teeth, in which case the revised gear ratio must be incorporated into the mask rotator firmware. In order to interface with the existing pinion assembly, the new mask must maintain the same circular pitch as the current design.[26] If the new telescope features a central obstruction whose proportional size is significantly different than that of the C11's

[26] This restriction means that a new mask's pitch diameter cannot be selected independently of the mask's tooth count, which must be an integer. Thus, a mask in a new telescope configuration may be forced to have a smaller-than-ideal pitch diameter to grant the mask a whole number of teeth. This will contribute to clearance between the mask and the pinion.

secondary mirror cylinder, then the masks should be redesigned, and not simply scaled, for best results. The motor bracket assembly should not have to be updated due to its adjustable three-point mount (Figure 71).

5. TESTING

5.1 Overview

The performance of the masks and rotation mechanism was evaluated over the course of several experiments. Early in the project, Jimmy Ray used a Gaussian donut mask to observe Rigel and confirm that the expected diffraction pattern appeared. Later, David Rowe built masking features into his Atmospheric Seeing Distortion simulator. Some time after, Loveland et al. (2016) placed the Gaussian donut mask and Zimmerman's bowtie mask on a C11 and recorded exposures of double stars that were either close, high-contrast, or both. Finally, Foley performed mechanical tests on the rotation mechanism focused on the device's accuracy and repeatability.

5.2 Ray's tests using the Gaussian donut mask

In 2014, before delving too far into the infinite variations of theoretical aperture masks, we wanted to verify that shaped apertures produced diffraction patterns resembling our simulated results. An agreement between the two would instill confidence that our methods were sound. At the time, only a prototype Gaussian donut mask had been designed and manufactured (Figure 106). Jimmy Ray used this mask to observe Rigel using a Celestron C11 at the Arizona Sonoran Desert Observatory, Glendale (ASDOG). Rigel is a system of multiple stars whose two brightest components have an apparent magnitude difference of about 6.5 in the visual band (National Aeronautics and Space Administration: Goddard Space Flight Center, 2019), equivalent to 2.6 in a log-10 scale. These two components were separated by 9.4 arcseconds at the time, placing them at approximately 23 λ/D assuming a 548-nanometer wavelength and an 11-inch telescope. We did not expect these to be particularly challenging parameters to work with because, while the magnitude difference nearly matched our target contrast level, the separation of the components far exceeded our minimum achievable inner working angle. Ray captured Rigel in Figure 107.

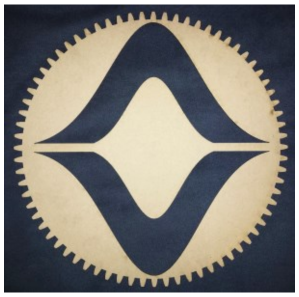


Figure 106. Prototype Gaussian donut mask used by Ray.

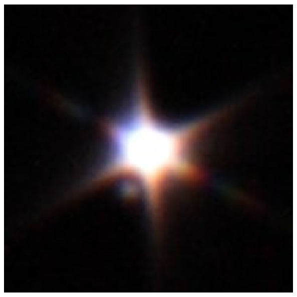


Figure 107. Rigel as seen through a prototype Gaussian donut mask, captured by Ray. (1.0-second exposure, ISO 800, f/10; captured with temperature below dew point in presence of high clouds).

In Figure 107, we clearly see one bright star and one dimmer star, as expected. (In fact, the dimmer "star" is itself multiple stars, though this is not perceptible in the image.) The hourglass shape predicted from simulations is immediately evident, with darker discovery regions along its narrow axis and brighter areas along its wide axis. An additional spike in the bright region, which we attribute to the mask's horizontal beam, is also visible. All results match the patterns we expect from our diffraction simulation. Figure 108 offers a visual comparison of Ray's results with simulations of the Gaussian donut produced by two programs.

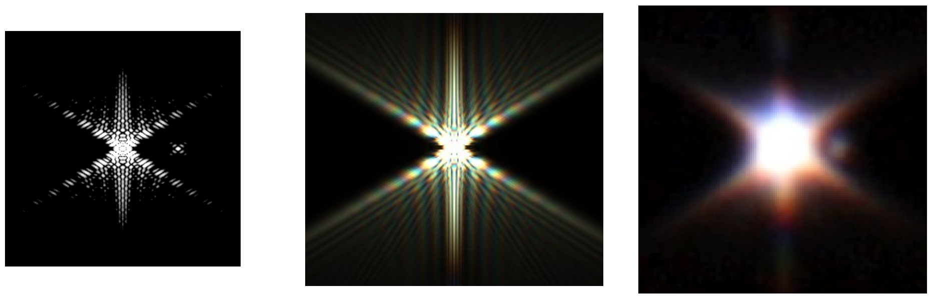
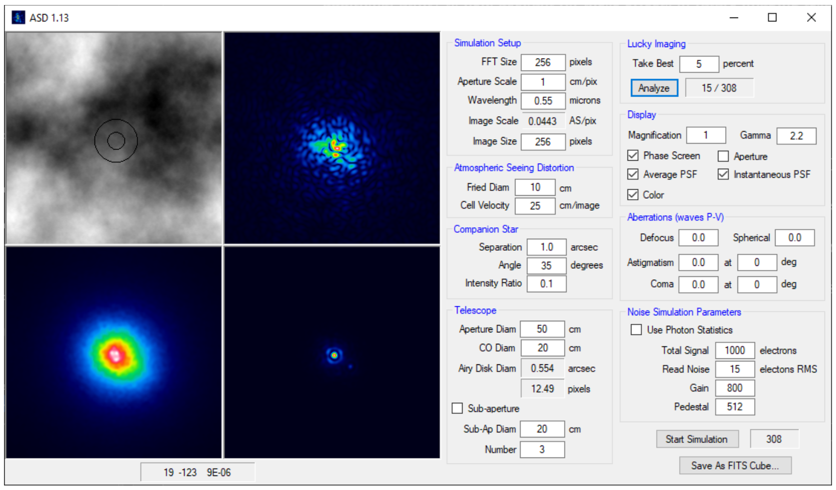

Monochromatic simulation of both stars using MATLAB
Polychromatic simulation of single star using Maskulator
Actual exposure (rotated)

Figure 108. Comparison of simulated diffraction patterns, created using MATLAB and Maskulator, to Ray's capture of Rigel using a prototype Gaussian donut mask. (Maskulator cannot simulate multiple stars simultaneously.)

The agreement between simulated and real results is not perfect: Ray's image exhibits considerable atmospheric diffusion. This is not surprising, as Ray describes capturing the image in the presence of sub-dew-point temperatures and high clouds. Meanwhile, neither our MATLAB program nor Maskulator is equipped to model atmospheric conditions. This does raise the question of how much atmospheric effects might impede the effectiveness of our masks in ways invisible to our models. While we did not have a complete answer at this stage, the question motivated us to use an additional utility to help understand these effects better, namely David Rowe's Atmospheric Seeing Distortion simulator.

5.3 Rowe's atmospheric seeing distortion simulations

Astronomer David Rowe's Atmospheric Seeing Distortion (ASD) program operates a complex optics model that accounts for diffraction, phase offsets due to atmospheric effects, aberrations in the telescope, and noise in the camera. Upon initiating the simulation, the program generates a multitude of synthesized images representing a double star system viewed at an instant in time in the presence of these effects. The user can later reduce these images via a lucky imaging function, which seeks images where atmospheric effects are minimized. Figure 109 shows this tool's graphical user interface with default settings.


ASD 1.13

Simulation Setup
FFT Size 256 pixels
Aperture Scale 1 cm/pix
Wavelength 0.55 microns
Image Scale 0.0443 AS/pix
Image Size 256 pixels

Atmospheric Seeing Distortion
Fried Diam 10 cm
Cell Velocity 25 cm/image

Companion Star
Separation 1.0 arcsec
Angle 35 degrees
Intensity Ratio 0.1

Telescope
Aperture Diam 50 cm
CO Diam 20 cm
Airy Disk Diam 0.554 arcsec
12.49 pixels
☐ Sub-aperture
Sub-Ap Diam 20 cm
Number 3

Lucky Imaging
Take Best 5 percent
Analyze 15 / 308

Display
Magnification 1 Gamma 2.2
☑ Phase Screen ☐ Aperture
☑ Average PSF ☑ Instantaneous PSF
☑ Color

Aberrations (waves P-V)
Defocus 0.0 Spherical 0.0
Astigmatism 0.0 at 0 deg
Coma 0.0 at 0 deg

Noise Simulation Parameters
☐ Use Photon Statistics
Total Signal 1000 electrons
Read Noise 15 electons RMS
Gain 800
Pedestal 512

Start Simulation 308
Save As FITS Cube...

19 -123 9E-06

Figure 109. Rowe's Atmospheric Seeing Distortion simulator.

In parallel with our project, Rowe outfitted ASD with the ability to model masked apertures. If our masks appeared to perform their intended function despite atmospheric effects, we would have even more confidence in their practical value. Indeed, as shown in Figure 110, applying a virtual Gaussian mask and post-processing the results with a deconvolution technique accentuate a secondary star that is otherwise difficult to spot.

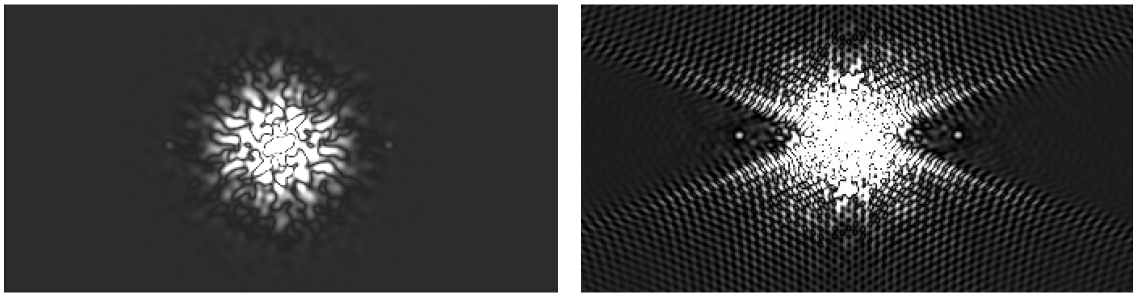


Figure 110. Comparison of a double-star deconvolution after running David Rowe's Atmospheric Seeing Distortion simulator without a mask, left, and with a Gaussian mask, right. Masking the aperture accentuates the secondary star. (The secondary appears to manifest itself twice due to deconvolution behaviors.) Both images are from David Rowe.

The quality of correlation seen in these early tests encouraged us enough to press forward with more rigorous experiments.

5.4 Loveland's tests using the Gaussian donut and bowtie masks

Loveland et al. (2016) describes aperture mask field tests performed by Donald Loveland at the Orion Observatory in Santa Margarita, California, on a Celestron C11 telescope. In these tests, the Gaussian donut and bowtie masks were used to observe four double-star systems with either high contrast, a small separation, or both. More precisely, Loveland et al. (2016) selected targets from the expansive Washington Double Star Catalog (WDS) that met the following criteria:

- Location in the northern hemisphere, where the Orion Observatory is located
- A separation angle between 1 and 15 arcseconds
- No component dimmer than an apparent visual magnitude of 12, for compatibility with the EMCCD camera used
- No primary component dimmer than 7th magnitude
- Right ascension between 13 and 20 hours

Applying these filters left 163 candidates from which Loveland et al. (2016) finally selected the four of Table 13. The group picked BU 287 to test a large delta magnitude and selected the others primarily to determine the masks' inner working angles.

Table 13. Double star systems selected for evaluation of the Gaussian donut and bowtie masks, duplicated from Loveland et. al (2016) with minor changes to headings.

Star name	**Sep. [as]**	**Mag. 1**	**Mag. 2**	**Delta mag.**	**Masks used**
BU 287	7.2	2.96	12	9.04	GD
STF 2140	4.7	3.48	5.4	1.92	GD, bowtie
STF 2579	2.5	2.89	6.27	3.38	Bowtie
BU 627	1.8	4.84	8.45	3.61	Bowtie

The Gaussian donut mask Loveland used is the same protype version that Ray used in Section 5.2 (Figure 106). This mask, seen on a C11 in Figure 111, has a somewhat narrower opening ($p = 0.36$) than that of the final design ($p = 0.50$). Consequently, its theoretical inner working angle is larger: 3.4 λ/D rather than 2.5 λ/D (Figure 114).

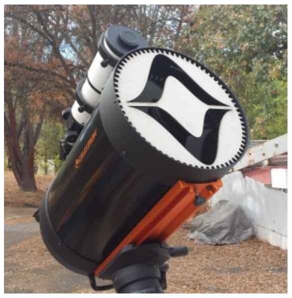


Figure 111. Gaussian donut mask on a Celestron C11 telescope.

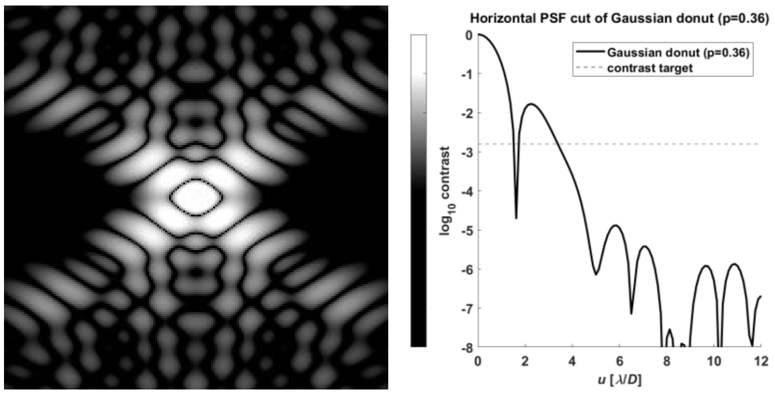
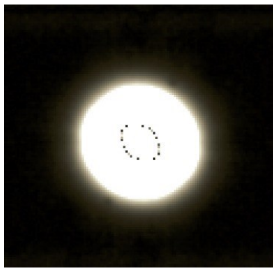
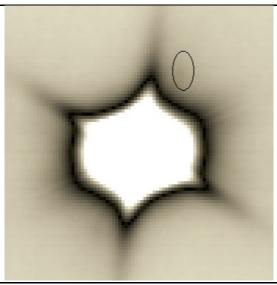
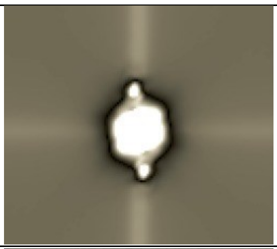
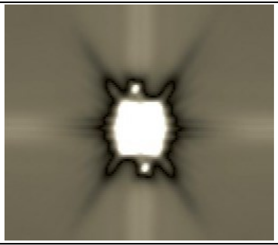
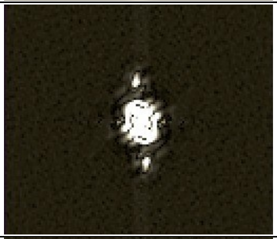
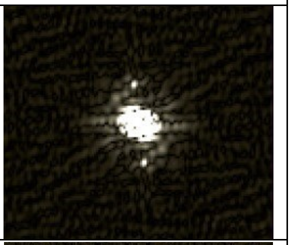
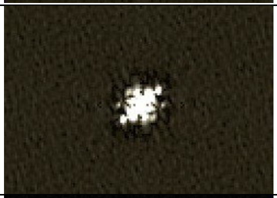
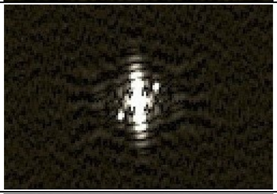
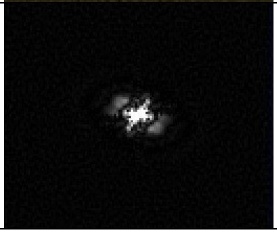
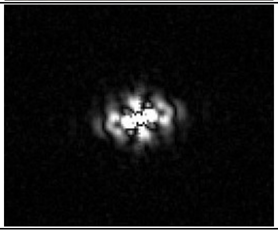


Figure 112. Diffraction behavior of the narrow Gaussian donut mask of Figure 106 used by Ray and later Loveland.

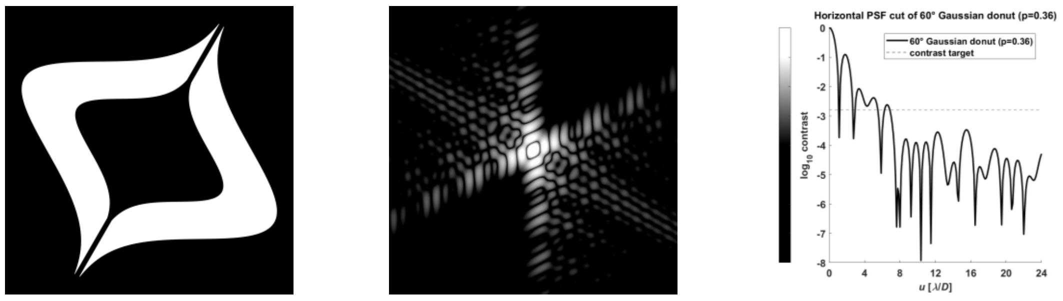
Images of the selected stars were recorded on an Andor Luca EMCCD camera and post-processed using speckle interferometry techniques, leveraging David Rowe's Plate Solve 3 program (Rowe & Genet, 2015). These images are collected in Table 14.

Table 14. Summary of images captured with and without masks by Loveland et al. (2016) using speckle interferometry techniques.

Target	**Deconv. star used**	**Unmasked**	**With Gaussian donut**	**With bowtie**
BU 287	No			—

Target	Deconv. star used	Unmasked	With Gaussian donut	With bowtie
STF 2140	No			—
	Yes		—	
STF 2579	Yes		—	
BU 627	Yes		—	

Loveland et al. (2016) found the secondary component of BU 287 to be "near the limit of the equipment's detecting ability" and "difficult to get desirable results [for]." They remarked that the reduced light throughput of the Gaussian donut mask relative to an unmasked aperture made this dim component even more difficult to detect. In both exposures, the star appears as a very subtle smudge toward the top right. It does seem that the secondary was not captured within the intended discovery zone of the Gaussian donut mask, but it is difficult to determine whether a correction would have provided materially different results. Our simulations suggest that the Gaussian donut provides a contrast of roughly 16.4 apparent visual magnitudes (−6.6 log-10 magnitudes) along the high-contrast axis at a working angle matching the stars' separation but only about 8.7 apparent visual magnitudes (−3.5 log-10 magnitudes) at the mask angle used for this exposure (Figure 113).[27] Still, because the secondary would remain at the limit of the detector's sensitivity and recognizing that these contrast predictions are sensitive to small and unpredictable optical imperfections, we cannot fully attribute the mask's lackluster performance to its orientation.



Rotated Gaussian donut Point spread function Horizontal PSF cut

Figure 113. Study of the contrast along a 60-degree axis in the narrow Gaussian donut.

STF 2140 was the only target observed with both masks. Loveland et al. (2016) began with the Gaussian donut, this time capturing the system squarely within the discovery zone. They observed that the secondary "still has some room to move inward," implying that the mask can resolve stars closer than the 4.7-arcsecond separation of this system. This is consistent with our simulations, which show a theoretical minimum inner working angle as low as 1.4 arcseconds in perfect conditions. The benefits were more pronounced with the bowtie mask: the group found that using it on the same system

[27] We modeled the rotated mask as being 60 degrees off the high-contrast axis.

produced an image with a “much clearer and [more] circular” secondary component than the equivalent in the unmasked image. The team estimated the bowtie’s theoretical inner working angle for this system under their atmospheric conditions to be about 1.8 arcseconds, which is greater than the bowtie’s theoretical 0.5-arcsecond limit in perfect conditions but still significantly better than the Gaussian donut.

Pressing toward smaller separations, Loveland et al. (2016) tested the bowtie mask on STF 2579 and BU 627. Due to the small angle between their components, both systems exhibited diffraction effects from the primary star that affected visibility of the secondary. The team determined that using the bowtie mask produced clearer diffraction nulls around each system’s secondary component than using an unmasked aperture, making each such star easier to place in the images. Of viewing STF 2579 through an unmasked aperture, Loveland et al. (2016) wrote that the “airy null pattern is neither complete nor is it as dark in comparison to the image with a [bowtie] mask.” They estimated a minimum working angle of 1.5 arcseconds. For BU 627, they wrote that “the secondary component has a clear airy null pattern surrounding it, making it much easier to pinpoint” and estimated a minimum working angle of 1.2 arcseconds.

The team ultimately found that the Gaussian donut mask offered few benefits relative to standard speckle reduction techniques on unmasked apertures, describing it as “unnecessary” and noting the loss of throughput. The multi-Gaussian mask was not available to the team, but it is likely that it would have fallen victim to the same throughput issue as the donut, pushing dim stars to the limit of the detector’s abilities.

The bowtie mask, on the other hand, “showed promising results by demonstrating very close working angles that were able to surpass the ability of speckle reduction without any mask" (Loveland, et al., 2016). Thus, Neil Zimmerman's bowtie mask passed a threshold crucial to the success of the project: using the mask produced better results in close double-star observations than not using it. Not determined in these tests is the bowtie mask's performance amid high-contrast stars with a brightness difference near the design value of 6.5 apparent visual magnitudes. We will leave the evaluation of the true practical limits of the bowtie mask for a follow-up project.

5.5 Foley's telescope-mounted rotation mechanism verification

Soon after the rotation mechanism components were machined and a C11 optical tube assembly became available, we performed an informal mechanical verification of the rotator hardware. We attached the entire device to the telescope and ran it continuously, alternating between forward and reverse, looking for any difficulties.

The rotator behaved well overall, though there were two aspects that required attention. First, when rotating the mask counter-clockwise for an extended period, the upper standoff comprising the axle began to unfasten itself, traveling slightly up the axle bolt. We resolved this, at least temporarily, by tightening the standoff. If we encountered the issue again, we could apply a thread glue to make the assembly more resistant to coming undone.

Second, when the optical tube assembly was placed at an angle nearly horizontal to the ground, we found that rotating the mask in either direction caused the axle cap assembly to work itself up the secondary mirror column. This increased the angle at which the mask rode the pinion lips, causing greater friction and more strain on the stepper motor (Figure 114). To resolve this issue requires a more robust way of attaching the axle cap to the secondary mirror column, perhaps by threading through the wall of the cap and incorporating a soft screw with a plastic or rubber tip. Thankfully, this issue should only manifest itself in practice when studying stellar targets close to the horizon, which is already not recommended. Shallow angles require light to pass through more of the atmosphere, increasing distortion.

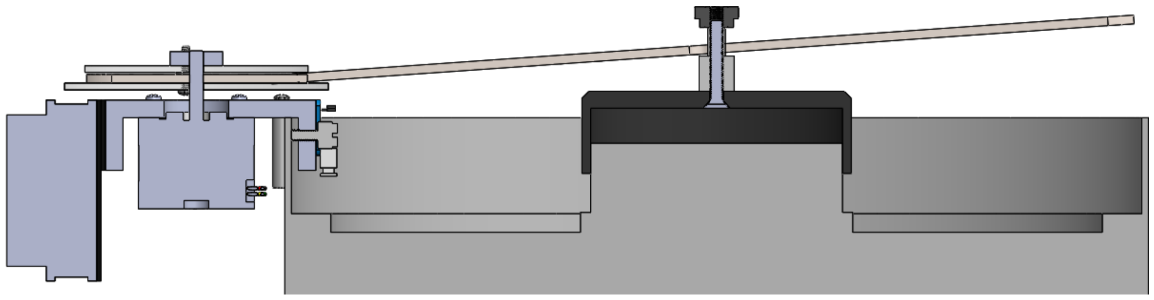


Figure 114. Cross-section of the mask rotator assembly showing mask misalignment when the axle cap is not properly seated. The effect is exaggerated here for clarity.

5.6 Foley's index quality tests

Locating angles consistently is a vital function of the mask rotator. To verify this behavior, we ran the mechanism continuously for several minutes in a controlled environment and recorded the reported mask angles at which the Hall effect sensor transitioned from high to low and from low to high. With the rotator properly calibrated and the Hall effect sensor operating correctly, we would expect to see transitions of the same type appear once every 180 degrees on average. This would reflect the presence of two magnets per mask placed in diametrical opposition to each other. If the average fell short of 180 degrees, we would suspect that we overstated the number of motor steps per revolution or misrepresented the gear ratio between the pinion and the mask. If the average exceeded 180 degrees, we could again look for an error in the number of motor steps per revolution or the gear ratio, but the more likely culprits would be missed index positions or missed motor steps. A missed index position could be caused by a weak or poorly positioned magnet. A missed motor step could result from a stall or a momentary glitch. It is also possible that we would see changes due to some temperature-dependent sensitivity of the Hall effect sensor, but we suspect this influence would be minor at most.

In our first test, we experienced an overwhelming number of missed index positions. A short investigation revealed that the separation between the magnet and the Hall effect switch was too great for the magnet's field to trigger the switch consistently. A picture of the area around the Hall effect switch (Figure 115) shows a sensor that, in agreement with Figure 90, is aligned nicely in the plane perpendicular to its sensing axis; however, the image also shows a prominent gap between the sensor and the magnet along that axis.

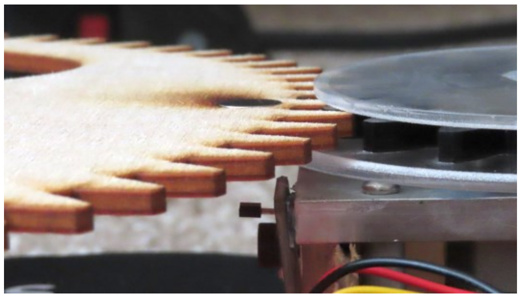


Figure 115. The Hall effect switch's position relative to the index feature.

Several remediation options were available: the magnets could be made stronger by using larger or higher-grade alternatives; the Hall effect switch breakout board could be brought closer to the mask; or the excitation voltage of the Hall effect switch could be increased. At first, inverting the mask also seemed to be an option, but doing so would reverse the polarity of the magnetic field sweeping past the Hall effect sensor—a sensor that, regrettably, detects only one magnetic pole. The new orientation would also allow the magnets to fall toward the telescope's corrector plate were their glue to fail. Instead, because the magnets and Hall effect switch breakout had already been fixed in place and because we had no safe means of increasing the circuit voltage, we opted to increase the strength of the magnetic field by adding 3/16-inch-diameter, 3/16-inch-tall neodymium magnets atop the existing magnets. The new configuration, shown in Figure 116, effortlessly and consistently triggered the Hall effect switch.

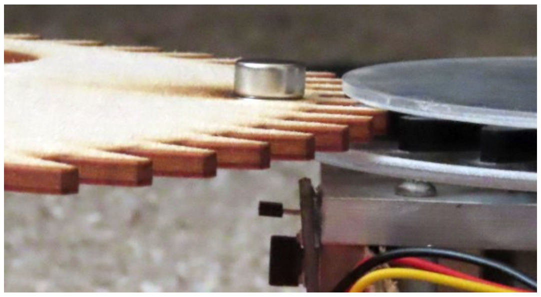


Figure 116. An extra magnet added to an existing indexing feature.

With our new magnets in place, we ran the test twice: once forward and once in reverse. In the course of each three-minute trial, we recorded 53 pairs of high-to-low and low-to-high Hall switch state transitions without missing a single location. We interpreted each set of 53 transitions as data describing 52 "gaps" between index positions that should approximate 180 degrees each. Table 15 summarizes our results. Note that the minimum resolvable mask angle increment is about 0.43 degrees, which results from the stepper motor's 200 steps per revolution and the 72:17 gear reduction:

$$\delta = (1\text{ step})\left(\frac{360°}{200\text{ steps}}\right)\left(\frac{17}{72}\right)$$
$$\rightarrow \boxed{\delta = 0.425°} \tag{11}$$

Table 15. Summary of rotation mechanism index quality test results.

Transition type	**Gaps**	**Min gap [deg]**	**Max gap [deg]**	**Avg. gap [deg]**
Forward, high-to-low	52	179.77	180.20	180.004
Forward, low-to-high	52	179.77	180.20	179.996
Reverse, high-to-low	52	−179.35	−180.63	−180.004
Reverse, low-to-high	52	−179.77	−180.20	−180.004

The results were stunningly close to perfect. The minimum and maximum gaps for three of the four transition types were separated by the smallest resolvable increment of 0.43 degrees. The remaining case contained only a single instance of a −179.35-degree gap and a complementary −180.63-degree gap, and these occurred back-to-back. By taking half the distance between these values, we estimate that a single Hall switch transition can be measured to an accuracy ±0.64 degrees when the mask is traveling one direction. This statement ignores the effects of mechanical slop.

When we seek an index position during normal mask rotator operation, we look for not just one or two Hall effect switch transition points but four. As explained in Section 4.6.2, we use the average of these four positions to identify the true zero point. If all measured transition positions were the maximum 0.64 degrees off their true value, our average would be off by the same amount. This would be a very pessimistic estimate. A more reasonable approach would be to find the root-sum-square error of this four-point average, which can be calculated as the following.

$$E_{index} = \pm \frac{\sqrt{4(0.64^2)}}{4} \quad (12)$$

$$\rightarrow \boxed{E_{index} = \pm 0.32°}$$

The value of ±0.32 degrees is half of the error from an individual measurement. We used this as a guide for our next test, where we repeatedly triggered an indexing operation and observed how much the angle changed. Running 100 consecutive indexing operations yielded 99 gaps which should average about 180 degrees each. Table 16 contains a summary of the results from this test.

Table 16. Results of repeatedly indexing the mechanism.

Transition type	**Gaps**	**Min gap [deg]**	**Max gap [deg]**	**Avg. gap [deg]**
Index operation	99	179.67	180.41	179.9993

Again, our results were surprisingly consistent, with gaps averaging almost exactly the expected 180 degrees. The variation in gaps between index positions spanned less than a degree at their extremes, implying an accuracy of ±0.37 degrees. This was slightly worse than our predicted tolerance of ±0.32 degrees, but it is still quite good. It is clear from the results that we did not miss a single motor step, that the Hall switch functioned as expected, and that there was no significant angular variation in the placement of the index magnets.

Full data for the tests in this section are available in Appendix J.

5.7 Summary of results

Though we did not test the rotation mechanism in imaging applications, the success seen in our optical and mechanical testing gave us confidence that the masks and rotation mechanism can be used together effectively. If we were to continue the project, we would verify the performance of the rotator in a stellar discovery or stellar observation application. This would help us evaluate the mechanism's resistance to changing orientations and better understand the assembly's tolerance for vibrations caused by the stepper. It would also give us valuable user experience data: How cumbersome is the process of setting up and tearing down the device? How easy is it to drop a thumb screw and have it land on the glass face of the telescope? Are there any bugs or missing features in the mask rotator firmware?

Using the rotator over the course of days and weeks would also inform us of matters regarding system life. Do the masks warp as they absorb moisture? Does anything rust and cause us problems? Do pieces of acrylic get chipped away and cause stress on the motor?

While an infinite number of tests could be added, we believe we have performed enough to establish the mask rotation mechanism as an imperfect but valuable proof of concept that performs its essential functions.

6. CONCLUSIONS AND FUTURE WORK

6.1 Summary

In the course of this project, we demonstrated that adding aperture masks to telescopes can enhance the visibility of dim secondary stars in a double-star configuration by producing diffraction patterns advantageous for the application. Using a combination of existing research and Fraunhofer diffraction simulation tools, we explored several mask configurations for the Celestron C11 optical tube assembly designed to optimize inner working angle and contrast parameters. We produced physical mask artifacts and supplemented them with a custom rotation mechanism that allows users to place masks at specific orientations. Finally, we performed and documented tests on the masks and the rotation mechanism itself. We found that the bowtie mask introduced by Neil Zimmerman improved a C11's ability to resolve close double stars and confirmed that the mask rotator successfully manipulates masks when attached to the optical tube assembly.

Though our solution works, there are many ways that it can made more effective.

6.2 Potential mask improvements

6.2.1 Tools of generation

There are an infinite number of different masks to evaluate. Some may be more effective than the three that were considered for this project. A more powerful mask creation routine that optimizes for working angle bounds and aperture geometry would be especially powerful. One method is described in Carlotti et al. (2011).

6.2.2 Gaussian boomerang mask

A Gaussian donut mask variation whose promise we discovered too late in the project to fully explore is one whose secondary Gaussian has a broadness factor different than that of the primary opening. Following a trend observed in Table 3 where wider Gaussian features produce wider discovery regions, we would expect a broad secondary to behave better in general than a narrower variant of the same height. Indeed, a cursory optimization of the primary broadness p_1 and secondary broadness p_2 found that values $p_1 = 0.50$ and $p_2 = 1.20$ result in a mask that has good diffraction characteristics along the horizontal axis in simulation (Appendix I). In the author's opinion, the shape of the mask's openings resembles a boomerang reflected about the horizontal axis; thus, we will call this mask a Gaussian boomerang (Figure 117). Unlike the Gaussian donut mask, no additional support structure is required since the truncation of the tails of the interior Gaussian shape leaves plenty of material connected to the outside.

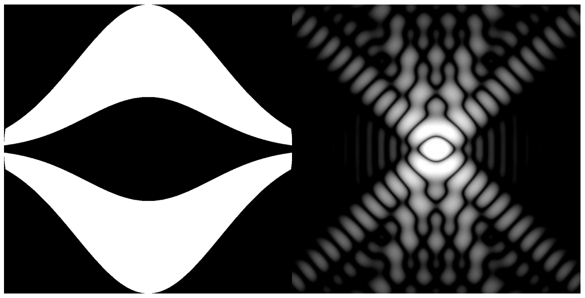


Figure 117. The Gaussian boomerang mask and its characteristic diffraction pattern, plotted on a nonlinear brightness scale.

Along the horizontal axis, the Gaussian boomerang eliminates a critical diffraction ring present in the C11 pattern (Figure 118). It reaches our contrast target at a small working angle of 1.38 λ/D, competitive with Zimmerman's bowtie mask of

Section 3.4.3, which reaches this threshold at $1.27\ \lambda/D$ (Figure 119). Unlike the bowtie mask, however, the Gaussian boomerang achieves this contrast for only a narrow range of position angles. It is thus likely worse suited for the discovery of close binaries because it requires more mask rotation steps. On the other hand, the mask maintains a high level of contrast to working angles larger than the bowtie offers and eliminates multiple diffraction spikes characteristic of the C11 aperture, so it may still perform well on high-contrast pairs.

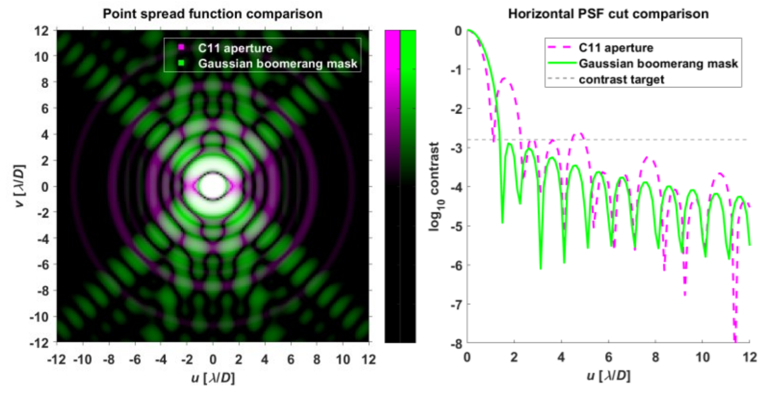


Figure 118. Point spread function comparison of the Gaussian boomerang mask and the C11 aperture.

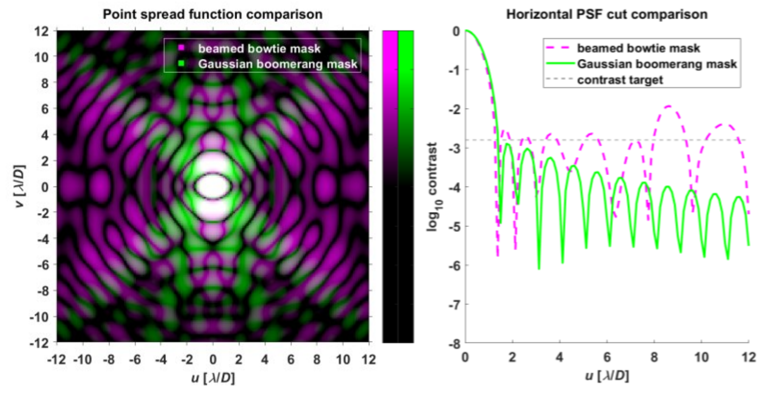


Figure 119. Point spread function comparison of the Gaussian boomerang and bowtie masks.

The apparently complex tradeoffs introduced by the bowtie and Gaussian boomerang masks may be worthwhile subjects for a follow-up study.

6.3 Diffraction simulation improvements

Though MATLAB is well suited for two-dimensional fast Fourier transforms, the operations demand significant amounts of memory and take a very long time to compute when acting on input images as large as ours—especially after padding (Appendix B.2). These issues are compounded by the number of locations in our simulation code where the gigantic complex matrices that result are copied to variables in memory, forcing expensive allocation operations. A better solution would recognize the symmetry in our masks and use this to reduce the quantity of redundant data being fed into the Fourier transform operation, accelerating the computation and cutting down the size of the output matrix. This output could also be shuttled to its destination more efficiently by preallocating memory and perhaps by implementing a cropping function that deletes transform data outside the domain of interest before variable assignment.

Still faster performance might be achieved by switching languages altogether. A compiled language like C++ has the potential to run significantly faster than an interpreted language like MATLAB because a compiler can use the context of the entire program to optimize execution, whereas an interpreter may only make optimization decisions one statement at a time. That said, fully switching away from MATLAB would require reimplementing not just the Fraunhofer diffraction routine but also the convenient plotting functions, so this decision should not be made lightly.

6.4 Mask rotator software improvements

Like the masks it controls, the software interface can be made more effective. By design, motor control runs in parallel with the task that reads and responds to commands sent over the serial connection. This implementation allows the user to stop a mask that is currently in transit regardless of the action the rotator is currently taking. However, in some circumstances, such as an indexing operation, a higher-level task expects uninterrupted access to the mask controller and behaves in unpredictable ways if this chain of command is broken while the task is active. The most robust way to solve this issue is to implement a semaphore that controls access to StepperController and MaskController objects. This addition would require new error handling logic that recognizes and reports conditions where operations fail to reserve a necessary resource.

