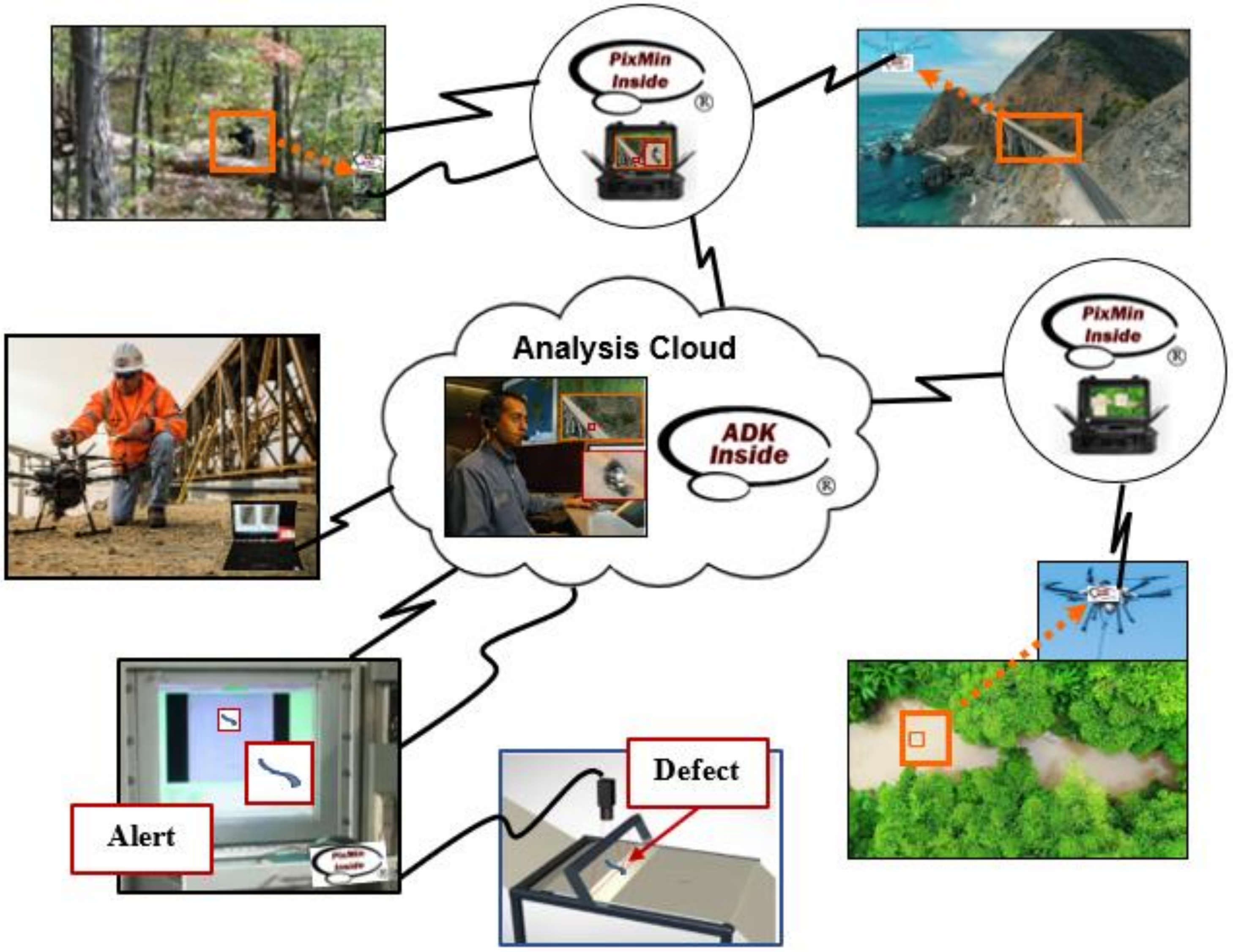
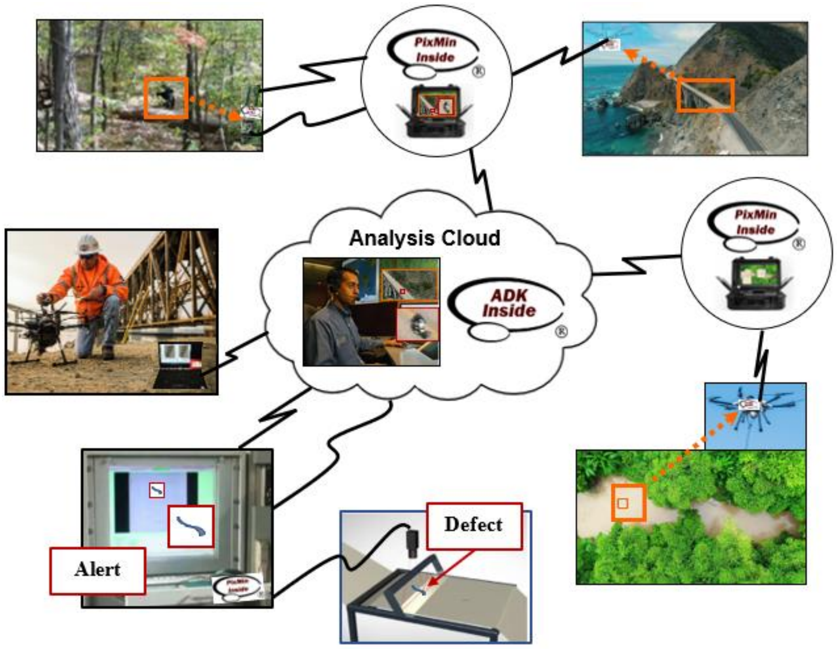
# PixMin ADK (Analyst Development Kit) Manual



## Contents

## Figures

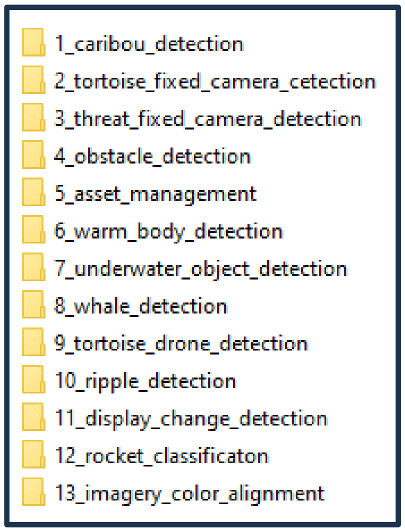
## 1. Introduction



PixMin™ processing *triages* imagery by detecting events within streams of sensor data—continuously, automatically, and at the sensor processing edge. By extracting only informative snippets from streamed imagery, PixMin reduces transmission bandwidth and analysis time by orders of magnitude. PixMin can process feeds from sensors above or on land as well as on or under water. Besides running at remote sensor sites, PixMin can run on cloud servers or at interim nodes, connected either via wired or wireless networks.

We call our process "PixMin" because it uncovers useful information by filtering out uninformative pixels. Reducing image pixels to essential information has become increasingly important, now that small cameras and other sensors can produce thousands of images affordably. Unattended cameras on drones and other platforms are now producing so many images that finding events within them manually takes too much time. Saving and transmitting all those images also takes too much storage and bandwidth. That's where PixMin comes in.

This Manual describes your PixMin analysis development kit (**ADK**). (You will find this and other bold font terms explained in our Appendix A Glossary.) The ADK will enable you to configure, run, and analyze **PixMin processor** results. Your PixMin ADK runs multiple image files as if they were run sequentially and in real-time by the PixMin processor.



1_caribou_detection
2_tortoise_fixed_camera_cetection
3_threat_fixed_camera_detection
4_obstacle_detection
5_asset_management
6_warm_body_detection
7_underwater_object_detection
8_whale_detection
9_tortoise_drone_detection
10_ripple_detection
11_display_change_detection
12_rocket_classificaton
13_imagery_color_alignment

The ADK will enable you to preconfigure the PixMin processor for a broad variety of automatic detection applications. You will also be able to reconfigure installed versions of PixMin remotely when field conditions or operational needs change.

Your ADK runs the **use-cases** within folders shown on the left. Each use-case folder includes input image files, configuration files, analysis files, and output files—all you will need to run each use-case on your own. Once you have read this Manual, you will be able to convert use-case configurations as you see fit to run PixMin with your own datasets as well.

Before reading this Manual further, please open your *PixMin_video.mp4* file located in your *0_ADK_video_files* folder and then watch the video.

You will find ADK contents, including, use-case folders, summary reports, our PixMin video presentation, descriptive slides, PixMin tool files, and a variety of image processing analysis folders, summarized in Appendix B and loaded on your PixMin Computer (**PixC**). You will also find Microsoft Office and other software preloaded on your PixC. As you will see, we have them to configure and analyze PixMin use-cases. Once you have read this Manual, you will be able to use PixMin as well as that other software.

The following Microsoft® Excel analogy may help explain the distinction between your PixMin ADK use and PixMin processor use.

Microsoft® provides Excel for analysis of spreadsheet data. Likewise, we provide the PixMin ADK for analysis of historical images. Excel may also run operationally for real-time data monitoring. Likewise, the PixMin processor runs operationally for real-time imagery monitoring. For example, commodities traders deploy Excel monitoring to receive stock price feeds, one line at a time. During each time slice, Excel examines each line item and triggers automatic alerts to highlight trading opportunities. Likewise, PixMin processes can continuously receive camera imagery, one image at a time. During each time slice PixMin examines each sector within an image and triggers automatic alerts to highlight events of interest.

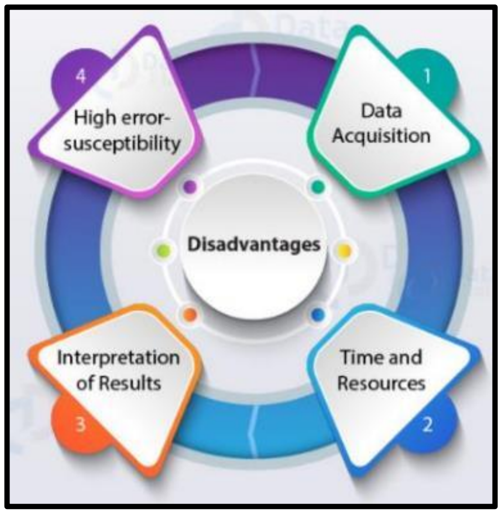
Before deploying Excel monitoring in this way, commodities traders configure and evaluate Excel monitoring models for eventual operational deployment. For example, trading analysts use Excel spreadsheets along with historical trading data for "paper trading" to determine how real-time Excel monitoring should be configured to trigger trading alerts effectively. Likewise, you will be able to use the PixMin ADK along with historical imagery to determine how the PixMin processor will trigger detection alerts operationally.

Analysts without programming language or machine learning (ML) backgrounds can readily use Excel spreadsheets for futures trading analysis and to meet many other needs. In the process, they refer as needed to Excel documentation. Likewise, analysts without programming language or ML backgrounds can readily use the PixMin ADK, referring as needed to our ADK documentation.

The term *triage* reflects what PixMin does. While medical triage identifies trauma victims with especially urgent needs, PixMin triage highlights regions within images that may contain especially important content. In evaluating a broad variety of image-based decision settings, we have found that trained observers must usually make final detection decisions instead of relying on fully automated methods to do so. For that reason, we have designed PixMin triage, like its medical counterpart, to help experts use their time effectively rather than replacing them completely.  We have designed PixMin to *triage* image data effectively so that higher level decisions (by analysts as well as computers) can be more precise and efficient.

We have designed the ADK and written this Manual for straightforward use and understanding, by those of you ranging from data analysis newcomers to image processing experts.  We encourage those of you newcomers to look check out background links that we have included in this Manual just for you.  You will find getting up to ADK speed to be straightforward.  We encourage those of you with advanced image processing backgrounds to compare PixMin with popular artificial intelligence and machine learning (**AI**/**ML**) alternatives.

As you will see, the use-cases and examples we have selected are well-suited for edge-based machine learning (EML) applications where PixMin works well.  In these applications, conventional AI/ML alternatives such as artificial neural networks  (ANNs) may not work as well because a) they typically require large training samples that contain many target events in operational environments, and b) they may not be as easy to preconfigure or reconfigure under changing field conditions.  We encourage you to help find out which machine learning alternatives work best for specific event detection applications.



While many articles cite **AI** promise for sensor processing at the sensory edge, conventional AI comes up short.  Effective edge-based AI must triage massive streams of sensor data (*e.g.*, camera imagery) into nuggets of useful information efficiently, adaptively, and deftly, in settings having low available size, weight, and power (SWaP), along with limited bandwidth.  Instead, conventional AI machine learning (CML) requires cloud-based computing.  CML also relies on expensive, time-consuming analytics that requires large training datasets.  As a result, frequent CML re-training under varying field conditions may not be feasible.  In addition to precluding these four disadvantages, PixMin processing runs quickly while meeting low SWaP requirements.

Since automated event detection may not be simple, we recommend weighing expected automatic detection benefits against delivery effort and cost.  Once you have read this Manual and used the ADK with your own datasets, you will be prepared to evaluate automatic detection return-on-investment (ROI) potential on your own.

We have set referenced terms in distinct fonts throughout this Manual.  You will find key terms, covered in the Manual Glossary (Appendix A), set in **bold** font.  You will find PixMin **configuration metric** names set in ***bold Italics*** font.  You will find folder names, folder paths and file names set in *Italics font*.

In the following sections, we will cover PixMin configurations ranging from simple to detailed. In some straightforward use-cases such as **river debris detection**, cameras may be rigidly pointed to regions of interest, enabling simple changes to be detected readily. In other applications such as **whale detection** from airborne imagery, target events are hard to see because they either show up rarely or they show up barely, or both. Even in the most detailed use-cases, however, you will find that each PixMin configuration has a visually clear and mathematically simple basis. Instead of being based on hidden layer configurations, which CML routinely employs, PixMin configurations use visually clear and mathematically simple **templates**. PixMin also looks for events within images based on mathematically simple **criterion** values, which evaluate how closely templates match **regions of interest** within images. In this Manual, we will fully explain how and why PixMin uses these configurations to detect anomalies and events of interest. You will be able to see exactly why and how PixMin produced **precise** or imprecise results by using analytics described in this Manual. By contrast, you may not be able to see why CML models produce imprecise results, due to CML's underlying hidden layer nature that may only be understandable by highly trained specialists.

You will find your **PixC** ADK organized into three essential folders. The *1_ADK_Manual_files* folder contains this Manual along with figures that it cites and use-case reports that it cites. The *2_use-case_files* folder contains one folder for each use-case. You will find each of these use-cases covered in this Manual. The *3_configuration_and_analysis_files* folder contains configuration templates and analysis files. You will also find each of these files covered in this Manual along with *0_ADC_videos* folder, which contains *PixMin_video.mp4* file. If you haven't yet opened that file and watched the video, reading the rest of this Manual will be easier if you do so at this point.

Along with these ADK folders, you will find a shortcut on your PixC desktop called *PixMin*, which is illustrated by our PixMin icon. When you click the *PixMin* icon a dialog box will open, prompting you to search for and then click a specific use-case **config** file. Once you click that config file, PixMin will run the use-case based on the config file contents. You will find details explaining how PixMin runs each use-case throughout this Manual.

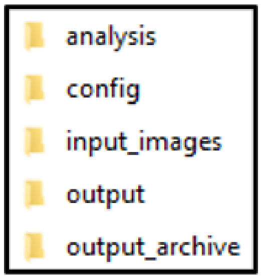
We have organized this Manual into two main sections. Section 2 shows how PixMin configurations produced ADK use-case results, without explaining how or why we configured PixMin to produce them. We have included details in Section 2 that will enable you to run all use-case examples on your own, modify the use-case configurations if you wish to do so, and see results that your modifications produce. Section 3 explains how and why we configured each PixMin use-case. After reading Section 3, you will be able to modify the use-case files to produce your own alternative PixMin configurations, based on a deeper understanding of how automatic event detection works. You will also be able to see how your alternative results compare. You will also be able to configure your own PixMin models to produce effective results with your own imagery, independently and on your own.

Sections 2 and 3 reference three Manual Appendices. Appendix A describes each **bold font** term in this Manual. Appendix B covers your **PixC** organization including all ADK folder contents and installed PixC applications. Appendix C explains each ADK **config** file metric.

In Sections 2 and 3, we will introduce PixMin elements sequentially as they figure into each use-case, which we have ordered from simple to detailed. That way, you will be able to see how configuration elements fit in by example and in context. In addition, for general reference and if you wish to jump ahead beyond example-based specifics, you will find each configuration element and technical term listed in our Appendix A Glossary. Each glossary term will point you to relevant specification details in Appendix B and Appendix C as well as relevant notes in Section 3.3.

## 2. Use case results

In this section, you will see how PixMin configurations produced results for each ADK use-case. If you open your ..\2*_use-case_files* folder, you will see that it contains one folder for each of our ADK 13 use-cases. Each of these folders has the same subfolder structure shown on the right. For each use-case, we will describe the contents of all but its analysis folder, which we will describe later in Section 3.


analysis
config
input_images
output
output_archive

For each use-case, files in the *config* folder specify how PixMin processes its input images. When you click the *PixMin_ADK\PixMin* icon, you will be prompted to find and open a **.pxm.csv* file in one of the use-case *config* folders. If you then click "run," PixMin will process the image files in the *input_images* folder and send results to the *output* folder. If you look in the *output_archive* folder for any of the use-cases, you will see output files we have created beforehand. We will be describing these archived files in this section and in Section 3.

We will begin describing each use-case by reviewing its corresponding report, which you will find in the ..*\Manual_cited_reports* folder.

### 2.1. Detecting caribou from aircraft imagery

For this use-case, we will first explain how we configured PixMin to detect caribou. We will then invite you to make your first PixMin run on your own. Along the way, we will introduce the role each config folder file plays and how selected **metrics** within each file fit in. Before reading this section, please check out the brief 2*_1_PixMin_caribou_detection.pdf* report. You will see that PixMin detected caribou within a high resolution image covering a large region.

Please open the PixMin ADK Explorer shortcut on your desktop and then navigate to your ..*\1_caribou_detection\input_images* folder. Within that folder, you will find a file named *caribou_subimage.png*. If you double click that image file, **IrfanView** will open and display the image. You will see several caribou in the image. If you hover your pointer over any **pixel** location, then click and hold, IrfanView will display your selected **pixel's** (*x*, *y*) coordinate at that point, or equivalently its (column, row) number. IrfanView will also display the pixel's three red, green, and blue (**RGB**) values at that pixel location. Each value is an integer between 0 for completely dark, to 255 for completely bright.

