INSTITUT FÜR
INFORMATIK
Computer Vision, Computer
Graphics and Pattern Recognition

Universitätsstr. 1 D–40225
Düsseldorf

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Detection of faint moving objects in sky survey data

Detektion lichtschwacher beweglicher Objekte in Sky Survey Daten

Philipp Schlüter

Bachelorarbeit

Beginn der Arbeit:	15. Januar 2015
Abgabe der Arbeit:	15. April 2015
Gutachter:	Prof. Dr. Stefan Harmeling
	Prof. Dr. Stefan Conrad

Erklärung

Hiermit versichere ich, dass ich diese Bachelorarbeit selbstständig verfasst habe. Ich habe dazu keine anderen als die angegebenen Quellen und Hilfsmittel verwendet.

Düsseldorf, den 15. April 2015

Philipp Schlüter

Abstract

A huge number of digital sky images are available to scientists world wide thanks to various sky survey programs in the last two decades. Detection of moving objects in this data is performed by fitting models to images from different points in time. If an object is too faint and too fast to be detectable on any single image or on the coadd of the single images, the necessary initial guess of parameters can not be made. We present the X-ray transform as a method to find some of these objects. We use the X-ray transform to sum up values along straight lines in a sequence of images. If an object moves along this lines, its signal adds up. If no object moves along a certain line, the random noise cancells out. The methods capabilities and limitations are tested on artificial data sets. Our method is capable of detecting objects in artificial data sets with signal-to-noise ratios smaller than 3 that move by more than two pixels in between single images. We apply our algorithm to raw data from the Wide Infrared Survey Explorer (WISE) and rediscover known objects. The code we created is publicly available at `https://github.com/harmeling/hhu_philipp_schlueter_bsc`.

Contents

1 Introduction

Faint astronomical objects are distant or have low luminosity. Both categories of objects are subject to scientific exploration. Distant objects as galaxies and quasars give insight into the young universe. Low luminosity objects as brown dwarfs, low mass stars and dwarf planets give insight into formation of stars, solar systems and planets.

Nearby galaxies and luminous stars can be observed with the naked eye or simple telescopes. Astronomical catalogues include millions of such objects. The first brown dwarf was only detected in 1995 and the number of confirmed brown dwarfs today is in the order of several hundred [Bas00]. That number is too small to perform proper statistics. So the among others the question remains how the Initial Mass Function can be continued in the substellar region.

Infrared telescopes and satellites scan the sky and produce huge amounts of data. Developing new methods to find brown dwarfs in the existing data is important. Applying those new methods is a cheap and fast addition to the search with advanced observation equipment. Even great computational effort is justified to find additional brown dwarfs, because even excessive computation time is usually cheaper than observation time.

Detecting faint non-moving objects is easy. You just need a number of images of the same sky region. The images are placed above each other and the pixel-wise sum or mean is calculated. The signals of a faint object add up while random noise cancels out. For each pixel the mean μ and the standard deviation σ over all images can be determined.

We call the ratio of these values the signal-to-noise ratio

$$SNR = \frac{\mu}{\sigma}. \tag{1}$$

This ratio increases with the square root of the number of images N as the standard deviation decreases

$$\sigma \propto \frac{1}{\sqrt{N}} \Rightarrow SNR \propto \sqrt{N}. \tag{2}$$

Thus a sufficient level of confidence for a detection can be achieved by increasing the number of images.

Detecting moving objects is based on fitting a model to data by reducing the residual sum of squares, [LHJR09]. A model is a function of parameters describing a movable object. These parameters may include shape, luminosity, parallax, proper motion, position at a certain epoch and additional properties. An initial guess for all parameters is made and the model predicts a brightness value for

all pixels off all images. The residual sum of squares, which is the sum over the squares of the differences between the prediction and the data, is calculated. Optimal parameters are found by performing a gradient descent towards a minimized residual sum of squares.

The algorithms success depends on the quality of the initial guess. Objects that are too faint to be detected on any single image and to fast to be detected by straight adding of several images may be found by testing initial guesses in a brute-force way.

We present, implement and test one method to find moving faint objects in the data, that may not have been found so far.

Images taken at different epochs are again placed above each other. Instead of calculating sums of matching pixels we calculate sums along various straight lines. Statistical noise is cancelled again. Objects moving along these lines produce peaks in the parameter space whose parameters characterize the straight lines.

The problem of detecting moving objects on a series of images is such reduced to a peak detection problem.

For continuous functions $f\colon \mathbb{R}^3 \to \mathbb{R}$ there is the X-ray transform. The X-ray transform integrates over f along straight lines. For discrete images the transform is not defined. We use an analogous approach by projecting each pixel along straight lines onto a plane instead of integrating.

To clarify this analogy we will first present a discrete analogy to the Radon transform. The Radon transform is an integral transform over continuous functions in $\mathbb{R}^2$. Afterwards we will present our discrete implementation of the X-ray transform.

We will then describe our algorithm that makes use of this implementation to find moving objects.

We performed detailed tests of our algorithm on artificial data sets with varying positions and velocities of artificial stars and various signal-to-noise ratios. We show the algorithms capabilities and limitations.

We applied our algorithm to images from the Wide-Field Infrared Survey Explorer (WISE) [WEM$^+$10]. WISE was a satellite mission equipped with infrared sensors mapping the whole sky. For a lot of points on the sky there are several exposures distributed over a period of about half a year available. We used code from [Lan14] for preprocessing the raw data and searched with our algorithm for objects in these images. We will show that our algorithm can be used to find various object on the same images by simple deletion of a found object followed by a second run.

2 Method

2.1 Radon transform

The Radon transform $R\colon \mathbb{R}^2 \to \mathbb{R}^2$ is a transformation on two dimensional functions $f\colon \mathbb{R}^2 \to \mathbb{R}$. It is defined as the integral over straight lines L in $\mathbb{R}^2$

$$R_f(L) = \int_L f(x,y)\,dxdy. \tag{3}$$

Straight lines in $\mathbb{R}^2$ can be parameterized with two numbers, an angle α and the distance s between the line and the origin. The angle α is the angle between the normal vector to L and the x axis. The point on L closest to the origin is

$$\begin{pmatrix} x \\ y \end{pmatrix} = s \begin{pmatrix} \cos\alpha \\ \sin\alpha \end{pmatrix}. \tag{4}$$

So every point on the line can be parameterized by

$$\begin{pmatrix} x(t) \\ y(t) \end{pmatrix} = t \begin{pmatrix} \cos(90^\circ - \alpha) \\ -\sin(90^\circ - \alpha) \end{pmatrix} + s \begin{pmatrix} \cos\alpha \\ \sin\alpha \end{pmatrix} \tag{5}$$

$$= t \begin{pmatrix} \sin\alpha \\ -\cos\alpha \end{pmatrix} + s \begin{pmatrix} \cos\alpha \\ \sin\alpha \end{pmatrix} \tag{6}$$

Thus the definition can be rewritten as

$$R_f(L) = \int_{-\infty}^{\infty} f\left((t\sin\alpha + s\cos\alpha), (-t\cos\alpha + s\sin\alpha)\right) dt. \tag{7}$$

The Radon transforms most prominent application is in computed tomography. A radiation source and radiation sensors are moved around an object or a human body in medical applications. The object between source and sensors absorbs the radiation. The measured absorption for a certain position of source and sensor equals the integral over the objects absorption coefficient along the straight line connecting source and sensor.

This absorption is measured for various angles and offsets. The underlying distribution of absorption coefficients within the object can be calculated with the inverse of the Radon transform.

Another example of applications is found in the field of geosciences where Radon transform is used to examine the inner earth structure using earthquakes as signal sources.

There are several methods to translate the Radon transform from continuous functions to discrete functions such as images (see for instance [ACDI03] and

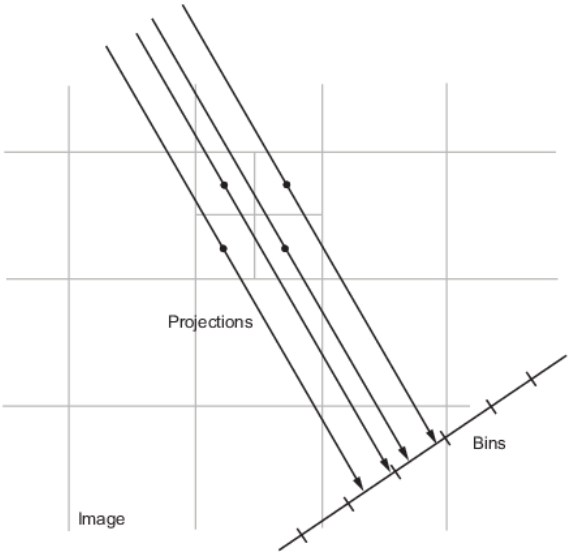

Figure 1: Each pixel is projected along a straight line. The pixels value is distributed among the nearest bins. Image source: [Mat15]

references therein). We present an algorithm that calculates an analogy to the Radon transform by projecting pixel values along straight lines onto a plane instead of integrating over values along lines. This is the algorithm implemented in Matlab's and Octave's image processing packages [Mat15].

