Mars 2020 EDL Image Processing

Descent Stage Down-Look Camera

Mars 2020 had 7 cameras for entry, descent, and landing (EDL). The Descent stage Downlook Camera was in the unique position to record images of the Mastcam-Z Radiometric Calibration Targets. This notebook contains transforms to “develop” raw image data into plausible-color images.

The transforms are produced by optimizing model parameters that transform samples of raw data to match corresponding colors on the primary Mastcam-Z Radiometric Calibration Target.

The principal model (`modelW10`) consists of an Input Device Transform (IDT) matrix that maps raw RGB to CIEXYZ values, and a black-body radiator spectral power distribution (SPD) illuminating the calibration target.

The secondary model (`modelW11`) consists of an IDT, with the CIE D65 standard illuminant SPD illuminating the calibration target.

Model parameters are fit by minimizing CIE2000 color distance between IDT transformed raw colors and calibration target colors produced from the illuminant SPD and calibration target reflectance data.

Process using `modelW10`,

- convert raw data to XYZ using IDT
- chromatic adaption from the model black-body illuminant to D65
- convert to “okLab” color space for saturation reduction in ares where raw data clipping. Clipping occurs in raw data due to over exposure of bright areas and camera recording format being 8-bit linear.
- convert from “okLab” back to XYZ and then to sRGB

Processing using `modelW11` is similar, however it does not include chromatic adaption step.

The Camera

The DDC camera detector model is Sony IMX265.

The actual camera model may be FLIR Chameleon3 CM3-U3-31S4C-CS https://www.flir.com/products/chameleon3-usb3/?model=CM3-U3-31S4C-CS

Notes from “TECHNICAL REFERENCE for FLIR CHAMELEON®3”:

8.4.5.2 For Resolution 2048 x 1536 Mode 0 RAW8 mode 55FPS, RAW16 31FPS, doesn’t show RAW12 option

8.13 Gama not available in RAW mode, doesn’t explicitly say LUT is not available

1.6 Least significant bits dropped

Figure 9 in (Maki et al 2020) shows frame rate of 12fps. In 8-bit 2048 x 1536 resolution raw mode, camera should have been able to operate at 55fps, so I assume lower rate due to data-handling/communication system rate limitations.

The Calibration Target

Reflectance measurements from (Buz el al. 2020) data are used to derive the targets colors from illumination SPDs. The Buz measurements are from “witness samples”. Per (Kinch et al. 2020) “Spectra from witness samples match spectra from the center of color and grayscale patches and match spectra acquired on grayscale rings well...”.

From “Page 7 of 51” of (Kinch et al. 2020):
The white ring and white patch are made from AluWhite98 sintered alumina provided by Avian technologies in the US (www.aviantechnologies.com). The rest (all colors, blacks, and greys) are made from glazed and matted aluminum-silicates provided by Lucideon in the UK (www.lucideon.com). The names of the 7 materials provided by Lucideon are Black, Grey33, Grey70, Cyan, Green, Bright Yellow and Red.

Prior to using the Buz spectra, reflectance of of grey targets was estimated as (1, .75, .4, .1) from (Kinch et al. 2020) Page 12/51 Fig 6. From Table 2 reflectance of of grey targets in visible light region was estimated as (1.01, .79, .43, .1)

References/Information Sources

Mars “RAW” images
https://mars.nasa.gov/mars2020/multimedia/raw-images/

“The Mars 2020 Engineering Cameras and Microphone on the Perseverance Rover: A Next-Generation Imaging System for Mars Exploration” https://doi.org/10.1007/s11214-020-00765-9

“Photometric characterization of Lucideon and Avian Technologies color standards including application for calibration of the Mastcam-Z instrument on the Mars 2020 rover” https://-doi.org/10.1117/1.OE.58.2.027108

“Dataset accompanying “Photometric characterization of Lucideon and Avian Technologies color

standards including application for calibration of the Mastcam-Z instrument on the Mars 2020 rover" https://doi.org/10.22002/D1.1154

"Radiometric Calibration Targets for the Mastcam-Z Camera on the Mars 2020 Rover Mission" https://-doi.org/10.1007/s11214-020-00774-8 https://link.springer.com/con-tent/pdf/10.1007/s11214-020-00774-8.pdf

"Perseverance Rover's Descent and Touchdown on Mars: Onboard Camera Views" JPL 4K EDL Video https://mars.nasa.gov/resources/25628/perseverance-rovers-descent-and-touchdown-on-mars-onboard-camera-views/

"TECHNICAL REFERENCE for FLIR CHAMELEON®3" https://flir.boxcn.net/s/xobncd08w5w3oc72tmvs33d-nttqfpjw9

"IMAGING PERFORMANCE SPECIFICATION FLIRCHAMELEON®3" https://flir.boxcn.net/s/62dooxqc-mu4chhxweyvp5wf9kocnliwr

"FLIR Machine Vision Cameras are Headed to Mars!" Blog entry https://www.flir.com/discover/iis/flir-machine-vision-cameras-are-headed-to-mars/
"FLIR Machine Vision Cameras Capture High-Definition Footage of NASA's Perseverance Rover Landing on Mars" Blog Entry https://www.flir.com/discover/iis/flir-machine-vision-cameras-capture-high-definition-footage-of-nasas-perseverance-rover-landing-on-mars/

"NOIP1SN1300A PYTHON 1.3/0.5/0.3 MegaPixels Global Shutter CMOS
Image Sensors" On Semi P1300 data-sheet for other EDL cameras
https://www.onsemi.com/pdf/datasheet/noip1sn1300a-d.pdf
(Response curve looks a lot like that for IMX264 sensor)

"FSM-IMX264 Datasheet" includes response curve for IMX264. Not sure how close this is to IMX265
https://www.framos.com/media/pdf/46/11/8e/FSM-IMX264_Datasheet_v1-0g_BriefsQt12dxNWYGtz.pdf

"The Cameras on the Mars 2020 Perseverance Rover" https://mars.nasa.gov/mars2020/space-craft/rover/cameras/
Mentions only 2 of 3 Parachute "up look" cameras successfully recorded the parachute.

ACES - DRAFT P-2013-001 : Recommended Procedures for the Creation and Use of Digital Camera System Input Device Transforms (IDTs) http://j.mp/P-2013-001 https://www.oscars.org/science-technol-ogy/aces/aces-documentation

Press Release: Avian Technologies Standards Land on Mars https://aviantechnologies.com/2021/02/21/-press-release-avian-technologies-standards-land-on-mars/

Additional Information

Early Navcam image of calibration target: “Mars Perseverance Sol 9: Left Navigation Camera (Navcam)” https://mars.nasa.gov/mars2020/multimedia/raw-images/NLE_0009_0667755636_926ECM_N0030000NCAM05000_13_0LLJ

“Mars 2020 Perseverance SHERLOC WATSON Camera Pre-delivery Characterization and Calibration Report”
https://www.researchgate.net/publication/345959204_Mars_2020_Perseverance_SHERLOC_WATSON_Camera_Pre-delivery_Characterization_and_Calibration_Report/link/5fb30ee945851518fdaca577/download

“Orientation to Perseverance’s Raw Image Data and Metadata” Emily Lakdawalla https://www.patreon.com/posts/orientation-to-48263650

“Decoding the Raw Publicly-Released Mastcam-Z Image Filenames” Jim Bell https://mastcamz.asu.edu/decoding-the-raw-publicly-released-mastcam-z-image-filenames/

Metadata/JSON for raw images discussions
http://www.unmannedspaceflight.com/index.php?showtopic=8603&view=findpost&p=250014
http://www.unmannedspaceflight.com/index.php?showtopic=8591&view=findpost&p=250010
https://twitter.com/nirajsanghvi/status/1363645689180291074?s=20
https://twitter.com/rover_18/status/1364309922167488512?s=20

Engineering (not EDL) Cameras sensor Model AMS CMV20000
Datasheet https://ams.com/documents/20143/36005/CMV20000_DS000440_2-00.pdf
(Linked from http://www.unmannedspaceflight.com/index.php?s=&showtopic=8603&view=findpost&p=250444
which incorrectly says are used by rover lookdown and lookup cameras)

“Mission to Mars: ams CMOS image sensors for pictures that are out of this world” Blog entry https://ams.com/sensor-blog/cmos-image-sensors-mars

Color Properties at the Mars InSight Landing Site https://doi.org/10.1029/2020EA001336 https://agupubs.onlinelibrary.wiley.com/doi/epdf/10.1029/2020EA001336

Color management and color science: Introduction http://www.normankoren.com/color_management.html

Vision Models for Wide Color Gamut Imaging in Cinema https://doi.org/10.1109/TPAMI.2019.2938499 https://ieeexplore.ieee.org/document/8901180

Pre-Flight Calibration of the Mars 2020 Rover Mastcam Zoom (Mastcam-Z) Multispectral, Stereoscopic Imager https://doi.org/10.1007/s11214-021-00795-x https://link.springer.com/article/10.1007/s11214-021-00795-x

The Mars 2020 Perseverance Rover Mast Camera Zoom (Mastcam-Z) Multispectral, Stereoscopic Imaging Investigation https://doi.org/10.1007/s11214-020-00755-x https://link.springer.com/article/10.1007%2Fs11214-020-00755-x

Cleanroom photos of Mars 2020
https://photojournal.jpl.nasa.gov/catalog/PIA23193
https://photojournal.jpl.nasa.gov/catalog/PIA23829
https://photojournal.jpl.nasa.gov/catalog/PIA23886
https://photojournal.jpl.nasa.gov/catalog/PIA23924
https://photojournal.jpl.nasa.gov/catalog/PIA23884

Solar Irradiation https://web.archive.org/web/20120702174159/http://www.crisp.nus.edu.sg/~research/tutorial/optical.htm

Introduction to Solar Radiation https://www.newport.com/t/introduction-to-solar-radiation

Color References

Color Matching Function Data http://www.cvrl.org/cmfs.htm

D65 Illuminant data http://www.cvrl.org/cie.htm in “Older CIE Standards” tab

Chromatic Adaptation equations http://www.brucelindbloom.com/Eqn_ChromAdapt.html

Black-body spectral distribution in Mathematica https://mathematica.stackexchange.com/questions/57389/convert-spectral-distribution-to-rgb-color

sRGB https://en.wikipedia.org/wiki/SRGB

RGB/XYZ conversion http://www.brucelindbloom.com/Eqn_RGB_XYZ_Matrix.html

“Two New von Kries Based Chromatic Adaptation Transforms Found by Numerical Optimization” S.Bianco,*R.Schettini http://dx.doi.org/10.1002/col.20573

https://www.researchgate.net/publication/227770362_Two_New_von_Kries_Based_Chromatic_Adaptation_Transforms_Found_by_Numerical_Optimization

