CIRN INTRODUCTION TO QUANTITATIVE COASTAL IMAGING TOOLBOX

Brittany L. Bruder + Katherine L. Brodie

Coastal and Hydraulics Laboratory
U.S. Army Engineer Research and Development Center
Duck, NC

May 26, 2020

Contents

Table of Figures

Executive Summary

The CIRN Introduction to Quantitative Coastal Imaging Toolbox is a collection of MATLAB scripts to produce geo-rectified images specifically tailored for quantitative analysis of coastal environments. The repository performs end-to-end georectification of oblique imagery from land-based multi/single-camera stations or stationary Unmanned Aircraft Systems (UAS). In addition to rectified frames, the toolbox produces ensemble products such as statistical or subsampled pixel collections for optical wave, current and bathymetric inversion analyses. While experts have employed similar scripts operationally for years, this toolbox is for photogrammetry novices. Demos teach fundamentals but also can be easily transitioned to turn-key applications for operational data processing.

1. Purpose

The CIRN Introduction to Quantitative Coastal Imaging Toolbox is series of MATLAB scripts that can be utilized to georectify sets of oblique imagery from stationary UAS and single (or multi) – camera land-based imagery for generation of classic coastal imaging data products [1]. The scripts prominently call and highlight a series of sub-functions that implement fundamental photogrammetry calculations (e.g. solving camera extrinsics and calculating a camera projection matrix). The scripts are prepopulated with demo UAS and multi-camera data and can be run as downloaded. Software and Data Requirements This user guide will outline: software and Data requirements; general workflow; and descriptions of scripts and sub-functions including input, output, and function dependencies. The last sections serve as references for equation and coordinate system definitions and literature references.

Software Requirements

The toolbox is a series of scripts designed in MATLAB Version 2019b. Older MATLAB versions may run the scripts, but this is not guaranteed. In addition, the toolbox requires the MATLAB Statistics and Machine Learning Toolbox for the *nlinfit* function.

The Coastal Imaging Research Network (CIRN) GitHub repository serves and manages the Introduction to Quantitative Coastal Imaging Toolbox. Updates, managed by git version control, are provided at this GitHub repository.

User-Data Requirements

To adapt to other data sets, the user is required to provide: image data with suitable ground control points (GCPs), GCP geographic coordinates, and camera intrinsic calibration parameters. More details on collection requirements for georectified imagery can be found in [2] and camera intrinsic calibrations at [3] .

2. Workflow

The application is a series of linear MATLAB m-scripts that can be run in different 'processing sequences' (Figure 1) depending on the collection mechanism (hovering UAS vs fixed land-based single (multi) -camera stations). Scripts work with multiple processing sequences to emphasize commonalities between processing steps between collection techniques as well as reduce replicative code. Sections of the processing sequence are characterized by their functionality.

Scripts are run sequentially as inferred by the alphanumeric prefixes in the filenames and Figure 1. Each script represents a key photogrammetric processing step or decision point, (e.g. what GCPs to use) and saves intermediate products (e.g. extrinsics, etc) for input into the next script. The intermediate products allow the user flexibility in processing and an ease of exploring processing decisions (e.g. changing coordinate systems or grid resolutions) without having to restart the entire end-to-end processing. Metadata such as GCP projection errors are saved in the output mat-files as well.

In addition to the demonstrative scripts in the main repository directory, there is a X_CoreFunctions folder that contains sub-functions that execute fundamental photogrammetry processes (e.g. plotting rectified images, undistorting images). These functions can also be used independently of the demonstrative scripts making them easily adaptable to a user's needs.

The remaining folders in the main directory contain demonstration data for UAS (X_UASDemoData) and mulit-camera land-based stations (X_FixedMultiCamDemoData). The processing sequences for each demo are highlighted in Figure 1. The demo scripts prefixed (A0-G2) are prepopulated to process the UASDemoData end-to-end. The FixedMultiCamDemoData path starts at D_gridGenExampleRect since geometry solutions are already provided, and requires users to uncomment blocks of code to execute. Scripts prefixed D, G1-G2 are also prepopulated for FixedMultiCamDemoData but these inputs are commented out. Both demos are demonstrated in Section 3. Sections 3 and 4 outline each processing script as well required inputs, outputs, and dependent functions. Example output is from demonstration data.

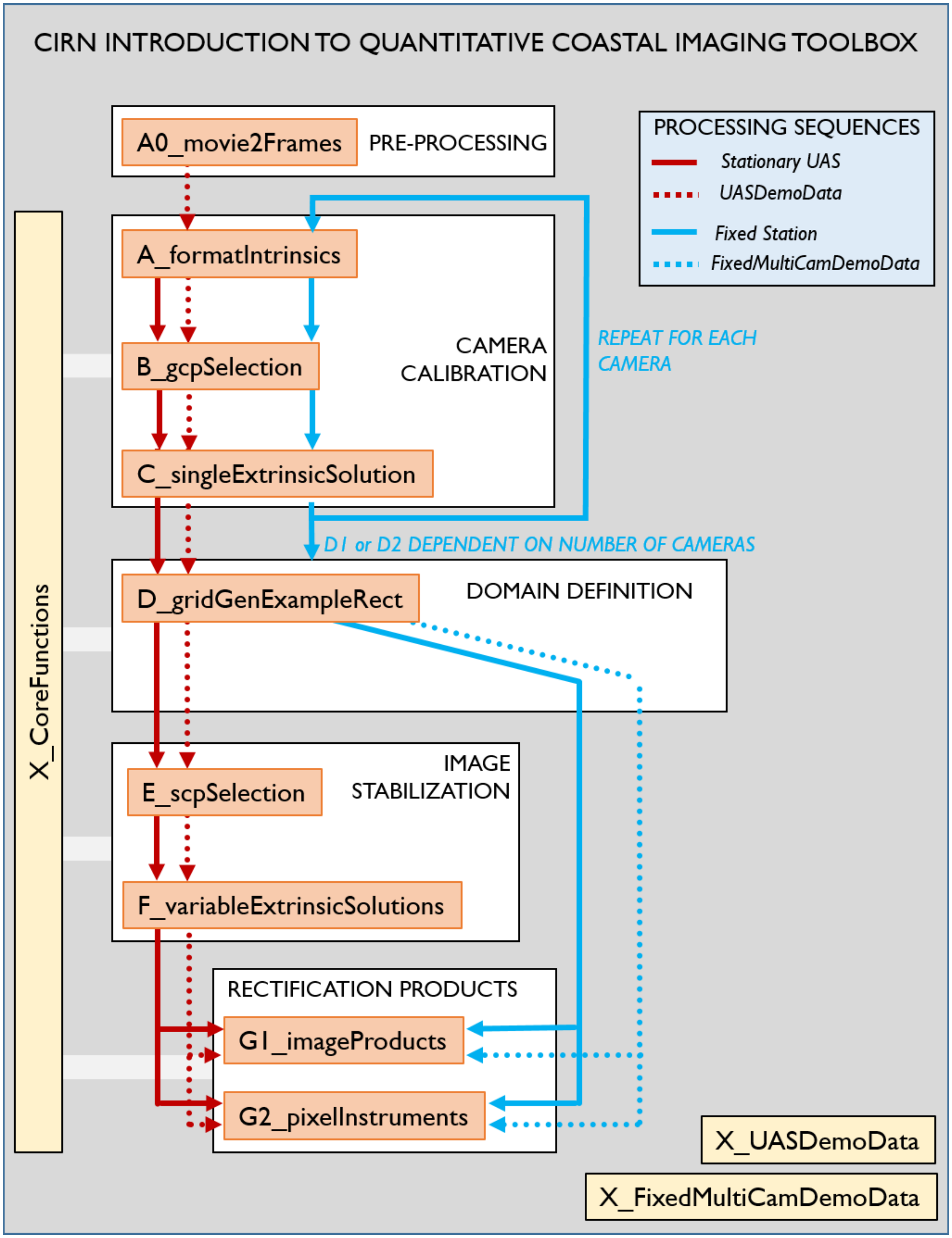


Figure 1: Introduction to Quantitative Coastal Imaging Toolbox repository and workflow. White boxes represents characterizations, yellow boxes represent folders, and orange boxes represent scripts.

3. Script Descriptions

A0_movie2frames

Description:

This function outputs frames (images) from a movie file with a specified frame rate. This function may or may not need to be run first in the progression. It will typically be needed for movie files from UAS collects to obtain images for analysis. However, it can be utilized for any movie file even from a fixed station.

Input:

Input is entered by user into the script in Section 1 and 2.

Section 1: Loading and Saving Information

Variable	*Description*
mpath	Filepath where movie is located. Movie should be a format readable by MATLAB and computer video CODEC.
oname	Output string of the basename prefix for the images to be saved under. Example: DuckC1 will be DuckC1_00001, DuckC2_00002, etc.
odir	Filepath where the output individual images will be saved.

Section 2: Timing Information and Framerate

Variable	*Description*
to	Enter the time of the first frame in [year, month, day, hr, min, sec] format. If unknown leave as all zeros. Timing will just refer the first frame as t=0s.
framerate	Enter the Desired Frame Rate in frames/second (fps). Note, desired frame rate should be a multiple or factor of the video orginal frame rate for equally timed frames. Example: for a video at 30 fps one can accurately export frames at 1,2,3,5,6,10,15 or 30 fps (multiples). Or for lower framerates (<1 fps) the framerate would be (1./(N+n/30)) where N is the integer number of seconds between frames and n is the fractional number of frames (A Frame every 2.5 seconds would be 1./(2+15/30)=.4 fps).

Output:

A series of images (.png) in odir. Timing information will be saved in the name of the file in epoch time. If initial time not specified, timing will be referenced to the first frame at 0s. Times will be rounded to the nearest milliscond and expressed in milliseconds.

Required Core Sub-Functions:

None.

A_formatIntrinsics

Description:

This function initializes the intrinsics matrix for a given camera. It is to be run first in the progression. It should be run for each camera in a multi-camera fixed station, every time the focus is adjusted, a new lens is added, or a new enclosure is introduced. For a UAS camera, it should be run for each type of recording mode (4K Video, 12MP snapshot, etc). It can entered manually or refer to a Cal-Tech Camera Calibration file, a third-party MATLAB repository to perform camera intrinsic calculations [3]. Note, this function assumes the calib_results is for a standard distortion model, not fish eye. Model is outlined in Section 5.

Input:

Input is entered by user into the script in Section 1, 2, and possibly 3.

Section 1: Loading and Saving Information

Variable	*Description*
oname	Output string for the basename for the intrinsic matfile to be saved under.
odir	Output filepath where the intrinsic mat file will be saved.

Section 2: Intrinsics

Variable	*Description*
iopath	Filepath of the saved Caltech calibration results (Calib_result.mat). If user is going to enter manually, this should be left empty {} and entered in Section 3.

Section 3: Manually Entered Intrinsics (Optional)

Variable	*Description*
intrinsics	A [1x11] vector describing the focal lengths and distortion of the camera/lens combination. Distortion coefficients are defined by [3] and defined in Section 5.

Output:

A .mat file saved as *oname_IO.mat* in *odir*. Will contain following variables.

Variable	*Description*
intrinsics	A [1x11] vector describing the focal lengths and distortion of the camera/lens combination. Distortion coefficients are defined by [3] and defined in Section 5.

Required Core Sub-Functions:

caltech2CIRN

B_gcpSelection

Description:

This function initializes the GCP structure for a given camera. The user will load a distorted image, click on GCPs, and the function will save the distorted UVd coordinates and image metadata for a given camera.

How to use clicking mechanism: The user can zoom and move the image how they please and then hit 'Enter' to begin clicking mode. A left click will select a point, a right click will delete the nearest point to the click. After a left click, the user will be asked to enter a GCP number to identify the GCP in the command window. The user can then zoom again until hitting enter to select the next point. To end the collection, hit enter to enter clicking mode (the cross hairs) and click below the image where it says 'Click Here to End Collection.' Be sure to be zoomed out completely when ending a collection. The user can click GCPs in any order they would like.

For the uasDemoData, the first image is has been modified to provide GCP suggestions. Users should click on targets and provide the same numbering to work with the rest of the demonstration workflow. Figure 2 shows what the final window for the uasDemoData progression should be.

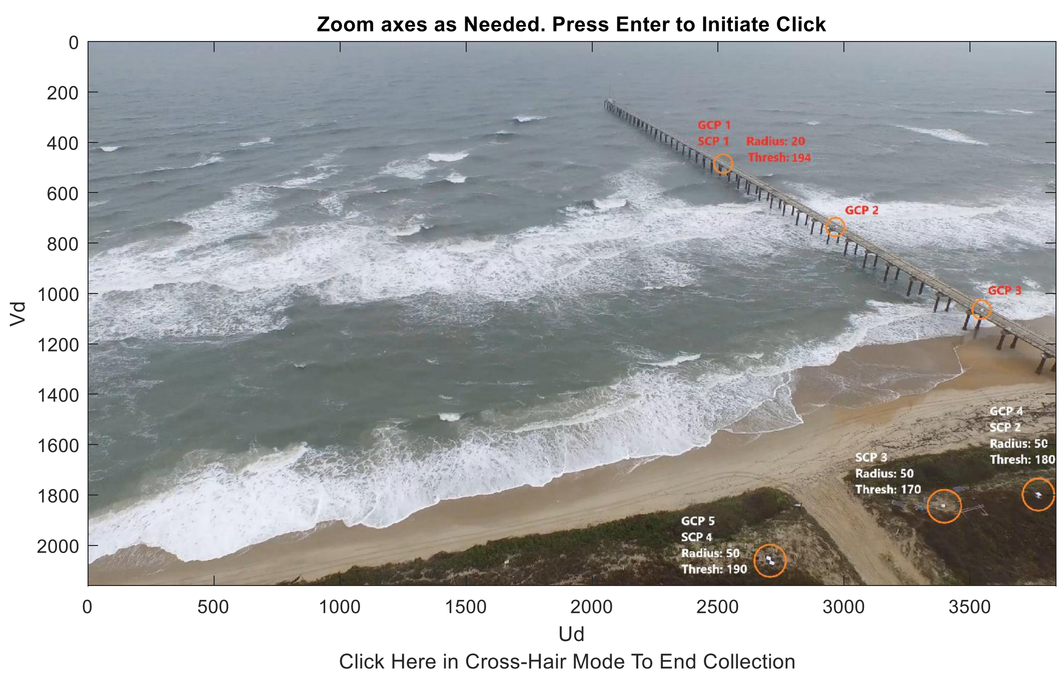


Figure 2: Example GCP click window with specified GCP numbers in B_gcpSelection for uasDemoData

This function is to be run second in the progression for each camera in a multi-camera fixed station or UAS flight (or if a recording mode was changed midflight). GCP calibration should occur any time a camera has moved for a fixed station, the first frame in a new UAS collect, or intrinsics have changed.

Input:

Input is entered by user into the script in Sections 1 and 2. Users will then enter information by clicking GCPs and entering an identifying number in the command window.

Section 1: Saving Information

Variable	*Description*
oname	Output string for the basename for the GCP mat files to be saved under.
odir	Output filepath where the GCP mat file will be saved.

Section 2:GCP Image

Variable	*Description*
imagePath	Filepath of the saved image for clicking. For UAS processing this should be the first image of the collection. For fixed station, it should be any frame where GCPs are visible.

Output:

A .mat file saved as *oname_gcpUVdInitial.mat* in *odir*. Will contain following variables.

<table>
<tr><td>Variable</td><td colspan="2">Description</td></tr>
<tr><td rowspan="4">gcp</td><td colspan="2">A structure with each entry corresponding to a GCP. Note, the index entry in the structure may not correspond with the GCP identifying number.