We independently calculate the Radon transform of a picture for any given angle ϕ to the x-axis. We project every pixel along a line with angle ϕ onto a projection plane. The origin lies on the projection plane and the projection plane is orthogonal to ϕ.

The projection plane has a certain binning. The value of a pixel is distributed linearly among the two bins closest to the point of projection. If the projection hits the center of a bin, the pixels value is completely added to the bins value. If the projection hits a bins boundary one half of the pixels value is added to both bins values.

2.2 X-ray transform

The X-ray transform is the integral transformation over straight lines L in $\mathbb{R}^3$ on three dimensional functions $f\colon \mathbb{R}^3 \to \mathbb{R}$

$$X_f(L) = \int_L f(x,y,z)\, dxdydz. \tag{8}$$

Straight lines in $\mathbb{R}^3$ can be parameterized using four parameters. For example using a three dimensional vector describing the lines direction and a scalar value

describing the distance of the line from the origin (this is the radius of a sphere with the straight line as a tangent).

If we omit straight lines parallel to the x-y-plane we can easily see, that straight lines can also be parameterized using two angles and two offset parameters. The former describe the lines direction the latter the point on the x-y-plane that the line crosses. We will call the angles θ and ϕ, the location parameters x_{offset} and y_{offset}.

Similar to the aforementioned discrete analogy to the Radon transform, we perform the discrete X-ray transform by projection rather than integration. In three dimensions we project onto a plane in contrary to the projection onto a line in two dimensions. This plane is gridded and the values of the projected pixels are distributed to the grid cells according to the exact location the pixel is projected to. If the pixel is projected onto the center of a grid cell the pixels value is completely added to this cells value. If a pixel is projected onto a point where four grid cells are adjacent, the pixels value is distributed evenly on all four grid cells. For all in-between cases the pixels value is distributed linearly to the four nearest grid cells.

2.3 Algorithms

2.3.1 singleXray

The algorithm singleXray calculates the X-ray transform for a given pair of angles θ and ϕ of a stack of Z images of size X by Y.

We set the output to be of size $2b+1$ by $2b+1$ where

$$b = \lceil 1 + \frac{1}{2}\sqrt{X^2 + Y^2 + Z^2} \rceil. \tag{9}$$

Thus we can be sure to have a central pixel and the stack of images can be projected onto the output plane even if we project along a line perpendicular to a diagonal of the image stack.

For each pixel $(x, y, z)^T$ of the stack of images we calculate the position on the output on which the pixel is projected.

$$x_{offset} = \left[\begin{pmatrix} x \\ y \\ z \end{pmatrix} - \begin{pmatrix} x_c \\ y_c \\ z_c \end{pmatrix} \right] \cdot \begin{pmatrix} \cos\theta \\ 0 \\ \sin\theta \end{pmatrix} + b + 1 \tag{10}$$

$$y_{offset} = \left[\begin{pmatrix} x \\ y \\ z \end{pmatrix} - \begin{pmatrix} x_c \\ y_c \\ z_c \end{pmatrix} \right] \cdot \begin{pmatrix} -\sin\phi\sin\theta \\ \cos\phi \\ \sin\phi\cos\theta \end{pmatrix} + b + 1 \tag{11}$$

With x_c, y_c and z_c as the coordinates of the center of the stack of images.

The value of the pixel $(x, y, z)^T$ is distributed to the projection planes grid cells at $(\lceil x_{offset} \rceil, \lceil y_{offset} \rceil)$, $(\lfloor x_{offset} \rfloor, \lceil y_{offset} \rceil)$, $(\lfloor x_{offset} \rfloor, \lfloor y_{offset} \rfloor)$ and $(\lceil x_{offset} \rceil, \lfloor y_{offset} \rfloor)$ according to the distance of (x_{offset}, y_{offset}) to the center of these nearest grid cell centers.

Along the diagonals of the imagestack the X-ray transform has a local maximum as more values are added up. Our starFinder algorithm will find objects by finding the maximum of the X-ray transform. For that reason we normalize the result of the transform. Therefore we calculate the X-ray transform of a imagestack with the same shape and with constant density. The X-ray transform of the original imagestack is then divided by the transform of the imagestack with constant density.

This normalization is ill-conditioned for the X-ray transform along lines that pass through the imagestacks edged. The X-ray transform on the imagestack with constant density along these lines may be very small. Dividing by this small values produces wrong normalization for those lines. Our starFinder algorithm will thus reject those values.

See Appendix singleXray on page 29 for the source code, written in C++.

2.3.2 Xray

The Xray algorithm calls several instances of singleXray and concatenates the results. It takes vectors of θ and ϕ in units of degree as an argument, where singleXray takes scalar values of θ and ϕ in units of radians.

This distinctions seems unnecessary complicated but for convenience's sake we wanted our Xray algorithm to match the behaviour of the radon function from Octave's image package that also takes an angle vector in units of degree.

An optional vector of dates can be included with values corresponding to the dates the images were produced. Otherwise equal spacing with distance one between the images is assumed.

See Appendix Xray on page 31 for the source code, written in Octave.

2.3.3 findStar

In the first step the Xray transform is calculated for eleven values of ϕ and θ equally spaced from $-89.99°$ too $89.99°$:

$$\phi_n = -89.99 + 17.998n, n \in \{0, \ldots, 10\} \tag{12}$$

and θ_n accordingly. Values of $\theta = 90°$ or $\phi = 90°$ correspond to straight lines in the plane of single images. Artifacts on the images like satellites, planes or shooting stars can produce such lines, but this is not what we are looking for. If a line with $\theta = 90°$ or $\phi = 90°$ was found, its velocity could not be calculated, because the tangent function used in (13) and (14) is not defined.

On each of the 121 images we reject the outermost pixels, because their values are off due too the ill-conditioning of normalization performed after transforming. Then we smooth all of the images with a Gaussian filter with $\sigma = 3\,\text{pixel}$. We search the maximum value on all the 21 images. The position of the maximum is at ϕ_m, θ_m, x_m and y_m. We then perform a finer spaced search.

We take some uncertainty of the position of the maximum into consideration and reduce the search space for ϕ to $\phi \in [\phi_{m-2}, \phi_{m+2}]$ if $m-2 \geq 0$ and $m+2 \leq 10$. Else we would reduce the search space to $\phi \in [\phi_0, \phi_{m+2}]$ or $\phi \in [\phi_{m-2}, \phi_{10}]$. The search space for θ is reduced accordingly. Thus the search space for each θ and ϕ is in general reduced by a factor of two, as $\phi_{10} - \phi_0 \approx 2\,(\phi_{m+2} - \phi_{m-2})$. For example the search space may be reduced from $179.98°$ (with $\phi_0 = -89.99°$ and $\phi_{10} = 89.99°$) to $89.99°$ (with $\phi_m = 0°$,$\phi_{m-2} = 35.996°$ and $\phi_{m+2} = -35.996°$).

We calculate estimates for $\tilde{v}_x$ and $\tilde{v}_y$ using our conservative estimates $\phi \in [\phi_{m-2}, \phi_{m+2}]$ and $\theta \in [\theta_{m-2}, \theta_{m+2}]$. For certain values of θ and ϕ we calculate $\tilde{v}_x$ using

$$\tilde{v}_x = -\tan\theta \tag{13}$$

and $\tilde{v}_y$ using

$$\tilde{v}_y = -\tan\phi/\cos\theta. \tag{14}$$

The uncertainties of ϕ and θ propagate to $\tilde{v}_x$ and $\tilde{v}_y$. Equations (13) and (14) are true for both equally and arbitrary spaced image sequences. In the equally spaced scenario the velocity is given in units of pixel-per-image. Otherwise its unit depends on the unit of the images dates. For dates given in years the unit of $\tilde{v}_x$ would be pixel-per-year, for dates given in seconds pixel-per-second.