OK Lab color space - “A perceptual color space for image processing” https://bottosson.github.io/posts/oklab/

An interactive review of Oklab https://raphlinus.github.io/color/2021/01/18/oklab-critique.html

sRGB ↔ linear RGB conversion in Mathematica https://mathematica.stackexchange.com/questions/15596/the-correct-way-to-linearize-colorspace-before-resizing-blurring-etc

XYZ ↔ CIE Lab conversion in Mathematica https://mathematica.stackexchange.com/questions/64986/how-to-replicate-v9s-color-conversion-to-lab-under-v10/105562#105562

D65 Illuminant https://en.wikipedia.org/wiki/Illuminant_D65
https://en.wikipedia.org/wiki/Standard_illuminant#Illuminant_series_D

CIE 15:2004 https://ia802802.us.archive.org/23/items/gov.law.cie.15.2004/cie.15.2004.pdf

CIELAB color space https://en.wikipedia.org/wiki/CIELAB_color_space

CIE XYZ color space https://en.wikipedia.org/wiki/CIE_1931_color_space

sRGB in ICC profiles https://www.w3.org/Graphics/Color/srgb

ACES - IDT Report v4 https://community.acescentral.com/uploads/short-url/2kdAkrmO79OIr1bU1S9J4eDEctC.pdf

Uniform colour spaces based on the DIN99 colour-difference formula https://www.researchgate.net/publication/229891006_Uniform_colour_spaces_based_on_the_DIN99_colour-difference_formula

Chromatic Adaptation Transform by Spectral Reconstruction https://doi.org/10.1002/col.22384 https://arxiv.org/pdf/1902.10160.pdf http://scottburns.us

Processing RAW images in Python http://dx.doi.org/10.13140/RG.2.2.21522.25287 https://www.researchgate.net/publication/314239357_Processing_RAW_images_in_Python

Academy Color Encoding System https://en.wikipedia.org/wiki/Academy_Color_Encoding_System

Colour Science & Digital Imaging https://github.com/colour-science

PhysLight https://github.com/wetadigital/physlight

A better “VisibleSpectrum” function? https://mathematica.stackexchange.com/questions/73161/a-better-visiblespectrum-function/73162#73162

ColorPy - A Python package for handling physical descriptions of color and light spectra http://markkness.net/colorpy/ColorPy.html

RGB COLOURSPACE TRANSFORMATION MATRIX https://www.colour-science.org:8010/apps/rgb_colourspace_transformation_matrix

Colour Python Package https://github.com/colour-science/colour

Colour Manual Bibliography https://colour.readthedocs.io/en/develop/bibliography.html#bianco2010a

free color converter https://www.nixsensor.com/free-color-converter/

Using the sRGB_v4_ICC_preference.icc profile https://color.org/ICC_White_Paper_26_Using_the_V4_sRGB_ICC_profile.pdf

dcraw source https://www.dechifro.org/dcraw/dcraw.c

ACES - Objective tests of the gamut mapper https://community.acescentral.com/t/objective-tests-of-the-gamut-mapper/3142

Getting To Know ACES https://mixinglight.com/color-grading-tutorials/getting-know-aces/

The Academy - Science & Technology Council https://github.com/ampas

Setup

In[•]:=
```
buzSpreadsheetFileName =
  "/Users/bswift/Documents/Mars/MCZ_Data_Packet/MCZ_Photom_Data.xlsx";
(* Downloaded from https://data.caltech.edu/records/1154 *)
```

In[•]:=
```
rawImagesDirectory = "/Users/bswift/Downloads/mars2020/raw/esf";
```

Note: processed images are written to a new directory path in which “frames” is substituted for “raw” in path `rawImagesDirectory`.

Confirm files present

In[•]:=
```
FileType[buzSpreadsheetFileName]
```
Out[•]=
```
File
```

In[•]:=
```
DirectoryQ[rawImagesDirectory]
```
Out[•]=
```
True
```

Image Processing Tools

Definitions and Functions

Data Tables Embedded in Notebook

CMF Data

Illuminant Data

Color Matching

In[•]:=
```
(* BlackBody emission at discrete CMF wavelengths normalized so Max == 1 *)
(* https://mathematica.stackexchange.com/questions/57389/convert-
  spectral-distribution-to-rgb-color *)
blackBodyDiscrete[t_] := Module[{h = 6.62607 * 10^-34,
   c = 2.998 * 10^8, k = 1.38065 * 10^-23}, {λXYZCMF[[1]],
   ((2 h c^2) / ((-1 + E^((h c / k) / (t (λXYZCMF[[1]] 10^-9)))) (λXYZCMF[[1]] 10^-9)^5)) //
    # / Max[#] &}]
```

In[•]:=
```
λXYZCMF = 1.0 lin2012xyz2eDownloaded[[ ;;  ;; 5]]ᵀ;
(* Tabular 1nm CMF reduced to working length 5nm. The wavelengths
 λ here are locations used throughout discrete operations *)
```

In[•]:=
```
blackBodyXYZ[t_] :=
 Module[{h = 6.62607 * 10^-34, c = 2.998 * 10^8, k = 1.38065 * 10^-23}, λXYZCMF[[2 ;; 4]].
    ((2 h c^2) / ((-1 + E^((h c / k) / (t (λXYZCMF[[1]] 10^-9)))) (λXYZCMF[[1]] 10^-9)^5)) //
   # / #[[2]] &] (* XYZ Color values for given temp. *)
```

In[•]:=
```
(* BlackBody emission at discrete CMF wavelengths normalized so Max == 1 *)
blackBodyDiscrete[t_] :=
 Module[{h = 6.62607 * 10^-34, c = 2.998 * 10^8, k = 1.38065 * 10^-23}, {λXYZCMF[[1]],
   ((2 h c^2) / ((-1 + E^((h c / k) / (t (λXYZCMF[[1]] 10^-9)))) (λXYZCMF[[1]] 10^-9)^5)) //
    # / Max[#] &}]
```

```
In[•]:= reflectanceToXYZDiscrete[reflectanceDiscrete_, illuminantDiscrete_] := Block[
          {normalization
           , xyzColorDerrived
          },
          normalization = Total[illuminantDiscrete[[2]] λXYZCMF[[3]];
          xyzColorDerrived = 100. / normalization
            (λXYZCMF[[2 ;; 4]].(reflectanceDiscrete[[2]] illuminantDiscrete[[2]]));
          xyzColorDerrived
         ]

In[•]:= d65XYZBL = {0.95047, 1.00000, 1.08883}
         (* From http://www.brucelindbloom.com/Eqn_ChromAdapt.html *)

Out[•]= {0.95047, 1., 1.08883}

In[•]:= d50XYZBL = {0.96422, 1.00000, 0.82521}
         (* From http://www.brucelindbloom.com/Eqn_ChromAdapt.html *)

Out[•]= {0.96422, 1., 0.82521}

In[•]:= illuminantD65IF = Interpolation[illuminantD65Table, InterpolationOrder → 1]
         (* CIE recommendations on using linear interpolation *)

Out[•]= InterpolatingFunction[ 
          
        ]

In[•]:= illuminantD65Discrete = {λXYZCMF[[1]], illuminantD65IF[λXYZCMF[[1]]]};
        (* illuminant resampled to common descrete wavelengths *)
```

XYZ to rgb

```
In[•]:= linearRGBtoXYZMat = .;
linearRGBtoXYZMat[{{xr_, yr_}, {xg_, yg_}, {xb_, yb_}}, {wpX_, wpY_, wpZ_}] := Block[{
   bigXr = xr / yr,
   bigYr = 1,
   bigZr = (1 - xr - yr) / yr,
   bigXg = xg / yg,
   bigYg = 1,
   bigZg = (1 - xg - yg) / yg,
   bigXb = xb / yb,
   bigYb = 1,
   bigZb = (1 - xb - yb) / yb
   , bigSr, bigSg, bigSb
  },
  {bigSr, bigSg, bigSb} = Inverse[{{bigXr, bigXg, bigXb}
      , {bigYr, bigYg, bigYb}
      , {bigZr, bigZg, bigZb}
     }].{wpX, wpY, wpZ};
  {{bigSr bigXr, bigSg bigXg, bigSb bigXb}
   , {bigSr bigYr, bigSg bigYg, bigSb bigYb}
   , {bigSr bigZr, bigSg bigZg, bigSb bigZb}
  }
 ]

In[•]:= srgbPrimaries = {{.6400, .3300}, {.3000, .6000}, {.1500, .0600}}
 (* from https://en.wikipedia.org/wiki/SRGB *)

Out[•]= {{0.64, 0.33}, {0.3, 0.6}, {0.15, 0.06}}

In[•]:= srgbPrimariesXYZ =
 {{0.436, 0.222, 0.014}, {0.385, 0.717, 0.097}, {0.143, 0.061, 0.714}}
 (* From ColorSync Utility display of Red/Green/Blue
  colorant tristimulus values for sRGB Profile.icc *)

Out[•]= {{0.436, 0.222, 0.014}, {0.385, 0.717, 0.097}, {0.143, 0.061, 0.714}}

In[•]:= linearRGBtoXYZMat[srgbPrimaries, d65XYZBL] //
 TraditionalForm (* Verified Matches values at http://
  www.brucelindbloom.com/Eqn_RGB_XYZ_Matrix.html *)

Out[•]//TraditionalForm=
```

$$\begin{pmatrix} 0.412456 & 0.357576 & 0.180437 \\ 0.212673 & 0.715152 & 0.072175 \\ 0.0193339 & 0.119192 & 0.950304 \end{pmatrix}$$

Chromatic Adaption

```
In[•]:= (catMatBS = {{0.8752`, 0.2787`, -0.1539`},
          {-0.8904`, 1.8709`, 0.0195`}, {-0.0061`, 0.0162`, 0.9899`}}) //
         TraditionalForm (* Chromatic Adaption Transfrom M_BS CAT from
          "Two New von Kries Based Chromatic Adaptation Transforms
            Found by Numerical Optimization" S.Bianco,*R.Schettini *)
```

