OMNIDIRECTIONAL IMAGING

David Mabius Jonathan Tang

Dec. 15, 2007

Boston University

Department of Electrical and Computer Engineering

Technical report No. ECE-2007-07

BOSTON
UNIVERSITY

OMNIDIRECTIONAL IMAGING

David Mabius Jonathan Tang

Boston University
Department of Electrical and Computer Engineering
8 Saint Mary's Street
Boston, MA 02215
www.bu.edu/ece

Dec. 15, 2007

Technical report No. ECE-2007-07

Summary

This project was completed as a part of the course EC 720 entitled “Digital Image Processing and Communications”. Minimizing the number of network cameras while maximizing the area of coverage involves maximizing the field of view for each sensor. Many types of systems exist that capture a much larger field of view compared to conventional cameras. In this report a system consisting of a camera and a curved mirror is investigated. Such a system is called a catadioptric imaging system. The images produced by this system contain information from a hemispherical region surrounding the mirror. This type of image is referred to as an omnidirectional image. Omnidirectional images are not easily interpreted and are generally undesirable for human visualization. The challenge is then to convert the omnidirectional image to a more familiar form with minimal loss of data. The particular interest is in using omnidirectional imaging to monitor wildlife along a shoreline, and the particular challenge is to unwrap the omnidirectional image into a panoramic image while minimizing distortion.

Contents

List of figures

1.Background

Minimizing the number of sensors in a camera network while maximizing the area of coverage involves maximizing the field of view for each sensor. Many types of systems exist that capture a much larger field of view compared to conventional cameras. In this report a system consisting of a camera and a curved mirror is investigated. Such a system is called a catadioptric imaging system. The images produced by this system contain information from a hemispherical region surrounding the mirror. This type of image is referred to as an omnidirectional image. Omnidirectional images are not easily interpreted and are generally undesirable for human visualization. The challenge is then to convert the omnidirectional image to a more familiar form with minimal loss of data.

Professor Little is interested in using a catadioptric imaging system in order to monitor wildlife along a shoreline. His system consists of a curved mirror suspended above a CCD camera pointed at the apex of the mirror. The resulting omnidirectional image contains data from a 360 degree field of view with heavy distortions. The goal of this investigation is to unwrap the image into a panoramic image while minimizing distortion.

2. Literature Review

Fixed Viewpoint Constraint

Baker and Nayar [1] discuss a method of choosing mirror shapes that will result in catadioptric images. In this particular report they choose mirrors that are constrained by a fixed viewpoint. The fixed viewpoint constraint requires that all rays captured by the imaging system would have passed through a single point in 3D space, *the effective viewpoint*, had it not been reflected by the mirror. A

common simplification is to force all points that are captured by the imaging system to pass through a single point which is called *the effective pinhole*. The model of our imaging system can be seen in Illustration 1. There are a number of different types of mirrors that satisfy the fixed viewpoint constraint including the planar, ellipsoidal, spherical, conical, and hyperboloidal mirrors. The contents of this report will focus on the hyperboloidal mirror.

Systems with multiple viewpoints

Swaminathan et. al. [2] explore the effects of using mirrors that do not have a fixed viewpoint. In such systems all points that are captured in the image plane would have passed through a point located on the locus of viewpoints called a

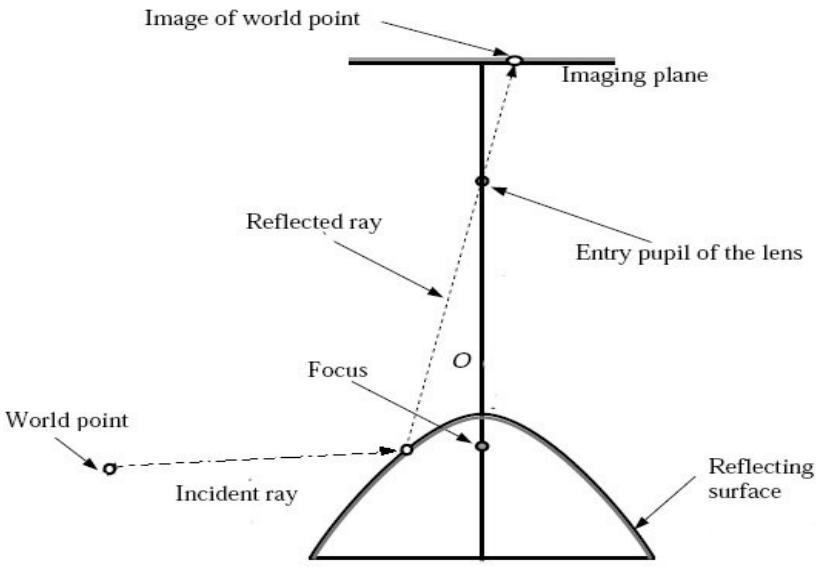

Illustration 1: Omnidirectional Imaging System Model taken from [2]

caustic had it not been reflected by the mirror. Systems with multiple viewpoints can be created by using a mirror not of the shapes listed in the previous section

or by slightly displacing the mirror from it's fixed viewpoint configuration, i.e. by moving the camera off of the 2nd focal point in a hyperboloidal case. This kind of setup behavior can be seen in Illustration 2. Each incident ray that is reflected through the pinhole would have intersected the caustic in a tangential manner if reflection did not take place. There are considerable differences between multiple viewpoint mirrors and single viewpoints; one such difference is resolution.

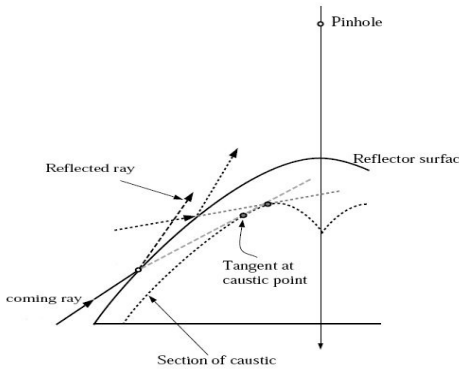

Illustration 2: Caustic from [2]

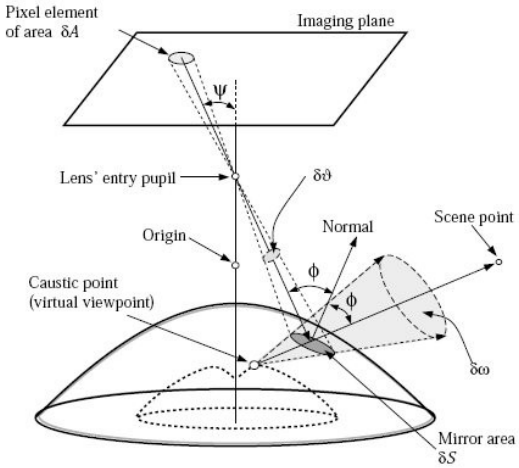

Illustration 3: Image Resolution from [2]

Illustration 3 above shows how a pixel element of area δA captures a world region of area δw. Resolution is defined as δA/ δw. A property of caustics in hyperboloidal mirrors is that they approach the the surface of the mirror as you travel radially away from the apex. Consequently, the region δw becomes larger as you travel radially away from the mirror while δA remains the same. Therefore resolution decreases as distance from the apex increases. The opposite is true for fixed viewpoint images. For a hyperboloidal mirror with a fixed viewpoint, rays that were reflected from near the apex of the mirror will have very low resolution because the focal point is so close to the mirror surface. However, the distance between viewpoint and mirror surface increases as you travel radially away from the apex. Therefore for the same δS (mirror area) as in the multiple viewpoint case, the corresponding δw will be smaller.

Thus, resolution increases as distance from the apex increases; a consequence that leads to the formation of the mirror used in this report. This is shown in Illustration 4 and Illustration 5 below.