We calculate a first estimate for the stars position on the first image using (10) solved for x (derivation in the appendix on page 28)

$$\tilde{x}_0 = \frac{x_m - (z_1 - z_c)\sin\theta - b - 1}{\cos\theta} + x_c \tag{15}$$

and using (11) solved for y (derivation in the appendix on page 28)

$$\tilde{y}_0 = \frac{y_m + (\tilde{x}_0 - x_c)\sin\phi\sin\theta - (z_1 - z_c)\sin\phi\cos\theta - b - 1}{\cos\phi} + y_c. \tag{16}$$

with b as defined in (9) and z_1 the z-coordinate of the first image (e.g. the date when the first image was taken).

We try to use these estimates to reduce the size of images by neglecting those parts where the star would not appear, considering our estimates for position and velocity. In the first iteration this usually does not happen, because of the coarse spacing of ϕ_n and θ_n and the resulting large uncertainties for all derived parameters.

This procedure is repeated ten times. For the iterations two to ten the X-ray transform is calculated for seven values of both θ and ϕ, equally spaced within in uncertainty range of the prior iterations estimate, instead of eleven values as in the first iteration. Calculating the X-ray transform for less angles is faster. But using only seven values each for the angles in the first iteration results in a too coarse spacing.

After the last iteration the algorithm returns estimates for the stars position on the first image $\tilde{x}_0$ and $\tilde{y}_0$ and the stars velocity $\tilde{v}_x$ and $\tilde{v}_y$ and their according estimates.

See Appendix findStar on page 32 for the source code, written in Octave.

3 Tests on artificial data

3.1 Data generation

Each set of artificial data consists of 100 equally spaced images because we will test our algorithm on images from the Wide Infrared Survey Explorer (WISE) and WISE has a median number of exposures of about 30 for any point on the sky and up to thousand exposures close to the ecliptic poles. See section WISE on page 21 for more information on the WISE mission.

Each image is 200-by-200 pixel in size. In our test with WISE data, we can see that images with 200-by-200 pixels are large enoug to contain at least ten resolvable objects. For larger images the computational effort necessary to perform the test would increase and we would have to reduce the number of tests we perform.

So a set of artificial data is basically a function

$$f\colon\ [1,\ldots,200]^2 \times [1,\ldots,100] \to \mathbb{R} \tag{17}$$

There is one artificial star and Gaussian noise.

The set is described by 5 parameters, listed in Table 1.

Parameter	Description
x_0	X-position of the artificial Star on the first image
y_0	Y-position of the artificial Star on the first image
v_x	Change in X-position from one image to the next
v_y	Change in Y-position from one image to the next
SNR	Signal-to-Noise-Ratio

Table 1: Data-set parameters

The center of the artificial star on image i can be calculated by $x_\star = x_0 + iv_x$ and $y_\star = y_0 + iv_y$ where $i \in \{1,\ldots,100\}$. On each image the star is a two dimensional Gaussian centered at $(x_\star, y_\star)$ with $\sigma_\star = 1$, normalized to have the value SNR at the center.

Gaussian noise with $\mu_{noise} = 0$ and $\sigma_{noise} = 1$ is added to all pixels. In astronomical raw data there are no negative values as intensities or fluxes are measured. But choosing μ is arbitrary, as any given image can be flattened by subtracting a global μ. Often images are flattened by local means to reduce brightness gradients that are introduced by sources like the moon.

3.2 Execution

We generated 6565 data sets and examined them with the findStar algorithm.

For each data set we choose the parameters from Table 1 randomly. x_0 and y_0 are integers, chosen from the set $\{0, \ldots, 200\}$. Thereby we omit scenarios in which a star is not on the first image but moves into the range of the images. But this case is symmetric to data sets where a the star is on the first image and then leaves the visible range. Actually we are only omitting scenarios where a star is neither on the first nor on the last image but only on some of the inner images.

v_x and v_y are normally distributed with $\mu_v = 0$ pixel and $\sigma_v = 1$ pixel. The signal-to-noise ratio is chosen from the set $\{2, 2.25, \ldots, 4.75, 5\}$.

A data set with the chosen parameters is generated. The findStar algorithm is applied to the data set. The parameters and the algorithms return-values are saved for later evaluation.

This took about 7 days on a notebook with an Intel(R) Core(TM) i5-4200M CPU, 16 GB of RAM and a 64 bit version of Ubuntu 14.10.

3.3 Analysis

The algorithm returns estimates for the stars position ($\tilde{x}_0$ and $\tilde{y}_0$) and velocity ($\tilde{v}_x$ and $\tilde{v}_y$) and estimates for the uncertainty of the guesses (σ_{x_0}, σ_{y_0}, σ_{v_x} and σ_{v_y}). We compare these estimates with the parameters used at data generation. We call the estimate correct, if the two parameters x_0, y_0 lie within a range of five times the uncertainty of the estimate

$$x_0 \in [\tilde{x}_0 - 5\sigma_{x_0}, \tilde{x}_0 + 5\sigma_{x_0}] \wedge y_0 \in [\tilde{y}_0 - 5\sigma_{y_0}, \tilde{y}_0 + 5\sigma_{y_0}]\,. \tag{18}$$

Of the 6565 data sets we generated 4120 do have correct estimates. That is $63\,\%$. This definition of a correct estimate will be used in our plots to distinguish between correct and incorrect identifications. Correct identifications will be marked with a green $+$ and incorrect identification with a red $\times$.

In Figure 2 we plot $\tilde{x}_0$ estimate by actual x_0, accordingly for $\tilde{y}_0$ and y_0 in Figure 3. We see that choosing a range of five times the uncertainty of the estimates and the uncertainties itself are reasonable, as the distribution of correct identified stars around the diagonal is not very wide and there is only a small number of stars close to either of the diagonals that are identified as incorrect. We can see that stars with very small or large x_0 and y_0 are more likely to be identified incorrectly. This observation will be followed up in the next plots.

The estimated positions of incorrect identified stars are either close to the images edges or uniformly distributed.

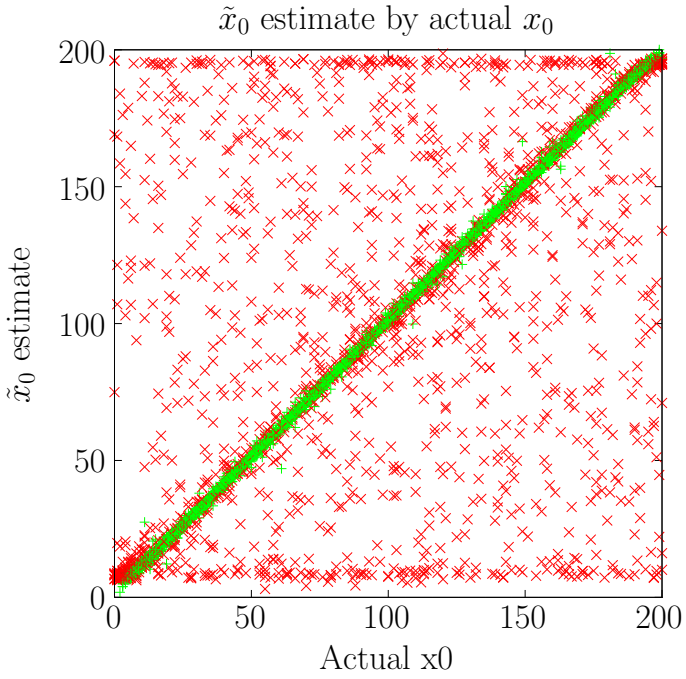

Figure 2: The uncertainty σ_{x_0} is usually small, so there a only few stars considered correct off the diagonal.

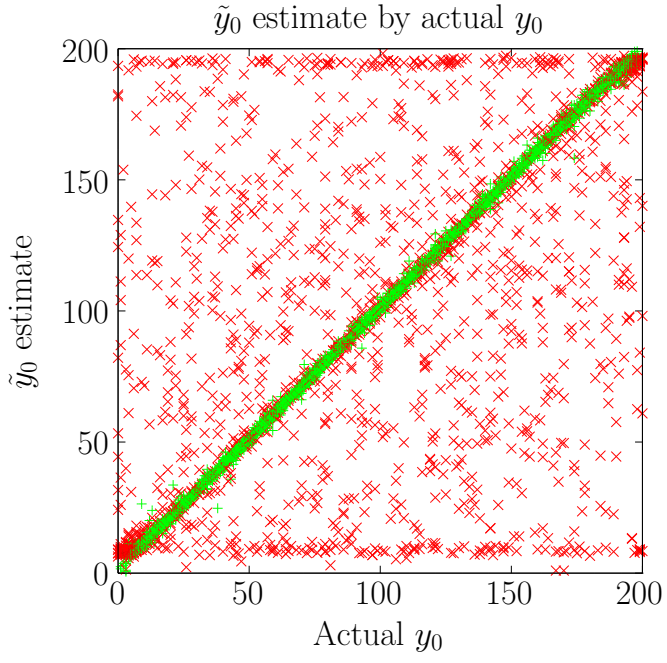

Figure 3: If a star is identified wrong, $\tilde{y}_0$ is distributed evenly or close to the images edges.