Next, on the IrfanView menu bar, please select Image – Convert to Grayscale. You will then see the IrfanView **grayscale** rendering of the image. If you then hover over any pixel location, then click and hold, you will see that all three RGB values are the same. IrfanView makes its grayscale conversion at each pixel location by replacing each of its three RGB values with the same value, which is a weighted sum of the three values. IrfanView does so by using a standard set of grayscale weights (see your **grayscale** Appendix A entry for details). PixMin enables you produce grayscale **color feature** values likewise, by entering those same standard weights in a *color_weights.csv* file.

PixMin can read pixel color values and convert them likewise to grayscale likewise. Please open the *color_weights.csv* file file for this use-case in its *..\config* folder. You will see the standard **grayscale** conversion weights in that file. By allowing you to enter these three values, PixMin enables you to input grayscale values or others that may highlight specific colors of interest. As you will when we describe other cases later, highlighting other colors can make PixMin detection more **precise**.

During **consecutive image processing**, PixMin processes one input image at a time. Before that, during **preliminary configuration**, PixMin verifies that each input image will be readable. First, PixMin verifies that each input image has the same number of rows and columns as those specified during configuration. If you right click your *caribou_subimage.png* file, then select Properties, and then select Details, you will see that the image has 920 rows and 1,642 columns. If you then open your *..\config\input_metrics.pxm.csv* file for this use-case, you will see the same number of rows and columns. If they were not the same, PixMin would display an error message and terminate. Each PixMin **input image** must also have the right color depth (as explained in your Appendix A **image file** entry). If not, PixMin will display an error message and terminate. If an input image file cannot be read for any other reason, PixMin will also display an error message and terminate.

We now introduce other **config** file **metrics** that apply to this us-case. If you want to learn about other metrics along the way, you will find brief descriptions of them in Appendix A and details in Appendix C. We share this caribou use-case first, because we have configured it to produce an alert within the input image simply. We have merely configured this case so that if a **grayscale** pixel value is sufficiently bright, that is, if a **color feature** value is high, then PixMin will produce an **alert**. Otherwise it will not. We have done so by setting the simplest **template** configuration among many possible PixMin configurations that could be used.

If you open the *..\template_values_level_1.csv* file for this use-case, you will see that PixMin can specify many Level 1 template types. (We will explain the Level 1 versus Level 2 distinction soon.) Each type may have a specified number of rows and columns. In this case, we specified one type with only one row and one column. We also set that corresponding template value to one. If you open the *..\template_versions_level_1.csv* file for this use-case, you will see that PixMin can specify many template versions for each type. Each version can have its own configured scale and rotation degrees value. In this case, we specified only one version for our single specified type, and we set its (scale, rotation degree) values to (1, 0). As a result, we configured PixMin with only one template type, template version **combination**.

PixMin generally identifies alerts within images by passing one or more **template combinations** over each pixel location within the image. At each pixel location, PixMin computes a **criterion** value that shows how closely the combination matches a region of interest (**ROI**) around the location. In this case, we have configured each pixel ROI as simply the pixel itself. We have also computed the criterion value as simply the color feature value itself. If you look again in your *..\input_metrics.pxm.csv* file for this use-case, you will see "Level 1 template matching basis" on line 13, column A followed by a value of 1 in column B. As explained in column C and further explained in Appendix C, this value has configured PixMin to compute **criterion** values for each pixel as **projection** values. In this case, since only one template has been configured with only one row and one column with a value of 1, the projection value that PixMin computes for each pixel is simply its color feature value.

PixMin always performs Level 1 triage and optionally performs Level 2 triage as well. During Level 1 triage, PixMin identifies image **alert blocks** that produce alerts. You will find an alert block size of 32 specified for the caribou use-case in line 12 of its *input_metrics.pxm.csv* file. Based that value for this use-case, PixMin looked for alerts within image alert blocks, which were 32 pixels by 32 pixels in size throughout the image (with minor exceptions—see Section 3.3, Note 1). PixMin identifies Level 1 alerts within each block based on two cutoff values: a **pixel match cutoff** value and an **alert block cutoff** value. If a template matching **criterion value** for a pixel within a Level 1 alert block exceeds the pixel match cutoff value, PixMin treats the pixel as a **pixel level alert**. If the resulting number of pixel level alerts within the block exceeds the alert block cutoff value, then PixMin marks the alert block as a **Level 1 alerted block**. In this case, we set the Level 1 pixel match cutoff value to 5.2, as shown in its *input_metrics.pxm.csv* file. (We will explain how and why we set that cutoff value, among others later in Section 3, where we cover configuration analysis.) Since we configured only one simple template type in this case, we produced a Level 1 alert for a pixel whenever its color feature value exceeded 5.2. In this case, we also set the Level 1 alert block cutoff value to 1. As a result, PixMin treated a Level 1 alert block as an alerted block if only one pixel within the block produced a pixel alert.

PixMin offers Level 2 triage as an option because it can add value to Level 1 triage for two reasons. First, Level 2 triage can further triage Level 1 triage by using larger and more diverse template combinations. PixMin can do Level 2 triage in this way without requiring much more computing time, because PixMin performs Level 2 triage within only Level 1 alerted blocks. Second, Level 2 triage results in **output centering** of alerted sub-images (which we call **chips**). We used Level 2 triage in this case for both reasons. We configured PixMin to perform Level 2 triage, as specified in line 17 of its *input_metrics.pxm.csv* file. During Level 2 triage, We set *template_values_level_1.csv* file settings and template and *template_versions_level_2.csv* file settings to the same values as their Level 1 counterparts. We also set Level 2 cutoffs to values shown in the *input_metrics.pxm.csv* file. As with the Level 1 cutoff settings, we will postpone discussing how and why we set those cutoff values until Section 3 of this Manual.

At this point, you can make your first PixMin run as follows.  First, if you double-click the *PixMin* icon just inside your *PixMin ADK* folder, a dialog box will open.  Next, if you click Open in the dialog box and navigate to *the ..\1_caribou_detection\config* file, you will see an *input_metrics.pxm.csv* filename.  Next, if you select the filename and click Open, you will see a dialog box with a Run button.  Next, if you click the Run button, PixMin will run the caribou use-case.  After a dialog box displaying  "Done" pops up,  if  you open your *..\caribou_detection*_output folder, you will see a *0_alerts* subfolder and an *alerts* subfolder.  The *0_alerts* folder would include copies of *input_image* files if they had not produced alerts.  In this case, you will find that the *0_alerts* folder is empty.  Next, if you open the *alerts* folder, you will see a *chips* sub-folder and a *full* sub-folder.  If you select the full subfolder and double click the *caribou_subimage.png* file, **IrfanView** will display the output **alert map** for this use-case.  You will see that each **chip** that PixMin detected is surrounded by a red box.  As you will see, PixMin detected nearly all caribou within the corresponding input image along with a few false detections.  If you then open the *chips* sub-folder, you will see one file for each **chip** that PixMin detected.  Each chip's filename ends with the row location, followed by the column location of the chip's center pixel within its image file.

Chip files enable PixMin analysts and operators to examine only alerted blocks in full resolution without having to look at other, uninformative image pixels. They also enable PixMin to produce and transmit triaged alerts with much lower transmission bandwidth.  In typical applications where informative events occur rarely, transmission bandwidth along with operator examination time can be reduced by a factor of 100K or more.  Alert map files give PixMin analysts and operators informative detection perspective.  In this case, for example, we can readily see that many false alerts occurred in a ravine where shiny rocks were prominent.

That concludes our first look at a PixMin use-case.  In the remainder of this Section 2, we will introduce the remaining use-cases in the same way, highlighting new configuration details as needs arise.  During each remaining use-case introduction, we will postpone describing analysis details until Section 3.  Along the way, if you wish to jump ahead to analysis details for this use-case in Section 3.1, feel free to do so.  Also, if you wish to modify configuration metrics along the way and then make your own PixMin runs, feel free to do so (but please first read Section 3.3, Note 2).

## 2.2.  Detecting tortoises from trail camera imagery

For this use-case, we will first explain how we configured PixMin to detect tortoises from trail camera imagery.  As before, we will then invite you to make a corresponding PixMin runs on your own.  Also as before, we will introduce configuration metrics that are new to you along the way.  Before reading this section, please check out the brief 2*_2_Wildlife_trail_camera_photo_triage.pdf*  report.  You will see that PixMin can reduce many trail camera snapshots to only those containing useful information, greatly reducing snapshot analysis time and required transmission bandwidth in the process.  In this case, the trail camera, facing the bottom of a culvert, was deployed to find endangered desert tortoise passing through.

Please navigate to your the *..\2_tortoise_fixed_camera_detection\input_images* folder. Within that folder, you will find 15 sequentially numbered **.png* files. Please open files and check them out using IrfanView. You will see that the first and last files among them show no animals. Like these first and last files, nearly all trail camera image files show nothing of interest. We configured PixMin to any target desert tortoise passing through without alerting any images that did not include a target.

Next, please open your *input_metrics.pxm.csv* file inside the *config* folder for this use-case. You will see that this configuration specifies **masking**. Briefly stated, masking enables PixMin to ignore regions within images that will contain only uninformative clutter by masking them out (see Appendix C, line 7 description for details). Briefly stated, differencing enables PixMin to detect changes between **color feature** values in each input image and their homologous color feature values in each corresponding *diff* folder image (see Appendix C, line 8 description for details). In this case, we configured masking so that PixMin would ignore metadata at the bottom of each input image. If you open any image in your *..\tortoise_detection\input_images* file you will see that the image has metadata on the bottom rows. If you then open your *..\tortoise_detection\config\default_mask.csv* file, you will see that all of its top rows have values of one except at the bottom rows, which have values of zero. PixMin uses that masking file to ignore the input image metadata accordingly. PixMin masking ignores clutter in other ways for other use-cases, as we will discuss later.

In the same input_metrics.csv file, you will also see that this configuration specifies **differencing**. Briefly stated, differencing enables PixMin to base detection on pixels within input images that have changed (see Appendix C, line 9 description for details). In this case, PixMin looked for changes in all input image pixels relative to their corresponding pixels in a *..\tortoise_detection\diff\default_diff.png* file, Since that image file contained no tortoise but otherwise looked the same, PixMin readily detected tortoise pixels within input images. If appropriate, we could also use differencing for this use-case **auto-adaptively** to make desert detection **robust**. PixMin differencing works in these other, more subtle ways for other use-cases as we will discuss later.

Other than specifying masking and differencing, our description for PixMin configuration in the previously described use-case applies to this use-case as well, with only two exceptions. First, we made the alert block size 512 for this use-case instead of 32 for the previous use-case so that output chips could cover all tortoise pixels. Second, we set template values to minus one instead of one so that PixMin would detect relatively dark pixels instead of relatively light pixels for the previous use-case.

At this point, you can make your next PixMin run as follows. First, if you double-click the *PixMin* icon just inside your *PixMin ADK* folder, a dialog box will open. Next, if you click Open in the dialog box and navigate to *the ..\2_tortoise_fixed_camera_detection\config* file, you will see an *input_metrics.pxm.csv* filename. Next, if you select the filename and click Open, you will see a dialog box with a Run button. Next, if you click the Run button, PixMin will run the use-case. After a dialog box displaying "Done" pops up, Please open your *..\output* folder for this use-case. You will see a *0_alerts* subfolder and an *alerts* subfolder. The *0_alerts* folder contains copies of *input_image* files that did not produce alerts. As you can see, we configured PixMin so that it would not produce an alert if an image did not include a full tortoise body. Next, if you open the *alerts* folder, you will see a *chips* sub-folder and a *full* sub-folder. If you select the full subfolder and check out its image files using **IrfanView**, you will see that PixMin produces an **alert map** for every image that contained a tortoise, just as we had configured PixMin to do. If you then open the *chips* sub-folder, you will see one file for each **chip** that PixMin detected. As before, each chip's filename ends with the row location, followed by the column location of the chip's center pixel within its image file.

That concludes our first look at your second use-case. As always, we will postpone describing analysis details until Section 3. If you wish to jump ahead to details for this use-case in Section 3.2, feel free to do so. If you wish to modify configuration metrics along the way and then make your own PixMin runs, feel free to do so (but please keep in mind Section 3.3, Note 2).

## 2.3. Detecting threats from fixed camera imagery

For this use-case, we will cover two examples. The first example shows how PixMin can detect traffic being monitored on an isolated bridge. The second example shows how PixMin can detect debris moving down a river. As always, we will then invite you to make a corresponding PixMin runs on your own and we will introduce configuration metrics that are new to you along the way. Before reading this section, please check out the brief 2*_3_PixMin_threat_detection_from_fixed_cameras.pdf* report. You will see that PixMin effectively can use **masking** as well as differencing to detect threats **precisely**, **robustly**, and **auto-adaptively**.

### 2.3.1. Bridge traffic detection

Please navigate to your *..\3_threat_fixed_camera_detection\3_1_bridge_traffic_detection\input_images* folder. Within that folder, you will find 12 sequentially numbered **.png* files. Please open files and check them out using IrfanView. You will see that the files show a motorcycle crossing a bridge. You will also see that images wobbled during the sequence. We set up these images (see Section 2.4.3) to include these wobbled images because fixed camera wobbling always occurs in practice. Camera wobbling often can't be seen, but it can often confound automatic detection unless dealt with properly as we did in this case. We did so by masking out all image pixels except those within a small region, as shown in the top figure within your 2*_3_PixMin_threat_detection_from_fixed_cameras.pdf* report. Next please open the *..\traffic\detection\config\mask\default_mask.csv* file for this use-case and the locate cell CNT1887. You will see that some cells in a region to the right of that point have values of 1 but cells outside that region have values of zero. Setting up the file values in that way resulted in the mask shown in the report.

Next, please run this use-case example like you ran the previous use-cases. Then, after you see the "Done" dialog box on your desktop, please open the *..\bridge_traffic_detection\output\alerts\full\video_029.png* file. You will see that PixMin detected the motorcycle within the input_images\*video 029.png* file, within the red **alert block** shown. If you open the *output\alerts\full\video 033.png* file, you will see that PixMin detected the motorcycle in next input image likewise. If, within either output image, you click and hold your mouse at the top left corner if its alert block and then drag your mouse to the bottom left corner as you keep holding, you will see at the top of your IrfanView window that the alert block size is 436. We chose that alert block size, as you will see in the *..\config\input_metrics.csv* file for this use-case, so that PixMin alerts could find the motorcycle, as well as larger vehicles, within it.

Based on our previous use-case discussions up until now, some other configuration metrics for this use-case should make sense at this point. However, please don't be concerned because many other metrics don't yet make any sense at all. We intend to go over all of them in detail as we introduce other use-cases later in this section and discuss them in detail later in the Manual.

### 2.3.2. River debris detection

Please navigate to your *..\3_threat_fixed_camera_detection\3_2_river_debris_detection\input_images* folder. Within that folder, you will find 17 **.png* files. Please open the files and check them out using IrfanView. You will see sequentially numbered images showing debris flowing down a river. As with our traffic detection example, we masked out all image pixels except those within a small region, as shown in the bottom figure within your 2*_3_PixMin_threat_detection_from_fixed_cameras.pdf* report. Just as in the traffic detection example we set values in the , *..\debris_detection\config\mask\default_mask.csv* file in keeping with that bottom figure in the report.

Next, please run this use-case example like you ran the previous use-cases. Then, after you see the "Done" dialog box on your desktop, please open the *..\river_debris_detection\output\alerts\full* folder and check out its contents using IrfanView. You will see that PixMin detected the large object, and only that large object when it passed through the unmasked region, just as we had configured PixMin to do so.

Our comment at the end of Section 2.3.1 applies here as well, just as it applies to all remaining Section 2 descriptions. We won't repeat the comment further, but please keep it in mind.

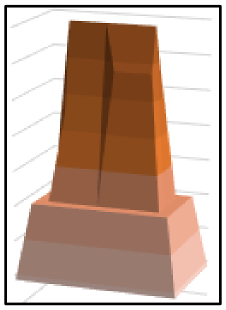
## 2.4. Detecting obstacles from driverless car imagery

First, please check out the brief *3_PixMin_threat_detection_from_fixed_cameras.pdf* report. Please keep that report open to follow our next paragraph discussion. Next, open 2*_4\obstacle_detection\input_images* folder. Within that folder, you will find 65 input image files. Next, open the last, *construction_google_car_frame_0068.jpg* file in that folder and keep that file open as well. Next, run this use-case just you have run previous use-cases. when you see the "Done" display, open the last **alert map** file in your *..\output\alerts\full* folder and keep that file open as well. Please note that your open input image file looks the same as the lower left image in the report and your open alert map file looks the same as the lower right image in the report.

We produced that alert map file, along with others in the *output\alerts\full* folder by using **masking**, construction cone-like **color feature weighting**, and a construction cone-like **template values**. If you open the *..\config\mask* folder for this use-case, you will see that we created one mask file for each input image (unlike previous examples where the *mask* folder contained a single mask file for all input images). The picture on right shows how PixMin used the last mask file to detect construction cones in the last *input_images* file. The gray values correspond to mask file values of zero and the other values, showing input image values and their alerts, correspond to mask file values of zero. (We will describe how we made that picture later in Section 3.2.4).

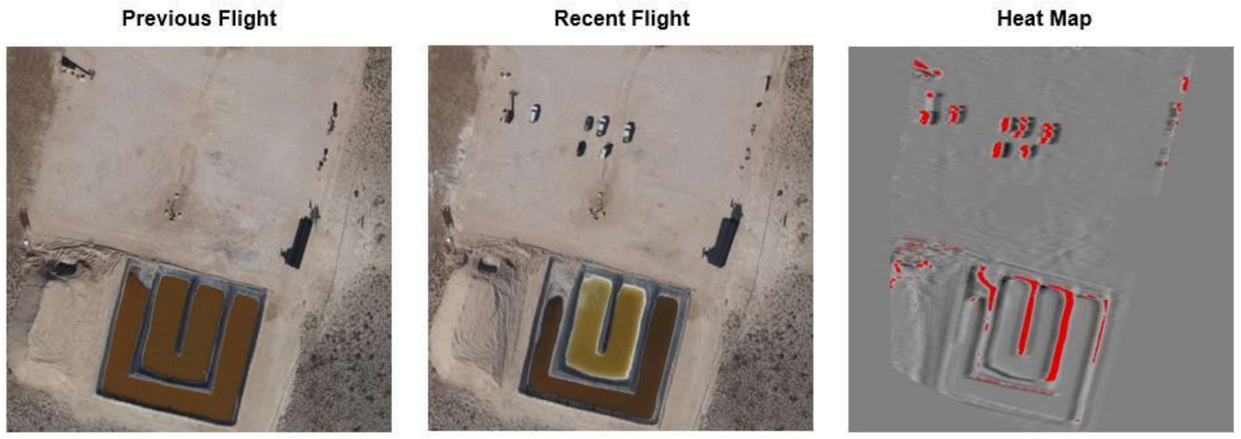
If you open the *color_weights.csv* file for this example, you will see that we did not use grayscale weighting as in our previous use-cases. Instead, we used a specialized set of **statistical contrast** weights. As described later (see Section 3.3, Note 3), contrast color weighting highlight pixels with positive weight values while suppressing pixels with other values. Since the contrast weights in the *color_weights.csv* file for this example had only a positive value corresponding to a red RGB color, they highlighted red, construction cone-like pixels in the image while suppressing non-red values.