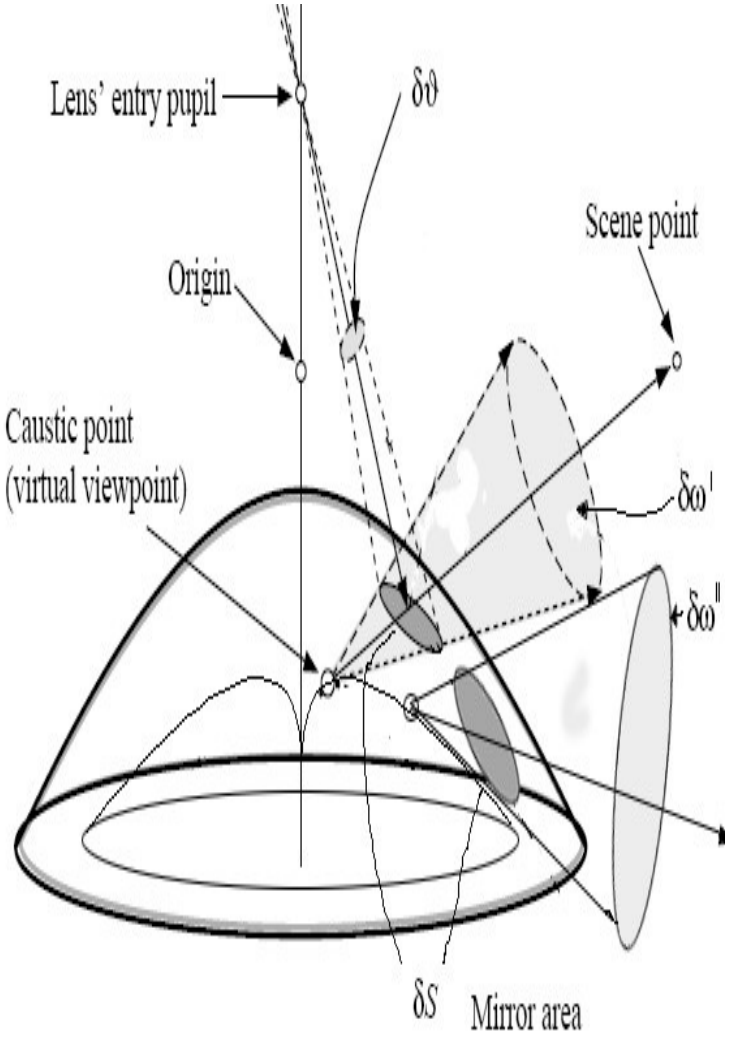

Illustration 4: Caustic

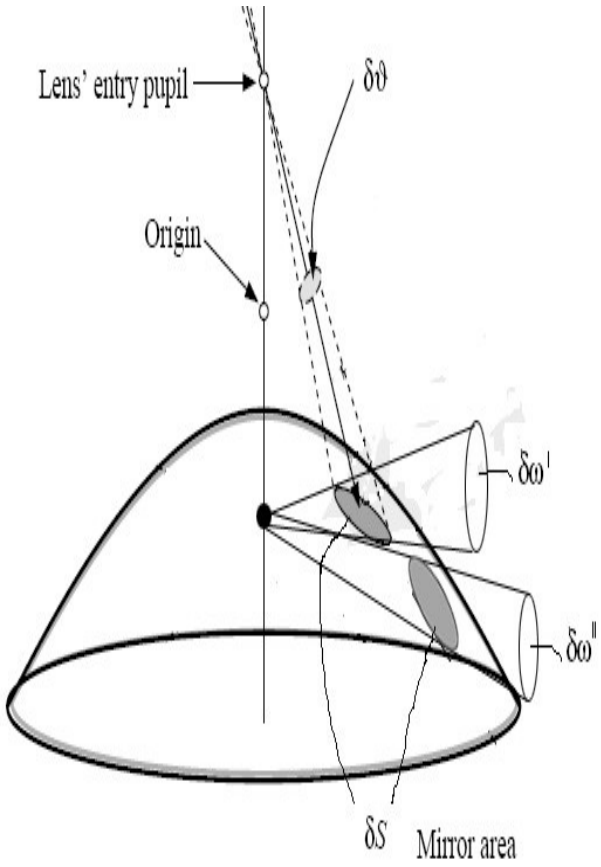

Illustration 5: Fixed Viewpoint

Mirror and Image Sensor Design

Decco et. al. [3] have created a partitioned mirror consisting of two hyperboloids. The inner region was designed to have constant horizontal resolution. Constant horizontal resolution signifies that equidistant points on the floor of Illustration 6 will be equidistant in the image plane as well. The outer region was designed to have constant vertical resolution at a fixed distance away from the mirror; 2m in this case. Decco et. al. [3] also introduce the idea of using a log polar image sensor. The log polar sensor in Illustration 7 would resolve the resolution issues

associated with unwrapping an omnidirectional image by having a denser pixel concentration near the center and gradually decreasing pixel concentration as you travel radially outward. Unwrapping omnidirectional images captured with this device would be trivial as these sensors typically allow the user to read pixels serially into panoramic images. Unfortunately these sensors are more expensive and less available than traditional image sensors which have cartesian pixel arrangements. The image sensor used in this report was a traditional cartesian image sensor and consequently some image processing must be performed to unwrap the omnidirectional image.

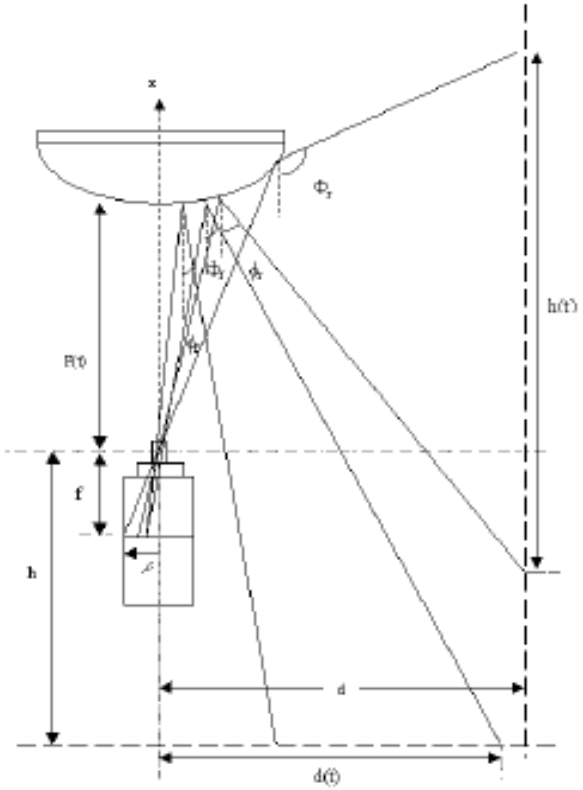

Illustration 6: Constant Resolution

Illustration 7: Log Polar Image Sensor

3. Problem Statement

To capture the omnidirectional images we used a setup similar to the one shown in Illustration 8 below, however, in our setup the camera was suspended from the tripod facing downward and the mirror was on the smaller tripod facing upward. This setup led to the image shown in Illustration 9 shown below.