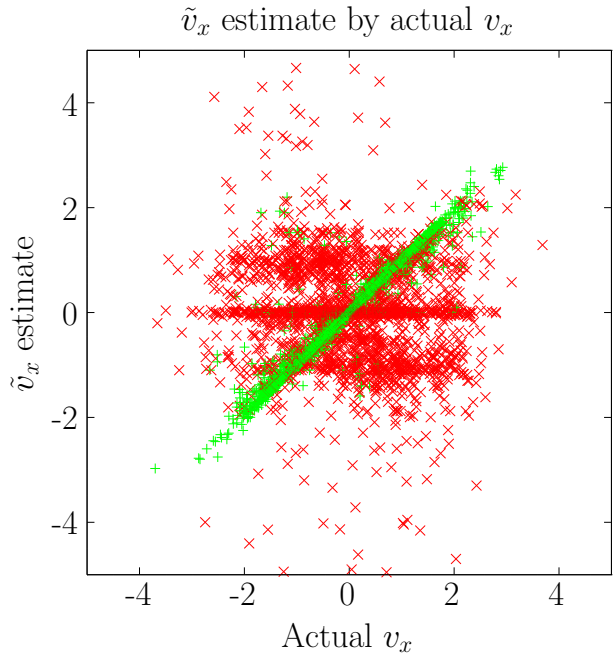

Figure 4: For wrong identifications the density of $\tilde{v}_x$ is increased around zero.

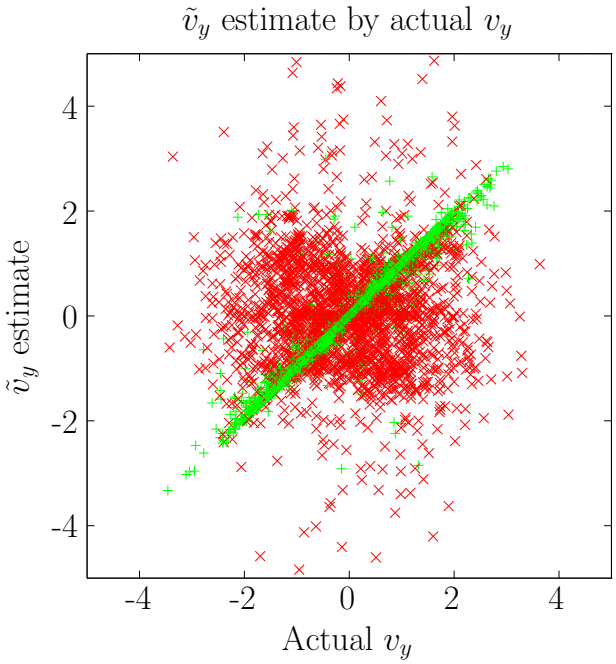

Figure 5: Wrong values of $\tilde{v}_y$ are distributed evenly

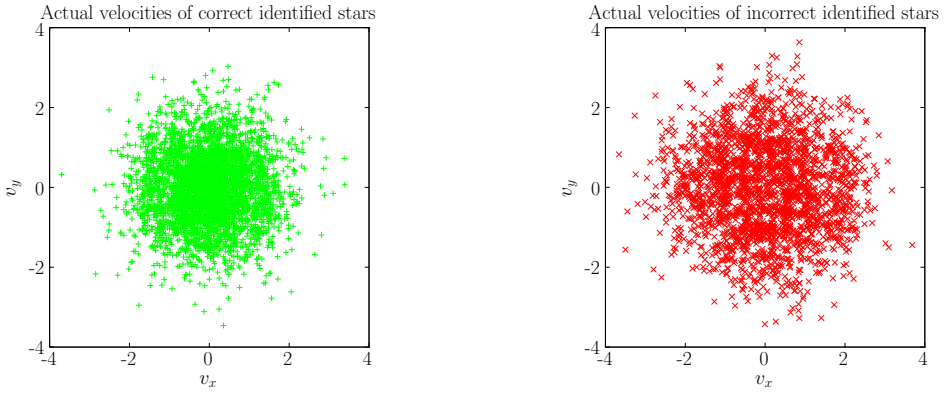

Figure 6: The distribution of velocities is isotropic for correct and incorrect identified star. The variance of the distribution of incorrect identified stars is larger, because the probability of correct identification decreases with increasing velocity.

In Figure 4 and 5 we show the dependence of the velocity estimates $\tilde{v}_x$ and $\tilde{v}_y$ to the actual velocities v_x and v_y. The number of correct identified stars with $\tilde{v}_x$ differing from the actual v_x or $\tilde{v}_y$ differing from the actual v_y is reasonable small. The velocity estimate $\tilde{v}_y$ for incorrect identified stars is scattered evenly, while the $\tilde{v}_x$ has a tendency of being close to zero. We suspect that this is because we calculate $\tilde{v}_x$ in (13) using tangent and handling the angles internally with units of degrees instead of radians and converting just before taking the tangent. Thus the functions slope

$$\frac{d}{d\phi}\tan\left(\frac{\pi\phi}{180^\circ}\right) = \frac{\pi}{180^\circ}\tan^2\left(\frac{\pi\phi}{180^\circ}\right) \tag{19}$$

is shallow for small values of ϕ with

$$\tan'(0^\circ) \approx 0.017/1^\circ. \tag{20}$$

This causes a wide range of values of ϕ to correspond to values of $\tilde{v}_x$ close to zero.

Figure 6 show the actual velocities of correct and incorrect identified stars. Both distributions are isotropic. Anisotropy in either of the distributions would be a strong indication for an error in the algorithm, because we created the artificial data isotropic sets with both normal distributed v_x and v_y.

Both figures have the same scale on the axis to show that the variance of velocities for incorrect identified stars is larger than the variance for correct identified stars. We will show in Figure 11 that the fraction of correct identifications decreases with speed.

In Figure 7 we show the correct detection rate by SNR. There is a linear increase

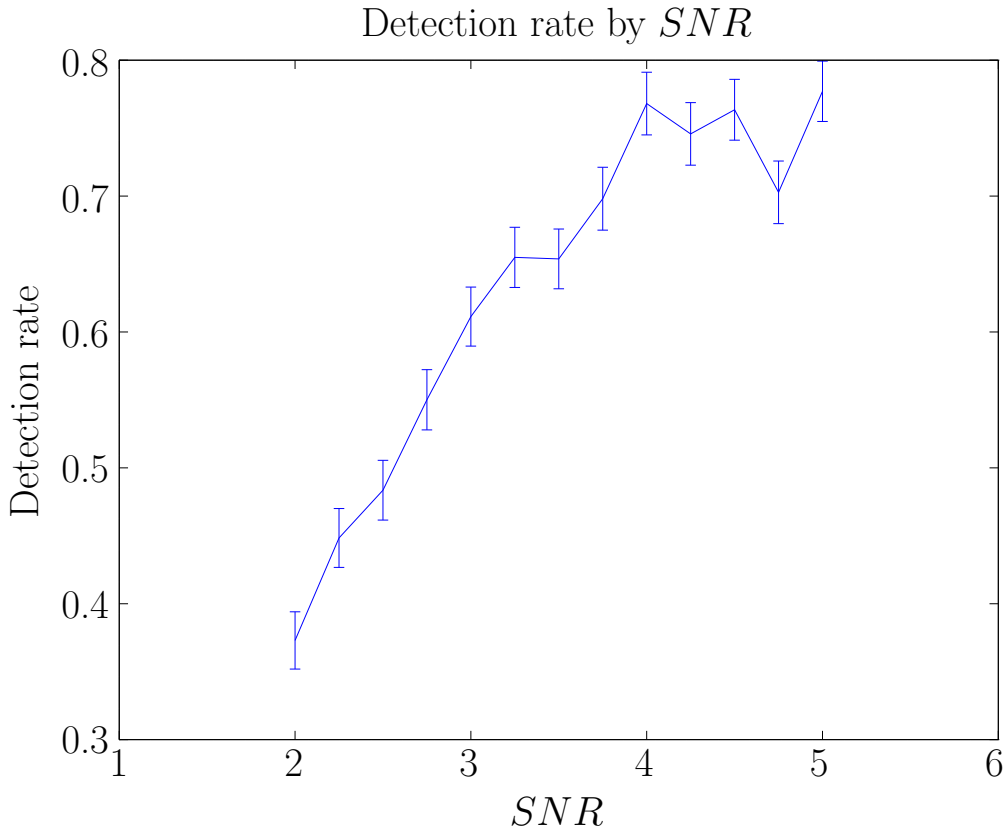

Figure 7: The detection rate plateaus for $SNR \geq 4$ and declines linearly for smaller SNR.

from $(37 \pm 3)\%$ at $SNR = 2$ to $(77 \pm 3)\%$ at $SNR = 4$. From there on detection rate seems to be constant for increased signal-to-noise ratio.