If you open template*_values_level_1.csv* file for this use-case, then select the template values, and then select Insert-Charts-All Charts-Surface, you will see that the template looks like the shape shown on the left. If you then select any cell outside the template, fill the cell with "=sum(B5:J18)" and then click the check mark (✓), you will see that the values sum to zero. That means our template for this example is a **statistical contrast template** (see Section 3.3, Note 4). By using this contrast along with a configured **projection** criterion, PixMin highlighted all template-shaped regions within input images while suppressing others. As a result, PixMin produced the **precise** cone detections that you will see in your *output* folder **alert maps.**



## 2.5. Detecting oil field changes from aircraft imagery

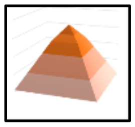
Please check out the brief 2_5_*PixMin_timely_asset_management.pdf* report. If you look at page 2 of the report, you will see how PixMin triaged gross changes within oil pads and then pinpointed changes within oil pads that showed gross changes. Next, please open your *..\use-case_files\asset_management* folder. Within that folder, you will see a *gross_oil_pad_change_detection* sub-folder and a *within_oil_pad_change_detection* sub-folder. Next, please run PixMin for the gross oil pad change detection use-case in the usual way. Once you see the Done display on your output, please look at the **alert map** for the use-case in its output\alerts\full folder. You will see that the alert map looks the same as Figure 1a) in the report, with eight alerted oil pad **chips**. Next, please look in the input_images folder within the *within_oil_pad_change_detection* sub-folder. You will then see in its ..*\input_images* folder that we supplied only eight input oil pad image files corresponding to those that produced gross oil pad alerts. Next, please run that use-case and then look at its eight output alert maps. Among them, you will see alert maps corresponding to those shown in Figure 1b) and Figure 1c). For each of these runs, we used **differencing** in ways that you will see in each of their *diff* folders. As always, we will supply more details in Section 3.5. You will see, among other things, how and why we used heat maps like the one shown in the right frame below, to detect oil pad changes.



## 2.6. Detecting warm bodies

Please browse the ..*\2_6\warm_body_event_detection.pdf* report and keep the report open. We produced the results shown on page 4 and page 6 of the report by first making a PixMin run with input thermal image files to detect thermal human targets. During that run, PixMin detected the thermal signature locations and wrote them to an *output\alerts\chips\chip_locations.csv* file, just as PixMin does routinely (see Section 3.3, Note 5). After that run, we placed that *chip_locations.csv* file into the **config** folder for this use-case. We then made a PixMin run with input **RGB** image files that were matched with their thermal counterparts. In that run, which we will ask you to repeat presently, PixMin used the ..*/config\chip_locations.csv* file locations to create RGB output alert maps and chips, just as we are about to ask you to do. Before doing so, please open the ..*\config\chip_locations.csv* file for this use-case. As you will see, each line in the file after the header line includes a filename, followed by a number that corresponds to the center of a detected chip, followed by a number that corresponds to the center column of the detected chip. Those line entries will determine the output of the PixMin run that you are about to make.

Next, please run PixMin for this use-case. Then, after the run has ended, look in the use-case *output\alerts\full* folder. You will see that one of the **alert maps** in that folder is the same as the right side alert map on page 4 of the report. Next, look in the use-case *output\alerts\chips* folder. You will see that some of the **chips** in that folder are the same as those shown in the report at the bottom of page 6. Later, in Section 3.6, we will explain how we first ran PixMin with thermal images and then ran PixMin just as you did to get these results. We will also explain how we determined color weights for this example and template values for this use-case that produced these results. For now, we will only direct you to the picture on the right below, which shows what the thermal *template_values_Level_1.csv* file values for this use-case looks like. Using analysis methods to be explained later, we found that this template distinctively

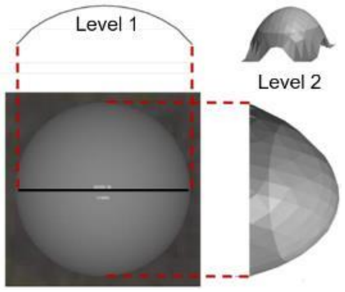


reflects the tops of human heads that were in the thermal images for this use-case. To see the resemblance, please compare the template shape on the right to the chip on the left, which shows what an input thermal image above a human shape looks like.

## 2.7. Detecting underwater objects from drone imagery

First, please browse your ..*\2_7\ PixMin_underwater_threat_detection.pdf* report and keep the report open. Next, please open your ..\ *7_underwater_object_detection* folder. You will see a *transact_search* subfolder with contents corresponding to Figures 2-3 in the report and a *descend* sub-folder corresponding to Figure 5 in the report. In this section, we will invite you to run PixMin within the *transact_search* folder, which we have preconfigured for you as with previous use-cases, as you can see. If you check out the *descend* sub-folder, you will see that we have not preconfigured it for you. We will invite you to preconfigure *descend* use-cases on your own and using your own analysis skills. after we have prepared you to do so in Section 3.

Next, please run PixMin for the ..*\transact_search* use-case. While PixMin is running, please open the use-case input_image folder and note that the image files in the folder correspond to the 63 images shown in the report. When the run completes, please note in the Done display that the run time was around two seconds per image. We wanted PixMin to run that quickly in the case, so that if we were running automatic detection on a drone, PixMin could keep with snapshats taken at that rate. We used PixMin differential **triage** functionality toward that end. We configured the Level 1 template to triage each of the image files quickly by making it small. We made the Level 2 template larger so it could validate the Level 1 detections effectively but also quickly because PixMin had detected only a small number of **alerted blocks** during Level 1 triage. If you open the two *template_values_level_*.csv* files for this use-case, you will see that the Level 1 template is much smaller than its Level 2 counterpart. If you graph. their contents, you that they look like the graphs on the right. We also used Level 1 and Level 2 **scale** functionality to make PixMin run much faster than it would have run otherwise, If you open the two *template_values_level_*.csv* files, you will see configured scale values that made PixMin run 10 times faster during Level 1 triage and 15 times faster during Level 2 triage (as explained in Section 3.3, Note 6). If look at the **alert maps** for this use-case, you will see that PixMin not only quickly but **precisely**. PixMin detected all five target mines with no false alerts.



## 2.8. Detecting whales from airborne imagery

First, please browse your ..*\2_8\ PixMin_underwater_threat_detection.pdf* report and keep the report open. Next, please open your ..*\8_whale_detection* use-case folder and begin a PixMin run for the use-case.   This run will take longer than others so far because the input image files for this use-case have higher **resolution** and because precise detection for this use-case required **correlation** criteria, which require more processing time.  (Our delivered product for this use-case took much less processing time, only about one second per image, because our operational process used **raw** image processing and  **multi-core** processing).

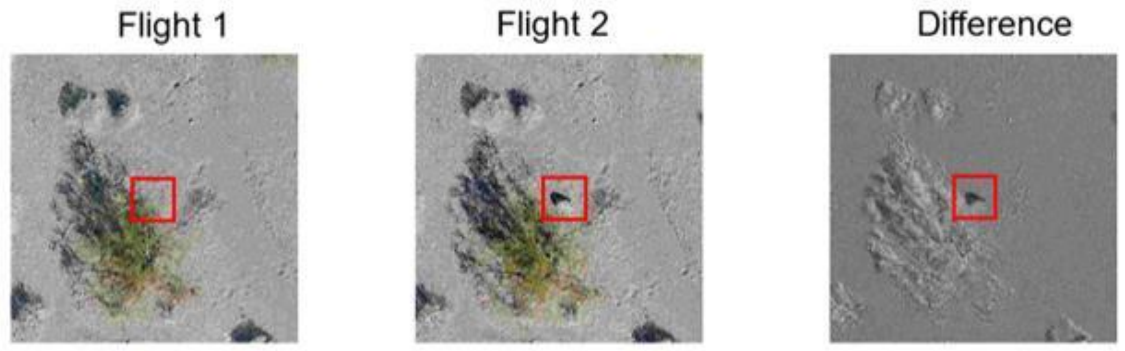
As with underwater object detection, we configured PixMin with to do fast, relatively coarse **triage** at Level 1,  followed by more granular Level 2 triage for this use-case.  If you open your *template_values_level_1.csv* file for this use-case, you will see that it has the same shape as shon the right.  We will cover what those templates look like and how we determined them in section 3.8.  If you open your *template_versions_level_*.csv* files, you will see that we configured scale values to produce faster processing as well,  In addition, you will see in the *template_versions_level_1.csv* file that we configured three rotation angles for the template.  As a result, PixMin was able to detect whales moving in different directions.

If you look in the input_images folder for this use-case, you will see three sets of image triplet files.  Each triplet within its set was captured sequentially.  The center image in the set contained a whale chip but the images that were captured before and after it did not.  If you look at the **alert maps** for this use-case, you will see that PixMin detected each chip containing whales and no others.  (In configuring PixMin operationally, we found that we could not reach the perfect precision shown for this use-case demonstration.  Instead, we configured PixMin to produce higher whale detection **sensitivity** (*i.e.*, higher whale detection rates).  Doing so resulted in lower sensitivity (*i.e.*, higher false detection rates), but we configured PixMin to present detections in a way that still added significant value—see *3_1_automatic_event_detection_value_determination.pdf*).

## 2.9.  Detecting desert tortoises from drone imagery

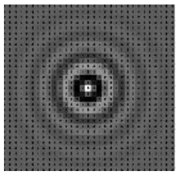
First, please browse your ..\2_9\ *PixMin_wildlife_drone_camera_triage.pdf* report and keep the report open. Next, navigate to the *..\9_tortoise_drone_detection\post-flight_detection\input_images\alerts\full* folder and then open the *ortho_02_day_2.png* file in the folder.  You will see that the file corresponds to frame c) on page 2 of the report.  If you zoom in to 50% using IrfanView and locate the region around (X,Y) = (1330,1130), you will see a desert tortoise.   If you will zoom pack out to 9.5%, you will see many rocks and bushes that look very much like the desert tortoise—so much alike that we had to configure PixMin with special attributes, which we will introduce presently. Before we do that, please run PixMin using the *input_metrics.csv* file in your *\post-flight_detection\config* folder.  Once your run has ended, please open the *ortho_02_day_2.png* file in the resulting *output\alerts\full* folder and then navigate to the same location where you saw the desert tortoise before.  You will see a red **alert map** box around the desert tortoise, just as intended.  Next, open the *output\alerts\chips* folder and then in Explorer, select View-Large icons.  Among the 19 **chip** files shown, you will desert tortoises in two of them.  These were the only desert tortoises within the entire survey described in the report. We configured PixMin in this case with sufficient sensitivity to detect both desert tortoises. In order ensure sufficient sensitivity, we allowed an acceptable number of false detections, as shown.

In order to achieve the necessary sensitivity to for desert tortoise in image files with so many tortoise-like objects, we had to configure **differencing** along with precise **pixel alignment**. To analyze the impact of differencing and pixel alignment options, we made several PixMin runs with neither option, with both options, and with only differencing. You will find related files in your *..\9_tortoise_drone_detection\pixel_alignment* folder. We will ask you to make the same runs and go over the results with you in Section 3.9. For now, please look at the picture below, which shows how differencing with pixel alignment performed. The frame on the left shows a chip covering a region that was photographed during the first drone flight. As you can see, that frame does not show a tortoise. The frame in the middle shows a chip covering the same region during the second drone flight. As you can see, that frame does show a tortoise. The frame on the right shows left frame pixels and middle frame pixels that were differenced after pixel alignment. As you can see, background pixel values do not stand out and tortoise body pixel values do not stand out, but the tortoise shadow pixel values do stand out. Background pixels don't stand out because they didn't change much (except for shadow and wind repositioning changes). Tortoise pixel values don't stand out because they look much like background pixels (because tortoise shells have evolved to look camouflaged in that way). Only the shadow pixels stood out as prominent difference values. We used shadow difference pixels accordingly to detect tortoise changes. If you look closely at the left and center frames below, you will also see that their pixels are not perfectly aligned. As we will demonstrate in Section 3.9, incorporating PixMin automatic pixel alignment was essential for effective background clutter differencing in this case.



## 2.10. Detecting ripples in water

First, please browse your *..\2_10\ PixMin_wildlife_drone_camera_triage.pdf* report and keep the report open. Next, run this use-case and look at the output **alert map**. You will see that it looks the same as Figure 2 in the report, except the alert map doesn't show the ripple that wasn't detected. Next, look at the output **chip** files. You will see that they look the same as those shown in Figure 4 of the report. (We will cover the report "discovery map" figure in Section 2.11.) As you will see by checking out the two *template_values_level_*.csv* files, PixMin detected ripples with the simple, one-dimensional template graphed on the right. We could have configured two-dimensional templates like the one shown on the left. We did not do



that because it would have slowed down processing speed more than would have been justified for this application. We could have also configured more **sensitive criteria** such as **correlation coefficients**, but we did not do that for the same reason. (To demonstrate resulting speed versus precision tradeoffs, we will invite you to try these alternative configurations as an exercise in Section 3.10.) Notably, even this simple one-dimensional template, which we configured for a fixed frequency, phase, and amplitude, detected seven out of eight ripples with different phases, frequencies, and amplitudes.

## 2.11. Detecting events from display imagery

Please browse your *..\2_11\PixMin_anomaly_detection_from_displays.pdf* report and then run the corresponding use-case. At this stage, you should be able to understand how the corresponding PixMin configuration produced the *output* folder results.

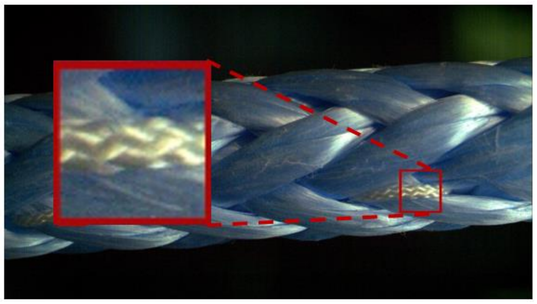
## 2.12. Classifying objects with template matching

If you read your *..\2_12\PixMin_rocket_classification.pdf* report, you will see that this use-case is distinct from those described earlier because we used PixMin to classify events in addition to detecting them. We used PixMin analytics to do so in ways that we will explain in Section 3.12. For now, please note in your *config* folder for this use-case that we created several *template_values_*.csv* files and several *template_versions_*.csv* files. We made distinct PixMin runs based on different combinations of the files to produce output analytics for each combination that we were able to tabulate, as shown in Figure 2 of the report. When used to produce results like these operationally, PixMin will run through all template combinations routinely and find the best fitting combination automatically, resulting in an automatic classification decision for each input image. PixMin will be able to to do so for regions of interest within images as well.

## 2.13. Correcting images for color variability

If you read your *..\2_13\PixMin_image_color_variability_correction.pdf* report and then run this use-case, you will see that the *output* folder images look the same as their counterparts in the report. This use-case is distinct from those described earlier because we used PixMin to color-correct input image files without either detecting or classifying events. When used operationally, PixMin will color-correct each input image as a first step before PixMin does automatic event detection as a next step. That way, PixMin event detection will be more precise, for reasons described in the report. Although PixMin color-correction is easy to implement operationally, the analytical basis for PixMin color-correction requires an explanation, which we will provide in Section 3.13. For now, you can see from any two output folder image files that their colors look the same. If you open both files in IrfanView and select Image-Histogram for both, you will see that both histograms look the same. Finally, please note that when configured for color correction, PixMin ignores *color_weights.csv* file, *template_values_level_*.csv* file and *template_versions_level_*.csv* file entries.

## 2.14. Detecting flaws within rope inspection images



We have saved this use-case for last because it deserves special attention. Among all use-cases we have considered, visual inspection can most readily meet the broadest event detection needs. If you first look at your..*\2_14_FlawView_introduction.pdf* report, you will see that PixMin produced automatic event detection results quickly and demonstrated broad automatic event detection value. Accordingly, we spent extra time selecting a strong event detection dataset for this use-case and producing analysis results, as explained in Section 3.2.14.

## 3. Analysis methods

In this Section 3, we will cover key analysis elements in Section 3.1, we will cover use-case analysis details in Section 3.2, and we will include analysis notes in Section 3.3 that you will find cited throughout this Manual.

### 3.1. Key analysis elements

PixMin analysis includes three key elements. For any prospective application, PixMin analysts must:

a) Determine if and how PixMin could be valuable.
b) Obtain datasets that could be used to determine PixMin value.
c) Configure PixMin to determine optimal event detection value.

Unless PixMin has potential to add significant value, and unless appropriate datasets are available to determine PixMin value, configuring and evaluating PixMin will not be worth the effort. In that regard, evaluating elements a) and b) constitutes essential PixMin utility triage, which should routinely be performed beforehand. We have introduced elements a) and b) in two reports that you will find in your *..\Manual-cited_reports folder*, entitled *3_1_PixMin_added_value_potential_determination.pdf* and *3_1_use-case_evaluation_imagery_desirables.pdf.* Please read both reports before continuing with this section.

We begin this section by describing slides that illustrate key PixMin configuration elements. Please find and open the file entitled *3_1_analysis_elements_figures.pdf*. Slide 1 shows the PixMin operational sequence, beginning with preliminary **configuration**. During configuration, PixMin reads and parses **config** folder file contents, identifying each input **metric** along the way. Based on input metric values, PixMin sets up internal storage and computes internal metrics as necessary for either **continuous image processing** or **color alignment**. Nearly all PixMin runs in your ADK, and in practice, do not perform color alignment. We will cover color alignment later in this section.

If continuous image processing as usual has been configured, PixMin parses all other folder metrics during preliminary configuration, as shown in slide 1. PixMin then proceeds with consecutive image processing, one input image at a time, as shown in the the slide 1 loop. As shown by the dashed boxes in the figure, PixMin may or may not perform differencing, masking, or analysis during concurrent information processing. In Section 2.2, we introduced some of the operations shown in slide 1. We will give detailed descriptions of all such operations in the remainder of this Manual, starting with these remaining slides.