6.5 Mechanical improvements

In this project, we went through great lengths to avoid having the rotation mechanism affect the optics of the telescope. This allowed us to evaluate the impact of our masks relative to the unobstructed aperture and to each other without having to qualify these conclusions with details of the mechanism's state. We rejected designs with thin structural elements or wires placed across the aperture because they would create diffraction spikes. It is possible that we overestimated the negative impact of these thin elements, which would cause interference at working angles likely too large to affect double-star observations. Admitting these elements into the design space would allow the mask axle to couple to something other than the telescope's secondary mirror cylinder, such as a spider attached to the telescope rim. The mask could even attach directly to a motor shaft and the motor’s wires could stretch across the aperture, removing the need for gears entirely. A continuation of this project might explore alternate configurations that sacrifice perfect optics for a less complicated, more reliable rotation mechanism that avoids mounting to sensitive telescope components.

6.6 Closing remarks

The last five and a half years have given me a deep appreciation for astronomers past and present. Their pursuit of a better understanding of the universe will never end, but nonetheless they continue out of pure love for the celestial bodies around them. I hope that the product of this thesis may advance us just a little farther down this endlessly enriching and fascinating path.

REFERENCES

Acrylite. (n.d.). *Technical Information: ACRYLITE® Cast and Extruded Acrylic.* Retrieved from ACRYLITE® - Colors, Patterns, and Functions: https://www.acrylite.net/product/acrylite/downloads/1213g-light-transmission-and-reflectance-tech-data.pdf

Arduino AG. (2019). *Arduino Motor Shield Rev3*. Retrieved October 4, 2019, from Arduino Official Store: https://store.arduino.cc/usa/arduino-motor-shield-rev3

Arduino AG. (2019). *Arduino Uno Rev3*. Retrieved October 4, 2019, from Arduino Official Store: https://store.arduino.cc/usa/arduino-uno-rev3

Astrozap. (2019). *Dew Shields*. Retrieved September 21, 2019, from Astrozap: Telescopes and Accessories: https://astrozap.com/collections/dew-shields

Atmel Corporation. (2015). *ATmega328P.* Retrieved October 4, 2019, from http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7810-Automotive-Microcontrollers-ATmega328P_Datasheet.pdf

Bahtinov, P. (2005, September 22). *Optimal Focusing Mask*. Retrieved August 14, 2019, from https://astronomy.ru/forum/index.php/topic,10421.0.html

Balasubramanian, K., White, V., Yee, K., Echternach, P., Muller, R., Dickie, M., . . . Kasdin, J. (2015). Exoplanet Coronagraph Shaped Pupil Masks and Laboratory Scale Star Shade Masks: Design, Fabrication and Characterization. In S. Shaklan (Ed.), *Proceedings Volume 9605, SPIE Optical Engineering + Applications: Techniques and Instrumentation for Detection of Exoplanets VII.* San Diego. doi:10.1117/12.2188954

Bennett, J., Donahue, M., Schneider, N., & Voit, M. (2014). *The Cosmic Perspective* (7th ed.). (N. Whilton, Ed.) San Francisco, California, United States of America: Pearson.

Bernat, D., Bouchez, A. H., Ireland, M., Tuthill, P., Martinache, F., Angione, J., . . . Lloyd, J. P. (2010, May). A Close Companion Search Around L Dwarfs Using Aperture Masking Interferometry and Palomar Laser Guide Star Adaptive Optics. *The Astrophysical Journal, 715*(2). doi:10.1088/0004-637X/715/2/724

Bernhard, P. (2012, November 12). *Ma Nouvelle Config Astrophoto Grand Champ, Grand Capteur, Grande Ouverture.* Retrieved July 10, 2019, from Astron'amis: https://www.astronamis.net/t2346-ma-nouvelle-config-astrophoto-grand-champ-grand-capteur-grande-ouverture

Boccaletti, A., Moutou, C., Mouillet, D., Lagrange, A.-M., & Augereau, J.-C. (2001, February). Dark-speckle Coronagraphic Detection of Binary Stars in the Near-IR. *Astronomy and Astrophysics, 367*(1), 371–380.

Caldwell, M. E., & Gray, P. F. (1997, October 1). Application of a generalized diffraction analysis to the design of nonstandard Lyot-stop systems for earth limb viewing radiometers. *Optical Engineering, 36*(10), 2793–2808. doi:10.1117/1.601506

Carey, G. (2009, March 26). *The Carey Mask.* Retrieved August 14, 2019, from FC Astronomy: http://www.geoastro.co.uk/careymask.htm

Carlotti, A., Vanderbei, R., & Kasdin, N. J. (2011). Optimal Pupil Apodizations of Arbitrary Apertures for High-Contrast Imaging. *Optics Express, 19*(27), 26796–26809. doi:10.1364/OE.19.026796

Coelho, R. C., Calvão, M. O., Reis, R. R., & Siffert, B. B. (2014, November). Standardization of Type Ia Supernovae. *European Journal of Physics, 36*(1). doi:10.1088/0143-0807/36/1/015007

Cross, M. C. (2000, January 16). *Chapter 6: Power Spectrum.* Retrieved August 23, 2019, from Condensed Matter Physics: http://www.cmp.caltech.edu/~mcc/Chaos_Course/Lesson6/Power.pdf

CutLaserCut Ltd. (2015). *Understanding the 'Kerf' of the Laser*. Retrieved April 29, 2019, from Cut Laser Cut: http://www.cutlasercut.com/resources/tips-and-advice/what-is-laser-kerf

Daley, J. A. (2007). A Method of Measuring High Delta m Doubles. *Journal of Double Star Observations, 3*(4), 159–164.

Daley, J. A. (2014, April 1). LSO Double Star Measures for the Year 2012. *Journal of Double Star Observations, 10*(2), 136–138.

Darrigol, O. (2012). *A History of Optics from Greek Antiquity to the Nineteenth Century.* New York, New York, United States of America: Oxford University Press Inc.

Debes, J. H., & Ge, J. (2004, July). High-Contrast Imaging with Gaussian Aperture Pupil Masks. *Publications of the Astronomical Society of the Pacific, 116*(821), 674–681. doi:10.1086/422764

Debes, J. H., Ge, J., & Chakraborty, A. (2002, June 20). First High-Contrast Imaging Using a Gaussian Aperture Mask. *The Astrophysical Journal, 572*(2). doi:10.1086/341876

Debes, J. H., Ge, J., Mandelowitz, C., & Watson, A. (2003, February). High-Contrast Imaging with Gaussian-Shaped Pupils. *Proceedings Volume 4860, High-Contrast*

Imaging for Exo-Planet Detection. 116, pp. 138–148. Waikoloa: SPIE. doi:10.1117/12.457670

Eagle, D. (2013, August 10). *Making an Occulting Bar*. Retrieved October 16, 2019, from Star-Gazing: http://www.eagleseye.me.uk/Resources/OccultingBar.pdf

Engineering ToolBox. (2004). *Density of Various Wood Species*. Retrieved November 23, 2019, from https://www.engineeringtoolbox.com/wood-density-d_40.html

Engineering ToolBox. (2009). *Density of Selected Solids*. Retrieved November 23, 2019, from https://www.engineeringtoolbox.com/density-solids-d_1265.html

Enya, K., & Abe, L. (2011, August). A Binary Shaped Mask Coronagraph for a Segmented Pupil. *Publications of the Astronomical Society of Japan, 62*(6). doi:10.1093/pasj/62.6.1407

Florentino, K. (2009, October 30). *A Do-It-Yourself Apodizing Mask*. Retrieved May 24, 2018, from Colorado Springs Astronomical Society: http://csastro.org/a-do-it-yourself-apodizing-mask/

Foley, E. L., Genet, R. M., Ridgely, J. R., Rowe, D., & Zimmerman, N. T. (2015, September). Observation of Large-Delta-Magnitude Close Binaries with Shaped Aperture Masks. *Journal of Double Star Observations, 11*(1S), 343–360.

Ge, J., Debes, J. H., Watson, A., & Chakraborty, A. (2002). Imaging Survey for Faint Companions with Shaped Pupil Masks. *Proceedings Volume 4835, Future Research Direction and Visions for Astronomy.* Waikoloa: SPIE. doi:10.1117/12.456513

Hecht, E. (2002). *Optics* (4th ed.). Reading, Massachusetts, United States of America: Addison–Wesley.

Herschel, J. F. (1847). *Results of Astronomical Observations Made During the Years 1834, 5, 6, 7, 8, at the Cape of Good Hope; Being the Completion of a Telescopic Survey of the Whole Surface of the Visible Heavens, Commenced in 1825.* London: Smith, Elder and Co.

Hessmer, R. (2014). *Involute Spur Gear Builder*. Retrieved October 2, 2019, from Rainer's Web: http://hessmer.org/gears/InvoluteSpurGearBuilder.html

Hillenbrand, L. A., & White, R. J. (2004, April 1). An Assessment of Dynamical Mass Constraints on Pre–Main-Sequence Evolutionary Tracks. *The Astrophysical Journal, 604*, 741–757. doi:10.1086/382021

Kasdin, N. J., Vanderbei, R. J., Littman, M. G., Carr, M., & Spergel, D. N. (2004). The Shaped Pupil Coronagraph for Planet Finding Coronagraphy: Optimization, Sensitivity, and Laboratory Testing. *Optical, Infrared, and Millimeter Space Telescopes. 5487.* Glasgow: SPIE Astronomical Telescopes + Instrumentation. doi:10.1117/12.552273

Kasdin, N. J., Vanderbei, R. J., Spergel, D. N., & Littman, M. G. (2003). Optimal Shaped Pupil Coronagraphs for Extrasolar Planet Finding. *Proceedings. 4860.* Waikoloa: SPIE. doi:10.1117/12.457675

Kasdin, N. J., Vanderbei, R., Littman, M., & Spergel, D. N. (2005, April). Optimal One-Dimensional Apodizations and Shaped Pupils for Planet Finding Coronagraphy. *Applied Optics, 44*(7), 1117–1128. doi:10.1364/AO.44.001117

Kenworthy, M. (2018, May 16). *Apodizing Phase Plate Coronagraph.* Retrieved May 29, 2018, from Matthew Kenworthy: http://home.strw.leidenuniv.nl/~kenworthy/app

Koenig, D. B. (2009). *United States of America Patent No. 7,595,942.*

Lacour, S., Tuthill, P., Amico, P., Ireland, M., Ehrenreich, D., Huelamo, N., & Lagrange, A.-M. (2011, July). Sparse Aperture Masking at the VLT: I. Faint Companion Detection Limits for the Two Debris Disk Stars HD 92945 and HD 141569. (E. Sciences, Ed.) *Astronomy and Astrophysics, 532*. doi:10.1051/0004-6361/201116712

Leibniz-Institut für Astrophysik Potsdam. (2014, February 13). *Johnson–Cousins UBVRI filter curves*. Retrieved February 16, 2019, from Leibniz-Institut für Astrophysik Potsdam: http://www.aip.de/en/research/facilities/stella/instruments/data/johnson-ubvri-filter-curves

Lindenblad, I. W. (1970, September). Relative Photographic Positions and Magnitude Difference of the Components of Sirius. *The Astronomical Journal, 75*(7), 841–847. doi:10.1086/111029

Loveland, D., Foley, E., Genet, R., Zimmerman, N., Rowe, D., Harshaw, R., & Ray, J. (2016, March). Detecting Faint Secondary Stars with Shaped Aperture Masks. *Journal of Double Star Observations, 12*(3).

Lovró, F. (n.d.). *DIY Apodizing Mask*. Retrieved May 24, 2018, from Graphite Galaxy: http://www.graphitegalaxy.com/index.cgi?a=diyapodmask

Malmquist, K. G. (1925, February). A Contribution to the Problem of Determining the Distribution in Space of the Stars. *Meddelanden fran Lunds Astronomiska Observatorium, 106*, 1–12.

Martinez, P., Dorrer, C., Carpentier, E. A., Kasper, M., Boccaletti, A., Dohlen, K., & Yaitskova, N. (2009, February). Design, Analysis, and Testing of a Microdot

Apodizer for the Apodized Pupil Lyot Coronagraph. *Astronomy and Astrophysics, 495*(1). doi:10.1051/0004-6361:200810918

MatWeb. (n.d.). *Overview of Materials for 6000 Series Aluminum Alloy*. Retrieved May 3, 2019, from Online Materials Information Resource - MatWeb: http://www.matweb.com/search/DataSheet.aspx?MatGUID=26d19f2d20654a489aefc0d9c247cebf

MatWeb. (n.d.). *Overview of Materials for Acetal Copolymer, Unreinforced*. Retrieved May 3, 2019, from Online Materials Information Resource - MatWeb: http://www.matweb.com/search/datasheet.aspx?MatGUID=c3039ef87c9245448cdebe961b19a54c

MatWeb. (n.d.). *Overview of Materials for AISI 4000 Series Steel*. Retrieved May 3, 2019, from Online Materials Information Resource - MatWeb: http://www.matweb.com/search/DataSheet.aspx?MatGUID=210fcd12132049d0a3e0cabe7d091eef

McMaster–Carr Supply Company. (2019). *Female Threaded Round Standoff, 18-8 Stainless Steel, 1/2" OD, 1/2" Long, 10-32 Thread Size*. Retrieved October 11, 2019, from McMaster-Carr: https://www.mcmaster.com/91125a381

NASA/CXC/SAO. (2015, March 9). *Pulsating Variable Stars and the Hertzsprung-Russell (H-R) Diagram.* Retrieved from Chandra X-ray Observatory: http://chandra.harvard.edu/edu/formal/variable_stars/bg_info.html

National Aeronautics and Space Administration: Goddard Space Flight Center. (2019, Feb 16). *HEASARC Browse: Query Results*. Retrieved Feb 16, 2019, from

NASA's HEASARC: High Energy Astrophysics Science Archive Research Center: https://heasarc.gsfc.nasa.gov/db-perl/W3Browse/w3query.pl

Noordhoek, N. (2009, July 14). *Simulate you[r] own mask diffraction pattern*. Retrieved October 16, 2019, from Niels Noordhoek's weblog: http://www.njnoordhoek.com/?p=376

Oppenheimer, B. R. (2003). *Coronagraphy.* Retrieved May 29, 2018, from The Lyot Project: http://lyot.org/background/coronagraphy.html

Osgood, B. G. (2007). *The Fourier Transform and Its Applications.* Retrieved May 11, 2019, from Stanford Engineering Everywhere: https://see.stanford.edu/materials/lsoftaee261/book-fall-07.pdf

Park, S., Kwon, Y., & Oh, H.-K. (2002, November). Resolution Enhancement Method Using an Apodized Mask. *Journal of the Korean Physical Society, 41*(5), 687–692.

Perryman, M. (2011). *The Exoplanet Handbook.* New York, New York, United States of America: Cambridge University Press.

Portugal, R. D., & Svaiter, B. F. (2011, February). Weber–Fechner Law and the Optimality of the Logarithmic Scale. *Minds and Machines, 21*(1), 73–81. doi:10.1007/s11023-010-9221-z

Reidl, M. J. (2001). *Optical Design Fundamentals for Infrared Systems* (2nd ed.). Bellingham, Washington, United States of America: SPIE Press.

Richmond, M. W. (2007, October 28). *Contributing Photometry of Novae and Supernovae*. Retrieved May 16, 2018, from Rochester Institute of Technology: http://spiff.rit.edu/richmond/sne/aavso/nova_sn.html

Roberts Jr., L. C. (1998). *Three Methods of Determining Magnitudes of Individual Stars in Resolved Binary Systems.* PhD Thesis, Georgia State University, College of Arts and Sciences, Georgia.

Roelfsema, P. R. (2019). *SPICA: Unveiling the Obscured Universe.* Retrieved November 1, 2019, from https://spica-mission.org/spica-mission.org/version2/wp-content/uploads/2019/06/SPICA-M5-Proposal-V1.1.pdf

Ross, R. J. (2010). *Wood Handbook: Wood as an Engineering Material.* USDA Forest Service, Forest Products Laboratory. Madison, WI: United States Department of Agriculture. Retrieved from https://www.fpl.fs.fed.us/documnts/fplgtr/fpl_gtr190.pdf

Rowe, D., & Genet, R. (2015, September). User's Guide to PS3 Speckle Interferometry Reduction Program. *Journal of Double Star Observations, 11*(1S).

Sacek, V. (2019, August 10). *6.7 Coherent Transfer Function, Fourier Transform*. Retrieved August 23, 2019, from Notes on Amateur Telescope Optics: https://www.telescope-optics.net/mtf2.htm

Sacek, V. (2019, August 8). *7.3 Apodizing Mask.* Retrieved August 23, 2019, from Amateur Telescope Optics: http://telescope-optics.net/apodizing_mask.htm

Slepian, D. (1965, September). Analytic Solution of Two Apodization Problems. *Journal of the Optical Society of America, 55*(9), 1110–1115.

Slepian, D., & Pollak, H. O. (1961, January). Prolate Spheroidal Wave Functions, Fourier Analysis and Uncertainty – I. *The Bell System Technical Journal*, 43–84. doi:10.1002/j.1538-7305.1961.tb03976.x

Smith, L. (n.d.). *Dealing with Diffraction in Telescopes*. Retrieved October 16, 2019, from Tumbleweed observatory's Astronomy Hints: https://astronomyhints.braintidbits.com/apodize.html

SparkFun Electronics, Inc. (2019). *Stepper Motor with Cable - ROB-09238*. Retrieved September 21, 2019, from SparkFun Electronics: https://www.sparkfun.com/products/9238

Specht, E. (2009, September 1). *12 circles in a square*. Retrieved October 16, 2019, from The best known packings of equal circles in a square: http://hydra.nat.uni-magdeburg.de/packing/csq/csq12.html

Spectral cross-correlation templates - SDSS DR7. (2005, June 14). Retrieved from Sloan Digital Sky Survey: http://classic.sdss.org/dr7/algorithms/spectemplates/index.html

Spergel, D. N. (2000). A New Pupil for Detecting Extrasolar Planets. doi:arXiv:astro-ph/0101142

STMicroelectronics. (2000). *L298 Dual Full-Bridge Driver.* Retrieved October 4, 2019, from https://www.st.com/en/motor-drivers/l298.html

Suiter, H. R. (2001, Summer). *Apodization for Obstructed Apertures.* Retrieved May 24, 2018, from Astronomical Society of Bay County: http://www.bay-astronomers.org/TM/ApodDes.pdf

Suiter, H. R. (2001, Summer). *Making an Apodizing Screen.* Retrieved May 24, 2018, from Astronomical Society of Bay County: http://www.bay-astronomers.org/TM/MakingApod.pdf

SunFounder. (2017, March 20). *Hall Sensor module*. Retrieved October 5, 2019, from Sunfounder - Starter Kits, Robots DIY Electronic Kits for STEM Education: http://wiki.sunfounder.cc/index.php?title=Hall_Sensor_module

Swinburne University of Technology. (n.d.). *Airy Disk*. Retrieved October 16, 2019, from COSMOS - The SAO Encyclopedia of Astronomy: http://astronomy.swin.edu.au/cosmos/A/Airy+Disk

Tanaka, S., Enya, K., Abe, L., Nakagawa, T., & Kataza, H. (2006, June 25). Binary-Shped Pupil Coronagraphs for High-Contrast Imaging Using a Space Telescope with Central Obstructions. *Publications of the Astronomical Society of Japan, 58*(3), 627–639. doi:10.1093/pasj/58.3.627

van Albada, G. B. (1958). Telescope Diaphragms for the Observation of Faint and Fairly Close Binary Components. *Contributions from the Bosscha Observatory*.

van Dokkum, P. G., & Conroy, C. (2010, December 16). A Substantial Population of Low Mass Stars in Luminous Elliptical Galaxies. *Nature, 468*, 940–942. doi:10.1038/nature09578

Vanderbei, R. J., Kasdin, N. J., & Spergel, D. N. (2004, November). Checkerboard-Mask Coronagraphs for High-Contrast Imaging. *The Astrophysical Journal, 615*(1). doi:10.1086/423947

Vanderbei, R. J., Kasdin, N. J., Spergel, D. N., & Kuchner, M. (2003). New Pupil Masks for High-Contrast Imaging. In D. R. Coulter (Ed.), *Proceedings of the International Society for Optical Engineering, Techniques and instrumentation for Detection of Exoplanets*, *5170.* San Diego. doi:10.1117/12.505307

Vanderbei, R. J., Spergel, D. N., & Kasdin, N. J. (2003, March). Spiderweb Masks for High-Contrast Imaging. *The Astrophysical Journal, 590*(1). doi:10.1086/374971

Vanderbei, R. J., Spergel, D. N., & Kasdin, N. J. (2008, December). Circularly Symmetric Apodization via Starshaped Masks. *The Astrophysical Journal, 599*(1). doi:10.1086/379238

W.S. Hampshire, Inc. (2019, May 30). *Other Data: Acetal Grades.* Retrieved September 23, 2019, from Non-Metallic Material Custom Fabricator - Hampshire, Illinois - W.S. Hampshire, Inc.: https://www.wshampshire.com/wp-content/uploads/acetal_grades.pdf

Wang, R. (2009, July 5). *Properties of Fourier Transform*. Retrieved May 10, 2019, from Harvey Mudd College: http://fourier.eng.hmc.edu/e101/lectures/handout3/node2.html

Weisstein, E. W. (2007). *Fraunhofer Diffraction*. Retrieved May 2, 2019, from Eric Weisstein's World of Physics: http://scienceworld.wolfram.com/physics/FraunhoferDiffraction.html

Wilkes, J. T. (2015). *Reverse First Principles: Weber's Law and Optimality in Different Senses.* Master's Thesis, University of California, Santa Barbara, Department of Psychology. Retrieved from https://escholarship.org/uc/item/6rj5j1zg

Yang, J., Cowan, N. B., & Abbot, D. S. (2013, June 27). Stabilizing Cloud Feedback Dramatically Expands the Habitable Zone of Tidally Locked Planets. *The Astrophysical Journal Letters, 771*(2), L45. doi:10.1088/2041-8205/771/2/L45

Zeus Industrial Products, Inc. (2005). *UV Properties of Plastics: Transmission & Resistance.* Technical white paper. Retrieved October 17, 2019, from http://www-eng.lbl.gov/~shuman/NEXT/MATERIALS&COMPONENTS/WLS_materials/Zeus_UV_Properties.pdf

Zimmerman, N. T. (2014). *Summary of Shaped Pupil Optimizations for Russ Genet's Binary Star Survey with 11-Inch SCT.*

Sloan Digital Sky Survey acknowledgment

Funding for the Sloan Digital Sky Survey (SDSS) and SDSS-II has been provided by the Alfred P. Sloan Foundation, the Participating Institutions, the National Science Foundation, the U.S. Department of Energy, the National Aeronautics and Space Administration, the Japanese Monbukagakusho, and the Max Planck Society, and the Higher Education Funding Council for England. The SDSS Web site is http://www.sdss.org/.

The SDSS is managed by the Astrophysical Research Consortium (ARC) for the Participating Institutions. The Participating Institutions are the American Museum of Natural History, Astrophysical Institute Potsdam, University of Basel, University of Cambridge, Case Western Reserve University, The University of Chicago, Drexel University, Fermilab, the Institute for Advanced Study, the Japan Participation Group, The Johns Hopkins University, the Joint Institute for Nuclear Astrophysics, the Kavli Institute for Particle Astrophysics and Cosmology, the Korean Scientist Group, the Chinese Academy of Sciences (LAMOST), Los Alamos National Laboratory, the Max-Planck-Institute for Astronomy (MPIA), the Max-Planck-Institute for Astrophysics (MPA), New Mexico State University, Ohio State University, University of Pittsburgh,

University of Portsmouth, Princeton University, the United States Naval Observatory, and the University of Washington.

A. OPTICS OF SHAPED APERTURES

A.1 Fraunhofer diffraction and its implications

The Huygens–Fresnel principle states that every point of a wave front can be modeled as a source of spherical waves, and so the optical field represents the superposition of the wavelets from these point sources (Hecht, 2002, p. 421). This principle relies on virtual point sources of light that have no obvious physical justification, but it nonetheless conveniently predicts diffraction effects in many optical configurations.

In 1882, Gustav Kirchhoff derived a more rigorous result from the homogeneous wave equation and showed it was mathematically equivalent to the Huygens–Fresnel model (Darrigol, 2012; Hecht, 2002, p. 422). This result, known today as the Fresnel–Kirchhoff diffraction formula, is valid when the wavelength of light is small compared to the diffracting aperture (Hecht, 2002, p. 422).[28] It can be written as follows, adapted from Weisstein (2007):

$$\psi(x, y) = C \int_{A} \xi(x', y') \exp\left[\frac{-2\pi i(xx' + yy')}{R\lambda}\right] dx'dy' \tag{13}$$

In this equation, ψ is the wavefunction in the projection plane given as a function of projection plane coordinates x and y; C is a constant; A is the aperture; ξ is a factor that accounts for transmission properties of the aperture as a function of coordinates x' and y' in the aperture plane. (In our case, $\xi \in [0, 1]$, where 0 is opaque and 1 is

[28] This condition readily applies to us. Even the smallest apertures in our design space, about 15 centimeters (6 inches), are far larger than the sub-micrometer wavelengths of light they admit. The smallest features we can machine in a mask using a laser cutter—approximately 250 micrometers (CutLaserCut Ltd., 2015)—are still hundreds of times larger than the wavelength of near-infrared light.

transparent.[29]) R represents the distance separating the aperture and projection planes, and λ is the wavelength of light.

The great distance between the telescope and our stellar subjects allows us to model the incoming rays of light as parallel. Under this condition, the rays incident on the aperture will converge to the focal plane of the telescope, which is where we record our image in a well-calibrated system. The angle at which the parallel rays strike the telescope relative to the optical axis is small, implying that the location to which the rays converge will be at a distance from the optical axis proportional to this small angle. This outcome gives us some flexibility to simplify the Kirchhoff–Fresnel formula. Whereas $u = \tan\frac{x}{R}$ and $v = \tan\frac{y}{R}$ in general, for small u and v, we can apply a small-angle approximation to write $u \approx \frac{x}{R}$ and $v \approx \frac{y}{R}$. After making this substitution, our simplified Kirchhoff diffraction formula becomes the following:

$$\psi(u,v) = C \int_A \xi(x',y') \exp\left[-2\pi i \left(\frac{u}{\lambda}x' + \frac{v}{\lambda}y'\right)\right] dx'dy' \qquad (14)$$

This equation is the Fraunhofer diffraction equation. Figure 120 offers some visual context for its variables.

[29] If our masks could change not just the amplitude but the phase of the incoming light, this factor would take the more general form $\xi = Te^{i\theta}$, where T is the translucency and θ is the phase offset.

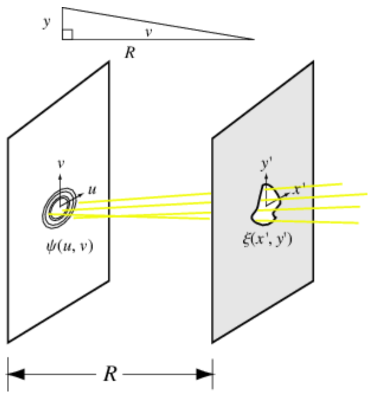


Figure 120. Geometry of Fraunhofer diffraction (Weisstein, 2007). Light shines through the aperture on the right to the projection plane on the left.

If we introduce variables $\hat{u} = \frac{u}{\lambda}$ and $\hat{v} = \frac{v}{\lambda}$, we reach Equation 15:

$$\psi(\hat{u}, \hat{v}) = C \int_A \xi(x', y') \exp[-2\pi i(\hat{u}x' + \hat{v}y')] \, dx'dy' \tag{15}$$

We recognize this integral as a Fourier transform of the aperture. This result is very convenient because it allows us to model aperture diffraction effects through elegant properties of the Fourier transform such as linearity, superposition, and convolution.

One such property, which concerns a dichotomy of space and frequency scaling (Wang, 2009; Osgood, 2007, pp. 349–350), shows

$$\mathcal{F}\{\xi(ax', ay')\} = \frac{1}{Ca^2} \psi\left(\frac{\hat{u}}{a}, \frac{\hat{v}}{a}\right) \tag{16}$$

This implies that reducing the scale of an aperture by setting $0 < a < 1$ increases the angular scale of the diffraction pattern to $\frac{1}{a}$ times its original value in both axes. This mathematically demonstrates what we already knew: smaller apertures produce broader diffraction patterns.

Other elegant relationships emerge from our definitions of $\hat{u}$ and $\hat{v}$, which can be inverted to yield $u = \lambda\hat{u}$ and $v = \lambda\hat{v}$. It becomes plain to see that the angular size of the diffraction pattern scales linearly with the wavelength of light.

Putting these two facts together, under the condition of Fraunhofer diffraction, the angular size of the diffraction pattern is proportional to the wavelength and inversely proportional to the aperture diameter.

A.2 Power spectrum

The bands of light in a diffraction pattern correspond to concentrations of power contained in the wavefunction, ψ. The field of such power concentrations is known as the power spectrum and can be calculated as the square of the complex amplitude of the wavefunction (Cross, 2000):

$$P(u,v) = \|\psi(u,v)\|^2 \tag{17}$$

Whereas the wavefunction is complex in general, the power spectrum will contain strictly non-negative real values. If the wavefunction represents a single point of light, then the power spectrum is the point spread function.

A.3 Superposition of apertures

The linearity property of the Fourier transform (Osgood, 2007, p. 347) states that

$$\mathcal{F}\{(\alpha f_1 + \beta f_2 + \cdots + \omega f_n)(\boldsymbol{\xi})\} = \alpha\mathcal{F}\{f_1(\boldsymbol{\xi})\} + \beta\mathcal{F}\{f_2(\boldsymbol{\xi})\} + \cdots + \omega\mathcal{F}\{f_n(\boldsymbol{\xi})\} \tag{18}$$

Adapting this equation to our Fraunhofer diffraction case, we can say that the wavefunction representing diffraction due to a composite aperture expressed as a combination of other apertures is simply a linear combination of the wavefunctions due to those apertures alone:

$$\psi_N = \alpha\psi_1 + \beta\psi_2 + \cdots + \Omega\psi_n \tag{19}$$

This is a powerful result because it offers insight into how shapes added or removed from an aperture affect the new diffraction pattern. When a coefficient is 1, a shape is added and a new opening is created; when the coefficient is −1, a shape is removed and an existing transparent area becomes opaque. We can conclude from superposition that modified apertures will exhibit diffraction effects from the shapes added or removed—a comforting and intuitive result for those of us born without an innate, lucid comprehension of two-dimensional Fourier transforms.

Some caution is warranted, however. The principle of superposition applies only to the complex wavefunctions and not to the power spectra in general:

$$\begin{gathered} \|\alpha\psi_1 + \beta\psi_2 + \cdots + \omega\psi_n\|^2 \neq \|\alpha\psi_1\|^2 + \|\beta\psi_2\|^2 + \cdots + \|\omega\psi_n\|^2 \\ \rightarrow P_N \neq P_1 + P_2 + \cdots + P_n \end{gathered} \tag{20}$$

Still, the composite power spectrum will typically contain artifacts seen in the power spectra of the shapes. To demonstrate this, we explore the effect of adding a square obstruction to a circular aperture in Figure 121.

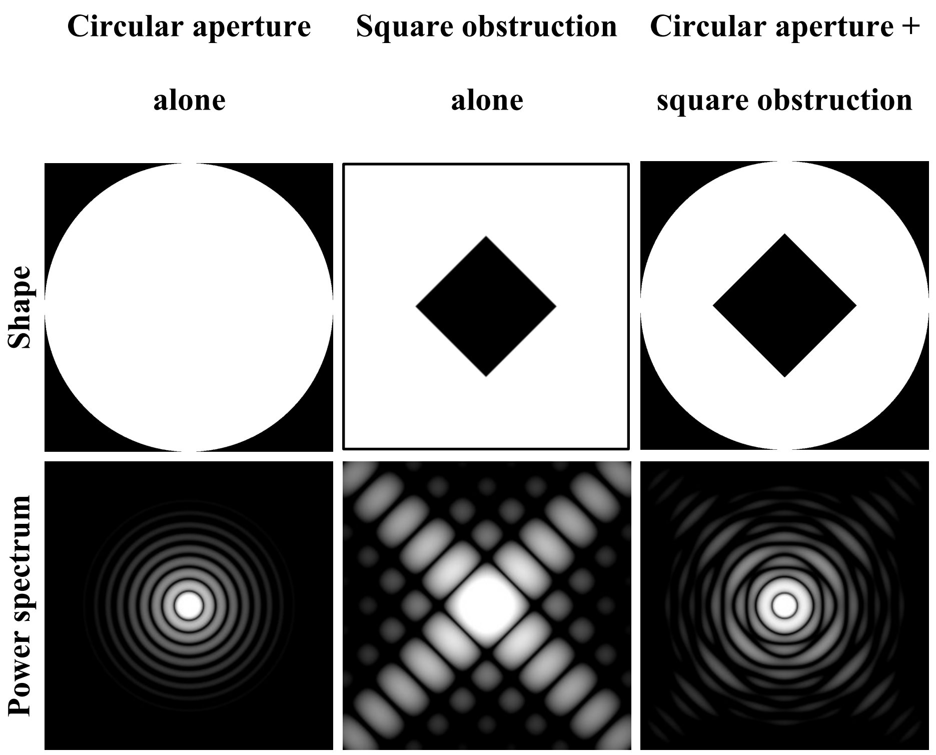


Figure 121. Demonstration of superposing a square obstruction on a circular aperture. The power spectrum of the composite aperture exhibits features of the constituent apertures' power spectra, including circular features at the image's center and a subtle X shape. Power spectra are plotted on a nonlinear brightness scale.

A.4 Superposition of light sources

We can predict the joint power spectrum of multiple stars by convolving the aperture's characteristic point spread function with the locations of stars in the field. For each star, we adjust the intensity of the pattern proportional to the star's brightness, as seen in Figure 122.

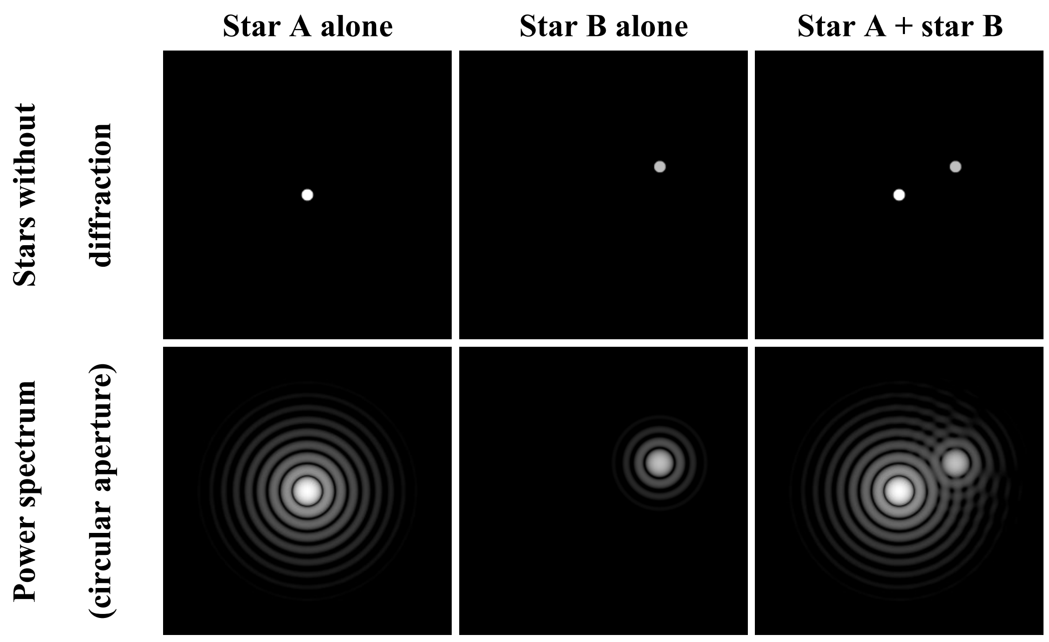


Figure 122. Demonstration of the effect of multiple light sources on the power spectrum. Diameter of stars in top row is exaggerated for clarity. Power spectra are plotted on a nonlinear brightness scale.

This approach to convolution is valid only because our stellar subjects represent incoherent light sources creating light whose mutual interference can be ignored: coherent light sources instead require convolution of the wavefunction, not the power spectrum (Sacek, 2019b).

B. MONOCHROMATIC DIFFRACTION SIMULATION USING MATLAB

B.1 Diffraction simulation overview

In general, diffraction patterns defy simple arithmetic representation. We turn to the computational power offered by programs such as MATLAB in order to obtain numerical results instead. Though these tools are optimized for performing quick calculations, some operations such as Fourier transforms can be very expensive without making certain accommodations.

The general approach to calculating diffraction patterns consists of the following steps. We assume the conditions of Fraunhofer diffraction, monochromatic light, perfect focus, and no atmospheric distortion.

1. Load the aperture image.
2. Optionally, downscale the aperture image.
3. Pad the image with opaque regions.
4. Take the fast two-dimensional Fourier transform.
5. Spatially shift the result of the Fourier transform to move the zero-frequency component to the center of the image.
6. Find the power spectrum by squaring the magnitude of the Fourier transform element-by-element.
7. Normalize the power spectrum by dividing each element by the spectrum's maximum value.
8. Place the result on a nonlinear brightness scale by taking the logarithm of the normalized power spectrum element-by-element.
9. Plot the result.

Each step is detailed in subsections that follow. Code that performs these steps is available in Appendix K.

B.2 Loading and conditioning the aperture image

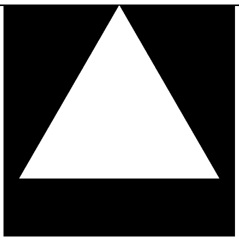
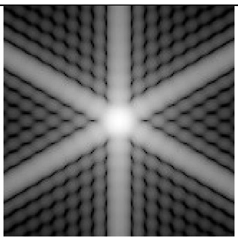
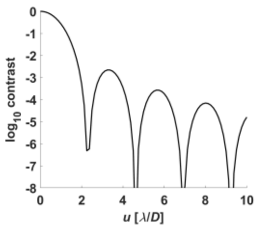
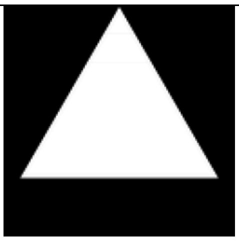
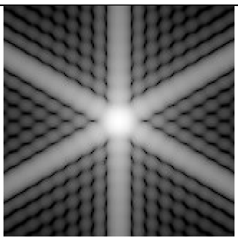
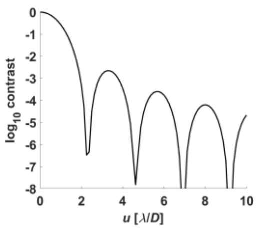
We load the aperture into the program as a square, grayscale image, with white pixels representing regions of perfect transparency and black pixels representing regions that are perfectly opaque. Shaped aperture masks appear as white regions bounded by black regions—pixels with intermediate gray values appear only along white–black borders and only when the mask image has been preprocessed with an anti-aliasing filter that smooths contours. The gradated apodizing masks of Section 1.3.4, true to their translucent nature, have gray pixel values throughout the image.