In Figure 8 we show the visibility of an object with different signal-to-noise ratios in single images. The images were created with the same algorithm that produces all artificial data sets. Four data sets with $x_0 = 100$, $y_0 = 100$, $v_x = 0$ and $v_y = 0$ and SNR from five down to two were generated. We show the central sector with length of a side of 20 pixels. In all the images the star is at the very center. The color mapping is identical for all images to provide comparability.

At $SNR = 5$ the star is clearly visible. At $SNR = 4$ the star is visible with the knowledge that it is located at the images center. It could be explained by noise. For lower signal-to-noise ratios the star is completely obstructed by noise.

In plots where we show our algorithms abilities and limitations we will compare the result for high-SNR and low-SNR data sets. High signal-to-noise ratios will be defined as $SNR \geq 4$ as this is the point from where on detection is independent from SNR as was shown in Figure 7. Low signal-to-noise ratios will be defined

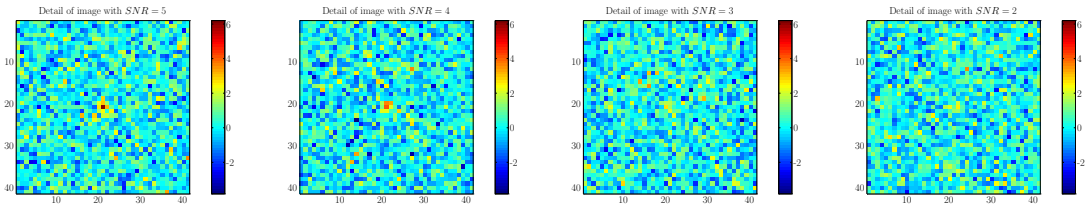

Figure 8: The star is clearly visible with $SNR = 5$. With $SNR = 3$ there are features stronger than the star. With $SNR = 2$ nothing can be seen but noise.

as $SNR \leq 3$. Thus the number of data sets will be of the same size for either case.

In Figure 9 we show the actual positions x_0 and y_0 for all data sets. The positions are evenly distributed as both x_0 and y_0 where uniformly chosen from 0 to 200. We see that most data sets with the stars position close to the edge can not be identified correctly.

We show in Figure 10 how the detection rate decreases with the stars distance from the center on the first image. The fraction is constant up to a distance of about 80 pixels from the the center. For data sets with $SNR \geq 4$ in this inner region $85\,\%$ of stars are identified correctly. From there on this number declines linear to $55\,\%$ at a distance of 125 pixels and then decreases sharply.

The same trend can be seen for data sets with $SNR \leq 3$. For distances smaller than 80 pixels the detection rate is about 40 to 55 %. The claim that this value is constant can be made, but is less strong than for high SNR data sets. Notice the statistical error for these values indicated by error bars in the plot. We can see the same linear decline for distances above 80 pixels and the sharp decline above 125 pixels.

Figure 11 shows the detection rate declining with increased speed of the star. In data sets with high SNR we detect 90 % of the slowest moving stars (speed smaller than 0.1 pixel per image). This value decreases to 45 % at a speed of 3 pixels per image. The number of data sets with speed of more than 3 pixels per image is too small and the statistical errors thereby too large to make any meaningful statements about the detection rate for speed above 3 pixels per image.

We assume that the main reason for the declining detection probabilities with increased velocity and distance of the star from the center is due to the fact that the star may leave the visible range. In Figure 12 we show the detection rate by the number of images the star appears.

A star with a position on the first image $x_0 = 190$ and $v_x = 1$ could only be visible on the first ten images, independent of y_0 and v_y. Only if the star is within the images boundaries according to both x and y position it is considered to be

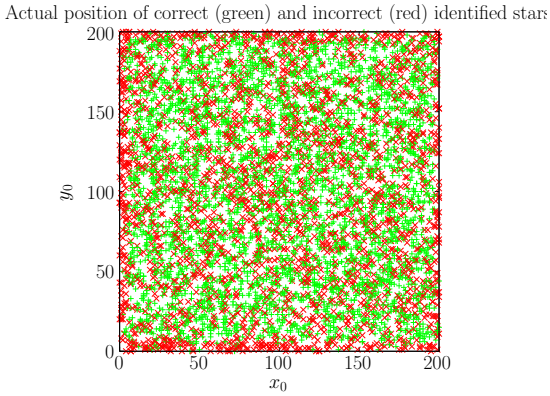

Figure 9: Correct and incorrect identified stars are evenly distributed in the images center. The number of incorrect identified stars increases sharply close to the edges.

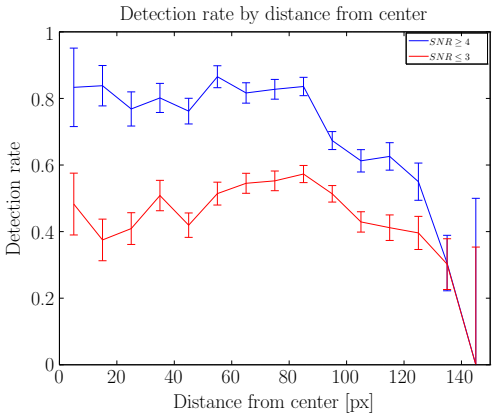

Figure 10: The detection rate is constant up to about a distance of 80 pixels from the center, declines linearly up to about 125 pixels and then sharply.

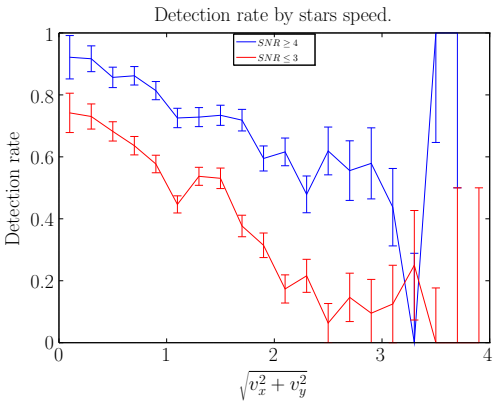

Figure 11: The detection rate decreases linearly to up to a speed of 3 pixels per image.

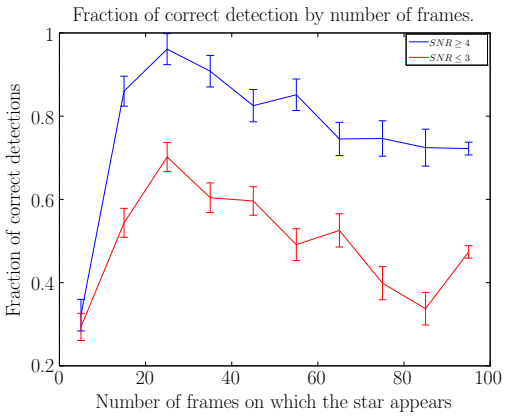

Figure 12: The detection rate is surprisingly maximized, if the star appears on 20 to 30 images.

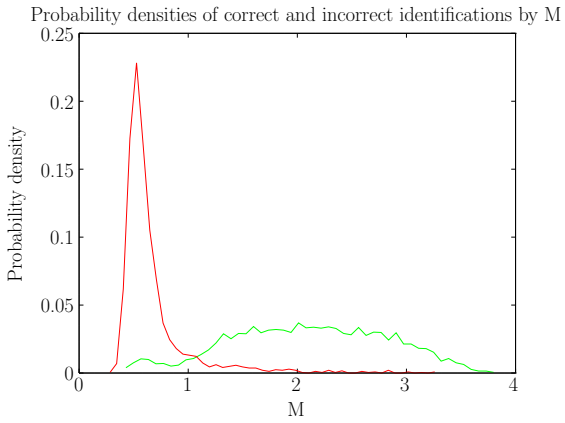

Figure 13: Correct and incorrect estimates can be distinguished using M.

visible on the image.