Out[•]//TraditionalForm=

$$\begin{pmatrix} 0.8752 & 0.2787 & -0.1539 \\ -0.8904 & 1.8709 & 0.0195 \\ -0.0061 & 0.0162 & 0.9899 \end{pmatrix}$$

```
In[•]:= (catMatBradford = {{0.8951000, 0.2664000, -0.1614000}
        , {-0.7502000, 1.7135000, 0.0367000}
        , {0.0389000, -0.0685000, 1.0296000}
           }) // TraditionalForm (* Bradford Chromatic Adaption Transfrom from http://
          www.brucelindbloom.com/Eqn_ChromAdapt.html *)
```

Out[•]//TraditionalForm=

$$\begin{pmatrix} 0.8951 & 0.2664 & -0.1614 \\ -0.7502 & 1.7135 & 0.0367 \\ 0.0389 & -0.0685 & 1.0296 \end{pmatrix}$$

```
In[•]:= adaptMat[mSubA_, xyzWhiteSource_, xyzWhiteDestination_] := Block[{
           ργβSource = mSubA .xyzWhiteSource,
           ργβDestination = mSubA .xyzWhiteDestination
          },
          Inverse[mSubA].DiagonalMatrix[ργβDestination / ργβSource].mSubA
         ]
```

```
In[•]:= (adaptMat[catMatBradford, d65XYZBL, d50XYZBL]) //
         TraditionalForm (* verified matches table values at http://
          www.brucelindbloom.com/Eqn_ChromAdapt.html *)
```

Out[•]//TraditionalForm=

$$\begin{pmatrix} 1.04781 & 0.0228866 & -0.050127 \\ 0.0295424 & 0.990484 & -0.0170491 \\ -0.00923449 & 0.0150436 & 0.752132 \end{pmatrix}$$

Oklab Color Space

New take on La*b* color space from https://bottosson.github.io/posts/gamutclipping/

```
In[•]:= (okLabM1 = {{0.8189330101, 0.3618667424, -0.1288597137},
          {0.0329845436, 0.9293118715, 0.0361456387},
          {0.0482003018, 0.2643662691, 0.6338517070}}) // MatrixForm
```

Out[•]//MatrixForm=

$$\begin{pmatrix} 0.818933 & 0.361867 & -0.12886 \\ 0.0329845 & 0.929312 & 0.0361456 \\ 0.0482003 & 0.264366 & 0.633852 \end{pmatrix}$$

```
In[•]:= (okLabM2 = {{0.2104542553, 0.7936177850, -0.0040720468},
          {1.9779984951, -2.4285922050, 0.4505937099},
          {0.0259040371, +0.7827717662, -0.8086757660}}) // MatrixForm
```

Out[•]//MatrixForm=

$$\begin{pmatrix} 0.210454 & 0.793618 & -0.00407205 \\ 1.978 & -2.42859 & 0.450594 \\ 0.025904 & 0.782772 & -0.808676 \end{pmatrix}$$

```
In[•]:= xyzToOkLab[uXYZ_] := okLabM2.(CubeRoot[okLabM1.uXYZ (* lms *)] (* l'm's'*))
          (* Note CubeRoot needed because x^(1/3) produces the
           complex principal root which has imiginary values *)

In[•]:= Map[{#, xyzToOkLab[#]} &, {{.95, 1, 1.089}, {1, 0, 0}, {0, 1, 0}, {0, 0, 1}}]
        (* verification of results at https://bottosson.github.io/posts/oklab/ *)
Out[•]= {{{0.95, 1, 1.089}, {0.999969, -0.000258006, -0.000114998}},
         {{1, 0, 0}, {0.449932, 1.23571, -0.0190276}},
         {{0, 1, 0}, {0.921817, -0.671238, 0.263324}},
         {{0, 0, 1}, {0.152603, -1.415, -0.448927}}}

In[•]:= {-0.1288597137`, 0.0361456387`, 0.633851707`}^(1 / 3)
Out[•]= {0.252547 + 0.437424 i, 0.330637, 0.859005}

In[•]:= (0.25254710339095093` + 0.4374244143774772` i) ×
         (0.25254710339095093` + 0.4374244143774772` i) ×
         (0.25254710339095093` + 0.4374244143774772` i)
Out[•]= -0.12886 + 4.16334 × 10^-17 i

In[•]:= okLabM1I = Inverse[okLabM1]
Out[•]= {{1.22701, -0.5578, 0.281256},
         {-0.0405802, 1.11226, -0.0716767}, {-0.0763813, -0.421482, 1.58616}}

In[•]:= okLabM2I = Inverse[okLabM2]
Out[•]= {{1., 0.396338, 0.215804}, {1., -0.105561, -0.0638542}, {1., -0.0894842, -1.29149}}

In[•]:= okLabToXYZ[uLab_] := okLabM1I.((okLabM2I.uLab (* l'm's'*))^3 (* lms *))
```

```
In[•]:= Map[{#, okLabToXYZ[#]} &,
 {{1, 0, 0}, {.45, 1.236, -.019}, {.922, -.671, .263}, {.153, -1.415, -0.449}}]
```

```
Out[•]= {{{1, 0, 0}, {0.95047, 1., 1.0883}},
 {{0.45, 1.236, -0.019}, {1.0006, -7.98469 × 10^-6, -0.0000383248}},
 {{0.922, -0.671, 0.263}, {0.000304855, 1.0005, 0.000898033}},
 {{0.153, -1.415, -0.449}, {0.000589544, 0.0000571048, 1.00165}}}
```

Linear to sRGB

```
In[•]:= (* From Alexey Popkov answer at https://
 mathematica.stackexchange.com/questions/15596/the-correct-
  way-to-linearize-colorspace-before-resizing-blurring-etc *)
linear2srgb = Compile[{{Clinear, _Real, 1}}, With[{α = 0.055}, Table[
     Piecewise[{{12.92 * C, C ≤ 0.0031308}, {(1 + α) * C ^ (1 / 2.4) - α, C > 0.0031308}}],
     {C, Clinear}]], RuntimeAttributes → {Listable},
   CompilationTarget → "C", Parallelization → True];
```

```
In[•]:= (*linear2srgbImage[linearImg_Image]:=Image[
  linear2srgb[
   ImageData[
    ColorConvert[
     Image[linearImg,"Real"]
     ,"RGB"]
    ,Interleaving→True]
  ]
  ,ColorSpace→"RGB"
 ]*) (* old, conversion promoted greyscale to RGB which
 is incorrect for processing single channel greyscale textures *)
```

```
In[•]:= (*linear2srgbImage[linearImg_Image]:=Image[
  linear2srgb[
   ImageData[
    ImageClip[Image[linearImg,"Real"]]
    ,Interleaving→True]
  ]
  ,ColorSpace→ImageColorSpace[linearImg]
 ]*)
```

```
In[•]:= linear2srgbImage[linearImg_Image] := Image[
          linear2srgb[
           ImageData[
            ImageClip[Image[linearImg, "Real"]]
            , Interleaving → True]
          ]
          (*,ImageType[linearImg]*)
          (*,ColorSpace→ImageColorSpace[linearImg]*)
         ] (* add conversion back from real to origional type *)

In[•]:= linear2srgbImage[linearImg_Image] := Image[
          Image[
           linear2srgb[
            ImageData[
             ImageClip[Image[linearImg, "Real"]]
             , Interleaving → True]
           ]
           , ColorSpace → ImageColorSpace[linearImg]
          ]
          , ImageType[linearImg]
          (* conversion from "data" to Bit16 image not working as expected.
            Need to first conver to Real32 and then to Bit16. *)
         ](* add conversion back from real to origional type *)

In[•]:= (* minor speedup by loosing generality *)
        linear2srgbImageReal[linearImg_Image] := Image[
          linear2srgb[
           ImageData[
            linearImg
           ]]
          , ColorSpace → "RGB"
         ] (* WARNING: promotes to RGB *)

In[•]:= srgb2linear = Compile[{{Csrgb, _Real, 1}}, With[{α = 0.055},
            Table[Piecewise[{{C / 12.92, C ≤ 0.04045}, {((C + α) / (1 + α)) ^ 2.4, C > 0.04045}}],
             {C, Csrgb}]], RuntimeAttributes → {Listable}, Parallelization → True];

In[•]:= srgb2linearImage[linearImg_Image] := Image[
          srgb2linear[
           ImageData[
            ImageClip[Image[linearImg, "Real"]]
            , Interleaving → True]
          ]
          , ColorSpace → ImageColorSpace[linearImg]
         ]
```

Misc

```
In[•]:= imageInfo[image_] :=
          ImageMeasurements[image, {"Channels", "ColorSpace", "DataRange",
            "DataType", "Dimensions", "SampleDepth", "Transparency", "Min", "Max",
            "Mean", "Median", "StandardDeviation"}, Association];
```

Replace border pixels with black and add cross hairs through mid-lines and yellow mark in center. For image with even pixel sizes, the mid-lines and center mark be two-pixels across.

```
In[•]:= nbImg[img_Image] := ImageResize[img, {1024}, Resampling → "Linear"] (* resize image
          to not blow up Notebook. "Linear" to prevent negative values being created. *)

In[•]:= Attributes[nbImg] = {Listable}
Out[•]= {Listable}

In[•]:= nbcImg[img_Image] := ImageResize[ImageCrop[img], {1024}, Resampling → "Linear"]
         (* resize image to not blow up Notebook. "Linear"
          to prevent negative values being created. *)

In[•]:= nbcImg[img_Image] :=
         ImageResize[ImagePad[img, -(BorderDimensions[img, .005] - 8 {{1, 1}, {1, 1}})],
          {1024}, Resampling → "Nearest"(*"Linear"*)] (* resize image to not
          blow up Notebook. "Linear" to prevent negative values being created. *)

In[•]:= borderColor = RGBColor[.2, .2, .2];
        ClearAll[markImage];
        markImage[img_, inset_] := ReplaceImageValue[img
             , {{All, 1 / 2 + inset} → borderColor
              , {All, ImageDimensions[img][[2]] - 1 / 2 - inset} → borderColor
              , {1 / 2 + inset, All} → borderColor
              , {ImageDimensions[img][[1]] / 2, All} → RGBColor[.7, 0, .9]
              , {All, ImageDimensions[img][[2]] / 2} → RGBColor[.7, 0, .9]
              , {ImageDimensions[img][[1]] - 1 / 2 - inset, All} → borderColor
             }
             , DataRange → Full]  // ReplaceImageValue[#
              , {{1 / 2, 1 / 2} → {.9, .7, 0, 1}(* place marker at center of image,
               which colors 4 pixels when image dimensions are even *)
              }
              , DataRange → {{0, 1}, {0, 1}}] & ;
        SetAttributes[markImage, Listable]

In[•]:= (* debug print *)
        SetAttributes[dp, HoldAll];
        dp[s_] := Echo[Evaluate[s], ToString[Unevaluated[s]]]
```

```
zeroPadLeft[n_Integer, pad_Integer] :=
 IntegerString[n, 10, Max[pad, IntegerLength[n]]]
 (* IntegerString drops negative sign *)
```

```
zeroPadLeft[n_String, pad_Integer] :=
 StringPadLeft[n, Max[pad, StringLength[n]], "0"]
```

```
b2g[img_Image] := ColorReplace[img, Black → GrayLevel[0.25`], 0.05`/65 536]
 (* black to gray for visualizing masked images *)
```