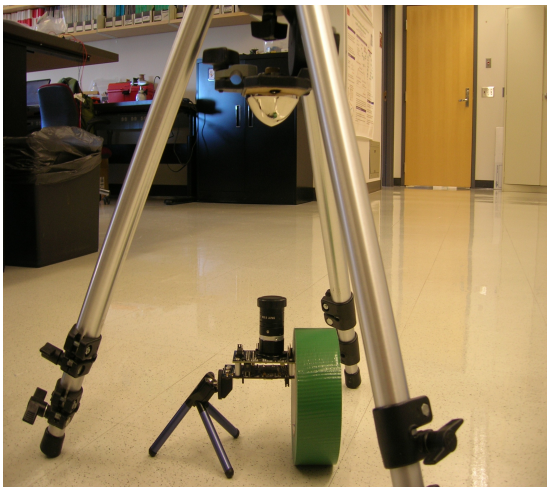

Illustration 8: Test Setup

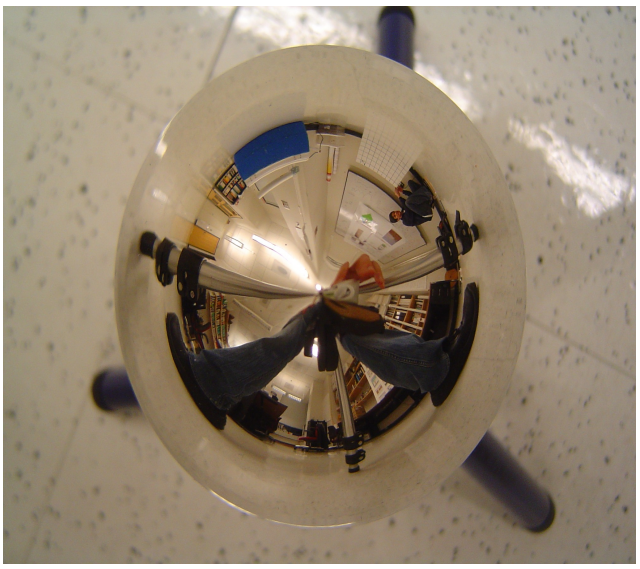

Illustration 9: Omnidirectional Image

There are two types of distortions associated with the omnidirectional image. The first type of distortion exists rotationally. Straight lines such as the junction of the floor and wall visible in Illustration 8 become circles in the omnidirectional image. The second type of distortion exists axially and causes scaling errors along every diameter of the omnidirectional image. It is difficult to see this type of distortion in the wrapped image but the effects still exist. Transforming the omnidirectional image to a panoramic image will require the resolution of both types of distortions.

Proposed Solution to Rotational Distortion

The mirror used in this setup is rotationally symmetric meaning any side-view of the mirror yields the same profile. This rotational symmetry ensures that uniform angular samples in a polar basis will produce uniformly spaced horizontal samples for the unwrapped panoramic image. However, because the sensor is a conventional CCD, the original sample space is not polar but Cartesian. In order to overcome this sampling mismatch, interpolation will be required. Given the output resolution of the panoramic, a polar sampling lattice can be defined. Points on this lattice will not coincide with points on the original imaging sampling lattice, but can be calculated via a form of two dimensional interpolation. Two common types of 2-D interpolation are nearest neighbor and bilinear

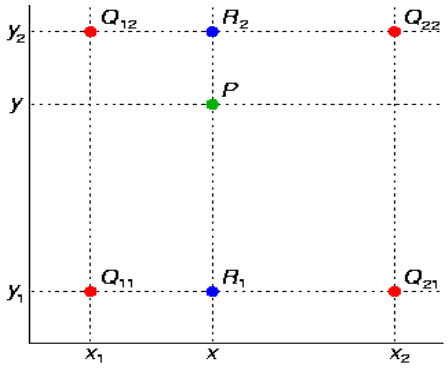

Illustration 10: Interpolation

interpolation.

Nearest neighbor interpolation simply replaces an unknown sample point with the value of the nearest existing sample point. In the case of Illustration 10, the point P would be given the value $f(Q_{12})$. This is computationally trivial and can be quickly implemented but produces jagged steps within the interpolated image.

Bilinear interpolation takes a weighted average of the four nearest points on a regular grid to estimate the unknown value. Looking at Illustration 10, the value at unknown point P can be defined by the function,

$$f(P) \approx \frac{(y_2 - y)}{(y_2 - y_1)} f(R_1) + \frac{(y - y_1)}{(y_2 - y_1)} f(R_2)$$

Equation 1:

where,

$$f(R_1) \approx \frac{(x_2 - x)}{(x_2 - x_1)} f(Q_{11}) + \frac{(x - x_1)}{(x_2 - x_1)} f(Q_{21})$$

Equation 2:

and

$$f(R_2) \approx \frac{(x_2 - x)}{(x_2 - x_1)} f(Q_{12}) + \frac{(x - x_1)}{(x_2 - x_1)} f(Q_{22})$$

Equation 3:

Proposed Solution to Axial Distortion

Mapping

The axial distortions cause spacing errors along diameters of the omnidirectional image. A series of mappings must be developed to correct these axial distortions. The first mapping links pixels in the image plane to points on the

mirror. In Illustration 11 is a cross section of the mirror suspended above the image plane.

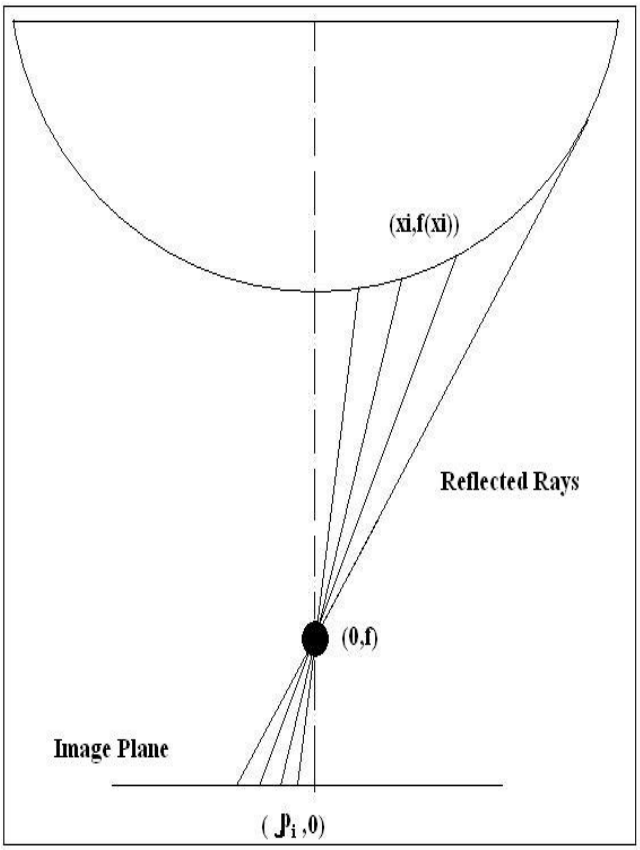

Illustration 11: Image Plane to Mirror Points Mapping

Since each reflected ray must pass through the point (0,f) the slope of each reflected ray terminating in a pixel of this particular omnidirectional image diameter will be $f/|\rho_i|$. Knowing a point on a line along with the slope allows the first mapping equation to be written:

$$y = \frac{f}{\rho i} \cdot z + f$$

Equation 4:

Setting y = f(x) and noting that f(x) includes the height of the mirror from the ground allows:

$$f(xi)=\frac{f}{\rho i}\cdot xi+f,i\in[1,2,\ldots,n]$$

Equation 5:

where n is the number of pixels along the diameter of the omnidirectional image. Also note that xi and consequently f(xi) will take discrete values because ρ_i is discrete even though f(x) may be a continuous function. Next the mapping from reflected ray to incident ray must be developed. Illustration 12 shows a blown up diagram of the mirror and some properties of the mirror for a given mirror point. If f(x) is known (or approximated) for each mirror point there exists a mirror slope and a mirror normal. The slope of the mirror at each mirror point can be given by the derivative, f'(x). In Illustration 12 the angle between the slope of the mirror at a particular point (xi) and a global horizontal line is given as $\Phi=\tan^{-1}(f'(xi))$.