To our surprise the maximum is reached if the star appears on 20 to 30 images and declines for stars that appear on more images. If the signal-to-noise ratio is large, $SNR \geq 4$, the probability of correct detection is $(95 \pm 5)\%$ if the stars appears on 20 to 30 images and decreases to $(72 \pm 2)\%$ if the star appears on 90 to 100 images. The same trend is visible for data sets with $SNR \leq 3$. The detection rate decreases from $(70 \pm 5)\%$ to $(46 \pm 2)\%$. Why this trend exists we do not know.

If the star appears on zero to ten images the probability of correct detection is $(30 \pm 5)\%$ independently of the signal-to-noise ratio.

Remember that our definition of correctness is based on comparing the algorithms position estimates to the real position values. In an application scenario the real values are obviously unknown. Our starFinder algorithm returns the value M defined as:

$$M = max\left\{X_f\left(\phi_m, \theta_m\right) - g\left(X_f\left(\phi_m, \theta_m\right)\right)\right\} \tag{21}$$

where $g\colon \mathbb{R}^2 \to \mathbb{R}^2$ is the two-dimensional Gaussian filter with $\sigma = 3\,$pixel.

We propose this value as a measure for the likelihood that the algorithms estimates for a stars position and velocity are correct.

In Figure 13 we show the probability density of the correct and incorrect identified stars as functions of M. We see that the distributions are very different. The distribution of M of incorrect identified stars peaks at about 0.6 and has a full width at half maximum of about 0.3. The distribution is much broader for the correct identified stars. The full width at half maximum is about 2. It peaks at $M \approx 2$. The distribution of the correct identified stars has a small secondary peak at $M \approx 0.6$ where the incorrect estimates have their peak too. If we look closely at Figure 2 and 3 you will see some green markers that are unusually far off of the

diagonal. Those may be considered correct, because the estimates uncertainty the algorithm returns may be exceptionally large. This itself might be an indicator to reject the algorithms estimate. But as these are correct by our definition and have estimates that differ from the real values they may have small values of M and thus contribute to the secondary peak.

If we take a closer look at Figure 4 and 5 we see several wrong velocity estimates. We suspect that the secondary peak of the correct estimates in Figure 13 at the position of the peak of the incorrect estimates, would decrease if our definition of correctness would include the matching of velocities.

Taking the definition as is and choosing $M = 1$ to separate likely from unlikely estimates yields a sensitivity (true positive rate) of 93 %. The correct rejection rate is 91 %. We have 6 % false positives as 94% of all data sets with $M > 1$ are correct. 89% of all data sets with $M < 1$ are incorrect, so the false negative rate is 11 %.

M can be used as a measure for the likelihood that the algorithms estimates are correct, albeit some correct estimates would be neglected. The sensitivity is high enough and the the false negative rate small enough for the measure to be useful.

4 Tests on real data

4.1 WISE

The Wide Infrared Survey Explorer (WISE) is described in detail in [WEM+10]. It is a satellite launched on 14 December 2009 surveying the sky in four infrared bands called W1 to W4 centered at wavelengths of $3.4\,\mu m$ (W1), $4.6\,\mu m$ (W2), $12\,\mu m$ (W3) and $22\,\mu m$ (W4). The satellite was operational from 14 January 2010 on. Full coverage of the sky took half a year and was completed on 17 July 2010. Observation in fully operational mode was continued until 6 August 2010 when the satellite ran out of cooling agent, necessary to cool down the infrared sensors. Observations continued using the W1 to W3 band until 29 September 2010 and in the W1 and W2 band until 1 February 2011. The sky was scanned from pole to pole, resulting in higher number of exposures at the poles. Median number of exposures is 33 for W1 and W2 band, 24 for W3 band and 16 for W4 band.

4.2 unwise-coadds

In [Lan14] a process is proposed for coadding single exposures to retrieve detection maps from WISE single exposures provided in the WISE level 1b data release. A python script called "unwise-coadds.py" is provided to produce these coadds.

The main reason we use this script is because the WISE single exposures need some preprocessing. First the script uses a look up table to find all exposures close to a requested coordinate. Images with bad quality scores from the WISE image processing-pipeline are neglected. Images within 2000 seconds after annealing of the satellites sensors are neglected. Images with to much of flux from the moon are neglected.

A stack of the remaining usable images can be given out. We modified the script to also produce a comma separated value list of the corresponding dates the images were produced. This date is given in Modified Julian Date (MJD).

4.3 Results

We produced a stack of WISE images for a coordinate of 2.7 RA 6.98 DEC , epoch J2000, for the W1 band. The coordinates were chosen to provide an above median number of exposures. We produced images with a size of 200-by-200 pixels in accordance to out artificial data sets. The unwise-coadds script yielded a stack of 43 images, spread over 180 days.

In Figure 14 we show three of the images from the same stack. The images differ

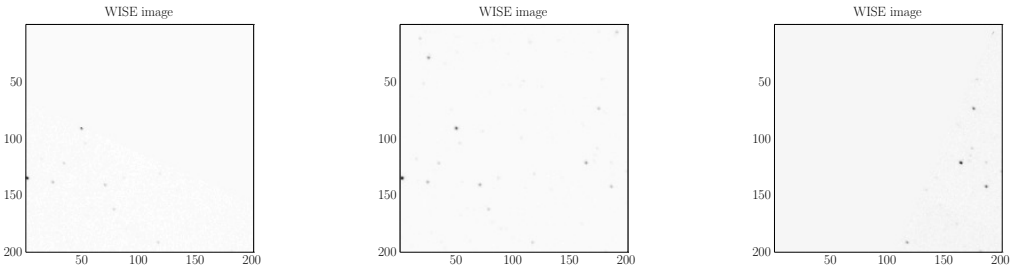

Figure 14: WISE images from the same stack are very heterogeneous. The image to the right is only partially filled.

from each other in that stars that are clearly visible on one of the images do not necessarily show on the others. The image to the right is only partially filled with data. The lower-right third is a cut from a WISE image that can contribute some data to the plotted range, but does not itself contain the the coordinate of 2.7 RA 6.98 DEC that is located in the center of the shown range.

We applied our findStar algorithm to the images. The result is summarized in Table 2.

Value	Estimate	Uncertainty
$\tilde{x}_0$	136.5	0.2
$\tilde{y}_0$	1.8	2.1
$\tilde{v}_x$	$0.0 \cdot 10^{-2}$	$2.3 \cdot 10^{-2}$
$\tilde{v}_y$	$2.3 \cdot 10^{-2}$	$2.3 \cdot 10^{-2}$
M	3.58	

Table 2: Results of applying findStar to the WISE images.

Value	Estimate	Uncertainty
$\tilde{x}_0$	93.0	0.2
$\tilde{y}_0$	50.9	1.7
$\tilde{v}_x$	$0.0 \cdot 10^{-2}$	$2.3 \cdot 10^{-2}$
$\tilde{v}_y$	-0.31	0.02
M	4.87	

Table 3: Results of 2nd run of findStar on the WISE images.

The value of M indicates a high likelihood that this is a correct result. The star is actually visible in the left and middle image of Figure 14 (Notice that the origin of the coordinate system is in the upper-left corner and increasing x is directing

down) . We have not implemented any conversion from a a position on the image to celestial coordinates, but by comparing our results to WISE source catalogs [IRS] we find this source to be J001043.72+065730.9.

We can delete this object from the images by setting all values on the images to zero, where $x \in [125, \ldots, 145]$ and $y \in [1, \ldots, 10]$. Applying the findStar algorithm again yields the results summarized in Table 3.

Again the value of M indicates a high likelihood that the algorithms estimates are meaningful. This star is also visible in the left and middle image of Figure 14. Notice that $\tilde{v}_y$ is distinguishable different from zero with a high level of confidence. By comparing the results to the WISE source catalogue we identify this object as J001045.70+065850.8.

5 Discussion

We have described and implemented a method that is able to find moving objects in low signal-to-noise images without any initial guess. This method works by calculating a discrete analogy to the three dimensional X-ray transform and by peak detection on the resulting parameter space of straight lines through a stack of initial images.