PCA

```
(* PCA matricies for image with a
 maskValue to be ignored (excluded from analysis) *)
kldMatFromMultichannelImage[im_Image, maskValue_] := KarhunenLoeveDecomposition[
   Transpose[Select[Flatten[ImageData[im], 1], # ≠ maskValue &]]][[2]]
```

```
(* apply kld matricies to image, returns list of images *)
klChannels[imageNoMask_Image, kldMat_] :=
 Map[Image, (kldMat.Transpose[ImageData[imageNoMask], {2, 3, 1}])]
```

URL Download

In[•]:=
```
(* URLDownload[] does not honor content-disposition "attachment; filename",
so make something that works *)
(* could add validation of inputs XXX *)
(* Developed in NewDownloadDev.nb *)
(* mar2020 downloads don't have "content-disposition" in Headers so use
 last component of URL. XXX probably bug here if URL has extra bits *)
urlDownloadToAttachmentFilename[url_, directory_] := Block[
  {downloadResults
   , contentDisposition
   , attachmentFilename
   , desiredFilename
   , result
  },
  downloadResults = URLDownload[url, directory, All]; (* returns Associoation
   with keys {"StatusCode","Headers","File","Cookies","HTTPRequest"} *)
  contentDisposition = "content-disposition" /. downloadResults[["Headers"]];
  If[contentDisposition ≠ "content-disposition"
   ,
   attachmentFilename = StringReplace[contentDisposition,
      "attachment; filename=\"" ~~ fileName__ ~~ "\"" :> fileName];
   ,
   attachmentFilename = FileNameTake[url]
  ];
  desiredFilename = FileNameJoin[{DirectoryName[downloadResults[["File"]]],
     FileNameTake[attachmentFilename, -1]}];
  (*Print["RenameFile[",downloadResults[["File"]],",",desiredFilename,"]"];*)
  result = RenameFile[downloadResults[["File"]], desiredFilename];
  If[result ≠ $Failed
   , Print[desiredFilename, " downloaded"]
   , Print[result, " - RenameFile[",
    downloadResults[["File"]], ",", desiredFilename, "]"]
  ];
  result
 ]
```

Processing

Spectral Data for Mastcam-Z Calibration Targets

In[•]:=
```
AbsoluteTiming[buzSpreadsheet = Import[buzSpreadsheetFileName];
]
```

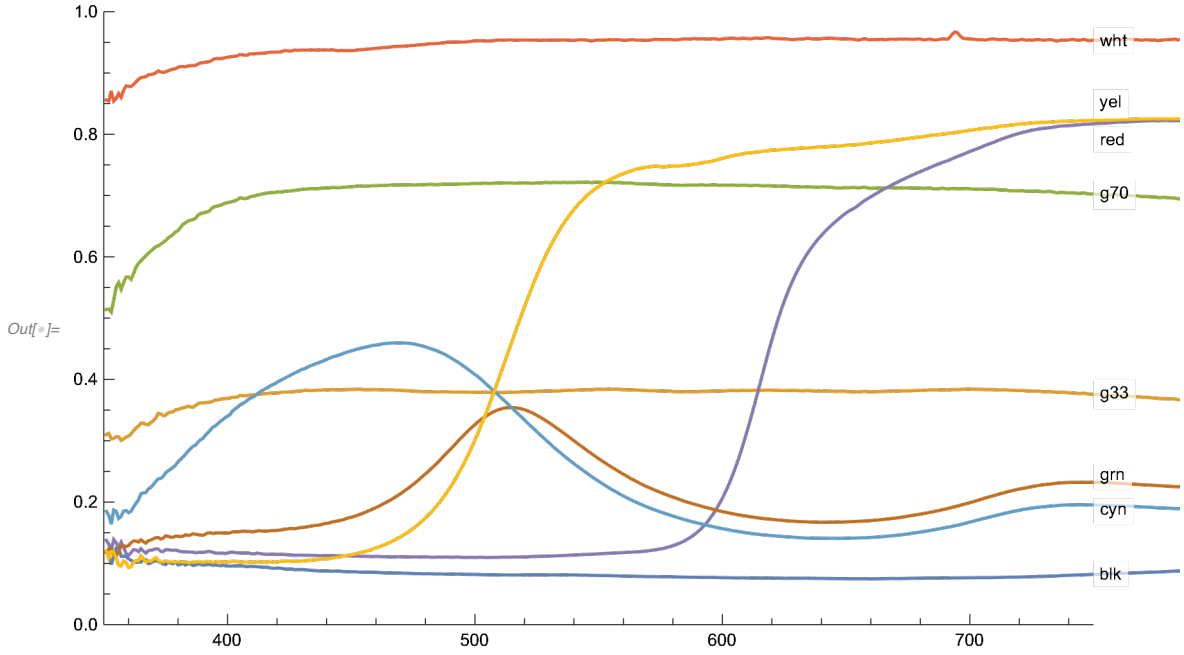
Out[•]=
```
{21.1935, Null}
```

In[•]:=
```
(* reformat imported spreadsheet table into Associoation *)
reformatBuzzContactProbeSheet[importedSpreadsheet_] := Block[{
   spectraTableT
   , wavelengths
  },
  spectraTableT = Transpose[importedSpreadsheet];
  wavelengths = spectraTableT[[1, 3 ;; -1]];
  Association[
   Map[#[[2]] → <|"reflectance" → Transpose[{wavelengths, #[[3 ;; -1]]}]
        , "reflectanceIF" → Interpolation[Transpose[{wavelengths, #[[3 ;; -1]]}]]
        (*,"CIEXYZ"→ #[[{-6,-5,-4}]]*)
        (*,"LAB"→  #[[{-3,-2,-1}]]*)
        , "Label" → #[[2]]
        , "Filename" → #[[1]]
        |> &, spectraTableT[[2 ;; -1]]]
  ]
 ]
```

```
In[•]:= (* reformat imported spreadsheet table into Associoation *)
reformatBuzzGoniometerSheet[importedSpreadsheet_] := Block[{
   spectraTableT
   , wavelengths
  },
  spectraTableT = Transpose[importedSpreadsheet];
  wavelengths = spectraTableT[[1, 8 ;; -1]];
  Association[
   Map[#[[1]] → <|"reflectance" → Transpose[{wavelengths, #[[8 ;; -1]]}]
        , "reflectanceIF" → Interpolation[Transpose[{wavelengths, #[[8 ;; -1]]}]]
        (*,"CIEXYZ"→ #[[{-6,-5,-4}]]*)
        (*,"LAB"→  #[[{-3,-2,-1}]]*)
        , "Filename" → #[[1]]
        , "number" → #[[2]]
        , "Label" → #[[3]]
        , "em" → #[[4]]
        , "in" → #[[5]]
        , "az" → #[[6]]
        |> &, spectraTableT[[2 ;; -1]]]
  ]
 ]

In[•]:= AbsoluteTiming[
 spectraTables = Association[
    (*"Grey450"→ reformatSectraTable[spectraTable1]
    ,"Color450"→ reformatSectraTable[spectraTable2]
    ,"ColorSPEX"→ reformatSectraTable[spectraTable3]
    ,"ColorSPIN"→ reformatSectraTable[spectraTable3]
    ,"BuzzContactProbe"→ reformatBuzzContactProbeSheet[buzSpreadsheet[[8]]]
    ,*)"BuzzFullIn" → reformatBuzzGoniometerSheet[buzSpreadsheet[[1]]]
   ];
]

Out[•]= {0.19893, Null}

In[•]:= calTargetSpectraSelected =
  Map[Function[{colorLabel}, Select[spectraTables["BuzzFullIn"],
     #["em"] == 0. && #["in"] == 30. &&#["in"] == 30. &&#["Label"] == colorLabel &]],
   {"blk", "g33", "g70", "wht", "red", "grn", "cyn", "yel"(*,"spc1","spc2"*)}
  ];(* darkToLightRGCY *)
(* Selecting samples with geometry that might
  be close to calibration image lighting. Guess-timated *)
```

In[•]:=
```
ListLinePlot[
 Labeled[#["reflectance"], #["Label"]] & /@
  Flatten[Values[calTargetSpectraSelected]]
 , PlotRange → {{350, 750}, {0, 1}}
 , ImageSize → Large
]
```

Out[•]=



Color Transformation Model Development

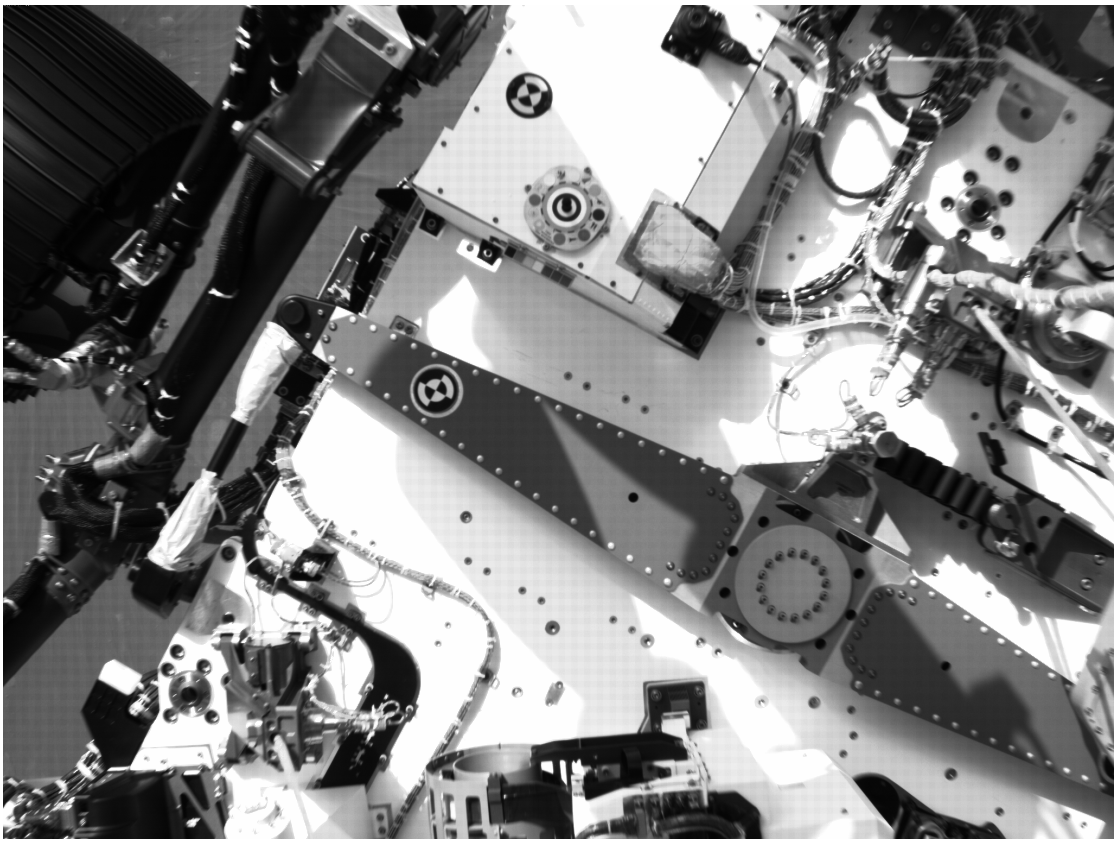
In[•]:=
```
imageFileNames = FileNames["E*ECM*.png", {rawImagesDirectory}];
(* Descent down look raw files *)
```