Fields of structure are:</td></tr>
<tr><td>UVd</td><td>[1 x 2] Vector of distorted Image coordinates of GCP</td></tr>
<tr><td>num</td><td>Identifying GCP number entered by user.</td></tr>
<tr><td>imagePath</td><td>String of filepath of image used for GCP clicking. Same as imagepath in Section 2.</td></tr>
</table>

Required Core Sub-Functions:

None

C_singleExtrinsicSolution

Description:

This function solves the extrinsics (EO) for a given camera for use in the toolbox. The user will load gcp and intrinsic (IO) information via input files. The user will specify coordinate system information as well as initial extrinsics rough guesses. The function will output the solved extrinsics in the form of the vector extrinsics, metadata information in initialCamSolutionMeta, and a reprojection error figure.

Note, the extrinsics solution will be in the same coordinate system as the GCPs. Regardless of what the user enters, this will be referred to as the WORLD coordinate system with a subscript W. It is encouraged that the user enter GCPs in a geographic coordinate system (State Plane, UTM, etc). The toolbox will complete a coordinate system rotation in subsequent functions. Also, the nlinfit solver is very sensitive to the initial guess; so it must be an educated guess. It is particularly sensitive to the guessed azimuth, tilt, and swing. If incorrect, nlinfit will error or provide a nonsensical answer. Please check veracity of provided extrinsics. Descriptions of the World Coordinate System and extrinsic definitions (particularly azimuth, tilt, and swing) are in Section 6.

This function is to be run third in the progression for each camera in a multi-camera fixed station or for each collection for a UAS platform. GCP calibration and geometry solution calculation should occur any time a camera has moved for a fixed station, the first frame in a new UAS collect, or intrinsics has changed.

Input:

Input is entered by user into the script in Sections 1-4.

Section 1: Saving Information

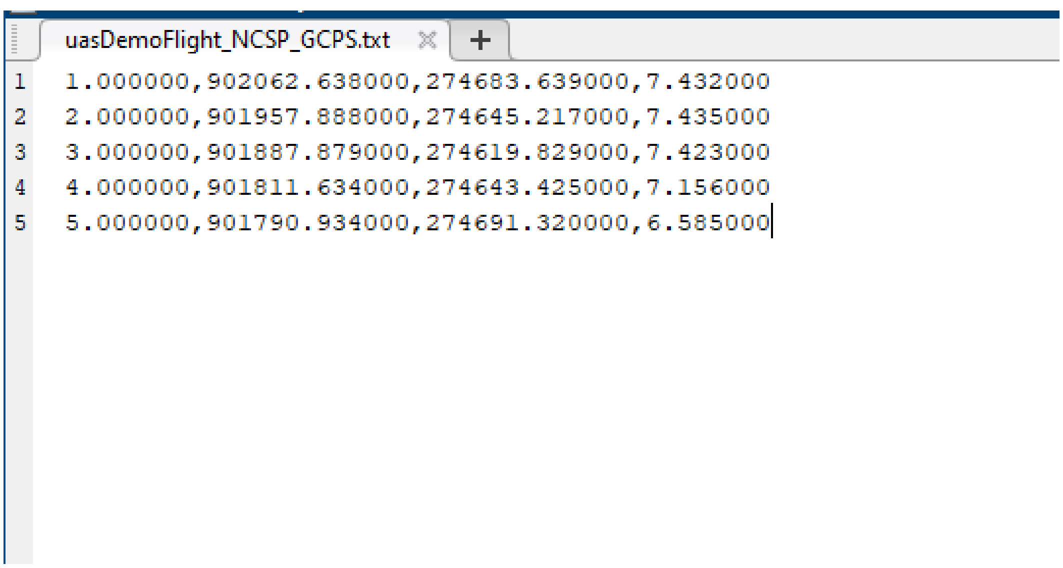
Variable	*Description*
oname	Output string for the basename for the Extrinsic/Intrinsic Solution mat files to be saved under.
odir	Output filepath where the Extrinsic/Intrinsic Solution mat file will be saved.

Section 2:Intrinsics

<table>
<tr><th>Variable</th><th colspan="2">Description</th></tr>
<tr><td rowspan="2">iopath</td><td colspan="2">Filepath of the intrinsics matfile output by A_formatIntrinsics. Matfile should contain at minimum the following variable. Note, the intrinsics should correspond to the recording mode and camera/lens for the image taken in B_gcpSelection, imagePath.</td></tr>
<tr><td>intrinsics</td><td>A [1x11] vector describing the focal lengths and distortion of the camera/lens combination. Distortion coefficients are defined by [3] and defined in Section 6.</td></tr>
</table>

Section 3: GCP Information

<table>
<tr><th>Variable</th><th colspan="3">Description</th></tr>
<tr><td rowspan="4">gcpUVdPath</td><td colspan="3">Filepath of the Distorted GCP UV Coordinates produced by B_gcpSelection. The intrinsics of the corresponding image from which the UVd GCP coordinates were derived from should match that entered in Section 2. If not produced by B, minimum variables are listed below.</td></tr>
<tr><td rowspan="3">gcp</td><td colspan="2">A structure with each entry corresponding to a GCP. Note, the index entry in the structure may not correspond with the GCP identifying number.
Required fields of structure are:</td></tr>
<tr><td>UVd</td><td>[1 x 2] Vector of distorted Image coordinates of GCP</td></tr>
<tr><td>num</td><td>Identifying GCP number entered by user.</td></tr>
<tr><td>gcpXyzPath</td><td colspan="3">filepath of the GCP World coordinates. File should be a four column comma delimted txt file with columns representing gcp number, x coordinate, y coordinate, and z coordinate. Rows will correspond to each GCP. GCP numbers should match with those entered in B_gcpSelection. Example text file structure in Figure 3. It is encouraged to enter GCPs in a geographic coordinate system (State Plane, UTM, etc).</td></tr>
<tr><td>gcpCoord</td><td colspan="3">String that describes the coordinate system (World) and units of the entered GCPs in gcpXyzpath</td></tr>
<tr><td>imagePath</td><td colspan="3">Filepath of the image you would like GCP reprojection checked against (plotted in). This should be the same image used in B_gcpSelection (imagePath) if you are doing a UAS collect or a moving camera. For a fixed camera, it can be any image where the GCPs are visible.</td></tr>
<tr><td>gcpsUsed</td><td colspan="3">The numbers of GCPs you would like to use for the solution. Numbers must match gcp.num values found in gpcUvPath and gcpXyzPath files. You do not have to use all of the clicked GCPS or GCPS listed in the file.</td></tr>
</table>


uasDemoFlight_NCSP_GCPS.txt

```
1.000000,902062.638000,274683.639000,7.432000
2.000000,901957.888000,274645.217000,7.435000
3.000000,901887.879000,274619.829000,7.423000
4.000000,901811.634000,274643.425000,7.156000
5.000000,901790.934000,274691.320000,6.585000
```

Figure 3: Example GCP input txt file for C_singleExtrinsicSolution

Section 4: Solution Information

Variable	*Description*

extrinsicsInitialGuess	A [1x6] vector of the initial guess of extrinsics, the EO solution, for the corresponding camera image. Extrinsics is formatted as [x y z azimuth tilt swing] where xyz correspond to the same world coordinate system as gcps entered in gcpXyzPath in Section 3. Azimuth, tilt and swing should be in radians. For UAS, this information can be estimated from the autopilot. For fixed camera stations it is suggested you survey in the location of the cameras. Further descriptions of the extrinsics vector and world coordinate systems can be found in Section 6.
extrinsicsKnownsFlag	Enter the number of knowns, or what you would like fixed in your EO solution. 1 represents fixed where 0 represents floating (solvable) for each value in extrinsics. Review [4] for a discussion of which extrinsics may float relative to the number of GCPs and the effect on rectification accuracy.

Output:

A .mat file saved as *oname_IOEOInitial.mat* in *odir*. Will contain following variables.

<table>
<tr><td>Variable</td><td colspan="3">Description</td></tr>
<tr><td rowspan="15">initialCamSolutionMeta</td><td colspan="3">Structure with metadata about extrinsic solution. Fields are listed below. Those left blank are the same entered in the input sections.</td></tr>
<tr><td>gcpUsed</td><td colspan="2"></td></tr>
<tr><td>gcpRMSE</td><td colspan="2">A [1 x 3] vector with the root mean squared error between the entered world XYZ coordinates of the GCPs in gcpXyzPath and world XYZ coordinates found using the extrinsic solution and UVd coordinates in gcpUvdPath. All GCPS used, not just those specified in gcpsUsed. Units are in units entered in gcpXYZPath.</td></tr>
<tr><td>extrinsicsInitialGuess</td><td colspan="2"></td></tr>
<tr><td>extrinsicsKnownsFlag</td><td colspan="2"></td></tr>
<tr><td>extrinsicsUncert</td><td colspan="2">A [1 x 6] vector with the uncertainty value of each solved extrinsic value provided by nlinfit. Units are in units entered in gcpXYZPath for position and radians for pose.</td></tr>
<tr><td>imagePath</td><td colspan="2"></td></tr>
<tr><td>worldCoordSys</td><td colspan="2">Description of world coordinate system, same as gcpCoord.</td></tr>
<tr><td rowspan="7">gcp</td><td colspan="2">Same as entered in Section 3 but with the additional fields</td></tr>
<tr><td>x</td><td rowspan="3">XYZ coordinates as entered in gcpXyzPath</td></tr>
<tr><td>y</td></tr>
<tr><td>z</td></tr>
<tr><td>CoordSys</td><td>Description of world coordinate system, same as gcpCoord.</td></tr>
<tr><td>xReprojError</td><td rowspan="2">Difference between the entered world XYZ coordinates of the GCPs</td></tr>
<tr><td>yReprojError</td></tr>
</table>

			in gcpXyzPath and world XYZ coordinates found using the extrinsic solution and UVd coordinates in gcpUvdPath.
intrinsics	A [1x11] vector describing the focal lengths and distortion of the camera/lens combination. Distortion coefficients are defined by [3] and defined in Section 6.		
extrinsics	A [1x6] vector describing the camera pose and position in WORLD coordinates (whatever coordinate system GCPs were entered in). Extrinsic values are defined in Section 6 along with coordinate systems.		

A figure with the original 'clicked' GCPs plotted on the imagePath in red along with the reprojected GCP values using the extrinsic solution in yellow. Example from uasDemoData shown In Figure 4. Note, red is difficult to see because it is a high accuracy solution and yellow is directly on top.

Figure 4: Example reprojection figure in C_singelExtrinsicSolution for uasDemoData to observe extrinsic solution accuracy.

Required Core Sub-Functions:

extrinsicsSolver
xyz2DistUV
intrinsicsExtrinsics2P
distortUV
distUV2XYZ
undistortUV
Requires Also: MATLAB Statistics and Machine Learning Toolbox

D_gridGenExampleRect

Description:

This function generates grids for rectification in both world global and world local rotated coordinate systems. The function rectifies a single oblique image into rectified imagery for a one or multiple cameras. The user will load an intrinsic and extrinsics solution (IOEO) for each camera via input files. The user will specify the resolution and limits of the rectified grid and local rotation parameters for a local coordinate system. The user can enter the grid parameters in local or world coordinates; the function will make a corresponding grid encompassing the other system. The function will produce figures and save .png images of the georectified imagery in both coordinate systems. In addition, the prescribed grids will be saved as well in a mat file.

The user input of the code is prescribed for UASDemoData. However, one can uncomment lines in Section 4 for the FixedMultiCamDemoData.

This function is to be run fourth in the progression to evaluate camera extrinsic solutions and georectification grids. For UAS it should be run on the first image used for camera IOEO calibration. For fixed camera stations it can be ran on any imagery with the corresponding IOEO.

Input:

Input is entered by user into the script in Sections 1-4.

Section 1: Saving + Output Information

Variable	*Description*
oname	Output string for the basename for the example rectified images to be saved under.
odir	Output filepath where the rectification grid mat file will be saved.
gname	String for the basename for the rectification grid.
teachingMode	A flag to display intermediate steps in rectification, i.e showing transformation of XYZ points to Image UVd points. It is implemented in the sub-function imageRectification. For demo purposes, it should =1 to show intermediate steps. Otherwise set to 0 if not desired.

Section 2: Loading Information

<table>
<tr><th>Variable</th><th colspan="2">Description</th></tr>
<tr><td>imagePath</td><td colspan="2">Filepath of the image you would like rectified. This should be the same image used in B_gcpSelection and C_singleExtrinsicSolution (imagePath) if you are doing a UAS collect or a moving camera. For a fixed camera, it can be any image where intrinsics and extrinsics are valid.</td></tr>
<tr><td rowspan="2">ioeopath</td><td colspan="2">Filepath of the saved CIRN IOEO calibration results produced by C_singleExtrinsicSolution. If not produced by C, minimum variables are listed below.</td></tr>
<tr><td>intrinsics</td><td>A [1x11] vector describing the focal lengths and distortion of the camera/lens combination. Distortion coefficients are defined by [3] and defined in Section 6.</td></tr>
</table>

	extrinsics	A [1x6] vector describing the camera pose and position in WORLD coordinates (whatever coordinate system GCPs were entered in). Extrinsic values are defined in Section 6 along with coordinate systems.
NOTE: For multiple cameras, these values are entered as cell entries for each camera. The extrinsics for each camera should all be in the same world coordinate system. It is up to the user to make sure the entries for both variables correspond to the same cameras.Even for a single camera, both impath and ioeopath need to be entered as cell functions with one entry.		

Section 3: Rectification Information

Variable	*Description*
wolrdCoord	String of description of the World coordinate system for your own records. The world coordinate system of these should be the same as the IOEO specified in C_singleExtrinsicSolution (gpsCoord, camSolutionMeta.worldCoordSys).
Coordinate System Info: Enter the origin and angle if you would prefer rectified output to be in a local rotated right hand coordinate system. CIRN local coordinate systems typically have the new local X pointed positive offshore, Y oriented alongshore, and the origin onshore of the shoreline. If fields are entered, user will still be able to rectify in both local and world coordinates. Note, if user already specified their world coordinates as a local system, or do not desire a rotation, set localOrigin and localAngle to [0,0] and 0. More description of coordinate systems is in Section 6.	
localOrigin	Origin of new local coordinate system in world coordinates. should be in the same coordinate system of GCPs (worldCoord).
LocalAngle	relative angle between the new (local) X axis and old (World) X axis, positive counter-clockwise from the old (world) X. Units are degrees.
localFlagInput	Flag if you would like to INPUT your grid in rotated local coordinates. 1=local coordinates, 0= world coordinates. The function will still rectify both coordinate systems regardless of localFlagInput value. LocalFlagInput only dictates the coordinate system of the input and which direction the rotation will occur. If localOrigin and localAngle =0; this value is irrelevant.
Grid Specifications:	
ixlim	A [2 x1] vector specifying the x or y limits of the rectification grid.Units should be consistent with worldCoord, coordinate system dependent on localFlagInput.
iylim	
idxdy	X and Y resolution of grid in units consistent with worldCoord
iz	The elevation you would like your gridrectified as. Typically, CIRN specifies an constant elevation across the entire XY grid consistent with the corresponding tidal level the world coordinate system. For more accurate results a spatially variable elevation grid should be entered if available. However, this code is not designed for that. If you would like to enter a spatially variable elevation, say from SFM along with tidal elevation, enter your grid as iZ. It is up to the user to make sure it is same size as iX and iY. Spatially variable Z values are more significant for run up calculations than for current or bathymetric inversion calculations. It can also affect image rectifications if concerned with topography representation. What does alter bathymetric inversion, surface current, and run-up calculations is a temporally variable z elevation. So make sure the elevation value corresponds to the correct tidal value in time and location. For short UAS collects, we can assume the elevation is relatively constant during the

	collect. However for fixed stations we would need a variable z in time. This function is designed for rectification of a single frame, so z is considered constant in time. Value should be entered in the World Coordinate system and units.