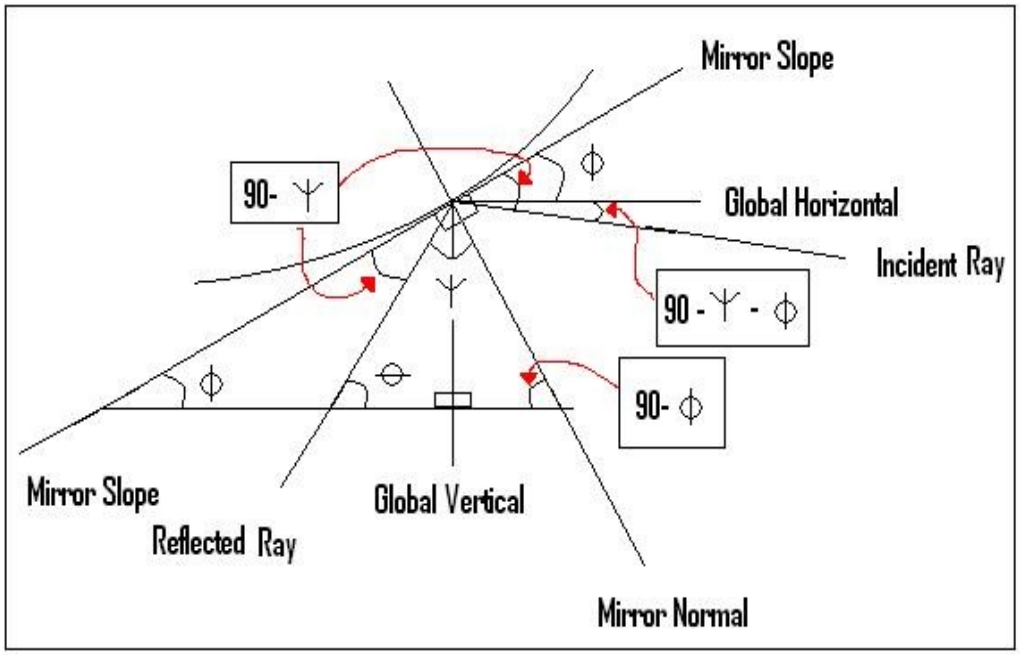

Illustration 12: Reflected Ray to Incident Ray Mapping

Since the angle between the mirror normal and the mirror slope is 90 and the angle, the angle between the mirror normal and a negative global horizontal is 90 – Φ. Using Equation 4 the angle between the reflected ray slope and a global horizontal is $\theta=\tan^{-1}(f'(xi))$. Using the property of triangles that all internal angles must add up to 180, the angle between the reflected tray slope and the mirror normal is designated as psi = 90 – phi + theta. Since the angle between the mirror slope and normal is 90, the angle between the mirror slope and reflected ray is 90 – psi. The Law of Reflection states that the angle between an incident ray of light and a mirror normal will be equal to the angle between the reflected ray of light and the normal. From this law falls the relationship that the angle between the mirror slope and reflected ray must be equal to the angle between the mirror slope and incident ray. This equality leads to the next mapping equation:

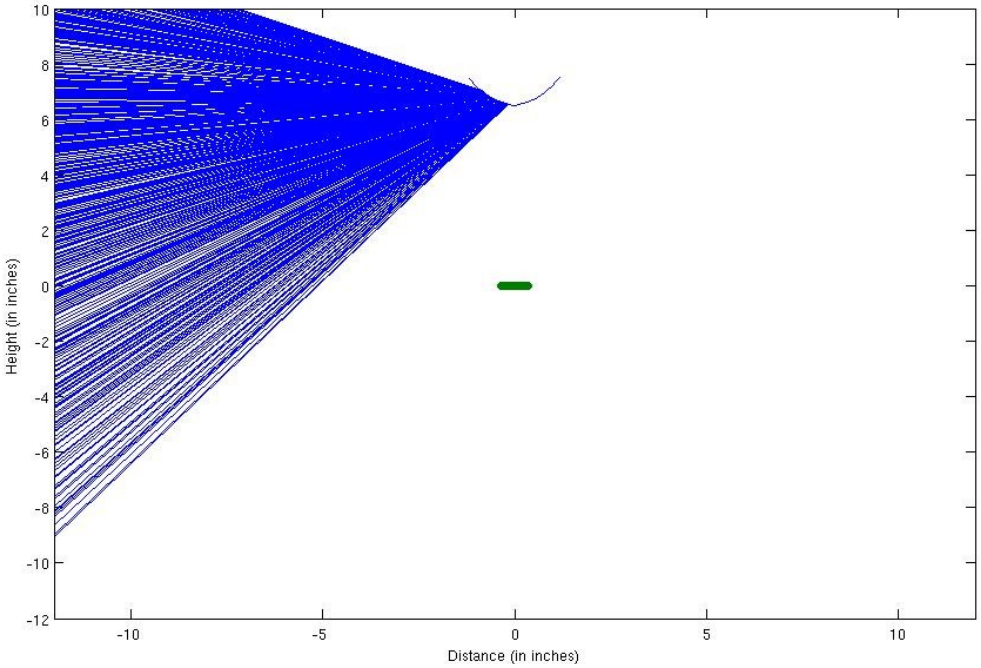

Illustration 13: Ray Mapping Diagram

$$angle\ between\ Global\ Horizontal,\ Incident\ Ray = 90 - \psi - \Phi = \theta - 2\Phi \quad where\ \psi = 90 - \Phi + \theta,\ \Phi = \arctan(f'(xi)),\ \theta = \arctan(f / \rho i)$$

Equation 6:

The final mapping links mirror points to real world points and thus links the image plane to the real world. In we see how an image plane point (ρ_i, 0) maps to mirror point (xi, f(xi)) and finally to an incident ray of slope θ - 2Φ. The incident ray must terminate somewhere in the world. To simplify computation we will assume that our world consists of a cylinder a distance d from the center of the image plane. Therefore, the mirror point (xi, f(xi)) maps to world point (0,hi). The final mapping equation is given by:

$$hi = d\tan(\theta - 2\Phi) + f(xi)$$

Equation 7:

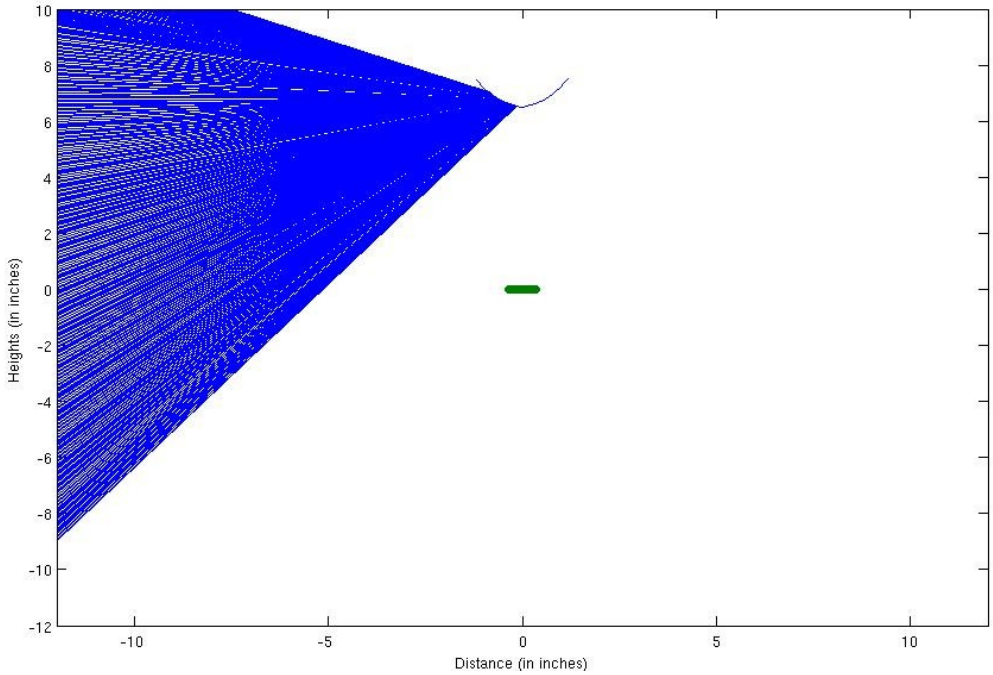

Illustration 14: Equi-spaced Ray Mapping

Correction

Now that the mapping equations have been developed the axial distortions can be corrected. The implementation is explained in detail Section 4. The general method is to:

1. Layout equidistant points on the wall that is horizontal distance d from the origin
2. Find where the equidistant points fall on the image plane
3. Find which actual image plane points are nearest to the equidistant points
4. Shift the actual image plane points to the proper equidistant locations
5. Interpolate

4. Implementation

Rotational Distortions

Overcoming the rotational distortions present in the omnidirectional image involves unwrapping it from a polar space to a Cartesian one as described in the previous section. A panoramic image can be considered as points on an imaginary cylinder at some distance away from the effective viewpoint (see Illustration 15). Given the omnidirectional image produced by our setup, unwrapping the image involves mapping points from the image to points on the imaginary cylinder. As mentioned interpolation is necessary to obtain values at the desired sample coordinates.

Using MATLAB, we made two scripts to effectively unwrap the image to a

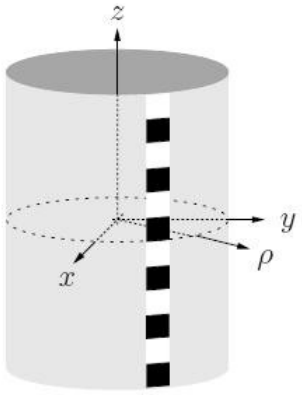

Illustration 15: Wrapped Panoramic

panoramic. Both operate by taking a fixed output resolution for the desired panoramic and find the corresponding coordinates in a polar space in the omnidirectional image. These points are then sampled via interpolation. One script uses the nearest neighbor interpolation (see Appendix A), while the other utilizes bilinear interpolation (see Appendix B). Both utilize the rotational symmetry of the mirror by sampling at equal angular intervals, but axial symmetry is ignored by sampling at equal radial intervals as well.

Axial Distortions

As explained in the previous section, the first task is to layout equidistant points on the wall. We chose the wall to be 1 foot away from the mirror because it was this test image that seemed to be most affected by this type of distortion. We found how many rays were captured between heights 10 inches and -9 inches in Illustration 13. Note that even before axial distortion correction the rays are not significantly skewed and appear approximately equi-spaced. This is the result of designing the mirror with constant vertical resolution. However, due to errors during manufacting and the relatively rough test setup, spacing errors do exist.

Illustration 14 shows rays equi-spaced terminating at the wall 1 foot away. The most noticeable differences between Illustration 13 and Illustration 14 is the region close to -9 inches. In Illustration 13 the rays become less dense in this area while in Illustration 14 the ray density remains constant. Now that we know where each equidistant point is, we adjust each ray in the original mapping to match it's corresponding equidistant ray. This leads to pixel shifts in the image plane since each incident ray is anchored to a unique reflected ray. Since we are dealing with discretely quantized spaces, shifting pixels to new locations leaves behind empty holes. To view the unwrapped image without these holes, we used linear interpolation.

5. Experimental Results

Two test images were acquired and unwrapped, one with an object of interest placed 1 foot away as shown in Illustration 16 and another image with an object of interest placed 6 feet away as shown in Illustration 17.

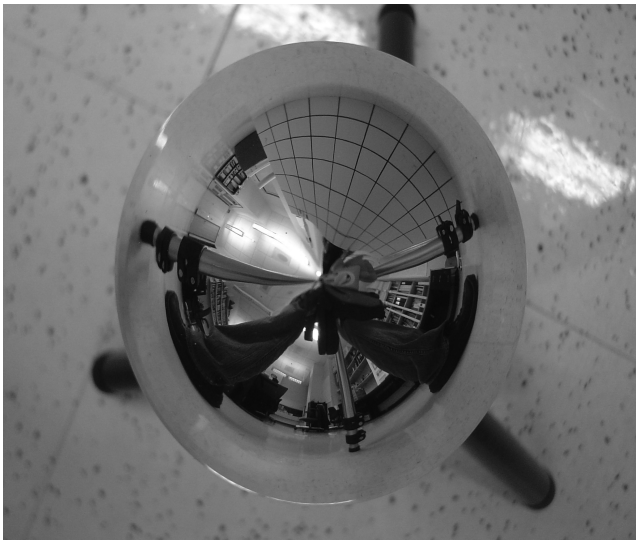

Illustration 16: 1-Foot Omnidirectional Image

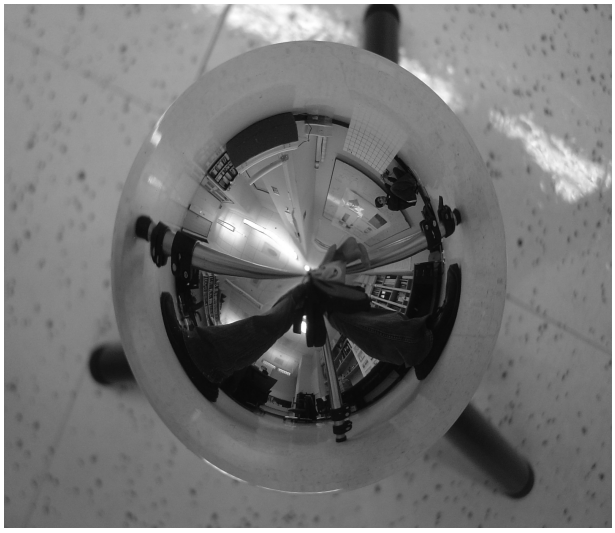

Illustration 17: 6-Foot Omnidirectional Image

Interpolation Method

Two types of panoramic images were generated, one that utilized nearest neighbor interpolation, the other utilized bilinear interpolation. Clearly images generated using bilinear interpolation produced more desirable results. Looking at the nearest neighbor implementation (see Illustration 18), jaggedness can be seen in high contrast diagonal edges, such as the tripod legs and the vertical edge of the white board. The bilinear implementation, on the other hand (see Illustration 19), produced much smoother edges. Given the power of MATLAB, the additional computational cost is trivial.

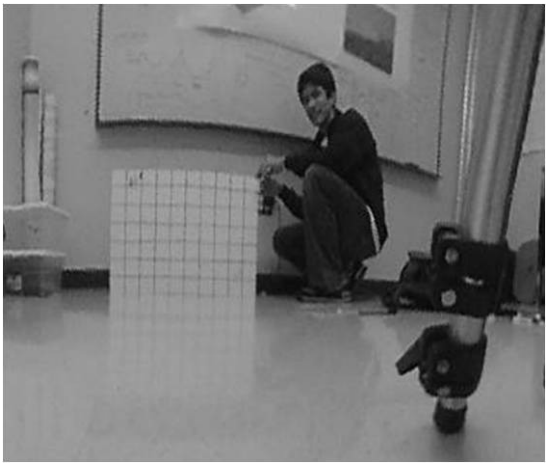

Illustration 18: Nearest Neighbor Interpolation

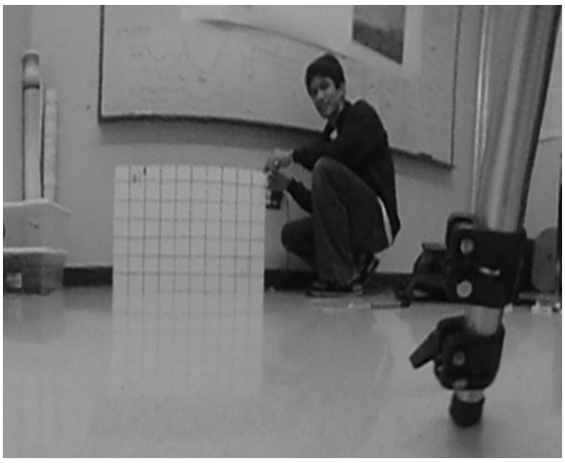

Illustration 19: Bilinear Interpolation

Axial Distortion and Object Distance

Two types of images were focused on, one that contained objects of interest close to the sensor, and one that contained objects of interest farther from the sensor. One interesting result is that objects near the top and bottom of the panoramic became heavily distorted. This is very noticeable in the image where the test grid (our object of interest) was at a distance of about 1 foot from the

sensor (see Illustration 20). The panoramic where our test grid is about six feet from the sensor, however (see Illustration 21), is very reasonable. This suggests that the resolution is fairly uniform in this middle region.