In[•]:=
```
calibrationImageFileName =
 First@Select[imageFileNames, StringMatchQ[__ ~~ "ESF_0042_0670681098_128" ~~ __]]
  (* ESF used for color calibration target *)
```

Out[•]=
```
/Users/bswift/Downloads/mars2020/raw/esf/ESF_0042_0670681098_128
 ECM_N0031392EDLC00042_0020LUJ01.png
```

In[•]:=
```
rawImageRGB = Import[calibrationImageFileName]
```

Out[•]=



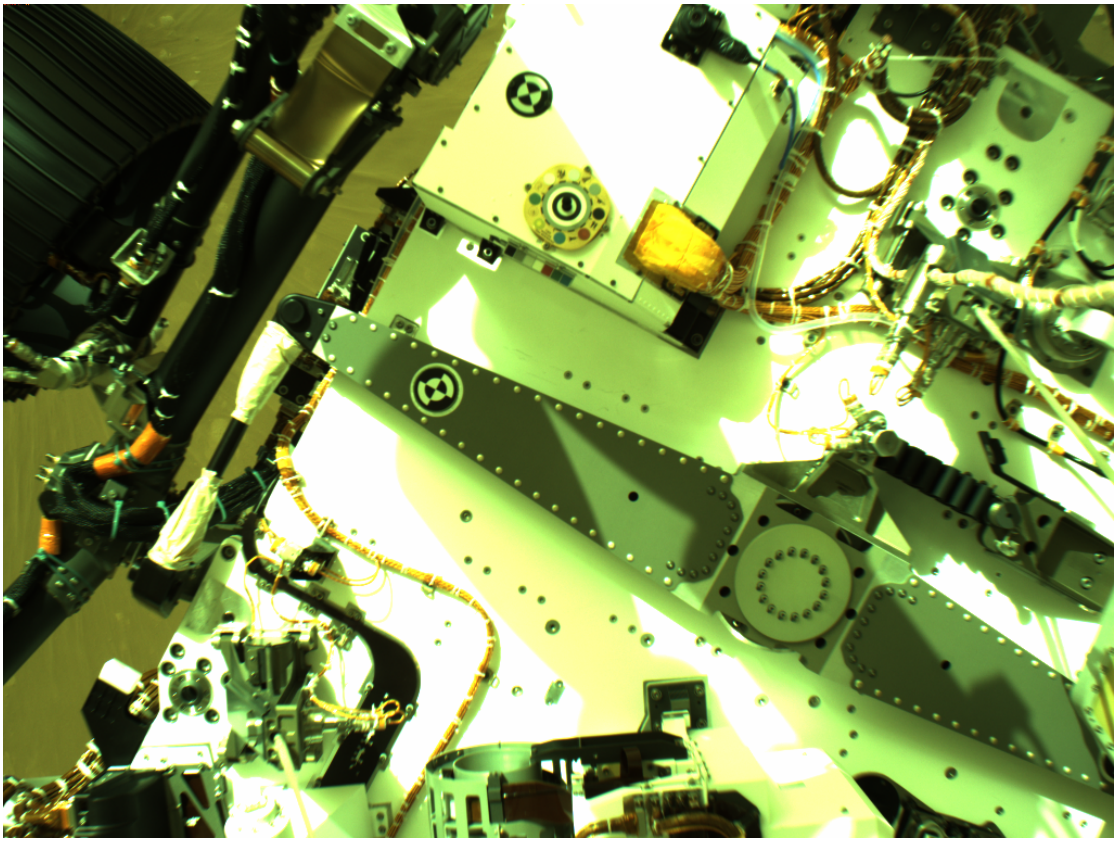
In[•]:=
```
csRawImage = ColorSeparate[rawImageRGB];
```

In[•]:=
```
(* rawImage to uncorrected color image.
  Currently only using 1 green of RGGB Bayer patern.
  MM ImageDemosaic functio not documented enough to be useful.
*)
debayerPlane1 = ImageTake[csRawImage[[1]], {1, -1, 2}, {1, -1, 2}]; (* Red *)
debayerPlane2 = ImageTake[csRawImage[[3]], {2, -1, 2}, {2, -1, 2}]; (* Blue *)
debayerPlane3 = ImageTake[csRawImage[[2]], {2, -1, 2}, {1, -1, 2}]; (* Green *)
debayerPlane4 = ImageTake[csRawImage[[2]], {1, -1, 2}, {2, -1, 2}];(* Green *)
```

In[•]:=
```
debayerPlanes = {debayerPlane1, debayerPlane2, debayerPlane3, debayerPlane4};
```

In[•]:=
```
debayerPlanesImage = ColorCombine[
  {debayerPlane1, 0.5 (debayerPlane3 + debayerPlane4), debayerPlane2}, "RGB"]
```

Out[•]=



Color Targets

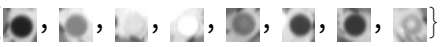
In[•]:=
```
darkToLightRGCY = {{548.361328125`, 575.390625`}, {545.005859375`, 597.822265625`},
   {525.638671875`, 610.7890625`}, {503.837890625`, 607.126953125`},
   {535.658203125`, 555.798828125`}, {489.93359375`, 588.14453125`},
   {512.962890625`, 552.681640625`}, {493.94140625`, 565.11328125`}};
 (* Primary target samples ordered dark to light then Red Green Cyan
 Yellow. Note two lightest samples experiencing cllipping of Red and
 Green.  "ESF_0042_0670681098_128ECM_N0031392EDLC00042_0020LUJ01.png"*)
```

In[•]:=
```
discsRoughCenterPane1 = darkToLightRGCY
```

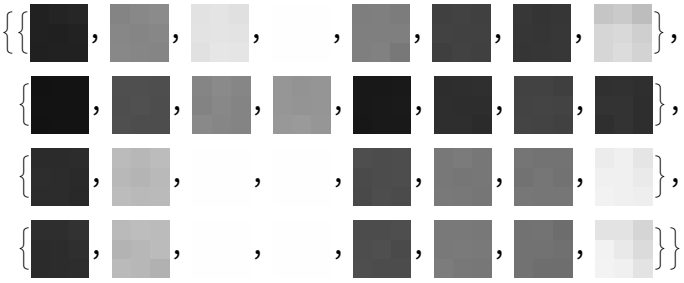
Out[•]=
```
{{548.361, 575.391}, {545.006, 597.822}, {525.639, 610.789}, {503.838, 607.127},
 {535.658, 555.799}, {489.934, 588.145}, {512.963, 552.682}, {493.941, 565.113}}
```

In[•]:=
```
cutOut[image_Image, {x_, y_}, {columns_, rows_}] := ImageTake[image
  , (ImageDimensions[image][[2]] - Floor[y]) + {-1, 1} Ceiling[(rows - 1) / 2]
  , Floor[x] + {-1, 1} Ceiling[(columns - 1) / 2]
 ]
```

```
In[•]:= Map[cutOut[debayerPlane1, # + {1, 0}, {1, 1} + 15] &, discsRoughCenterPane1]
```

Out[•]= 

```
In[•]:= cutoutsPlaneChip =
         Outer[cutOut[#1, #2 + {1, 0} + {1, 1}, {1, 1} + 2(* up to 8 for vertical
             chips avoiding shadow *)(* up to 11 for horizontal chips *)
           ] &, debayerPlanes, discsRoughCenterPane1, 1] (* from each color
         plane extract calibration chip image around RoughCenter points *)
```

Out[•]= 

```
In[•]:= Map[Times @@ ImageDimensions[#] &, cutoutsPlaneChip, {2}]
```

```
Out[•]= {{9, 9, 9, 9, 9, 9, 9, 9}, {9, 9, 9, 9, 9, 9, 9, 9},
         {9, 9, 9, 9, 9, 9, 9, 9}, {9, 9, 9, 9, 9, 9, 9, 9}}
```

```
In[•]:= darkToLightPlanes = Map[TrimmedMean[#, 2 / Times @@ ImageDimensions[#]] &,
          cutoutsPlaneChip, {2}] (* RBGG *)
```

```
Out[•]= {{0.130196, 0.527843, 0.89098, 0.996078, 0.497255, 0.254118, 0.211765, 0.821961},
         {0.0745098, 0.306667, 0.524706, 0.586667, 0.0964706, 0.178824, 0.261176, 0.189804},
         {0.17098, 0.727059, 0.996078, 0.996078, 0.298824, 0.46902, 0.457255, 0.936471},
         {0.176471, 0.72549, 0.996078, 0.996078, 0.300392, 0.473725, 0.442353, 0.905098}}
```

```
In[•]:= Map[(ImageData[255 #] // MatrixForm) &, cutoutsPlaneChip, {2}]
```