Section 4: Multi-Camera Demo
Uncomment this section for the multi-camera demo. Impath and ioeopath should be entered as cells, with each entry representing a different camera. It is up to the user to ensure that entries between the two variables correspond. Extrinsics between all cameras should be in the same World Coordinate System. Note that no new grid is specified in this example since the data for the multi-cameras cover the same region as the UAV data; hence, the cameras and images are all rectified to the same gird.

Output:

A .mat file saved as *GRID_gname.mat* in *odir*. Will contain following variables. Variables left blank are the same as what is input by user above.

Variable	*Description*
X	NxM grids in World coordinates where N is the number of rows (direction of varying Y) and M is the number of columns (direction of varying X).
Y	
Z	
worldCoord	
localAngle	
localOrigin	
localX	NxM grids in rotated Local coordinates (according to localAngle and localOrigin) where N is the number of rows (direction of varying Y) and M is the number of columns (direction of varying X).
localY	
localZ	

Example georectified images of those specified in impath and using specified grid parameters saved as *oname_World.png and oname_Local.png* in *odir*.

Depending on teachingMode input, two figures for each camera showing complete georectification as well as XYZ grid points reprojected into UVd image space for both world and local coordinates. Examples shown below in Figure 5.

Required Core Sub-Functions:

imageRectification
xyz2DistUV
cameraSeamBlend
intrinsicsExtrinsics2P
distortUV
localTransformExtrinsics
localTransformPoints
localTrasnfromEquiGrid

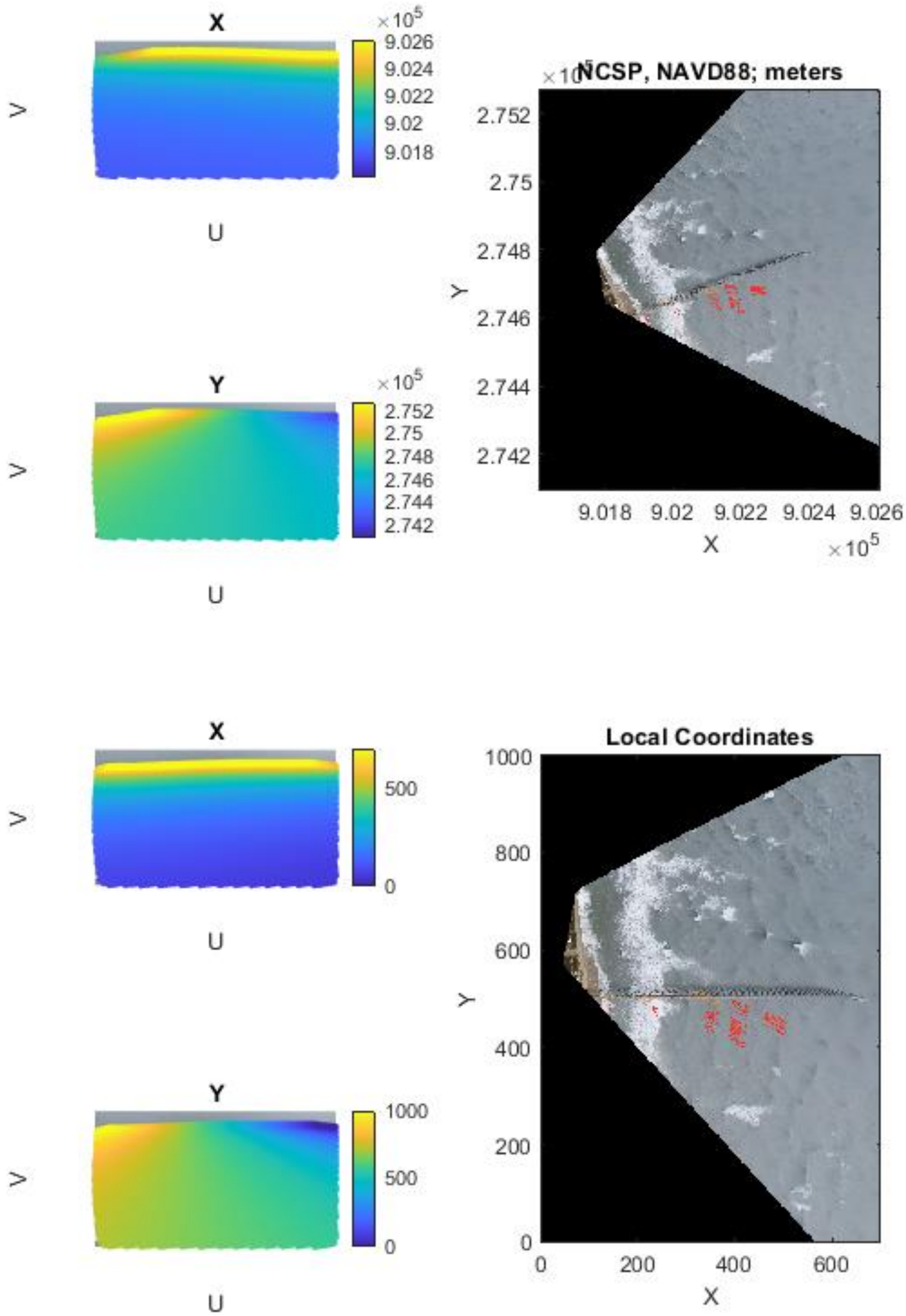


Figure 5: Example figure output for D_gridGenExampleRect for uasDemoData for World (Top) and Local (Bottom) coordinates.

E_scpSelection

Description:

This function initializes the SCP (stabilization control points) structure for a given camera for use in the toolbox. The user will load a given DISTORTED image for where the IOEO has been calculated already using C_singleExtrinsicSolution and then select at least 4 bright or dark points that will be used to stabilize the image (i.e. they must be in the frame throughout the collection). Additional parameters such as expected movement radius and intensity threshold will also be entered.

The clicking mechanism works similar to B_gcpSelection. The user can zoom and move the image until they hit enter to go into clicking mode (cross-hairs). Note, the click does not have to be as precise as B_gcpSelection. The user should click a bright (dark) target that is brighter (darker) than what is around it and does not move. The bright target should not be in the water with breaking waves rushing past. The user will enter a SCP number, and radius (in pixels) to search for the point. It is best to be as small as possible and not include bright (dark) pixels of other objects. For example, if on a pier, the radius should be small enough to exclude any water pixels. The radius should appear in the figure after entry in red. Hit enter with an empty input when done. An example SCP Click and Radius is in Figure 6.

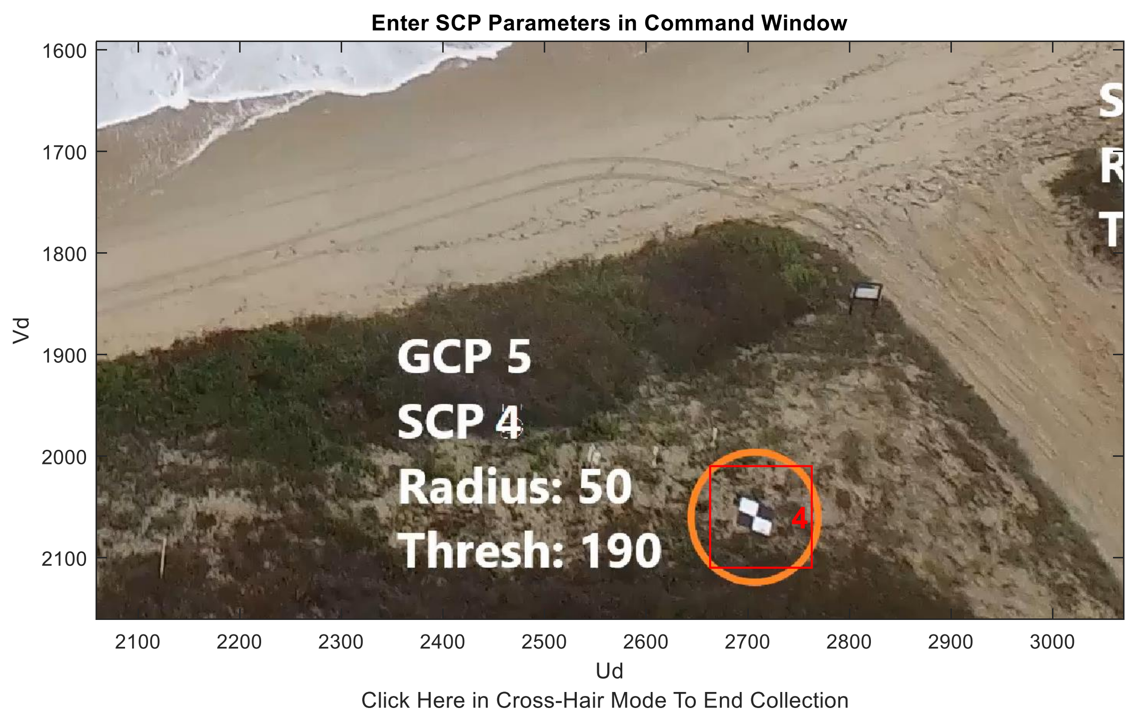


Figure 6: Example SCP click and radius (red) in E_scpSelection for uasDemoData

Then the user will enter a threshold value in the command window deliminating bright points from dark points, it is the center of the bright (dark) points that will be considered the SCP point. Ultimately, any points above this threshold in the search radius will be considered good and below not; then the center of area of the good points will be considered the SCP. This will be updated in anew figure to show the estimated SCP center; an example of which is in Figure 7. If the threshold is very high, it is probably not a good SCP point and may error if any slight changes in pixel value (>245). Hit enter with an empty value when complete. When done with selection go in to clicking mode and click below the X axis.

The user should pick at least 4 points. Note, if GCPs are selected, SCP and GCP point numbers do not have to match. For uasDemoData, suggested thresholds, numbers, and radii are suggested in the image frame.

Note: SCP 1 for uasDemoData is small and slightly difficult to see. However it is important to get a spread in SCPs across the image just like GCPs. This is why it is selected.

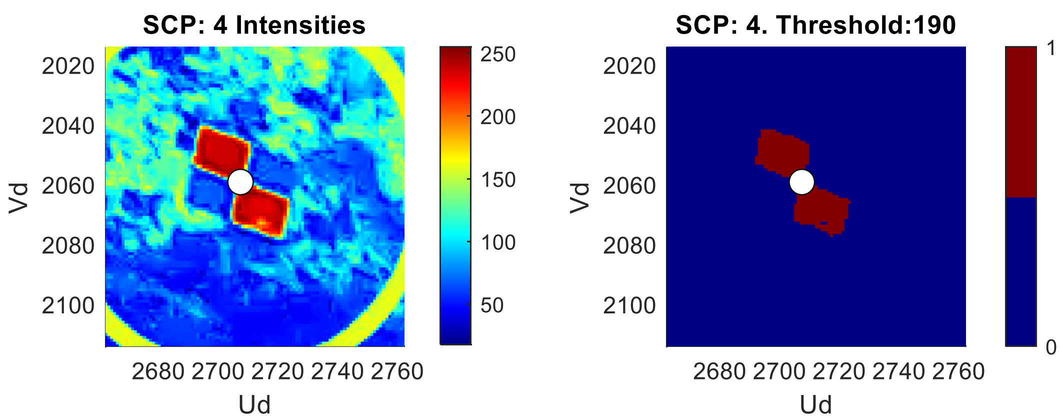


Figure 7: Panel displaying pixel intensity values within search radius of SCP 1 and resultant center of area calculations (white dot) when applying binary threshold in E_scpSelection for uasDemoData

This function is to be run fifth in the progression to identify stabilization control points (SCP) for collections of imagery where the camera may have moved. For UAS it should be run on the first image used for camera IOEO calibration. For fixed camera stations, trying to correct for movement, it should be run on the last known image where the IOEO is known to be valid.

Input:

Input is entered by user into the script in Sections 1 and 2. Users will then enter information by clicking SCPs and entering an identifying number, radii, and threshold in the command window.

Section 1: Saving Information

Variable	*Description*
oname	String for the basename for the SCP mat files to be saved under.
odir	Filepath where the SCP mat file will be saved.

Section 2: SCP Image

Variable	*Description*
imagePath	Filepath of the saved image for clicking. For UAS processing this should be the first image of the collection used in C-SingelExtrinsicSolution and B_gcpSelection (imagePath). For fixed station, it should be any frame where SCPs are visible and the previously known solution is viable.
brightFlag	String of either 'bright' or 'dark' to identify whether bright or dark SCPs will be identified. White objects on dark backgrounds should be 'bright' where dark objects on light backgrounds should be 'dark'.

Output:

A .mat file saved as *oname_scpUVdInitial.mat* in *odir*. Will contain following variables; variables left blank are defined in the user input.

<table>
<tr><th>Variable</th><th colspan="2">Description</th></tr>
<tr><td rowspan="7">scp</td><td colspan="2">A structure with each entry corresponding to a SCP. Note, the index entry in the structure may not correspond with the SCP identifying number.
Fields of structure are:</td></tr>
<tr><td>UVd</td><td>[1 x 2] Vector of distorted Image coordinates of SCP</td></tr>
<tr><td>num</td><td>Identifying SCP number entered by user.</td></tr>
<tr><td>R</td><td>Search Radius in Pixels (expected movement)</td></tr>
<tr><td>T</td><td>Threshold pixel intensity value for determining Center of Area for Bright Pixels (value 0-254).</td></tr>
<tr><td>imagePath</td><td></td></tr>
<tr><td>brightFlag</td><td></td></tr>
</table>

Required Core Sub-Functions:

thresholdCenter

F_variableExtrinsicSolution

Description:

This function generates extrinsics for a series of subsequent images for where the extrinsic solution of the first frame is known, calculated in C_singleExtrinsicSoltuion. Given a list of images, Stabilization Control Points UVd coordinates (from E_scpSelection), and SCP elevations, this function will determine the extrinsics of each subsequent frames.

Similarly to C_singleExtrinsicSolution, the script uses a nonlinear solver to minimize the error between the reprojected XYZ values reprojected in UV space and those indentified in the imagery. However, the images are not 'clicked' but rather found by the threshold center of area calculations. The XYZ SCP files are not provided by a text file, but rather determined from georectifying the UV coordinates using the previous frames extrinsics (the user does specify a SCP elevation estimate in the script). The script runs autonomously with no user-interaction, calculated SCP center of areas are plotted for each new frame in a figure; the function outputs a mat file with IOEO solutions for each frame as well as figure plotting the camera extrinsics as a function of time.

This function is to be run sixth in the progression to solve extrinsics for a moving camera.

Input:

Input is entered by user into the script in Sections 1-3.

Section 1: Saving Information

Variable	*Description*
oname	Output string for the basename for the extrinsic solution mat file to be saved under.
odir	Output filepath where the SCP mat file will be saved.