The algorithm calculates estimates for the stars position on the first image and the stars velocity components in both x and y direction. Furthermore M as defined in 21 is calculated. We show that M can be used as a measure for the likelihood that the algorithm found a existing signal instead of a local maximum made up by noise.

Our tests on 6565 artificial data sets have demonstrated our methods capabilities and limitations.

We showed that if a star is identified correctly according to the estimated position, the estimate of the velocity is usually also correct with only a small number of outliers.

From our data sets $(37 \pm 3)\%$ of stars with signal-to-noise ratios as low as $SNR = 2$ were identified correctly. For data sets with $SNR \geq 4$ detection rates of $(77 \pm 3)\%$ could be achieved.

Detection rates are independent from the stars position unless it is in the very vicinity of the images edges.

Detection rates decrease linearly with the objects speed. Objects with a $SNR \leq 3$ have a detection rate of about 20% within our data sets. This SNR is too low to detect the objects in single images while the speed is so high, that the signals do not add up when the images are simply coadded.

We could show that M can actually be used as measure for the estimates likelihood. Future versions of the findStar algorithm should take the value of M into consideration. If no object with a sufficient large M can be found in a certain interval of angles, it does not make sense to continue the search on any smaller nested interval. Instead the search could be repeated with a finer spacing of search parameters over the whole interval of possible values.

We performed a single test of our method on real data from the WISE mission. We rediscovered two known objects. Further systematic testing is necessary and the results need to be compared to existing algorithms for detection of moving objects.

Our test on the real data from the WISE mission included a very simple method of removing a found object from the data to find additional objects. This method

is not implemented in any of our algorithms so far but was carried out manually.

As our method is only searching for objects moving along straight lines, parallax is omitted. Thus objects can not be found in images if their movement due to parallax is more important than the proper motion over the time the images were taken.

Stars with high proper motion and small parallax can be found if the images are taken over the course of a short time, because in that case parallax can be neglected. If the images are taken at the same time of the year every year, the parallax cancells and the proper motion can be measured using our method.

Our methods performance on stars with different proper motions and parallaxes should be measured.

References

[ACDI03] A. Averbuch, R. Coifman, D. L. Donoho, and M. Israeili. Fast Slant Stack: A notion of Radon transform for data in a Cartesian grid which is rapidly computible, algebraically exact, geometrically faithful and invertible. Technical report, Statistics Department, Stanford University, 2003.

[Bas00] G. Basri. Observations of Brown Dwarfs. *ARA&A*, 38:485–519, 2000.

[IRS] NASA/IPAC Infrared Science Archive. `https://irsa.ipac.caltech.edu/Missions/wise.html`.

[Lan14] D. Lang. unWISE: Unblurred Coadds of the WISE Imaging. *AJ*, 147:108, May 2014.

[LHJR09] D. Lang, D. W. Hogg, S. Jester, and H.-W. Rix. Measuring the Undetectable: Proper Motions and Parallaxes of Very Faint Sources. *AJ*, 137:4400–4411, May 2009.

[Mat15] The MathWorks, Inc. Matlab Image Processing Toolbox Documentation. `https://de.mathworks.com/help/images/ref/radon.html`, April 2015.

[WEM+10] E. L. Wright, P. R. M. Eisenhardt, A. K. Mainzer, M. E. Ressler, R. M. Cutri, T. Jarrett, J. D. Kirkpatrick, D. Padgett, R. S. McMillan, M. Skrutskie, S. A. Stanford, M. Cohen, R. G. Walker, J. C. Mather, D. Leisawitz, T. N. Gautier, III, I. McLean, D. Benford, C. J. Lonsdale, A. Blain, B. Mendez, W. R. Irace, V. Duval, F. Liu, D. Royer, I. Heinrichsen, J. Howard, M. Shannon, M. Kendall, A. L. Walsh, M. Larsen, J. G. Cardon, S. Schick, M. Schwalm, M. Abid, B. Fabinsky, L. Naes, and C.-W. Tsai. The Wide-field Infrared Survey Explorer (WISE): Mission Description and Initial On-orbit Performance. *AJ*, 140:1868–1881, December 2010.

List of Figures

List of Tables

Appendices

Derivations

$\tilde{x}_0$ estimate

Solve (10) for x:

$$x_{offset} = \left[\begin{pmatrix} x \\ y \\ z \end{pmatrix} - \begin{pmatrix} x_c \\ y_c \\ z_c \end{pmatrix} \right] \cdot \begin{pmatrix} \cos\theta \\ 0 \\ \sin\theta \end{pmatrix} + b + 1$$

$$x_{offset} = (x - x_c)\cos\theta + (z - z_c)\sin\theta + b + 1$$

$$x_{offset} - b - 1 - (z - z_c)\sin\theta = (x - x_c)\cos\theta$$

$$\frac{x_{offset} - b - 1 - (z - z_c)\sin\theta}{\cos\theta} + x_c = x$$

If $\cos\theta \neq 0$.

$\tilde{y}_0$ estimate

Solve (11) for y:

$$y_{offset} = \left[\begin{pmatrix} x \\ y \\ z \end{pmatrix} - \begin{pmatrix} x_c \\ y_c \\ z_c \end{pmatrix} \right] \cdot \begin{pmatrix} -\sin\phi\sin\theta \\ \cos\phi \\ \sin\phi\cos\theta \end{pmatrix} + b + 1$$

$$y_{offset} = -(x - x_c)\sin\phi\sin\theta + (y - y_c)\cos\phi + (z - z_c)\sin\phi\cos\theta + b + 1$$

$$y_{offset} + (x - x_c)\sin\phi\sin\theta - (z - z_c)\sin\phi\cos\theta - b - 1 = (y - y_c)\cos\phi$$

$$\frac{y_{offset} + (x - x_c)\sin\phi\sin\theta - (z - z_c)\sin\phi\cos\theta - b - 1}{\cos\phi} + y_c = y$$

If $\cos\phi \neq 0$.

singleXray

```
/*
 Copyright (C) 2015 Philipp Schlueter

 This program is free software; you can redistribute it and/or modify
 it under the terms of the GNU General Public License as published by
 the Free Software Foundation; either version 3 of the License, or
 (at your option) any later version.

 This program is distributed in the hope that it will be useful, but
 WITHOUT ANY WARRANTY; without even the implied warranty of
 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
 General Public License for more details.

 You should have received a copy of the GNU General Public License
 along with this program; if not, see <http://www.gnu.org/licenses/>.
*/

#include <octave/oct.h>
#include <math.h>

DEFUN_DLD (singleXray, args, , "\
 -- Function File: XT = Xray(IMAGESTACK, THETA, PHI, MJD)\n\
\n\
     Calculates the 3D-Xray transform of the 3D matrix IMAGESTACK at\n\
     angles given in THETA and PHI. In MJD you can declare the date\n\
     the images in IMAGESTACK were made. THETA and PHI are expected\n\
     to be in units of radians. \n")
{
  int nargin = args.length ();

  if (nargin != 4)
    print_usage ();
  else
    {

    NDArray A = args(0).array_value ();
    NDArray THETA = args(1).array_value();
    NDArray PHI = args(2).array_value();
    NDArray MJD = args(3).array_value();

    dim_vector dA = A.dims();
    dim_vector dTHETA = THETA.dims();
    dim_vector dPHI = PHI.dims();
    float zextent = MJD(dA(2)-1)-MJD(0)+1;
    int b = ceil(sqrt((dA(0)*dA(0)+dA(1)*dA(1) + zextent * zextent ) )
        / 2 + 1);
    float theta = THETA.elem(0);
    float phi = PHI.elem(0);
```

```
Matrix singleout(2*b+1,2*b+1);
Matrix normalizer(2*b+1,2*b+1);
for(unsigned int i = 0; i<2*b+1; i++)
  for(unsigned int j = 0; j<2*b+1; j++){
    singleout(i,j)=NAN;
    normalizer(i,j)=NAN;
  }

float xc = (dA(0)-1.0)/2.0;
float yc = (dA(1)-1.0)/2.0;
float zc = (MJD(dA(2)-1)+MJD(0))/2.0;
float xoffset,yoffset,xfrac,yfrac;
int xk,yk;

for(unsigned int z = 0; z<dA(2); z++)
  for(unsigned int x = 0; x<dA(0); x++)
    for(unsigned int y = 0; y<dA(1); y++){
      //Calulate Offsets
      xoffset = ((x-xc)*cos(theta)+(MJD(z)-zc)*sin(theta)+b+1);
      yoffset = (-sin(phi)*sin(theta)*(x-xc)+cos(phi)*(y-yc)+sin(
          phi)*cos(theta)*(MJD(z)-zc)+b+1);
      xk = floor(xoffset);
      yk = floor(yoffset);
      xfrac = xoffset-xk;
      yfrac = yoffset-yk;

                        //Distribute pixelvalue to four nearest
                           grit cells
      if(singleout(xk,yk) != singleout(xk,yk) ) {
        singleout(xk,yk) = 0;
        normalizer(xk,yk) = 0;
      }
      singleout(xk,yk) += A(x,y,z)*(1-xfrac)*(1-yfrac);
      normalizer(xk,yk) += (1-xfrac)*(1-yfrac);

      if(singleout(xk,yk+1) != singleout(xk,yk+1)) {
        singleout(xk,yk+1) = 0;
        normalizer(xk,yk+1) = 0;
      }
      singleout(xk,yk+1) += A(x,y,z) * (1-xfrac) * yfrac;
      normalizer(xk,yk+1) += (1-xfrac) * yfrac;

      if(singleout(xk+1,yk) != singleout(xk+1,yk) ) {
        singleout(xk+1,yk) = 0;
        normalizer(xk+1,yk) = 0;
      }
      singleout(xk+1,yk) += A(x,y,z)*xfrac*(1-yfrac);
      normalizer(xk+1,yk) += xfrac*(1-yfrac);

      if(singleout(xk+1,yk+1) != singleout(xk+1,yk+1) ) {
      singleout(xk+1,yk+1) = 0;
```

```
            normalizer(xk+1,yk+1) = 0;
          }
          singleout(xk+1,yk+1) += A(x,y,z)*xfrac*yfrac;
          normalizer(xk+1,yk+1) += xfrac*yfrac;
        }

      for(unsigned int i = 0; i<2*b+1; i++)
        for(unsigned int j = 0; j<2*b+1; j++){
          singleout(i,j)/=normalizer(i,j);
        }

    if (! error_state)
      return octave_value (singleout);
    }

  return octave_value_list ();
}
```