Slide 2 shows **config** folder contents, including snippets from some configuration files that we introduced in Section 2. Among those files, the only file we haven't yet introduced is the *config_errors.txt* file. PixMin produces that file when it has found errors in configuration file syntax during preliminary configuration. PixMin also displays all such errors in desktop dialog boxes when they occur.

PixMin runs after being configured by the ADK. After configuration, PixMin runs consecutively, either by processing each *input* folder file during ADK use-case runs or by processing input images during continuous operation deployment. During continuous deployment, the "Perform analysis" blocks shown will not be in effect because PixMin will have already been configured to operate quickly and precisely. Slide 3 shows one continuous operation deployment example corresponding to our Detecting Warm Bodies (Section 2.6) use-case. Once PixMin has been configured beforehand, PixMin gets triggered during consecutive time slices. At the beginning of each time slice, PixMin receives pixel data from an input image. At the end of each time slice, PixMin produces alerts only if they have been detected within the image. PixMin then stands by to receive pixel data from the next image at the beginning of the next time slice, and so on.

Slide 4 shows ADK use-case organization. As you have seen in Section 2, the ADK use-cases organizes use-cases within corresponding folders, reports, and Section 2 sub-sections. We have organized ADK use-case analysis descriptions likewise in Section 3.3 below. Slide 4 also shows use-case sub-folder organization. Along with *config*, *input*, and *output* folders, which we have already introduced, each use-case includes an *analysis* and an *output_archive* folder. The *analysis* folder for each use-case describes how we set up the use-case run as well as a series of use-case runs we made during configuration analysis. The *output_archive* folder for each use-case includes *output_** sub-folders from each of its configuration runs. The highest numbered *output_** sub-folder for each use-case contains the same output for the use-case that we asked you to produce in Section 2. We will cover analysis folder details for each use-case in Section 3.3 below.

Slide 5 shows how input image file color values are organized within **image files**. Within each file, PixMin reads color **RGB** value triplets consecutively. As we introduced in Section 2.1, PixMin converts each triplet to a **color feature** value while reading the image during **consecutive image processing**. Slide 6 shows how PixMin organizes resulting color feature values into one value per **pixel**.

Slide 7 shows how PixMin organizes **Level 1 alert blocks** and output **chips**. For some configurations such the one shown, alert block size does not evenly divide the number of input image rows and columns. As a result, PixMin makes boundary alert block sizes smaller as necessary to cover all input image pixels. PixMin always produces output chips that are the same size as configured level alert block sizes. If PixMin finds alerts within boundary alert blocks that are smaller than alert block sizes, PixMin outputs chips with boundaries shown by dashed lines in the figure to keep them the same size.

Slide 8 shows input and output folder contents for the whale detection use-case. In this case, PixMin output included **alert maps**, **alerted chips**, and a *chip_locations.csv* file as shown. As introduced earlier, and as we will explain further below, PixMin may use output *chip_locations.csv* files as input *config* folder files during analysis operations. For the same whale detection use-case, slide 9 shows **alert block** boundaries and a **template** graph for **Level 1** on the right, as well as alert block boundaries and a template graph for **Level 2** on the left.

Slide 10 shows some templates that we have used in our 13 use-case configurations. As you can see, most of these are statistical contrast templates, for reasons we have introduced earlier and will explain in more detail below. You will find corresponding templates files in your *..\configuration_files\template_values_files* folder. Slide 11 shows how the **rotation** and **scale** settings in your *template_versions_level_*csv* files affect template matching. The top, left frame shows that when they are 0 and 1, respectively, **template feature** values at each pixel point in an image are computed around each Region of Interest (**ROI**) **pixel** location by comparing each *template_values_level_*csv* file value with its homologous ROI value centered at that location. The top right figure shows how ROI values are matched with template values with a scale factor value of 1 and a rotation angle of 45 degrees. The two bottom frames show how ROI values are matched with template values when scale values are greater than one. Using different rotation and scale values in this way enables the same template to be matched to a variety of image orientations with a variety of sizes, as shown. Using scale values greater than one also allows a shape specified by a small size to be matched with larger sized objects, with taking any more processing time than matching the shape to a small sized object would take.

Slide 12 shows how template regions of interest (**ROI**) coverage can go beyond image boundaries. PixMin creates **frames** around image images that enable template matching values to be computed at all image boundary locations. Slide 13 shows how PixMin uses **mirroring** to compute frame pixel values.

Slide 14 shows how PixMin uses **masking** and **differencing** to highlight changes of interest with high **sensitivity** by using change detection as well as masking. Slide 15 shows how pixel alignment can make differencing even more sensitive when pixel location differences between *input_images* files and corresponding *diff* file images can otherwise confound actual changes. As shown in the right Slide 15 file, pixel alignment results in some edge pixels being **masked**, because some pixels at the edge cannot be paired after pixel alignment. Slide 16 shows how PixMin uses **crosshair matching** to align *input_images* files with corresponding *diff* folder folder files.

Slide 17 shows how image colors varied while a drone was capturing the images during repeated flights. The color variation was caused by varying sunlight and other ambient conditions. Color variation like that shown in the slide significantly reduces sensitivity, especially when change detection is configured, because changes in color can obscure subtle changes that PixMin must detect. Slide 18 shows that PixMin removes color variation by making image color **cumulative distributions** in input images the look same as the color cumulative distribution in a base input image. As result color-aligned histograms all look the same and color variation does not affect detection sensitivity.

As image processing analysts, we must configure automatic event detection solutions that involve essential tradeoffs. One significant tradeoff is between processing speed, weight, and power (**SWaP**) on the one hand and detection **precision** on the other hand. If we were to ignore SWaP constraints completely, we can always envision solutions that would improve simply by having an unlimited number of sensors looking for events over an unlimited period of time. Alternatively, if we were to use sensors and processors with negligible SWaP profiles, such as low resolution cameras taking only a snapshot every decade, we would also wind up with negligible event detection precision. With our analysis elements slides so far, we have introduced analysis components including color features, specialized templates, template features, change detection, and masking, which PixMin uses to produce both sensitive and efficient event detection, but only up to a point. With our remaining analysis elements slides, we will describe how these components can be thoughtfully used to produce highly sensitive event detection, while meeting operationally essential SWaP constraints. The most tangible constraint among these that we keep in mind is processing speed. For all of our use-cases, we have configured solutions that could find events within images arriving every few seconds or more often, on edge-based processors.

Slide 19 shows how speed figures into automatic event detection, in terms of PixMin computing consecutive image processing. As shown, after preliminary configuration, for each input image PixMin must first input images and convert them to feature values. These initial steps require only sub-second processing time on modern, low-SWaP processors including laptops and more specialized computers. Once these initial steps have been completed, however, PixMin must take for more processing time working through the six nested loops shown. Unless PixMin has been carefully configured, traversing these loops can require so many calculations completing them all would not be possible on any low SWaP processor. The example shown, based on the whale detection use-case image size, reasonable **template** sizes, and a reasonable number of template **combinations** is one case in point.

As you have seen in Section 2, we have configured our use-cases so all of them require orders-of-magnitude fewer calculations than those shown in the Slide 19 example. Instead, we have used much smaller templates and many fewer template combinations than in the example. Our **Level 1** configurations have been especially frugal, because Level 1 processing requires traversing every input image pixel. Our **Level 2** configurations have been less frugal, because a) Level 2 processing requires traversing only **alerted blocks** that have been triaged during Level 1 processing, and b) Level 1 alerted blocks typically have orders-of-magnitude fewer pixels than input images. In addition, we have routinely configured template **scaling** factors, which reduce processing calculations by configured scaling factor values for a template times the template's dimensionality. We also configured **differencing** and **masking**, both of which make simpler templates more effective. We have also configured less computing intensive **criteria** like **sums of absolute differences** and **projections**, instead of more computing intensive alternatives like **correlation coefficients**. We have used all such faster alternatives whenever they have produced satisfactory **precision**. Perhaps most effectively, we have carefully determined template configurations that are target-shaped as well as **robust** like **statistical contrast** templates to ensure both high **sensitivity** and high **specificity**.

Slide 20 shows how we configured the whale detection use-case to produce both acceptably high precision and acceptably high processing speed. First, we determined the effective templates shown (see Section 3.2.8 for details). Second, we configured only a one-dimensional Level 1 template. Doing so substantially triaged pixels to only those within Level 1 alerted blocks. Third, we used template version scaling, which further reduced processing speed.

Beyond configuring PixMin to run efficiently for this use-case, we have designed PixMin to run efficiently in several ways. For example, as we introduced in slide 11, PixMin takes no longer to evaluate criterion values for a template's scale value, however large, that for evaluating any other scale value, even though a large scale values' **ROI** will be correspondingly large. We have included many other powerful speedup features, not covered in this manual. You will be able to measure their effectiveness are by comparing PixMin processing speed for specific use-cases with that of alternatives such as OpenCV template matching.

The remaining analysis elements slides cover PixMin analysis output files and tools. Slide 21 shows PixMin selected analysis output folders and files from our whale detection use-case configuration. Along with normal output structure, PixMin will always produce Level 1 analysis like that shown when you configure **analysis option a**, **b**, or **c**. PixMin will do so for all configured **template combinations**, producing *alerts* folder **heat maps** like those shown. Looking at heat maps in your analysis *chips* folders will help you determine which template combinations detect selected targets specified in your *chip_locations.csv* file with the most **sensitivity**. Looking at heat maps in your analysis full folders will help you find template combinations may improperly classify **regions of interest** as targets due to low **sensitivity**.

Slide 22 shows, for the whale detection use-case, how its *chip_locations.csv* file determined its analysis output folder content when analysis option **b** was selected. As you can see, PixMin produced image heat maps in the *0_alerts* folder when an input image file had no corresponding *chip_locations.csv* file entry; PixMin produced image heat maps in the *alerts/full* folder when an input image had one or more *chip_locations.csv* file entries; and PixMin produced chip heat maps in the *alerts/chips* folder for every chip location that was specified in the *chip_locations.csv* file. As you can also see, PixMin also produced a *level_1\analysis_statistics.csv* file, which we will cover next.

Whenever we configure **analysis option a**, **b**, or **d**, PixMin always produces an *analysis_statistics_level_1.csv* file. To help make best use of the file contents, we often convert it to a **.xlsx* file, like the *1a_analysis_statistics_level_1_example.xlsx* file that you will find in your ..*\3_configuration_and_analysis_tool_files\analysis_tool_files* folder (see Appendix B). Before going into details on that file's contents, please recall the following configuration details from Section 2.8, whale use-case description.

- The use-case has input nine image files. Three of them labeled *config_snippet_1b.jpg*, *config_snippet_2g*.jpg, and *config_snippet_3b.jpg* have ROIs with whales. The other six do not.
- The use-case has one Level 1 template type and 12 versions for that type based on six rotation angle values times two rotation scale values.
- The Level 1 use-case is a one-dimensional splash detector and the Level 2 template is a two-dimensional back detector.

With those use-case details in mind, please keep slide 22 open, because we will be referencing the use-case *chip_locations.csv* file contents and *input_image* folder contents next. Now please open your..*1a_analysis_statistics_level_1_example.xlsx* file. You will see three tabs in the file.

Slides 23 through 26 contain excerpts from those tabs. Slide 23 shows a screen dump from the "from_ADK" tab. We made this tab by copying and pasting the contents into an output *analysis_statistics_level_1.csv* file from a whale detection analysis run into it. The ADK version of PixMin wrote each line of the *analysis_statistics_level_1.csv* file as it was created during the analysis run, in a sequence that isn't analyst friendly. After copying and pasting its contents, we made analyst-friendly counterparts to it by first highlighting the header line as shown using Excel's Format Cells option. We then used Excel's View-Freeze Panes option to enable scrolling down the tab while maintaining the header line at the top. Even so, the "from_ADK" tab analysis makes little immediate sense, as you can see.

To make more sense of the Level 1 statistics for this example, we copied the "from_ADK" tab into a new tab. We then we sorted the new tab's contents and named it "sorted_Chip_Row_then_Image." Slide 24 shows a screen dump from the top of that tab. The slide shows Total analysis statistics for each of the nine input images. We have highlighted the lines corresponding to images that included whales. For each image, the tab has one line for each of its Level 1 template versions. Each of these lines has a "# Alerts" entry, which is the number of Level 1 **pixel alert count** value that occurred within the line's image, template version combination. As you can see, the pixel alert counts were highest within images that contained whales, as it should be. Each of these lines also has a "Max Val" entry, which is the maximum Level 1 **pixel alert** value that occurred within the line's image for that template version. As you can also see, the maximum alert values were also highest within images that contained whales, as it should be. The Total values pointed to other points for potential further analysis. For example, images 1a, 1c, 2a, and 2c had zero alert counts but images 3a and 3c did not. That means PixMin found some ROIs within images 3a and 3c that may have been target-like. Looking closely at those ROIs, starting with image **heat maps**, might help determine alternative template combinations with better **specificity**.

Slide 25 shows a screen dump from the bottom of the same tab that we showed in slide 24. The slide shows Level 1 statistics for the chips that were specified in the *chip_locations.csv* file, shown in slide 22. For each chip location and template type line, you will find its number of alerts. As for Total image entries in slide 24, the "#Alerts" entries shown here are higher within the five chips containing whales than in the four chips containing no whales. Each line's Hit Ratio entry is the ratio of alert counts within its chip to the alert counts for pixels in the line's entire image other than pixels within chips for that image specified in the chip_locations.csv file. Entry values of "INF" mean that its divisor is zero, or equivalently that alerts only occurred within specified chips. Since that was the case for all snippet 01b entries, that means that no false alerts whatsoever were found an any its 4912x7360 pixels, other than its specified 513x513 chip pixel—a good indication that within-image false alerts may be unlikely. As you can see, the same pattern of promising results was held for snippets 02b and 03b as well. Also, the Hit Ratio values were higher for the snippets containing whales than for others. Likewise, the "Max Val" values were higher for snippets containing whales than for others.

Comparing template versions within images in slide 25 can also add analysis insight. We see by comparing them within snippet 01b, for example, that template version 0, corresponding to a horizontal white spike template, had the most hits, version 1, corresponding to a 45 degree template had the second most alerts, and version 2 corresponding to a 90 degree templated had only 1 hit.  That makes sense because the sippet 01 whale was moving in a vertical direction.  If we were concerned about processing time, we might look do further analysis to see if all versions were necessary by making another Level 1 run with only a zero degree version and a 90 degree version.  We might try larger scale factors with shorter Level 1 templates to see if they work as well.

Besides determining how well PixMin templates perform, ADK statistics can also help determine **pixel alert** and **alert count** cutoff values as shown in slide 26, which is a screen dump of the "cutoff_highlights" tab in your *1a_analysis_statistics_level_1_example.xlsx* file.  As shown in the top of slide 26, we configured that tab with an "alert count cutoff" cell value.  We also configured each "# alerts" value below it to contain a total count for each chip, summed all over each of its three template version alert counts.   We also set excel Conditional Format for each of those values so that they would be pink if they were at or above the "alert count cutoff" value and green otherwise.  Doing so allowed us to find a suitable "alert count cutoff" value small enough to produce **Level 1 alerted blocks** containing targets but large enough to produce as few Level 1 alerted blocks containing no targets.  We find such suitable alert count cutoff values routinely in this way while configuring use-cases as you will see in Section 3.2 below.

Turning next to Level 2 templates, please first recall the following Level 2 details.

a) PixMin performs Level 2 triage only on **Level 1 alerted** blocks.
b) PixMin creates Level 2 alert blocks only within Level 1 alerted blocks.
c) Level 2 alert blocks sizes must evenly divide Level 1 alert block sizes.  As shown in slide 9 for the whale use-case and in slide 20, which shows its *input_metrics.pxm.csv* file contents, we configured 81 57×57 Level 2 alert blocks to be evaluated within each 513×513 Level 1 alerted block.
d) Level 1 templates are compared to their **ROI**s centered at each pixel within each Level 1 alert block, even when the ROI falls outside the alert block.  For example, we showed how PixMin can evaluate ROIs outside alert blocks earlier in slide 12.  Likewise, Level 2 templates are compared to their ROIs centered at each pixel within each Level 2 alert block, even when the ROI falls outside the alert block.  For example, as you can see in slide 9 for our whale detection use-case, the Level 2 template for is larger than Level 2 alert block size.  Even so, PixMin compared that template to every ROI of the same size within every Level 2 alert block.
e) Just as Level 1 triage detects events within Level 1 alert blocks like those shown in the large slide 9 grid, Level 2 triage detects events within Level 1 alerted block, like the one shown in the small slide 9 grid.  Level 2 pixel alert count cutoff values are evaluated within Level 2 alert blocks rather than Level 1 alert blocks.
f) Level 1 triage can find multiple alerted blocks within an image, but Level 2 triage can find only one alerted block within each Level 1 alerted image.  Level 2 triage finds the Level 2 alert block within each Level 1 alerted block having the highest pixel alert count value.  PixMin then classifies the Level 1 alerted block as a Level 2 alerted block if that highest pixel count equals or exceeds the specified Level 2 alert count cutoff value.
g) As shown in slide 9, we made one only one Level 2 template to detect whale backs.  To make it cover different whale sizes and orientations, we configured 12 versions of that type to detect whale backs at six different angles and with two different sizes.

With a) through g) in mind, please open your ..*1b_analysis_statistics_level_2_example.xlsx* file in your ..*analysis_tool_files* folder (see Appendix B). You will see three tabs in the file, just as you saw earlier in its Level 1 statistics counterpart. The first "from_ADK" tab contains copies of the lines in an *analysis_statistics_level_2.csv* file that the ADK generated during a whale use-case analysis run. Slide 28 shows excerpts from that tab. As with its Level 1 counterpart, to make more sense of that tab we copied its contents into a new tab and then sorted its contents. In this instance, we sorted by "Type" in Column K and then by "Image" in Column A. We then named the new tab "sorted_Type_then_image." Slide 29 shows excerpts from that tab. The "Level 2 All" label at the end of each line in the tab means that the statistics on that line are based on all 81 of its corresponding Level 2 alert blocks. For example, the orange colored entries in the "# Alerts" column give the number of Level 2 alerts that PixMin detected in all 81 alert blocks for snippet 2b, broken down by the Level 2 template versions (six rotation angles times two sizes). If you compare those 12 entries to the 12 entries below corresponding to snippet 3a, you will see that the snippet 2b counts are much higher. That is as it should be because snippet 2b included a whale image but snippet 3a did not. You can verify that the came results held for all snippets in the tab, showing a clear path toward using the Level 2 template to triage only snippets containing whales. If you take a closer look at slide 29, focusing on the six target snippets and keeping the nature of the 12 template types in mind, you will see that templates with the same size and orientation as their whale ROI counterparts had the highest Level 2 alert counts, just as expected.