Section 2: Initial Frame Information

<table>
<tr><td>Variable</td><td colspan="2">Description</td></tr>
<tr><td>imagePath</td><td colspan="2">Filename of the first image where IOEO is known. It should be the same image you did the initial solution for in C_singleExtrinsicsSolution (imagepath). It should be in the same directory as your other collection images imagesDirectory.</td></tr>
<tr><td rowspan="3">ioeopath</td><td colspan="2">Filepath of the saved CIRN IOEO calibration results produced by C_singleExtrinsicSolution. If not produced by C, minimum variables are listed below.</td></tr>
<tr><td>intrinsics</td><td>A [1x11] vector describing the focal lengths and distortion of the camera/lens combination. Distortion coefficients are defined by [3] and defined in Section 6.</td></tr>
<tr><td>extrinsics</td><td>A [1x6] vector describing the camera pose and position in WORLD coordinates (whatever coordinate system GCPs were entered in). Extrinsic values are defined in Section 6 along with coordinate systems.</td></tr>
</table>

<table>
<tr><td rowspan="6">scppath</td><td colspan="3">Filepath of the Distorted SCP UV Coordinates produced by E_scpSelection. SCP point should correspond to those in imagePath. If not produced by E, minimum variables/fields are listed below.</td></tr>
<tr><td rowspan="5">scp</td><td colspan="2">A structure with each entry corresponding to a SCP. Note, the index entry in the structure may not correspond with the SCP identifying number.
Fields of structure are:</td></tr>
<tr><td>UVd</td><td>[1 x 2] Vector of distorted Image coordinates of SCP in imagepath</td></tr>
<tr><td>num</td><td>Identifying SCP number entered by user.</td></tr>
<tr><td>R</td><td>Search Radius in Pixels (expected movement)</td></tr>
<tr><td>T</td><td>Threshold pixel intensity value for determining Center of Area for Bright Pixels (value 0-254).</td></tr>
<tr><td>scpz</td><td colspan="3">Elevations, or z values for each SCP entry in the scp structure saved in scppath. The first column is the corresponding scp.num and the second value is the estimated elevation for the point. Z value should be in same world coordinate system and units as IOEO (worldCoord).</td></tr>
<tr><td>scpZcoord</td><td colspan="3">Coordinate system of the elevations entered in for scp. Elevations should be in same world coordinate system and units as IOEO.</td></tr>
</table>

Section 3: Collection Imagery

Variable	*Description*
imageDirectory	The directory where the collection images are stored. Note, the names of the images should be so that MATLAB lists them in order in the current directory window. To do this, images must be numbered with the same number of digits. Example: numbering should be 00001,00002,00003 etc. NOT 1,2,3. Also acceptible is any date number either in matlab datenum or epochtime as output in A0_move2frames. Also, only have the images in the folder. Nothing else.
dts	Enter the dt in seconds of the collect. Should be 1./frameRate frin A0_movie2frames. If not known, leave as {}. Under t, images will just be numbered 1,2,3,4.
to	The time of the initial image in MATLAB Datenum format; if unknown % leave empty {}, to will be set to 0.

Output:

A .mat file saved as *oname_IOEOVariable.mat* in *odir*. Will contain following variables; variables left blank are defined in the user input.

<table>
<tr><td>Variable</td><td colspan="2">Description</td></tr>
<tr><td rowspan="3">variableCamSolutionMeta</td><td colspan="2">Structure with metadata about extrinsic solutions. Fields are listed below. Those left blank are the same entered in the input sections.</td></tr>
<tr><td>sccpath</td><td></td></tr>
<tr><td>scp</td><td></td></tr>
</table>

	scpZcoord	
	ioeopath	
	imageDirectory	
	dts	
	to	
intrinsics	A [1x11] vector describing the focal lengths and distortion of the camera/lens combination. Distortion coefficients are defined by [3] and defined in Section 6.	
extrinsics	A [Tx6] vector describing the camera pose and position in WORLD coordinates (whatever coordinate system GCPs were entered in). T is the number of frames extrinsics are solved for. Extrinsic values are defined in Section 6 along with coordinate systems.	
imageNames	A Tx1 string where T is the number of frames extrinsics are solved for. Each string is the filename of the image solved for.	
t	A Tx1 vector of timing of each frame in datenum format. If t is not specified, vector is just numbers 1,2,3, etc.	

A figure plotting the tracked (yellow) and reprojected (red) SCP UVd coordinates using the new extrinsic solution. The figure will update for each image/frame solved for (Figure 8).

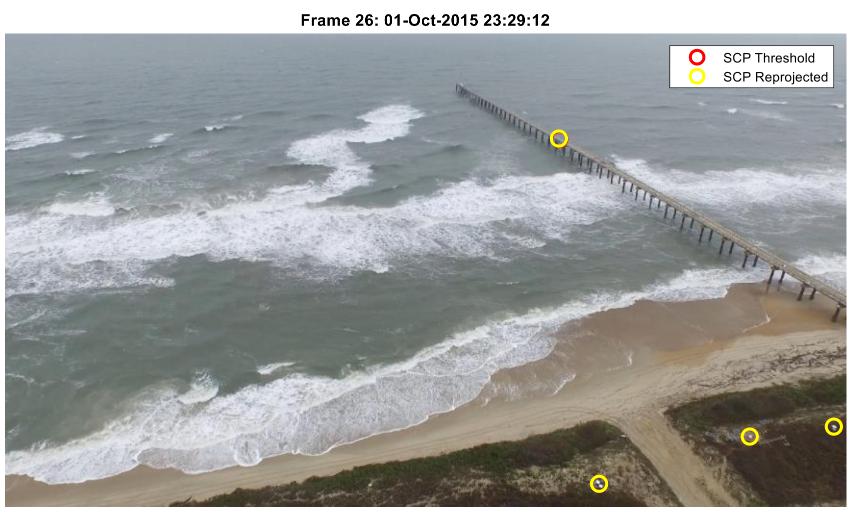


Figure 8: Figure in F_variableExtrinsicSolutions plotting tracked (yellow) and reprojected (red) SCPs for uasDemoData.

A figure plotting the extrinsics (relative to the first frame) as a function of time or index once the script is complete (Figure 9).

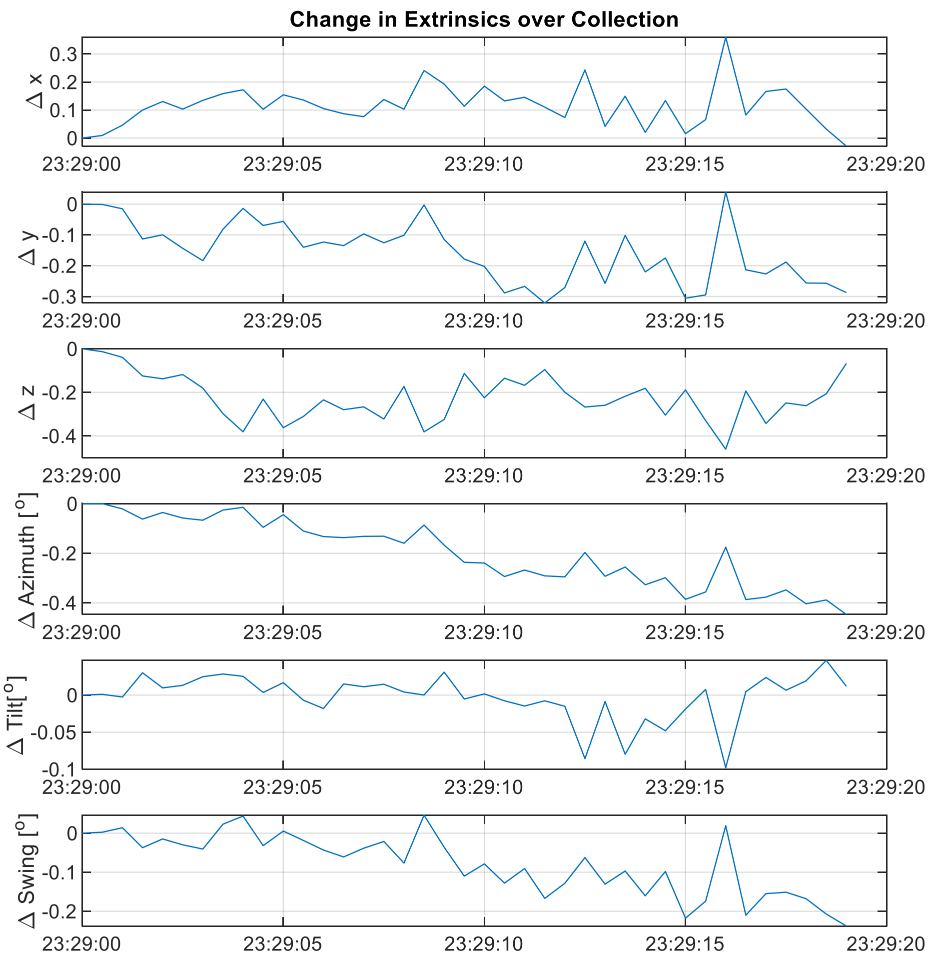


Figure 9: Figure produced by F_variableExtrinsicSolution for uasDemoData highlighting solved for variable extrinsics (relative to the first frame).

Required Core Sub-Functions:

thresholdCenter
extrinsicsSolver
xyz2DistUV
distUV2XYZ
intrinsicsExtrinsics2P
undistortUV
xyz2DistUV
distortUV

Description:

This function generates statistical image products for a given set of images and corresponding extrinsics/intrinsics. The statistical image products are the timex (average intensity), brightest (maximum intensity), variance (variance in intensity), and darkest (minimum intensity). If specified, rectified imagery for each image/frame can be produced and saved as an png file. Rectified images and image products can be produced in world or local coordinates if specified in the grid provided by D_gridGenExampleRect. This can function can be used for a collection with variable (UAS) and fixed intrinsics in addition to single/multi camera capability.

Note: Function can either have a temporally constant elevation grid with spatially varying Z or a spatially constant elevation grid with a temporally varying elevation value. The code needs to be modified to have both. If zVariable is non-empty, this will take precedent and make a spatially constant but temporally varying Z grid to rectify to

This function is to be run sixth or fifth in the progression for fixed and UAS collections.

The user input of the code is prescribed for UASDemoData. However, one can uncomment lines in Section 5 for the FixedMultiCamDemoData.

Input:

Input is entered by user into the script in Sections 1-5.

Section 1: Saving Information

Variable	*Description*
oname	Output string for the basename for the georectified images to be saved as.
odir	Output filepath where the SCP mat file will be saved.
outputFlag	Flag if you would like individual frames rectified and saved in the *odir*. 1= yes, output individual frames. 0= no, only output image products.

Section 2: Collection Information

Variable	*Description*
imageDirectory	The directory where the collection images are stored. Note, for UAS or collects with variable extrinsics solved in F, The images should be ordered in the same order listed in imageNames F_variableExtrinsicSolutions. For variable IOEO, the code will attempt to rectify all images in in the folder, if the number of images do not match it will crash. If the order does not match, they will be rectified incorrectly. For fixed IOEO, the code will rectify all images in the folder regardless with the same IOEO. Note for multi-camera collections, each camera will require a separate imagedirectory with the order of the images the same across all folders.

	It is up to the user to name files correctly so that images at the same index are simultaneous.	
ioeopath	Filepath of the saved CIRN IOEO calibration results produced by C_singleExtrinsicSolution or F_variableExtrinsicSolutions . If not produced by C or F, minimum variables are listed below.	
	intrinsics	A [1x11] vector describing the focal lengths and distortion of the camera/lens combination. Distortion coefficients are defined by [3] and defined in Section 6.
	extrinsics	A [Tx6] vector describing the camera pose and position in WORLD coordinates (whatever coordinate system GCPs were entered in). T is the number of frames extrinsics are solved for and number of images in imageDirectory. Extrinsic values are defined in Section 6 along with coordinate systems. The extrinsics should correspond to the images as ordered by matlab as was solved for in F.
	t	A Tx1 vector of timing of each frame in datenum format produced by F_variableExtrinsicsSolutions. This is not necessarily required. User can specify it manually in Section 4. If t is not specified or loaded in the ioeopath mat file, vector is just numbers 1,2,3, etc. T should be the number of images in imageDirectory

Section 3: Grid Information

gridpath	Filepath of the saved rectification grid created in D_gridGenExampleRect. Minimum required variables are listed below. Local values only required if localFlag==1	
	X Y Z	NxM grids in World coordinates where N is the number of rows (direction of varying Y) and M is the number of columns (direction of varying X). Grid needs to be in meshgrid format and be in same World coordinates as extrinsics in ioeopath
	localOrigin	Origin of new local coordinate system in world coordinates. should be in the same coordinate system of GCPs (worldCoord).
	localAngle	relative angle between the new (local) X axis and old (World) X axis, positive counter-clockwise from the old (world) X. Units are degrees.
	localX localY localZ	NxM grids in rotated Local coordinates (according to localAngle and localOrigin) where N is the number of rows (direction of varying Y) and M is the number of columns (direction of varying X).
	worldCoord	String of description of the World coordinate system for IOEO specified ioeopath.
localFlag	Enter if the user prefers local (localFlag==1) or world (localFlag==0) coordinates. Not if localFlag==1, localAngle, localOrigin, and localX,Y,Z in the ioeopath must all be non-empty.	

Section 4: Manual Entry of Time and Elevation

Variable	*Description*
t	A Tx1 vector of timing of each frame in datenum format. If t is not specified or loaded in the ioeopath mat file (as if produced from F), vector is just numbers 1,2,3, etc. T should be the number of images in imageDirectory.
zVariable	A Tx1 vector of elevations in world coordinates and units. f a Fixed station, most likely images will span times where Z is no longer constant. We have to account for this in our Z grid. To do this, enter a z vector below that is the same length as t. For each frame, the entire z grid will be assigned to this value. If UAS or not desired, leave empty. It is assumed elevation is constant during a short collect.

Section 5: Multi-Camera Demo

Uncomment this section for the multi-camera demo. ImageDirectory and ioeopath should be entered as cells, with each entry representing a different camera. It is up to the user that entries between the two variables correspond. Extrinsics between all cameras should be in the same World Coordinate System. Note that no new grid is specified, the cameras and images are all rectified to the same grid and time varying elevation. Also it is important to note for imageDirectory, each camera should have its own directory for images. The number of images in each directory should be the same (T) as well as ordered by MATLAB so images in the same order are simultaneous across cameras (i.e. the third image in c1 is taken at t=1s, the third image in c2 is taken at t=1s, etc).

Output:

A .mat file saved as *oname_gridmeta.mat* in *odir*. Will contain following variables; variables left blank are defined in the user input.

<table>
<tr><th>Variable</th><th colspan="2">Description</th></tr>
<tr><td rowspan="11">rectMeta</td><td colspan="2">Structure with metadata about output rectifications. Fields are listed below. Those left blank are the same entered in the input sections.</td></tr>
<tr><td>ioeopath</td><td></td></tr>
<tr><td>imageDirectory</td><td></td></tr>
<tr><td>gridPath</td><td></td></tr>
<tr><td>imageNames</td><td>Filenames of all images rectified.</td></tr>
<tr><td>t</td><td></td></tr>
<tr><td>zVariable</td><td></td></tr>
<tr><td>worldCoord</td><td></td></tr>
<tr><td>localFlag</td><td rowspan="3">All only if localFlag==1</td></tr>
<tr><td>localAngle</td></tr>
<tr><td>localOrigin</td></tr>
<tr><td>X</td><td colspan="2" rowspan="3">Flipped grids corresponding to rectified images. Before saved as an image, the image is flipped using flipUD so the image looks correct. (This is due to the image coordinate system, see section 6. Thus, so</td></tr>
<tr><td>Y</td></tr>
<tr><td>Z</td></tr>
</table>

	XYZ corresponds to the same pixels, we filp the grids as well for correct plotting. Values are local or world depending on localFlag.