Out[•]=

$$\left\{\left\{\begin{pmatrix}33. & 36. & 38.\\ 33. & 32. & 34.\\ 32. & 33. & 33.\end{pmatrix}, \begin{pmatrix}132. & 136. & 139.\\ 135. & 133. & 135.\\ 136. & 134. & 133.\end{pmatrix}, \begin{pmatrix}226. & 228. & 224.\\ 227. & 228. & 227.\\ 225. & 230. & 229.\end{pmatrix}, \begin{pmatrix}254. & 254. & 254.\\ 254. & 254. & 254.\\ 254. & 254. & 254.\end{pmatrix},\right.\right.$$

$$\left.\begin{pmatrix}126. & 127. & 124.\\ 127. & 127. & 127.\\ 127. & 129. & 120.\end{pmatrix}, \begin{pmatrix}64. & 66. & 64.\\ 66. & 65. & 65.\\ 63. & 66. & 64.\end{pmatrix}, \begin{pmatrix}55. & 52. & 54.\\ 54. & 53. & 55.\\ 53. & 54. & 55.\end{pmatrix}, \begin{pmatrix}197. & 202. & 189.\\ 214. & 217. & 207.\\ 214. & 220. & 211.\end{pmatrix}\right\},$$

$$\left\{\begin{pmatrix}18. & 19. & 19.\\ 19. & 19. & 19.\\ 18. & 19. & 19.\end{pmatrix}, \begin{pmatrix}79. & 79. & 78.\\ 78. & 80. & 77.\\ 79. & 77. & 77.\end{pmatrix}, \begin{pmatrix}133. & 138. & 134.\\ 131. & 136. & 132.\\ 137. & 134. & 132.\end{pmatrix}, \begin{pmatrix}150. & 147. & 149.\\ 150. & 148. & 149.\\ 152. & 154. & 150.\end{pmatrix},\right.$$

$$\left.\begin{pmatrix}24. & 24. & 25.\\ 24. & 25. & 26.\\ 23. & 25. & 25.\end{pmatrix}, \begin{pmatrix}45. & 46. & 47.\\ 46. & 45. & 45.\\ 46. & 46. & 43.\end{pmatrix}, \begin{pmatrix}66. & 66. & 63.\\ 67. & 68. & 67.\\ 68. & 67. & 65.\end{pmatrix}, \begin{pmatrix}48. & 49. & 47.\\ 51. & 51. & 46.\\ 47. & 51. & 46.\end{pmatrix}\right\},$$

$$\left\{\begin{pmatrix}43. & 43. & 44.\\ 45. & 44. & 42.\\ 44. & 44. & 41.\end{pmatrix}, \begin{pmatrix}187. & 183. & 188.\\ 184. & 180. & 185.\\ 188. & 185. & 186.\end{pmatrix}, \begin{pmatrix}254. & 254. & 254.\\ 254. & 254. & 254.\\ 254. & 254. & 254.\end{pmatrix}, \begin{pmatrix}254. & 254. & 254.\\ 254. & 254. & 254.\\ 254. & 254. & 254.\end{pmatrix},\right.$$

$$\left.\begin{pmatrix}78. & 76. & 77.\\ 74. & 76. & 77.\\ 73. & 77. & 75.\end{pmatrix}, \begin{pmatrix}120. & 123. & 119.\\ 120. & 118. & 119.\\ 121. & 120. & 118.\end{pmatrix}, \begin{pmatrix}117. & 115. & 115.\\ 114. & 118. & 114.\\ 119. & 118. & 118.\end{pmatrix}, \begin{pmatrix}236. & 240. & 229.\\ 240. & 240. & 234.\\ 242. & 240. & 238.\end{pmatrix}\right\},$$

$$\left\{\begin{pmatrix}46. & 46. & 50.\\ 43. & 44. & 47.\\ 43. & 44. & 45.\end{pmatrix}, \begin{pmatrix}186. & 189. & 187.\\ 179. & 183. & 187.\\ 186. & 183. & 177.\end{pmatrix}, \begin{pmatrix}254. & 254. & 254.\\ 254. & 254. & 254.\\ 254. & 254. & 254.\end{pmatrix}, \begin{pmatrix}254. & 254. & 254.\\ 254. & 254. & 254.\\ 254. & 254. & 254.\end{pmatrix},\right.$$

$$\left.\left.\begin{pmatrix}76. & 76. & 78.\\ 77. & 79. & 74.\\ 78. & 76. & 76.\end{pmatrix}, \begin{pmatrix}121. & 121. & 120.\\ 121. & 122. & 121.\\ 124. & 120. & 119.\end{pmatrix}, \begin{pmatrix}114. & 115. & 109.\\ 114. & 114. & 112.\\ 114. & 110. & 110.\end{pmatrix}, \begin{pmatrix}227. & 227. & 213.\\ 242. & 233. & 219.\\ 243. & 240. & 227.\end{pmatrix}\right\}\right\}$$

Multi-parameter Optimization

```
In[•]:= calTargetSpectraSelected[[All, 1, "Label"]]
```

Out[•]= {blk, g33, g70, wht, red, grn, cyn, yel}

```
In[•]:= calTargetRAWRGB =
         RGBColor @@@ Transpose[{darkToLightPlanes[[1]], Mean[darkToLightPlanes[[{3, 4}]]],
            darkToLightPlanes[[2]]}] (* darkToLightRGCY *)
```

Out[•]= {■, ■, ■, ■, ■, ■, ■, ■}

```
In[•]:= calTargetSpectraDiscrete = Map[{λXYZCMF[[1]], #[[1, "reflectanceIF"]][λXYZCMF[[1]]]} &
           , calTargetSpectraSelected];
         (* resample calibration target spectra to color matching function wavelengths *)
```

```
In[•]:= calTargetRAWRGBWeights = {1, 1, .01, .01, .5, .5, .5, .4}(* darkToLightRGCY *)
```

Out[•]= {1, 1, 0.01, 0.01, 0.5, 0.5, 0.5, 0.4}

In[•]:=
```
(* modelW10 - IDT and black-body illuminant temp. *)
(* IDT model from DRAFT P-2013-001:
   Recommended Procedures for the Creation and Use of Digital Camera
    System Input Device Transforms (IDTs) http://j.mp/P-2013-001 https://
 www.oscars.org/science-technology/aces/aces-documentation *)

modelW10 =.;
modelW10[
  β1_?NumericQ, β2_, β3_, β4_, β5_, β6_ (* IDT *)
  , bands_
  , weights_ (* for each sample *)
 ] :=
 Block[
  {illuminantDiscrete = blackBodyDiscrete[bands[[1]]],
   optimizedIlluminantXYZ
  },
  (*optimizedIlluminantXYZ=(λXYZCMF[[2;;4]].illuminantDiscrete[[2]])//#/#[[2]]&;*)
  RootMeanSquare[
   WeightedData[
    MapThread[
     ColorDistance[#1, #2, DistanceFunction → "CIE2000" ] &
     , {(XYZColor[
           ({{β1, β2, 1 - β1 - β2}
                , {β3, β4, 1 - β3 - β4}
                , {β5, β6, 1 - β5 - β6}
               }.{ #[[1]]
                , #[[2]]
                , #[[3]]
               })[[1 ;; 3]]
         ] & /@ calTargetRAWRGB)
       , (XYZColor[.01 #] & /@ ( Map[reflectanceToXYZDiscrete[#, illuminantDiscrete] &,
            calTargetSpectraDiscrete]))}],
    weights]
  ]
 ]
```

In[•]:=
```
(* modelW11 - IDT assuming D65 illuminant. *)
modelW11 =.;
modelW11[
  β1_?NumericQ, β2_, β3_, β4_, β5_, β6_ (* IDT *)
  , weights_ (* for each sample *)
 ] :=
 Block[
  {illuminantDiscrete = illuminantD65Discrete,
   optimizedIlluminantXYZ
  },
  (*optimizedIlluminantXYZ=(λXYZCMF[[2;;4]].illuminantDiscrete[[2]])//#/#[[2]]&;*)
  RootMeanSquare[
   WeightedData[
    MapThread[
     ColorDistance[#1, #2, DistanceFunction → "CIE2000" ] &
     , {(XYZColor[
           ({{β1, β2, 1 - β1 - β2}
             , {β3, β4, 1 - β3 - β4}
             , {β5, β6, 1 - β5 - β6}
            }.{ #[[1]]
             , #[[2]]
             , #[[3]]
            })[[1 ;; 3]]
         ] & /@ calTargetRAWRGB)
       , (XYZColor[.01 #] & /@ ( Map[reflectanceToXYZDiscrete[#, illuminantDiscrete] &,
           calTargetSpectraDiscrete]))}],
    weights]
  ]
 ]
```

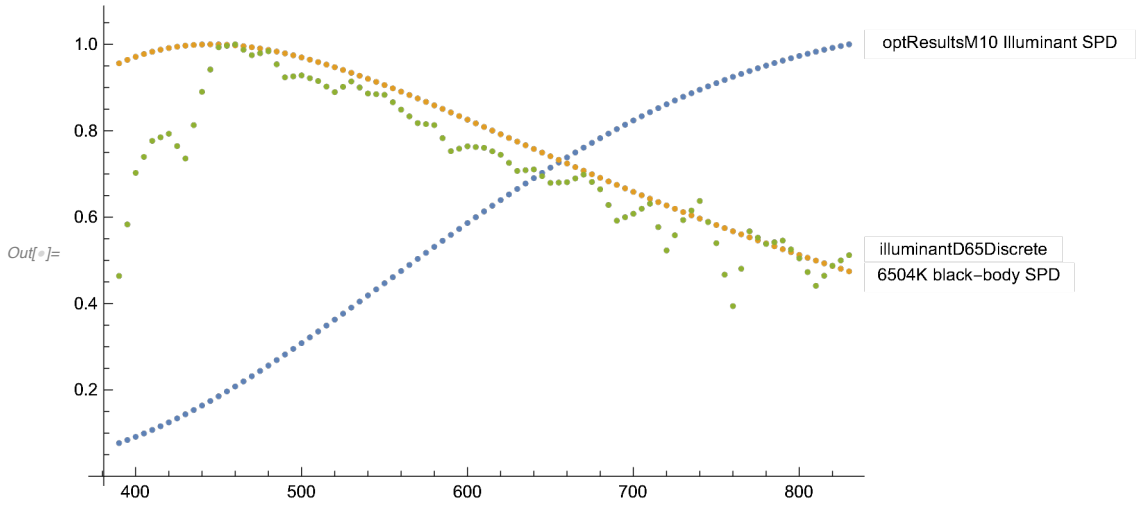
In[•]:=
```
modelW10[1, 0
 , 0, 1
 , 0, 0
 , {5777.} (* Solar AM0 *)
 , calTargetRAWRGBWeights
]
```