Next, please look at slide 30, which shows "Level 2 Block" statistics instead of the "Level 2 All" statistics in slide 29. Each entry in this slide shows the Level 2 alert block location within its corresponding Level 1 alert block where the highest Level 2 alert counts occurred as well as how many alerts occurred in that block. The hit ratios for each entry are computed based on the entry's template version, just as in the Level 1 case, except these hit ratios are computed only within its corresponding Level 1 alert block. These entries show the relative impact of each template version on detection within the Level 2 alert block having its highest counts.

Next, please look at slide 30, which is a screen dump of your "cutoff_highlights" tab in your *1b_analysis_statistics_level_21_example.xlsx* file. As shown in the top of slide 30, we configured that tab with an "alert count cutoff" cell value. We also configured each "# alerts" value below it to contain a total count for each chip, summed all over each of its three template version alert counts. We also set excel Conditional Format for each of those values so that they would be pink if they were at or above the "alert count cutoff" value and green otherwise. Doing so allowed us to find a suitable "alert count cutoff" value small enough to produce **Level 2 alerted blocks** containing targets but large enough to produce as few Level 2 alerted blocks containing no targets. We find such suitable alert count cutoff values routinely in this way while configuring use-cases as you will see in Section 3.2 below.

Turning next to Analysis Tools, slide 31 shows input and output to our *3a_feature_array_reorienting_tool_257x257.xlsx* file, which you will find in your ..*3_configuration_and_analysis_tool_files* folder. We routinely align **Regions of Interest** containing targets so that we can make a single templates and template versions that will cover all of them. We do so by first selecting **color feature** values within ROIs containing targets.

Such targets often have different sizes and orientations. We then reorient them so that their sizes and orientations match more closely. We then choose similar patterns within them and then construct templates that cover all of them accordingly. For example, the reorienting tool may transform the input chip values shown on the left side of slide 31 to output chip values that are in line with other target chip values. Besides allowing target color feature values to be input, the tool allows **scale** and **rotation** values to be input, resulting in output color feature values like the one shown on the right side of slide 31. As you can see in the folder, our tool kit includes one reorienting tool for smaller feature sizes and another reorienting tool for larger feature sizes.

Slide 32 shows how you can build **mask** files using our 3a_mask_region_building_tool.xlsx file, which you also find in your *..\3_configuration_and_analysis_tool_files* folder. We used that file extensively to build masks that have rectangular inner borders, but sometimes have internal borders with more sides, such as in our Asset Management use-case (see Section 3.2.5 below). In the same folder, you will also find a *3b_mask_region_insertion_tool.xlsx* file, which enables smaller built masking files to be inserted within larger masking files.

Slide 33 shows some **IrfanView** options, which we use routinely to examine PixMin input and output image files as well as set up input image files, as you will see in Section 3.2. Slide 34 shows how we convert input **HDFView** *.h5 files, which are output from the ADK when analysis option **b** is selected, to spreadsheets, which can be pasted into Excel and files like our color feature reorienting tool for further analysis. We intend much more HDFView use eventually to meet other future needs (see Section 4).

Slide 35 shows the layout for *5_drone_flight_plan_coverage_calculator.xlsx* file, which you will find in the same tools folder. We have used this file extensively to design drone flights that will provide necessary camera image file coverage, resolution, snap-shot frame rate and other settings for producing optimal PixMin input for automatic ground-target event detection.

## 3.2. Use-case analysis details

This section describes the same use-cases as Section 2 in that same order, but with more focus on why and how we configured PixMin for each use-case. Just as in Section 2 where we introduced **configuration** files and **metrics** sequentially as they figure into each use-case, we will introduce analysis elements sequentially as they figure into each use-case sequentially in this section. But first, we will outline the analysis sequence that we use routinely and recommend strongly.

For nearly all use-case analyses, we have routinely gone through the following standard analysis sequence for each use-case.

1) <u>Determine substantial automatic event detection value</u>. As explained in our *3_2_PixMin_added_value_potential_determination.pdf* report, automation may not add value in many event detection applications. Even among those where automation does add value, setting simple, fixed cutoff values may suffice. We have chosen ADK use-cases with strong potential automation added value. We encourage you to do likewise as your first, essential use-case selection step.

2) Obtain relevant analysis imagery. As explained in our *3_2_use-case_evaluation_imagery_desirables.pdf* report, use-case datasets should satisfy key requirements including operational relevance, among others. We have chosen many of our use-cases to meet those requirements, but we have chosen others that don't meet all of them in order to demonstrate how PixMin works. In operational use-cases, we encourage you to meet all such requirements, just as we have done routinely.
3) Focus analysis on minimizing false alerts subject to precise event detection. We routinely begin analysis by entering locations in the *..\config\chip_location.csv* and then making initial detection models that will cover all of them. We then refine analysis configurations to minimize false detections, run quickly, and meet other operational needs. Along the way, however, we ensure that our refinements will ensure precise target detection. (Our approach is consistent with, and has been inspired by, established **statistical optimization** methods.)
4) Make a **Level 1** run. Start with template values, template versions, and a **pixel alert** cutoff values that will detect all (or nearly all) targets. Choose Level 1 configurations that will run quickly by keeping the number of Level 1 template values low.
5) Determine satisfactory Level 1 cutoff values. Use analysis statistics to determine an appropriate pixel alert cutoff value and pixel count cutoff value combination that will either a) produce satisfactory results using Level 1 triage only or b) produce potentially satisfactory results using Level 2 triage after Level 1 triage.
6) Configure Level 2 if necessary. If Level 1 triage produces too many false alerts, configure Level 2 templates, template versions, and cutoff values that will reduce them.

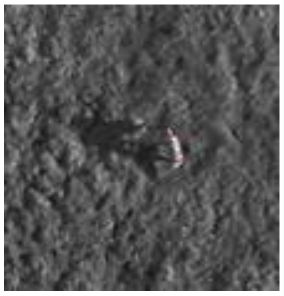
We will now describe specific analysis methods that we have employed for each of our ADK use-cases. For each case, we will refer to its *analysis* folder, its *analysis*_analysis_notes.pdf* file, its *analysis\setup* folder, and its *analysis\setup\setup_notes.pdf* file.

### 3.2.1. Detecting caribou from aircraft imagery

We begin this section, just as we began Section 3.2, with our caribou detection use-case because its analysis configuration is the simplest of all. As explained in its **._setup_notes.pdf* file, we selected a sub-image file within a much larger **orthomosaic** file that contained many caribou as an input image. We then configured PixMin to simply detect bright patches in the input image. Nearly all of them appeared within **Regions of Interest** containing caribou. As explained in the **.analysis_notes.pdf* file for this use-case, we made a few straightforward analysis runs, following the first five steps in our standard analysis sequence. We concluded that perfect precision would not be possible for this use-case due to inadequate **ground pixel resolution**. Even so, detecting caribou as shown could alert aircraft or drone pilots to the presence, so that they could reduce camera altitude to take closer looks. with better resolution.

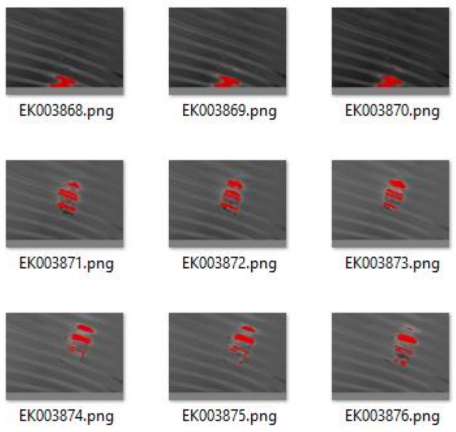
As with other our other use-cases we encourage you to make analysis runs on your own. For example, you could scan the orthomosaic file in this use-case setup folder for a sub-image with targets present, then save that sub-image, and then make analysis runs based on it.

The picture on the right is a sub-image of within an analysis **heat map** that PixMin produced as part of its *output_2* folder results for this use-case. As you can see, PixMin detected the caribou's bright antler and tail. As you can also see from the two shadows, the detected caribou was a female who had a nursing calf under her. The information in this sub-image typifies the kind of visual information that PixMin heat maps contain.



### 3.2.2. Detecting tortoises from trail camera imagery

As explained in the *._setup_notes.pdf* file for this use-case, we resized input image file **ground pixel resolution** considerably to enable higher speed detection because the original higher resolution wasn't necessary to produce perfect results. Details in the *.analysis_notes.pdf* file for this use-case are noteworthy but self-explanatory.



The picture on left shows analysis **heat maps** that PixMin produced as part of its *output_2* folder results for this use-case. These heat maps guided us to a configuration that highlighted images containing entire desert tortoise bodies while ignoring images showing only partial bodies and false alerts. As in the caribou detection use-case, the information in these heat maps typifies the kind of analysis guidance that they can provide.

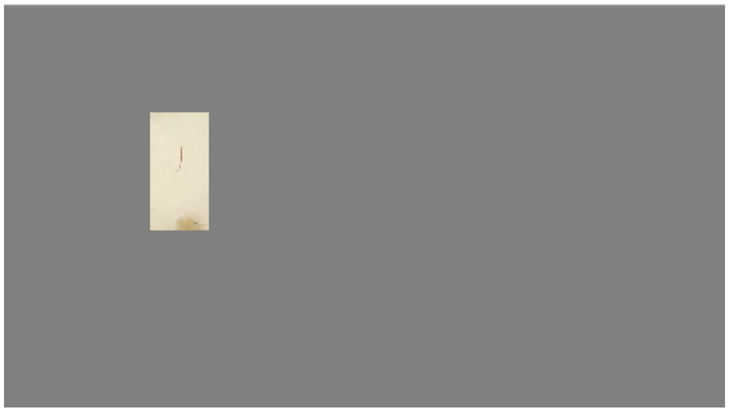
#### 3.2.3.1. Detecting threats from fixed camera imagery: bridge traffic detection

As explained in the *._setup_notes.pdf* file and the *.analysis_notes.pdf* file for this use-case, constructing relevant input images from a public website video was not easy. You will come across many such use-cases where operational imagery has not yet been obtained but a determination of potential value must be made based on the best dataset available. Details in the *.analysis_notes.pdf* file for this use-case are noteworthy but self-explanatory.

The picture on the right shows analysis **heat maps** that PixMin produced as part of its *output_2* folder results for this use-case. The heat maps show the effect that masking had fur this use-case. They also show how PixMin detected the threat when it appeared within its masked window for two of the four frames. Looking over heat maps like these considerably speeds up configuration refinement.

### 3.2.3.2. Detecting threats from fixed camera imagery: river debris detection

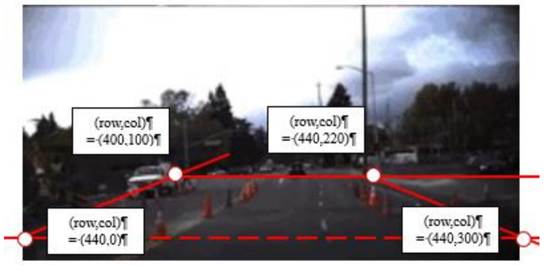
As in other use-cases and as explained in the **._setup_notes.pdf* file and the **.analysis_notes.pdf* file for this use-case, constructing relevant input images from a public website video was not easy. Details in the **.analysis_notes.pdf* file for this use-case are noteworthy but self-explanatory. Worthy of special note, once setup was completed configuring this use-case to produce perfect results took only one hour!



The picture on the left shows one analysis **heat map** that PixMin produced as part of its *output_2* folder results for this use-case. The red marks over the large object show that PixMin detected it, but the lack of red marks over the dark object show that PixMin did not detect it. Setting cutoff values in this and other settings depends strongly on desired target sensitivity. In this case, we chose cutoffs that would only detect large objects. In applications requiring more sensitivity, we would lower cutoff values accordingly. In all such cases, available analysis heat maps and other tools speeds up analysis considerably.

### 3.2.4. Detecting obstacles from driverless car imagery

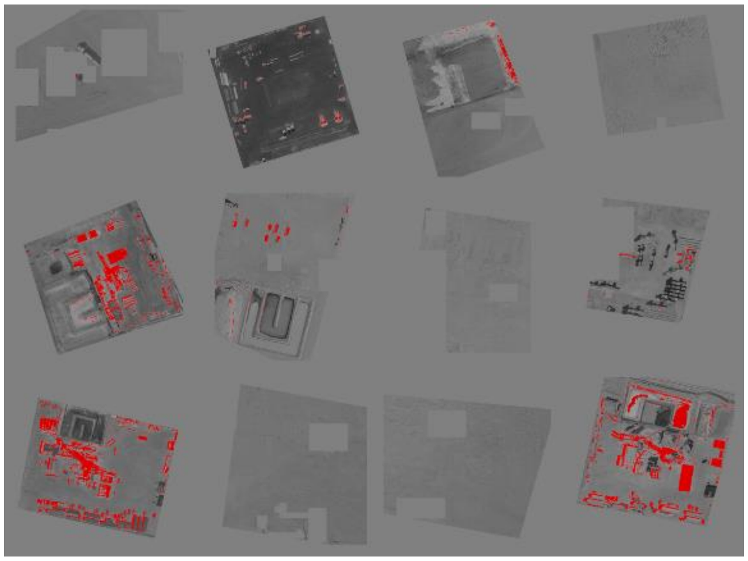
Details in the **._setup_notes.pdf* file and the **.analysis_notes.pdf* file and the **.analysis_notes.pdf* file for this use-case are noteworthy, especially two of them. First, we found that false alerts due to off-the-road **ROIs** containing cones and cone-like features were a problem. This problem could be (and in operational practice has been) resolved by effectively masking out all pixels that are not on roadways. For our use-case, we built masks that were adequate by going through a more tedious masking setup process, as shown in the figure on the right and explained in the **._setup_notes.pdf* file. This was one of several instances where setting up a use-case with our available ADK tools was more tedious and less adequate than configuring an operational PixMin solution would be.



Second, we were able to show that PixMin can meet operational requirements shown on the left and described in 2_4_PixMin_obstacle_detection.pdf report. Most notably, PixMin can identify obstacles like construction cones as well as unexpected anomalies within a few milliseconds and upload chips quickly over low bandwidth channels to meet requirements like those outlined in the report. As with most other ADK use-cases, we set this one up mainly to demonstrate that its operational concept was feasible.

### 3.2.5. Detecting oil field changes from aircraft imagery

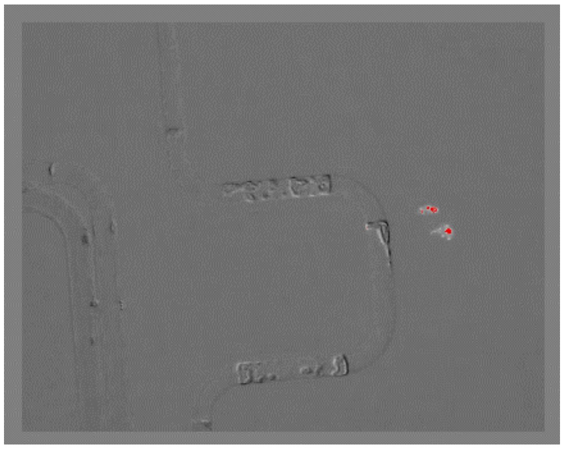
We organized files for this use-case in folder named *5_asset_management,* containing two sub-folders named *gross-oil-pad-change_detection* and *within_oil-pad_change_detection*. You will find our rationale for organizing the use-case in this way, explained in the ..\ *within_oil-pad_change_detection_notes.pdf* file. Other details in the two**.analysis_notes.pdf* files and the two **.analysis_notes.pdf* files for this important use-case are noteworthy but self-explanatory. They are especially noteworthy because many asset management applications that depend heavily on airborne imagery and satellite imagery could benefit strongly from the kind of triage that we have demonstrated for this use-case.



We have already seen that change detection can be not only highly sensitive to substantive events but also overly sensitive to confounding effects, especially pixel misalignment, color variation, and uninformative changes outside **regions of interest**. We have shown how PixMin mitigates these three effects through pixel alignment, color correction, and masking, respectively. The heat map on the right highlights how PixMin can mitigate shadow differences within oil pads through masking as well. As you can see, however, shadow differences can be so substantial that masking them can also block out large regions that also could contain changes of interest. In such substantial shadowing cases alternatives, such as flying over regions when shadows are the same or non-existent or using sensors that produce shadow-free images may be needed.

### 3.2.6. Detecting warm bodies

This use-case brought in a new analysis element because it required detected chips from thermal images to trigger output chips from optical (**RGB**) images. We configured the ADK version of PixMin with analysis option **c** specifically to meet this requirement. Deployed versions of PixMin C may be integrated to suit alternative camera configurations in real-time, along the same lines of option **c** but by processing one pair of images at a time instead of processing all thermal images followed by processing all RGB images. Deployed versions of PixMin may also benefit from dual cameras that already align paired images spatially as well as temporally. Many steps that you will see in the **._setup_notes.pdf* file and the **.analysis_notes.pdf* file for this use-case may not be needed in practice. This was yet another use-case instance where setting up a use-case with our available ADK tools was more tedious than configuring an operational PixMin solution would be, but doing so was still essential to demonstrate use-case feasibility. Other than that new analysis element, details in the **.setup_notes.pdf* file and the **.analysis_notes.pdf* file for this use-case are noteworthy but self-explanatory.



The picture on the right shows an image from the *output_6* folder for this use-case. It shows how PixMin can produce highly precise detection from thermal imagery coupled with change detection and pixel alignment.

### 3.2.7. Detecting underwater objects from drone imagery

Setting up the input datasets for this use-case took substantial time and effort. As fully explained **._setup_notes.pdf* file, we first designed design an operationally relevant drone-based surveillance flight plan and then generated simulated imagery that the flight plan would produce. We did so to demonstrate what could be observed for the use-case in keeping with its *2_7_PixMin_underwater_threat_detection.pdf* report.

For this use-case:

a) We found that using the correlation criterion produced perfect results.
b) We evaluated runs for many other criteria.
c) We set up input images to reflect what they would look like if a drone were to descend after making a transact search detection but we did not do any analysis based on them.

We believe that completing runs along the lines of b) and c) would be valuable exercises for you. Make it so!!