For convenience, we interpret the image as if its square dimension is the nominal diameter of the telescope. This lets us easily attach axes to the original image that span $[-0.5D, +0.5D]$ in both dimensions. It also associates a scale with the image so that we can later represent power spectrum coordinates in terms of λ/D.

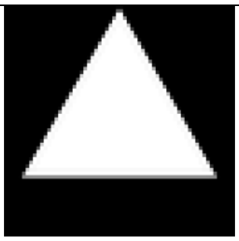
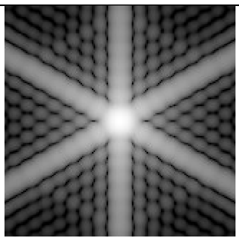
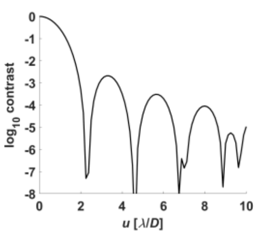
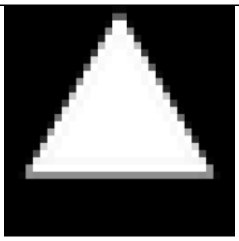
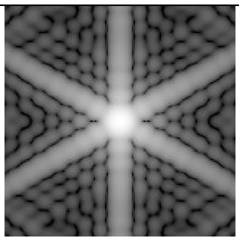
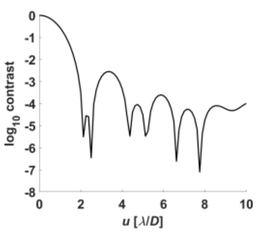
Optionally, we downscale the mask image before taking its transform. A smaller mask image significantly reduces intermediate variable storage during the fast Fourier transform process. It also drastically accelerates the calculation of the transform. The savings in execution time can be traded for larger padding factors in step 3, which increase resolution. Unfortunately, in most cases, downscaling incurs an irreversible loss of fidelity in the original image that propagates through dependent calculations, meaning that while the output transform may be more precise, it is less accurate. The more the image is downscaled, the more the antialiasing filter used to resample the image affects the result, usually by diminishing high-frequency components in the pattern. In addition, the algorithm by which an image is downscaled may differ between implementations, leading to unpredictable outcomes.

Table 17 explores the effect of gradually downscaling a triangular aperture image from 2048 by 2048 pixels to just 32 by 32 pixels. Though the fast Fourier transform output displays remarkable resistance to the downscaling of the input image, artifacts do become noticeable when the image reaches about 64 by 64 pixels or so.

Table 17. Demonstration of aperture image downscaling effects.

Scaling factor	**FFT input (before 8 × pad[30])**	**FFT output[31]**	**Horizontal cut**
1	 (2048×2048)		
$\frac{1}{16}$	 (128×128)		

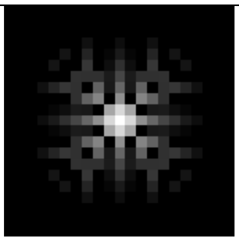
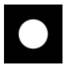
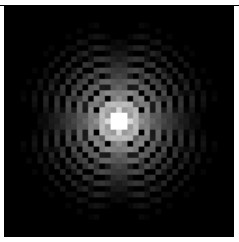
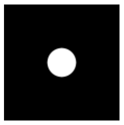
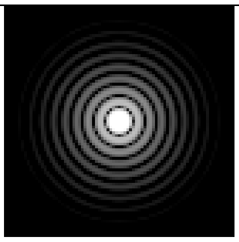
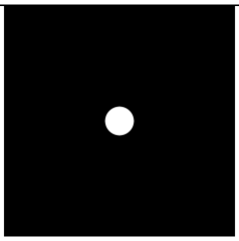
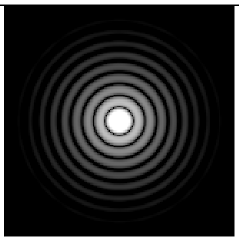
[30] See following paragraphs for an explanation of the padding factor. Padding is included here to provide good resolution in the output FFTs.
[31] Square of magnitude of Fourier transform, shifted to place (0, 0) frequency component at image center and plotted on a nonlinear brightness scale. All outputs are shown limited to the same frequency domain of $u \in [-10, 10]D^{-1}; v \in [-10, 10]D^{-1}$.

$\frac{1}{32}$	 (64×64)		
$\frac{1}{64}$	 (32×32)		

Before deciding to downscale an image, it is important to run a study across multiple scaling factors to ensure that the factor in question offers an appropriate balance of accuracy and fidelity for the application. A scaling factor of one will provide the best results, but smaller fractions run significantly faster and can be useful for drafts.

Padding the aperture image with black pixels before taking the Fourier transform is another useful technique. Black pixels increase the image dimensions and allow us to calculate spatial frequency content at non-integer multiples of the reciprocal of the aperture diameter. In other words, padding the image improves the spatial resolution of our Fourier transform. We define the *padding factor* to be the ratio of the square dimension of the padded image to the square dimension of the unpadded image. By this definition, a 256-by-256-pixel image that is padded to a size of 1024 by 1024 pixels would have a padding factor of 4. See Table 18 for a comparison of different padding factors and their impact on the output transform.

Table 18. Demonstration of padding factor effects.

Padding factor	**FFT input**	**FFT output**[32]	**Spat. freq. resolution [px/$\boldsymbol{D^{-1}}$]**
1	(256×256)		1
2	 (512×512)		2
4	 (1024×1024)		4
8	 (2048×2048)		8

[32] Square of magnitude of Fourier transform, shifted to place (0, 0) frequency component at image center and plotted on a nonlinear brightness scale. All outputs are shown limited to the same frequency domain of $u \in [-10, 10]D^{-1}$; $v \in [-10, 10]D^{-1}$ and have been scaled to the same size on the page to illustrate the differences in spatial resolution.

Throughout this paper, we use 2048-by-2048 pixel aperture images wherever possible.[33] We generally apply a padding factor of 8 before taking the Fourier transform.

B.3 Performing the two-dimensional fast Fourier transform

We next invoke a function that calculates the complex factors of the Fourier transform. In order to perform this expensive operation expediently, we use a two-dimensional Fourier transform method that is specialized for square images with dimensions of 2^n pixels by 2^n pixels—n a positive integer—called the *two-dimensional fast Fourier transform*. The output of this operation is a two-dimensional array of complex numbers corresponding to the amplitude and phase of all distinguishable spatial frequency components in the input image. MATLAB's function for performing a two-dimensional fast Fourier transform is `fft2()`.

The fast Fourier transform assumes a periodic input image whereas in practice our inputs are aperiodic. Padding images before taking the two-dimensional fast Fourier transform more faithfully reflects the non-repeating nature of the real masks and produces higher-fidelity results at the expense of added computation time.

B.4 Converting the Fourier transform output to a normalized power spectrum

In our application, $(u, v) = (0, 0)$ represents the lowest frequency component of the image, which would be seen collocated with a star in an actual exposure. The component at this frequency is natively represented at the top-left pixel when displayed using a raster coordinate convention. For ease of interpretation, we translate the output so

[33] The bowtie mask of Section 3.4.4, provided to us at a resolution of 1000 by 1000 pixels, is one notable exception.

that the low-frequency components appear at the center of the image rather than its corners. MATLAB's `fftshift()` function performs this translation. It can also be executed manually by defining

$$\begin{aligned} x' &= \left(x + \frac{h}{2}\right) \% \, h \\ y' &= \left(y + \frac{w}{2}\right) \% \, w \end{aligned} \tag{21}$$

where *h* is the image height, *w* is the image width, and the % symbol represents the modulo (remainder) operator. Note that per raster coordinate convention, *x* is the vertical coordinate and *y* is the horizontal coordinate.

As we saw in Appendix A.2, we can get the power spectrum by squaring the magnitude of the Fourier transform. In MATLAB, if `F` represents the shifted Fourier transform output, getting the power spectrum is as simple as calling `abs(F).^2`.

The values within the power spectrum will depend not only on the shape of the aperture that creates it but also the size of the image representing the aperture. In our application, however, we care only about the shape of the aperture. To remove the variance caused by the image size, we normalize the entire spectrum by the maximum value within it. With MATLAB, using `P` for our power spectrum, we can write `P = P./max(max(P))`. This results in the value 1 representing the brightest point of the spectrum—almost always its center—and all other values representing the fraction of power density relative to this brightest point. We have established a normalized contrast scale!

B.5 Plotting the power spectrum

Our normalized contrast scale has a huge dynamic range spanning many orders of magnitude. To illustrate all important features of the power spectrum, we must use a nonlinear brightness scale. One way to do this would be to apply a gamma correction, raising every element in the spectrum to the power $1/4$, say. In this paper, we instead take the logarithm so that the pixel values within our image correspond linearly to the astronomical apparent visual magnitude scale. In MATLAB, we simply call `log10()` to get the base-10 logarithm and multiply by -2.5 to get the difference in apparent visual magnitude (see Equation 3 in Section 1.2).

Since our maximum component is 1 before performing this transformation, the base-10 logarithm will yield strictly nonpositive values and the apparent visual magnitude matrix will contain strictly nonnegative values. With either scale, we must select which value corresponds to black pixels and which value corresponds to white pixels. The decision is a matter of personal taste, but it helps to pick values that reveal important details of the spectrum without washing out the image. In this paper, we plot most power spectra on a log-10 range of $[-4, -1]$, which leaves components -4 or less black and -1 or greater white. Exceptions are made where appropriate.

B.6 Plotting a multi-star image

We know from Appendix A.4 that we can simulate a monochromatic multi-star image by convolving the power spectrum with the star positions, adjusting the brightness of each pattern in proportion to each star's brightness. To this point, our power spectrum has been defined in normalized angular units of λ/D, but star positions are given in absolute angular units of arcseconds. Converting λ/D into arcseconds requires us to at last specify our aperture diameter and wavelength of interest. In this paper, we default to an aperture diameter of 11 inches to match the Celestron C11. Our wavelength selection depends on the context.

To prepare the convolution, we calculate the theoretical indices where stars would be present within a matrix built to the same spatial scale as the point spread function. (This is where the contextualization of the λ/D units occurs.) We also calculate each star's absolute brightness. We then convolve the non-log-scaled point spread function with our theoretical star matrix, lightening or attenuating each instance of the pattern in proportion to the calculated brightness. MATLAB's `conv2()` function can perform this action if provided a real matrix, but we implement an alternative that leverages the sparseness of the star data to achieve much better performance. (See documentation within the getStarView.m file of Appendix K.3 for details.)

Once the convolution is complete, we can take the logarithm of the data and plot the result in much the same way as we did the point spread function.

C. NOTES ON MASKULATOR

C.1 Downloading and running Maskulator

(Please exercise caution when downloading, installing, and executing programs from the Internet. The author used these applications and libraries in this section without issue but cannot guarantee that they are safe. Continue at your own risk.)

Niels Noordhoek's diffraction simulation utility, Maskulator (Section 2.2.2), is available at http://www.njnoordhoek.com/?p=376. It will not function correctly after being downloaded to a 64-bit PC without additional steps. By default, the button labeled **calculate** will have no effect when clicked. To resolve the issue, perform the sequence of steps below.

1. Download Maskulator v5.0 from http://www.njnoordhoek.com/?p=376.
2. Open the Maskulator zip file and extract its contents to a new folder. We will call this folder the Maskulator folder.
3. Download precompiled 64-bit Windows DLLs for the Fastest Fast Fourier Transform in the West (FFTW) C subroutine library. These are available at http://fftw.org/install/windows.html. Make sure to select the 64-bit version. The author used FFTW v3.3.5, but other versions are likely to work.
4. Open the FFTW zip file and copy only the file called libfftw3f-3.dll to your Maskulator folder. When prompted, choose to replace the existing file.
5. Open Maskulator from your Maskulator folder, press **Load mask**, select an image from your Maskulator folder, and press the **calculate** button. Confirm that a diffraction pattern appears in the display.

C.2 Relevant parameters within Maskulator

Maskulator offers several parameters that affect the generated diffraction pattern. The parameters most relevant to this project are summarized in Table 19.

Table 19. Description of selected Maskulator parameters.

Parameter	**Units**	**Min**	**Max**	**Effect**
annotation	–	–	–	Uncheck to visualize the diffraction pattern without overlaid information about the telescope focus.
matrix size N ()	px	256	2048	Increasing this number increases the angular domain covered by the Fast Fourier transform output but does not increase its resolution. In effect, a smaller matrix size produces an output that is a cropped version of the output from a larger matrix size—but note its effects on the “brightness” parameter.
start	nm	350	780	The lower end of the spectrum displayed in the diffraction pattern. Slices with wavelengths less than the default value of 450 nanometers or greater than 650 nanometers are not displayed.
steps	–	1	64	How many steps into which to divide the spectral range defined by the “start” and “stop” parameters, including the endpoints. A value of 1

Parameter	**Units**	**Min**	**Max**	**Effect**
				uses only the “start” parameter. The output produced by Maskulator is a combination of this number of slices, where each slice is a different wavelength. Larger numbers produce smoother spectra at the expense of additional computation time.
stop	nm	350	780	The upper end of the spectrum displayed in the diffraction pattern. Slices with wavelengths less than 450 nanometers or greater than the default value of 650 nanometers are not displayed.
Brightness	(unk.)	0	10000	Higher values correspond to a brighter image, bringing dim or even otherwise invisible features into view. The natural brightness of an image appears to increase linearly with the “matrix size N ()” parameter, requiring a reciprocal adjustment to brightness for equivalent results. For example, an image to be generated with a matrix size of 1024 should have its brightness set to one-fourth the value of the brightness of an image generated using a matrix size of 256.

Parameters not listed in Table 19, including “aperture D (m),” “focal length f (m),” “Barlow magnification(),” and “Defocus (microns),” are used to explore in detail the effects of telescope focus on diffraction patterns and are not vital to the conclusions of this thesis.

To achieve increased resolution in the diffraction pattern, the original mask images must be manually padded with black pixels using another tool before supplying them to Maskulator. Where focus is important and the input image does not represent the precise extent of the aperture to be studied, one must also modify the “aperture D (m)” parameter to reflect the theoretical extent of the entire input image in the aperture plane. For example, if a 1024-by-1024-pixel image represents a 0.25-meter aperture and it is padded with black pixels to form a 2048-by-2048-pixel image, “aperture D (m)” should be set to 0.50 even though 0.50 meters is larger than the telescope aperture’s diameter.

D. CALCULATION OF REQUIRED STEPPER MOTOR TORQUE

To begin calculating the amount of motor torque needed to rotate our masks, we make several assumptions:

- The mask can be modeled as a solid, circular plate of birch 0.125 inches thick and 12 inches in diameter. These quantities match the material thickness and pitch diameter of our mask gear. Naturally, our masks are not solid, so this assumption should yield an overestimate of the component's mass moment of inertia.
- The magnets in the mask assembly can be ignored.
- The pinion can be modeled as a solid, circular cylinder of acrylic 0.375 inches thick and 2.833 inches in diameter. These quantities match the height of the acrylic stack in the pinion assembly (pinion plus two lips) and the pinion's pitch diameter.
- The leading face of a pinion tooth begins in contact with a mask tooth such that when the stepper motor begins to move, the mask too begins to move.
- The stepper motor accelerates the pinion and the mask to their steady-state speeds in one pulse. We drive the stepper motor at 125 hertz, so each pulse is 8 milliseconds long.

With these assumptions in place, we perform the calculations in Table 20.

Table 20. Steps of a calculation that estimates the required torque of our stepper motor.

Term	Value	Notes
Mask diameter	$d_m = 12 \text{ in}$	–

Term	Value	Notes
Mask radius	$r_m = \frac{d_m}{2}$ $\rightarrow r_m = 6 \text{ in}$	
Mask thickness	$a_m = 0.125 \text{ in}$	–
Mask density	$\rho_m = 0.023\underline{1}\,\frac{\text{lbm}}{\text{in}^3}$	[34]
Mask mass moment of inertia	$I_m = \frac{1}{2} m_m r_m^2$ $\rightarrow I_m = \frac{\pi}{2} \rho_m a_m r_m^4$ $\rightarrow I_m = 5.8\underline{8} \text{ lbm} \cdot \text{in}^2$	–
Pinion diameter	$d_p = 2.833 \text{ in}$	–
Pinion radius	$r_p = \frac{d_p}{2}$ $\rightarrow r_p = 1.41\underline{7} \text{ in}$	–
Pinion thickness	$a_p = 0.375 \text{ in}$	–
Pinion density	$\rho_p = 0.043\underline{0}\,\frac{\text{lbm}}{\text{in}^3}$	[35]
Pinion mass moment of inertia	$I_p = \frac{1}{2} m_p r_p^2$ $\rightarrow I_p = \frac{\pi}{2} \rho_p a_p r_p^4$ $\rightarrow I_p = 0.10\underline{2} \text{ lbm} \cdot \text{in}^2$	–

[34] Taking the average of the values quoted in Engineering ToolBox (2004) yields 640 kg/m^3, which is equivalent to 0.0231 lbm/in^3.
[35] This is the 1190 kg/m^3 quoted in Engineering ToolBox (2009) converted to lbm/in^3.

Term	Value	Notes
Stepper angle advanced per step	$\Delta\theta_s = 1.8\text{ deg}$	[36]
Stepper driver step period	$\Delta t_s = 0.008\text{ s}$	[37]
Pinion steady-state angular velocity	$\omega_{p,ss} = \left(\frac{\Delta\theta_s}{\Delta t_s}\right)\left(\frac{2\pi\text{ rad}}{360\text{ deg}}\right)$ $\rightarrow \omega_{p,ss} = 3.92\underline{7}\,\frac{\text{rad}}{\text{s}}$	–
Gear ratio	$K = \frac{72}{17}$	–
Mask steady-state angular velocity	$\omega_{m,ss} = \frac{\omega_{p,ss}}{K}$ $\rightarrow \omega_{m,ss} = 0.92\underline{7}\,\frac{\text{rad}}{\text{s}}$	–
Pinion angular impulse	$L_p = I_p\omega_{p,ss}$ $\rightarrow L_p = 0.40\underline{1}\,\frac{\text{lbm}\cdot\text{in}^2}{\text{s}}$	–
Mask angular impulse	$L_m = I_m\omega_{m,ss}$ $\rightarrow L_m = 5.4\underline{5}\,\frac{\text{lbm}\cdot\text{in}^2}{\text{s}}$	–
Shaft torque required to accelerate pinion and mask to steady-state speeds in one step (assuming constant application for the step duration)	$T = \frac{L_m + L_p}{\Delta t_s}$ $\rightarrow T = 73\underline{1}\,\frac{\text{lbm}\cdot\text{in}^2}{\text{s}^2}$ $\rightarrow \boxed{T = 0.21\underline{4}\text{ N}\cdot\text{m}}$	–

[36] Per SparkFun Electronics, Inc. (2019).
[37] We found experimentally that a rate of 125 hertz works well.

E. CALCULATION OF REQUIRED CLEARANCE BETWEEN PINION GEAR AND MASK GEAR

To facilitate a smooth interface between the pinion's and the mask's gear teeth, we include a clearance between the top lands of the teeth on one gear and the bottom lands between the teeth on the other gear. The required clearance is a function of machining and assembly tolerances elsewhere in the system that are summarized in Table 21. Most variance comes from generous hole location tolerances in the motor bracket.

Table 21. Summary of tolerances factored into gear clearance calculations.

Dimension	**Value**	**Notes**
Maximum axle cap clearance relative to secondary mirror cylinder	$A = .016''$	–
Maximum axis placement variation due to axle cap inner diameter clearance	$B = \frac{A}{2}$ $\rightarrow B = .008''$	[38]
Axis placement variation due to axle bolt hole clearance	$C = 0''$	[39]
Maximum deviation in parallelism between axle cap exterior and interior flat surfaces	$D = .03''$	–
Minimum cap inner diameter	$E = 3.597''$	–

[38] The maximum clearance between axle cap and secondary mirror cylinder is 0.016 inches, but the axle is centered, meaning the position variance is half.
[39] Countersink is self-centering regardless of hole clearance.

Dimension	**Value**	**Notes**
Axle angle due to maximum parallelism tolerance and minimum inner diameter	$F = \operatorname{atan}\left(\frac{D}{E}\right)$ $\rightarrow F = 0.0083\underline{4}$ rad	[40]
Maximum thickness of axle cap top	$G = .27''$	–
Maximum length of lower standoff	$H = .505''$	[41]
Maximum mask thickness	$I = .1875''$	[42]
Maximum transverse travel of mask top in presence of maximum axle angle	$J = (G + H + I)\sin F$ $\rightarrow J = .008\underline{0}''$	–
Maximum variation in motor bracket guide post placement	$K = .02''$	–
Maximum variation caused by motor bracket guide post hole diameter tolerance	$L = \frac{.012''}{2}$ $\rightarrow L = .006''$	[43]
Maximum variance caused by motor mount hole placement	$M = .008''$	–
Maximum variation caused by motor mount hole diameter tolerance	$N = \frac{.012''}{2}$ $\rightarrow N = .006''$	[44]
Maximum total variance	$O = B + C + J + K + L + M + N$ $\rightarrow \boxed{O = .05\underline{6}''}$	–

[40] Largest parallelism deviation combined with smallest distance over which it is achieved.
[41] (McMaster–Carr Supply Company, 2019)
[42] Per our own spec. See Section 4.4.4.
[43] Axis placement tolerance is half the diameter tolerance.
[44] Axis placement tolerance is half the diameter tolerance.

F. METHOD OF GENERATING GEAR PROFILES

A reliable method of representing accurate gear shapes in a vector drawing format proved surprisingly difficult to find. Several scripts found online did not properly account for mechanical interference effects of meshing gears. After some trial, we arrived at a successful and repeatable, if perhaps non-ideal, process:

1. Using a web browser, access Rainer Hessmer's Involute Spur Gear Builder utility, available at http://hessmer.org/gears/InvoluteSpurGearBuilder.html.
2. Set parameters per Table 22.
3. Press **Update**. When processing is complete, your output should show one of the two gears shown in Figure 123.
4. Press **Generate DXF**.
5. Press **Download DXF**.
6. Save the file to disc.
7. Open Inkscape.[45] We used Inkscape 0.92.3 64-bit on Windows.
8. Press **Ctrl+O** (alternatively, access **File > Open**) and open the DXF file created in step 6. A DXF Input window will appear.
9. Set the parameters in the DXF Input window to match Table 23. Leave other parameters at their default values.
10. Press **OK**. The gear shape should appear on the canvas.

[45] Unfortunately, the DXF output from the Involute Spur Gear Builder cannot be interpreted by most vector applications. Inkscape is one of the few that can read the file, so we use it to convert the file into a more flexible format.

11. Press **Ctrl+Shift+R** (alternatively, access **Edit > Resize Page to Selection**). This will move the canvas boundaries to encompass the gear shape. Your window should now appear similar to Figure 124.
12. Press **Ctrl+Shift+S** (alternatively, access **File > Save As…**). A dialog box will appear.
13. Change the “Save as type” to “Encapsulated PostScript (*.eps)”.
14. Give the file an appropriate name and location, then press **Save**.
15. Using Adobe Illustrator, open the EPS file created in step 14. We used Adobe Illustrator 23.0.6.
16. Press **Ctrl+A** (alternatively, **Select > All**) to highlight the entire mask profile.
17. Press **Ctrl+F10** (alternatively, **Window > Stroke**) to open up the Stroke window.
18. In the Stroke window, set the size to 0.001 pt. (We do this so that the laser cutter will cut these lines rather than engrave them.)
19. With the gear profile still selected, access **Object > Path > Simplify**. A settings prompt will appear.
20. In the Simplify window, set Curve Precision to 100% and Angle Threshold to 0°, then press **OK**. (We simplify the profile to remove redundant nodes that can slow down the laser cutting process.)
21. With the gear profile still selected, access **Object > Artboards > Fit to Selected Art**. This will expand the canvas to cover the entire gear, allowing the laser cutter to process the figure’s complete geometry.
22. Save your file using **Ctrl+S** or by selecting **File > Save**. The Save window may appear.

23. If the Save window appears, give the file an appropriate name and press the **Save** button.

Now that we have a representation of gear teeth in an Adobe Illustrator format, we can treat this file as a template. Creating new geared mask definitions is as simple as copying the Wheel 1 template and combining the copy with an aperture pattern. See Appendix G for more information on this process.

Table 22. Parameters used with Rainer Hessmer's Involute Spur Gear Builder to produce gear shapes for this project. In this table, Wheel 1 is the mask-side gear and Wheel 2 is the motor-side pinion.

Parameter name	**Value**	**Units**[46]
Circular pitch	0.5236	in. / tooth
Pressure angle	20	degrees
Clearance	0.056	in.
Backlash	0.02	in.
Profile shift	0	in.
Wheel 1 tooth count	72	teeth
Wheel 1 center hole diameter	0.25	in.
Wheel 2 tooth count	17	teeth
Wheel 2 center hole diameter	0.1869	in.
Show	see note[47]	–

[46] The utility enforces no particular length unit as long as the selection is consistent. The units in this column are correct in the context this project.
[47] Select either "Wheel 1 Only" or "Wheel 2 Only" depending on which gear you want to create.

Parameter name	**Value**	**Units**[46]
Rotation steps per tooth angle	10	–
Number of segments per 360 degrees of rotation	90	–

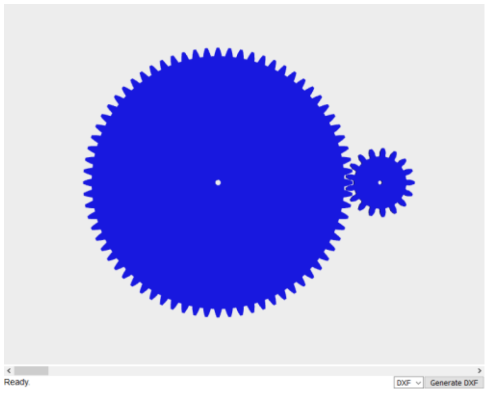
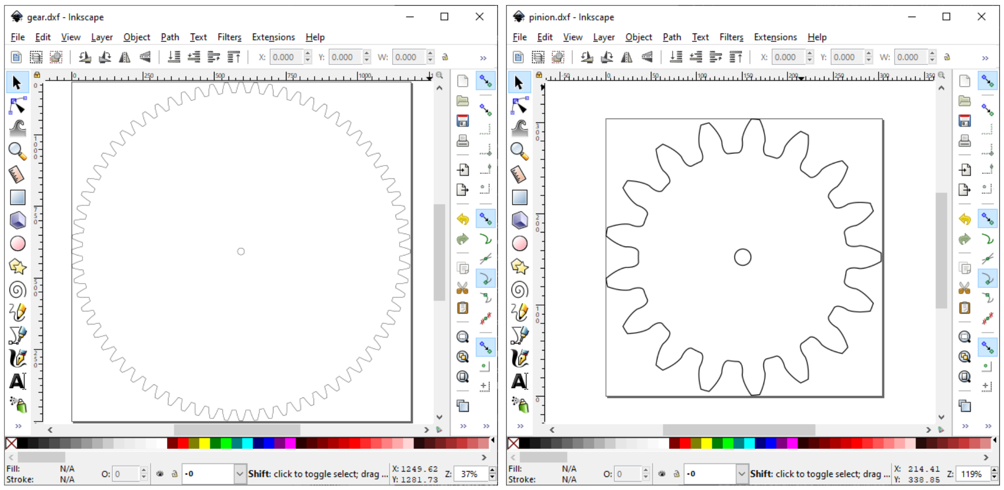


Figure 123. Involute Spur Gear Builder display after completing Step 3 with "Wheel 1 and Wheel 2" selected as the "Show" parameter. For the gears to appear as large as they do here, you will need to zoom in by sliding the gray scrollbar to the left.

Table 23. DXF Input parameters for Inkscape.

Parameter	**Value**
Method of scaling	Manual scale
Manual scale factor	25.4
Manual x-axis origin	0.0
Manual y-axis origin	0.0


gear.dxf - Inkscape
File Edit View Layer Object Path Text Filters Extensions Help
pinion.dxf - Inkscape
File Edit View Layer Object Path Text Filters Extensions Help

Fill: N/A
Stroke: N/A
O: 0
Shift: click to toggle select; drag ...
X: 1249.62
Y: 1281.73
Z: 37%

Fill: N/A
Stroke: N/A
O: 0
Shift: click to toggle select; drag ...
X: 214.41
Y: 338.85
Z: 119%

Figure 124. Result of performing step 11 for Wheel 1, left, and Wheel 2, right.

G. CONVERTING MASK IMAGES TO A LASER-CUTTER FORMAT

Just as the gear tooth profiles need to be converted to a laser-cutter-compatible format, so too must the mask images. The mask images begin in a raster format, which describes graphics using pixel values. This format has no notion of lines or curves, so this context must be communicated another way. In general, one would provide this information by generating vector files from scratch, but this approach is impractical for the project because the boundaries of many masks have no convenient mathematical representation.[48]

Thankfully, tools exist that interpret raster images and suggest appropriate vector-based curves. This operation is not a true conversion from raster to vector because the two formats describe fundamentally different data. Still, the process is good enough for our purposes.

We performed the following steps using Adobe Illustrator 23.0.6.

1. Open the PNG version of the mask in Adobe Illustrator. (This PNG would have been created by running the makeApertures.m script described in Appendix L.)
2. Open the Image Trace window by accessing **Window > Image Trace**.
3. In the Image Trace window, press the arrow next to "Advanced".
4. Change settings to match Table 24.
5. Press **Trace**.
6. Expand the trace with **Object > Image Trace > Expand**.

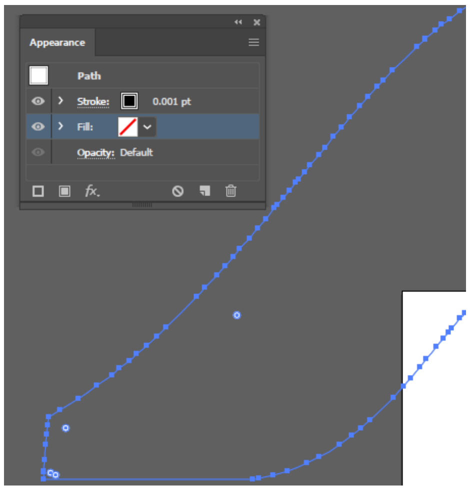
[48] Fourier transforms run on a computer are almost always discrete fast Fourier transforms that have no notion of a continuous signal. The function accepts discrete values as inputs and produces discrete values as outputs. In this project, we provide the discrete inputs in matrix form, where each element of the matrix describes the opacity at a small region of a mask. These matrices need not be formed using any particular mathematical strategy.

7. Click off the mask image to deselect it.
8. For each black region in the image, click the region using the Direct Selection Tool (white cursor), then delete the region by pressing the **Delete** key on your keyboard.
9. Open the Appearance window by pressing **Shift+F6** or via **Window > Appearance**.
10. For each white region in the image, click the region, then, using the Appearance window, change the Stroke to a 0.001-pt black line and the Fill to transparency (denoted by a red slash). See Figure 125.
11. Resize the artboard bounds to the extents of the mask shape via **Object > Artboards > Fit to Artwork Bounds**. See Figure 126.
12. Save the result via **Ctrl+S** or **File > Save**.

The resulting file can now be combined with the gear output from Appendix F to form the basis for the cut pattern.

Table 24. Image Trace parameters for importing mask images in Adobe Illustrator.

Parameter	**Value**
Paths	95%
Corners	95%
Noise	2 px
Create	☑ Fills (not Strokes)


Appearance

Path

Stroke: 0.001 pt

Fill:

Opacity: Default

Figure 125. How the Adobe Illustrator interface might appear after performing step 10.

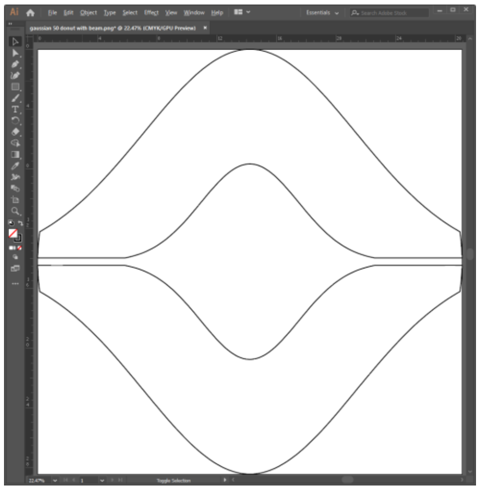


Figure 126. Result of step 11. (Contour is accentuated in this figure for printing purposes.)

H. MASK ROTATOR SOFTWARE AND ELECTRICAL INTERFACES

H.1 Mask rotator software interface

As discussed in Section 4.6, the mask rotator communicates using a UART serial interface over USB. Table 25 is a complete list of commands and associated actions. The code itself is included in Appendix M.

In this table, the Cmd column represents the character or characters sent to the mask rotator. The Return column is what the mask rotator sends back. For example, if the mask rotator receives an `s` (“stop”) command, it will send back a reciprocal `s` to indicate the rotator read the original command correctly.

- Some commands have special formats or significance that is described in a note following the table.
- All return codes are followed by one carriage return and one newline character (`\r\n`).
- The program begins in “absolute” mode. (See entries for `r` and `a`.)
- The interface operates at 19200 baud.

Table 25. Index of mask rotator commands.

Cmd	**Name**	**Action**	**Return**
`a`	enter absolute mode	Enters absolute mode, causing targets to be interpreted with respect to the index position	`a`

Cmd	**Name**	**Action**	**Return**
`b`	backward	Rotates mask backward continuously	`b`
`f`	forward	Rotates mask forward continuously	`f`
`g`(1)	go to	Sets a new target position for the mask	`g`(1) representing the new absolute mask target position
`i`	locate index	Triggers an indexing operation (see Section 4.6.2)	`i` initially, then either `I` if index was acquired or ~ if index was not acquired
`p`	get position	Gets current mask position	`p` (1) representing the current absolute mask position
`r`	enter relative mode	Enters relative mode, causing targets to be interpreted as changes in position	`r`
`s`	stop	Stops mask rotation	`s`
`t`	get target	Gets current mask target position	`t` (1) representing the current absolute mask target position

Cmd	Name	Action	Return
`z`	set zero	Stops mask rotation and sets current position as the new zero point	`z`
`?`	ping	Takes no action (used for verifying communication)	`!`
(else)	N/A	Unrecognized command	`x`

(1) A positive, negative, or zero integer representing degrees multiplied by 100. For example, the characters `-6311` represent −63.11 degrees. The precision of this number should not be confused for its accuracy, which is governed by the mechanical system and is much worse. The range of valid integers is $[-2^{-31}, 2^{31}-1]$, or $[-2147483648, 2147483647]$. This is enough to represent about 59,652 full rotations on either side of zero, which should be more than enough travel for any reasonable application. The program holds some numbers in a floating-point format, so precision may degrade slightly for extremely large integers toward the limits of this range.

H.2 Mask rotator wiring

Tables 26 and 27 show how to wire the stepper motor and Hall switch to the Arduino Motor Shield in the configuration the mask rotator program expects.

Table 26. Hall switch electrical connections.

Hall switch label	Arduino pin
SIG	5
VCC	4
GND	GND

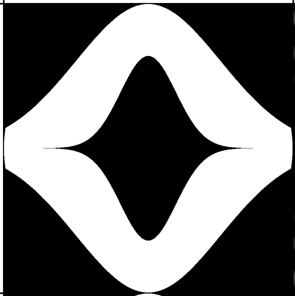
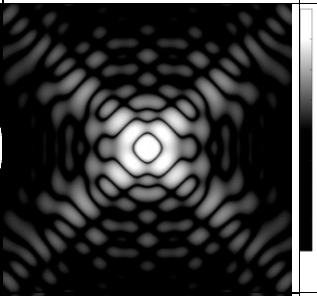
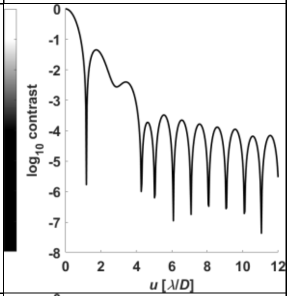
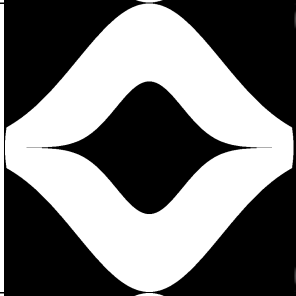
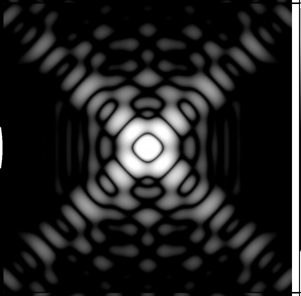
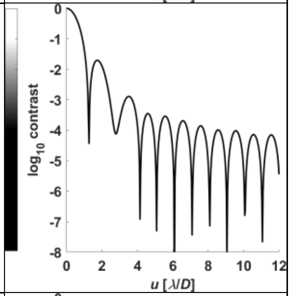
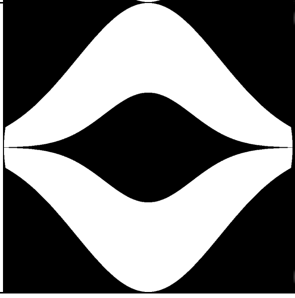
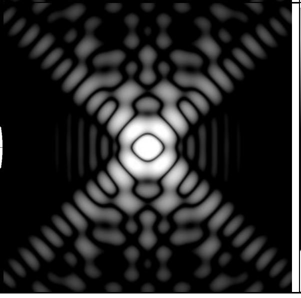
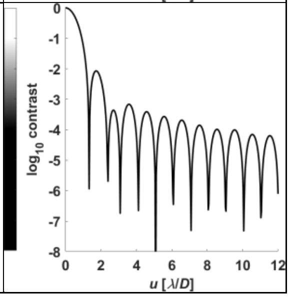
Table 27. Stepper motor electrical connections.