Out[•]= 0.298783

```
In[•]:= modelW11[1, 0
         , 0, 1
         , 0, 0
         , calTargetRAWRGBWeights
        ]
Out[•]= 0.308398

In[•]:= AbsoluteTiming[
         optResultsM11 = NMinimize[
           {
            modelW11[
             β1, β2, β3, β4, β5, β6
             , calTargetRAWRGBWeights]
           }
           , {β1, β2, β3, β4, β5, β6
           }
           , Method → {"RandomSearch", "SearchPoints" → 30}
          ]
        ]
Out[•]= {16.3833, {0.113158, {β1 → 0.415247, β2 → -0.0741459,
           β3 → 0.216901, β4 → 0.106853, β5 → -0.0462251, β6 → 0.167533}}}

In[•]:= AbsoluteTiming[
         optResultsM10 = NMinimize[
           {
            modelW10[
             β1, β2, β3, β4, β5, β6
             , {b1}
             , calTargetRAWRGBWeights]
            , 1000 < b1 < 10 000
           }
           , {β1, β2, β3, β4, β5, β6
            , b1
           }
           , Method → {"RandomSearch", "SearchPoints" → 30}
          ]
        ]
Out[•]= {55.1468, {0.0539836, {β1 → 0.636906, β2 → -0.0370224, β3 → 0.337894,
           β4 → 0.0735785, β5 → 0.123875, β6 → -0.351916, b1 → 3086.25}}}
```

```
In[•]:= ListPlot[{Labeled[
          blackBodyDiscrete[b1 /. optResultsM10[[2]]]^T, "optResultsM10 Illuminant SPD"]
        , Labeled[blackBodyDiscrete[6504.]^T, "6504K black-body SPD"]
        , Labeled[
         ({#[[1]], #[[2]] / Max[#[[2]]]} &@illuminantD65Discrete)^T, "illuminantD65Discrete"]}
       , ImageSize → Large]
```

Out[•]=



Color De-saturate clipping pixels

```
In[•]:= (* Blue channel on IMX264 is less sensitive than Red and Green,
         so doesn't clip until much higher brightness, so can be used to modulate
          final image saturation in areas where Red or Green raw data is clipping *)
```

```
In[•]:= saturatedMaskPlanes = (Binarize[#, 253.99 / 255] & /@ debayerPlanes);
        allSaturatedMask = ImageMultiply @@ saturatedMaskPlanes;
        noneSaturatedMask = ImageMultiply @@ (1 - saturatedMaskPlanes);
        someSaturatedMask = (1 - noneSaturatedMask) × (1 - allSaturatedMask);
```

```
In[•]:= x =.
```

```
In[•]:= deSaturateFactorLM = LinearModelFit[{{.5, 1}, {.98, 0}}, x, x]
```

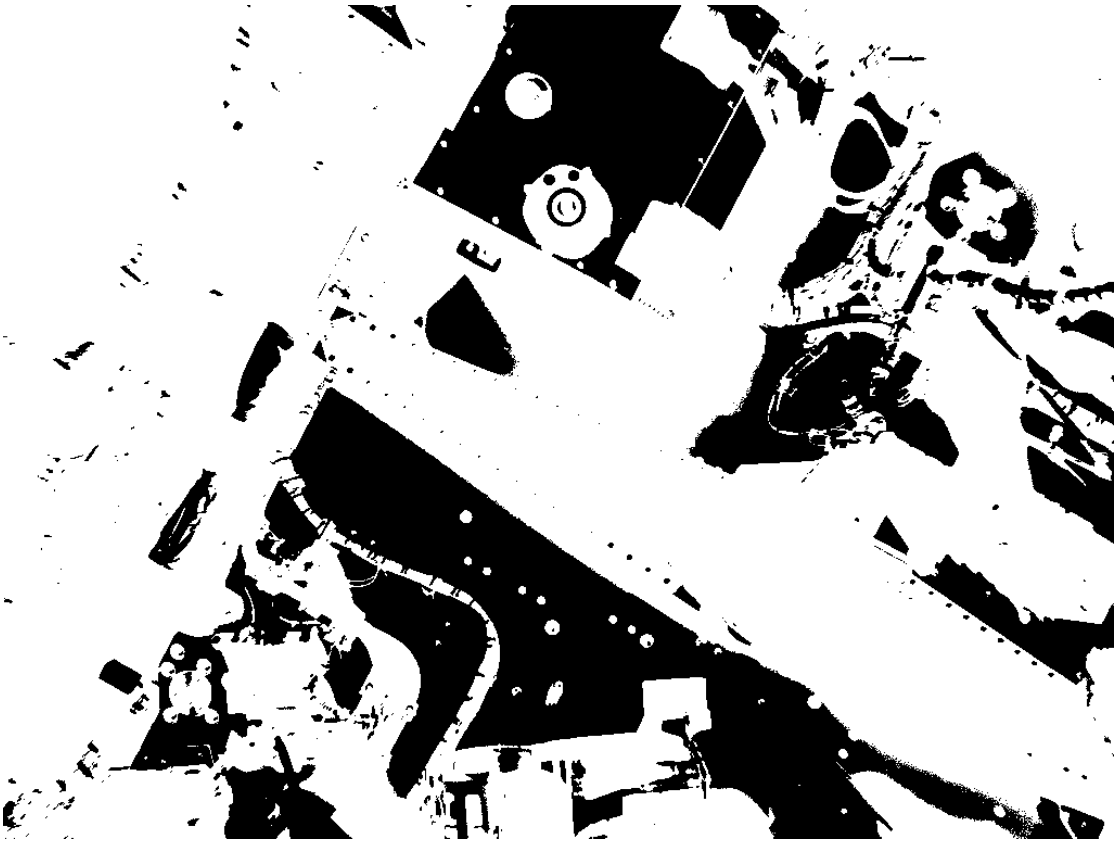
Out[•]= FittedModel[2.04167 – 2.08333 x]

```
In[•]:= deSaturateFactorLM = LinearModelFit[{{.6, .25}, {.98, 0}}, x, x]
```

Out[•]= FittedModel[0.644737 – 0.657895 x]

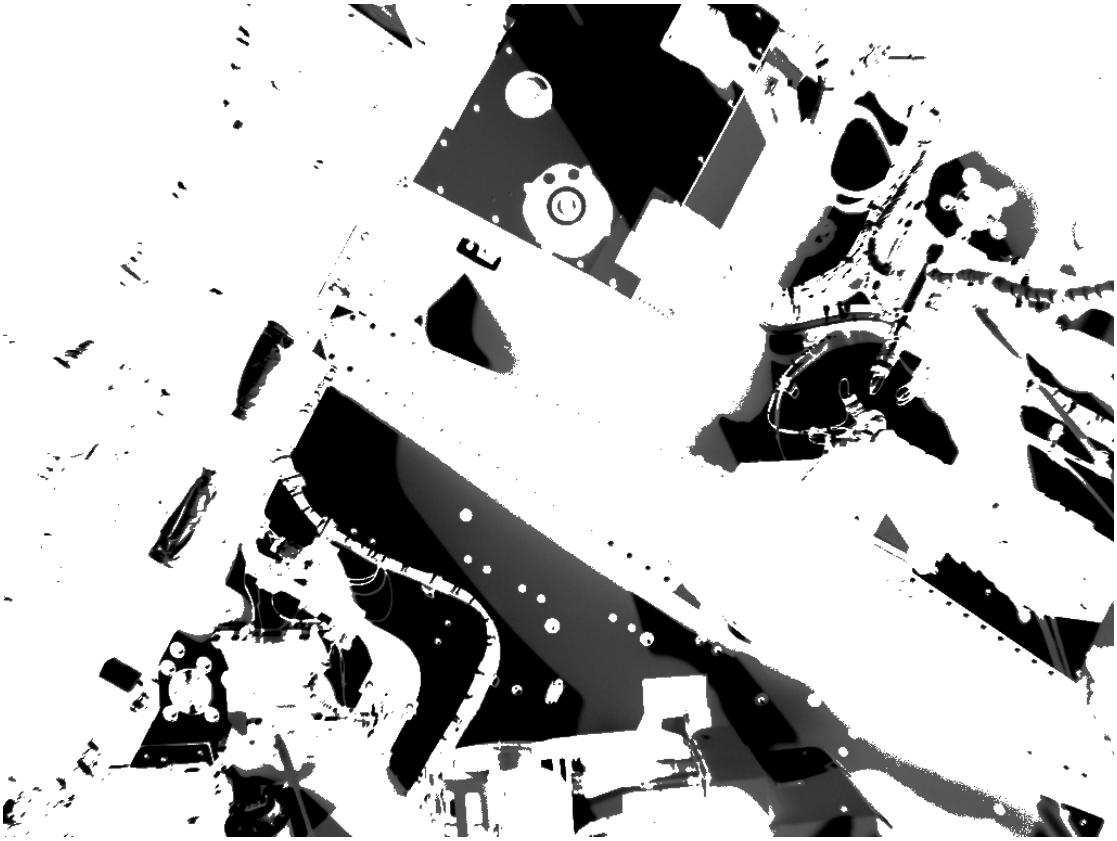
In[•]:=
```
disableDesaturationMask = (1 - Binarize[debayerPlanes[[2]], 100. / 254]) +
  noneSaturatedMask (* Blue < 100 or none saturated
    So only getting applied when blue getting large. Don't want
    capton tape getting desaturated. *)(* + to "or" masks together *)
```

Out[•]=



```
In[•]:= deSaturateFactorImage =
         ImageClip[deSaturateFactorLM[debayerPlanes[[2]]] + 2 disableDesaturationMask]
         (* +2x to "or" disableDesaturationMask on top of blue channel based
          desaturation *)(* result, black areas will be completely desaturated,
         grey areas will be somewhat desaturated,
         white areas will not be left unchanged *)
```

Out[•]=



Transform Raw Image via model

```
In[•]:= rawToXYZIDT = ({{β1, β2, 1 - β1 - β2}
              , {β3, β4, 1 - β3 - β4}
              , {β5, β6, 1 - β5 - β6}
             } /. optResultsM10[[2]])
```

```
Out[•]= {{0.636906, -0.0370224, 0.400116},
         {0.337894, 0.0735785, 0.588528}, {0.123875, -0.351916, 1.22804}}
```

```
In[•]:= optimizedIlluminantXYZ = blackBodyXYZ[b1] /. optResultsM10[[2]]
```

```
Out[•]= {1.08065, 1, 0.398841}
```

```
In[•]:= XYZColor[optimizedIlluminantXYZ]
```