### 3.2.8. Detecting whales from airborne images

The **._setup_notes.pdf* file and the **.analysis_notes.pdf* file contents for this use-case are extensive, but they should be self-explanatory.

### 3.2.9. Detecting desert tortoises from drone imagery

For this use-case, we made four three sets of runs. The first set of runs, included in the use-case *pixel_alignment* folder, showed that **pixel alignment** with **differencing** was essential for this use-case. The second set of runs, included in the post-flight_detection folder, showed that using pixel alignment to compare paired **orthomosaics** covering the same ground on two consecutive days, produced good results. The third set of runs, included in the *real-time_detection* folder, showed that much higher resolution than we had in available imagery would have been needed to detect tortoises without differencing. Each of these three sets has its own **._setup_notes.pdf* file and the **.analysis_notes.pdf* file. Some of the notes are extensive but they should be self-explanatory.

### 3.2.10. Detecting ripples in water

As stated at the beginning of the **._setup_notes.pdf* file for this use-case:

> "We created the input image for this example to include eight target ripples within a VGA-sized input image. We created the input image by embedding the eight targets within a seawater image file. We created the eight targets in keeping with a two (frequencies) by two (phases) by two (damping factors) design. We created the target sub-images for embedding and then embedded them into the image carefully, to make the input image realistic."

The remainder of that file provides corresponding setup details.

As also stated in the **.analysis_notes.pdf* file for this use-case:

> "We made a variety of runs using different **template matching criteria**. Among those tried, the **projection** criterion performed best in terms of target detection **precision** and processing speed."

The remainder of that file provides corresponding analysis details.

### 3.2.11. Detecting events from display imagery

The **._setup_notes.pdf* file and the **.analysis_notes.pdf* file contents for this use-case should be self-explanatory.

### 3.2.12. Classifying objects with template matching

Please open your *2_12_PixMin_rocket_classification.pdf* report and look at Figure 2. As shown by the TARGETS images on the left side of the figure, PixMin produced criterion matching values from each of four input images. For each of these input images, PixMin produced 12 criterion values as shown in the 12 columns of values in the figure. You will find output criterion values that PixMin produced corresponding to each column in an *output_archive* folder for this use-case with the same corresponding name. For example, you will find output criterion values of 4.7, 3.5, 4.0, and 3.2 in the *analysis_statistics_level_1.csv* file in the *output_archive\output_1_A_full_left* folder.

We made 12 PixMin runs to produce each set of outputs in the *output_archive* folder. We did so by making a pair of runs for each of the six aptly named **.template_values.csv* files in the *..\config* folder. For each of those six **.template_values.csv* files, we made one run using the *template_versions_level_1.csv* file and one run using the *template_versions_level_2.csv* file.

With the above layout and Figure 2 results in mind, the **._setup_notes.pdf* file and the **.analysis_notes.pdf* file contents for this use-case should be self-explanatory.

### 3.2.13. Correcting images for color variability

We did not create an analysis folder for this use-case because its setup and analysis are evident from its *2_13_PixMin_image_color_variability_correction.pdf* report, the Section 2.13 explanation, and the slide 18 explanation in Section 3.1.

### 3.2.14. Detecting flaws within rope inpspection images

As introduced in Section 2.14, this use-case stands out among all others because visual inspection can most readily meet the broadest event detection needs. As you will see in the *..\analysis* folder for this use-case, we spent extra time selecting a strong event detection dataset for this use-case and producing detailed analysis results. Before digging into the *..\analysis* folder contents for this use-case, we recomment that you open the *PixMin_video.mp4 in your 0_ADK_videos* folder and review its FlawView Detailed Description section.

## 3.3. Analysis Notes

The notes in this section, which are cited by note number throughout this manual, contain details that are not covered elsewhere in the Manual.

<u>Note 1: Level 1 **alert block** size</u>. If the Level 1 alert block size evenly divides the number of image rows and columns, each Level 1 alert block will be exactly this size. Otherwise, Level 1 alert blocks at image borders may have fewer rows or columns as necessary to result in every image **pixel** falling within exactly one alert block.

<u>Note 2: making your own PixMin runs</u>. At any point while reading this Manual, we encourage you to make your own PixMin runs based on any use-case modifications you wish to make. If you wish to modify the configuration for any use-case, we recommend that you first copy that use-case folder to a location in your own workspace, then modify only the copy in your workspace, and then run PixMin by selecting only the **.pxm.csv* file in your copy. That way, you will retain the all original use-case folders for later comparison and proper referrals from this Manual. If instead you inadvertently change the configuration within any of the use-case folders that originally came with this PixC, we have included an *ADK_use-case_archive* folder that will enable you to recover each original use-case folder. That archive folder includes a **.zip* file containing the original ADK folders for each use-case.

Note 3: color weights determination. PixMin configuration requires specifying three **color weight**s in *..\config\color_weights.csv* files. Color weights determine a weighted sum of **RGB** values, computed for each pixel. When PixMin computes a **grayscale feature** value based on standard grayscale weight values of (0.2.26,0.7152, 0.0722), the resulting image viewer rendering appears to be gray instead of colored.

PixMin may also be configured with **statistical** contrast (see Note 4 below) **color weights**, which sum algebraically to zero. When an input **pixel's** three **RGB** values are gray (or nearly gray), contrast color weighting will make its **color feature** value zero (or nearly zero), because all three grayscale input RGB values will be the same (or nearly the same). When the highest input pixel RGB value corresponds to the most positive contrast color weight value and its lowest input pixel value corresponds to its most negative contrast color value, its color feature value will be positive. In the opposite case, its color feature value will be most negative. That way, contrast color weights will produce the most positive color feature values when for pixels that have the same profile, while grayscale (or grayscale-like) values will be zero (or nearly zero), and their values will be either less positive or negative. For example, contrast color weights if (05. -0.25, -.25) will produce high color feature values when input RGB values are pure red.

Note 4: **statistical contrast templates**. When PixMin configures template **contrasts** for **templates** along with **projection** matching criteria, **ROI**s that have uniform clutter will be unlikely to produce **alerts** because high and low pixel values within ROIs will cancel each other out. When configuring PixMin contrasts, we sometimes configure template values of zero outside target-shaped template values and then place other template values that will make all values sum to zero further outside the zero values. We do that to create buffer zones around target-shaped template zones that will enable regions around targets, which can be highly variable and reduce detection **precision**, to be ignored.

Note 5: **chip locations files**. These files of the form *chip_locations.csv* (see Section 3.1 explanation of slide 22; Appendix B) may be either produced as part of **normal output** or received as analysis input. When produced as part of normal output, the file identifies image alerted block output chips. When placed in the *..\config* folder with **analysis** options selected, the file identifies alert block output chips for which **analysis statistics file** output will be produced. In either case, chip locations files identify chip locations within input images.

Note 6: **template** size versus **consecutive image processing** speed.

During **Level 1** processing, PixMin **triages** Level 1 **alert blocks** that satisfy Level 1 **pixel alert cutoff** value and Level 1 template matching **criteria**. PixMin classifies alert blocks that meet those criteria as Level 1 **alerted blocks**. If Level 2 triage has not been specified in the **.pxm.csv* file, PixMin produces **normal output** based on those Level 1 alerted blocks. If Level 2 triage has been specified in the **.pxm.csv* file, PixMin looks further within Level 1 alerted blocks for Level 2 alert blocks that satisfy Level 2 template matching criteria. PixMin classifies alert blocks that meet those criteria as Level 2 alerted blocks. If Level 2 triage has been specified, PixMin produces normal output based on those Level 2 alerted blocks.

PixMin processing speed depends directly on the total number of configured level 1 template values. For example, when only one **template version** has been configured, PixMin processing speed will be 10 times slower if a *template_values_level_1.csv* file has 20 rows and 10 columns that if the file has 5 rows and 4 columns. Likewise, PixMin processing will be four times slower if four level 1 template versions have been configured than if only one template version has been configured. PixMin processing speed depends on the number of level 1 template values in this way whether or not **Level 2 triage** has been configured.

We have designed PixMin two have two configurable **triage levels** to allow substantial processing time reduction while maintaining high detection **precision**. In our whale detection use-case, for example, we configured the total number of level 1 template values to be small, enabling fast level 1 triage that produced only a small number of **level 1 alerted blocks**. We configured the total number of level 2 template values to be much larger to enable higher detection precision. Configuring more level 2 template values took much less time than if we had configured more level 1 template values because level 2 triage took place only within level 1 alerted blocks, which was much smaller that the total number of level 1 **alert blocks**.

Note 7: **HDFView** numerical value clamping requirement. When HDFView converts an input ***.jpg** file to a ***.h5** file, it automatically rescales (and distorts) input image values based on its maximum and minimum as described in the link below. If the maximum value is 255 and the minimum value is 0 no rescaling will occur (see https://support.hdfgroup.org/products/java/hdfview/UsersGuide/ug06imageview.html and scroll down to "Converting non-byte data to byte data"). Rescaling distortion can be precluded by making a "clamped" copy of an input image file using IrfanView. The clamped file can be made by converting a few pixel values to 0 at one corner of the image file and other pixel values to 255 at another corner of the image file. When the clamped image file is read into HDFView, all HDFView pixels will have the same values as their corresponding clamped image pixels. Clamping does not work when HDFView converts a grayscale image. (We found nothing explaining the problem on the web.) For that reason, we only used HDFView to convert clamped RGB **.jpg* files.

Note 8: **analysis** options. Produces analysis output as specified by **pxm.csv* file line 22 (see Appendix C), as follows.

- If left blank, no analysis will take place, enabling PixMin to run much faster than when other analysis options are selected.
- if set to **a**, PixMin will produce Level 1 **heat maps** and **analysis statistics files** as well as their counterparts if Level 2 has been selected.
- if set to **b**, PixMin will produce the same output and **analysis statistics files** as when set to **a**, except instead of producing heat maps, PixMin will write Level 1 feature values to **.h5* files (see Appendix B).
- if set to **c**, PixMin will receive **chip location** values from the *chip_locations.csv* file and produce **normal** output as if PixMin had produced alerts at the same chip locations.

- if set to **d**, PixMin will produce the same output and **analysis statistics files** as when set to **a**, except PixMin will color output image file **pixels** and output **heat map** pixels red at all locations where PixMin produced alerts based on them.

Resulting **analysis statistics files** files of the form *analysis_statistics_level_1.csv* and *analysis_statistics_level_2.csv* and their **.xlsx* counterparts (see Section 3.1 explanations of slides 21-30), contain **alert counts**, broken down by input **image file**, **chip locations file** chip, and **template combination**, along with other **ADK** output statistics. Analysis statistics files, along with analysis **heat maps**, enable analysts to determine performance of each template file for each chip specified in a **chip locations file**.

Note 9: **standardizing** effects. PixMin routinely **standardizes criterion** values so that their cutoff values will be **normalized** to the same distribution scale. PixMin optionally standardizes input **color feature** values (see Appendix C, line 6 entry) so that PixMin detection results will not vary over images due to differences in image mean (*cf.* **brightness**) and variance (*cf.* **contrast**) values.

Note 10: **change detection**. PixMin performs change detection (*cf.* **differencing**) as follows.

- PixMin finds a *..\config\diff* folder. The *diff* folder may contain either one file for each *..\input_image* folder or a single file named *..\default_diff.png*. Each file in the diff folder must be in acceptable **input file** format.
- PixMin matches a file in the *diff* folder with a file in the *..\input_image* folder. If PixMin finds a single *..\default_diff.** file in the folder, then it matches every file in the *..\input_image* file with that single file. Otherwise, PixMin matches the first pair of files in each folder with each other, the second pair with each other, and so on.
- PixMin then performs **differencing** without pixel alignment, during which each image color feature value in a *diff* folder file is subtracted from its homologous color feature value in its corresponding *input_image* folder file.
- If set to a value greater than one, PixMin **performs differencing with pixel alignment**.

Note 11: **correlation matching**. Correlation **criterion** matching values do not depend on template means, template standard deviation , or their corresponding feature means and standard deviations. As a result, correlation matching automatically adjusts regions within **image files** for differences in brightness and contrast, much like **SAD matching** with **centering**.

Note 12: PixMin **criterion** setting alternatives. PixMin can **match** templates to **ROIs**, according to one of ten methods: **masked** or unmasked versions of **sums of absolute difference (SAD) values**, **projections**, **correlations**, **"centered"** SAD values, or **standard deviations**, and **correlations**; PixMin must be configured to process one criterion at **Level 1**, whether or not **Level 2** processing has been configured. PixMin must also be configured to process one criterion during each run when Level 2 processing is configured.

Note 13: **grayscale** weighting. When PixMin computes **color feature** values based on standard grayscale weight values of (0.2.26,0.7152, 0.0722) specified in the *color_weights.csv* file, the resulting image viewer rendering appears to be gray instead of colored. Also, when producing **analysis** output, PixMin marks **criterion** values that don't exceed **pixel** cutoff values values with grayscale values.

Note 14: **heat map** values. A **heat map** output **PNG** file shows image **criterion pixel** values ranging from the lowest (black) values that would be least likely to produce a **pixel alert** to the highest (white or red) values that would be most likely to produce a pixel alert. Heat map values exceeding pixel alert cutoff values are shown as red. Other criterion pixel values are shown on a **grayscale**.

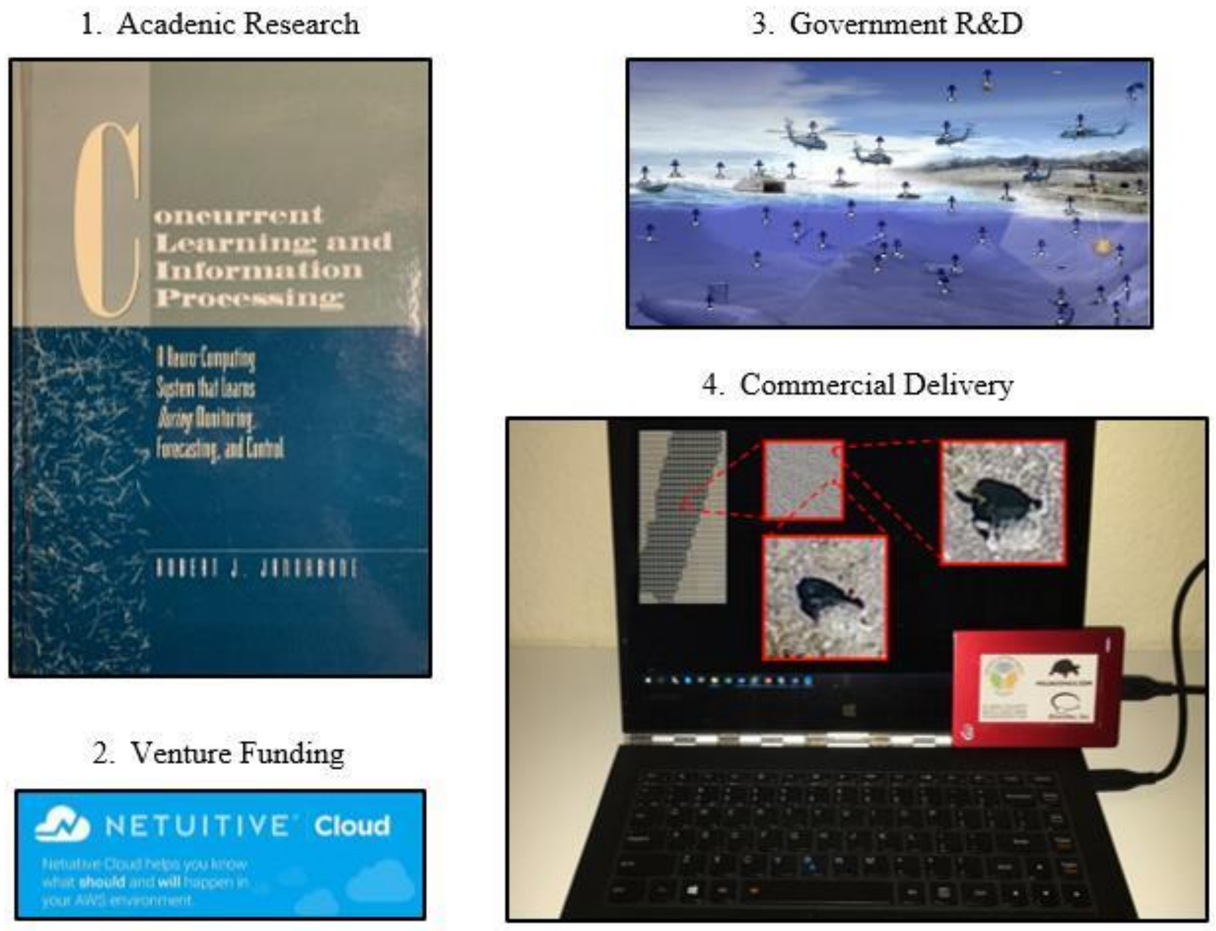
Note 15: feature . PixMin performs **masking** as follows.

- PixMin finds a *..\config\mask* folder. The *mask* folder may contain either one **.csv* file for each *..\input_image* folder or a single file named *..\default_mask.png*.
- PixMin matches a file in the *mask* folder with a file in the *..\input_image* folder. If PixMin finds a single *..\default_mask.csv* file in the folder, then it matches every file in the *..\input_image* file with that single file. Otherwise, PixMin matches the first pair of files in each folder with each other, the second pair with each other, and so on.
- PixMin evaluates selected pixels within each input image, as specified by input masking values that PixMin reads from the *..\mask* folder.
- If a masking value is zero at a corresponding **pixel** location, PixMin does not perform automatic detection at that pixel location or at any pixel location in the **region of interest** (ROI) that covers that pixel location. As a result, PixMin will only perform automatic detection at those pixel locations with corresponding mask values of 1, and only if all ROI pixels surrounding those pixel locations have corresponding mask values of 1.
- Masking values may be either input as files in the *mask* folder (see Appendix B) or determined during **pixel alignment**, or both (see Section 3.1, explanations of slides 13-14).

Note 15: **projection** meaning. We call sums of products between values between **templates** and homologous **feature** values **projections** because projection values are highest when template shapes match their shapes match homologous feature shapes. Projection based template matching can be especially effective when used with statistical contrast templates. Projection-based template matching can be especially effective when used with **statistical contrast templates**. **Correlation** matching may produce more **precise** detections than projection matching, but projection matching takes much less processing time.