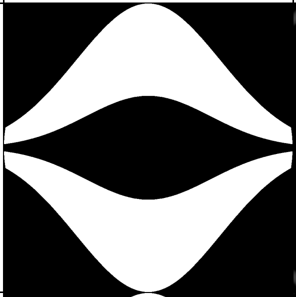
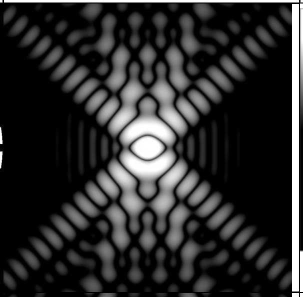
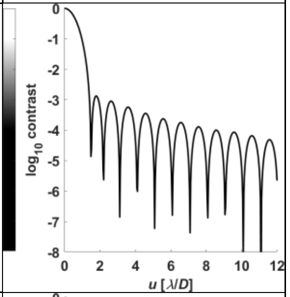
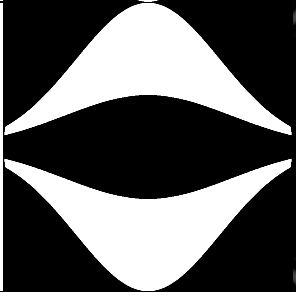
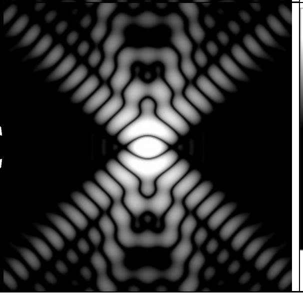
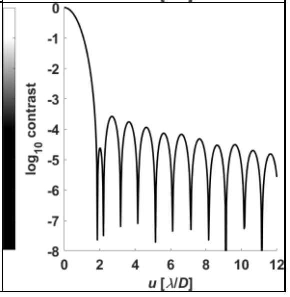
Stepper wire	Arduino terminal
A (red)	A+
B (yellow)	B+
C (green)	A−
D (blue)	B−

I. STUDY OF SECONDARY BROADNESS IN A GAUSSIAN DONUT

Table 28 explores the effect of changing the broadness of a secondary Gaussian obstruction relative to that of the primary shape. This follows an earlier discovery in Section 3.4.1 that an outer broadness factor of $p = 0.50$ performs well with respect to our contrast and working angle targets.

Table 28. Study of different broadness choices for the secondary Gaussian while holding the broadness of the primary constant.

Outer p_1	**Inner p_2**	**Aperture shape**	**Point spread function**	**Horizontal PSF cut**
0.50	0.30			
0.50	0.50			
0.50	0.80			

Outer p_1	Inner p_2	Aperture shape	Point spread function	Horizontal PSF cut
0.50	1.20			
0.50	1.60			

Of these options, the aperture with secondary broadness $p_2 = 1.20$ has the most promising simulated response. Along the horizontal axis, the aperture's power spectrum reaches our target contrast threshold at a small working angle of $1.38\ \lambda/D$ and forever maintains this contrast as the angle increases. While some other configurations achieve the target contrast at smaller angles, they do so only at narrow valleys of destructive interference. The configuration with $p_2 = 1.60$ does not have this issue but reaches the target contrast at a greater angle ($1.59\ \lambda/D$) than the $p_2 = 1.20$ configuration.

J. INDEX QUALITY TEST DATA

This section contains data collected while testing the angular accuracy of the mask rotation mechanism. See Section 5.6 for more details.

Mask rotating in forward direction

Table 29. Data collected during index quality testing with mask running in the forward direction. The leftmost four columns contain raw data and the remaining columns contain derived values.

Hilo time [ms]	Hilo pos [deg]	Lohi time [ms]	Lohi pos [deg]	Gap [deg]	Hilo delta [deg]	Lohi delta [deg]
2017	107.10	2190	116.45	9.35		
5407	287.30	5578	296.22	8.92	180.20	179.77
8798	467.50	8962	476.00	8.50	180.20	179.78
12183	647.27	12352	656.20	8.93	179.77	180.20
15574	827.47	15740	836.40	8.93	180.20	180.20
18958	1007.25	19131	1016.60	9.35	179.78	180.20
22351	1187.45	22521	1196.37	8.92	180.20	179.77
25738	1367.22	25906	1376.15	8.93	179.77	179.78
29126	1547.42	29295	1556.35	8.93	180.20	180.20
32513	1727.20	32682	1736.12	8.92	179.78	179.77
35903	1907.40	36070	1916.32	8.92	180.20	180.20
39292	2087.60	39460	2096.52	8.92	180.20	180.20
42677	2267.37	42848	2276.30	8.93	179.77	179.78
46069	2447.57	46237	2456.50	8.93	180.20	180.20
49456	2627.35	49624	2636.27	8.92	179.78	179.77
52847	2807.55	53009	2816.05	8.50	180.20	179.78
56233	2987.32	56401	2996.25	8.93	179.77	180.20
59617	3167.10	59787	3176.45	9.35	179.78	180.20
63008	3347.30	63176	3356.22	8.92	180.20	179.77
66395	3527.07	66565	3536.42	9.35	179.77	180.20
69783	3707.27	69954	3716.20	8.93	180.20	179.78
73173	3887.47	73342	3896.40	8.93	180.20	180.20
76563	4067.67	76730	4076.17	8.50	180.20	179.77
79948	4247.45	80118	4256.37	8.92	179.78	180.20
83336	4427.22	83507	4436.57	9.35	179.77	180.20

86727	4607.42	86892	4616.35	8.93	180.20	179.78
90115	4787.62	90283	4796.12	8.50	180.20	179.77
93500	4967.40	93671	4976.32	8.92	179.78	180.20
96892	5147.60	97060	5156.52	8.92	180.20	180.20
100280	5327.37	100447	5336.30	8.93	179.77	179.78
103671	5507.57	103837	5516.50	8.93	180.20	180.20
107057	5687.35	107225	5696.27	8.92	179.78	179.77
110444	5867.55	110612	5876.47	8.92	180.20	180.20
113832	6047.32	114000	6056.25	8.93	179.77	179.78
117219	6227.52	117388	6236.45	8.93	180.20	180.20
120609	6407.30	120772	6416.22	8.92	179.78	179.77
123995	6587.50	124164	6596.42	8.92	180.20	180.20
127387	6767.27	127553	6776.20	8.93	179.77	179.78
130773	6947.47	130942	6956.40	8.93	180.20	180.20
134160	7127.25	134331	7136.17	8.92	179.78	179.77
137551	7307.45	137719	7316.37	8.92	180.20	180.20
140936	7487.22	141105	7496.15	8.93	179.77	179.78
144325	7667.42	144497	7676.35	8.93	180.20	180.20
147715	7847.62	147882	7856.12	8.50	180.20	179.77
151103	8027.40	151269	8036.32	8.92	179.78	180.20
154488	8207.17	154660	8216.52	9.35	179.77	180.20
157878	8387.37	158048	8396.30	8.93	180.20	179.78
161266	8567.15	161437	8576.50	9.35	179.78	180.20
164653	8747.35	164825	8756.27	8.92	180.20	179.77
168045	8927.55	168213	8936.47	8.92	180.20	180.20
171432	9107.32	171600	9116.25	8.93	179.77	179.78
174817	9287.10	174988	9296.45	9.35	179.78	180.20
178208	9467.30	178377	9476.22	8.92	180.20	179.77
				Min	179.7700	179.7700
				Max	180.2000	180.2000
				Avg	180.0038	179.9956

Mask rotating in reverse direction

Table 30. Data collected during index quality testing with mask running in the reverse direction. The leftmost four columns contain raw data and the remaining columns contain derived values.

Hilo time [ms]	Hilo pos [deg]	Lohi time [ms]	Lohi pos [deg]	Diff [deg]	Hilo delta [deg]	Lohi delta [deg]
1672	9473.67	1834	9465.17	-8.50		
5055	9293.90	5221	9284.97	-8.93	-179.77	-180.20
8443	9114.12	8612	9105.20	-8.92	-179.78	-179.77
11833	8933.92	11998	8925.00	-8.92	-180.20	-180.20
15224	8753.72	15390	8744.80	-8.92	-180.20	-180.20
18611	8573.95	18778	8565.02	-8.93	-179.77	-179.78
22001	8393.75	22162	8385.25	-8.50	-180.20	-179.77
25387	8213.97	25554	8205.05	-8.92	-179.78	-180.20
28776	8033.77	28940	8025.27	-8.50	-180.20	-179.78
32164	7854.00	32327	7845.07	-8.93	-179.77	-180.20
35554	7673.80	35716	7664.87	-8.93	-180.20	-180.20
38939	7494.02	39103	7485.10	-8.92	-179.78	-179.77
42328	7313.82	42494	7304.90	-8.92	-180.20	-180.20
45718	7133.62	45883	7125.12	-8.50	-180.20	-179.78
49109	6953.42	49273	6944.92	-8.50	-180.20	-180.20
52495	6773.65	52659	6765.15	-8.50	-179.77	-179.77
55882	6593.87	56049	6584.95	-8.92	-179.78	-180.20
59270	6413.67	59437	6404.75	-8.92	-180.20	-180.20
62662	6233.47	62823	6224.97	-8.50	-180.20	-179.78
66045	6053.70	66208	6045.20	-8.50	-179.77	-179.77
69437	5873.50	69598	5865.00	-8.50	-180.20	-180.20
72820	5694.15	72990	5684.80	-9.35	-179.35	-180.20
76215	5513.52	76378	5505.02	-8.50	-180.63	-179.78
79597	5333.75	79764	5324.82	-8.93	-179.77	-180.20
82990	5153.55	83154	5145.05	-8.50	-180.20	-179.77
86375	4973.77	86539	4965.27	-8.50	-179.78	-179.78
89765	4793.57	89930	4785.07	-8.50	-180.20	-180.20
93152	4613.80	93318	4604.87	-8.93	-179.77	-180.20
96543	4433.60	96704	4425.10	-8.50	-180.20	-179.77
99929	4253.82	100092	4245.32	-8.50	-179.78	-179.78
103320	4073.62	103484	4065.12	-8.50	-180.20	-180.20
106704	3893.85	106869	3884.92	-8.93	-179.77	-180.20
110092	3713.65	110260	3705.15	-8.50	-180.20	-179.77

113482	3533.87	113648	3524.95	-8.92	-179.78	-180.20
116872	3353.67	117037	3344.75	-8.92	-180.20	-180.20
120254	3173.90	120422	3164.97	-8.93	-179.77	-179.78
123647	2993.70	123813	2984.77	-8.93	-180.20	-180.20
127033	2813.92	127196	2805.00	-8.92	-179.78	-179.77
130426	2633.72	130588	2625.22	-8.50	-180.20	-179.78
133809	2453.95	133976	2445.02	-8.93	-179.77	-180.20
137200	2273.75	137364	2264.82	-8.93	-180.20	-180.20
140586	2093.97	140752	2085.05	-8.92	-179.78	-179.77
143978	1913.77	144139	1905.27	-8.50	-180.20	-179.78
147362	1734.00	147529	1725.07	-8.93	-179.77	-180.20
150751	1553.80	150918	1544.87	-8.93	-180.20	-180.20
154136	1374.02	154302	1365.10	-8.92	-179.78	-179.77
157531	1193.82	157692	1185.32	-8.50	-180.20	-179.78
160917	1013.62	161081	1005.12	-8.50	-180.20	-180.20
164309	833.42	164473	824.92	-8.50	-180.20	-180.20
167694	653.65	167859	645.15	-8.50	-179.77	-179.77
171083	473.87	171248	464.95	-8.92	-179.78	-180.20
174470	293.67	174634	285.17	-8.50	-180.20	-179.78
177861	113.47	178023	104.97	-8.50	-180.20	-180.20
				Min	-180.6300	-180.2000
				Max	-179.3500	-179.7700
				Avg	-180.0038	-180.0038

Repeated indexing operations

Table 31. Data from repeated indexing operations. The leftmost three columns are raw data and the remaining column is derived.

#	Time [ms]	Angle [deg]	Delta [deg]
1	3715	180.63	
2	7929	360.61	179.98
3	12109	540.49	179.88
4	16304	720.38	179.89
5	20492	900.36	179.98
6	24690	1080.46	180.10
7	28900	1260.55	180.09
8	33112	1440.54	179.99
9	37314	1620.52	179.98
10	41529	1800.51	179.99
11	45732	1980.50	179.99
12	49941	2160.59	180.09
13	54137	2340.47	179.88
14	58341	2520.67	180.20
15	62538	2700.45	179.78
16	66750	2880.65	180.20
17	70943	3060.53	179.88
18	75149	3240.62	180.09

#	Time [ms]	Angle [deg]	Delta [deg]
19	79345	3420.40	179.78
20	83549	3600.60	180.20
21	87745	3780.37	179.77
22	91959	3960.57	180.20
23	96155	4140.35	179.78
24	100368	4320.44	180.09
25	104562	4500.32	179.88
26	108772	4680.52	180.20
27	112964	4860.51	179.99
28	117165	5040.50	179.99
29	121361	5220.27	179.77
30	125567	5400.47	180.20
31	129763	5580.25	179.78
32	133989	5760.66	180.41
33	138189	5940.65	179.99
34	142388	6120.53	179.88
35	146596	6300.62	180.09
36	150792	6480.40	179.78
37	154981	6660.39	179.99
38	159199	6840.59	180.20
39	163391	7020.47	179.88
40	167602	7200.56	180.09
41	171793	7380.34	179.78
42	175986	7560.33	179.99
43	180197	7740.52	180.19
44	184409	7920.51	179.99
45	188610	8100.50	179.99
46	192821	8280.59	180.09
47	197024	8460.37	179.78
48	201234	8640.57	180.20
49	205439	8820.45	179.88
50	209649	9000.54	180.09
51	213853	9180.42	179.88
52	218064	9360.52	180.10
53	222260	9540.40	179.88
54	226469	9720.60	180.20
55	230668	9900.37	179.77
56	234896	10080.57	180.20

#	Time [ms]	Angle [deg]	Delta [deg]
57	239091	10260.35	179.78
58	243315	10440.55	180.20
59	247514	10620.43	179.88
60	251720	10800.52	180.09
61	255914	10980.41	179.89
62	260124	11160.50	180.09
63	264318	11340.49	179.99
64	268533	11520.58	180.09
65	272732	11700.25	179.67
66	276942	11880.56	180.31
67	281132	12060.33	179.77
68	285336	12240.42	180.09
69	289553	12420.41	179.99
70	293771	12600.61	180.20
71	297966	12780.39	179.78
72	302173	12960.69	180.30
73	306374	13140.58	179.89
74	310588	13320.67	180.09
75	314794	13500.34	179.67
76	319012	13680.64	180.30
77	323215	13860.42	179.78
78	327424	14040.62	180.20
79	331601	14220.29	179.67
80	335810	14400.59	180.30
81	339996	14580.37	179.78
82	344209	14760.57	180.20
83	348406	14940.45	179.88
84	352618	15120.54	180.09
85	356822	15300.42	179.88
86	361031	15480.52	180.10
87	365214	15660.29	179.77
88	369421	15840.60	180.31
89	373622	16020.37	179.77
90	377826	16200.57	180.20
91	382027	16380.35	179.78
92	386239	16560.44	180.09
93	390433	16740.33	179.89
94	394645	16920.52	180.19

# 	Time [ms]	Angle [deg]	Delta [deg]
95	398837	17100.41	179.89
96	403044	17280.50	180.09
97	407241	17460.27	179.77
98	411449	17640.48	180.21
99	415647	17820.46	179.98

# 	Time [ms]	Angle [deg]	Delta [deg]
100	419861	18000.56	180.10
		Min	179.67
		Max	180.41
		Avg	179.9993

K. DIFFRACTION SIMULATION CODE

K.1 Prerequisites

Running the diffraction simulation software requires a copy of MATLAB. We used MATLAB R2018a. No toolboxes are required.

The most recent version of code—potentially including features or bugfixes added after this paper was published—can be downloaded from https://github.com/e-foley/FraunhoferSim.

The software operates on image files that represent apertures. Two such images representing the bowtie mask and the beamed bowtie mask are included in the software repository linked above. Others can be generated by running the makeApertures.m script described in Appendix L. Please note that the makeApertures.m script has slightly different prerequisites than the diffraction simulation code itself.

K.2 Architecture and usage

The diffraction code generates and manipulates two key objects:

- A **Psf** object represents a point spread function for an aperture. The object holds information about the distribution of energy within the point spread function along with meta-information about the object, including its resolution and angular bounds. Spatial information within the Psf object is normalized to λ/D.
- A **StarView** object represents the convolution of a point spread function with stars. Like a Psf object, a StarView also holds information about itself, including its resolution and angular bounds. Unlike a Psf, however, a StarView operates in angles of arcseconds rather than normalized units. In order to contextualize the

Psf's normalized units as arcseconds, the StarView must know the diameter of the telescope and what wavelength of light is being observed.

Once a Psf or StarView object has been created, it can be plotted using an appropriate plotting function. Psfs are compatible with psfGetImage(), which creates an image of the point spread function without axes or other labels; psfPlot(), which creates a formatted plot of the point spread function; and psfCut(), which creates a formatted plot of a *u*-axis cut through the point spread function. StarView objects can be plotted using svGetImage() and svPlot(), which behave almost identically to their psfGetImage() and psfPlot() counterparts.

Functions that plot Psfs and StarViews accept the object to plot as well as an assortment of formatting parameters. These formatting parameters differ function-to-function and are generally combined into structures with many different fields. For example, psfPlot() accepts an ImagescProps struct and an IoProps struct. The fields within the ImagescProps struct can either be set manually or generated using getPsfPlotDefaults(). IoProps determines how and where the image is saved to disc, and its fields will be set manually by the user.

The functions psfPlot() and psfCut() can operate on multiple Psf objects simultaneously, in which case the Psfs will be combined on the same plot. This is useful for comparisons. To provide multiple Psf objects to these functions, place the objects in an array.

All the operations are demonstrated in a script called demo.m, which is designed to run successfully out of the box. Use this script as a basis for further development. To configure and perform individual operations, run runAperturePlot.m, runSinglePsf.m, runMultiPsf.m or runStarView.m.

Figure 127 offers a diagram of the architecture described in this section.

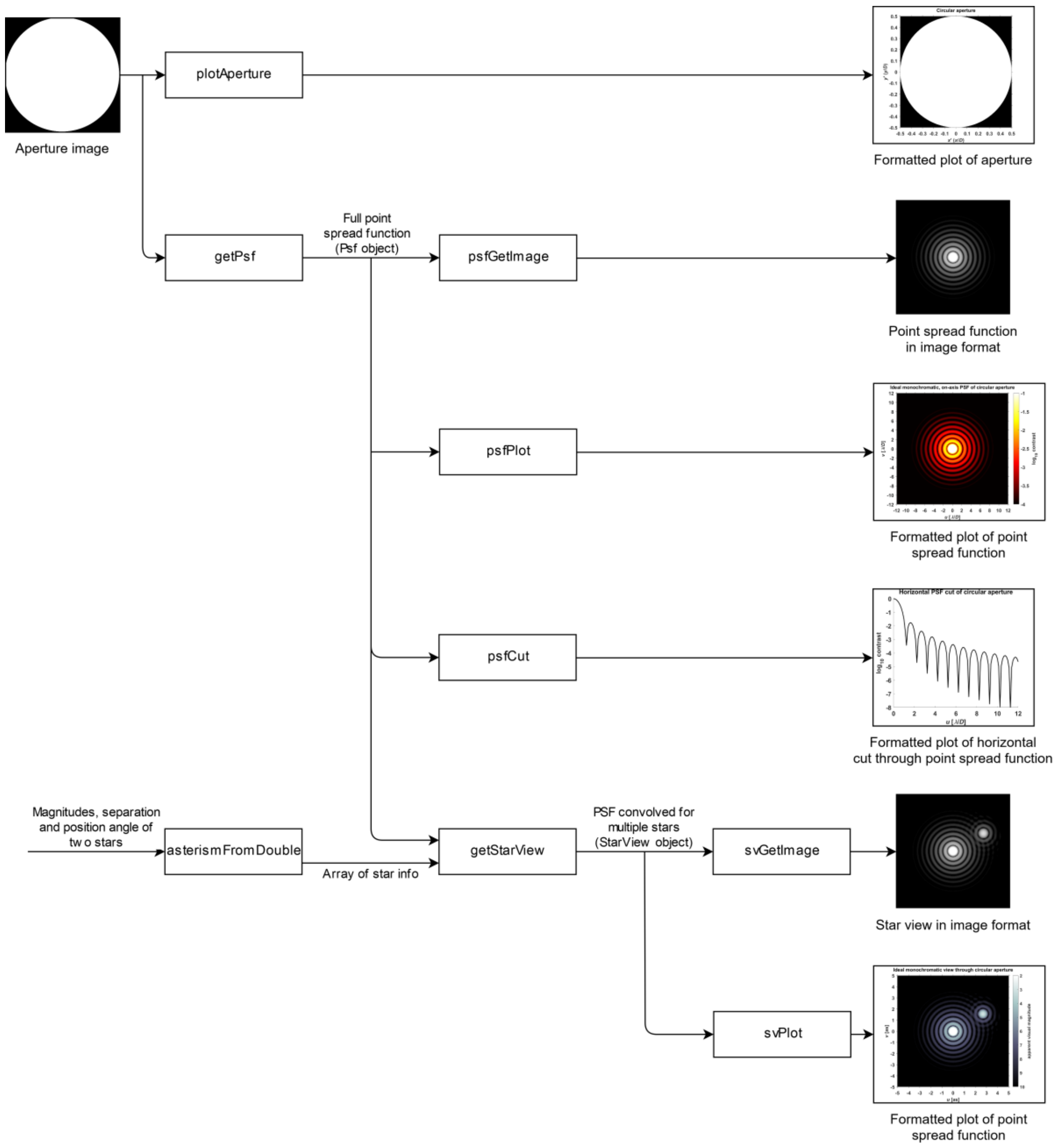

Aperture image
plotAperture
Formatted plot of aperture
getPsf
Full point spread function (Psf object)
psfGetImage
Point spread function in image format
psfPlot
Formatted plot of point spread function
psfCut
Formatted plot of horizontal cut through point spread function
Magnitudes, separation and position angle of two stars
asterismFromDouble
Array of star info
getStarView
PSF convolved for multiple stars (StarView object)
svGetImage
Star view in image format
svPlot
Formatted plot of point spread function

Figure 127. Architectural layout of diffraction simulation functions. White textboxes represent functions and are each labeled with the function's name. (Various formatting-related parameters accepted by the functions are omitted for clarity.)

K.3 Code

Files are listed alphabetically by file name. For code describing functions related to aperture creation, consult Appendix L.3 on page 218 instead.

asFromLd.m

```
% Calculates the factor that converts lambda/diameter into arcseconds for the
% given wavelength and aperture diameter.
%
% wavelength_nm  The wavelength of light (nanometers)
% diameter_in    The aperture diameter (inches)
%
% as             The angle (arcseconds) corresponding to 1 lambda/diameter

function [as] = asFromLd(wavelength_nm, diameter_in)
    % Convert both arguments to meters to find angle in radians. (Small
    % angle approximation is used.)
    ld_rad = (wavelength_nm / 1e9) / (diameter_in * (1/12) * (1/3.28));

    % Convert angle from radians to arcseconds.
    as = ld_rad * 3600 * (360/(2*pi));
end
```

asterismFromDouble.m

```
% Converts information describing a double star system into an array of Star
% objects for use in other functions. The primary star will be placed at
% (u, v) = (0, 0); the secondary star will be placed according to the separation
% and position angle arguments.
%
% separation_as  The angular separation of the stars (arcseconds)
% app_vis_mags   Apparent visual magnitudes of the stars [m1,m2]. Larger numbers
%                correspond to dimmer stars.
% pa_deg         Position angle of second star relative to first star (degrees).
%                Angles of 0, 90, 180, 270 degrees will place second star along
%                -v, +u, +v, and -u.
%
% asterism       Star objects representing the system [Star1,Star2]

function [asterism] = asterismFromDouble(separation_as, app_vis_mags, pa_deg)

% Construct the Star objects. 90-degree offsets within trig functions align us
% with astronomical conventions for star placement, with 0 degrees being north
% (down), 90 degrees being east (right), and so forth.
star1 = Star([0 0], app_vis_mags(1));
star2 = Star(separation_as * [cosd(pa_deg-90) sind(pa_deg-90)], app_vis_mags(2));

asterism = [star1 star2];

end
```

CutProps.m

```
% Properties related to plotting horizontal cuts through a point spread
% function.

classdef CutProps
    properties
        % Title of the plot
        plot_title = 'Horizontal PSF cut'

        % Nominal plot size (pixels) [width,height]. MATLAB may adapt the
        % numbers in unexpected ways.
        nominal_plot_size_px = [620 528]

        % Fudge factor for vertically aligning plot title
        extra_title_margin = 0.14

        % u-axis label, axis limits [low,high], and tick spacing
        u_title = '{\itu} [{\it\lambda}/{\itD}]'
        u_limits = [0 12]
        u_spacing = 2

        % w-axis label, axis limits [low,high], and tick spacing
        w_title = 'log_1_0 contrast'
        w_limits = [-8 0]
        w_spacing = 1

        % Whether to show color bars beside the plot
        show_color_bars = false

        % Color maps to use for color bars {map1,map2,...,mapN}
        color_maps = {}

        % w-axis limits over which color bar range is applied
        c_limits = [-4 -1]

        % Color bar tick spacing
        c_spacing = 1

        % Legend entries describing plotted lines {label1,label2,...,labelN}
        labels = {'Aperture'}

        % Line colors {[r1,g1,b1],[r2,g2,b2],...,[rN,gN,bN]}
        line_colors = [0 0 0]

        % Thickness of cut profile lines (points)
        cut_line_thickness_pt = 2

        % Font size of all text in figure (points)
        font_size_pt = 14

        % Whether to draw a horizontal "contrast target" line on the plot
        show_target = true

        % The place along the w-axis to draw a contrast target line
        target = -2.6

        % Thickness of target line (points) and its color [r,g,b]
        target_line_thickness_pt = 1
        target_line_color = [0.4 0.4 0.4]
    end
end
```

demo.m

Note that demo.m invokes makeApertures(), which is defined in Appendix L.

```
% Demonstration script that shows how to generate mask figures and diffraction
% pattern figure for the C11 aperture and bowtie mask. Plots will be created as
% figures and saved in Portable Network Graphics (PNG) format in a "plots"
% folder.
%
% Prerequisites: folders called "apertures" and "plots" exist in the same
% directory as this script; "apertures" folder contains at least a file called
% "bowtie.png".
%
% Script tested using MATLAB R2018a with the Image Processing Toolbox.

% Clear existing variables and figures for repeatability reasons.
clearvars;
close all;

% Define miscellaneous input and output variables.
input_prefix = 'apertures/';
output_prefix = 'plots/';
io_props = IoProps;
io_props.save_png = true;
io_props.save_eps = false;

% Generate our aperture shapes, including the C11.
makeApertures;

% Load images corresponding to the C11 aperture and bowtie mask.
c11_aperture = imread([input_prefix 'c11.png']);
bowtie_mask = imread([input_prefix 'bowtie.png']);

% Generate plots of the C11 aperture and bowtie mask.
aperture_plot_props = getAperturePlotDefaults;
aperture_plot_props.plot_title = 'C11 aperture';
io_props.png_location = [output_prefix 'c11 aperture plot.png'];
plotAperture(c11_aperture, aperture_plot_props, io_props);
aperture_plot_props.plot_title = 'Bowtie mask';
io_props.png_location = [output_prefix 'bowtie mask plot.png'];
plotAperture(bowtie_mask, aperture_plot_props, io_props);

% Calculate PSFs for the C11 aperture and bowtie mask and store them in objects.
% (We will feed these objects into functions that plot them in different ways.)
aperture_scale = 0.25;  % Lower: faster execution, less accurate
fft_scale = 8;  % Higher: better resolution, slower processing
c11_psf = getPsf(c11_aperture, aperture_scale, fft_scale);
bowtie_psf = getPsf(bowtie_mask, aperture_scale, fft_scale);

% Plot the individual PSFs of the C11 aperture and bowtie mask.
psf_plot_props = getPsfPlotDefaults;
psf_plot_props.plot_title = 'Ideal monochromatic, on-axis PSF of C11 aperture';
io_props.png_location = [output_prefix 'c11 psf plot.png'];
psfPlot(c11_psf, psf_plot_props, io_props);
psf_plot_props.plot_title = 'Ideal monochromatic, on-axis PSF of bowtie mask';
```

```
io_props.png_location = [output_prefix 'bowtie psf plot.png'];
psfPlot(bowtie_psf, psf_plot_props, io_props);

% Plot horizontal cuts of our two PSFs.
cut_plot_props = CutProps;
cut_plot_props.labels = {'C11 aperture'};
io_props.png_location = [output_prefix 'c11 cut plot.png'];
psfCut(c11_psf, cut_plot_props, io_props);
cut_plot_props.labels = {'bowtie mask'};
io_props.png_location = [output_prefix 'bowtie cut plot.png'];
psfCut(bowtie_psf, cut_plot_props, io_props);

% Create a figure showing one PSF overlaid on the other. This is as simple as
% supplying both Psf objects and appropriate colors to psfPlot.
psf_plot_props.plot_title = 'PSF comparison';
% Show C11 in shades of fuchsia, bowtie in shades of green.
psf_plot_props.color_maps = {[1 0 1] .* gray(256), [0 1 0] .* gray(256)};
psf_plot_props.labels = {'C11 aperture', 'bowtie mask'};
io_props.png_location = [output_prefix 'c11 bowtie psf comparison plot.png'];
psfPlot([c11_psf bowtie_psf], psf_plot_props, io_props);

% Create a figure showing one PSF "cut plot" overlaid on the other. Again, this
% is as simple as supplying both Psf objects, colors and labels.
cut_plot_props.plot_title = 'PSF horizontal cut comparison';
cut_plot_props.show_color_bars = true;
cut_plot_props.color_maps = psf_plot_props.color_maps;
cut_plot_props.labels = psf_plot_props.labels;
cut_plot_props.line_colors = {[1 0 1], [0 1 0]};
io_props.png_location = [output_prefix 'c11 bowtie cut comparison plot.png'];
psfCut([c11_psf bowtie_psf], cut_plot_props, io_props);

% Create two objects that represent visualizing a binary system through the C11
% aperture and through the bowtie mask. This takes a few steps, but it's worth
% it. First, define the telescope diameter and light wavelength (so we know what
% lambda/D actually is).
telescope_diameter_in = 11;  % (inches)
wavelength_nm = 680;  % (nanometers)
% Then, define properties of our double-star system. Let's use Lambda Cygni.
% (Properties from https://en.wikipedia.org/wiki/Lambda_Cygni.)
separation_as = 0.77;  % (arcseconds)
app_vis_mags = [4.54 6.26];  % apparent visual magnitudes (not log-10)
pa_deg = 90;  % position angle (degrees) -- pretend it's 90 to align with mask
stars = asterismFromDouble(separation_as, app_vis_mags, pa_deg);
% Generate the actual StarView objects, which are used in analogous ways as Psf
% objects.
c11_sv = getStarView(stars, c11_psf, telescope_diameter_in, wavelength_nm);
bowtie_sv = getStarView(stars, bowtie_psf, telescope_diameter_in, wavelength_nm);

% Now we plot the StarView objects, setting up a few display properties first.
sv_plot_props = getStarViewPlotDefaults;
sv_plot_props.output_limits = [10 4];
sv_plot_props.plot_title = 'Monochromatic view of stars through C11 aperture';
io_props.png_location = [output_prefix 'c11 star view plot.png'];
svPlot(c11_sv, sv_plot_props, io_props);
sv_plot_props.plot_title = 'Monochromatic view of stars through bowtie mask';
io_props.png_location = [output_prefix 'bowtie star view plot.png'];
svPlot(bowtie_sv, sv_plot_props, io_props);
```

formatImagescPlot.m

```
% Formats an existing plot created using imagesc to apply scaling and standard
% graphical elements in a consistent manner.
%
% the_figure     Handle to the figure produced by imagesc
% imagesc_props  Plot formatting properties to apply

function formatImagescPlot(the_figure, imagesc_props)
s = imagesc_props;

set(the_figure, 'Position', [0 0 s.nominal_plot_size_px]);

% Axis operations seem redundant, but performance consistent only with them all.
axis on;
axis square;
axis equal;

% Configure axes.
xlim(s.field_limits(1,:));
ylim(s.field_limits(2,:));
xticks(s.field_limits(1,1):s.h_axis_tick_spacing:s.field_limits(1,2));
yticks(s.field_limits(2,1):s.v_axis_tick_spacing:s.field_limits(2,2));
set(gca, 'YDir', 'normal');
set(gca, 'TickDir', 'out');
xlabel(s.h_axis_title);
ylabel(s.v_axis_title);

% Construct and place plot title.
my_title = title(s.plot_title);
title_pos = get(my_title, 'Position');
set(my_title, 'Position', title_pos + [0 s.extra_title_margin 0]);

% Apply font across whole figure.
set(gca,'FontSize',s.font_size_pt,'fontWeight','bold');
set(findall(gcf,'type','text'),'FontSize',s.font_size_pt,'fontWeight','bold');

end
```

getAperturePlotDefaults.m

```
% Generates a default ImagescProps object configured for aperture plots. The
% object can be supplied to the plotAperture function.
%
% defaults  The default ImagescProps object configured for aperture plots.

function [defaults] = getAperturePlotDefaults
defaults = ImagescProps;
defaults.plot_title = '';
defaults.nominal_plot_size_px = [620 528];
defaults.extra_title_margin = 0.02;
defaults.field_limits = [-0.5 0.5; -0.5 0.5];
defaults.output_limits = [0 1];
defaults.h_axis_title = '{\itx}'' ({\itx}/{\itD})';
defaults.h_axis_tick_spacing = 0.1;
defaults.v_axis_title = '{\ity}'' ({\ity}/{\itD})';
defaults.v_axis_tick_spacing = 0.1;
defaults.labels = {};
defaults.show_color_bars = false;
defaults.color_maps = gray(256);
defaults.font_size_pt = 14;
end
```

getPsf.m

```
% Generates the characteristic point spread function corresponding to a given
% aperture. The function interprets the entirety of the aperture image,
% including any opaque padding, as the aperture for the purpose of calculating
% the aperture's size.
%
% aperture        The image representing the entirety of the aperture, where
%                 white pixels indicate transparent regions and black pixels
%                 indicate opaque areas. Grayscale values convey translucency.
% aperture_scale  The factor the function will use to scale the aperture image
%                 before the fast Fourier transform is taken. Smaller factors
%                 greatly improve speed at the cost of introducing aliasing
%                 noise into the point spread function.
% fft_scale       The dimensions the scaled aperture image will be padded to
%                 as a ratio of the scaled aperture image size. Larger fft_scale
%                 values improve point spread function resolution but
%                 significantly increase computation time.
%
% psf                 A Psf object representing the point spread function
%                     created by the aperture
% scaled_aperture_px  The dimensions of the aperture image after it was scaled
%                     scaled by aperture_scale (pixels) [height,width]
% fft_size_px         The dimensions of the fast Fourier transform output
%                     (pixels) [height,width]

function [psf, scaled_aperture_size_px, fft_size_px] = ...
    getPsf(aperture, aperture_scale, fft_scale)

psf = Psf;

% Convert to grayscale if necessary.
if (size(aperture, 3) > 1)
    aperture = rgb2gray(aperture);
end

% Scale dims of the mask/aperture. Scaling down allows FFT to use less memory.
% We rotate the matrix such that we can store the PSF with (u, v) indices.
scaled_aperture = imresize(aperture, aperture_scale);
scaled_aperture_size_px = size(scaled_aperture);
scaled_aperture = rot90(scaled_aperture, 3);
fft_size_px = fft_scale * [1 1] * max(size(scaled_aperture));

% Find FFT of this mask/aperture (not power spectrum yet), padding the FFT to
% dimensions of fft_size_px and placing zero-frequency component in the center
% of the image.
psf.data = fftshift(fft2(scaled_aperture, fft_size_px(1), fft_size_px(2)));

% Power spectrum is the square of the complex amplitude.
psf.data = abs(psf.data) .^ 2;

% With unity FFT scale, 1.0 lambda/D is lowest resolvable frequency; larger FFT
% scales give better resolution.
```

```
psf.pixels_per_ld = fft_scale;

% Calculating bounds is slightly tricky because fftshift places zero-frequency
% to lower-right of center when dimension is even.
psf.ld_bounds(:,2) = floor((fft_size_px' - 1) / 2);
psf.ld_bounds(:,1) = psf.ld_bounds(:,2) - fft_size_px' + 1;
psf.ld_bounds = psf.ld_bounds / fft_scale;

end
```

getPsfPlotDefaults.m

```
% Generates a default ImagescProps object configured for PSF plots. The object
% can be supplied to the psfPlot function.
%
% defaults  The default ImagescProps object configured for PSF plots.

function [defaults] = getPsfPlotDefaults
defaults = ImagescProps;
defaults.plot_title = 'Power spectrum';
defaults.nominal_plot_size_px = [660 528];
defaults.extra_title_margin = 0.5;
defaults.field_limits = [-12 12; -12 12];
defaults.output_limits = [-4 -1];
defaults.h_axis_title = '{\itu} [{\it\lambda}/{\itD}]';
defaults.h_axis_tick_spacing = 2;
defaults.v_axis_title = '{\itv} [{\it\lambda}/{\itD}]';
defaults.v_axis_tick_spacing = 2;
defaults.labels = {};
defaults.show_color_bars = true;
defaults.color_maps = {hot(256)};
defaults.font_size_pt = 14;
end
```

getStarView.m

```
% Produces a StarView object that captures the convolution of stars as viewed
% through an aperture producing the supplied point spread function.
%
% stars          An array of Star objects that are to appear in the star view
% psf            A Psf object describing the characteristic point spread
%                function of the aperture the stars are being viewed through
% diameter_in    The aperture diameter of the telescope (inches)
% wavelength_nm  The wavelength of light to use to draw the convolved power
%                spectra (nanometers)
%
% sv             The StarView object produced

function [sv] = getStarView(stars, psf, diameter_in, wavelength_nm)

sv = StarView;

% Cache (L/D -> arcsecond) factor for easy reference.
as_from_ld = asFromLd(wavelength_nm, diameter_in);

% Calculate pixel scale in pixels per arcseconds.
```

```
sv.pixels_per_as = psf.pixels_per_ld / as_from_ld;

% If no stars, no result.
if numel(stars) == 0
    return;
end

% Precompute the bounds of the domain we'll want to fill. Since the PSF for each
% star will be the same, the result comes down to how the star positions expand
% the region.
min_star_u =  inf();
max_star_u = -inf();
min_star_v =  inf();
max_star_v = -inf();
for i = 1:numel(stars)
    star_u = stars(i).pos_as(1);
    if star_u < min_star_u
        min_star_u = star_u;
    end
    if star_u > max_star_u
        max_star_u = star_u;
    end

    star_v = stars(i).pos_as(2);
    if star_v < min_star_v
        min_star_v = star_v;
    end
    if star_v > max_star_v
        max_star_v = star_v;
    end
end

% Calculate padding and use this to form a canvas that can hold everything.
upx_min_pad = -round(min_star_u * sv.pixels_per_as);
upx_max_pad =  round(max_star_u * sv.pixels_per_as);
vpx_min_pad = -round(min_star_v * sv.pixels_per_as);
vpx_max_pad =  round(max_star_v * sv.pixels_per_as);
upx_total = size(psf.data, 1) + upx_min_pad + upx_max_pad;
vpx_total = size(psf.data, 2) + vpx_min_pad + vpx_max_pad;
sv.data = zeros(upx_total, vpx_total);

% Assign arcsecond bounds accordingly.
sv.as_bounds(1,1) = min_star_u + psf.ld_bounds(1,1) * as_from_ld;
sv.as_bounds(1,2) = max_star_u + psf.ld_bounds(1,2) * as_from_ld;
sv.as_bounds(2,1) = min_star_v + psf.ld_bounds(2,1) * as_from_ld;
sv.as_bounds(2,2) = max_star_v + psf.ld_bounds(2,2) * as_from_ld;

% Compose the image in different slices--one slice per convolution member.
for i = 1:numel(stars)
    upx_shift =  round(stars(i).pos_as(1) * sv.pixels_per_as);
    vpx_shift =  round(stars(i).pos_as(2) * sv.pixels_per_as);

    slice = zeros(size(sv.data));

    % Calculate the shifted domain over which to place the new star image.
    upx_range = upx_min_pad + upx_shift + (1:size(psf.data,1));
    vpx_range = vpx_min_pad + vpx_shift + (1:size(psf.data,2));
    slice(upx_range, vpx_range) = psf.data / max(max(psf.data));

    % Amplify the star by its brightness as we add it to the convolution.
    sv.data = sv.data + 100^(-stars(i).app_vis_mag / 5) * slice;
```

```
end

end
```

getStarViewPlotDefaults.m

```
% Generates a default ImagescProps object configured for StarView plots. The
% object can be supplied to the svPlot function.
%
% defaults  The default ImagescProps object configured for StarView plots.
function [defaults] = getStarViewPlotDefaults
defaults = ImagescProps;
defaults.plot_title = 'Simulated monochromatic view';
defaults.nominal_plot_size_px = [660 528];
defaults.extra_title_margin = 0.2;
defaults.field_limits = [-5 5; -5 5];
defaults.output_limits = [10 2];
defaults.h_axis_title = '{\itu} [as]';
defaults.h_axis_tick_spacing = 1;
defaults.v_axis_title = '{\itv} [as]';
defaults.v_axis_tick_spacing = 1;
defaults.labels = {};
defaults.show_color_bars = true;
defaults.color_maps = {bone(256)};
defaults.font_size_pt = 14;
end
```