Three pngs saved as oname_timex.png, oname_dark.png and oname_bright.png in odir. In addition, these are displayed in a Figures (Figure 10).

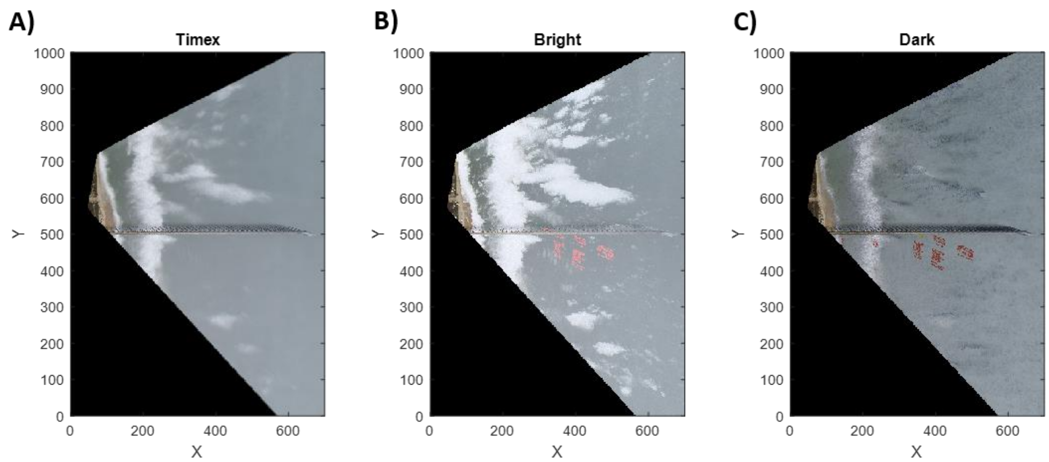


Figure 10: Example ensemble output of G1_imageProducts for uasDemoData in a local rotated coordinate system.

If outputFlag=1, T geo-rectified images in odir with the filename oname_(time or index).png. If time, time is referenced in epoch time in milliseconds. World or local coordinates is dependent on input localFlag value.

Required Core Sub-Functions:

imageRectification
cameraSeamBlend
xyz2DistUV
distortUV
intrinsicsExtrinsics2P
localTransformExtrinsics
localTransformPoints
plotRectification

Description:

This function generates pixel instruments for a given set of images and corresponding extrinsics/intrinsics. It works very similar to G1_imageProducts however instead of rectifying a uniform grid for rectified images, we rectify a set of points for bathymetric inversion, surface current, or run-up calculations. Rectifications can be made in both world and a local rotated coordinate system. However, for ease and accuracy, if planned to use in bathymetric inversion, surface current, or run-up applications, it should occur in local coordinates.

Note: Function can either have a temporally constant elevation grid with spatially varying Z or a spatially constant elevation grid with a temporally varying elevation value. The code needs to be modified to have both. If zFixedCam is non-empty, this will take precident and make a spatially constant but temporally varying Z grid to rectifiy to.

This function is to be run sixth or fifth in the progression for fixed and UAS collections.

The user input of the code is prescribed for UASDemoData. However, one can uncomment lines in Section 5 for the FixedMultiCamDemoData.

Input:

Input is entered by user into the script in Sections 1-5.

Section 1: Saving Information

Variable	*Description*
oname	Output string for the basename for the georectified images to be saved as.
odir	Output filepath where the SCP mat file will be saved.
outputFlag	Flag if you would like individual frames rectified and saved in the *odir*. 1= yes, output individual frames. 0= no, only output image products.

Section 2: Collection Information

Variable	*Description*
imageDirectory	The directory where the collection images are stored. Note, for UAS or collects with variable extrinsics solved in F, The images should be ordered in the same order listed in imageNames F_variableExtrinsicSolutions. For variable IOEO, the code will attempt to rectify all images in in the folder, if the number of images do not match it will crash. If the order does not match, they will be rectified incorrectly. For fixed IOEO, the code will rectify all images in the folder regardless with the same IOEO. Note for multi-camera collections, each camera will require a separate imagedirectory with the order of the images the same across all folders. It is up to the user to name files correctly so that images at the same index are simultaneous.

ioeopath	Filepath of the saved CIRN IOEO calibration results produced by C_singleExtrinsicSolution or F_variableExtrinsicSolutions . If not produced by C or F, minimum variables are listed below.	
	intrinsics	A [1x11] vector describing the focal lengths and distortion of the camera/lens combination. Distortion coefficients are defined by [3] and defined in Section 6.
	extrinsics	A [Tx6] vector describing the camera pose and position in WORLD coordinates (whatever coordinate system GCPs were entered in). T is the number of frames extrinsics are solved for and number of images in imageDirectory. Extrinsic values are defined in Section 6 along with coordinate systems. The extrinsics should correspond to the images as ordered by matlab as was solved for in F.
	t	A Tx1 vector of timing of each frame in datenum format produced by F_variableExtrinsicsSolutions. This is not necessarily required. User can specify it manually in Section 4. If t is not specified or loaded in the ioeopath mat file, vector is just numbers 1,2,3, etc. T should be the number of images in imageDirectory

Section 3: Manual Entry of Time and Elevation

Variable	*Description*
t	A Tx1 vector of timing of each frame in datenum format. If t is not specified or loaded in the ioeopath mat file (as if produced from F), vector is just numbers 1,2,3, etc. T should be the number of images in imageDirectory.
zFixedCam	A Tx1 vector of elevations in world coordinates and units. f a Fixed station, most likely images will span times where Z is no longer constant. We have to account for this in our Z grid. To do this, enter a z vector below that is the same length as t. For each frame, the entire z grid will be assigned to this value. If UAS or not desired, leave empty. It is assumed elevation is constant during a short collect.

Section 4: Instrument Information

Variable	*Description*	
gridpath	Filepath of the saved rectification grid created in D_gridGenExampleRect. Minimum required variables are listed below. Local values only required if localFlag==1. Grid world coordinates need to be same coordinates as those in the extrinsics in ieopath. Note, resolution of the grid is irrelevant for instruments you desire. THe grid is used for defining alocal coordinate system, and pulling z elevation values. THus, if you have a spatially variable Z grid, you may want grid dx,dy resolutions to be similar to your instruments.	
	X Y	NxM grids in World coordinates where N is the number of rows (direction of varying Y) and M is the number of

	Z	columns (direction of varying X). Grid needs to be in meshgrid format and be in same World coordinates as extrinsics in ioeopath
	localOrigin	Origin of new local coordinate system in world coordinates. should be in the same coordinate system of GCPs (worldCoord).
	localAngle	relative angle between the new (local) X axis and old (World) X axis, positive counter-clockwise from the old (world) X. Units are degrees.
	localX localY localZ	NxM grids in rotated Local coordinates (according to localAngle and localOrigin) where N is the number of rows (direction of varying Y) and M is the number of columns (direction of varying X).
	worldCoord	String of description of the World coordinate system for IOEO specified ioeopath.
pixInst	Stucture where each entry is a type of pixel instruments. Values with coordinate systems and units should match those specified in Section 2 and 4. If localFlag==1, the specified parameters should be in local coordinates. Field Entries are entered below depending on what is entered for the field 'type.'	
	type	Descriptor of instrument type. Can either be 'grid','xtransect', or 'ytransect.' For each type, input is defined below
	Grid- Good for bathymetric inversion algorithms and reduced resolution geo-rectified images	
	type	String of 'Grid'
	dx dy	Single Value of uniform grid resolution in world/local units
	xlim ylim	[1 x 2] vector of minimum grid extents in world/local units
	z	Elevation of instrument. Leave empty if you would like it interpolated from input Z grid or zFixedCam. If entered here it is assumed constant across domain and in time.
	yTransect- Good for alongshore current estimations or variations in time.	
	type	String of 'yTransect'
	x	Single value X coordinate of transect, for points along transect this value is constant.
	ylim	[1 x 2] vector of alongshore transect extents in world/local units
	dy	Single Value of uniform transect resolution in world/local units
	z	Elevation of instrument. Leave empty if you would like it interpolated from input Z grid or zFixedCam. If entered here it is assumed constant across transect and in time.
	xTransect- Good for observing run-up and other cross shore variations in time.	
	type	String of 'xTransect'
	y	Single value Y coordinate of transect, for points along transect this value is constant.
	xlim	[1 x 2] vector of crossshore transect extents in world/local units

	dz	Single Value of uniform transect resolution in world/local units
	z	Elevation of instrument. Leave empty if you would like it interpolated from input Z grid or zFixedCam. If entered here it is assumed constant across transect and in time.

Section 5: Multi-Camera Demo

Uncomment this section for the multi-camera demo. Impath and ioeopath should be entered as cells, with each entry representing a different camera. It is up to the user that entries between the two variables correspond. Extrinsics between all cameras should be in the same World Coordinate System. Note that no new grid or instrument file is specified, the cameras and images are all rectified to the same grid, instrument points, and time varying elevation. Also it is important to note for ImageDirectory, each camera should have its own directory for images. The number of images in each directory should be the same (T) as well as ordered by MATLAB so images in the same order are simultaneous across cameras (i.e. the third image in c1 is taken at t=1s, the third image in c2 is taken at t=1s, etc).

Output:

A .mat file saved as *oname_pixInst.mat* in *odir*. Will contain following variables; variables left blank are defined in the user input.

<table>
<tr><th>Variable</th><th colspan="4">Description</th></tr>
<tr><td rowspan="11">rectMeta</td><td colspan="4">Structure with metadata about the grid data, images, and coordinate systems the pixInstruments were derived from. Fields are listed below.</td></tr>
<tr><td>ioeopath</td><td colspan="3"></td></tr>
<tr><td>imageDirectory</td><td colspan="3"></td></tr>
<tr><td>gridPath</td><td colspan="3"></td></tr>
<tr><td>imageNames</td><td colspan="3">Filenames of all images rectified.</td></tr>
<tr><td>t</td><td colspan="3"></td></tr>
<tr><td>worldCoord</td><td colspan="3"></td></tr>
<tr><td>localFlag</td><td colspan="3" rowspan="3">All only if localFlag==1</td></tr>
<tr><td>localAngle</td></tr>
<tr><td>localOrigin</td></tr>
<tr><td colspan="4" style="display:none"></td></tr>
<tr><td>t</td><td colspan="4"></td></tr>
<tr><td rowspan="6">pixInst</td><td colspan="4">Stucture where each entry is a type of pixel instruments. Each ‘type’ will have the same fields but different vector/matrix sizes as indicated in each column. N is number of points in Y Direction, M is number of points in X direction (Number of points as specified by limits and resolution). Spatial values will have coordinate system and units as specified in gridPath and localFlag.</td></tr>
<tr><td>field</td><td>grid</td><td>yTransect</td><td>xTransect</td></tr>
<tr><td rowspan="2">type</td><td colspan="3">String of instrument description</td></tr>
<tr><td>‘grid’</td><td>‘yTransect’</td><td>‘xTransect’</td></tr>
<tr><td rowspan="2">dx</td><td colspan="3">X resolution</td></tr>
<tr><td>[1x1]</td><td>[]</td><td>[1x1]</td></tr>
</table>

	dy	Y resolution		
		[1x1]	[1x1]	[]
	xlim	X limits		
		[1x2]	[]	[1x2]
	ylim	Y limits		
		[1x2]	[1x2]	[]
	z	Input Elevation of Instrument		
	y	Input Y coordinate of Instrument		
		[]	[1x1]	[]
	x	Input X coordinate of Instrument		
		[]	[]	[1x1]
	X	Instrument X Coordinates		
		[NxM]	[Nx1]	[Mx1]
	Y	Instrument X Coordinates		
		[NxM]	[Nx1]	[Mx1]
	Z	Instrument Z Coordinates		
		[NxM]	[Nx1]	[Mx1]
	Irgb	Uint8 of RGB pixel intensity values for each point in instrument for each frame (T dimension)		
		[NxMx3xT]	[N xTx3]	[MxTx3]
	Igray	Uint8 of grayscale pixel intensity values for each point in instrument for each frame (T dimension)		
		[NxMxT]	[NxT]	[MxT]

A figure for each camera with pixel instruments reprojected onto the first oblique image used (Figure 11).

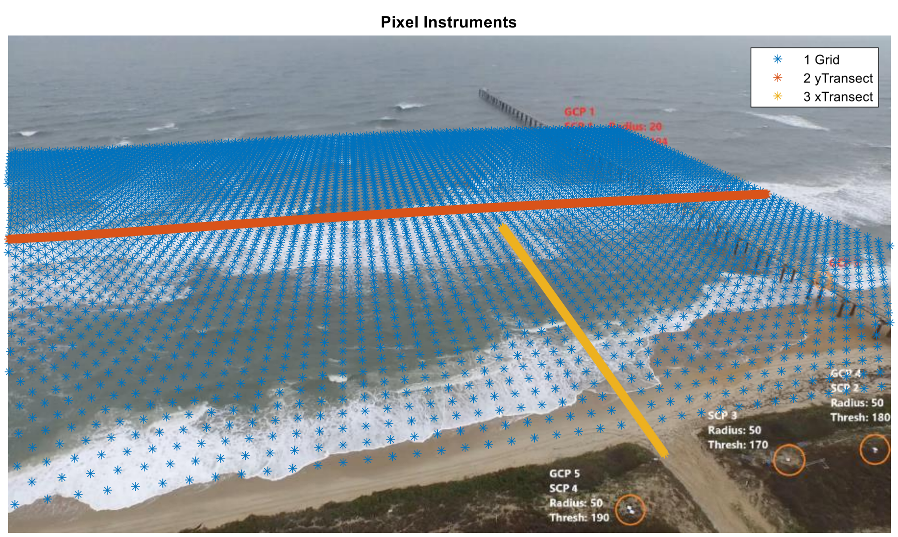


Figure 11: Example figure output for G2_pixelInstruments with instrument locations projected onto the first oblique image for uasDemoData

Figures plotting each pixel instrument Irgb value. Note, for grid instruments only the first snapshot in time is plotted (Figure 12).

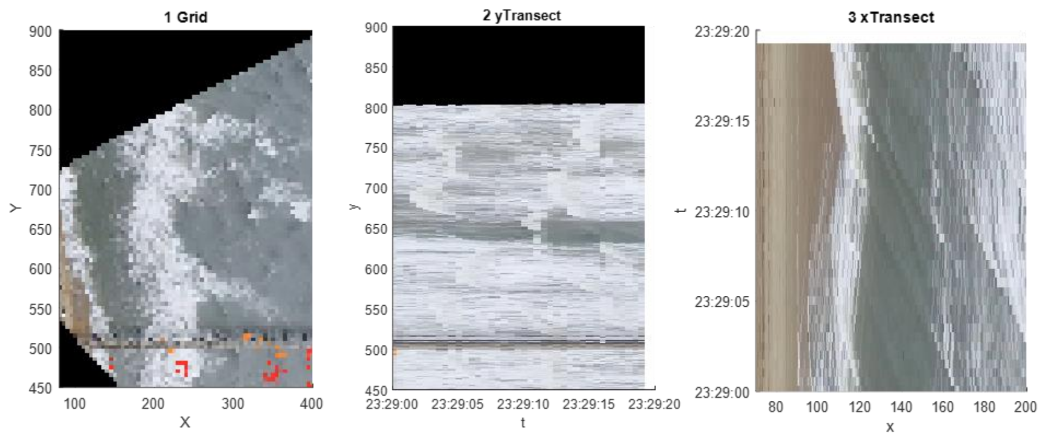


Figure 12: Example Figure output for G2_pixelInstruments for (left to right): Grid, yTransect, and xTransect for uasDemoData

Required Core Sub-Functions:

imageRectification
cameraSeamBlend
xyz2DistUV
distortUV
intrinsicsExtrinsics2P
localTransformExtrinsics
localTransformPoints
plotRectification
stackPlotter
pixInstPrepXY

4. Sub-Function Descriptions

localTransformPoints

Description:

This function tranforms between Local Coordinates and Geographical Coordinates. Local refers to the rotated coordinate system where X is positive offshore and y is oriented alongshore. The function can go from Local to World and in reverse. Note: Regardless of transformation direction, local Angle and localOrigin should stay the same. Input can be vectors or matrices. Note, for rotation to equidistant grids, localTransformEquiGrid should be used. Note, this only performs horizontal rotations/ transformations. More informations on coordinate systems is in Section 5.