Note 16: **template types**, **template versions**, and **template combinations**. For any PixMin **configuration** at **Level 1** and optionally at **Level 2**, you may configure any number of template **types** in *template_values_level_*.csv* files (Appendix B) as well as any number of template **versions** in your *template_versions_level_*.csv* files. For each type that you specify in a *template_values_level_*.csv* file, you must include a set of corresponding template values in that file. For each version that you specify in a *template_versions_level_*.csv* file, you must specify a pair of scale and rotation values in that file. Based on those specifications, PixMin produces a criterion value at each input image **pixel** location for each corresponding pixel type by pixel version **combination**.

## 4. PixMin Evolution


1. Acadenic Research
Concurrent Learning and Information Processing
A Neuro-Computing System that Learns During Monitoring, Forecasting, and Control
ROBERT J. JANNARONE
2. Venture Funding
NETUITIVE Cloud
Netuitive Cloud helps you know what should and will happen in your AWS environment.
3. Government R&D
4. Commercial Delivery

The PixMin concept took shape when our founder proposed that machine learning and detection should take place all at once, unlike separate training and detection required by conventional machine learning. With venture funding, his teams developed related products for real-time, automatic event forecasting. With Government R&D funding, his teams refined related solutions to detect threats from a broad variety of sensors. These four key developments have evolved PixMin into a powerful product for triaging massive streams of remote sensor imagery into nuggets of valuable information—automatically, at the sensory edge, and in real time.

Most recently, we have focused on making PixMin analysis more powerful, general, interpretable, and visually interpretable. Our extensive and understandable ADK use-case library, configuration, and analysis options have resulted, enabling analysts without deep learning or statistical analysis backgrounds to configure, run, analyze PixMin's edge machine learning results, independently and effectively with their own datasets and without any outside help.

## 5. Future Directions

With your help, we intend to greatly expand out collection of ADK use-cases and our library of use-cases. As integrated PixMin solutions and ADK use-cases increase, we anticipate other developments, including the following:

- Increasing triage to include more than two levels.
- Including provision for automatic optimization based on finding templates, cutoff values, and other configuration options that will produce the most **precise** detections possible, subject to specified **SWaP** constraints.
- Added tools to streamline PixMin use-case setup and analysis.
- Database management tools that will enable quick access to configuration, template library, tools, and use-case metadata. Efficient database organization will be essential for streamlining configuration and analysis as well as for enabling automatic optimization.

## 6. Conclusion

In this Manual we have introduced the PixMin Analyst Development Kit and we have showed you how to run datasets that are included in the ADK. Now that you have read the Manual and run some examples, you should understand why PixMin automatic triage is valuable and straightforward. You should also be able to begin analyzing your own datasets and configure PixMin, on your own and in your own way. We are here to help along the way. Feel free to contact us at any point. More than anything, we want PixMin to help you save time and money by detecting nuggets of useful information within massive streams of imagery—with your own data, deftly and on your own!

## Appendix A.  Terms glossary

This Glossary describes key terms, which you find set in **bold font** below and throughout this Manual.  Each term entry that is not self-explanatory points you to sections in the Manual where it is introduced and explained.

- analysis option) **a**: a **config file** option that produces output analysis **chip** level and **image** level **heat map PNG**  files showing **criterion** values, along with **analysis alert counts** files (see Section 3.3, Note 8).
- (PixMin) **ADK**: **Analyst Development Kit**.
- **AI**: **artificial intelligence**.
- **alert block**: an **image file** sub-image within which PixMin identifies and counts **alerts** (see Section 3.3, Note 1).
- **alert block layout tools**: see Appendix B.
- **alert maps**: **PixMin** output images highlighting alerted **blocks** with black boundaries.
- **alerted block**: an **alert block** with its number of  **pixel level alerts** exceeding the **alert block cutoff** value.
- **alerts**: output events within **image files** that have been detected, based on **config folder metrics** that analysts have determined using the **Analysis Development Kit**.  PixMin produces each alert as an **alerted chip**, an **alert map**, and a *chip_locations_.csv* file entry.
- (PixMin) **analysis**: determining **precise** PixMin **configurations** subject to **SWaP** constraints, by examining **analysis option** output.
- **analysis alert counts files**: **analysis output** files of the form *analysis_counts_level_1.csv* and  *analysis_counts_level_2.csv*, containing **pixel alert counts**, broken down by input **image file**, **chip locations file** chip, and **template combination**.  These counts, along with analysis **heat maps**, enable analysts to determine template performance for each image file within the chip locations file.
- **analysis option**: a **configuration input metric** that can specify one of four **analysis options**: blank (producing no analysis output), **a**, **b**, **c**, or **d** (see Section 3.3, Note 8).
- **analysis output**: a collection of **image files** or **h5 files** including **heat map** files and **analysis output chip** files under either analysis option **a**, **b**, or **d**, along with **analysis alert count files.**
- **analysis statistics files**: analysis **output files** of the form *analysis_statistics_level_1.csv* (see Section 3.3, Note 8), containing **alert counts**, broken down by input **image file**, **chip locations file** chip,.
- **Analyst Development Kit (ADK)**: A collection of use-case folders, tools, and documentation, along with a **PixMin** executable file.  The ADK enables you to run each use-case in the collection using PixMin.  The ADK also provides you with sufficient tools and understanding to prepare and run your own use-case files, independently and on your own.
- **ANNs**: **artificial neural networks**.
- **artificial intelligence (AI)**: see Wikipedia artificial intelligence.
- **artificial neural networks**: see Wikipedia artificial neural networks.

- (analysis option) **b**: a **config file** option that produces output **analysis chip** level and **image level HDFView** files containing **criterion** values, along with **analysis alert counts** files (see Section 3.3, Note 8).
- **auto-adaptive processing**: using **lagged image differencing**, **contrast templates**, image **standardizing**, and other methods to identify events of interest, while controlling for changes in background brightness and contrast over time, background changes within images over time, and background differences within images. Auto-adaptive processing increases PixMin detection **precision**, makes PixMin detection **robust**, and eliminates or reduces needs to recalibrate PixMin configurations periodically.
- **Bayer filter mosaic**: see Wikipedia bayer filter mosaic.
- **binary (metric)**: see Wikipedia binary data.
- **block alert count**: the number of **pixel** locations within an **alert block** having **criterion values** that exceed a configured **pixel alert cutoff** value.
- **alert block cutoff**: if a **block alert count** exceeds this value, which is specified in the **pxm.csv* file, PixMin classifies the alert block as **an alerted block**.
- (chip) **borders**: black boundaries that PixMin places at the edge of **alerted chips** within **alert maps**.
- **brightness**: (see Section 3.3, Note 9; Wikipedia brightness).
- (analysis option) **c**: a **config file** option that produces **normal** output based on input **chip locations file** entries (see Section 3.3, Note 8).
- **C++** (programming language): see Wikipedia C++ programming language.
- **change detection**: highlighting pixels that have changed between images shot at different time points by subtracting **pixel** values in a **diff folder** from homologous pixel values in a corresponding **input image file** (see Section 3.1 explanations of slides 14-16; Section 3.3, Note 10; Wikipedia statistical change detection). We use the term interchangeably with **differencing**.,
- **chip**: A chip may be a subset of an input image corresponding to an **alerted block** that PixMin has identified and produced as a part of **normal output**. Alternatively, a chip may be represented by its location with an input **chip locations file**. In either case, a chip must be the same size as the "Level 1 alert block and output chip size" specified in the **config file**.
- (analysis or output) **chip locations files**: files of the form *chip_locations.csv* that may be either produced as part of **normal output** or received as analysis input (see Section 3.3, Note 5).
- **coding conventions**: see Wikipedia coding conventions.
- **color alignment**: a PixMin process that converts each input image to a counterpart that has the same histogram as a *Basis.jpg* file in the config folder. Color alignment corrects images for color variability that could otherwise produce variation that would inhibit automatic event detection (see Section 2.13).
- **color feature** (values): single **feature** PixMin values that PixMin computes at **each input image pixel** location as a function of its three **RGB** values .
- **color weights**: three numerical input *color_weights.csv* file (Appendix B).values that PixMin uses at each **input image pixel** location to compute its **color feature** value as a weighted sum of its three **RGB** values (see Section 3.3, Note 3).

- **config file**: a file of the form **.pxm.csv*, containing **configuration input metrics**.
- **config metrics**: numerical-valued or text-valued input entities with labels that analysts set in **config files** (Appendix B-C).
- **config folder**: a collection of sub-folders and files, most notably a **config file**, containing **configuration input metrics** (Appendix B).
- **consecutive image processing**: the phase where PixMin identifies events one image at time after the **preliminary configuration** phase.
- **contrast**: a measure of image sharpness. (see Section 3.3, Note 9; Wikipedia Contrast (vision).
- **correlation match**: a **matching** value, computed as the correlation coefficient between the values for any given **template** and homologous image **feature** values (see Section 3.3, Note 11).
- **criterion**: a basis for **matching** templates to **ROIs**, according to one of ten methods (see Section 3.3, note 12).
- (template) **criterion values**: **feature** values, computed at each **pixel** location, as a function of **template** values centered at each pixel location and their corresponding **ROI** values at that location..
- (pixel alignment) **crosshairs**: cross-shaped arrangements (when represented in two dimensions) of input image **feature** values and corresponding *diff* folder images used for **pixel alignment**.
- **crosshair match**: **SAD** value between two pairs of crosshairs.
- **cumulative distribution function (CDF):** see Wikipedia cumulative distribution.
- **cutoff** (values): numerical valued **configuration input metrics** that PixMin uses to determine if alerts have occurred. For each **triage level**, cutoff values come in pairs made up of a **pixel alert cutoff** value and a **block alert cutoff** count.
- (analysis option) **d**: a **config file** option that produces output analysis **chip** level and **image** level **discovery map PNG** files showing **criterion** values, along with **analysis alert counts** files (see Section 3.3, Note 8).
- **diff folder**: a **config** subfolder containing one or more **image files** that will be used during **change detection** (Section 3.1).
- **differencing**: we use this term interchangeably with **change detection** (others may give the terms distinct meanings -- see Wikipedia image subtraction ).
- **differencing file**: a ***.PNG** file within a **diff** folder that PixMin uses for **differencing** (see Appendix B).
- **discovery** (maps): event detection representations produced by PixMin when analysis type = **d**, so named because they enable analysts to "discover" event pixels embedded within input **image files**.
- **display contrast**: see Wikipedia display contrast.
- **dynamic:** see Wikipedia dynamic memory allocation.
- **empty** (criterion value): a criterion value that cannot be computed during consecutive image processing because either a) one or more **ROI** values that go into computing it is masked, b) its **standard deviation** match value is zero, or c) its **correlation** match value has a zero

denominator. Once a criterion value has been identified as empty, PixMin marks it as such and acts accordingly.

- (PixMin) **event detection:** identifying **alerted blocks** within **input image files** and producing corresponding **normal output** files or **analysis output** files.
- **feature**: a functional combination of **pixel** values, for example **color feature** values and **template criterion** values .
- **feature array**: a two-dimensional **feature** representation.
- **feature array reorienting tools**: see Appendix B.
- **fps**: **image** capture rate in frames per second.
- **frame**: a border of **pixels** surrounding a **feature array**.
- **framed image**: a two-dimensional array containing pixel values within a frame.
- **GIF** (format): see Wikipedia GIF.
- **GPU**: see Wikipedia GPUs.
- **grazing angle**: see Wikipedia  Angle of incidence.
- **grayscale**: A weighted sum of the **RGB** values, computed for each pixel, that makes all pixels look varying shades of gray (see Section 3.3, Note 13; Wikipedia grayscale).
- **ground pixel resolution**: pixel width on the Earth's surface (see Wikipedia Ground sample distance).
- (image or chip) **heat map**: an analysis output **PNG** file showing alerted pixel values if varying levels of red and unalerted values in varying shades of gray )see Section 3.3, Note 14).
- **.h5*: an output file format produced by analysis option b, producing PixMin feature values that can be opened in spreadsheet form using **HDFView** (see Wikipedia Hierarchical Data Format; Appendix B).
- **HDFView**: a viewer that can open ***.h5** files in spreadsheet form (Wikipedia Hierarchical Data Format; Appendix B).
- **heat map**: an **analysis** output **image file** containing transformed image **criterion** values with **alerted chips** in a form that highlights the criterion's alert generation capability.
- **histogram:** see Wikipedia histogram).
- (Digital) **image**: a two-dimensional array of pixels that can readily be printed or shown by an image viewer.
- **hit ratio**: an **analysis** metric computed for each *chip_locations.csv* file entry and computed as the ratio of the chip's alert counts within a combination to the alert counts for the entire combination.
- **image file** (format): a file that is suitable for PixMin input by being in **JPEG**, **PNG**, or **JIF** format and having 24-bit color depth (see Wikipedia  color depth).
- **integer (metric)**: see Wikipedia Wikipedia Integer .
- **internal border removal**: When alerted chips overlap, chip **borders** inside the overlapping regions can obscure their content. As a workaround the PixMin ADK offers an internal border removal option. When that option is selected, PixMin removes internal chip borders while producing output alert maps.

- **IrfanView**: an image viewer, editor, organizer, and converter program (see Wikipedia IrfanView; Appendix B).
- **JPEG** (format): see Wikipedia JPEG.
- (triage) **level**: PixMin **triages** pixels during either **Level 1** processing only or Level 1 processing followed by Level 2 processing (see Section 3.3, Note 6).
- **lagged image differencing**: subtracting pixel values from each image that was shot a fixed number of time points ago from values in each current image.
- **machine learning (ML)**: see Wikipedia machine learning.
- **mask file**: a **.csv* file, inside **a mask folder**, containing binary masking values homologous to input image **pixel** values (see 7) in Appendix B).
- **mask file building tool**: an Excel spreadsheet that enables analysts to quickly build **mask files** (Appendix B).
- **mask folder**: a **config** subfolder containing **mask files** (see Appendix B)
- **mask region building tool**: see Appendix B.
- **mask region insertion tool**: see Appendix B.
- **masking**: a PixMin **config** option that forces PixMin to detect events only within allowable pixels, as specified by a **mask file**.
- (criterion) **matching value**: a value, computed at each **pixel location**, which reflects how closely a set of values for a template **combination** resembles its corresponding **ROI** values, relative to the location. PixMin computes matching values based on one among several configurable matching **criteria**.
- **ML**: **machine learning**.
- (frame) **mirroring**: flipping feature values at image borders into outlying frame values.
- **multi-core processor**: see Wikipedia, multi-core processing. PixMin can run on multi-core processers quickly, by running separate instances of PixMin on different processors at the same time.
- **normal output**: a collection of **image files** including **alert maps** and **chips**, along with **chip location files** and other text files that PixMin routinely provides.
- **normalization**: adjusting values measured on different scales to a notionally common scale --see Wikipedia, normalization (statistics).
- **optimizing** (tools): a variety of computing methods that may be use to maximize performance (*e.g.*, PixMin **precision**), subject to operational constraints (*e.g.*, low **SWaP**) see Wikipedia numerical optimization
- **orthomosaic (ORTHO)** (file): made by merging component image files Wikipedia orthophoto.
- **output alert centering**: centering output **chips** as well as their locations within **alert maps** around the point where the most **pixel alerts** occurred.
- **passthrough**: Using **criterion** values that were computed during **consecutive image processing** for **Level 1** as a basis for **Level 2** consecutive image processing.
- **pixel**: A point within an image represented by one or more **RGB** values. The PixMin Manual [1] also uses the term to describe **feature** values, **standardized** values, **masking** values, and region of interest **locations** at each point.

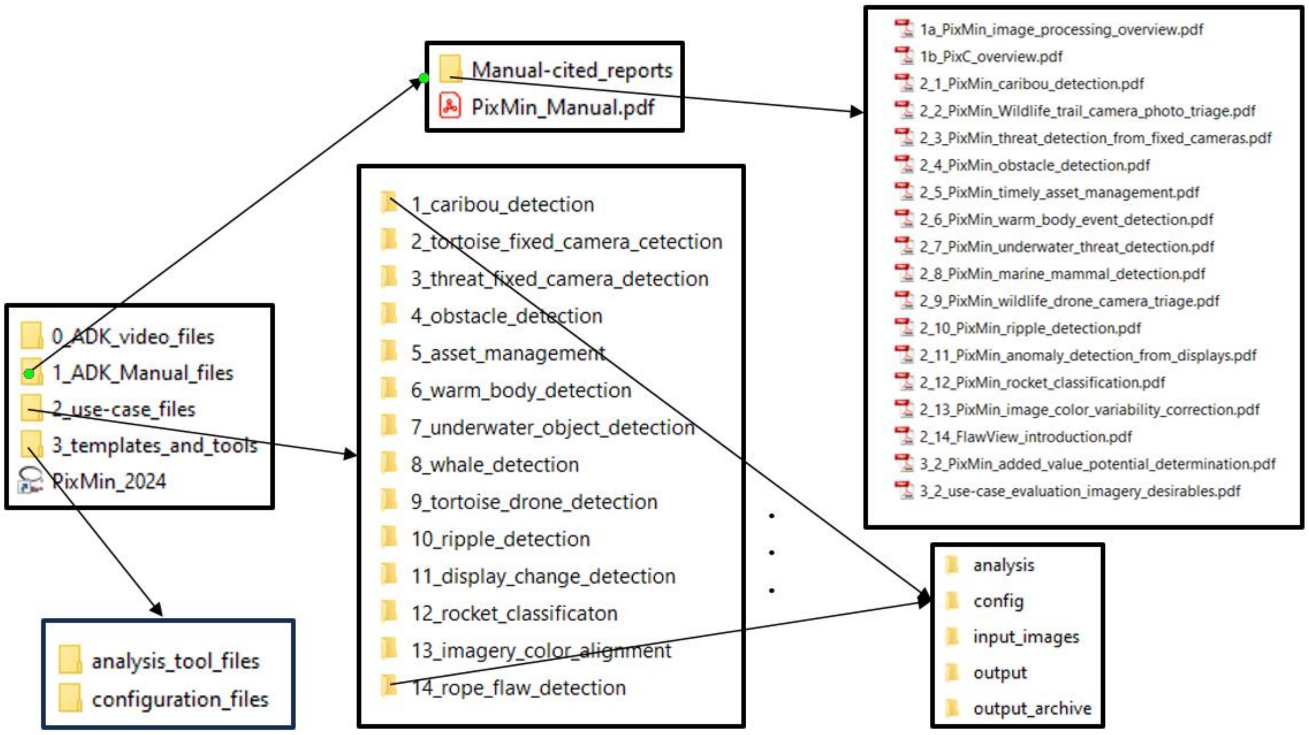
- **pixel alert**: a determination made by PixMin at each pixel **location** that its **matching** value has exceeded the **pixel alert cutoff** value.
- **pixel alignment**: A PixMin option that enables **input image pixels** to be aligned with corresponding **diff** image pixels. Change detection with pixel alignment can neutralize unchanged pixels more effectively than change detection without it.
- **pixel match cutoff** (value): a real-valued **metric**, set in the **config file**, that PixMin uses to identify **pixel level alerts**. For each template **combination** and at every pixel **location**, if a template matching **criterion** value exceeds its pixel match cutoff value, then PixMin marks the pixel at that location as a pixel level alert.
- **PixC**: a personal computer that contains the PixMin **ADK** along with a variety of data analysis and image processing computer applications that the **ADK** cites..
- **PixMin code**: Low level C code designed for fast operation in Windows and other computing environments on conventional as well as special purpose processors.
- **PixMin processor**: a software application that processes images one at a time to detect, highlight, and display **alerts**.
- **PNG** (format): see Wikipedia PNG.
- **precise** (event detection): classifying **regions of interest** that contain **targets** as **alerts** with high likelihood where they exist but classifying **ROI**s that do not contain targets with low likelihood. In statistical terms, **precise** detection must have high **sensitivity** and **specificity**, which can best be achieved when criterion values have low variability in pixel regions where targets are absent.
- **preliminary configuration**: the phase where PixMin checks for **input image** or **config** file errors and configures PixMin based on config file **metrics**, in preparation for **consecutive image processing**.
- (template version) **rotation degrees**: A value, specified for each **template type**, that **PixMin** uses to transform its corresponding **ROI** values, specified in the template versions file (see the Section 3.1 explanation of slide 11).
- **projection (**match): a **matching** value, computed as the sum of cross-products between values for a given **template** and homologous **feature** values within its corresponding **ROI** (see Section 3.3, Note 15).
- **.pxm.csv*: the basic PixMin **config** file extension Appendix C).
- **raw** (image format)**:** see Wikipedia raw image format. PixMin special-purpose use of raw image format can speed processing by up substantially because the format represents each quartet of Bayer pixels as only one **grayscale** number instead of 12 RGB numbers.
- **region of interest (ROI)**: a group of **color feature** values, centered at a **pixel location** that envelops all **color feature** values around the location that will be combined with **template combination** values to compute **criterion** values for the template combination at that location.
- (image) **resolution**: an image file pixel count, which governs PixMin run time as well as potential PixMin precision (see Wikipedia image resolution).
- **RGB**: a triplet of red, green, and blue color integer values, with each value ranging from 0 through 255. RGB values for each **pixel** may be found in any three-color **image file**.