ImagescProps.m

```
% Properties for formatting plots created from the imagesc function.
% ImagescProps objects are used as an input to several functions that produce
% plots, including plotAperture, psfPlot, and svPlot.

classdef ImagescProps
    properties
        % Title of the plot
        plot_title

        % Nominal plot size (pixels) [width,height]. MATLAB may adapt the
        % numbers in unexpected ways.
        nominal_plot_size_px

        % Fudge factor for vertically aligning plot title
        extra_title_margin

        % Bounds of plot domain--i.e. what to crop the plot to [x1,x2;y1,y2]
        field_limits

        % Range of dependent axis values to display [low,high]. This also
        % determines how colors are mapped to these values.
        output_limits

        % Horizontal axis title and tick spacing
        h_axis_title
        h_axis_tick_spacing
        % Vertical axis title and tick spacing
        v_axis_title
        v_axis_tick_spacing

        % Legend entries describing plotted objects {label1,label2,...,labelN}
        labels

        % Whether to show color bars beside the plot
        show_color_bars

        % Color maps to use for plotted objects {map1,map2,...,mapN}
        color_maps

        % Font size of all text in figure (points)
        font_size_pt
    end
end
```

IoProps.m

```
% Holds properties related to the saving of figures to disc.

classdef IoProps
    properties
        % Whether and where to save a portable network graphics (PNG) file
        save_png = true;
        png_location = 'output.png';

        % Whether and where to save an encapsulated postscript (EPS) file
        save_eps = false;
        eps_location = 'output.eps';
    end
end
```

plotAperture.m

```
% Plots an aperture image with labeled coordinate axes.
%
% aperture        The aperture image to plot
% imagesc_props   An ImagescProps object governing how the plot will appear
% io_props        An IoProps object governing whether and how the output is saved
%
% figure_out      A handle to the figure created by this function

function [figure_out] = plotAperture(aperture, imagesc_props, io_props)
s = imagesc_props;
o = io_props;

% Convert color_maps to cell array to allow proper indexing later.
if (~iscell(s.color_maps))
    s.color_maps = {s.color_maps};
end

% Create and scale figure. Aperture image is assumed to represent exactly the
% entire aperture--no more, no less. Larger dimension establishes diameter.
figure_out = figure;
```

```
d_px = max(size(aperture));
imagesc([-0.5 0.5] .* size(aperture, 2) / d_px, ...
        [0.5 -0.5] .* size(aperture, 1) / d_px, aperture);
formatImagescPlot(figure_out, s);

% Apply color map. We can only plot one thing, so we use first map in the list.
colormap(s.color_maps{1});

% Save the image to disc if needed.
if o.save_eps
    print('-depsc', '-painters', o.eps_location);
end
if o.save_png
    print('-dpng', o.png_location);
end

end
```

Psf.m

```
% Represents a point spread function. Contains information about the
% intensity of the electromagnetic field at discrete angular positions (which
% correspond to known ratios of light wavelength to aperture diameter).

classdef Psf
    properties
        % Matrix with values proportional to the intensity (square of complex
        % amplitude) of the electromagnetic field at discrete angles. First
        % index is values of u spanning ld_bounds(1,:); second index is values
        % of v spanning ld_bounds(2,:).
        data

        % Number of pixels per ratio of light wavelength to aperture diameter
        % (pixels/(L/D))
        pixels_per_ld

        % The angular domain of the PSF data (L/D) [umin,umax;vmin,vmax]
        ld_bounds
    end
end
```

psfCut.m

```
% Creates a figure displaying the intensity of the electromagnetic field along
% the u-axis of one or more point spread functions.
%
% psfs        Psf objects representing the PSFs to cut
% cut_props   A CutProps object describing how to format the figure.
% io_props    An IoProps object determining whether and how the figure is saved
%
% figure_out  A handle to the generated figure.

function [figure_out] = psfCut(psfs, cut_props, io_props)
c = cut_props;
o = io_props;
```

```
% Condition some arguments into cell arrays so that they work in loops.
if (~iscell(c.color_maps))
    c.color_maps = {c.color_maps};
end
if (~iscell(c.labels))
    c.labels = {c.labels};
end
if (~iscell(c.line_colors))
    c.line_colors = {c.line_colors};
end

num_psfs = numel(psfs);
num_maps = numel(c.color_maps);

% Create cell arrays that will store values of u and intensity for each PSF cut.
u = cell(1, num_psfs);
w = cell(1, num_psfs);

% For each PSF, log-normalize the intensities and collect data along the u-axis.
for i=1:num_psfs
    image = log10(psfs(i).data ./ max(max(psfs(i).data)));
    upx_min = 1 + round(psfs(i).pixels_per_ld * ...
        (c.u_limits(1) - psfs(i).ld_bounds(1,1)));
    upx_max = 1 + round(psfs(i).pixels_per_ld * ...
        (c.u_limits(2) - psfs(i).ld_bounds(1,1)));
    v_px =    1 + round(psfs(i).pixels_per_ld * ...
        (0 - psfs(i).ld_bounds(1,1)));

    % Because we rounded to find u bound indices closest to requested limits, we
    % calculate what values of u *actually* correspond to those indices.
    u{i} = psfs(i).ld_bounds(1,1) + ((upx_min:upx_max) - 1) / psfs(i).pixels_per_ld;
    w{i} = image(upx_min:upx_max,v_px);
end

% Begin boring plot formatting stuff...
figure_out = figure;
set(figure_out, 'Position', [0 0 c.nominal_plot_size_px]);
hold on;

% Create an array of line handles, adding one extra slot if we need to show the
% contrast target also.
h = zeros(1, num_psfs + c.show_target);

% Show the contrast target if requested.
if (c.show_target)
    h(end) = plot([c.u_limits(1) c.u_limits(2)], [c.target c.target], ...
        'Color', c.target_line_color, 'LineStyle', '--', 'LineWidth', ...
        c.target_line_thickness_pt);
end

% Actually plot the cut data.
line_styles = {'-', '--', ':', '-.'};
for i=1:num_psfs
    h(i) = plot(u{i}, w{i}, 'Color', c.line_colors{i});
    set(h(i), 'LineWidth', c.cut_line_thickness_pt);
    set(h(i), 'LineStyle', line_styles{1 + num_psfs - i});
end

hold off;

% Create the legend.
```

```
if (c.show_target)
    legend(h, c.labels, 'contrast target');
else
    if numel(c.labels) > 0
        legend(h, c.labels);
    end
end

% Configure the axis displays.
xlabel(c.u_title);
ylabel(c.w_title);
xlim(c.u_limits);
ylim(c.w_limits);
set(gca,'FontSize', c.font_size_pt, 'fontWeight', 'bold');
set(gca, 'XTick', (c.u_limits(1)):c.u_spacing:c.u_limits(2));
set(gca, 'YTick', (c.w_limits(1)):c.w_spacing:c.w_limits(2));

% The logic to show the color bars is convoluted because we cheat our way around
% the typical MATLAB restriction of one color bar per plot.
if (c.show_color_bars)
    cb = colorbar('westoutside');
    colormap(cb, c.color_maps{end});
    caxis(c.c_limits);
    % Cache initial color bar's position so we can place subsequent bars.
    color_bar_pos = cb.Position;
    set(cb, 'TickLabels', []);
    set(cb, 'AxisLocation', 'in');
    set(cb, 'Limits', c.w_limits);
    set(cb, 'Ticks', (c.w_limits(1)):c.w_spacing:c.w_limits(2));

    for i=1:(num_maps-1)
        cb = colorbar;
        colormap(cb, c.color_maps{i});
        caxis(c.c_limits);
        % Place color bars one standard color bar's width apart so they're
        % adjacent.
        cb.Position = color_bar_pos - [(num_maps-i)*color_bar_pos(3) 0 0 0];
        set(cb, 'TickLabels', []);
        set(cb, 'AxisLocation', 'in');
        set(cb, 'Limits', c.w_limits);
        set(cb, 'Ticks', (c.w_limits(1)):c.w_spacing:c.w_limits(2));
    end
end

% Add labels and change font size.
my_title = title(c.plot_title);
title_pos = get(my_title, 'Position');
set(gca,'FontSize',c.font_size_pt,'fontWeight','bold');
set(findall(gcf,'type','text'),'FontSize',c.font_size_pt,'fontWeight','bold');
set(my_title, 'Position', title_pos + [0 c.extra_title_margin 0]);

% Save the plot to disc if requested.
if o.save_eps
    print('-depsc', '-painters', o.eps_location);
end
if o.save_png
    print('-dpng', o.png_location);
end

end
```

psfGetImage.m

```
% Generates a graphical representation of a point spread function encoded by a
% Psf object across a specified angle domain.
%
% psf                 The Psf object representing the PSF to visualize
% new_ld_bounds       The angular bounds of the image in wavelengths per aperture
%                     diameter [umin,umax;vmin,vmax]
% log_10_mag_limits   Log-10 magnitude limits corresponding to black and white
%                     with intermediate values in grayscale [min,max]
%
% image               The generated PSF image as a matrix of grayscale values

function [image] = psfGetImage(psf, new_ld_bounds, log_10_mag_limits)

% Log-normalize the power spectrum.
image = log10(psf.data ./ max(max(psf.data)));

% Crop the spectrum as close as possible to ld_bounds.
upx_min = 1 + floor(psf.pixels_per_ld * (new_ld_bounds(1,1) - psf.ld_bounds(1,1)));
upx_max = 1 +  ceil(psf.pixels_per_ld * (new_ld_bounds(1,2) - psf.ld_bounds(1,1)));
vpx_min = 1 + floor(psf.pixels_per_ld * (new_ld_bounds(2,1) - psf.ld_bounds(2,1)));
vpx_max = 1 +  ceil(psf.pixels_per_ld * (new_ld_bounds(2,2) - psf.ld_bounds(2,1)));
image = image(upx_min:upx_max,vpx_min:vpx_max);

% Map the log-10 magnitude limits to [0, 1]. No clamping is applied.
mag_delta = log_10_mag_limits(2) - log_10_mag_limits(1);
image = (image - log_10_mag_limits(1)) / mag_delta;

% Rotate the image to same conventions as original aperture image.
image = rot90(image);

end
```

psfPlot.m

```
% Creates a formatted plot of a point spread function encoded by a Psf object.

% psfs            The Psf objects to plot
% imagesc_props   An ImagescProps object describing how to format the plot
% io_props        An IoProps object determining whether and where to save output
%
% figure_out      Handle to the figure generated by this function

function [figure_out] = psfPlot(psfs, imagesc_props, io_props)
s = imagesc_props;
o = io_props;

% Condition color_maps and labels into cell arrays so that they work in loops.
if (~iscell(s.color_maps))
    s.color_maps = {s.color_maps};
end
if (~iscell(s.labels))
    s.labels = {s.labels};
end

num_psfs = numel(psfs);
psf_images = cell(num_psfs, 1);
```

```
num_maps = numel(s.color_maps);
num_labels = numel(s.labels);

% Find the canvas size required to display all PSFs, calling it 'max_size_px'.
max_size_px = [0 0];
for i=1:num_psfs
    psf_images{i} = psfGetImage(psfs(i), s.field_limits, s.output_limits);
    max_size_px = max(max_size_px, size(psf_images{i}));
end

% Construct an RGB canvas to this size and additively combine every PSF image.
composite = zeros([max_size_px 3]);
for i=1:num_psfs
    map = s.color_maps{i};
    num_colors = size(map, 1);
    % Resize small images to size of largest.
    psf_images{i} = imresize(psf_images{i}, max_size_px);
    % Convert psf_images to indices within color map.
    psf_images{i} = round(1 + (num_colors - 1) * max(0, min(1, psf_images{i})));
    for x=1:size(psf_images{i}, 1)
        for y=1:size(psf_images{i}, 2)
            % Add colors element-wise (concise 3D matrix ops are challenging).
            for c=1:3
                composite(x,y,c) = composite(x,y,c) + map(psf_images{i}(x,y),c);
            end
        end
    end
end

figure_out = figure;
ax = axes;

% Display and do initial formatting on plot.
imagesc(s.field_limits(1,:), fliplr(s.field_limits(2,:)), composite);
formatImagescPlot(figure_out, s);

% If we have PSFs to label, create a legend.
if (num_labels > 0)
    hold on;
    h = zeros(num_labels, 1);

    % Usually, legends are used on line plots only, so we create an empty "line
    % plot" in order to display a legend.
    for i=1:num_labels
        h(i) = plot(NaN, NaN, 'Marker', 's', 'MarkerSize', 8, ...
            'MarkerFaceColor', s.color_maps{i}(end, :), 'MarkerEdgeColor', ...
            'none', 'LineStyle', 'none');
    end

    l = legend(h, s.labels);
    l.Color = 'none';
    l.TextColor = [0.99 0.99 0.99];  % Pure white doesn't display.
    l.EdgeColor = [0.99 0.99 0.99];  % Pure white doesn't display.
    hold off;
end

% Establish baseline color bar position so we can position extras (if needed).
% MATLAB usually limits us to one color bar per plot, but we get around this by
% positioning the second and subsequent bars manually.
if (s.show_color_bars)
    cb = colorbar(ax);
    colormap(cb, s.color_maps{1});
    caxis(s.output_limits);
    color_bar_pos = cb.Position;

    for i=2:num_maps
         % Cancel last color bar's labels if more bars to show.
         set(cb, 'TickLabels', []);

         % Must create dummy hidden axes to place extra color bar.
         ax = axes;
         ax.Visible = 'off';
         ax.XTick = [];
         ax.YTick = [];

         % Create and format new color bar on dummy axes.
         cb = colorbar(ax);
         colormap(cb, s.color_maps{i});
         caxis(s.output_limits);

         % Match color bar's dimensions to baseline, translating it by its
         % width.
         cb.Position = color_bar_pos + [(i-1)*color_bar_pos(3) 0 0 0];
    end

    set(cb,'FontSize', s.font_size_pt, 'fontWeight', 'bold');
    ylabel(cb, 'log_1_0 contrast');
end

% Same the image if requested.
if o.save_eps
    print('-depsc', '-painters', o.eps_location);
end
if o.save_png
    print('-dpng', o.png_location);
end

end
```

runAperturePlot.m

```
% Utility script that produces a formatted aperture plot, isolating the most
% common parameters for easy customization.

% Focus on below variables =====================================================

input_name = 'gaussian 50 donut 160';  % Aperture image file name less extension
aperture_title = 'Gaussian boomerang';  % Plot title

% End important variables ======================================================

input_prefix = 'apertures/';
output_prefix = 'plots/';
io_props = IoProps;
io_props.save_png = true;
io_props.save_eps = false;
io_props.png_location = [output_prefix input_name ' aperture plot.png'];
aperture = imread([input_prefix input_name '.png']);
aperture_plot_props = getAperturePlotDefaults;
```

```
aperture_plot_props.plot_title = aperture_title;
close(plotAperture(aperture, aperture_plot_props, io_props));
```

runMultiPsf.m

```
% Utility script that produces a formatted overlay plot of the point spread
% functions of given apertures along with a plot comparing their contrast
% through a horizontal cut. Isolates the most common parameters for easy
% customization.

% Focus on below variables =====================================================

% 1 is base; 2 is addition
input1_name = 'c11';  % Aperture image file name less extension
input2_name = 'gaussian 50 donut 160';
aperture1_title = 'C11 aperture';  % Shows up in plot titles
aperture2_title = 'Gaussian boomerang';
aperture_scale = 1.0;  % Default: 1.0 (can use 0.25 for draft)
fft_scale = 8;  % Default: 8
ld_bound = 12;  % Max magnitude of u and v dimensions in PSF plots; default: 12
mag_lims_psf = [-4 -1];  % Default: [-4 -1]
mag_lims_cut = [-8 0];  % Default: [-8 0]
show_target = true;  % Default: true
target = -2.8;  % Default: -2.8
labels = {aperture1_title aperture2_title};

% End important variables ======================================================

input_prefix = 'apertures/';
output_prefix = 'plots/';
io_props = IoProps;
io_props.save_png = true;
io_props.save_eps = false;
io_props.png_location = [output_prefix input2_name ' vs ' input1_name ' psf plot.png'];
aperture1 = imread([input_prefix input1_name '.png']);
aperture2 = imread([input_prefix input2_name '.png']);
psf1 = getPsf(aperture1, aperture_scale, fft_scale);
psf2 = getPsf(aperture2, aperture_scale, fft_scale);
psf_plot_props = getPsfPlotDefaults;
psf_plot_props.plot_title = 'Point spread function comparison';
psf_plot_props.field_limits = ld_bound .* [-1 1; -1 1];
psf_plot_props.output_limits = mag_lims_psf;
psf_plot_props.color_maps = {[1 0 1] .* gray(256) [0 1 0] .* gray(256)};
psf_plot_props.labels = labels;
close(psfPlot([psf1 psf2], psf_plot_props, io_props));
cut_props = CutProps;
cut_props.plot_title = 'Horizontal PSF cut comparison';
cut_props.u_limits = [0 ld_bound];
cut_props.w_limits = mag_lims_cut;
cut_props.show_color_bars = true;
cut_props.color_maps = psf_plot_props.color_maps;
cut_props.c_limits = mag_lims_psf;
cut_props.show_target = show_target;
cut_props.target = target;
cut_props.labels = labels;
cut_props.font_size_pt = 14;
cut_props.line_colors = {cut_props.color_maps{1}(end,:)
cut_props.color_maps{2}(end,:)};
```

```
io_props.png_location = [output_prefix input2_name ' vs ' input1_name ' psf cut.png'];
close(psfCut([psf1 psf2], cut_props, io_props));
```

runSinglePsf.m

```
% Utility script that produces a formatted plot of the point spread function
% of a given aperture along with a plot of contrast through a horizontal cut.
% Isolates the most common parameters for easy customization.

% Focus on below variables ====================================================

input_name = 'gaussian 50 donut 160';  % Aperture image file name less extension
aperture_title = 'Gaussian boomerang';  % Shows up in plot titles
aperture_scale = 1.0;  % Default: 1.0 (can use 0.25 for draft)
fft_scale = 8;  % Default: 8
ld_bound = 12;  % Max magnitude of u and v dimensions in PSF plots; default: 12
mag_lims_psf = [-4 -1];  % Default: [-4 -1]
mag_lims_cut = [-8 0];  % Default: [-8 0]
show_target = true;  % Default: true
target = -2.8;  % Default: -2.8
labels = {aperture_title};

% End important variables =====================================================

input_prefix = 'apertures/';
output_prefix = 'plots/';
io_props = IoProps;
io_props.save_png = true;
io_props.save_eps = false;
io_props.png_location = [output_prefix input_name ' psf plot.png'];
aperture = imread([input_prefix input_name '.png']);
[psf, scaled_aperture_size_px, fft_size_px] = ...
    getPsf(aperture, aperture_scale, fft_scale);
psf_image = psfGetImage(psf, ld_bound .* [-1 1; -1 1], mag_lims_psf);
imwrite(psf_image, [output_prefix input_name ' psf.png']);
psf_plot_props = getPsfPlotDefaults;
psf_plot_props.plot_title = ['Ideal monochromatic, on-axis PSF of ' aperture_title];
psf_plot_props.field_limits = ld_bound .* [-1 1; -1 1];
psf_plot_props.output_limits = mag_lims_psf;
close(psfPlot(psf, psf_plot_props, io_props));
cut_props = CutProps;
cut_props.plot_title = ['Horizontal PSF cut of ' aperture_title];
cut_props.u_limits = [0 ld_bound];
cut_props.w_limits = mag_lims_cut;
cut_props.show_color_bars = true;
cut_props.color_maps = gray(256);
cut_props.c_limits = mag_lims_psf;
cut_props.show_target = show_target;
cut_props.target = target;
cut_props.labels = labels;
cut_props.font_size_pt = 14;
io_props.png_location = [output_prefix input_name ' psf cut.png'];
close(psfCut(psf, cut_props, io_props));
savePsfSpecs(size(aperture), aperture_scale, scaled_aperture_size_px, ...
    fft_scale, fft_size_px, [output_prefix input_name ' psf specs.txt']);
```

runStarView.m

```
% Utility script that produces a formatted plot of stars viewed through an
% aperture, isolating the most common parameters for easy customization.

% Focus on below variables =====================================================

input_name = 'gaussian 50 donut 160';  % Aperture image file name less extension
aperture_title = 'Gaussian boomerang';  % Shows up in plot titles
aperture_scale = 1.0;  % Default: 1.0 (can use 0.25 for draft)
fft_scale = 8;  % Default: 8
as_bound = 5;
app_vis_mag_lims = [10 2];
star_separation_as = 3.1;
star_app_mags = [0 3.0];
star_angle_deg = 120;
aperture_diam_in = 11;
wavelength_nm = 548;

% End important variables ======================================================

input_prefix = 'apertures/';
output_prefix = 'plots/';
io_props = IoProps;
io_props.save_png = true;
io_props.save_eps = false;
io_props.png_location = [output_prefix input_name ' sv plot.png'];
stars = asterismFromDouble(star_separation_as, star_app_mags, star_angle_deg);
psf = getPsf(imread([input_prefix input_name '.png']), aperture_scale, fft_scale);
sv = getStarView(stars, psf, aperture_diam_in, wavelength_nm);
sv_image = svGetImage(sv, as_bound * [-1 1; -1 1], app_vis_mag_lims);
imwrite(sv_image, [output_prefix input_name ' sv.png']);
sv_plot_props = getStarViewPlotDefaults;
sv_plot_props.plot_title = ['Ideal monochromatic view through ' aperture_title];
sv_plot_props.field_limits = as_bound .* [-1 1; -1 1];
sv_plot_props.output_limits = app_vis_mag_lims;
svPlot(sv, sv_plot_props, io_props);
```

savePsfSpecs.m

```
% Creates a text file with PSF generation details for future reference.
%
% aperture_size_px         Dimensions of original aperture image (pixels)
%                          [height,width]
% aperture_scale           Factor by which aperture image is scaled before
%                          padding it in advance of taking the fast Fourier
%                          transform
% scaled_aperture_size_px  Size of aperture image after being rescaled (pixels)
%                          [height,width]
% fft_scale                Dimensions the scaled aperture image will be padded
%                          to as a ratio of the scaled aperture image size
%                          prior to taking the fast Fourier transform
% fft_size_px              Size of fast Fourier transform input (pixels)
%                          [height,width]
% output_location          Path at which the text file will be saved

function savePsfSpecs(aperture_size_px, aperture_scale, ...
    scaled_aperture_size_px, fft_scale, fft_size_px, output_location)
```

```
fid = fopen(output_location, 'w');
fprintf(fid, ['original aperture size:  ' num2str(aperture_size_px(1)) ...
    ' px X ' num2str(aperture_size_px(2)) ' px\r\n']);
fprintf(fid, ['aperture scale:  ' num2str(aperture_scale) '\r\n']);
fprintf(fid, ['reduced aperture size:  ' num2str(scaled_aperture_size_px(1)) ...
    ' px X ' num2str(scaled_aperture_size_px(2)) ' px\r\n']);
fprintf(fid, ['FFT scale:  ' num2str(fft_scale) '\r\n']);
fprintf(fid, ['FFT size:  ' num2str(fft_size_px(1)) ' px X ' ...
    num2str(fft_size_px(2)) ' px\r\n']);
fprintf(fid, ['timestamp:  ' datestr(now)]);
fclose(fid);

end
```

Star.m

```
% Object representing a single star. Holds position and magnitude.

classdef Star
    properties
        % Angular position of the star relative to center of field of view
        % (arcseconds) [u,v]
        pos_as

        % Apparent visual magnitude of the star (NOT log-10). Higher numbers are
        % dimmer stars per astronomical convention.
        app_vis_mag
    end

    methods
        % Constructs a star with given position and apparent visual magnitude.
        %
        % pos_as       Angular position of the star relative to center of the
        %              field of view (arcseconds) [u,v]
        % app_vis_mag  Apparent visual magnitude of the star (NOT log-10)
        %
        % star         The constructed Star object
        function star = Star(pos_as, app_vis_mag)
            star.pos_as = pos_as;
            star.app_vis_mag = app_vis_mag;
        end
    end
end
```

StarView.m

```
% Represents the convolution of stars with point spread functions for a
% particular wavelength and a specific telescope diameter. Importantly, a
% StarView object differs from a Psf object in that its dimensions are not
% normalized to lambda/diameter: instead, they are in arcseconds.

classdef StarView
    properties
        % A 2-D matrix of values proportional to the power density at known
        % angular coordinates. First index is values of u spanning
        % as_bounds(1,:); second index is values of v spanning as_bounds(2,:).
        % Proportionality constant has no concise definition.
        data

        % Spatial resolution of the data in pixels per arcsecond (pixels/as)
        pixels_per_as

        % Angular bounds of data (arcseconds) [umin,umax;vmin,vmax]
        as_bounds

        % Diameter of aperture "viewing" the StarView (inches)
        diameter_in

        % Wavelength of light captured in this StarView (nanometers)
        wavelength_nm
    end
end
```

svGetImage.m

```
% Generates a graphical representation of stars viewed with diffraction effects
% as encoded by a StarView object. The image is plotted on a specified angular
% domain with specified magnitude limits.
%
% sv                  The StarView object to visualize
% new_as_bounds       The angular bounds of the region to visualize (arcseconds)
%                     [umin,umax;vmin,vmax]
% app_vis_mag_limits  The apparent visual magnitude limits--not log-10
%                     limits--that define black and white pixels. Because larger
%                     apparent visual magnitudes correspond to dimmer objects,
%                     the bounds are listed in descending magnitude order as
%                     [max,min].
%
% image               The generated StarView image as a matrix of grayscale
%                     values

function [image] = svGetImage(sv, new_as_bounds, app_vis_mag_limits)

% Calculate apparent visual magnitude in existing StarView data.
image = -2.5 * log10(sv.data);

% Crop the field as close as possible to as_bounds.
upx_min = 1 + floor(sv.pixels_per_as * (new_as_bounds(1,1) - sv.as_bounds(1,1)));
upx_max = 1 +  ceil(sv.pixels_per_as * (new_as_bounds(1,2) - sv.as_bounds(1,1)));
vpx_min = 1 + floor(sv.pixels_per_as * (new_as_bounds(2,1) - sv.as_bounds(2,1)));
vpx_max = 1 +  ceil(sv.pixels_per_as * (new_as_bounds(2,2) - sv.as_bounds(2,1)));
image = image(upx_min:upx_max, vpx_min:vpx_max);

% Map the apparent visual magnitude limits to [0, 1]. No clamping is applied.
mag_delta = app_vis_mag_limits(2) - app_vis_mag_limits(1);
image = (image - app_vis_mag_limits(1)) / mag_delta;

% Rotate the image to same conventions as original aperture image.
image = rot90(image);

end
```

svPlot.m

```
% Creates a formatted plot of a StarView object across a specified angular
% range.

% sv              The StarView object to plot
% imagesc_props   An ImagescProps object describing how to format the plot
% io_props        An IoProps object determining whether and where to save output
%
% figure_out      Handle to the figure generated by this function

function [figure_out] = svPlot(sv, imagesc_props, imagesc_io_props)
s = imagesc_props;
o = imagesc_io_props;

% Condition color_maps into cell array for consistency with psfPlot.
if (~iscell(s.color_maps))
    s.color_maps = {s.color_maps};
end

% Preformat the data within the StarView. We convert log-10 magnitudes to
% apparent visual magnitudes and rotate the image to get (x,y) from (u,v).
image = sv.data;
image = -2.5 * log10(image);
image = rot90(image);

% Create and reformat the StarView plot by using imagesc followed by
% formatImagescPlot with the arguments supplied to svPlot.
figure_out = figure;
imagesc(sv.as_bounds(1,:), fliplr(sv.as_bounds(2,:)), image);
formatImagescPlot(figure_out, s);

% Format the color axis appropriately.
caxis(fliplr(s.output_limits));
h = colorbar;
colormap(flipud(s.color_maps{1}));
drawnow;  % MATLAB bug: color bar colors don't update without this line.
% Reverse the color bar axis because large magnitudes (dim objects) should
% appear lower in the scale.
set(h, 'YDir', 'reverse');
ylabel(h, 'apparent visual magnitude');

% Save the output if requested.
if o.save_eps
    print('-depsc', '-painters', o.eps_location);
end
if o.save_png
    print('-dpng', o.png_location);
end

end
```

L. APERTURE IMAGE CREATION CODE

L.1 Prerequisites

To run the aperture image creation code requires MATLAB. We used version R2018a to verify the code in this section. MATLAB's Image Processing Toolbox, while not mandatory, contains functions that allow additional apertures to be created.

None of the masks proposed in this paper—bowtie, Gaussian donut or multi-Gaussian—requires the Image Processing Toolbox. The bowtie mask and its beamed variant are provided in the code repository; the Gaussian-family masks call upon elementary MATLAB functions included in the standard MATLAB installation. Table 32 lists shapes that use the Image Processing Toolbox and the required functions the toolbox provides.

Table 32. Elementary shapes in the aperture image creation code suite and their Image Processing Toolbox dependencies.

Shape	**Example**	**Image Processing Toolbox functions invoked**
Regular polygons other than those provided by formRectangle()	Figure 14	poly2mask() within formPolygon.m
Oriented apodizing screens	Figure 12	imrotate() within makeApertures.m
Diagonal spiders	Debes et al.'s (2003) Figure 3	imrotate() within makeApertures.m

All aperture creation code can be found on GitHub alongside the diffraction simulation code at https://github.com/e-foley/FraunhoferSim.

L.2 Architecture and usage

The structure of the aperture creation code is very simple: it relies on a collection of "form" functions, where each function is responsible for creating a different type of shape. For example, formCircle() creates a circle; formRectangle() creates a rectangle, and formGaussian() creates a shape in the form of a reflected Gaussian curve. All such functions accept a canvas size parameter and operate on normalized coordinates such that shapes can easily be scaled to different dimensions.

In some cases, the output from a single "form" function is enough to define a mask. Most mask images, however, are compound shapes whose components originate from multiple different functions. Each "form" function's output is simply a matrix that can be logically combined with other matrices on an element-wise basis. For example, the logical "or" operator || opens holes in an existing shape because it favors values of 1, which correspond to transparency. The logical "and" operator && adds obstructions instead because it favors values of 0, which correspond to opaqueness. The logical complement operator ~ finds its use when a shape needs to be interpreted as open in one context and closed in another.

The script formApertures.m generates every mask seen in this paper. Along the way, it demonstrates how to begin with a set of elementary shapes and combine them to form complex apertures. A section toward the end of this script executes conditionally based on whether the Image Processing Toolbox is installed. Thus, formApertures.m should always execute to completion.