Input:

Variable	*Size*	*Description*
localAngle	[1x1]	The localAngle should be the relative angle between the new (local) X axis and old (Geo) X axis, positive counter- clockwise from the old (Geo) X. Units are degrees.
localOrigin	[1x2]	Location of Local (0,0) in Geo Coordinates. Typically first entry is E and second is N coordinate.
Xin	Variable	Local (X) or Geo (E) coord depending on transformation direction. Units should be same as localOrigin.
Yin	Variable	Local (Y) or Geo (N) coord depending on transformation direction. Units should be same as localOrigin.
directionFlag	[1x1]	directionFlag = 1 or zero to indicate whether you are going from Geo ->Local (1) OR Local--> Geo (0)

Output:

Variable	*Size*	*Description*
Xout	Variable	Local (X) or Geo (E) coord depending on transformation direction
Yout	Variable	Local (Y) or Geo (N) coord depending on transformation direction

Required Core Sub-Functions:

None.

localTransformExtrinsics

Description:

This function tranforms between Local Coordinates and Geo Coordinates for the extrinsics vector. Local refers to the rotated coordinate system where X is positive offshore and y is oriented alongshore. The function can go from Local to Geo and in reverse. Note: Regardless of transformation direction, local Angle and localOrigin should stay the same. Can handle multiple extrinsics. Note, this only performs horizontal rotations/transformations. More information on Coordinate Systems is in Section 6.

Input:

Variable	*Size*	*Description*
localAngle	[1x1]	The localAngle should be the relative angle between the new (local) X axis and old (Geo) X axis, positive counter- clockwise from the old (World) X. Units are degrees.
localOrigin	[1x2]	Location of Local (0,0) in Geo Coordinates. Typically first entry is E and second is N coordinate.
extrinsicsIn	[Tx6]	Local (XY) or Geo (EN) coord depending on transformation direction. Should be Tx6 Matrix, T number of positions. Order should be [X Y Z Azimuth, Tilt, Roll] where A,T,R are in radians. Note units of localOrigin and XYZ should be the same.
directionFlag	[1x1]	directionFlag = 1 or zero to indicate whether you are going from Geo -->Local (1) OR Local--> Geo (0)

Output:

Variable	*Size*	*Description*
extrinsicsOut	[Tx6]	Local (XY) or world (EN) coord depending on transformation direction. Should be Nx6 Matrix, T number of positions. Order should be [X Y Z Azimuth, Tilt, Roll] where A,T,R are in radians.

Required Core Sub-Functions:

localTransformPoints

localTransformEquiGrid

Description:

This function tranforms between Local World Coordinates and Geo Coordinates for equidistant grids. However, we cannot just rotate our local grid. The resolution would no longer be constant. So we find the input limits, rotate them in to output coordinates, and then make an equidistant grid. Local refers to the rotated coordinate system where X is positive offshore and y is oriented alongshore. The function can go from Local to Geo and in reverse. Note, this only performs horizontal rotations/transformations. Function assumes transformed grid will have same square resolution as input grid. More information on Coordinate Systems is in Section 5.

Input:

Variable	*Size*	*Description*
localAngle	[1x1]	The localAngle should be the relative angle between the new (local) X axis and old (World) X axis, positive counter- clockwise from the old (Geo) X. Units are degrees.
localOrigin	[1x2]	Location of Local (0,0) in Geo Coordinates. Typically first entry is E and second is N coordinate.
Xin	[NxM]	Local (X) or Geo (E) Grid depending on transformation direction. Should be equidistant in both directions and a valid meshgrid. N is number of points in Y Direction, M is number of points in X direction
Yin	[NxM}	Local (Y) or Geo (N) Grid depending on transformation direction. Should be equidistant in both directions and a valid meshgrid. N is number of points in Y Direction, M is number of points in X direction
directionFlag	[1x1]	directionFlag = 1 or zero to indicate whether you are going from World-->Local (1) OR Local-->World (0)

Output:

Variable	*Size*	*Description*
Xin	Variable, similar to [NxM]	Local (X) or Geo (E) Grid depending on transformation direction. Should be equidistant in both directions and a valid meshgrid. N is number of points in Y Direction, M is number of points in X direction
Yin	Variable, similar to [NxM]	Local (Y) or Geo (N) Grid depending on transformation direction. Should be equidistant in both directions and a valid meshgrid. N is number of points in Y Direction, M is number of points in X direction

Required Core Sub-Functions:

localTransformPoints

Description:

This function computes the distorted UV coordinates (UVd) that correspond to a set of world xyz points for a given camera EO and IO specified by extrinsics and intrinsics respectively. Function also produces a flag variable to indicate if the UVd point is valid. See Section 6 for intrinsics and coordinate system definitions.

Input:

Variable	*Size*	*Description*
intrinsics	[1x11]	Intrinsics Vector Formatted as in A_formatIntrinsics.
extrinsics	[1x6]	1x6 Vector representing [x y z azimuth tilt swing] of the camera in world coordinates. Position values should be in the same units as xyz points to be converted and azimuth, tilt, and swing should be in radians.
xyz	[Px3]	Lx3 list of world coordinates of P points to be transformed to UVd coordinates. Columns represent X,Y, and Z coordinates.

Output:

Variable	*Size*	*Description*
UVd	[2*P x1]	Px1 list of distorted UVd coordinates for specified xyz world coordinates with 1:L being Ud and (P+1):2L being Vd coordinates. It is formatted as a 2Px1 vector so it can be used in an nlinfit solver in extrinsicsSolver.
flag	Px1	Px1 vector marking if the UVd coordinate is valid(1) or not(0)

Required Core Sub-Functions:

intrinsicsExtrinsics2P
distortUV

undistortUV

Description:

This function undistorts distorted UVd coordinates using distortion models from from the Caltech lens distortion manuals. This function is solving distortUV backwards essentially. However, if one looks at distortUV, it becomes very difficult to untangle all of the coefficients to solve for undistorted camera coordinates x and y. In fact, there is no analytical inverse distortion equation. So we solve for it iteratively. Aggregate Distortion coeffcients fr, dx, and dy should be solved for with undistorted camera coordinates x and y. However, we will solve for them using distorted xd and yd, and then use fr,dx, and dy to to solve for x and y. Then, the new x and y will be used to calculate fr, dx, and dy until the difference between subsequent fr,dx,and dy solutions are less than .001%. Then the final dx,dy, and fr will be used to solve for undistorted U,V. See Section 6 for intrinsics and coordinate system definitions. Notes on distortion model can be found in Section 5; function cannot handle fish-eye lenses.

Input:

Variable	*Size*	*Description*
intrinsics	[1x11]	Intrinsics Vector Formatted as in A_formatIntrinsics.
Ud	[Px1]	Distorted U coordinates for P points
Vd	[Px1]	Distorted V coordinates for P points

Output:

Variable	*Size*	*Description*
U	[Px1]	UnDistorted U coordinates for P points
V	[Px1]	UnDistorted V coordinates for P points

Required Core Sub-Functions:

none

rectificationPlotter

Description:

This function plots a rectified image in Matlab given a corresponding X and Y meshgrid in world coordinates on the current axes. The user can plot it as an RGB image using imagesc or a grayscale image using pcolor depending on the flag. Take note of how the matrices align for each, and how Matlab plots images versus matrices. When loading an image, and then using this function be sure to use the XYZ grid output by G1_imageProducts or flipud your grid. If using this to plot products directly from imageRectifier, use the same grid you input in imageRectifier.

Input:

Variable	*Size*	*Description*
Ir	NxMx3 uint8 or NxM	Rectified image with rgb or grayscale values. N is number of points in Y Direction, M is number of points in X direction
X	[NxM]	Meshgrid of X coordinates of NxM size
Y	[NxM]	Meshgrid of Y coordinates of NxM size
imageFlag	[1x1]	Flag of whether to plot as an rgb image (=1) or grayscale pcolor (=0). Note if input as grayscale, output will be grayscale regardless of this value.

Output:

Variable	*Size*	*Description*
-	-	Plotted rectified frame on current axes with axes corresponding to world coordinates (Figure 10, one of sub-figures)

Required Core Sub-Functions:

none

intrinsicsExtrinsicsToP

Description:

This function creates a camera P matrix from a specified camera extrinsics and intrinsics in addition to the comprising K, R, and IC matrices. See Section 5 for more information on matrix formulation. Note, output P is normalized for homogenous coordinates.

Input:

Variable	*Size*	*Description*
intrinsics	[1x11]	Intrinsics Vector Formatted as in A_formatIntrinsics.
extrinsics	[1x6]	1x6 Vector representing [x y z azimuth tilt swing] of the camera in world coordinates. Position values should be in the same units as xyz points to be converted and azimuth, tilt, and swing should be in radians.

Output:

Variable	*Size*	*Description*
P	[4x4]	Projection matrix to convert XYZ world coordinates to undistorted UV coordinates.
K	[3 x 3]	Matrix to convert XYZc (camera) Coordinates to undistorted UV coordinates
R	[3x3]	Rotation matrix to rotate XYZ world coordinates to Camera Coordinates XYZc
IC	[4 x 3]	Translation matrix to translate XYZ world coordinates to Camera Coordinates XYZc

Required Core Sub-Functions:

none

imageRectifier

Description:

This function performs image rectifications given the associated extrinsics, intrinsics, distorted image, and xyz points. The function utilizes xyz2DistUV to find corresponding UVd values to the input grid and pulls the rgb pixel intensity for each value. If the teachingMode flag is =1, the function will plot corresponding steps (xyz-->UVd transformation) as well as rectified output. If a multi-camera rectification is desired, I, intrinsics, and extrinsics can be input as cell values for each camera. The function will then call on cameraSeamBlend to merge the values.

Input:

Note k can be 1:K, where K is the number of cameras. Input order between I, Intrinsics, and Extrinsics must match.

<table>
<tr><th>Variable</th><th>Size</th><th>Description</th></tr>
<tr><td>I{k}</td><td>NNxMMx3</td><td>Distorted Image that pixel values will be taken from for rectification. Image should have been captured when entered intrinsics and extrinsics are valid. NN is the pixel height of the image and MM is the pixel width.</td></tr>
<tr><td>Intrinsics{k}</td><td>[1x11]</td><td>Intrinsics Vector Formatted as in A_formatIntrinsics.</td></tr>
<tr><td>Extrinsics{k}</td><td>[1x6]</td><td>1x6 Vector representing [x y z azimuth tilt swing] of the camera in world coordinates. Position values should be in the same units as xyz points to be converted and azimuth, tilt, and swing should be in radians. Extrinsics for all K cameras should be in same coordinate systems.</td></tr>
<tr><td>X</td><td rowspan="3">Variable, NxM or 1xP</td><td rowspan="3">Vector or Gird of X,Y, and Z world coordinates to rectify. Note this is not a cell entry. All K cameras will be rectified to the same grid.</td></tr>
<tr><td>Y</td></tr>
<tr><td>Z</td></tr>
<tr><td>teachingMode</td><td>[1x1]</td><td>Flag to indicate whether intermediate steps and output will be plotted (Figure 5).</td></tr>
</table>

Output:

Variable	*Size*	*Description*
Ir	Variable, NxMx3 or Px1x3	Image intensities for xyz points. Dimensions depend if input entered as a grid or vector. For both, an additional dimension with size 3 is added for r, g, and b intensities. Output will be a uint8 format.

Required Core Sub-Functions:

xyz2DistUV
intrinsicsExtrinsics2P
distortUV
rectificationPlotter
cameraSeamBlend

extrinsicsSolver

Description:

This function solves for a single camera geometry EO (extrinsics) and associated errors given: specified known values for extrinsics, initial guesses for unknown values of extrinsics, camera IO (intrinsics), world GCP coordinates (xyz), and corresponding UV coordinates of GCPs (UV). More information concerning coordinate systems is in Section 6.

Input:

Variable	*Size*	*Description*
intrinsics	[1x11]	Intrinsics Vector Formatted as in A_formatIntrinsics.
extrinsics	[1x6]	1x6 Vector representing [x y z azimuth tilt swing] of the camera in world coordinates. Position values should be in the same units as GCP xyz points to be converted and azimuth, tilt, and swing should be in radians. Include both known and initial guesses of unknown values.
xyz	[Px3]	Px3 list of world coordinates of P GCP points. Columns represent x,y, and z coordinates of GCPs. Rows are each GCP.
UV	[Px2]	List of image UV coordinates of P GCP points. Columns represent GCP U and V coordinates. Rows should correspond to same GCP point as xyz rows.
extrinsicsKnownFlags	[1 x 6]	Vector of 1s and 0s marking whether values in extrinsics are known (1) or are initial guesses (0) and should be solved for.

Output:

Variable	*Size*	*Description*
extrinsics	[1x6]	1x6 Vector representing [x y z azimuth tilt swing] of the camera in world coordinates. Position values should be in the same units as GCP xyz points to be converted and azimuth, tilt, and swing should be in radians. Include both known and initial guesses of unknown values.
extrinsicsUncert	[1x6]	A [1 x 6] vector with the uncertainty value of each solved extrinsic value provided by nlinfit. Units are in units entered in extrinsics for position and radians for pose.

Required Core Sub-Functions:

xyz2DistUV
intrinsicsExtrinsics2P
distortUV

Description:

This function computes the world XYZ coordinates that correspond to a set of distorted UVd image coordinates for a given camera extrinsics and intrinsics In order for UV to be solved, one dimension of xyz must be specified and provided for each point. Coordinate system information is in Section 6.

Input:

Variable	*Size*	*Description*
intrinsics	[1x11]	Intrinsics Vector Formatted as in A_formatIntrinsics.
extrinsics	[1x6]	1x6 Vector representing [x y z azimuth tilt swing] of the camera in world coordinates. Position values should be in the same units as xyz points to be converted and azimuth, tilt, and swing should be in radians.
UVd	[Px2]	List of distorted image UVd coordinates P points. Columns represent Ud and Vd coordinates.
knownDim	[1x1]	String of either 'x','y','z' specifying which dimension is known and provided by the user.
knownValu	[Px1]	Px1 vector of known world coordinates of UVd points specified. World dimension is that of knownDim and rows should correspond to samepoints as UVd.

Output:

Variable	*Size*	*Description*
xyz	[Px3]	Px3 list of world coordinates of P UVd points. Columns represent world x,y, and z coordinates. Rows correspond to UVd points.

Required Core Sub-Functions:

intrinsicsExtrinsics2P
undistortUV

distortUV

Description:

This function distorts undistorted UV coordinates using distortion models from from the Caltech lens distortion manuals. The function also suggests whether the UVd coordinate is valid (not having tangential distortion values bigger than what is at the corners and being within the image). Notes on distortion model can be found in Section 6; function cannot handle fish-eye lenses.