Illustration 20: 1-Foot Panoramic Test Image

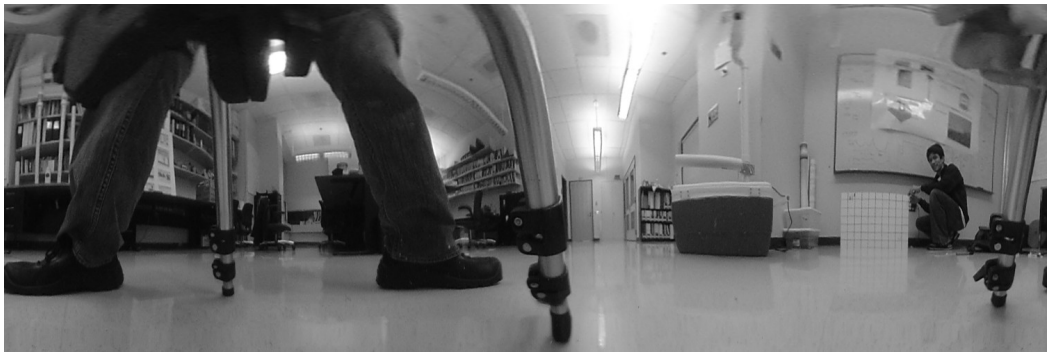

Illustration 21: 6-Foot Panoramic Test Image

Producing a panoramic image from an omnidirectional image by resolving the rotational and axial distortions is a useful approach for transforming an otherwise unfamiliar image into a familiar form. While we did not fully and completely resolve the axial distortions produced by our catadioptric system, we have

demonstrated a well defined mirror profile can produce a relationship capable of resolving such distortions. Given the capabilities of MATLAB this approach is much simpler than designing a custom sensor for a given mirror arrangement.

6. Conclusions

Omnidirectional image systems consisting of a mirror and conventional camera prove to be a useful technique for broadening the field of view of the conventional camera. The obtained results will be adequate for monitoring wildlife at relatively far distances from the camera if the camera is of sufficiently high quality (5 megapixels or more).

The next implementation of our solution would involve resolving axial and rotational distortions in one step and thus eliminating one stage of interpolation. This method is more difficult to physically realize but may save computation time and improve image quality. Furthermore, a more precise setup should be used to ensure that the mirror is centered with respect to the camera and that the spacing is all within the specifications of the specific mirror chosen.

Other applications of omnidirectional imaging systems include robotic vision and 3D scene restoration. These popular fields use systems similar to the one found in this report to steer a robot around a scene, and have it navigate successfully to some endpoint. Using more than one omnidirectional imaging system in a given implementation also leads to stereoscopic imaging systems [4]. These types of systems can reconstruct 3D scenes with accurate sizing of objects based on the processing of two images of the same scene. They are typically used in industrial applications where robots navigate warehouses.

Another successful application of omnidirectional imaging systems was created at Columbia University and is called OMNICAMERA [4]. OMNICAMERA is a

system used for video conferencing. The omndirectional imaging system is placed on the table and members of the conference appear in small perspective windows after some video processing. These perspective windows are possible by using systems with a fixed viewpoint. The advantage of this system over using a single camera without a mirror and attempting to capture all parties present is that each individual will take up an entire window of view. It will be easier to capture subtle body language cues which comprise a significant portion of human interaction.