Xray

```
## Copyright (C) 2015 Philipp Schlueter
##
## This program is free software; you can redistribute it and/or modify
## it under the terms of the GNU General Public License as published by
## the Free Software Foundation; either version 3 of the License, or
## (at your option) any later version.
##
## This program is distributed in the hope that it will be useful, but
## WITHOUT ANY WARRANTY; without even the implied warranty of
## MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
## General Public License for more details.
##
## You should have received a copy of the GNU General Public License
## along with this program; if not, see <http://www.gnu.org/licenses/>.

## -- Function File: [XT,XP] = Xray(IMAGESTACK, THETA, PHI, MJD)
## -- Function File: [XT,XP] = Xray(IMAGESTACK, THETA, PHI)
## -- Function File: [XT,XP] = Xray(IMAGESTACK, THETA)
## -- Function File: [XT,XP] = Xray(IMAGESTACK)
##
##     Calculates the 3D-Xray transform of the 3D matrix IMAGESTACK at
##     angles given in THETA and PHI.  To each element of THETA and PHI
##     corresponds a matrix in XT.  In MJD you can declare the date the
##     images in IMAGESTACK were made.  The variable XP represents the
##     x-axis and y-axis of the rotated coordinate.  If THETA is not
##     defined, then -90:90 is assumed.  If PHI is not defined, then
##     -90:90 is assumed.  If MJD is not defined, then it is assumed,
##     that all images are separated by one.
```

```
function [XT,xp] = Xray(Imagestack,theta,phi, mjd)

  ## Input checking
  if (nargin == 0 || nargin > 4)
    print_usage ();
  elseif (nargin == 3)
    mjd = NaN;
  elseif (nargin == 2)
    mjd = NaN;
    phi = -90:90;
  elseif (nargin == 1)
    mjd = NaN;
    phi = -90:90;
    theta = -90:90;
  endif

  if (!ismatrix(Imagestack) || ndims(Imagestack) != 3)
    error('First input must be a 3d "Matrix"');
  endif

  [xlength,ylength,zlength]=size(Imagestack);
  if (isnan(mjd))
    mjd = 1:zlength;
  end

  if not(length(mjd)==zlength)
    error('Size of mjd vector does not equal number of images.');
  end

  ## Output allocation
  b = ceil(sqrt(sum(xlength^2+ylength^2+(max(mjd)-min(mjd)+1)^2))/2+1);
  xp = [-b:b]';

  RT = zeros(2*b+1,2*b+1,length(theta),length(phi));

  th = theta'*pi/180;
  ph = phi'*pi/180;
  pkg load ndpar;

  for p = 1: length(phi)
    XT(:,:,:,p) = ndpar_arrayfun(nproc,@singleXray,Imagestack,th,ph(p),
        mjd,"IdxDimensions",[0,1,0,0],"CatDimensions", [3]);
  endfor
endfunction
```

findStar

```
function res = findStar(Imagestack,MJD = -99999)
  [~,~,zlength]=size(Imagestack);
  if (MJD ==  -99999)
```

```
          MJD = 1:zlength;
  endif

  if not(length(MJD)==zlength)
    error('Size of MJD vector does not equal number of images.');
  endif

MJD = MJD-mean(MJD);

  pkg load image;
  pkg load statistics;
  phi = linspace(-89.99,89.99,11);
  theta = linspace(-89.99,89.99,11);
  xOffset = 0;
  yOffset = 0;
  xLowerLimit = 1;
  yLowerLimit = 1;

  minmax = @(A) [min(A(:)) max(A(:))];

  for i = 1:15
    xOffset = xOffset+xLowerLimit-1;
    yOffset = yOffset+yLowerLimit-1;
    [xlength,ylength,~]=size(Imagestack);
    xc = (xlength+1)/2;
    yc = (ylength+1)/2;
    zc = (min(MJD)+max(MJD))/2;
    zlength = max(MJD)-min(MJD)+1;

    RT = Xray(Imagestack,theta,phi,MJD);
    [RTxlength,~,~,~] = size(RT);
    b = (RTxlength-1)/2;

    RTc = convn(convn(RT,[0.0001,1,0.0001],'same'),[0.0001;1;0.0001],'
        same');
    RTc = convn(convn(RTc,normpdf(1:9,5,3)',''same'),normpdf(1:9,5,3),'
        same');
    [m,im]=max(RTc(:));
    [i1,i2,i3,i4]=ind2sub(size(RTc),im);

    th = [theta(max(i3-2,1)) theta(min(i3+2,length(theta)))]*pi/180;
    ph = [phi(max(i4-2,1)) phi(min(i4+2,length(phi)))]*pi/180;
    vx = minmax(-tan(th));
    vy = minmax(-tan(ph)'*(1./cos(th)));
    x0 = minmax((i1-(min(MJD)-zc)*sin(th)-b-1)'*(1./cos(th))+xc-1);
    [X,Y,Z] = meshgrid(1:2,1:2,1:2);
    y0 = minmax((i2+sin(ph(X(:))).*sin(th(Y(:))).*(x0(Z(:))-xc)-b-1-sin
        (ph(X(:))).*cos(th(Y(:)))*(min(MJD)-zc))./cos(ph(X(:)))+yc-1);

    xLowerLimit = max(1,min(floor([x0(1),x0(1)+vx*zlength]-10)));
    xUpperLimit = min(xlength,max(ceil([x0(2),x0(2)+vx*zlength]+10)));
```

```
    yLowerLimit = max(1,min(floor([y0(1),y0(1)+vy*zlength]-10)));
    yUpperLimit = min(ylength,max(ceil([y0(2),y0(2)+vy*zlength]+10)));

    Imagestack = Imagestack(xLowerLimit:xUpperLimit,yLowerLimit:
       yUpperLimit,:);
    theta = linspace(theta(max(i3-2,1)),theta(min(i3+2,length(theta)))
       ,7);
    phi = linspace(phi(max(i4-2,1)),phi(min(i4+2,length(phi))),7);
  endfor

  M = nanmax((RT(:,:,i3,i4)-RTc(:,:,i3,i4))(:));
  M2 = nanmax((-(RT(:,:,i3,i4)-RTc(:,:,i3,i4))(:)));
  res = [ sum(x0)/2+xOffset std(x0);
          sum(y0)/2+yOffset std(y0);
          sum(vx)/2 std(vx);
          sum(vy)/2 std(vy);
          M M2];
endfunction
```