Input:

Variable	*Size*	*Description*
intrinsics	[1x11]	Intrinsics Vector Formatted as in A_formatIntrinsics.
U	[Px1]	UnDistorted U coordinates for P points
V	[Px1]	UnDistorted V coordinates for P points

Output:

Variable	*Size*	*Description*
Ud	[Px1]	Distorted U coordinates for P points
Vd	[Px1]	Distorted V coordinates for P points
flag	[Px1]	Vector marking if the UVd coordinate is valid(1) or not(0)

Required Core Sub-Functions:

none

CIRNangles2R

Description:

This function creates a Rotation matrix R that takes real world coordinates and converts them to camera coordinates (Not UV, but rather Xc,Yc, and Zc). The inputs are the pose angles as defined by CIRN, referenced and explained below. The camera axes are defined as Zc positive out of the lens, positive Yc pointing towards the top of the image plane, and positive Xc pointing from right to left if looking from behind the camera. The R is created from a ZXZ rotation of these angles in the order of azimuth, tilt, and swing. Diagrams of coordinate systems can be found in Section 5. All values should be in radians.

If angles are defined another way (omega,kappa,phi, etc) this function will have to be replaced or altered for a new R definition. Note, the R should be the same between angle definitions, it is the order of rotations and signage to achieve this R that differs.

Input:

Variable	*Size*	*Description*
azimuth	[1x1]	The horizontal direction the camera is pointing and positive CW from World Z Axis.
tilt	[1x1]	The up/down tilt of the camera. 0 is the camera looking nadir, +90 is the camera looking at the horizon right side up. 180 is looking up at the sky and so on.
swing	[1x1]	The side to side tilt of the camera. 0 degrees is a horizontal flat camera. Looking from behind the camera, CCW rotation of the camera would provide a positive swing.

Output:

Variable	*Size*	*Description*
R	[3x3]	Rotation matrix to transform World to Camera Coordinates

Required Core Sub-Functions:

none

cameraSeamBlend

Description:

This function takes rectified points from different cameras (but same grid or vector) and merges them together into a single rectification. To do this, the function performs a weighted average where pixels closest to the seams are not represented as strongly as those closest to the center of the camera rectification.

Input:

Variable	*Size*	*Description*
IrIndv	NxMxCxK	A NxMxCxK matrix where N and M are the grid lengths for the rectified image. N is number of points in Y Direction, M is number of points in X direction. C is the number of color channels (3 for rgb, 1 for bw) and K is the number of cameras. Even if using bw images, the matrix must be four dimensions with C=1; Each k entry is a rectified set if points for a camera. Input does not have to be uint8.

Output:

Variable	*Size*	*Description*
Ir	NxMzC	A uint8 matrix of the merged rectified image. N and M are the grid lengths for therectified image. N is number of points in Y Direction, M is number of points in X direction. C is the number of color channels (3 for rgb, 1 for bw).

Required Core Sub-Functions:

none

caltech2CIRN

Description:

This function converts the intrinsics calculated from the caltech toolbox to nomenclature congruent with the CRIN architecture. Note this only works with non-fisheye lenses and calibration models. More information on the distortion model is in Section 5.

Input:

Variable	*Size*	*Description*	
caltechpath	String	filepath of saved calibration results from Caltech Toolbox (calib_results). Minimum needed variables listed below.	
		nx	Number of pixel columns
		ny	Number of pixel rows
		cc(1)	U component of principal point, defined from top left corner of image
		cc(2)	V component of principal point, defined from top left corner of image
		fc(1)	U components of focal lengths (in pixels)
		fc(2)	V components of focal lengths (in pixels)
		kc(1)	Radial Distortion Coefficient
		kc(2)	Radial Distortion Coefficient
		kc(5)	Radial Distortion Coefficient
		kc(3)	Tangential Distortion Coefficient
		kc(4)	Tangential Distortion Coefficient

Output:

Variable	*Size*	*Description*
intrinsics	[1x11]	Intrinsics Vector

Required Core Sub-Functions:

none

stackPlotter

Description:

This function plots a timestack in matlab provided a pixel intensity matrix, a corresponding coordinate vector (X or Y) and a corresponding time vector t. If t is not specified {}, time will just be represented as an index, 1,2,3,4. The user can plot it as an RGB image using imagesc or a grayscale image using pcolor depending on the what I is entered. User can also specify if what type of transect it is, x or y. This only dictates whether time is represented on the vertical or horizontal axis.

Input:

Variable	*Size*	*Description*
Ir	[PxTxC]	RGB pixel intensities (C=3) or grayscale (C=1). P is number of points along transect. T is number of transects or points in time.
d	[1xP]	Vector of spatial coordinate that transect spans along (X or Y).
t	[1xP]	Vector of time in user desired units.
typ	string	String of 'x' or 'y' specifying if an cross-shore (x) or alongshore (y) transect. Dictates whether time is on the vertical (x) or horizontal (y) axis.

Output:

Variable	*Size*	*Description*
-	-	Plotted rectified frame on current axes with axes corresponding to world coordinates (Figure 12, last two sub-figures)

Required Core Sub-Functions:

none

pixInstPrepXYZ

Description:

This function constructs XYZ vectors and matrices for pixel instruments specfied in a pixInst structure. The vectors and matrices are of the correct size for the function imageRectifier. The format of the input structure is described below. Note: if the z coordinate is not known, pixInst.z can be left empty.

Input:

Variable	*Description*	
pixInst	Stucture where each entry is a type of pixel instruments. Field Entries are entered below depending on what is entered for the field 'type.'	
	type	Descriptor of instrument type. Can either be 'grid','xtransect', or 'ytransect.' For each type, input is defined below
	Grid- Good for bathymetric inversion algorithms and reduced resolution geo-rectified images	
	type	String of 'Grid'
	dx	Single Value of uniform grid resolution in world/local units
	dy	
	xlim	[1 x 2] vector of minimum grid extents in world/local units
	ylim	
	z	Elevation of instrument. Leave empty if not known. If entered here it is assumed constant across domain and in time.
	yTransect- Good for alongshore current estimations or other alongshore variations in time.	
	type	String of 'yTransect'
	x	Single value X coordinate of transect, for points along transect this value is constant.
	ylim	[1 x 2] vector of alongshore transect extents in world/local units
	dy	Single Value of uniform transect resolution in world/local units
	z	Elevation of instrument. Leave empty if not known. If entered here it is assumed constant across domain and in time
	xTransect- Good for observing run-up and other cross shore variations in time.	
	type	String of 'xTransect'
	y	Single value Y coordinate of transect, for points along transect this value is constant.
	xlim	[1 x 2] vector of crossshore transect extents in world/local units
	dz	Single Value of uniform transect resolution in world/local units

	z	Elevation of instrument. Leave empty if not known. If entered here it is assumed constant across domain and in time

Output:

<table>
<tr><td>Variable</td><td colspan="4">Description</td></tr>
<tr><td rowspan="26">pixInst</td><td colspan="4">Stucture where each entry is a type of pixel instruments. Each ‘type’ will have the same fields but different vector/matrix sizes as indicated in each column. N is number of points in Y Direction, M is number of points in X direction (Number of points as specified by limits and resolution).</td></tr>
<tr><td>field</td><td>grid</td><td>yTransect</td><td>xTransect</td></tr>
<tr><td rowspan="2">type</td><td colspan="3">String of instrument description</td></tr>
<tr><td>‘grid’</td><td>‘yTransect’</td><td>‘xTransect’</td></tr>
<tr><td rowspan="2">dx</td><td colspan="3">X resolution</td></tr>
<tr><td>[1x1]</td><td>[]</td><td>[1x1]</td></tr>
<tr><td rowspan="2">dy</td><td colspan="3">Y resolution</td></tr>
<tr><td>[1x1]</td><td>[1x1]</td><td>[]</td></tr>
<tr><td rowspan="2">xlim</td><td colspan="3">X limits</td></tr>
<tr><td>[1x2]</td><td>[]</td><td>[1x2]</td></tr>
<tr><td rowspan="2">ylim</td><td colspan="3">Y limits</td></tr>
<tr><td>[1x2]</td><td>[1x2]</td><td>[]</td></tr>
<tr><td rowspan="2">z</td><td colspan="3">Input Elevation of Instrument</td></tr>
<tr><td></td><td></td><td></td></tr>
<tr><td rowspan="2">y</td><td colspan="3">Input Y coordinate of Instrument</td></tr>
<tr><td>[]</td><td>[1x1]</td><td>[]</td></tr>
<tr><td rowspan="2">x</td><td colspan="3">Input X coordinate of Instrument</td></tr>
<tr><td>[]</td><td>[]</td><td>[1x1]</td></tr>
<tr><td rowspan="2">X</td><td colspan="3">Instrument X Coordinates</td></tr>
<tr><td>[NxM]</td><td>[Nx1]</td><td>[Mx1]</td></tr>
<tr><td rowspan="2">Y</td><td colspan="3">Instrument X Coordinates</td></tr>
<tr><td>[NxM]</td><td>[Nx1]</td><td>[Mx1]</td></tr>
<tr><td rowspan="2">Z</td><td colspan="3">Instrument Z Coordinates</td></tr>
<tr><td>[NxM]</td><td>[Nx1]</td><td>[Mx1]</td></tr>
<tr><td rowspan="2">Irgb</td><td colspan="3">Empty matrix of nans</td></tr>
<tr><td>[NxMx3x1]</td><td>[N x1x3]</td><td>[Mx1x3]</td></tr>
<tr><td></td><td rowspan="2">Igray</td><td colspan="3">Empty matrix of nans</td></tr>
<tr><td></td><td>[NxMx1]</td><td>[Nx1]</td><td>[Mx1]</td></tr>
</table>

Required Core Sub-Functions:

none

thresholdCenter

Description:

This function finds the center of area of pixels above a specified threshold in a region of interest (ROI) in a given image. This function is useful for finding the center of a bright area of pixels. However, function can be altered to find dark area of pixels by changing > to < in last two lines of code in Section 2. Accordingly, mentions of 'above' below should be understood as 'below' and vice versa.

Input:

Variable	*Size*	*Description*
I	[NNxMMx3]	Image where points of interest reside such as SCPs. NN is the pixel height of the image and MM is the pixel width.
Udo	[1x1]	Initial Ud center coordinate of Region of interest Value must be within size(2) of I. (<=MM)
Vdo	[1x1]	Initial Vd center coordinate of Region of interest Value must be within size(1) of I. (<=NN)
R	[1x1]	Radius for area of interest. Does not define circle, but rather ½ the length of a square. [Value should be in pixels and greater than 1.
Th	[1x1]	Threshold Lower Limit for selected bright pixel intensities. Is value that can be 0-255. If code is altered for dark pixels this is the upper limit.
brightFlag	[1x1]	String of either 'bright' or 'dark' to identify whether bright or dark SCPs will be identified. White objects on dark backgrounds should be 'bright' where dark objects on light backgrounds should be 'dark'.

Output:

Variable	*Size*	*Description*
Udn	[1x1]	Ud coordinate of New Center of Region of Interest considering only pixels above threshold
Vdn	[1x1]	Vd coordinate of New Center of Region of Interest considering only pixels above threshold
i	[nxm]	Subset of image I that is searched (ROI) as defied by R. n is the pixel height and m is the pixel length of the ROI. Values are binary for above (1) and below (0) threshold.
udi	[mx1]	The Ud coordinates of the subset I (Ud axes)
vdi	[1xn]	The Vd coordinates of the subset I (Vd axes)

Required Core Sub-Functions:

none

5. Equation and Coordinate System Definitions

Coordinate System Definitions

Note: World can be either coordinate system, local or geographical. However, in order to work, extrinsics and points in question need to be in same coordinate system. I.E. you cannot use the projective matrix on geographic points when extrinsic re defined in local and vice versa.

Nomenclature	Coordinate System
XYZ_C	Camera
xyz	Camera Homogeneous
XYZ_{WL}	World Local
XYZ_{WG}	World Geographical
XYZ_W	World: Either Local or Geographical
UV_D	Image Distorted (lens calibration applied)
UV	Image Undistorted (no lens correction; assumes pinhole camera model)

Local and Geographic World Coordinates

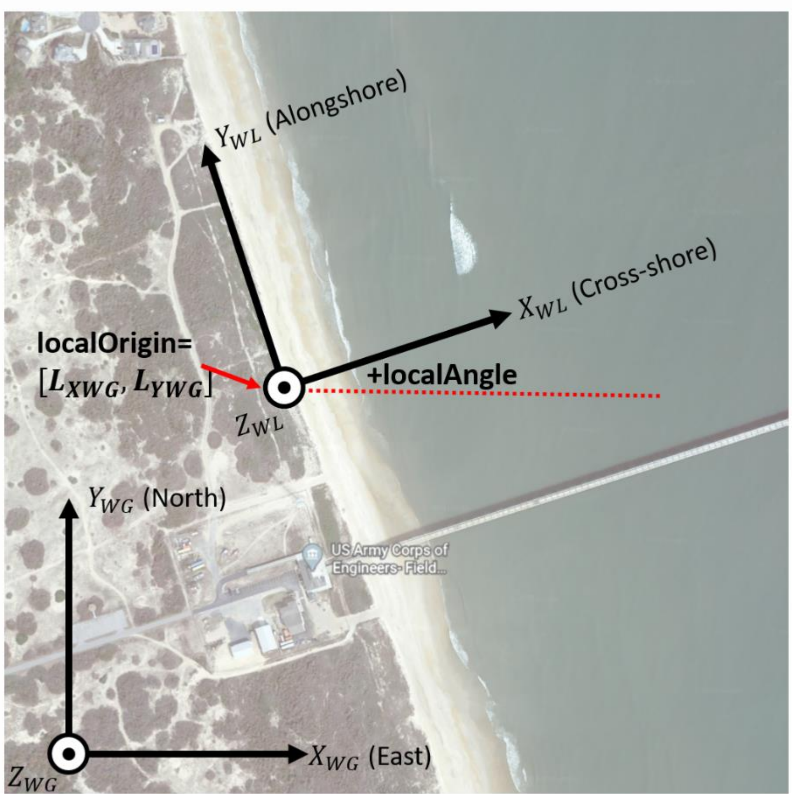


Figure 13: Relationship between World Coordinate Systems, Geographical and Local

Extrinsics

$$extrinsics = [\, C_{XW} \;\; C_{YW} \;\; C_{ZW} \;\; a \;\; t \;\; s]$$

where

C_{XW}	Position of Camera Coordinate System origin (Camera Position) in World coordinates.
C_{YW}	
C_{ZW}	
a	Azimuth: The horizontal direction the camera is pointing and positive CW from World Z Axis.
t	Tilt: The up/down tilt of the camera. 0 is the camera looking nadir, +90 is the camera looking at the horizon right side up. 180 is looking up at the sky and so on.
s	The side to side tilt of the camera. 0 degrees is a horizontal flat camera. Looking from behind the camera, CCW rotation of the camera would provide a positive swing.