Appendix A

```
% First Implementation of the Unwrapping Transformation
% Utilizes Nearest Neighbor Interpolation
% Author: Jonathan Tang

clear all;

test1=imread('6_feet.JPG');
test1=rgb2gray(test1);
test1=mat2gray(test1);
figure(1)
imshow(test1, [min(min(test1)) max(max(test1))]);
title('Original');
truesize;

% parameters
RADOUT=700;
RADIN=1;

% using 0 error assumes that the center of the input image is the center of
% the mirror
yerr = 0;
xerr=0;
CENT = [round(size(test1,1)/2)+yerr round(size(test1,2)/2)+xerr];
ANGRES=deg2rad(0.25);

% create polar parameters and relate to cartesian coordinates
angle = 0:ANGRES:2*pi-ANGRES;
[theta,hyp]=meshgrid(angle,RADOUT:-1:RADIN);

% these are flipped from the conventional because the angle is taken with
% respect to -90 degrees not zero.
```

```
y = hyp.*cos(theta) + CENT(1);
x = hyp.*sin(theta) + CENT(2);

% we will use x and y as matrix indices so they must be integers
% This is necessary to eliminate a rotation that occurs otherwise.
% yint = round(y);
% xint = round(x);

% yintRepeats = round(y);
% xintRepeats = round(x);
%
%
% xycoords=zeros(length(xintRepeats(:)),2);
% xycoords(:,1)=xintRepeats(:);
% xycoords(:,2)=yintRepeats(:);
% %works
% unique_coords = unique(xycoords,'rows');
% unique_coords_index=size(unique_coords,1);
% for i=1:size(unique_coords,1)
%     unique_coords_index(i)=find(xycoords(:,1)==unique_coords(i,1) &
xycoords(:,2)==unique_coords(i,2),1,'first');
% %     test=[Unique(1,i);Unique(2,i)];
% %     rep=find(Unique(1,:)==test(1,1) & Unique(2,:)==test(2,1),1,'first');
% end
%
% xycoords_up=zeros(size(xycoords));
% xycoords_up(unique_coords_index,:)=xycoords(unique_coords_index,:);
%
%
%
%
% yint=zeros(size(yintRepeats));
% xint=zeros(size(xintRepeats));
%
```

```
%
% xint(:)=xycoords_up(:,1);
% yint(:)=xycoords_up(:,2);
yint=round(y);
xint=round(x);
% this converts x and y coordinates of a matrix to the element location
% within the matrix.
eleloc = (xint-1)*size(test1,1) + yint;

% places zeros in areas outside of the image that are inside of the circle
eleloc(~(yint>=1 & yint<=size(test1,1) & xint>=1 & xint<=size(test1,2)))=0;
nz=find(eleloc);
% creates unwrapped matrix of zeros.  The zeros are important because they
% are place holders for the unwrapped image in the areas that are inside of
% the circle but outside of the image

unwrapped = zeros(RADOUT-RADIN+1,length(angle));
unwrapped(nz) = test1(eleloc(nz));
%unwrapped = test1(eleloc);
% figure(2)
% imshow(xint, [min(min(xint)) max(max(xint))]);
% truesize;
%
% figure(3)
% imshow(yint, [min(min(yint)) max(max(yint))]);
% truesize;

unwrapped_upside_down=imrotate(unwrapped,180);
% figure(4)
% imshow(unwrapped, [min(min(unwrapped)) max(max(unwrapped))]);
% truesize;
figure(4)
imshow(unwrapped_upside_down, [min(min(unwrapped_upside_down))
max(max(unwrapped_upside_down))]);
```

```
truesize;


% Resolve Axial Distortions
% Author: Jonathan Tang

% assume f=2 inch
f=2;
% assume height=4.5 inches
height=4.5;
% place the mirror above the ground at a specified distance
mirr_in_world=y+f+height+1;
mirr_in_world=interp(mirr_in_world,10);
x=-(2.36/2-2.36/length(mirr_in_world)):2.36/length(mirr_in_world):2.36/2;
vres=1000;
ccdlength=7*vres*10^(-4);
pixelen=ccdlength/vres;
p=-(ccdlength/2-pixelen):pixelen:ccdlength/2;
implane=zeros(1,length(p));

%the_mirr_ray_slope is the slope of the line connecting a point on the mirror
%to the focus
% the_pixel_slope is the slope of the line connecting a point on the image
% plane to the focus
%the_mirr_ray_slope scans from left to right, the _pixel_slope scans from right
%to left
%pixel_2_mirr is an index to be used with the_mirr_ray_slope such that
%the_mirr_ray_slope(pixel_2_mirr(i1))=the_pixel_slope(i1)

% also for i=1:length(p) we have
%x(pixel_2_mirr(i)) is the mirror point horizontal distance
%mirr_in_world(pixel_2_mirr(i)) is the mirror point height
% the _mirror_slope (pixel_2_mirr(i)) is the mirror slope at the mirror
% point
```

```
% the_pixel_slope(i) is the slope of the point on the image plane
% p(i) is the point on the image plane

% furthest points in pixel plane map to 131 and 911 in mirror plane
rayx=linspace(0,10,1040);
tpt=1000;
figure(1)
plot(x,mirr_in_world,'-',p,implane,'o')
line([x(tpt) -12],[mirr_in_world(tpt) 24]);
%grline=line([x(tpt) 0],[mirr_in_world(tpt) f]);

dist=12;
pmin=47;
pmax=415;



axis([-dist dist -dist 10])
the_mirr_ray_slope=(mirr_in_world-f)./x;
j=2:length(x)-1;
% estimate the slope of the mirror using the ratio of the difference of the
% point before and point after

the_mirr_slope(j)=(mirr_in_world(j+1)-mirr_in_world(j-1))./(x(j+1)-x(j-1));
the_mirr_slope(1)=the_mirr_slope(2);
the_mirr_slope(length(x))=the_mirr_slope(length(x)-1);
% slope of the normal at each point on the mirror
the_mirr_norm_slope=tan(atan(the_mirr_slope)-pi/2);
the_mirr_refl_ray_slope=-tan(pi-2*(atan(the_mirr_norm_slope)-atan(the_mirr_ray_slope))-
atan(the_mirr_ray_slope));
j=length(x)/2+1:length(x);

% slope of the reflected ray at each point on the mirror
% only a subset of the points on the mirror will be used
```

```
the_mirr_refl_ray_slope(j)=tan(-pi/2-atan(the_mirr_norm_slope(j))+atan(the_mirr_ray_slope(j))-
atan(the_mirr_slope(j)));


% sort the refl ray slopes to account for errors
the_mirr_refl_ray_slope(1:length(the_mirr_refl_ray_slope)/2)=sort(the_mirr_refl_ray_slope(1:lengt
h(the_mirr_refl_ray_slope)/2));
the_mirr_refl_ray_slope(length(the_mirr_refl_ray_slope)/2:length(the_mirr_refl_ray_slope))=sort(t
he_mirr_refl_ray_slope(length(the_mirr_refl_ray_slope)/2:length(the_mirr_refl_ray_slope)));
p=-p;
the_pixel_slope=-f./p;
pixel_2_mirr=zeros(1,length(p));

% map pixel indices to mirror indices
for i=1:length(p)/2-1
   pixel_2_mirr(i)=find(the_mirr_ray_slope<=the_pixel_slope(i),1,'first');

end
for i=length(p)/2+1:length(p)
   if ~isempty(find(the_mirr_ray_slope>=the_pixel_slope(i),1,'last'))
      pixel_2_mirr(i)=find(the_mirr_ray_slope>=the_pixel_slope(i),1,'last');
   end
end

tpt=131;

% correct the reflected ray slopes to make uniform for a wall 1 ft away
the_mirr_refl_corrected_ray_slope(pixel_2_mirr(pmin:pmax))=linspace(the_mirr_refl_ray_slope(pi
xel_2_mirr(pmin)),the_mirr_refl_ray_slope(pixel_2_mirr(pmax)),pmax-pmin+1);


i=pmin:pmax;

%o=349:921;
```

```
o=pixel_2_mirr(pmin):pixel_2_mirr(pmax);
wall_pts(o)=the_mirr_refl_ray_slope(o)*-dist+mirr_in_world(o);


equidist_wall_pts(o)=linspace(12,-9,pixel_2_mirr(pmax)-pixel_2_mirr(pmin)+1);


error_wall_pts=wall_pts-equidist_wall_pts;

the_section=the_mirr_refl_ray_slope(pixel_2_mirr(pmin:pmax));
corr_ind=zeros(1,length(the_section));


adj_ind=zeros(1,pixel_2_mirr(pmax)-pixel_2_mirr(pmin)+1);

newx=zeros(1,pixel_2_mirr(pmax)-pixel_2_mirr(pmin)+1);
adj_ind_pixel_plane=zeros(1,pixel_2_mirr(pmax)-pixel_2_mirr(pmin)+1);
for m=pixel_2_mirr(pmin):pixel_2_mirr(pmax);

   % this tell you how much the index of the real world must move to get
   % from the original image to the corrected image
   horiz_err=abs(wall_pts-equidist_wall_pts(m));
   [c adj_ind(m)]=min(horiz_err);
   newx(adj_ind(m))=x(m);
   if ~isempty(find(pixel_2_mirr==adj_ind(m)))
      adj_ind_pixel_plane(m)=find(pixel_2_mirr==adj_ind(m));
   end
end
%

[XI YI]=meshgrid(1:2*size(IM_OUT,2),1:2*size(IM_OUT,1));
[X Y]=meshgrid(1:size(IM_OUT,2),1:size(IM_OUT,1));

[b,m,n]=unique(adj_ind_pixel_plane,'first');
```

```
newp=adj_ind_pixel_plane(m);
IM_OUT2=IM_OUT;

new_IM=zeros(newp(length(newp)),size(IM_OUT2,2));
% make the first row of the new image = 1 because the search compares
% values to zero and we need to establish an upper bound
new_IM(1,:)=1;

% unaffected section
for i=1:pmin-1
   new_IM(i,:)=IM_OUT2(i,:);
end
j=pmin;

% affected section
for i= 2:length(newp)
   new_IM(newp(i),:)=IM_OUT2(i,:);
   j=j+1;
end

% perform manual linear interpolation along the vertical axis to fill in
% the zeros
for i=2:size(new_IM,1)
   if sum(new_IM(i,:)==0)
      uj=1;
      dj=1;
      upp=new_IM(i-uj,:);
      while sum(upp)==0
         uj=uj+1;

         upp=new_IM(i-uj,:);
      end
      lo=new_IM(i+dj,:);
      while sum(lo)==0
```

```
        dj=dj+1;
        lo=new_IM(i+dj,:);
      end

      betw_slope=(new_IM(i-uj,:)-new_IM(i+dj,:))/(dj+uj);
      for k=i-uj+1:i+dj-1
        new_IM(k,:)=new_IM(k-1,:)+betw_slope;
      end
    end
end

% unaffected section
for i= length(newp):size(IM_OUT2,1)
   new_IM(newp(length(newp))-length(newp)+i,:)=IM_OUT2(i,:);
end

% pre filter before downsampling
% new_IM=interp2(X,Y,new_IM,XI,YI);
% new_IM=upsample(new_IM,2);
% f=[0 0.30 0.40 1]; a=[1 1 0 0];
% b=firpm(22,f,a);   % obtaining filter coefficients
%
% [impResp,N] = impz(b,1);
% sepImpRespFir2d=compFir2D(impResp);
% new_IM=filter2(sepImpRespFir2d,new_IM);
%new_IM=downsample(new_IM,2);
%new_IM=interp2(X,Y,new_IM,XI,YI);
%new_IM=resample(new_IM,1,2);
figure(3)
%subplot(2,1,1)
new_IM=uint8(new_IM);
imshow(new_IM);%,[min(new_IM(:)) max(new_IM(:))]);

% i=pmin:pmax;
```

```
figure(2)
% test image
plot(x,mirr_in_world,'-',p,implane,'o')
%line([x(tpt) -12],[mirr_in_world(tpt) 24]);
%grline=line([x(tpt) 0],[mirr_in_world(tpt) f]);

dist=12;
pmin=1;
pmax=415;



axis([-dist dist -dist 10])
the_mirr_ray_slope=(mirr_in_world-f)./x;
for i=1:400
   tpt=pixel_2_mirr(i);
   line([x(pixel_2_mirr(i)) -dist],[mirr_in_world(pixel_2_mirr(i))
the_mirr_refl_ray_slope(pixel_2_mirr(i))*-dist+mirr_in_world(pixel_2_mirr(501))])
end

% for i=pmin:pmax
%     tpt=pixel_2_mirr(i);
%     line([x(pixel_2_mirr(i)) -dist],[mirr_in_world(pixel_2_mirr(i))
equidist_wall_pts(pixel_2_mirr(i))])
% end
```