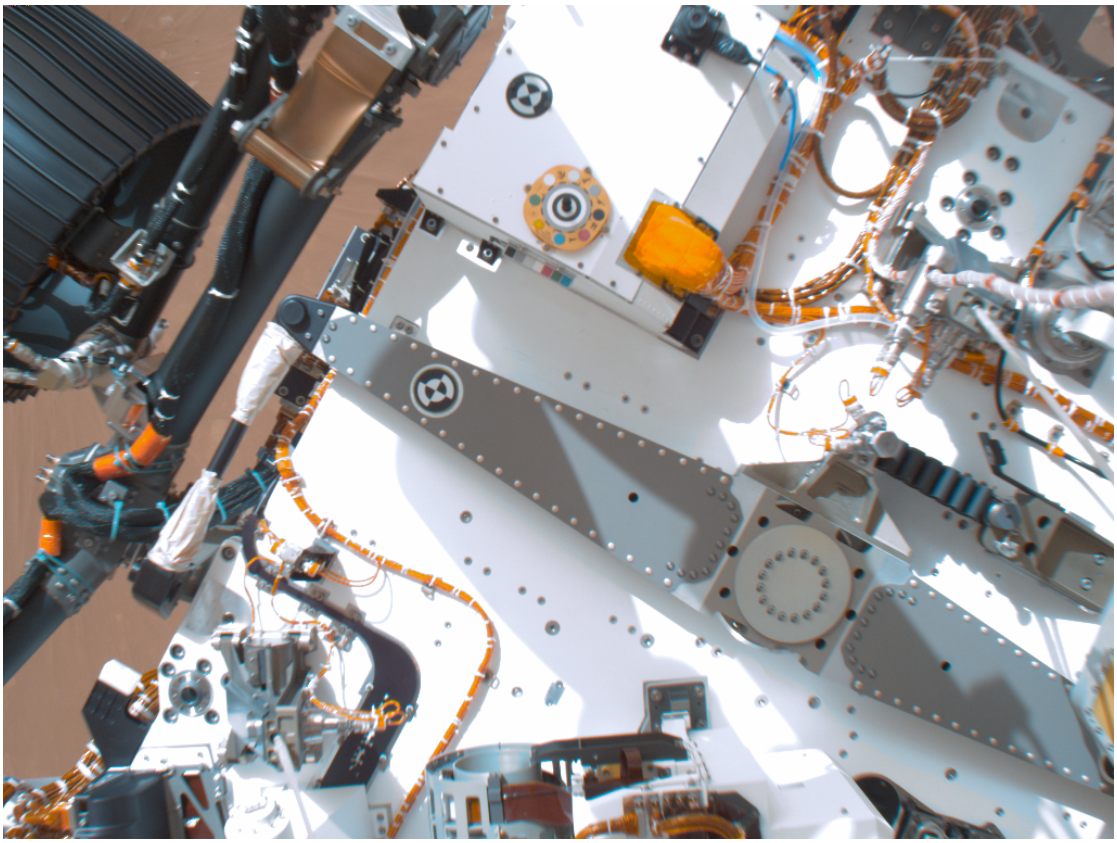
Out[•]=

In[•]:=
```
(* scaled rawRGB → XYZ → chromatic adaption XYZ → okLab *)
intermediatOkLabPlanes = xyzToOkLab[
    xyzAdaptedPlanes =
     (adaptMat[catMatBS(*catMatBradford*), optimizedIlluminantXYZ, d65XYZBL].
       (xyzFromRawPlanes = rawToXYZIDT.( .91 ColorSeparate[debayerPlanesImage])
       )
     )
  ]; (* Model 10,  .91 brings max area white in sRGB to 254 *)
```

In[•]:=
```
(* selective desaturation by scaling of "a" and "b" in okLab color-space,
transform okLab back to XYZ and then XYZ to sRGB-linear *)
sRGBLinearDesatImage = ColorCombine[
    sRGBLinearDesatPlanes = Inverse[linearRGBtoXYZMat[srgbPrimaries, d65XYZBL]].
      okLabToXYZ[{1, (*1+0*)deSaturateFactorImage, (*1+0*)
         deSaturateFactorImage} intermediatOkLabPlanes]
    ,
    "RGB"];
```

In[•]:=
```
gamutReducedImage = linear2srgbImageReal[ImageClip[sRGBLinearDesatImage]]
```

Out[•]=



Bulk Processing

Process multiple RAW ECAM images

In[•]:=
```
(* transform raw image to video frame using model10 and model11 *)
processECAM2[image_Image] := Block[{
    rawImageRGB
    , csRawImage
    , debayerPlane1
    , debayerPlane2
    , debayerPlane3
    , debayerPlane4
    , debayerPlanes
    , debayerPlanesImage
    , saturatedMaskPlanes
    , allSaturatedMask
    , noneSaturatedMask
    , someSaturatedMask
    , anySaturatedMask

    , disableDesaturationMask
    , deSaturateFactorImage

    , rawToXYZIDT
    , optimizedIlluminantXYZ
    , intermediatOkLabPlanes
    , sRGBLinearDesatImage
    , gamutReducedImageM10
    , gamutReducedImageM11

    , finalImage
   }
   ,
   rawImageRGB = image;
   csRawImage = ColorSeparate[rawImageRGB];
   debayerPlane1 = ImageTake[csRawImage[[1]], {1, -1, 2}, {1, -1, 2}]; (* Red *)
   debayerPlane2 = ImageTake[csRawImage[[3]], {2, -1, 2}, {2, -1, 2}]; (* Blue *)
   debayerPlane3 = ImageTake[csRawImage[[2]], {2, -1, 2}, {1, -1, 2}]; (* Green *)
   debayerPlane4 = ImageTake[csRawImage[[2]], {1, -1, 2}, {2, -1, 2}];
   (* Green *)
   debayerPlanes = {debayerPlane1, debayerPlane2, debayerPlane3, debayerPlane4};
   debayerPlanesImage = ColorCombine[
     {debayerPlane1, 0.5 (debayerPlane3 + debayerPlane4), debayerPlane2}, "RGB"];
```

```
saturatedMaskPlanes = (Binarize[#, 253.99 / 255] & /@ debayerPlanes);
allSaturatedMask = ImageMultiply @@ saturatedMaskPlanes;
noneSaturatedMask = ImageMultiply @@ (1 - saturatedMaskPlanes);
someSaturatedMask = (1 - noneSaturatedMask) × (1 - allSaturatedMask);
(* some but not all saturated *)
anySaturatedMask = (1 - noneSaturatedMask);
disableDesaturationMask =
 (1 - Binarize[debayerPlanes[[2]], 100. / 254]) + noneSaturatedMask;
deSaturateFactorImage = ImageClip[
  deSaturateFactorLM[debayerPlanes[[2]]] + 2 disableDesaturationMask];

(* Model 10 *)
rawToXYZIDT = ({{β1, β2, 1 - β1 - β2}
    , {β3, β4, 1 - β3 - β4}
    , {β5, β6, 1 - β5 - β6}
   } /. optResultsM10[[2]]);
optimizedIlluminantXYZ = blackBodyXYZ[b1] /. optResultsM10[[2]];
intermediatOkLabPlanes = xyzToOkLab[
  (adaptMat[catMatBS(*catMatBradford*), optimizedIlluminantXYZ, d65XYZBL].
    (rawToXYZIDT.( .91 ColorSeparate[debayerPlanesImage])
    )
  )
 ]; (* Model 10,  .91 brings max area white in sRGB to 254 *)
sRGBLinearDesatImage =
 ColorCombine[Inverse[linearRGBtoXYZMat[srgbPrimaries, d65XYZBL]].
   okLabToXYZ[{1, (*1+0*)deSaturateFactorImage, (*1+0*)
      deSaturateFactorImage} intermediatOkLabPlanes]
  , "RGB"];
gamutReducedImageM10 = linear2srgbImageReal[ImageClip[sRGBLinearDesatImage]];

(* Model 11 *)
rawToXYZIDT = ({{β1, β2, 1 - β1 - β2}
    , {β3, β4, 1 - β3 - β4}
    , {β5, β6, 1 - β5 - β6}
   } /. optResultsM11[[2]]);
intermediatOkLabPlanes = xyzToOkLab[
  rawToXYZIDT.( .98 ColorSeparate[debayerPlanesImage])
 ]; (* Model 11 , .98 brings max area white in sRGB to 254 *)
sRGBLinearDesatImage =
 ColorCombine[Inverse[linearRGBtoXYZMat[srgbPrimaries, d65XYZBL]].
   okLabToXYZ[{1, (*1+0*)deSaturateFactorImage, (*1+0*)
      deSaturateFactorImage} intermediatOkLabPlanes]
  , "RGB"];
gamutReducedImageM11 = linear2srgbImageReal[ImageClip[sRGBLinearDesatImage]];
```

```
  finalImage = ImageAssemble[{{gamutReducedImageM10, gamutReducedImageM11 }}];
  finalImage
  (*ImageRotate[gamutReducedImageM10,180°]*)
  (* 180° rotation to match JPL video *)
 ]
```

In[•]:=
```
processedDirectory =
 StringReplace[DirectoryName[imageFileNames[[1]]], "/raw/" → "/frames/"]
```

Out[•]= /Users/bswift/Downloads/mars2020/frames/esf/

In[•]:=
```
CreateDirectory[processedDirectory]
```

Out[•]= /Users/bswift/Downloads/mars2020/frames/esf/

In[•]:=
```
DirectoryQ[processedDirectory]
```

Out[•]= True

In[•]:=
```
framePrefix = FileNameTake[processedDirectory] <>
  "_" (* set to give Compressor a prefix for movie filename *)
```

Out[•]= esf_

In[•]:=
```
imageFilenameSelector = 6(*6 esf *) ;;
  -1 (* Skip first 6 raw images. These were processed differently at JPL. *)
```

Out[•]= 6 ;; -1

In[•]:=
```
AbsoluteTiming[
 Parallelize[
  processResults = MapIndexed[
      Export[
        FileNameJoin[{processedDirectory, framePrefix <>
           ToString[50 000 + First[#2]] <> ".png"(*"_"<>FileNameTake[#1]*)}]
        (* 50000 to get fixed lenght frame number and have
         range to add sequences before *)
        , processECAM2[Import[#]]] &
      , imageFileNames[[imageFilenameSelector]]
     ];
 ]
]
```

Out[•]= {62.8013, Null}

In[•]:=
```
processResults // Length
```

Out[•]= 72