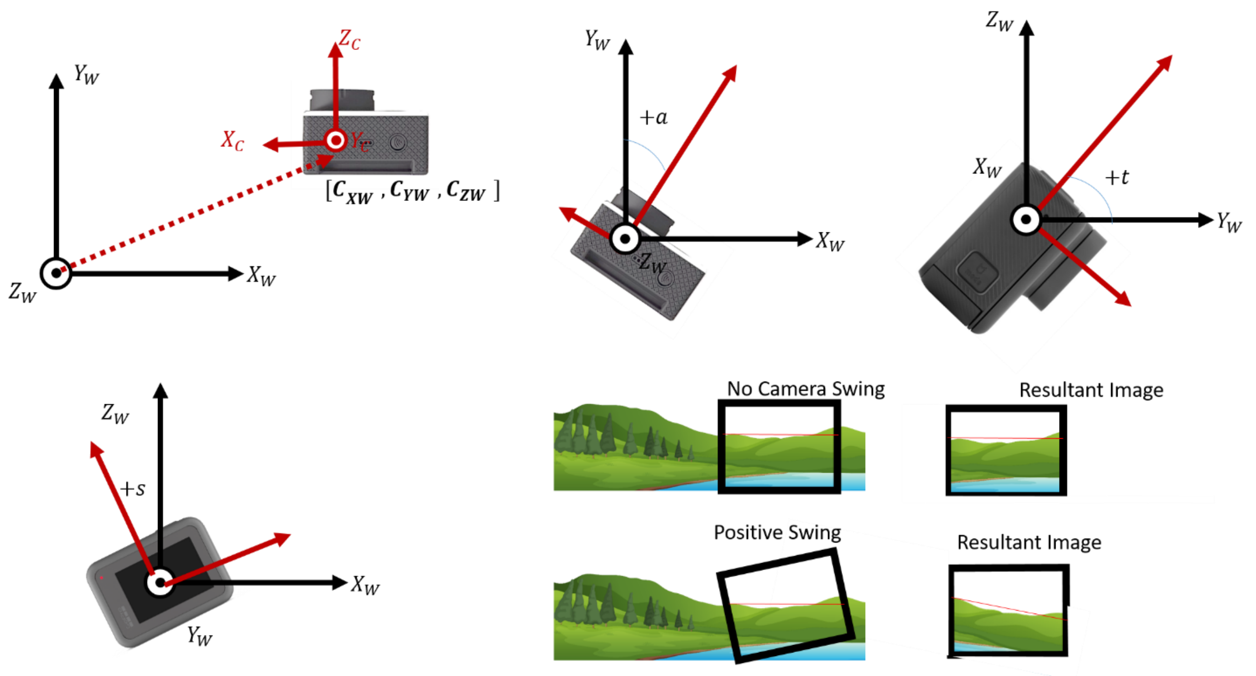


Figure 14: Definition of extrinsic vector in world coordinates.

Image and Camera Coordinates

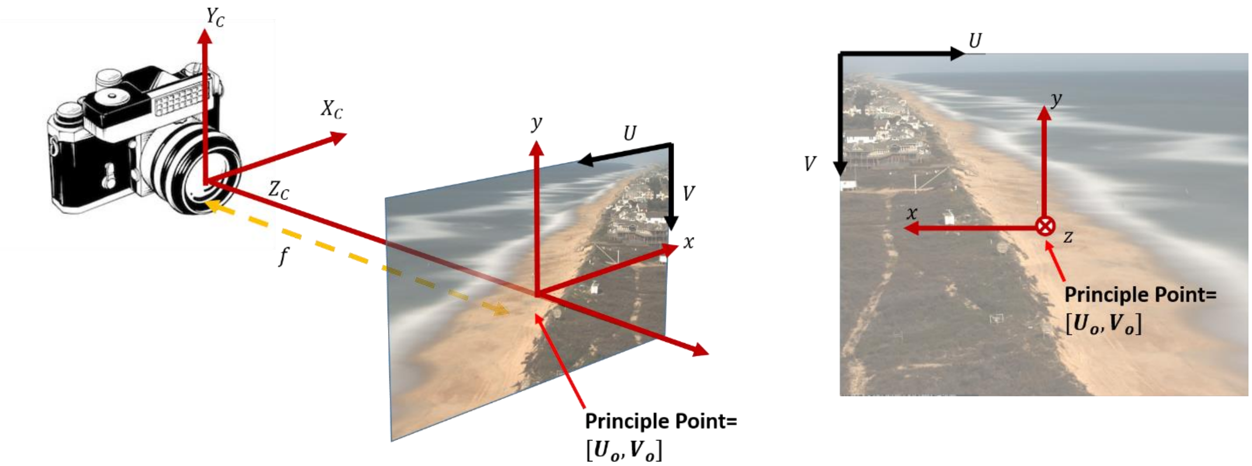


Figure 15: Relationship between Camera XYZ_C, Homogeneous Camera xyz, and undistorted Image (UV) coordinates.

Projection Matrix Construction

The projection matrix to go from XYZ_W to UV coordinates as defined above is:

$$\begin{bmatrix} U \\ V \\ 1 \end{bmatrix} = \overbrace{\underbrace{\begin{bmatrix} -f_x & 0 & U_o \\ 0 & -f_y & V_o \\ 0 & 0 & 1 \end{bmatrix}}_{K} \underbrace{\begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{22} & r_{33} \end{bmatrix}}_{R} \underbrace{\begin{bmatrix} 1 & 0 & 0 & -C_{XW} \\ 0 & 1 & 0 & -C_{YW} \\ 0 & 0 & 1 & -C_{ZW} \end{bmatrix}}_{T}}^{P} \begin{bmatrix} X_W \\ Y_W \\ Z_W \\ 1 \end{bmatrix}$$

Figure 16: P Matrix Construction to go from World to undistorted Camera Coordinates.

Where

K is matrix to go from camera coordinates to UV undistorted coordinates. (It combines the matrices to go from camera to camera homogeneous and camera homogeneous to Image).

R is rotation matrix to go from world coordinates to camera coordinates. It is a ZXZ rotation with azimuth, tilt, and swing respectively and is defined in the next section.

T is the translation matrix to go from world coordinates to camera coordinates

Rotation Matrix Definition

To construct R, we need to rotate the World Axes to match the Camera Axis in a ZXZ rotation. With azimuth, tilt, and swing (A,T,S) or (a,t,s). The camera axis is defined in Figures 14-15.

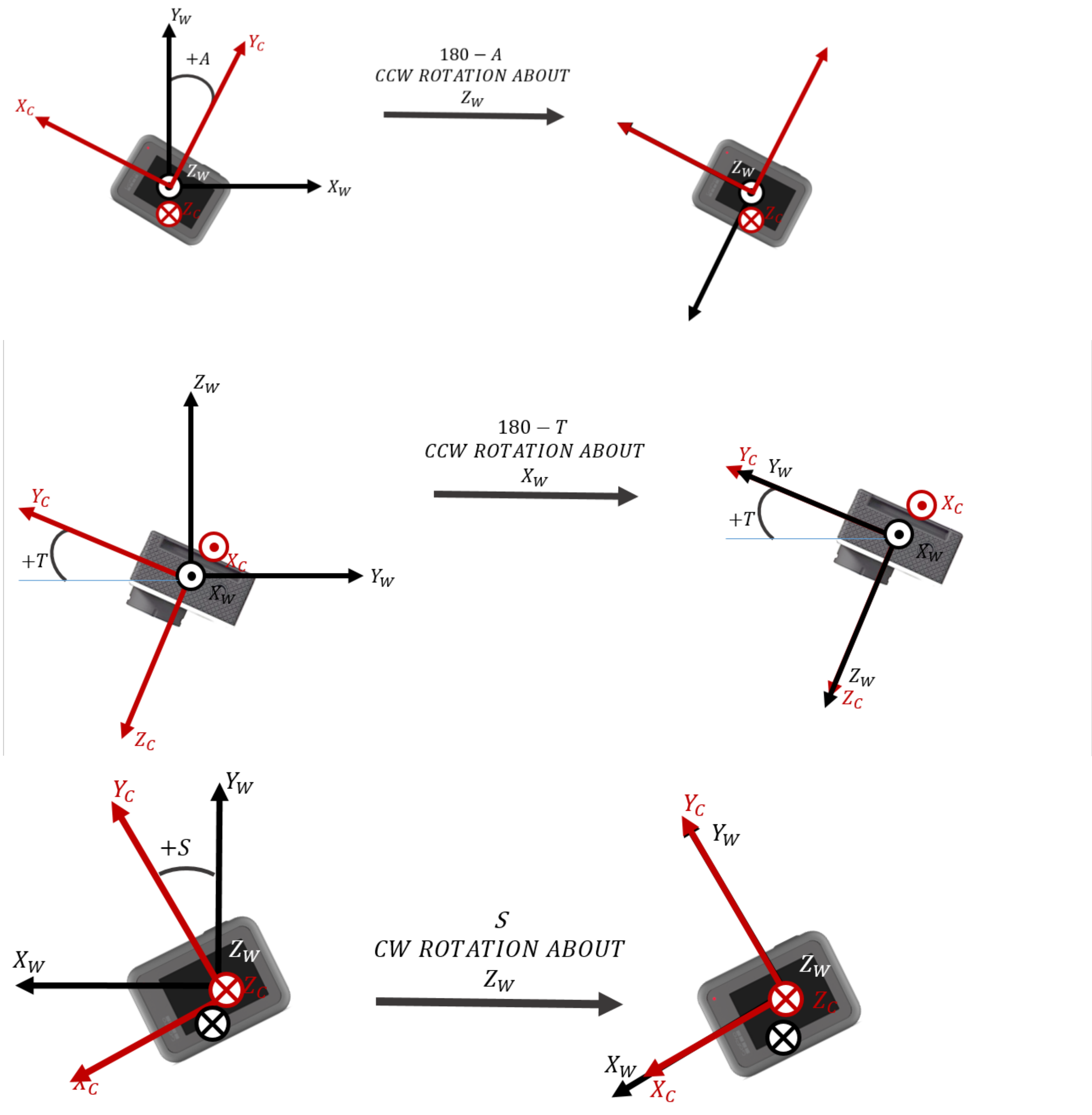


Figure 17: ZXZ Rotation formulation of World to Camera R Rotation Matrix

The top panel in Figure 17 shows the camera axes as if it were looking down (T=0) with a positive Azimuth A (Defined as positive CCW about X_C or in other words positive CW deviation of Y axes). The top panel shows the following rotation to get the World X axes to match the Camera X axes.

$$R_Z(A) = \begin{bmatrix} \cos(180-A) & \sin(180-A) & 0 \\ -\sin(180-A) & \cos(180-A) & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} -\cos(\mathrm{A}) & \sin(A) & 0 \\ -\sin(A) & -\cos(A) & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \text{CCW Rotation.}$$

Using fact that cos(180-X)=-cos(x) and sin(180-X)=sin(x).

Looking at the middle panel in Figure 17, Positive tilt is defined as clockwise rotation about Xc or CW Deviation of Yc from the horizontal. (+90 degrees would be the camera looking out to North if A=0). The middle panel shows the rotation to get the World Y and Z Axes to match the camera Axis Y and Z axes.

$$R_X(T) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(180-\mathrm{T}) & \sin(180-T) \\ 0 & -\sin(180-T) & \cos(180-T) \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -\cos(\mathrm{T}) & \sin(T) \\ 0 & -\sin(\mathrm{T}) & -\cos(T) \end{bmatrix} \quad \text{CCW Rotation.}$$

In the bottom panel in Figure 17 Positive Swing is defined as clockwise rotation about Zc. If you imagine that the camera is up looking at the horizon (Tilt 90); this would be if the camera was rotated counterclockwise looking from the back of the camera.

$$R_Z(S) = \begin{bmatrix} \cos(S) & -\sin(S) & 0 \\ \sin(S) & \cos(S) & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \text{CW Rotation.}$$

We then multiply the matrices. Remember, the order of rotation goes right to left!

$$R_{W\to C} = R_Z(S)R_X(T)R_Z(A)$$

$$R_{W\to C} = \begin{bmatrix} \cos(S) & -\sin(S) & 0 \\ \sin(S) & \cos(S) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & -\cos(\mathrm{T}) & \sin(T) \\ 0 & -\sin(\mathrm{T}) & -\cos(T) \end{bmatrix} \begin{bmatrix} -\cos(\mathrm{A}) & \sin(A) & 0 \\ -\sin(A) & -\cos(A) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$R_{W\to C} =$$
$$\begin{bmatrix} -\sin(\mathrm{A})\sin(\mathrm{S})\cos(\mathrm{T}) - \cos(S)\cos(A) & \cos(\mathrm{S})\,\mathrm{Sin}(\mathrm{A}) - \mathrm{Sins}(\mathrm{S})\mathrm{Cos}(\mathrm{T})\cos(\mathrm{A}) & -\sin(T)\sin(S) \\ \sin(\mathrm{A})\cos(\mathrm{T})\,\mathrm{Cos}(\mathrm{S}) - \cos(\mathrm{A})\sin(\mathrm{s}) & \sin(\mathrm{S})\,\mathrm{Sin}(\mathrm{A}) + \cos(\mathrm{A})\,\mathrm{Cos}(\mathrm{T})\cos(\mathrm{S}) & \cos(S)\sin(T) \\ \sin(A)\sin(T) & \sin(T)\cos(A) & -\cos(T) \end{bmatrix}$$

So this $R_{W\to C} = R$ takes world coordinates (XYZ, and transforms them to Camera Coordinates XcYc,Zc).

Intrinsics and Distortion

The intrinsic vector is defined as the following:

CIRN Intrinsics Variable	Caltech Variable	Symbol	Description
intrinsics(1)	nx		Number of pixel columns
intrinsics (2)	ny		Number of pixel rows
intrinsics (3)	cc(1)	U_o	U component of principal point, defined from top left corner of image
intrinsics (4)	cc(2)	V_o	V component of principal point, defined from top left corner of image
intrinsics (5)	fc(1)	f_x	U components of focal lengths (in pixels)
intrinsics (6)	fc(2)	f_y	V components of focal lengths (in pixels)
intrinsics (7)	kc(1)	d_1	Radial Distortion Coefficient
intrinsics (8)	kc(2)	d_2	Radial Distortion Coefficient
intrinsics (9)	kc(5)	d_3	Radial Distortion Coefficient
intrinsics (10)	kc(3)	t_1	Tangential Distortion Coefficient
intrinsics (11)	kc(4)	t_2	Tangential Distortion Coefficient

The equations for converting undistorted UV image coordinates to distorted coordinates is the following using symbols defined above. The model is defined in [3].

Normalize Distances in homogenous camera coordinates

$$x = (U - U_o)/f_x$$

$$y = (U - V_o)/f_y$$

Radial Distortion

$$r^2 = x^2 + y^2$$
$$f_r = 1 + d_1 r^2 + d_2 r^4 + d_2 r^6$$

Tangential Distortion

$$x' = 2t_1 xy + t_2(y^2 + 3x^2)$$
$$y' = 2t_2 xy + t_1(3y^2 + x^2)$$

Apply Distortion Correction

$$x_D = x f_r + x'$$

$$y_D = y f_r + y'$$

Convert Back to Image Coordinates

$$U_D = x_D f_x + U_o$$

$$V_D = y_D f_y + V_o$$

6. References

[1] R. A. Holman and J. Stanley, “The history and technical capabilities of Argus,” *Coast. Eng.*, vol. 54, no. 6, pp. 477–491, 2007, doi: http://dx.doi.org/10.1016/j.coastaleng.2007.01.003.

[2] K. T. Holland, R. A. Holman, T. C. Lippmann, J. Stanley, and N. Plant, “Practical use of video imagery in nearshore oceanographic field studies,” *IEEE J. Ocean. Eng.*, vol. 22, no. 1, pp. 81–91, 1997, doi: 10.1109/48.557542.

[3] J.-Y. Bouguet, “Camera calibration toolbox for Matlab (2008),” *URL http//www. vision. caltech. edu/bouguetj/calib_doc*, vol. 1080, 2008.