Appendix B

```
function IM_OUT = omni_unwrap2(image_path,t,F)
% omni_unwrap2: takes an located at IMAGE_PATH and
%   unwraps it by taking a constant number of samples along concentric
%   rings about the center of the image and stores the values in IM_OUT.
%   Values are interpolated using a bilinear interpolation algorithm.
%   Additionally, approximate height values calculated given the mirror
%   profile F at radius t and specifications at the beginning of the code.
%   Note that this aspect of the code is not fully developed and therefore
%   not included in the report.
%
%
%


%-------------------------------------
% Rev#    Date       Who     Purpose
% ----  --------  ---------  ---------
%  000  11/18/07  D.Mabius   Original
%  001  11/19/07  D.Mabius   Fixed calculation of inner/outer radii.  Now
%                            draws center and inner/ outer circles.
%  002  12/07/07  D.Mabius   Added height function
```

```
%  003  12/09/07  D.Mabius   Added temporary section to produce plots for
%                           the presentation demonstrating sampling
%                           mismatch.
%
%-----------------------------------

%% Parameters

OUTPUT_RES = [1500 500];
IM_OUT = [];
F0 = 6.6675; %cm
F1 = 8.89; %cm (distance mirror is off the ground
D = 30.48; %cm
f = 1.5; %cm

%% Main

% Attempt to load image
try
   IM_IN = imread([image_path]);
catch
   error('Unable to load image');
```

```
end

% Convert to grayscale
IM_IN = rgb2gray(IM_IN);
IM_IN = double(IM_IN);

% Display image and ask user for input regarding center and inner/outer
% radii
figure(1);
imshow(IM_IN, []);
title('Original - Grayscale');
pause(0.001);
h = msgbox('Click to indicate the center, followed by the inner radius, then the
outer radius, then the edge of the mirror');
waitfor(h);
[x_in, y_in] = ginput(4);
center = [x_in(1), y_in(1)];
radin  = round(dist([x_in(2), y_in(2)], center));
radout = round(dist([x_in(3), y_in(3)], center));
radmax = round(dist([x_in(4), y_in(4)], center));

% Ensure inner radius is smaller than outer radius
```

```
if (radin >= radout)
   error('The outer radius must be larger than the inner radius');
end

% Display selected center and inner and outer limits
hold on;
plot(center(1),center(2),'g+');
circle(center,radin,1000,'g');
circle(center,radout,1000,'g');
hold off;

% Build radius and angle vectors
radii = radin:(radout-radin)/(OUTPUT_RES(2)-1):radout;
theta = 0:(2*pi)/OUTPUT_RES(1):2*pi-(2*pi)/OUTPUT_RES(1);   % don't sample
0 and 2 PI, only 0

% Build transform coordinate matrices
[T,R] = meshgrid(theta,radii);
[XI,YI] = pol2cart(T,R);
XI = XI+center(1);
YI = YI+center(2);
```

```
% Temporary plots for presentation
%
% disp_step = 10;
% disp_XI = XI(1:disp_step:end,1:disp_step:end);
% disp_YI = YI(1:disp_step:end,1:disp_step:end);
% hold on;
% plot(disp_XI(:),disp_YI(:),'r.','MarkerSize',5);
% hold off;
%
% figure(3);
% imshow(IM_IN, []);
% title('Original - Grayscale');
% hold on;
% plot(center(1),center(2),'g+');
% circle(center,radin,1000,'g');
% circle(center,radout,1000,'g');
% plot(disp_XI(:),disp_YI(:),'r.','MarkerSize',5);
% hold off;
% [disp_X_grid, disp_Y_grid] = meshgrid([1:disp_step:size(IM_IN,2)],
[1:disp_step:size(IM_IN,1)]);
% hold on;
% plot(disp_X_grid(:),disp_Y_grid(:),'b.','MarkerSize',5);
```

```
% hold off;



% Unwrap image
IM_OUT = interp2(IM_IN,XI,YI);

% Calculate height function
m_radius = t(end);
ti = (radii/radmax)*m_radius;
h = -1*height(t,F,F0,f,D,ti) + F1;

% Display image
figure(2);
imshow(IM_OUT, []);
figure(3);
plot(zeros(1,length(h)),h,'.');
plot(ti,h);

% Convert to uint8 before outputting
IM_OUT = uint8(IM_OUT);
```

```
%% Subfunctions
function d = dist(P1,P2)
% returns the distance between P1 and P2 where both P1 and P2 are two
% element vectors containing X and Y coordinates
d = sqrt((P1(1)-P2(1))^2 + (P1(2)-P2(2))^2);

function circle(center, radius, NOP, style)
% Plots a circle in current axis
THETA=linspace(0,2*pi,NOP);
RHO=ones(1,NOP)*radius;
[X,Y] = pol2cart(THETA,RHO);
X=X+center(1);
Y=Y+center(2);
plot(X,Y,style);

function h = height(t,F,F0,f,d,ti)
F = F+F0;
Fi = interp1(t,F,ti);
dFi = diff(Fi)./diff(ti);
dFi(end+1) = dFi(end);

h = Fi + ((2.*ti.*dFi-(Fi-f).*(1-dFi.^2))./(2.*(Fi-f).*dFi+ti.*(1-dFi.^2))).*(d-ti);
```

References

[1] Baker, Simon; Nayar, Shree K. "A Tutorial on Catadioptric Image Formation" available at

http://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/BAKER/main.html

[2] Swaminathan, Rahul; Grossberg, Michael D.; Nayar, Shree K. "Non-Single Viewpoint Catadioptric Cameras: Geometry and Analysis available at http://www.cs.columbia.edu/techreports/cucs-004-01.pdf

[3] Decco, Claudia; Gaspar, Jose; Winters, Niall; Santos-Victor, Jose. "OMNIVIEWS Mirror Design and Software Tools" IST-1999-29017 September 2001.

[4] Drocourt, Cyril; Delahoche, Laurent et. al. "Mobile Robot Localization Based on an Omnidirectional Stereoscopic Vision Perception System". IEEE International Conference on Robotics & Automation, May 1999.

[5] Nayar, Shree K.; Peri, Venkat. "OMNICAMERA: Omnidirectional Video Camera" available at http://www1.cs.columbia.edu/CAVE//omnicam/.