L.3 Code

Files are listed alphabetically by file name. For code describing functions related to the simulation of diffraction patterns, consult Appendix K.3 on page 194 instead.

formApodization.m

```
% Generates a Gaussian apodization profile with transparency maximized at the
% center of the image, tapering toward opaque outward at a rate inversely
% related to a standard deviation value. Note that the Gaussian function will
% produce values greater than zero for any real input arguments, but the domain
% must be truncated to the dimensions of the canvas size within this function.
% (In other words, if this image is to be padded with opaque pixels, there will
% be discontinuities along the borders of the original canvas.)
%
% This function should not be confused with formGaussian, which utilizes the
% Gaussian function output to form a transparent region representing a normal
% distribution rather than to modify translucency in a continuous profile.
%
% canvas_size_px  Dimensions of the image to create (pixels) [height,width]
% rel_std_dev     The standard deviation of the Gaussian distribution as a ratio
%                 of the larger canvas dimension
%
% M               The Gaussian apodization profile image, with transparent areas
%                 1, opaque areas 0, and intermediate values corresponding to
%                 translucent regions [2D array]

function [M] = formApodization(canvas_size_px, rel_std_dev)
    M = zeros(canvas_size_px, 'double');
    center_px = (canvas_size_px + 1) / 2;
    max_dim_px = max(canvas_size_px);
    % Translucency decays as a Gaussian function of distance to image center.
    for x = 1:canvas_size_px(1)
        for y = 1:canvas_size_px(2)
            M(x,y) = exp(-(((x-center_px(1))^2 + (y-center_px(2))^2) / ...
                (2*(rel_std_dev*max_dim_px)^2)));
        end
    end
end
```

formCircle.m

```
% Creates a circular transparent region in an image with specified canvas size.
% No anti-aliasing is applied.
```

```
%
% canvas_size_px  Dimensions of the image to create (pixels) [height,width]
% rel_radius      Radius of the circle as a ratio of the larger canvas dimension
%
% M               A 2D matrix with 1s in a center circular region and 0s
%                 elsewhere

function [M] = formCircle(canvas_size_px, rel_radius)
    M = zeros(canvas_size_px, 'logical');
    center_px = (canvas_size_px + 1) / 2;
    max_dim_px = max(canvas_size_px);
    for x=1:canvas_size_px(1)
        for y=1:canvas_size_px(2)
            % Pixel is transparent (1) if its distance from the image center is
            % no greater than the circle's radius.
            M(x,y) = sum(([x,y] - center_px) .^ 2) <= (max_dim_px * rel_radius) ^ 2;
        end
    end
end
```

formGaussian.m

```
% Forms an image showing the region enclosed by a Gaussian profile and its
% reflection across the horizontal axis as transparent, and other areas opaque.
% No anti-aliasing is applied.
%
% canvas_size_px   Dimensions of the image to create (pixels) [height,width]
% rel_peak_height  Amplitude of the Gaussian function (a half that's reflected)
%                  as a ratio of the canvas height
% rel_std_dev      The standard deviation of the Gaussian distribution as a
%                  ratio of the peak height
%
% M                The resulting image, with transparent regions 1 and opaque
%                  regions 0 (2D array)

function [M] = formGaussian(canvas_size_px, rel_peak_height, rel_std_dev)
M = ones(canvas_size_px, 'logical');
peak_height_px = rel_peak_height * canvas_size_px(1);
std_dev_px = rel_std_dev * peak_height_px;
mean_px = (canvas_size_px(2) + 1) / 2;
vert_center_px = (canvas_size_px(1) + 1) / 2;

% Calculate Gaussian in pixel scale.
h = 1:canvas_size_px(2);
vert_scaling = peak_height_px * std_dev_px * sqrt(2*pi);
norm_px = vert_center_px + vert_scaling * ...
    1/(std_dev_px*sqrt(2*pi)) * exp(-(h-mean_px).^2/(2*std_dev_px^2));

% Only calculate values for top half, then reflect result.
x_limit_px = round(canvas_size_px(1) / 2);
for x=1:x_limit_px
    for y=h
        if x < (canvas_size_px(1) + 1 - norm_px(y))
            M(x,y) = 0;
        end
    end
end
```

```
% Duplicate result about horizontal axis.
M = M .* flipud(M);

end
```

formMultigaussian.m

```
% Creates an image representing multiple openings in the shape of a region
% enclosed by the profile of a normal distribution and its reflection about its
% horizontal axis. Image size, subaperture placement, and opening dimensions can
% be selected. No antialiasing is applied.
%
% canvas_size_px   Dimensions of the image to create (pixels) [height,width]
% rel_centers      Locations of subapertures as ratios of the larger canvas
%                  dimension [vert1,horiz1;vert2,horiz2;...;vertN,horizN]
% rel_peak_height  Amplitude of the Gaussian function (a half that's reflected)
%                  as a ratio of the canvas height
% rel_std_dev      The standard deviation of the Gaussian distribution as a
%                  ratio of the peak height
%
% M                The resulting image, with transparent regions 1 and opaque
%                  regions 0 (2D array)

function [M] = formMultigaussian(canvas_size_px, rel_centers, ...
    rel_peak_height, rel_std_dev)
M = zeros(canvas_size_px, 'logical');

% Create slices representing single Gaussians and combine them onto the canvas
% by shifting each one by a number of pixels appropriate for the rel_centers
% argument.
for i = 1:size(rel_centers, 1)
    single = formGaussian(canvas_size_px, rel_peak_height, rel_std_dev);
    shift_px = -round(rel_centers(i,:) .* canvas_size_px);
    for j=1:canvas_size_px
        for k=1:canvas_size_px
            if (j + shift_px(1) >= 1 && j + shift_px(1) <= canvas_size_px(1) && ...
                    k + shift_px(2) >= 1 && k + shift_px(2) <= canvas_size_px(2))
                % Combine images using an "or" function to keep the output range
                % within [0, 1].
                M(j,k) = M(j,k) || single(j + shift_px(1), k + shift_px(2));
            end
        end
    end
end

end
```

formPolygon.m

```
% Creates a mask with a regular-polygonal-shaped transparent region at its
% center. The size, number of sides, and orientation of the polygon can be
% configured along with the size of the image. No antialiasing is applied.
%
% canvas_size_px  Dimensions of the image to create (pixels) [height,width]
% max_rel_radius  The radius of the circle circumscribing the polygon as a ratio
%                 of the greater dimension of the canvas
% num_sides       The number of sides the polygon has
% rot_deg         The angle at which the first vertex is placed relative to the
%                 origin (degrees)
%
% M               The resulting image, with transparent regions 1 and opaque
%                 regions 0 (2D array)

function [M] = formPolygon(canvas_size_px, max_rel_radius, num_sides, rot_deg)
max_dim_px = max(canvas_size_px);

% Mark vertices along the circumference of the bounding circle. We first
% calculate the angles at which these vertices will appear, then do basic trig
% to find the (x,y) coordinates.
theta = deg2rad(rot_deg) + linspace(0, 2*pi, num_sides + 1);

% Minus sign below converts raster coordinates to Cartesian coordinates.
v = (1 + canvas_size_px(1)) / 2 - max_dim_px * max_rel_radius * sin(theta);
h = (1 + canvas_size_px(2)) / 2 + max_dim_px * max_rel_radius * cos(theta);

% MATLAB's poly2mask function does all the heavy lifting. The function's x and y
% parameters act in Cartesian space unlike most MATLAB functions.
M = poly2mask(h, v, canvas_size_px(1), canvas_size_px(2));

end
```

formRectangle.m

```
% Creates an image representing a mask with a rectangular area removed. The
% rectangle's edges are aligned to rows and columns of the matrix. No
% antialiasing is applied.
%
% canvas_size_px  Dimensions of the image to create (pixels) [height,width]
% rel_center      The coordinates of the center of the rectangle as a ratio of
%                 the larger canvas dimension [vert,horiz]
% rel_dims        The height and width of the rectangle as ratios of the larger
%                 canvas dimension [height,width]
%
% M               The resulting image, with transparent regions 1 and opaque
%                 regions 0 (2D array)

function [M] = formRectangle(canvas_size_px, rel_center, rel_dims)
    M = zeros(canvas_size_px, 'logical');
    max_dim_px = max(canvas_size_px);

    % Calculate center and bounds.
    center = ((canvas_size_px + 1) / 2) + max_dim_px .* rel_center;
    top =    max(center(1) - max_dim_px * rel_dims(1) / 2, 1);
    bottom = min(center(1) + max_dim_px * rel_dims(1) / 2, canvas_size_px(1));
    left =   max(center(2) - max_dim_px * rel_dims(2) / 2, 1);
    right =  min(center(2) + max_dim_px * rel_dims(2) / 2, canvas_size_px(2));

    % Populate regions contained by bounds.
    M(floor(top):ceil(bottom),floor(left):ceil(right)) = 1;
end
```

formScreen.m

```
% Creates an image representing a mask with alternating opaque and transparent
% ridges aligned with the vertical axis. The function operates on parameters
% provided in pixels rather than as ratios of the canvas size like other
% functions, so it is best for specialized, precise use cases.
%
% canvas_size_px  Dimensions of the image to create (pixels) [height,width]
% line_width_px   Width of each opaque line (pixels)
% spacing_px      Spacing between opaque lines, including their width (pixels)
%
% M               The resulting image, with transparent regions 1 and opaque
%                 regions 0 (2D array)

function [M] = formScreen(canvas_size_px, line_width_px, spacing_px)
    M = ones(canvas_size_px, 'logical');
    for i = 1:line_width_px
        M(:,i:spacing_px:canvas_size_px) = 0;
    end
end
```

formSineGrating.m

```
% Creates an image representing a gradated mask whose translucency varies
% sinusoidally along a specified axis at a specified spatial frequency and
% phase.
%
% canvas_size_px     Dimensions of the image to create (pixels) [height,width]
% frequency_1_px     The spatial frequency of the sinusoidal function (1/pixel)
% phase_deg          The phase of the sinusoid (degrees). The image's top-left
%                    pixel's translucency is effectively calculated using this
%                    value as its angle.
% grating_angle_deg  The orientation of the sine grating (degrees). When 0, the
%                    spatial wave "propagates" along the horizontal axis to form
%                    vertical bars. A positive value rotates the grating
%                    counterclockwise.
%
% M                  A 2D matrix representing the sine grating image, with
%                    transparent areas 1, opaque areas 0, and translucent
%                    regions somewhere between 0 and 1.

function [M] = formSineGrating(canvas_size_px, frequency_1_px, phase_deg, ...
    grating_angle_deg)
M = zeros(canvas_size_px, 'double');

% Find normal unit vector for purpose of calculating distances.
n = [-sind(grating_angle_deg) cosd(grating_angle_deg)];

% For every element in matrix, find the distance along the direction of the unit
% vector by projecting the indices onto it. Use that distance to calculate our
% progression through the sine wave.
for i=1:size(M, 1)
    for j=1:size(M, 2)
        dist_px = dot([i j] - 1, n);
        M(i,j) = 0.5 + 0.5 * sind(360 * frequency_1_px * dist_px + phase_deg);
    end
end
end
```

makeApertures.m

```
% Creates all images representing apertures and masked apertures and saves them
% in the "apertures" folder.

function makeApertures
output_prefix = 'apertures/';
canvas_dim_px = 2048;
canvas_size_px = canvas_dim_px * [1 1];
make_tifs = true;

% CIRCULAR APERTURE
circle = formCircle(canvas_size_px, 0.5);
imwrite(circle, [output_prefix 'circle.png']);

% C11 OBSTRUCTION
% Assuming entire image represents 11", cut 3.881" circle out.
c11_obstruction = ~formCircle(canvas_size_px, 3.881/11*0.5);
imwrite(c11_obstruction, [output_prefix 'c11 obstruction.png']);

% C11 APERTURE
c11 = circle & c11_obstruction;
imwrite(c11, [output_prefix 'c11.png']);

% GAUSSIAN VARIANTS (NO OBSTRUCTION)
for i=10:5:65
    gaussian = circle & formGaussian(canvas_size_px, 0.5, i / 100);
    imwrite(gaussian, [output_prefix 'gaussian ' num2str(i) '.png']);
end
clear i gaussian;

% GAUSSIAN OBSTRUCTIONS
% Gaussian secondaries are numerically sized to overlap C11 secondary.
gaussian_30_obstruction = ~formGaussian(canvas_size_px, 0.32, 0.30);
imwrite(gaussian_30_obstruction, [output_prefix 'gaussian 30 obstruction.png']);
gaussian_35_obstruction = ~formGaussian(canvas_size_px, 0.29, 0.35);
imwrite(gaussian_35_obstruction, [output_prefix 'gaussian 35 obstruction.png']);
gaussian_36_obstruction = ~formGaussian(canvas_size_px, 0.29, 0.36);
imwrite(gaussian_36_obstruction, [output_prefix 'gaussian 36 obstruction.png']);
gaussian_40_obstruction = ~formGaussian(canvas_size_px, 0.27, 0.40);
imwrite(gaussian_40_obstruction, [output_prefix 'gaussian 40 obstruction.png']);
gaussian_45_obstruction = ~formGaussian(canvas_size_px, 0.25, 0.45);
imwrite(gaussian_45_obstruction, [output_prefix 'gaussian 45 obstruction.png']);
gaussian_50_obstruction = ~formGaussian(canvas_size_px, 0.23, 0.50);
imwrite(gaussian_50_obstruction, [output_prefix 'gaussian 50 obstruction.png']);
gaussian_55_obstruction = ~formGaussian(canvas_size_px, 0.22, 0.55);
imwrite(gaussian_55_obstruction, [output_prefix 'gaussian 55 obstruction.png']);
gaussian_60_obstruction = ~formGaussian(canvas_size_px, 0.21, 0.60);
imwrite(gaussian_60_obstruction, [output_prefix 'gaussian 60 obstruction.png']);
gaussian_70_obstruction = ~formGaussian(canvas_size_px, 0.20, 0.70);
imwrite(gaussian_70_obstruction, [output_prefix 'gaussian 70 obstruction.png']);
gaussian_80_obstruction = ~formGaussian(canvas_size_px, 0.19, 0.80);
imwrite(gaussian_80_obstruction, [output_prefix 'gaussian 80 obstruction.png']);
gaussian_100_obstruction = ~formGaussian(canvas_size_px, 0.18, 1.00);
imwrite(gaussian_100_obstruction, [output_prefix 'gaussian 100 obstruction.png']);
gaussian_120_obstruction = ~formGaussian(canvas_size_px, 0.18, 1.20);
imwrite(gaussian_120_obstruction, [output_prefix 'gaussian 120 obstruction.png']);
gaussian_160_obstruction = ~formGaussian(canvas_size_px, 0.18, 1.60);
imwrite(gaussian_160_obstruction, [output_prefix 'gaussian 160 obstruction.png']);
```

```
% GAUSSIAN DONUT, STDDEV FACTOR 0.30 (NO SUPPORT)
gaussian_30_donut = ...
    circle & formGaussian(canvas_size_px, 0.5, 0.30) & gaussian_30_obstruction;
imwrite(gaussian_30_donut, [output_prefix 'gaussian 30 donut.png']);

% GAUSSIAN DONUT, STDDEV FACTOR 0.35 (NO SUPPORT)
gaussian_35_donut = ...
    circle & formGaussian(canvas_size_px, 0.5, 0.35) & gaussian_35_obstruction;
imwrite(gaussian_35_donut, [output_prefix 'gaussian 35 donut.png']);

% GAUSSIAN DONUT, STDDEV FACTOR 0.36 (NO SUPPORT)
gaussian_36_donut = ...
    circle & formGaussian(canvas_size_px, 0.5, 0.36) & gaussian_36_obstruction;
imwrite(gaussian_36_donut, [output_prefix 'gaussian 36 donut.png']);

% GAUSSIAN DONUT, STDDEV FACTOR 0.40 (NO SUPPORT)
gaussian_40_donut = ...
    circle & formGaussian(canvas_size_px, 0.5, 0.40) & gaussian_40_obstruction;
imwrite(gaussian_40_donut, [output_prefix 'gaussian 40 donut.png']);

% GAUSSIAN DONUT, STDDEV FACTOR 0.45 (NO SUPPORT)
gaussian_45_donut = ...
    circle & formGaussian(canvas_size_px, 0.5, 0.45) & gaussian_45_obstruction;
imwrite(gaussian_45_donut, [output_prefix 'gaussian 45 donut.png']);

% GAUSSIAN DONUT, STDDEV FACTOR 0.50 (NO SUPPORT)
gaussian_50_donut = ...
    circle & formGaussian(canvas_size_px, 0.5, 0.50) & gaussian_50_obstruction;
imwrite(gaussian_50_donut, [output_prefix 'gaussian 50 donut.png']);

% GAUSSIAN DONUT, STDDEV FACTOR 0.55 (NO SUPPORT)
gaussian_55_donut = ...
    circle & formGaussian(canvas_size_px, 0.5, 0.55) & gaussian_55_obstruction;
imwrite(gaussian_55_donut, [output_prefix 'gaussian 55 donut.png']);

% GAUSSIAN DONUT, STDDEV FACTORS 0.45, 1.20
gaussian_45_donut_120 = ...
    circle & formGaussian(canvas_size_px, 0.5, 0.45) & gaussian_120_obstruction;
imwrite(gaussian_45_donut_120, [output_prefix 'gaussian 45 donut 120.png']);

% GAUSSIAN DONUT, STDDEV FACTORS 0.50, 0.30
gaussian_50_donut_30 = ...
    circle & formGaussian(canvas_size_px, 0.5, 0.50) & gaussian_30_obstruction;
imwrite(gaussian_50_donut_30, [output_prefix 'gaussian 50 donut 30.png']);

% GAUSSIAN DONUT, STDDEV FACTORS 0.50, 0.50
gaussian_50_donut_50 = ...
    circle & formGaussian(canvas_size_px, 0.5, 0.50) & gaussian_50_obstruction;
imwrite(gaussian_50_donut_50, [output_prefix 'gaussian 50 donut 50.png']);

% GAUSSIAN DONUT, STDDEV FACTORS 0.50, 0.80
gaussian_50_donut_80 = ...
    circle & formGaussian(canvas_size_px, 0.5, 0.50) & gaussian_80_obstruction;
imwrite(gaussian_50_donut_80, [output_prefix 'gaussian 50 donut 80.png']);

% GAUSSIAN DONUT, STDDEV FACTORS 0.50, 1.20
gaussian_50_donut_120 = ...
    circle & formGaussian(canvas_size_px, 0.5, 0.50) & gaussian_120_obstruction;
imwrite(gaussian_50_donut_120, [output_prefix 'gaussian 50 donut 120.png']);

% GAUSSIAN_DONUT, STDDEV FACTORS 0.50, 1.60
```

```
gaussian_50_donut_160 = ...
    circle & formGaussian(canvas_size_px, 0.5, 0.50) & gaussian_160_obstruction;
imwrite(gaussian_50_donut_160, [output_prefix 'gaussian 50 donut 160.png']);

% GAUSSIAN DONUT, STDDEV FACTORS 0.55, 1.20
gaussian_55_donut_120 = ...
    circle & formGaussian(canvas_size_px, 0.5, 0.55) & gaussian_120_obstruction;
imwrite(gaussian_55_donut_120, [output_prefix 'gaussian 55 donut 120.png']);

% BEAM
beam_rel_width = (3/16) / 11;
beam = ~formRectangle(canvas_size_px, [0 0], [beam_rel_width 1]);
imwrite(beam, [output_prefix 'bar.png']);
clear beam_rel_width

% GAUSSIAN_DONUT WITH BEAM, STDDEV FACTOR 0.30
gaussian_30_donut_with_beam = gaussian_30_donut & beam;
imwrite(gaussian_30_donut_with_beam, ...
    [output_prefix 'gaussian 30 donut with beam.png']);

% GAUSSIAN_DONUT WITH BEAM, STDDEV FACTOR 0.35
gaussian_35_donut_with_beam = gaussian_35_donut & beam;
imwrite(gaussian_35_donut_with_beam, ...
    [output_prefix 'gaussian 35 donut with beam.png']);

% GAUSSIAN_DONUT WITH BEAM, STDDEV FACTOR 0.36
gaussian_36_donut_with_beam = gaussian_36_donut & beam;
imwrite(gaussian_36_donut_with_beam, ...
    [output_prefix 'gaussian 36 donut with beam.png']);

% GAUSSIAN_DONUT WITH BEAM, STDDEV FACTOR 0.40
gaussian_40_donut_with_beam = gaussian_40_donut & beam;
imwrite(gaussian_40_donut_with_beam, ...
    [output_prefix 'gaussian 40 donut with beam.png']);

% GAUSSIAN_DONUT WITH BEAM, STDDEV FACTOR 0.45
gaussian_45_donut_with_beam = gaussian_45_donut & beam;
imwrite(gaussian_45_donut_with_beam, ...
    [output_prefix 'gaussian 45 donut with beam.png']);

% GAUSSIAN_DONUT WITH BEAM, STDDEV FACTOR 0.50
gaussian_50_donut_with_beam = gaussian_50_donut & beam;
imwrite(gaussian_50_donut_with_beam, ...
    [output_prefix 'gaussian 50 donut with beam.png']);

% APODIZATION, STDDEV FACTOR 0.18
apodization_18 = formApodization(canvas_size_px, 0.18);
imwrite(apodization_18, [output_prefix 'apodization 18.png']);

% CIRCLE WITH APODIZATION, STDDEV FACTOR 0.18
circle_with_apodization_18 = circle .* apodization_18;
imwrite(circle_with_apodization_18, [output_prefix 'circle with apodization 18.png']);

% SCREEN, VERTICAL, 8 PIXELS SPACED BY 32 PIXELS
screen_vertical = formScreen(canvas_dim_px, 8, 32);
imwrite(screen_vertical, [output_prefix 'screen vertical 8 32.png']);

% SCREEN, SQUARE, 8 PIXELS SPACED BY 32 PIXELS
screen_square = screen_vertical & screen_vertical';
imwrite(screen_square, [output_prefix 'screen square 8 32.png']);
```

```
% 1/16" SPIDER (FOUR-LEGGED)
rel_spider_width = (1/16) / 11;
spider_1_16 = ~formRectangle(canvas_size_px, [0 0], [rel_spider_width 1]) & ...
         ~formRectangle(canvas_size_px, [0 0], [1 rel_spider_width]);
imwrite(spider_1_16, [output_prefix 'spider 1 16.png']);

% 1/8" SPIDER (FOUR-LEGGED)
rel_spider_width = (1/8) / 11;
spider_1_8 = ~formRectangle(canvas_size_px, [0 0], [rel_spider_width 1]) & ...
         ~formRectangle(canvas_size_px, [0 0], [1 rel_spider_width]);
imwrite(spider_1_8, [output_prefix 'spider 1 8.png']);

clear rel_spider_width

% C11 WITH SPIDER (FOUR-LEGGED)
c11_with_spider = c11 & spider_1_16;
imwrite(c11_with_spider, [output_prefix 'c11 with spider.png']);

% C11 WITH GAUSSIAN, STDDEV FACTOR 0.30
c11_with_gaussian_30 = c11 & formGaussian(canvas_size_px, 0.5, 0.30);
imwrite(c11_with_gaussian_30, [output_prefix 'c11 with gaussian 30.png']);

% GAUSSIAN MULTI, STDDEV FACTOR 0.30
rel_horiz = 0.225;
rel_vert = 0.225;
rel_height = 0.2175;
rel_matrix = [-rel_vert -rel_horiz;
               rel_vert -rel_horiz;
              -rel_vert  rel_horiz;
               rel_vert  rel_horiz];
gaussian_30_multi = formMultigaussian(canvas_size_px, rel_matrix, rel_height, 0.30);
gaussian_30_multi = gaussian_30_multi & c11;
imwrite(gaussian_30_multi, [output_prefix 'gaussian 30 multi.png']);
clear rel_horiz rel_vert rel_height rel_matrix

% GAUSSIAN_MULTI, STDDEV FACTOR 0.35
rel_horiz = 0.225;
rel_vert = 0.225;
rel_height = 0.2125;
rel_matrix = [-rel_vert -rel_horiz;
               rel_vert -rel_horiz;
              -rel_vert  rel_horiz;
               rel_vert  rel_horiz];
gaussian_35_multi = formMultigaussian(canvas_size_px, rel_matrix, rel_height, 0.35);
gaussian_35_multi = gaussian_35_multi & c11;
imwrite(gaussian_35_multi, [output_prefix 'gaussian 35 multi.png']);
clear rel_horiz rel_vert rel_height rel_matrix

% GAUSSIAN_MULTI, STDDEV FACTOR 0.50
rel_horiz = 0.225;
rel_vert = 0.225;
rel_height = 0.18;
rel_matrix = [-rel_vert -rel_horiz;
               rel_vert -rel_horiz;
              -rel_vert  rel_horiz;
               rel_vert  rel_horiz];
gaussian_50_multi = formMultigaussian(canvas_size_px, rel_matrix, rel_height, 0.50);
gaussian_50_multi = gaussian_50_multi & c11;
imwrite(gaussian_50_multi, [output_prefix 'gaussian 50 multi.png']);
clear rel_horiz rel_vert rel_height rel_matrix
```

```
% GAUSSIAN_MULTI, STDDEV FACTOR 0.65
rel_horiz = 0.225;
rel_vert = 0.225;
rel_height = 0.155;
rel_matrix = [-rel_vert -rel_horiz;
               rel_vert -rel_horiz;
              -rel_vert  rel_horiz;
               rel_vert  rel_horiz];
gaussian_65_multi = formMultigaussian(canvas_size_px, rel_matrix, rel_height, 0.65);
gaussian_65_multi = gaussian_65_multi & c11;
imwrite(gaussian_65_multi, [output_prefix 'gaussian 65 multi.png']);
clear rel_horiz rel_vert rel_height rel_matrix

% SINE GRATINGS
for wavenumber_px=[8 16 32 64 128]
    sine_grating = formSineGrating(canvas_size_px, 1/wavenumber_px, 0, 90);
    imwrite(sine_grating, ...
        [output_prefix 'sine grating ' num2str(wavenumber_px) '.png']);
end
clear wavenumber_px sine_grating;

% BEGIN IMAGE PROCESSING TOOLBOX QUARANTINE ====================================

% Check if Image Processing Toolbox is installed.
v = ver;
if any(strcmp('Image Processing Toolbox', {v.Name}))
    % TRIANGLE
    triangle = formPolygon(canvas_size_px, 0.5, 3, 90);
    imwrite(triangle, [output_prefix 'triangle.png']);

    % SQUARE
    square = formPolygon(canvas_size_px, 0.5, 4, 0);
    imwrite(square, [output_prefix 'square.png']);

    % HEXAGON
    hexagon = formPolygon(canvas_size_px, 0.5, 6, 0);
    imwrite(hexagon, [output_prefix 'hexagon.png']);

    % SQUARE OBSTRUCTION (FOR SUPERPOSITION DEMO IN PAPER)
    % Size obstruction to cover C11 secondary mirror.
    square_obstruction = 1 - formPolygon(canvas_size_px, 3.881/11 * sqrt(2)/2, 4, 0);
    imwrite(square_obstruction, [output_prefix 'square obstruction.png']);

    % CIRCULAR APERTURE WITH SQUARE OBSTRUCTION
    circle_with_square_obstruction = circle & square_obstruction;
    imwrite(circle_with_square_obstruction, ...
        [output_prefix 'circle with square obstruction.png']);

    % APODIZING SCREEN, 8 PIXELS SPACED BY 32 PIXELS
    % Dimensions of circular screen cutouts from Lovro
    % (http://www.graphitegalaxy.com/index.cgi?a=diyapodmask).
    apodizing_screen = circle & ...
        (screen_square | formCircle(canvas_size_px, 0.55/2));
    apodizing_screen = apodizing_screen & ...
        (imrotate(screen_square, 30, 'crop') | formCircle(canvas_size_px, 0.78/2));
    apodizing_screen = apodizing_screen & ...
        (imrotate(screen_square, 60, 'crop') | formCircle(canvas_size_px, 0.90/2));
    imwrite(apodizing_screen, [output_prefix 'apodizing screen 8 32.png']);

    % GAUSSIAN DONUT WITH ORIENTED SPIDER, STDDEV FACTOR 0.30
    gaussian_30_with_oriented_spider = ...
```

```
        gaussian_30_donut & imrotate(spider_1_8, 45, 'crop');
    imwrite(gaussian_30_with_oriented_spider, ...
        [output_prefix 'gaussian 30 donut with oriented spider.png']);

    % GAUSSIAN DONUT WITH ORIENTED SPIDER, STDDEV FACTOR 0.35
    gaussian_35_with_oriented_spider = ...
        gaussian_35_donut & imrotate(spider_1_8, 45, 'crop');
    imwrite(gaussian_35_with_oriented_spider, ...
        [output_prefix 'gaussian 35 donut with oriented spider.png']);

    % GAUSSIAN DONUT WITH ORIENTED SPIDER, STDDEV FACTOR 0.40
    gaussian_40_with_oriented_spider = ...
        gaussian_40_donut & imrotate(spider_1_8, 45, 'crop');
    imwrite(gaussian_40_with_oriented_spider, ...
        [output_prefix 'gaussian 40 donut with oriented spider.png']);

    % GAUSSIAN DONUT WITH ORIENTED SPIDER, STDDEV FACTOR 0.45
    gaussian_45_with_oriented_spider = ...
        gaussian_45_donut & imrotate(spider_1_8, 45, 'crop');
    imwrite(gaussian_45_with_oriented_spider, ...
        [output_prefix 'gaussian 45 donut with oriented spider.png']);

    % GAUSSIAN DONUT WITH ORIENTED SPIDER, STDDEV FACTOR 0.50
    gaussian_50_with_oriented_spider = ...
        gaussian_50_donut & imrotate(spider_1_8, 45, 'crop');
    imwrite(gaussian_50_with_oriented_spider, ...
        [output_prefix 'gaussian 50 donut with oriented spider.png']);
end

% END IMAGE PROCESSING TOOLBOX QUARANTINE ======================================

% Optionally copy every .png to .tif.
if make_tifs
    pattern = fullfile(output_prefix, '*.png');
    files = dir(pattern);
    for i=1:length(files)
        new_name = strrep(files(i).name, '.png', '.tif');
        imwrite(imread([output_prefix files(i).name]), [output_prefix new_name]);
    end
    clear pattern files i new_name
end
end
```

M. MASK ROTATOR CODE

M.1 Prerequisites

We use the Arduino IDE to compile and upload mask rotator code to an Arduino Uno. This software can be found at https://www.arduino.cc/en/Main/Software. We used version 1.8.5, but newer versions should also work. After downloading or cloning the mask rotator source from https://github.com/e-foley/MaskRotator, configure the Arduino IDE to use the MaskRotator directory as the sketchbook location. In version 1.8.5 of the IDE, this can be accomplished via **File > Preferences**.

To compile and upload code, connect the power supply and USB cable to the Arduino Uno, open mask_rotator.ino using the IDE, and press Ctrl+U. The device can now be communicated with by either using the serial monitor (Ctrl+Shift+M) or another serial terminal.

M.2 Note on TimerOne library

The mask rotator software leverages an open-source library called TimerOne, originally written by Jesse Tane, to perform timer-based interrupts in support of the stepper motor driver. This library is licensed under a Creative Commons Attribution 3.0 United States license, the full details of which can be found at http://creativecommons.org/licenses/by/3.0/us/. No changes have been made to the library in support of this thesis project.

In order to avoid any misunderstandings about authorship, we do not include the source of the TimerOne library in this paper.

M.3 Code

Files are listed alphabetically. When a unit has both a header and source, the header is listed first.

bipolar_stepper.h

```
#ifndef BIPOLAR_STEPPER_H_
#define BIPOLAR_STEPPER_H_

// Represents a bipolar stepper motor.
class BipolarStepper {
 public:
  // Constructs a BipolarStepper by denoting Arduino pins to be used for motor
  // functions. The object will be created in an uninitialized, disabled state.
  //
  // brka: The Arduino pin corresponding to the motor's BRKA line.
  // dira: The Arduino pin corresponding to the motor's DIRA line.
  // (etc.)
  BipolarStepper(int brka, int dira, int pwma, int brkb, int dirb, int pwmb);

  // Destroys a BipolarStepper object, attempting to set the motor into a
  // deenergized state first.
  ~BipolarStepper();

  // Initializes a BipolarStepper object. This must be called in order for
  // actuation commands to function properly.
  void initialize();

  // Checks whether the BipolarStepper object has been initialized.
  //
  // Returns: True if the BipolarStepper object has been initialized.
  bool isInitialized() const;

  // Enables the motor. This must be called in order for actuation commands to
  // succeed.
  void enable();

  // Disables the motor. After disable() is called, actuation commands will be
  // ignored until
  void disable();

  // Checks whether the motor is enabled.
  //
  // Returns: True if the BipolarStepper object is enabled.
  bool isEnabled() const;

  // Steps the motor forward once. Will fail if the motor is not both
  // initialized and enabled.
  void stepForward();

  // Steps the motor backward once. Will fail if the motor is not both
  // initialized and enabled.
  void stepBackward();
 private:
  // The number of unique states that are cycled through via the stepForward()
  // and stepBackward() functions.
  static const int NUM_STATES = 4;

  // Internal function that executes a particular motor state by energizing pins
  // in a pattern appropriate for the current state.
  void doState(int state);

  // Arduino pin assignments for motor functions.
  const int brka_;
  const int dira_;
  const int pwma_;
  const int brkb_;
  const int dirb_;
  const int pwmb_;

  // Which energization state is currently active. Normally between 0 and
  // (NUM_STATES - 1).
  int state_;

  // Other status variables.
  bool initialized_;
  bool enabled_;
};

#endif
```

bipolar_stepper.cpp

```
#include "bipolar_stepper.h"
#include <Arduino.h>

BipolarStepper::BipolarStepper(int brka, int dira, int pwma, int brkb, int dirb,
    int pwmb) : brka_(brka), dira_(dira), pwma_(pwma), brkb_(brkb), dirb_(dirb),
    pwmb_(pwmb), state_(0), initialized_(false), enabled_(false) {}

BipolarStepper::~BipolarStepper() {
  // Put our outputs in what should be a safe state before destroying the object
  // that controls them.
  digitalWrite(brka_, LOW);
  digitalWrite(dira_, LOW);
  digitalWrite(pwma_, LOW);
  digitalWrite(brkb_, LOW);
  digitalWrite(dirb_, LOW);
  digitalWrite(pwmb_, LOW);
}

void BipolarStepper::initialize() {
  pinMode(brka_, OUTPUT);
  pinMode(dira_, OUTPUT);
  pinMode(pwma_, OUTPUT);
  pinMode(brkb_, OUTPUT);
  pinMode(dirb_, OUTPUT);
  pinMode(pwmb_, OUTPUT);
  doState(state_);
  initialized_ = true;
}
```

```
bool BipolarStepper::isInitialized() const {
  return initialized_;
}

void BipolarStepper::enable() {
  enabled_ = true;
}

void BipolarStepper::disable() {
  enabled_ = false;
}

bool BipolarStepper::isEnabled() const {
  return enabled_;
}

void BipolarStepper::stepForward() {
  if (!initialized_ || !enabled_) {
    return;
  }

  state_ = (state_ + 1) % NUM_STATES;
  doState(state_);
}

void BipolarStepper::stepBackward() {
  if (!initialized_ || !enabled_) {
    return;
  }

  state_ = (state_ + NUM_STATES - 1) % NUM_STATES;
  doState(state_);
}

void BipolarStepper::doState(int state) {
  state %= 4;
  switch (state) {
    case 0:
      digitalWrite(brka_, LOW);
      digitalWrite(brkb_, HIGH);
      digitalWrite(dira_, HIGH);
      analogWrite(pwma_, 255);
      break;
    case 1:
      digitalWrite(brka_, HIGH);
      digitalWrite(brkb_, LOW);
      digitalWrite(dirb_, LOW);
      analogWrite(pwmb_, 255);
      break;
    case 2:
      digitalWrite(brka_, LOW);
      digitalWrite(brkb_, HIGH);
      digitalWrite(dira_, LOW);
      analogWrite(pwma_, 255);
      break;
    case 3:
      digitalWrite(brka_, HIGH);
      digitalWrite(brkb_, LOW);
      digitalWrite(dirb_, HIGH);
      analogWrite(pwmb_, 255);
      break;
    default:
      // Can't get here.
      break;
  }
}
```

hall_switch.h

```
#ifndef HALL_SWITCH_H_
#define HALL_SWITCH_H_

// Represents a binary Hall effect switch that detects the presence of a nearby
// magnetic field.
class HallSwitch {
 public:
  // Constructs a HallSwitch object, delegating Arduino pins for its functions.
  // The HallSwitch object is constructed in an uninitialized state.
  //
  // power_pin: The Arduino pin used to power the hall effect switch.
  // state_pin: The Arduino pin delegated to read the digital state of the Hall
  //            effect switch.
  HallSwitch(int power_pin, int state_pin);

  // Initializes the HallEffect object. This must be called before setting the
  // power state of the switch or reading the switch's state.
  void init();

  // Checks whether the Hall effect switch has been initialized.
  //
  // Returns: True if the switch has been initialized.
  bool isInitialized() const;

  // Powers on or off the Hall effect switch.
  //
  // power_state: True to energize the hall effect switch. (Unenergized switches
  //              cannot trigger.)
  void setPowerState(bool power_state);

  // Checks whether the Hall switch is currently triggered by a magnetic field.
  //
  // Returns: True if the switch is triggered by a magnetic field.
  bool isTriggered() const;

 private:
   // Arduino pins delegated for Hall effects switch functions.
   const int power_pin_;
   const int state_pin_;

   // Whether the swiltch has been initialized.
   bool is_initialized_;
};

#endif
```

hall_switch.cpp

```
#include "hall_switch.h"
#include <Arduino.h>

HallSwitch::HallSwitch(const int power_pin, const int state_pin) :
    power_pin_(power_pin), state_pin_(state_pin), is_initialized_(false) {}

void HallSwitch::init() {
  pinMode(power_pin_, OUTPUT);
  digitalWrite(power_pin_, LOW);
  pinMode(state_pin_, INPUT);
  is_initialized_ = true;
}

bool HallSwitch::isInitialized() const {
  return is_initialized_;
}

void HallSwitch::setPowerState(bool power_state) {
  if (!is_initialized_) {
    return;
  }

  digitalWrite(power_pin_, power_state);
}

bool HallSwitch::isTriggered() const {
  if (!is_initialized_) {
    return false;
  }

  return !digitalRead(state_pin_);
}
```

index_task.h

```
#ifndef INDEX_TASK_H_
#define INDEX_TASK_H_

#include "hall_switch.h"
#include "mask_controller.h"
#include <Arduino.h>  // For size_t

// Operates a cooperative task whose responsibility is to drive a MaskController
// and HallSwitch in conjunction to determine a new index position for the mask.
// Physically, this index position is determined a location of peak magnetic
// field. No other functions should attempt to manipulate the HallSwitch,
// MaskController, or the MaskController's dependencies while indexing is
// active.
//
// The method used to determine an index is to advance the mask forward,
// recording angular positions at which the Hall effect switch triggers from
// low to high and from high to low; then doing the same in reverse; then taking
// the average of all four positions. Finally, the mask homes to its new zero
// point to show the operator where the device believes this location to be.
class IndexTask {
 public:
```

```
 // List of possible states the IndexTask can be in.
 enum class State : int {
   START = 0,                // Starting state.
   INIT,                     // State following a request to initialize.
   WAITING_FOR_FORWARD_LOW,  // Forward, waiting to be in a low state.
   FORWARD_LOW,              // Forward, waiting for low-to-high transition.
   FORWARD_HIGH,             // Forward, waiting for high-to-low transition.
   REVERSE_LOW,              // Backward, waiting for low-to-high transition.
   REVERSE_HIGH,             // Backward, waiting for high-to-low transition.
   INDEXED,                  // Index acquired; waiting for next action.
   CANNOT_INDEX              // Index can't be found; waiting for next action.
 };

 // Results of indexing operations.
 enum class IndexEvent {
   NONE,             // Default value.
   INDEX_FOUND,      // Index has been located.
   INDEX_NOT_FOUND   // We failed to find the index.
 };

 // Amount of time we are willing to wait for a HallSwitch state transition
 // before declaring that the device is  unable to find an index [ms].
 static const int INDEX_TIMEOUT_MS = 10000u;

 // Construct a new IndexTask, designating  the MaskController and
 // HallSwitch the task will operate.
 //
 // mask_controller: The MaskController to operate.
 // hall_switch: The HallSwitch to read.
 IndexTask(MaskController* mask_controller, HallSwitch* hall_switch);

 // Initialize the IndexTask. This must be requested before calling index().
 void init();

 // Checks for state transitions and takes actions accordingly. Call this as
 // frequently as possible to improve indexing resolution.
 void step();

 // Seek an index position for the mask. An index position will be established
 // where the task estimates a local peak in magnetic field strength, which
 // will typically be triggered when the magnet is directly above the physical
 // Hall effect sensor. Note that this operation will change the index of the
 // MaskController, affecting all subsequent MaskController actions.
 void index();

 // Retrieves the current state of the IndexTask. See the State enumeration.
 //
 // Returns: The current State enumerator describing the state of the task.
 State getState() const;

 // Establishes a function to call when we have finished looking for an index.
 //
 // cb: The function to invoke when we have finished looking for an index.
 //  -> event: The outcome of the indexing operation.
 //  -> index_offset_deg: The angle the index position has been adjusted by as
 //                       a result of the indexing operation [deg]. Set to
 //                       nullptr to remove the callback.
 void setIndexEventCallback(void (*cb)(IndexEvent event, float index_offset_deg));

private:
 // Length of array in which we store positions to use in calculating an
```

```
  // index position.
  static const size_t NUM_KEY_POSITIONS = 4u;

  // Utility method to check when an index timeout has occurred.
  //
  // Returns: True if a timeout is active.
  bool timedOut() const;

  // Utility method announcing via callback that an index could not be located.
  void announceIndexNotFound() const;

  // The MaskController to manipulate.
  MaskController* const mask_controller_;

  // The HallSwitch to read.
  HallSwitch* const hall_switch_;

  // Flags for requested actions.
  bool init_requested_;
  bool index_requested_;

  // Current state of the IndexTask.
  State state_;

  // Time of last HallSwitch state change or request to index. Used as a
  // reference for index timeouts.
  unsigned long last_index_progress_stamp_ms_;

  // Container for angle datapoints used in the determination of the True
  // index position.
  float key_positions_deg_[NUM_KEY_POSITIONS];

  // Callback to invoke when we have finished looking for an index.
  void (*index_event_callback_)(IndexEvent event, float index_offset_deg);
};

#endif
```