- **Robust detection**: event detection that remains **precise** over changing brightness, contrast, or background conditions, between images as well as over different regions within images (see Wikipedia, Robust Statistics). PixMin incorporates **auto-adaptive processing**, **contrasts**, **differencing**, and **standardizing** to ensure robust detection.
- **ROI: region of interest**.
- (template version) **rotation degrees**: An integer valued **configuration metric** for each **template combination**, specified in *template_versions_level_1.csv* and *template_versions_level_2.csv* files, that **PixMin** uses to rotate each of its **Region of Interest** pixel locations around its center pixel location (see the Section 3.1 explanations of slides 11-12).
- (template version) **scale**: A positive valued **configuration metric** each **template combination**, specified in *template_versions_level_1.csv* and *template_versions_level_2.csv* files, that **PixMin** uses to expand the two-dimensional distance from each **ROI** pixel to its center pixel location (see the Section 3.1 explanations of slides 11-12).
- **sensitivity**: see Wikipedia sensitivity and specificity.
- **simple pixel feature values**: each such value as a weighted sum of its three **RGB** values, with weights specified by the *color_weights.csv* configuration file.
- **specificity**: see Wikipedia sensitivity and specificity.
- **standard correlation formula**: see Wikipedia correlation.
- **standard covariance formula**: see Wikipedia covariance.
- **standard deviation**: see Wikipedia standard deviation.
- **standard deviation match**: a **matching** value, computed as the **standard deviation** among **feature** values that are homologous to corresponding **ROI.**
- **standardizing**: transforming pixel values to their standard scores subtracting their mean from each of them and then dividing by their standard deviation (see Section 3.3, Note 9; Wikipedia standard score).
- **statistical contrast templates**: **templates** having values that sum algebraically to zero (see Section 3.3, Note 4; Wikipedia statistical contrasts).
- **statistical optimization**: configuring an event detection process so that it will produce acceptably high specificity, subject to meeting acceptable sensitivity requirements (see Wikipedia, Receiver Operating Characteristic).
- **sum of absolute differences (SAD) match**: a **matching** value, computed as the **sum of absolute differences** between the values for a given template and homologous feature **value**s within its **ROI**.
- **sum of absolute differences (SAD)**: see Wikipedia sum of absolute differences.
- **SWaP**: computing solution size, weight, and power: see Wikipedia power-to-weight ratio.
- **target** (events)**:** One or more **regions of interest** within **input image files** that PixMin **templates** have been made to highlight.
- **template**: a rectangular array of **pixel** values that has been configured to match corresponding pixel values within a **region of interest**.
- **template building tool**: see Appendix B.

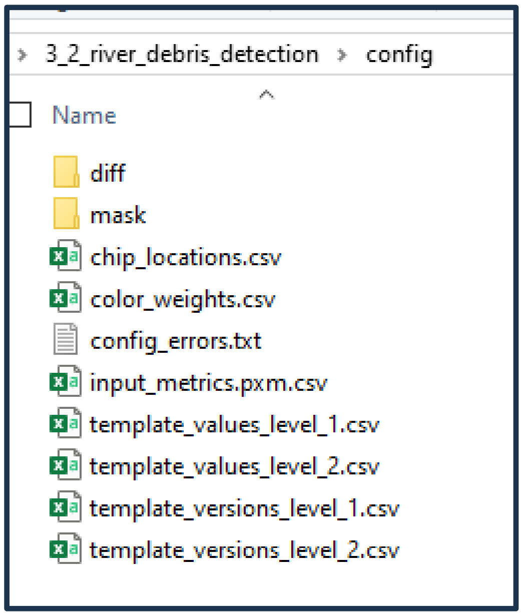
- **template centering**: subtracting mean values from template values and subtracting mean values from feature **ROI** values during "centered" **sum of absolute differences (SAD) matching**. As a result, centered SAD matching along automatically adjusts regions within **image files** for differences in brightness.
- **template combination**: a specific **template type** as specified in a *template_values_level_1.csv* file or its *template_values_level_2.csv* file, along with one of its specific **template versions** as specified in its corresponding *template_versions_level_1.csv* or *template_values_level_2.csv* file (see Section 3.3, Note 16).
- **template criterion value**: a **feature** value, computed at each **pixel** location as a **criterion-**based function of values for a **template combination** and **ROI color feature** values centered at that location.
- **template types**: A positive integer at the beginning **of template values** file (see Appendix B), specifying number of templates to be specified later in the file (see Section 3.3, Note 16; Appendix B).
- **template values file**: A file of the form **.template_values.csv* and located in the **config folder**. This file contains the number of **template types**, the rows and columns for each type, and values for each type (see Section 3.3, Note 16; Appendix B).
- **template versions file**: A **template** arrangement for any given **type**, specified by its **rotation degree** value and its **scale** value within a **.template_versions.csv* file. For each **template type**, this file contains the number of paired **rotation degrees** and **scale** values along with those values (see Section 3.3, Note 16; Appendix B).
- **triage**: Reduction of **pixels** within **image files** to pixels within **alerted blocks** during **Level 1** or **Level 2** processing at that location.
- (template) **types**: The number of distinct **Level 1** or Level 2 **templates**.
- (ADK) **use-case**: One of thirteen sensor processing applications, contained in your ADK. Each use-case includes its own aptly named ADK folder along with sections in this Manual that describe it.
- (template) **types**: The number of distinct **Level 1** or Level 2 **templates**.
- **variance**: see Wikipedia variance.
- **vector**: a one-dimensional **array**.
- **VGA (**image file): an **image file** that has 480 rows and 640 columns in keeping with its corresponding computer display standard (see Wikipedia Video Graphics Display).
- **video to JPG converter**: an application that extracts a configurable number of **JPG** files from video files (see Appendix B).

# Appendix B. ADK / PixC Components

## B.1. ADK / PixC folder structure


0_ADK_video_files
1_ADK_Manual_files
2_use-case_files
3_templates_and_tools
PixMin_2024
Manual-cited_reports
PixMin_Manual.pdf
1a_PixMin_image_processing_overview.pdf
1b_PixC_overview.pdf
2_1_PixMin_caribou_detection.pdf
2_2_PixMin_Wildlife_trail_camera_photo_triage.pdf
2_3_PixMin_threat_detection_from_fixed_cameras.pdf
2_4_PixMin_obstacle_detection.pdf
2_5_PixMin_timely_asset_management.pdf
2_6_PixMin_warm_body_event_detection.pdf
2_7_PixMin_underwater_threat_detection.pdf
2_8_PixMin_marine_mammal_detection.pdf
2_9_PixMin_wildlife_drone_camera_triage.pdf
2_10_PixMin_ripple_detection.pdf
2_11_PixMin_anomaly_detection_from_displays.pdf
2_12_PixMin_rocket_classification.pdf
2_13_PixMin_image_color_variability_correction.pdf
2_14_FlawView_introduction.pdf
3_2_PixMin_added_value_potential_determination.pdf
3_2_use-case_evaluation_imagery_desirables.pdf
1_caribou_detection
2_tortoise_fixed_camera_cetection
3_threat_fixed_camera_detection
4_obstacle_detection
5_asset_management
6_warm_body_detection
7_underwater_object_detection
8_whale_detection
9_tortoise_drone_detection
10_ripple_detection
11_display_change_detection
12_rocket_classificaton
13_imagery_color_alignment
14_rope_flaw_detection
analysis
config
input_images
output
output_archive
analysis_tool_files
configuration_files

The above figure shows your ADK folder structure, which should be mostly familiar to you, since you have been navigating through it while reading the Manual up to this point. The arrows point to sub-folders, stemming first from the parent folder on the left. You will find that folder, named *PixMin_ADK_2024*, with the *Documents* folder on your **PixC** computer. That folder includes the four ADK folders, along with a shortcut the the PixMin_2024 executable file. Each *use-case_files folder* has the same sub-folder structure shown on the lower left.


3_2_river_debris_detection > config
Name
diff
mask
chip_locations.csv
color_weights.csv
config_errors.txt
input_metrics.pxm.csv
template_values_level_1.csv
template_values_level_2.csv
template_versions_level_1.csv
template_versions_level_2.csv

The Figure on the right shows config folder contents for our river detection use-case. Among those, the *diff* folder, the *mask* folder, and the *chip_locations* file are optional. All others are required. You will find details on their contents explained for our river threat uses-case descriptions (see Sections 2.2 and 3.2), among others. You will find *input_metrics.pxm.csv* file details in Appendix C as well as throughout Sections 2 and 3.2.

## B2. Contents within the *configuration files* folder.

You will find the following files in your *configuration_files* folder. Since we have introduced them all previously, their contents should be self-explanatory with the exception of the *2_input_metrics.pxm.csv* file. You will find details on the contents of that file in Appendix C.

- *1_color_weights_[GRAYSCALE].csv*
- *2_input_metrics_[TEMPLATE]. pxm.csv* file.
- *3a_template_values_[SIMPLE_PROJECTION_LIGHT]_level_[#].csv*
- *3b_template_values_[SIMPLE_PROJECTION_DARK]_level_[#].csv*
- *4_template_versions_[SIMPLE]_level_ [#].csv*
- *5_chip_locations_[TEMPLATE].csv*

The folder also include an examples sub-folder containing copies of configuration files copied from selected use-case *config* folders.

**B3. Analysis Tools folder contents.**

You will find the following files in your *analysis_tool_files* folder. Since we have introduced them all previously, their contents should be self-explanatory.

- *1a_analysis_statistics_level_1_example.xlsx*
- *1b_analysis_statistics_level_2_example..xlsx*
- *2a_feature_array_reorienting_tool_257x257*
- *2b_feature_array_reorienting_tool_513x513.xlsx*
- *3a_mask_region_building_tool.xlsx*
- *3b_mask_region_insertion_tool.xlsx*
- *4_template_building_tool_example.xlsx*
- *5_drone_flight_plan_coverage_calculator.xlsx*
- *5a_alert_block_layout_4K_240_block_size.docx*
- *5b_alert_block_layout_4K_432_block_size.docx*
- *5c_alert_block_layout_4K_540_block_size.docx*
- *5d_alert_block_layout_metrics_4K.xlsx*

You will also find the following applications downloaded on your PixC computer.

- https://www.dvdvideosoft.com/products/dvd/Free-Video-to-JPG-Converter.htm
- https://www.videolan.org/vlc/
- https://www.irfanview.com/64bit.htm
- https://portal.hdfgroup.org/display/support/HDFView+3.1.3?

We use the https://clideo.com/merge-video converter to create **.mp4* files. The converter includes many useful options, most notably a wide range of output video file frame rates. We also use https://clideo.com/merge-mp4 and https://clideo.com/crop-video .

To make a side-by-side video, first click https://clideo.com/resources/how-to-make-side-by-side-video -- First open the link on the left, then open a new instance of the browser and then use it t open https://clideo.com/video-editor . Then upload the files to be shown side by side,, then click the + button for each clip on the left. Doing will get both frames into the main window. Then resize / move around as desired.

# Appendix C.  Configuration metrics glossary

As you have seen throughout this Manual, we have explained and introduced key configuration **metrics** located in a variety of ADK **.pxm.csv* files.   We have included details on each metrics below for quick reference.  We have numbered the items according to their required **.pxm.csv* file lines (see Appendix B, *2_input_metrics_[TEMPLATE].pxm.csv* file description).  Along with each line's standard description that you will find in all **.pxm.csv*, we have added operational details below that will help you determine each line's metric value.  You will find metric fields inside  < > delimiters, which show where the metrics belong.  The delimiters should not be included when you set their corresponding **.pxm.csv* file values.

1) input image folder < {input_metric_path}\..\input_images >.  This path points to a folder containing PixMin input image files.
2) input pixel rows < positive integer >.  This positive integer should be the same as the "Height" value, found by right-clicking the image file icon and then selecting Properties-Advanced.
3) input pixel columns < positive integer >.  This positive integer should be the same as the "Width" value, found by right-clicking the image file icon and then selecting Properties-Advanced.
4) configuration metrics folder < {input_metric_path} >.  This path points to a folder containing configuration files along with this **.pxm.csv* file.
5) color alignment < 0 or 1 >.  If set to one, PixMin performs image color alignment.  If set to zero, PixMin does not.
6) feature standardizing option < 0 or 1 >.  If set to one, PixMin **standardizes** feature values.  If set to zero, PixMin does not (see Section 3.3, Note 9).
7) feature masking option < 0 or 1 >.  If set to zero, PixMin does not perform **masking**.  If set to one, PixMin performs masking(see Section 3.3, Note 15).
8) feature diff option < non-negative integer >.  If set to to zero, PixMin does not perform **differencing**.  If set to one, PixMin performs differencing.
9) Level 1 alert block and output chip size < positive integer >.  This is the number of pixel rows and columns in each Level 1 **alert block** and output **chip** (see Section 3.3, Note 1).
10) Level 1 template matching basis < 0, 1, 2, 3, or 4 >.  Each pixel's Level 1 match value measures how close its surrounding values match its template according to its basis, which will be a **SAD** basis if set to zero, a **projection** basis if set to one, **a correlation** basis if set to two, a SAD with "centering" basis if set to three, and a **standard deviation** basis if set to four.
11) Level 1 pixel match cutoff value < numerical value >.  PixMin produces **Level 1 pixel alerts** when pixel's **standardized criterion values** exceed this cutoff value (see Section 3.3, Note 9). PixMin always standardizes criterion values within each template **combination** so that they will not be affected by combination criterion value mean or standard deviation differences.  That way, criterion values for each combination will be on the same scale and a single cutoff criterion value will impact alerting each combination in the same way.
12) Level 1 alert block cutoff value < positive integer >.  Each **Level 1 alert block's pixel level alert** count determines its alert status.   PixMin produces a Level 1 block alert if the block's pixel level alert count equals or exceeds this value.

13) Level 1 maximum alerts < positive integer >.  This is the maximum number of **Level 1 alerts** per image that PixMin will produce.  If the number of Level 1 alerts that PixMin has detected exceeds this number, PixMin will produce alerts that had the most alert counts within them.
14) Level 2 skip option < 0 or 1 >.  If set to one, PixMin will not attempt **Level 2** detection. As a result, output alerts will be determined only by **Level 1** detection.  If set to zero, PixMin will evaluate Level 2 alerts among Level 1 **alerted blocks**.
15) criteria pass-through option < 0 or 1 >.  If set to one, the **criterion** values calculated from Level 1 will be used as the input **feature values** for **Level 2**.  If set to zero, they will not.
16) Level 2 alert block size  < positive integer >.  This is the number of pixel rows and columns in each **Level 2 alert block**.  This number should evenly divide the Level 1 alert block size.
17) Level 2 template matching basis < 0, 1, 2, 3, or 4 >.  The same Level 1 template matching basis description in 10) above applies here.
18) Level 2 pixel match cutoff value < numerical value >.  The same Level 1 pixel match cutoff value description in 11) above applies here.
19) Level 2 alert block cutoff value < positive integer >.  The same Level 2 alert block cutoff value description in 11) above applies here.
20) Level 2 maximum alerts < positive integer >.   This is the maximum number of **Level 2 alerts** per image that PixMin will produce.
21) analysis < blank, **a**, **b**, **c**, or **d** >.  This metric determines output analysis data.  If left blank, PixMin produces detection output but no analysis output.  If set to a, b, or c, PixMin produces analysis output (see Section 3.3, Note 8).
22) output detection folder < {input_metric_path}\..\output >.  This path points to a folder containing PixMin output detection alert map files, chip files, and optional analysis files.
23) no normal output option < 0 or 1 >.  **Normal** image output may not be necessary when using some analysis options, in which case PixMin produces no normal image output when this metric is set to 1.  Otherwise, it should be set to zero.
24) output alert map border width <positive integer >.   PixMin distinguishes **alerted regions** within **alert maps** by surrounding the regions with borders.  This metric sets the border width, in pixels.  If alerted block boundaries fall on image borders, their alert blocks will be pushed inside image borders as necessary to allow fitting boundaries around them at the borders.
25) internal border removal option < 0 or 1 >.  If set to one, PixMin removes borders within overlapping alerted blocks in alert maps.  If set to zero, PixMin does not.