index_task.cpp

```
#include "index_task.h"
#include "hall_switch.h"
#include "mask_controller.h"
#include <Arduino.h>

IndexTask::IndexTask(MaskController* const mask_controller,
    HallSwitch* const hall_switch) : mask_controller_(mask_controller),
    hall_switch_(hall_switch), init_requested_(false), index_requested_(false),
    state_(State::START), last_index_progress_stamp_ms_(0u),
    index_event_callback_(nullptr) {
  for (size_t i = 0u; i < NUM_KEY_POSITIONS; ++i) {
    key_positions_deg_[i] = 0.0f;
  }
}

void IndexTask::init() {
  init_requested_ = true;
}
```

```
void IndexTask::step() {
  switch (state_) {
    case State::START:
      // Just wait for an init command...
      if (init_requested_) {
        init_requested_ = false;
        mask_controller_->stop();
        hall_switch_->setPowerState(false);
        state_ = State::INIT;
      }
      break;
    case State::INIT:
      // State reserved for more functionality... In the meantime, just wait for
      // an index command.
      if (index_requested_) {
        index_requested_ = false;
        mask_controller_->forward();
        hall_switch_->setPowerState(true);
        last_index_progress_stamp_ms_ = millis();
        state_ = State::WAITING_FOR_FORWARD_LOW;
      }
      break;
    case State::WAITING_FOR_FORWARD_LOW:
      // Wait for a low signal. (This is important if an index is requested when
      // we are currently near the index position.)
      if (!hall_switch_->isTriggered()) {
        last_index_progress_stamp_ms_ = millis();
        state_ = State::FORWARD_LOW;
      } else if (timedOut()) {
        mask_controller_->stop();
        hall_switch_->setPowerState(false);
        announceIndexNotFound();
        state_ = State::CANNOT_INDEX;
      }
      break;
    case State::FORWARD_LOW:
      // Continue forward as we wait for a triggered sensor.
      if (hall_switch_->isTriggered()) {
        key_positions_deg_[0] = mask_controller_->getPositionDeg(false);
        last_index_progress_stamp_ms_ = millis();
        state_ = State::FORWARD_HIGH;
      } else if (timedOut()) {
        mask_controller_->stop();
        hall_switch_->setPowerState(false);
        announceIndexNotFound();
        state_ = State::CANNOT_INDEX;
      }
      break;
    case State::FORWARD_HIGH:
      // We currently have a triggered sensor... Continue until it's not
      // triggered anymore.
      if (!hall_switch_->isTriggered()) {
        key_positions_deg_[1] = mask_controller_->getPositionDeg(false);
        mask_controller_->reverse();
        last_index_progress_stamp_ms_ = millis();
        state_ = State::REVERSE_LOW;
      } else if (timedOut()) {
        mask_controller_->stop();
        hall_switch_->setPowerState(false);
        announceIndexNotFound();
        state_ = State::CANNOT_INDEX;
      }
      break;
    case State::REVERSE_LOW:
      // Retread our ground in reverse until sensor is high again...
      if (hall_switch_->isTriggered()) {
        key_positions_deg_[2] = mask_controller_->getPositionDeg(false);
        last_index_progress_stamp_ms_ = millis();
        state_ = State::REVERSE_HIGH;
      } else if (timedOut()) {
        mask_controller_->stop();
        hall_switch_->setPowerState(false);
        announceIndexNotFound();
        state_ = State::CANNOT_INDEX;
      }
      break;
    case State::REVERSE_HIGH:
      // Last step in reverse...
      if (!hall_switch_->isTriggered()) {
        key_positions_deg_[3] = mask_controller_->getPositionDeg(false);
        mask_controller_->stop();
        hall_switch_->setPowerState(false);

        // Calculate average transition position.
        float angle_sum_deg = 0.0f;
        for (size_t i = 0u; i < NUM_KEY_POSITIONS; ++i) {
          angle_sum_deg += key_positions_deg_[i];
        }
        const float offset_deg = angle_sum_deg / NUM_KEY_POSITIONS;

        // Apply new index position and communicate it via callback.
        mask_controller_->offsetZero(offset_deg);
        if (index_event_callback_ != nullptr) {
          index_event_callback_(IndexEvent::INDEX_FOUND, offset_deg);
        }

        // Rotate to new zero to show users where we think it is.
        mask_controller_->rotateTo(0.0f);
        last_index_progress_stamp_ms_ = millis();
        state_ = State::INDEXED;
      } else if (timedOut()) {
        mask_controller_->stop();
        hall_switch_->setPowerState(false);
        announceIndexNotFound();
        state_ = State::CANNOT_INDEX;
      }
      break;
    case State::INDEXED:
      // We did it! Now wait for the next command to index so we can restart the
      // process.
      if (index_requested_) {
        index_requested_ = false;
        mask_controller_->forward();
        hall_switch_->setPowerState(true);
        last_index_progress_stamp_ms_ = millis();
        state_ = State::WAITING_FOR_FORWARD_LOW;
      }
      break;
    case State::CANNOT_INDEX:
      // Not a lot we can do in an error state except wait for instructions.
      if (index_requested_) {
        index_requested_ = false;
        mask_controller_->forward();
        hall_switch_->setPowerState(true);
        last_index_progress_stamp_ms_ = millis();
        state_ = State::WAITING_FOR_FORWARD_LOW;
      }
      break;
    default:
      // No clue how we got here... Let's reset everything.
      mask_controller_->stop();
      hall_switch_->setPowerState(false);
      init_requested_ = false;
      index_requested_ = false;
      state_ = State::START;
      break;
  }
}

void IndexTask::index() {
  index_requested_ = true;
}

IndexTask::State IndexTask::getState() const {
  return state_;
}

void IndexTask::setIndexEventCallback(
    void (*const cb)(IndexEvent event, float index_offset_deg)) {
  index_event_callback_ = cb;
}

bool IndexTask::timedOut() const {
  return (int)(millis() - last_index_progress_stamp_ms_) > INDEX_TIMEOUT_MS;
}

void IndexTask::announceIndexNotFound() const {
  if (index_event_callback_ != nullptr) {
    index_event_callback_(IndexEvent::INDEX_NOT_FOUND, 0.0f);
  }
}
```

mask_controller.h

```
#ifndef MASK_CONTROLLER_H_
#define MASK_CONTROLLER_H_

#include "stepper_controller.h"

// Operates a StepperController to manipulate a mask interfacing with a stepper
// motor. Maintains knowledge of the gear ratio between motor and mask in order
// to drive the motor to the desired angles.
class MaskController {
  public:
    // Preferences for direction of motion.
    enum class Direction : int {
      NONE = 0,  // No motion: default value.
      FORWARD,   // Forward direction.
      REVERSE,   // Reverse direction.
      AUTO       // Direction that will reach the target the fastest.
    };

    // Constructs a MaskController that operates a specified StepperController
    // using a given gear ratio between motor and mask.
    //
    // stepper_controller: The StepperController to drive.
    // gear_ratio: Rotations of motor per one rotation of mask.
    MaskController(volatile StepperController* stepper_controller,
        float gear_ratio);

    // Drives the mask forward continuously.
    void forward();

    // Drives the mask backward continuously.
    void reverse();

    // Halts mask motion.
    void stop();

    // Rotates the mask to an absolute angle.
    //
    // target_deg: Absolute angle to rotate the mask to [deg].
    // direction: Preferred direction of motion.
    // wrap_result: Whether the angle returned from the function is wrapped to
    //              the range [0, 360) degrees.
    // Returns: The actual absolute angle rotated to [deg]. May not match the
    //          specified angle exactly due to motor resolution limits.
    float rotateTo(float target_deg, Direction direction = Direction::AUTO,
        bool wrap_result = true);

    // Rotates the mask by a relative angle.
    //
    // angle_deg: Relative angle to rotate the mask by [deg].
    // direction: Preferred direction of motion.
    // wrap_result: Whether the angle returned from the function is wrapped to
    //              the range [0, 360) degrees.
    // Returns: The actual absolute angle rotated to [deg]. May not match the
    //          specified angle exactly due to motor resolution limits.
    float rotateBy(float angle_deg, bool wrap_result = true);

    // Retrieves the current absolute position of the mask.
    //
    // wrap_result: Whether the angle returned from the function is wrapped to
    //              the range [0, 360) degrees.
    // Returns: The current absolute position of the mask [deg].
    float getPositionDeg(bool wrap_result = true) const;

    // Retrieves the current target position of the mask.
    //
    // wrap_result: Whether the angle returned from the function is wrapped to
    //              the range [0, 360) degrees.
    // Returns: The current target position of the mask [deg].
    float getTargetDeg(bool wrap_result = true) const;

    // Establishes the current mask position to be an absolute angle of zero.
    void setZero();

    // Offsets the existing zero reference by an angle.
    //
    // relative_angle_deg: The angle to offset the zero reference by [deg].
```

```
    void offsetZero(float relative_angle_deg);

    // Converts a mask angle to a motor angle.
    //
    // mask_angle_deg: An absolute mask angle [deg].
    // Returns: The motor angle corresponding to the mask angle [deg].
    float maskToMotorAngleDeg(float mask_angle_deg) const;

    // Converts a motor angle to a mask angle.
    //
    // motor_angle_deg: An absolute motor angle [deg].
    // Returns: The mask angle corresponding to the motor angle [deg].
    float motorToMaskAngleDeg(float motor_angle_deg) const;

  private:
    // Wraps an unbounded angle to the range [0, 360) degrees.
    //
    // nominal_deg: The unbounded angle [deg].
    // Returns: An equivalent angle on the range [0, 360) degrees.
    static float wrapAngleDeg(float nominal_deg);

    // The StepperController this MaskController manipulates.
    volatile StepperController* const stepper_controller_;

    // Rotations of motor per one rotation of mask.
    const float gear_ratio_;

    // Current absolute target angle [deg].
    float target_deg_;
};

#endif
```

mask_controller.cpp

```
#include "mask_controller.h"
#include "stepper_controller.h"
#include <Math.h>

MaskController::MaskController(
    volatile StepperController* const stepper_controller,
    const float gear_ratio) : stepper_controller_(stepper_controller),
    gear_ratio_(gear_ratio), target_deg_(0.0f) {}

void MaskController::forward() {
  if (stepper_controller_ == nullptr) {
    return;
  } else if (gear_ratio_ > 0.0f) {
    stepper_controller_->forward();
  } else {
    stepper_controller_->reverse();
  }
}

void MaskController::reverse() {
  if (stepper_controller_ == nullptr) {
    return;
  } else if (gear_ratio_ > 0.0f) {
    stepper_controller_->reverse();
  } else {
    stepper_controller_->forward();
  }
}

void MaskController::stop() {
  if (stepper_controller_ == nullptr) {
    return;
  } else {
    stepper_controller_->stop();
  }
}

float MaskController::rotateTo(const float target_deg,
    const Direction direction, const bool wrap_result) {
  if (stepper_controller_ == nullptr) {
    return NAN;
  }

  // Stop and record because the motor angle would theoretically change as we
  // progress through the function if we didn't do this.
  stepper_controller_->stop();
  const float current_deg = getPositionDeg(false);
  const float forward_delta_deg = wrapAngleDeg(target_deg - current_deg);
  const float reverse_delta_deg = wrapAngleDeg(current_deg - target_deg);

  float delta_to_use_deg = 0.0f;
  switch (direction) {
    default:
    case Direction::NONE:
      break;
    case Direction::FORWARD:
      delta_to_use_deg = forward_delta_deg;
      break;
    case Direction::REVERSE:
      delta_to_use_deg = -reverse_delta_deg;
      break;
    case Direction::AUTO:
      delta_to_use_deg = forward_delta_deg < reverse_delta_deg ?
          forward_delta_deg : -reverse_delta_deg;
      break;
  }

  return rotateBy(delta_to_use_deg, wrap_result);
}

float MaskController::rotateBy(const float angle_deg, const bool wrap_result) {
  target_deg_ = getPositionDeg(false) + angle_deg;
  // We use rotateTo() below rather than rotateBy() so that we don't accumulate
  // roundoff error  between target_deg_ and the converted motor angle target in
  // repeated calls to this function.
  const float nominal_deg = motorToMaskAngleDeg(
      stepper_controller_->rotateTo(maskToMotorAngleDeg(target_deg_)));
  return wrap_result ? wrapAngleDeg(nominal_deg) : nominal_deg;
}

float MaskController::getPositionDeg(const bool wrap_result) const {
  const float nominal_deg =
      motorToMaskAngleDeg(stepper_controller_->getPositionDeg());
  return wrap_result ? wrapAngleDeg(nominal_deg) : nominal_deg;
```

```
}

float MaskController::getTargetDeg(const bool wrap_result) const {
  return wrap_result ? wrapAngleDeg(target_deg_) : target_deg_;
}

void MaskController::setZero() {
  if (stepper_controller_ == nullptr) {
    return;
  }
  stepper_controller_->stop();
  stepper_controller_->setZero();
}

void MaskController::offsetZero(const float relative_angle_deg) {
  if (stepper_controller_ == nullptr) {
    return;
  }
  stepper_controller_->stop();
  stepper_controller_->offsetZero(maskToMotorAngleDeg(relative_angle_deg));
}

float MaskController::wrapAngleDeg(const float nominal) {
  return nominal - 360.0f * floor(nominal / 360.0f);
}

float MaskController::maskToMotorAngleDeg(const float mask_angle_deg) const {
  return mask_angle_deg * gear_ratio_;
}

float MaskController::motorToMaskAngleDeg(const float motor_angle_deg) const {
  return motor_angle_deg / gear_ratio_;
}
```

mask_rotator.ino

```
#include <Arduino.h>
#include "bipolar_stepper.h"
#include "hall_switch.h"
#include "mask_controller.h"
#include "index_task.h"
#include "stepper_controller.h"
#include "timer_one.h"

// Serial config
const int SERIAL_BAUD_RATE = 19200;
const int SERIAL_TIMEOUT_MS = 10;  // [ms]
enum Command : char {
  FORWARD_COMMAND = 'f',
  BACKWARD_COMMAND = 'b',
  STOP_COMMAND = 's',
  GET_POSITION_COMMAND = 'p',
  GET_TARGET_COMMAND = 't',
  SET_ZERO_COMMAND = 'z',
  ENTER_RELATIVE_MODE_COMMAND = 'r',
  ENTER_ABSOLUTE_MODE_COMMAND = 'a',
  LOCATE_INDEX_COMMAND = 'i',
  FOUND_INDEX_RESPONSE = 'I',
  COULD_NOT_FIND_INDEX_RESPONSE = '~',
  PING_COMMAND = '?',
  PING_RESPONSE = '!',
  GO_TO_COMMAND = 'g',
  UNRECOGNIZED_COMMAND = 'x'
};

// Motor/mask config
const int BRKA_PIN = 9;
const int DIRA_PIN = 12;
const int PWMA_PIN = 3;
const int BRKB_PIN = 8;
const int DIRB_PIN = 13;
const int PWMB_PIN = 11;
const float GEAR_RATIO = 72.0/17.0;
const int16_t MOTOR_STEPS = 200u;  // Motor steps per revolution
uint32_t STEP_PERIOD_US = 8000u;  // [us]
const MaskController::Direction PREFERRED_DIRECTION =
    MaskController::Direction::AUTO;

// Hall switch config
const int HALL_SWITCH_POWER_PIN = 4;
const int HALL_SWITCH_STATE_PIN = 5;

// Objects, state variables, etc.
BipolarStepper stepper(BRKA_PIN, DIRA_PIN, PWMA_PIN, BRKB_PIN, DIRB_PIN, PWMB_PIN);
HallSwitch hall_switch(HALL_SWITCH_POWER_PIN, HALL_SWITCH_STATE_PIN);
StepperController motor_controller(&stepper, MOTOR_STEPS);
MaskController mask_controller(&motor_controller, GEAR_RATIO);
IndexTask index_task(&mask_controller, &hall_switch);
TimerOne timer;
enum class Mode {
  NONE,
  ABSOLUTE,
  RELATIVE
} mode = Mode::ABSOLUTE;

// Called once at the start of the progrom; initializes all hardware and tasks.
void setup() {
  Serial.begin(SERIAL_BAUD_RATE);
  Serial.setTimeout(SERIAL_TIMEOUT_MS);
  stepper.initialize();
  stepper.enable();
  hall_switch.init();
  index_task.init();
  index_task.setIndexEventCallback(&actOnIndexEvent);
  timer.initialize();
  timer.attachInterrupt(update, STEP_PERIOD_US);
}

// Called repeatedly: updates tasks and looks for new actions to take based on
// command inputs.
void loop() {
  index_task.step();

  // Process input.
  if (Serial.available()) {
    const char command = Serial.peek();
    switch (command) {
      case FORWARD_COMMAND:
        Serial.read();
        mask_controller.forward();
        Serial.write(FORWARD_COMMAND);
        Serial.println();
        break;
      case BACKWARD_COMMAND:
        Serial.read();
        mask_controller.reverse();
        Serial.write(BACKWARD_COMMAND);
        Serial.println();
        break;
      case STOP_COMMAND:
        Serial.read();
        mask_controller.stop();
        Serial.write(STOP_COMMAND);
        Serial.println();
        break;
      case GET_POSITION_COMMAND:
        Serial.read();
        Serial.write(GET_POSITION_COMMAND);
        Serial.println(degreesToSerial(mask_controller.getPositionDeg(true)));
        break;
      case GET_TARGET_COMMAND:
        Serial.read();
        Serial.write(GET_TARGET_COMMAND);
        Serial.println(degreesToSerial(mask_controller.getTargetDeg(true)));
        break;
      case SET_ZERO_COMMAND:
        Serial.read();
        mask_controller.setZero();
        Serial.write(SET_ZERO_COMMAND);
        Serial.println();
        break;
      case ENTER_RELATIVE_MODE_COMMAND:
        Serial.read();
        mode = Mode::RELATIVE;
        Serial.write(ENTER_RELATIVE_MODE_COMMAND);
        Serial.println();
        break;
      case ENTER_ABSOLUTE_MODE_COMMAND:
        Serial.read();
        mode = Mode::ABSOLUTE;
        Serial.write(ENTER_ABSOLUTE_MODE_COMMAND);
        Serial.println();
        break;
      case LOCATE_INDEX_COMMAND:
        Serial.read();
        index_task.index();
        Serial.write(LOCATE_INDEX_COMMAND);
        Serial.println();
        break;
      case PING_COMMAND:
        Serial.read();
        Serial.write(PING_RESPONSE);
        Serial.println();
        break;
      case GO_TO_COMMAND: {
        Serial.read();  // Get the command character out of the buffer.
        float serial_deg = serialToDegrees(Serial.parseInt());
        float actual_deg = 0.0f;
        if (mode == Mode::ABSOLUTE) {
          actual_deg = mask_controller.rotateTo(serial_deg, PREFERRED_DIRECTION);
        } else if (mode == Mode::RELATIVE) {
          actual_deg = mask_controller.rotateBy(serial_deg);
        }
        Serial.write(GO_TO_COMMAND);
        Serial.println(degreesToSerial(actual_deg));
        break;
      }
      default:
        Serial.read();  // Discard character if we don't recognize it.
        Serial.write(UNRECOGNIZED_COMMAND);
        Serial.println();
        break;
    }
  }
}

// Converts an angle from serial convention to degrees.
float serialToDegrees(const int32_t serial) {
  return serial / 100.0f;
}

// Convets an angle from degrees to serial convention. There is no overflow
// protection.
int32_t degreesToSerial(const float degrees) {
  return static_cast<int32_t>(round(degrees * 100.0f));
}

void actOnIndexEvent(const IndexTask::IndexEvent event,
    const float index_offset_deg) {
  (void)(index_offset_deg);  // Denote index offset parameter as unused.
  if (event == IndexTask::IndexEvent::INDEX_FOUND) {
    Serial.write(FOUND_INDEX_RESPONSE);
    Serial.println();
  } else if (event == IndexTask::IndexEvent::INDEX_NOT_FOUND) {
    Serial.write(COULD_NOT_FIND_INDEX_RESPONSE);
    Serial.println();
  }
}

// Function run via timer interrupt to actuate motor.
void update() {
  motor_controller.update();
}
```

stepper_controller.h

```
#ifndef STEPPER_CONTROLLER_H_
#define STEPPER_CONTROLLER_H_

#include "bipolar_stepper.h"
#include <Arduino.h>  // For int16_t, int32_t

// Drives a motor represented by BipolarStepper object.
class StepperController {
  public:
    // Current motor action.
    enum class Behavior : int {
      STOPPED = 0,    // Motor is stopped. Default value.
      FORWARD,        // Motor is moving forward continuously.
      REVERSE,        // Motor is moving backward continuously.
      TARGETING,      // Motor is currently approaching its target position.
      REACHED_TARGET  // Motor has successfully reached its target position.
    };

    // Constructs a StepperController, delegating a BipolarStepper to manipulate
    // and a number of steps per rotation. The update() function should be
    // invoked within a timer interrupt at approximately 125 Hz.
    //
    // stepper: The BipolarStepper to manipulate.
    // steps_per_rotation: Number of steps that form one full motor rotation.
    StepperController(BipolarStepper* stepper, int16_t steps_per_rotation);

    // Drives the motor forward continuously.
    void forward() volatile;

    // Drives the motor backward continuously.
    void reverse() volatile;

    // Halts motor motion.
    void stop() volatile;

    // Rotates the motor to an absolute angle.
    //
    // target_deg: Absolute angle to rotate the motor to [deg].
    // Returns: The actual absolute angle rotated to [deg]. May not match the
    //          specified angle exactly due to the finite number of steps per
    //          rotation.
    float rotateTo(float target_deg) volatile;

    // Rotates the motor by a relative angle.
    //
    // angle_deg: Relative angle to rotate the motor by [deg].
    // Returns: The actual absolute angle rotated to [deg]. May not match the
    //          specified angle exactly due to the finite number of steps per
    //          rotation.
    float rotateBy(float angle_deg) volatile;

    // Retrieves the current absolute position of the motor.
    //
    // Returns: The current absolute position of the motor [deg].
    float getPositionDeg() const volatile;

    // Retrieves the current target position of the motor.
    //
    // Returns: The current target position of the motor [deg].
    float getTarget() const volatile;

    // Establishes the current motor position to be an absolute angle of zero.
    void setZero() volatile;

    // Offsets the existing zero reference by an angle.
    //
    // relative_angle_deg: The angle to offset the zero reference by [deg].
    void offsetZero(float relative_angle_deg) volatile;

    // Updates the state of the motor. For best results, this should be called
    // within a timer interrupt triggering at approximately 125 Hz.
    void update() volatile;
```

```
    // Converts an absolute motor position to an absolute number of motor steps.
    //
    // degrees: The absolute angle to convert [deg].
    // Returns: The integral number of steps forming an angle closest to the
    //          given angle.
    int32_t degreesToSteps(float degrees) const volatile;

    // Converts a number of motor steps to an absolute angular position.
    //
    // steps: The number of steps.
    // Returns: The angle formed by traveling the given number of steps [deg].
    float stepsToDegrees(int32_t steps) const volatile;

  private:
    // The BipolarStepper driver this StepperController manipulates.
    BipolarStepper* const stepper_;

    // The number of steps of the motor constituting one full revolution.
    const int16_t steps_per_rotation_;

    // Current position of the motor in steps relative to zero.
    volatile int32_t position_steps_;

    // Current target absolute angle of the motor [deg].
    float target_deg_;

    // Current target absolute position of the motor in steps.
    int32_t target_steps_;

    // Currently active behavior.
    volatile Behavior behavior_;
};

#endif
```

stepper_controller.cpp

```
#include "stepper_controller.h"
#include "bipolar_stepper.h"
#include <Arduino.h>
#include <Math.h>

StepperController::StepperController(BipolarStepper* const stepper,
    const int steps_per_rotation) : stepper_(stepper),
    steps_per_rotation_(steps_per_rotation), position_steps_(0),
    target_deg_(0.0f), target_steps_(0), behavior_(Behavior::STOPPED) {}

void StepperController::forward() volatile {
  behavior_ = Behavior::FORWARD;
}

void StepperController::reverse() volatile {
  behavior_ = Behavior::REVERSE;
}

void StepperController::stop() volatile {
  behavior_ = Behavior::STOPPED;
}
```

```
float StepperController::rotateTo(const float target_deg) volatile {
  // Very brief pause to avoid potential momentary direction change.
  behavior_ = Behavior::STOPPED;
  target_deg_ = target_deg;
  target_steps_ = degreesToSteps(target_deg_);
  behavior_ = Behavior::TARGETING;
  return stepsToDegrees(target_steps_);
}

float StepperController::rotateBy(const float angle_deg) volatile {
  // Very brief pause to avoid position changes.
  behavior_ = Behavior::STOPPED;
  target_deg_ = stepsToDegrees(position_steps_) + angle_deg;
  target_steps_ = degreesToSteps(target_deg_);
  behavior_ = Behavior::TARGETING;
  return target_deg_;
}

float StepperController::getPositionDeg() const volatile {
  // TODO: Disable interrupts here.
  return stepsToDegrees(position_steps_);
}

float StepperController::getTarget() const volatile {
  return target_deg_;
}

void StepperController::setZero() volatile {
  position_steps_ = 0;
}

void StepperController::offsetZero(const float relative_angle_deg) volatile {
  position_steps_ -= degreesToSteps(relative_angle_deg);
}

// Note: Instead of a switch tree, we could set a function pointer (to a private
// helper function) whenever we alter behavior_. Snazzy but probably overkill.
void StepperController::update() volatile {
  if (stepper_ == nullptr) {
    return;
  }

  switch (behavior_) {
    default:
    case Behavior::STOPPED:
    case Behavior::REACHED_TARGET:
      break;
    case Behavior::FORWARD:
      stepper_->stepForward();
      position_steps_++;
      break;
    case Behavior::REVERSE:
      stepper_->stepBackward();
      position_steps_--;
      break;
    case Behavior::TARGETING:
      if (position_steps_ < target_steps_) {
        stepper_->stepForward();
        position_steps_++;
      } else if (position_steps_ > target_steps_) {
        stepper_->stepBackward();
        position_steps_--;
      } else /*position_steps_ == target_steps_*/ {
        behavior_ = Behavior::REACHED_TARGET;
      }
      break;
  }
}

int32_t StepperController::degreesToSteps(const float degrees) const volatile {
  return static_cast<int32_t>(round(degrees / 360.0f * steps_per_rotation_));
}

float StepperController::stepsToDegrees(const int32_t steps) const volatile {
  return 360.0f * steps / steps_per_rotation_;
}
```

timer_one.h, timer_one.cpp

TimerOne is an open-source library written by a third party. Our program uses TimerOne, but we did not modify its source code. See Appendix M.2.

N. BILLS OF MATERIALS

The bill of materials is divided into six parts to show the cost of each major subassembly. Unit prices are calculated at quantity where applicable. Costs from machining and assembly labor are excluded. Prices are accurate as of October 12, 2019.

N.1 Axle cap assembly

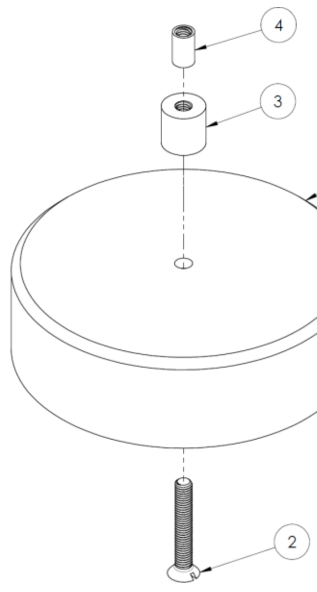


Table 33. Bill of materials for axle cap assembly.

#	Description	Vendor/ ID	Unit	Qty	Extended
1	4”-diameter acetal cylinder (per foot)	McMaster–Carr #8497K533	$69.33	0.1	$6.93
2	Stainless steel flathead screw, 10-32 × 1 1/4	McMaster–Carr #91781A835	$0.18	1	$0.18

#	Description	Vendor/ ID	Unit	Qty	Extended
3	Stainless steel standoff, female, 1/2 OD, 10-32 × 1/2	McMaster–Carr #91125A381	\$3.12	1	\$3.12
4	Stainless steel standoff, female, 1/4 OD, 10-32 × 13/32	McMaster–Carr #91125A591	\$1.91	1	\$1.91
TOT.					**\$12.14**

N.2 Mask assembly

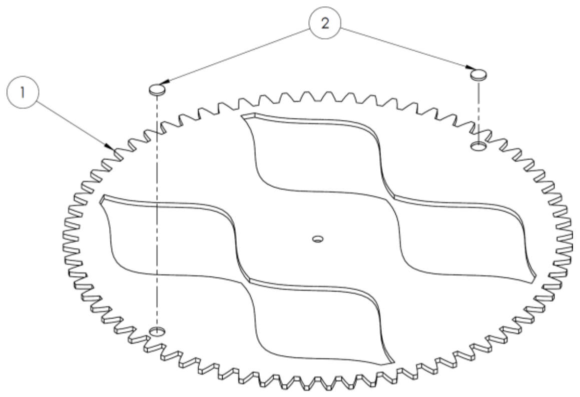


Table 34. Bill of materials for mask assembly.

#	Description	Vendor/ ID	Unit	Qty	Extended
1	Birch, 1/8 thick (per 30 × 24 sheet[49])	Woodcraft #131400	\$9.50	0.25	\$2.38

[49] Each 30-by-24 sheet provides four masks if the cuts are placed carefully.

#	Description	Vendor/ ID	Unit	Qty	Extended
2	Neodymium magnet, 3/8 × 1/16	totalElement	$0.26	2	$0.52
TOT.					**$2.90**

N.3 Motor bracket assembly

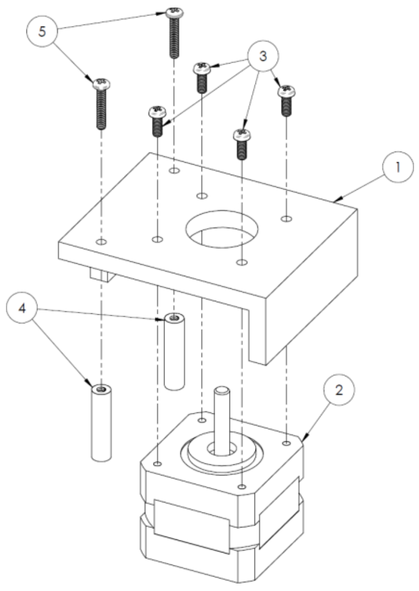
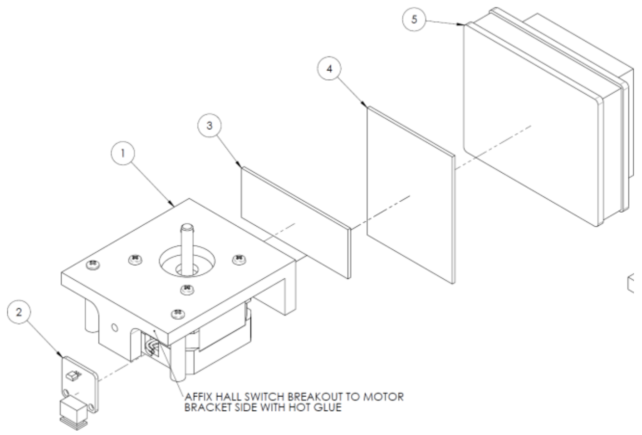


Table 35. Bill of materials for motor bracket assembly.

#	Description	Vendor/ ID	Unit	Qty	Extended
1	6063 aluminum U-channel, 1 H × 3 W × 1/4 thick (per 4 feet)	McMaster–Carr #9001K104	$43.98	0.052	$2.29

#	Description	Vendor/ ID	Unit	Qty	Extended
2	Bipolar stepper motor, NEMA 17, 12 V, 0.33 A, 1.8° step, 0.23 N-m holding torque	SparkFun #ROB-09238	\$15.95	1	\$15.95
3	Stainless steel cheesehead screw, M3×0.5×8	McMaster–Carr #94017A204	\$0.22	4	\$0.88
4	Aluminum standoff, female, 1/4 OD, 4-40 × 1	McMaster–Carr #93330A439	\$0.60	2	\$1.20
5	Stainless steel panhead screw, 4-40 × 5/8[50]	McMaster–Carr #91772A112	\$0.04	2	\$0.08
TOT.					**\$20.40**

N.4 Motor bracket electronics assembly


5
4
3
1
2
AFFIX HALL SWITCH BREAKOUT TO MOTOR BRACKET SIDE WITH HOT GLUE

[50] Part is also used for a different purpose in the pinion assembly.

Table 36. Bill of materials for motor bracket electronics assembly.

#	Description	Vendor/ ID	Unit	Qty	Extended
1	Motor bracket assembly	N/A	$20.40	1	$20.40
2	Hall switch breakout	Amazon (SunFounder) #0701715366763	$6.99	1	$6.99
3, 4	2” hook and loop strip (per 5 yd)	Amazon (Strenco)	$12.92	0.014	$0.18
5a	Case for Arduino Uno	Amazon (DAOKI)	$3.98	1	$3.98
5b	Arduino Uno Rev3	Arduino.cc	$22.00	1	$22.00
5c	Arduino Motor Shield Rev3	Arduino.cc	$22.00	1	$22.00
TOT.					**$75.55**

N.5 Pinion assembly

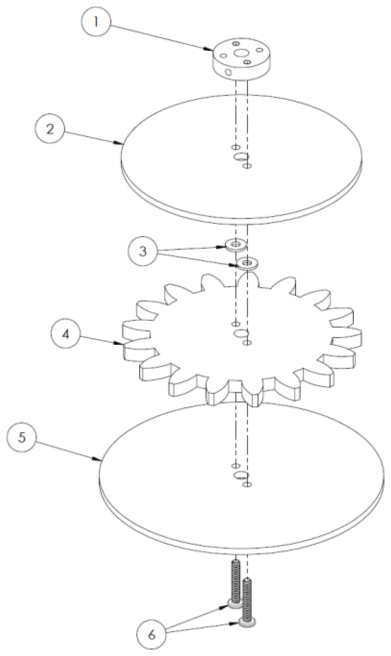


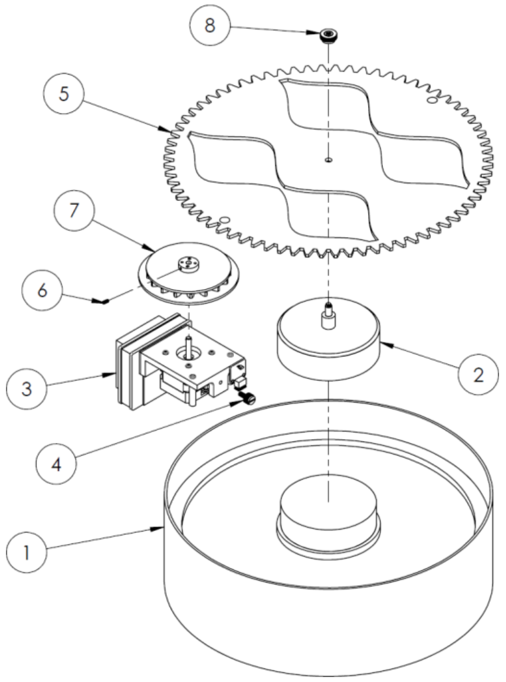
Table 37. Bill of materials for pinion assembly.

#	Description	Vendor/ ID	Unit	Qty	Extended
1	Aluminum mounting hub, 3/4" OD, 5-mm ID; 4 holes with 4-40 thread at 1/2" BD; .2" thick; 4-40 set screw (per 2 hubs)	SparkFun	\$3.75	1	\$3.75
2, 5	Clear acrylic, 3/32 thick (per 12 × 12 sheet[51])	McMaster–Carr #8560K181	\$5.24	0.25	\$1.32

[51] Each 12-inch-by-12-inch sheet can easily provide four pairs of lips.

#	Description	Vendor/ ID	Unit	Qty	Extended
3	Nickel-plated brass washer, #4, .281 OD	McMaster–Carr #92917A110	\$0.03	2	\$0.06
4	Black acrylic, 3/16 thick (per 12 × 12 sheet[52])	estreetplastics #B011871212	\$5.99	0.083	\$0.50
6	Stainless steel panhead screw, 4-40 × 5/8[53]	McMaster–Carr #91772A112	\$0.04	2	\$0.08
TOT.					**\$5.71**

N.6 Master assembly



[52] Each 12-inch-by-12-inch sheet can provide at least twelve 3.167-inch-diameter pinions if cut carefully (Specht, 2009).
[53] Part is also used for a different purpose in the motor bracket assembly.

Table 38. Bill of materials for master assembly.

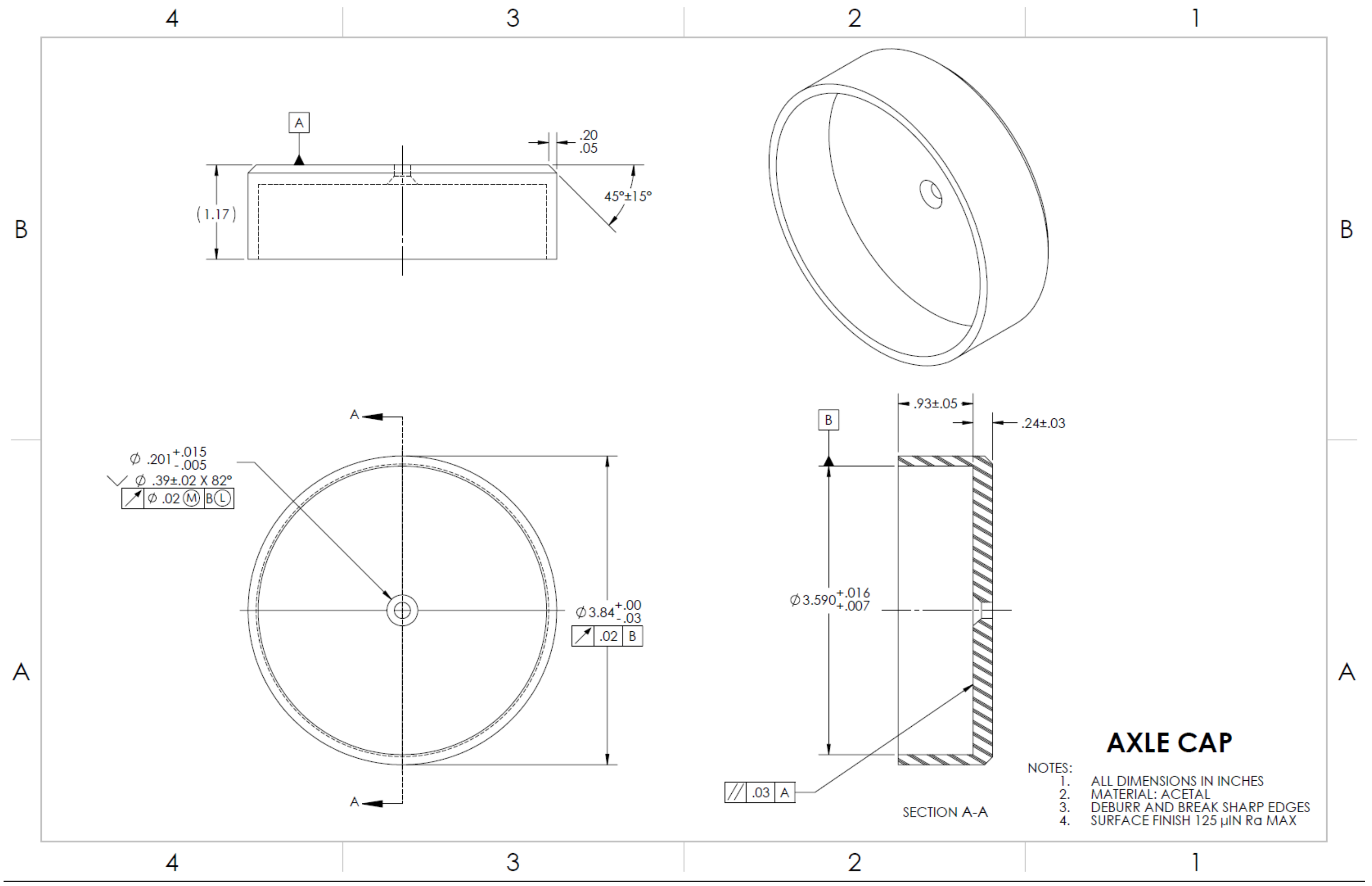
#	Description	Vendor/ ID	Unit	Qty	Extended
1	11” optical tube assembly	Celestron C11	N/A	1	N/A
2	Axle cap assembly	N/A	$12.14	1	$12.14
3	Motor bracket electronics assembly	N/A	$75.55	1	$75.55
4	Nylon thumb screw, 8/32 × 3/8	McMaster–Carr #94320A393	$0.09	1	$0.09
5	Mask assembly	N/A	$2.90	1	$2.90
6	Stainless steel hex set screw, 4-40 × 1/4	McMaster–Carr #92311A106	$0.04	0[54]	$0.00
7	Pinion assembly	N/A	$5.71	1	$5.71
8	Plastic thumb nut, 10-32 thread	McMaster–Carr #93886A130	$0.85	1	$0.85
–	9-V, 650-mA power supply	Amazon (SunFounder)	$7.99	1	$7.99
–	16’ USB A–B cable	Amazon (Basics)	$5.99	1	$5.99
TOT.					**$111.22**

[54] Mounting hub purchased for motor bracket assembly includes one such screw.

O. TECHNICAL DRAWINGS

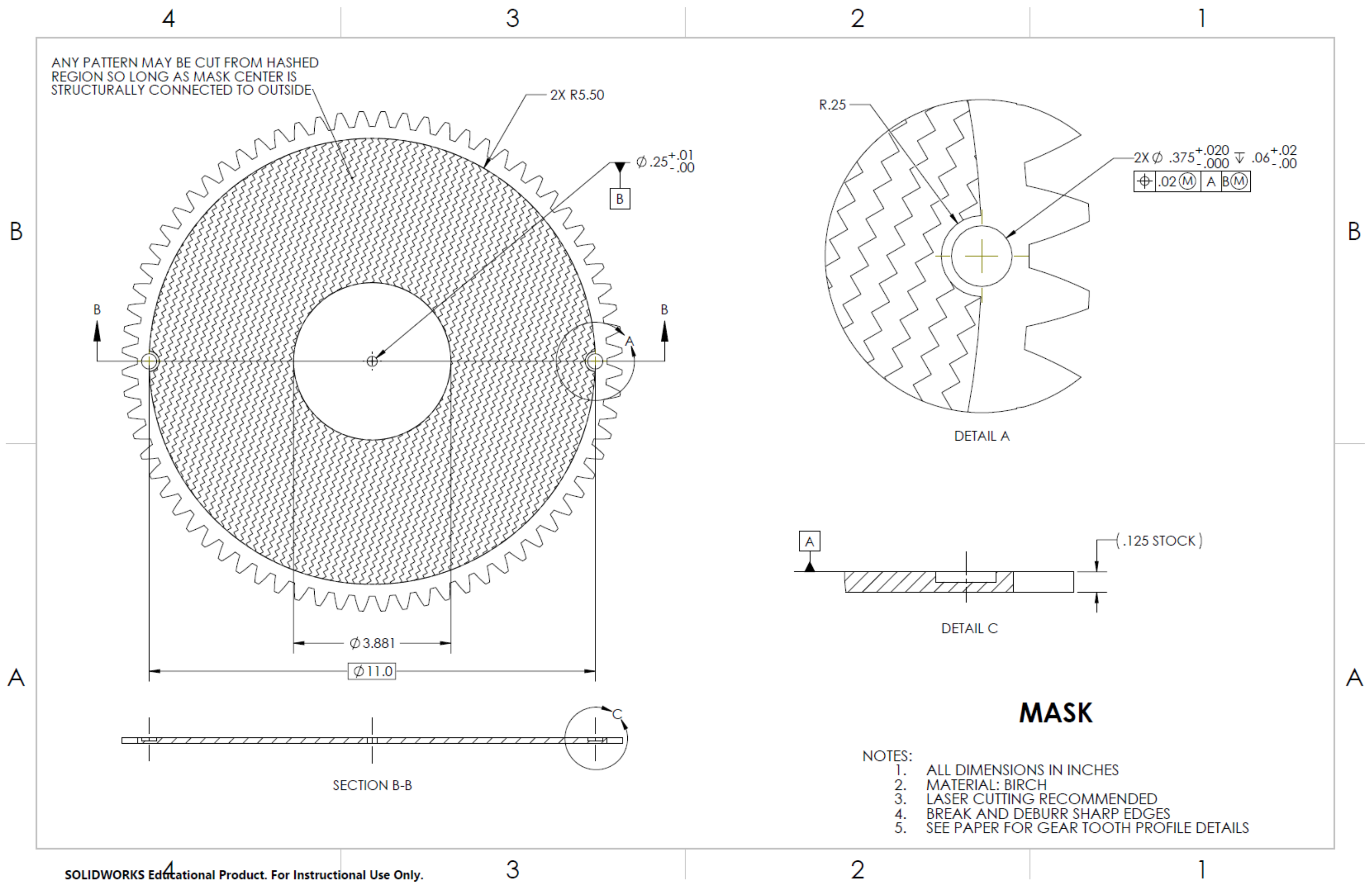
The following pages contain engineering drawings that describe the shapes of the custom parts along with their interfaces with off-the-shelf components. Appendix N contains a complete list of these components.

Due to formatting restrictions imposed on this paper, these drawings have been resized such that their scales do not align with common fractions.



AXLE CAP

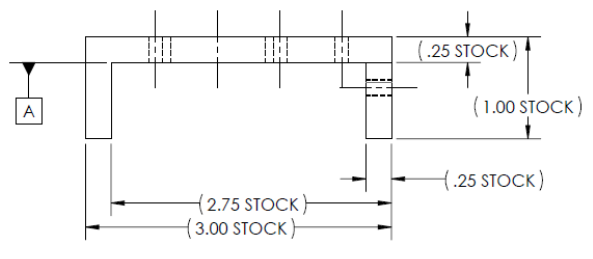
NOTES:
1. ALL DIMENSIONS IN INCHES
2. MATERIAL: ACETAL
3. DEBURR AND BREAK SHARP EDGES
4. SURFACE FINISH 125 µIN Ra MAX


4
3
2
1
B
A
ANY PATTERN MAY BE CUT FROM HASHED REGION SO LONG AS MASK CENTER IS STRUCTURALLY CONNECTED TO OUTSIDE
2X R5.50
Ø .25 +.01 -.00
B
B
B
A
Ø3.881
Ø11.0
C
SECTION B-B
R.25
2X Ø .375 +.020 -.000 ↧ .06 +.02 -.00
⌖ .02 Ⓜ A BⓂ
DETAIL A
A
(.125 STOCK)
DETAIL C

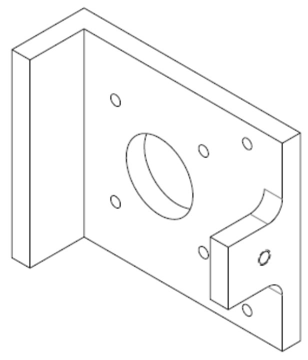
MASK

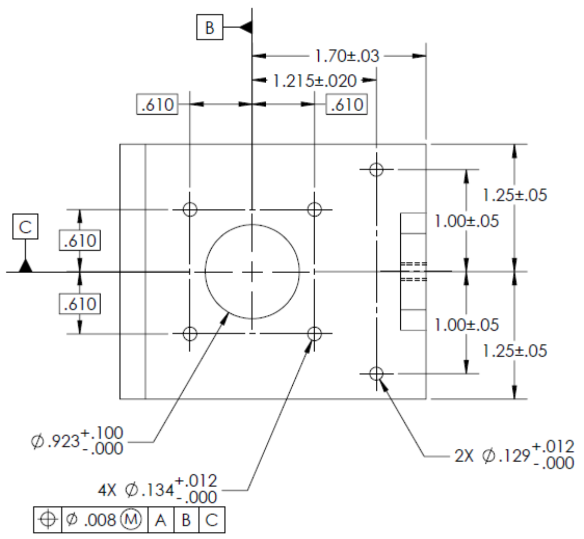
NOTES:

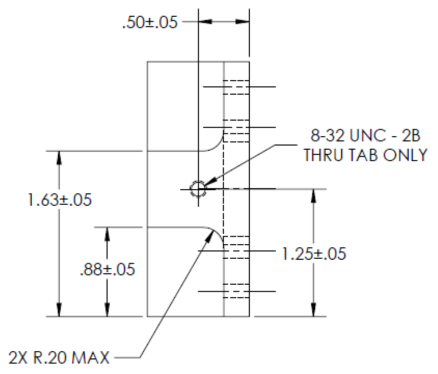
1. ALL DIMENSIONS IN INCHES
2. MATERIAL: BIRCH
3. LASER CUTTING RECOMMENDED
4. BREAK AND DEBURR SHARP EDGES
5. SEE PAPER FOR GEAR TOOTH PROFILE DETAILS



U-CHANNEL STOCK (E.G. McMASTER–CARR 9001K1)

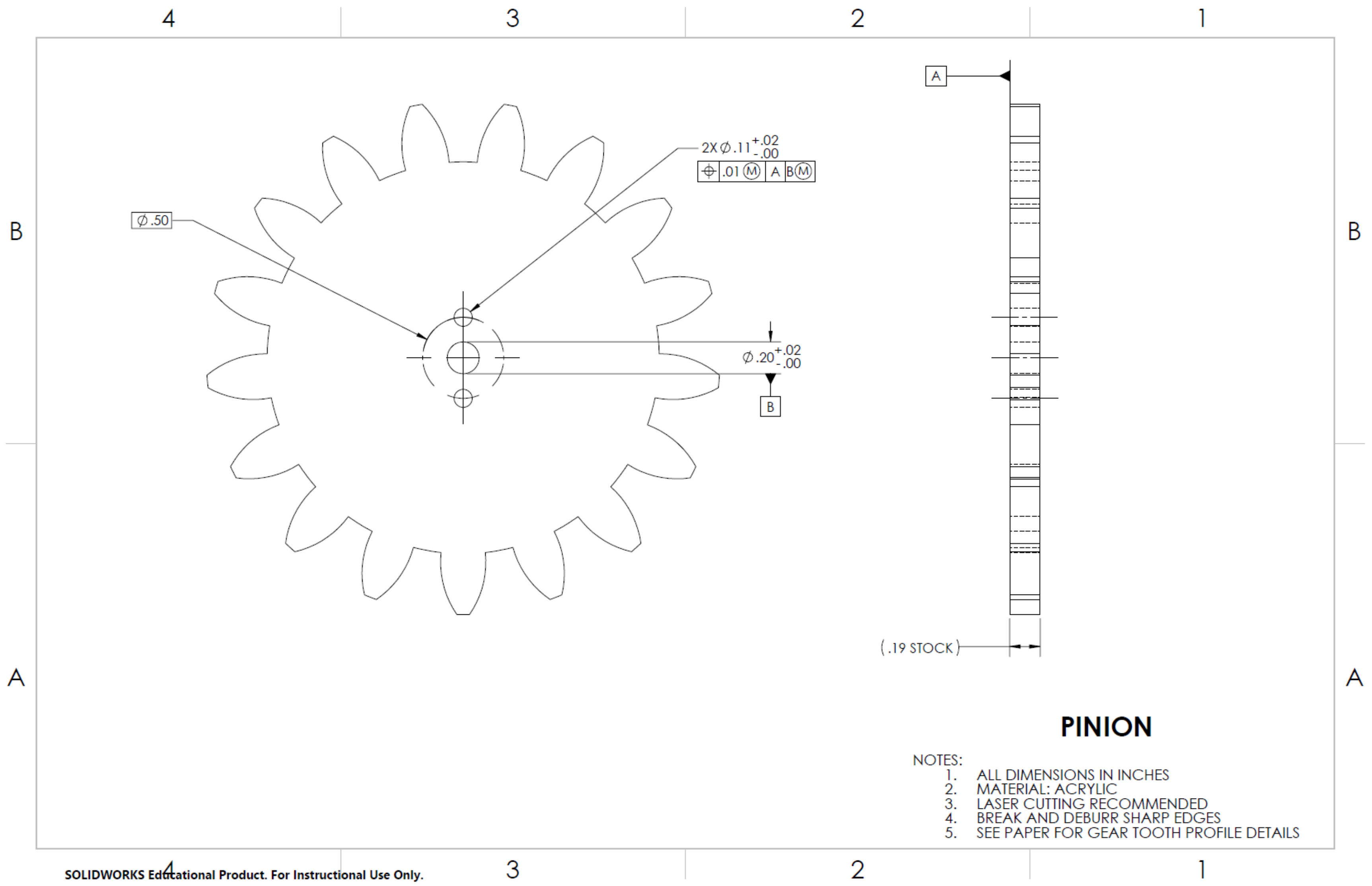






MOTOR BRACKET

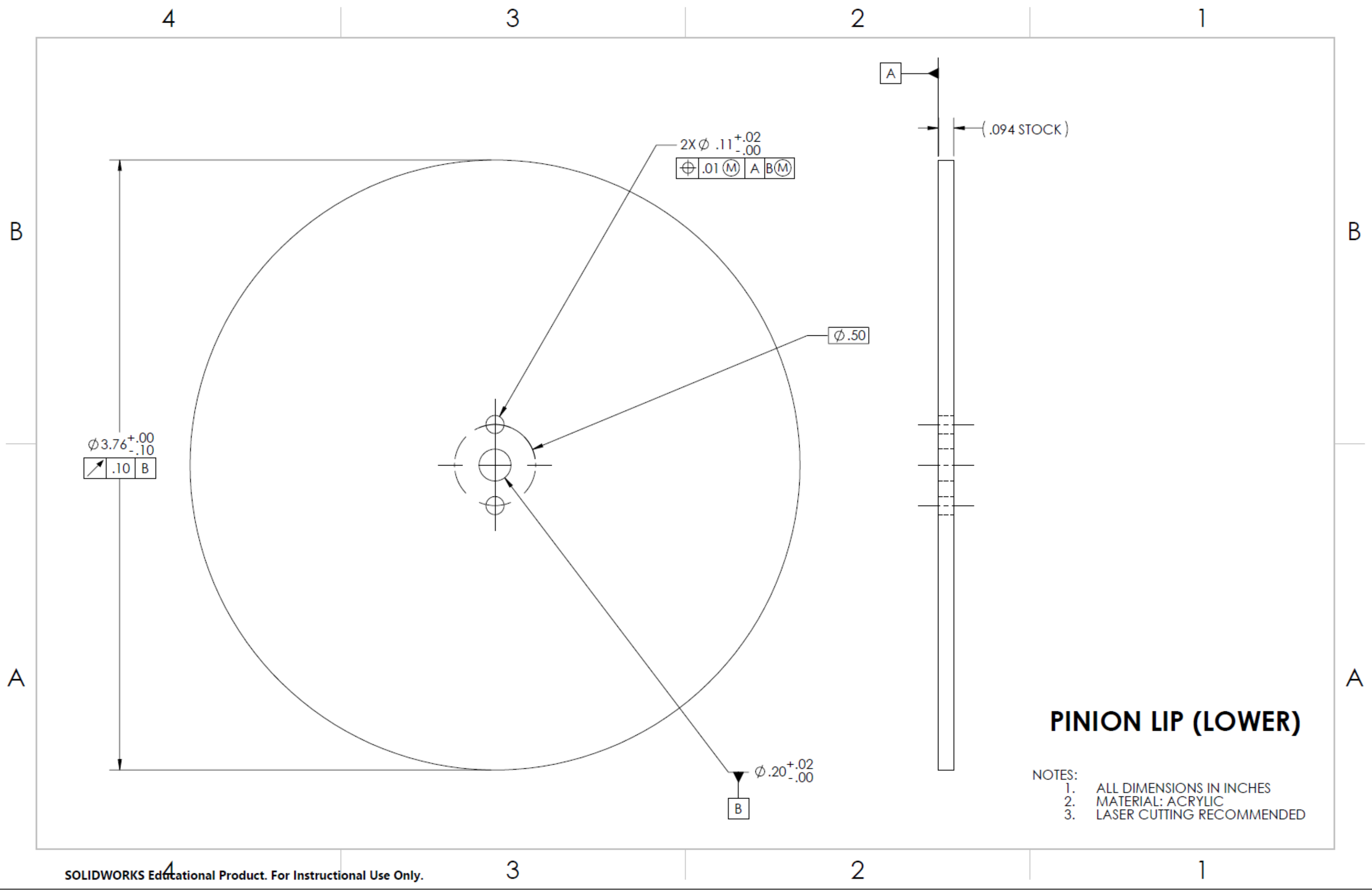
NOTES:
1. ALL DIMENSIONS IN INCHES
2. MATERIAL: ALUMINUM
3. BREAK AND DEBURR SHARP EDGES
4. SURFACE FINISH 250 µIN Ra MAX



PINION

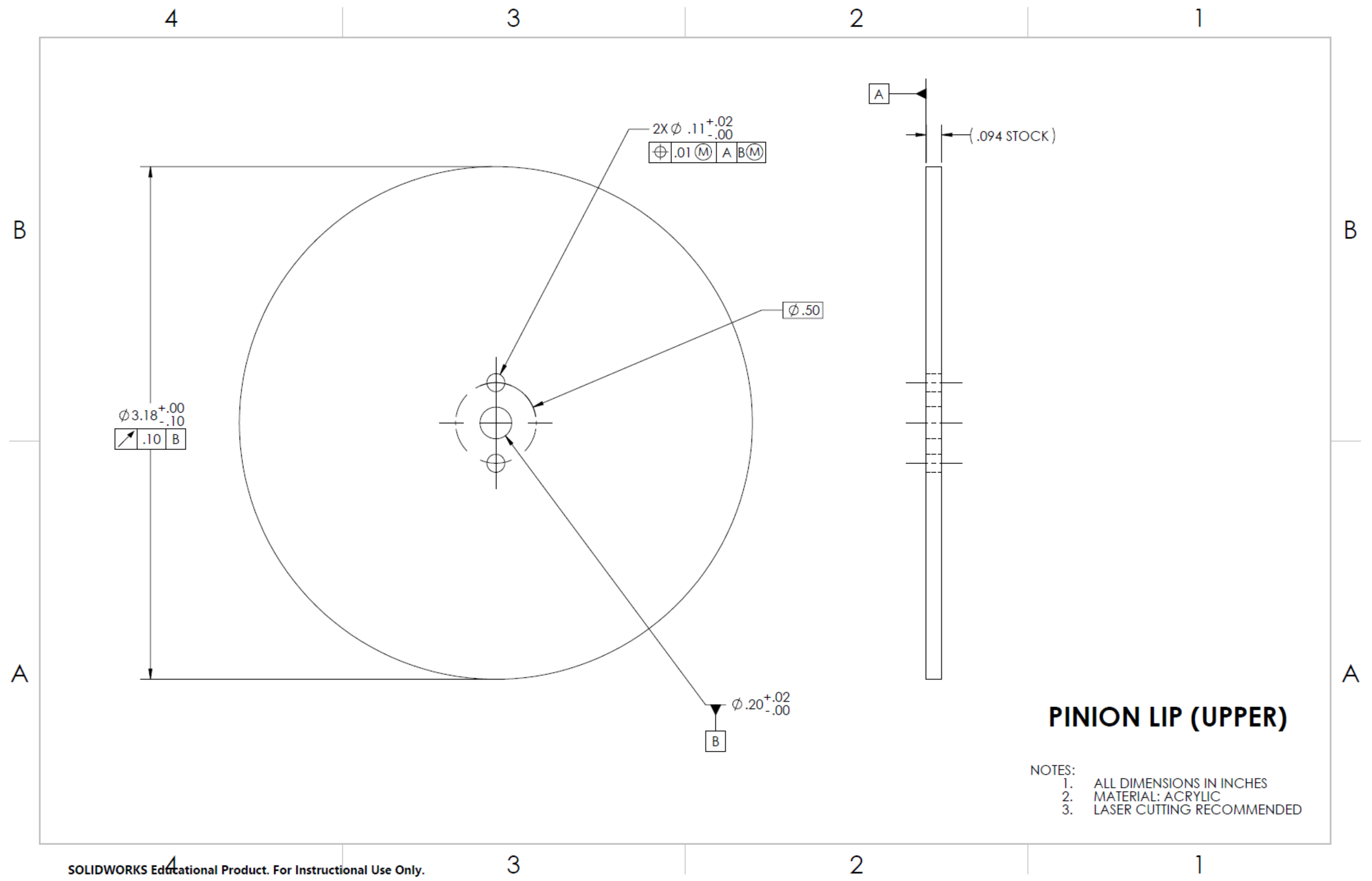
NOTES:

1. ALL DIMENSIONS IN INCHES
2. MATERIAL: ACRYLIC
3. LASER CUTTING RECOMMENDED
4. BREAK AND DEBURR SHARP EDGES
5. SEE PAPER FOR GEAR TOOTH PROFILE DETAILS



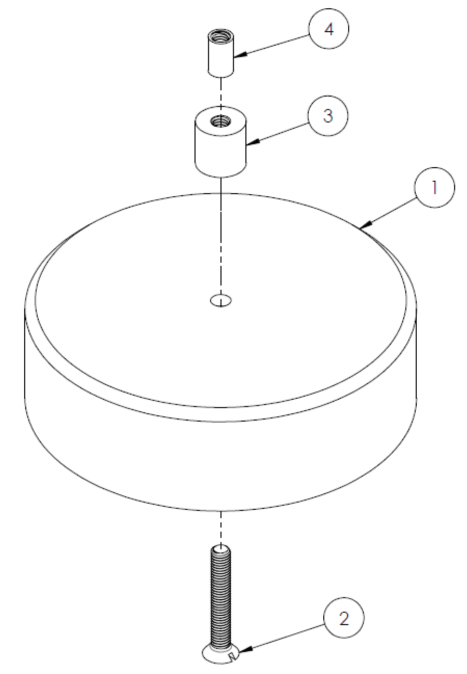
PINION LIP (LOWER)

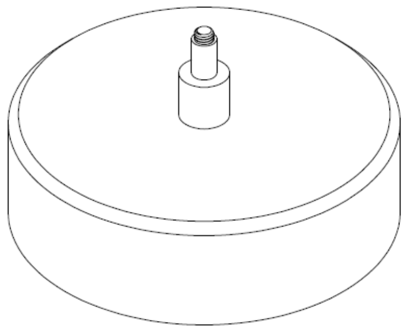
NOTES:
1. ALL DIMENSIONS IN INCHES
2. MATERIAL: ACRYLIC
3. LASER CUTTING RECOMMENDED



PINION LIP (UPPER)

NOTES:
1. ALL DIMENSIONS IN INCHES
2. MATERIAL: ACRYLIC
3. LASER CUTTING RECOMMENDED

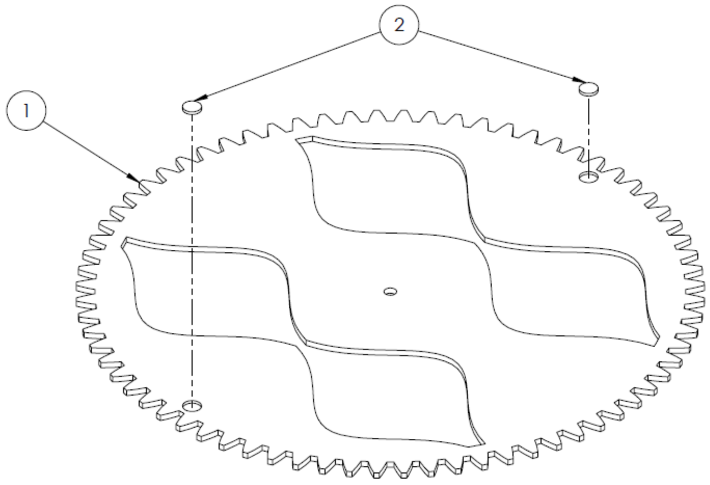


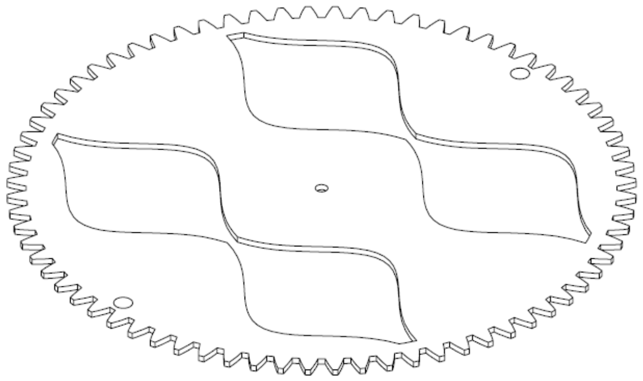


FULLY-ASSEMBLED VIEW

AXLE CAP ASSEMBLY

ITEM NO.	DESCRIPTION	COTS PART EXAMPLE	QTY.
1	AXLE CAP	–	1
2	SS FLATHEAD SCREW, 10-32×1 1/4	McMASTER–CARR #91781A835	1
3	SS STANDOFF, 1/2 OD, 10-32×1/2	McMASTER–CARR #91125A381	1
4	SS STANDOFF, 1/4 OD, 10-32×13/32	McMASTER–CARR #91125A591	1



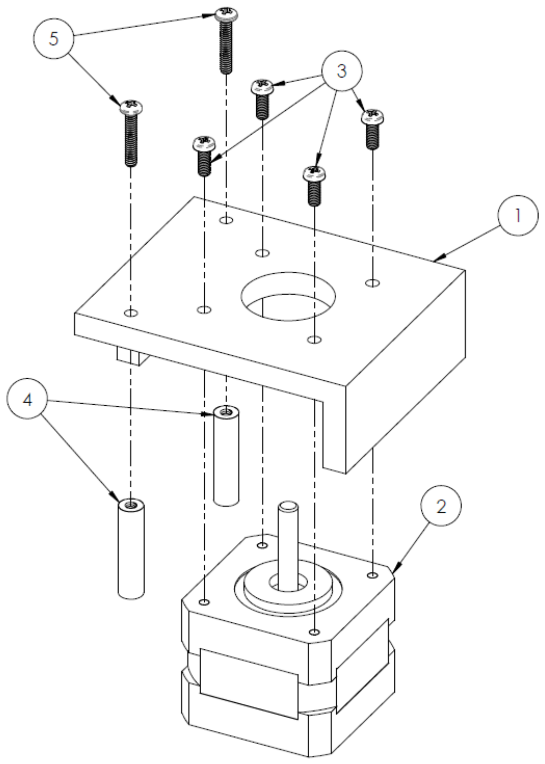


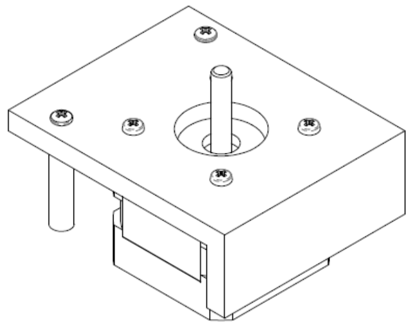
FULLY-ASSEMBLED VIEW

MASK ASSEMBLY

NOTES:
1. AFFIX MAGNETS TO MASK WITH A STRONG GLUE

ITEM NO.	DESCRIPTION	COTS PART EXAMPLE	QTY.
1	MASK	–	1
2	NEODYMIUM MAGNET, 3/8×1/16	APPLIED MAGNETS #ND019	2

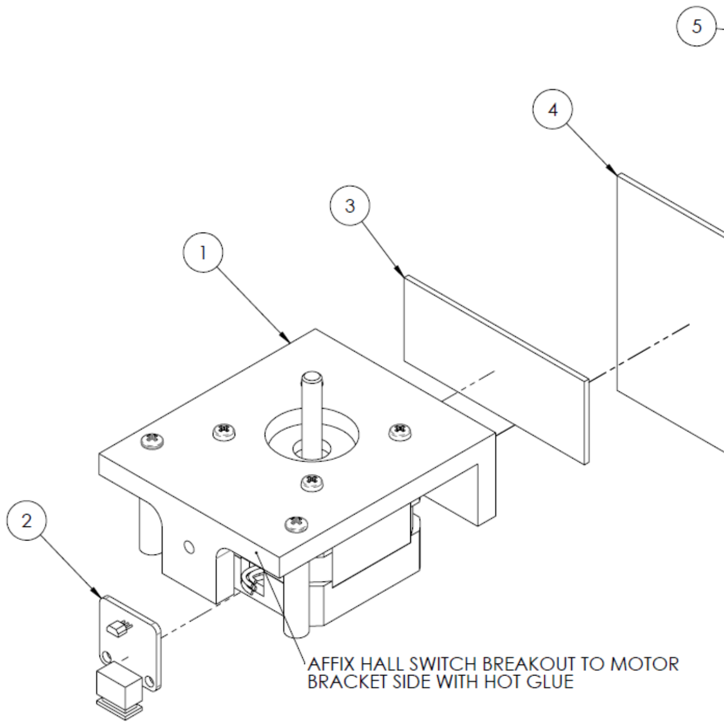


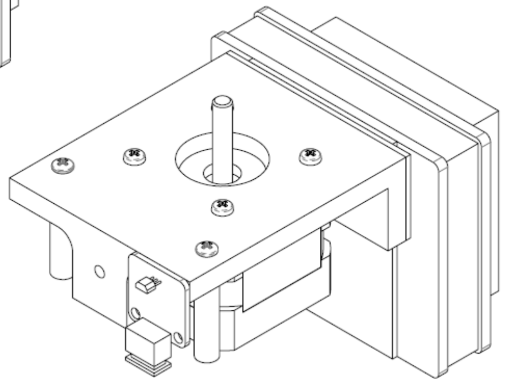


FULLY-ASSEMBLED VIEW

MOTOR BRACKET ASSEMBLY

ITEM NO.	DESCRIPTION	COTS PART EXAMPLE	QTY.
1	MOTOR BRACKET	–	1
2	NEMA 17 STEPPER MOTOR	SPARKFUN #ROB-09238	1
3	SS CHEESEHEAD SCREW M3×0.5×8	McMASTER–CARR #94017A204	4
4	AL STANDOFF, 1/4 OD, 4-40×1	McMASTER–CARR #93330A439	2
5	SS PANHEAD SCREW, 4-40×5/8	McMASTER–CARR #91772A112	2


1
2
3
4
5
AFFIX HALL SWITCH BREAKOUT TO MOTOR BRACKET SIDE WITH HOT GLUE

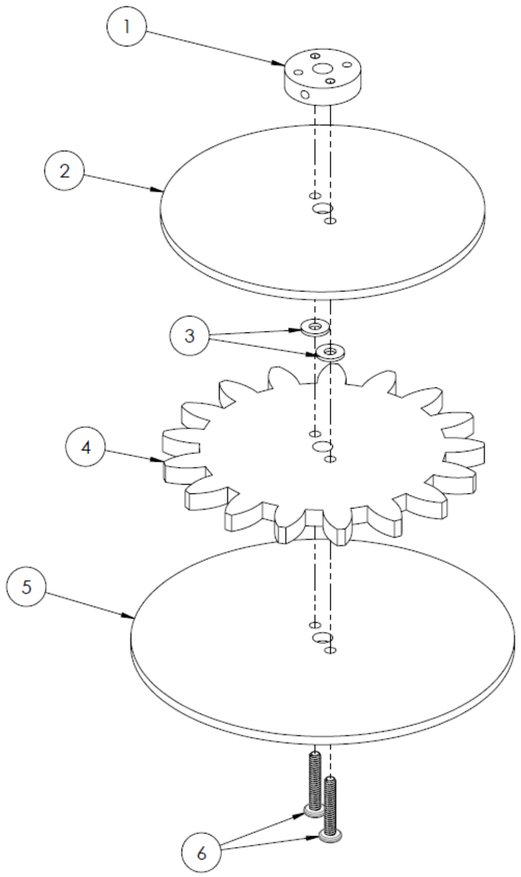


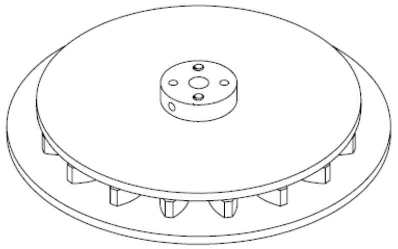
FULLY-ASSEMBLED VIEW

MOTOR BRACKET ELECTRONICS ASSEMBLY

NOTES:
1. WIRES EXCLUDED FOR SIMPLICITY

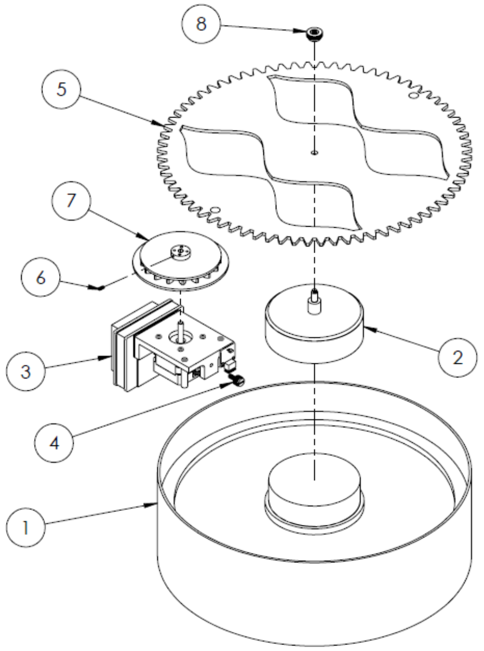
ITEM NO.	DESCRIPTION	COTS PART EXAMPLE	QTY.
1	MOTOR BRACKET ASSEMBLY	–	1
2	HALL SWITCH BREAKOUT	SUNFOUNDER HALL SENSOR MODULE	1
3	2 1/2×1 ADHESIVE-BACK HOOK STRIP	–	1
4	2 37/64×2 ADHESIVE-BACK HOOK STRIP	–	1
5	CASE FOR ARDUINO UNO AND MOTOR SHIELD	–	1

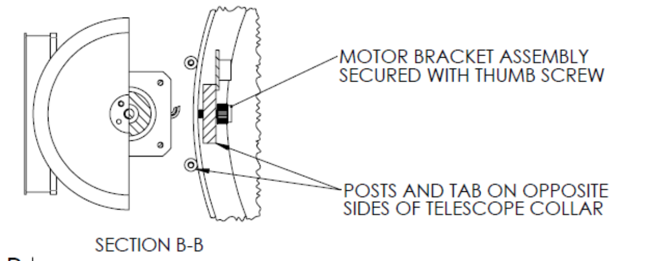



FULLY-ASSEMBLED VIEW

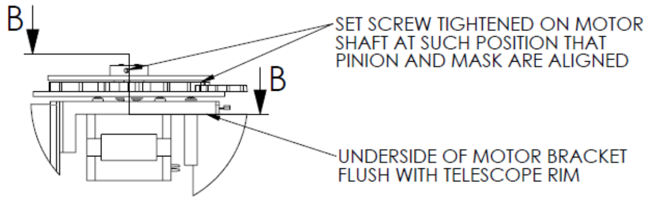
PINION ASSEMBLY

ITEM NO.	DESCRIPTION	COTS PART EXAMPLE	QTY.
1	AL MOUNTING HUB, 5-MM ID; MOUNTING HOLES 4-40, 1/2 BD	SPARKFUN #ROB-10006	1
2	UPPER PINION LIP	–	1
3	NI-PLATED BRASS WASHER, #4, .281 OD	McMASTER–CARR #92917A110	2
4	PINION	–	1
5	LOWER PINION LIP	–	1
6	SS PANHEAD SCREW, 4-40×5/8	McMASTER–CARR #91772A112	2

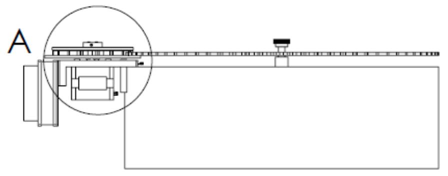



MOTOR BRACKET ASSEMBLY SECURED WITH THUMB SCREW
POSTS AND TAB ON OPPOSITE SIDES OF TELESCOPE COLLAR

SECTION B-B


B
B
SET SCREW TIGHTENED ON MOTOR SHAFT AT SUCH POSITION THAT PINION AND MASK ARE ALIGNED
UNDERSIDE OF MOTOR BRACKET FLUSH WITH TELESCOPE RIM

DETAIL A



FULLY-ASSEMBLED VIEW

ITEM NO.	DESCRIPTION	COTS PART EXAMPLE	QTY.
1	TELESCOPE END	CELESTRON C11	1
2	AXLE CAP ASSEMBLY	–	1
3	MOTOR BRACKET ELECTRONICS ASSEMBLY	–	1
4	NYLON THUMB SCREW, 8-32×3/8	McMASTER–CARR #94320A393	1
5	MASK ASSEMBLY	–	1
6	SS SET SCREW, 4-40×1/4	McMASTER–CARR #92311A106	1
7	PINION ASSEMBLY	–	1
8	PLASTIC THUMB NUT, 10-32 THREAD	McMASTER–CARR #93886A130	1

MASTER ASSEMBLY

NOTES:
1. TELESCOPE END SHOWN TRUNCATED FOR CLARITY