Deepspace Interplanetary Navigation Operations Colorado Research EXplorer (DINO C-REx)

DINO C-REx Technical Memorandum

Document ID: DINO_C-REx-Image Generation

SYSTEMS ENGINEERING REPORT 4.8: CAMERA MODULE OBJECT MODEL

Prepared by	Matt Muszynski

Status: Initial Version
Scope/Contents
User documentation for the DINO C-REx Camera Module.

Rev:	**Change Description**	**By**
1.0	Initial Release	Matt Muszynski

Contents

1 Overview

This document covers the overall design of the camera module, focusing mostly on the object model for the DINO C-REx camera module. This document describes the model qualitatively and would be best used while simultaneously following along through the code in camera.py.

2 Object Model

The camera module's object model contains three classes, the camera, image, and scene. Each class can be seen as holding data specific to a smaller period of time than the last. The camera object is used to carry all data that persists at any time during the simulation, such as camera attributes, camera to spacecraft body orientation, and information about individual stars. The image object contains the same information plus information about beacons, but only for the object that are visible at some time during the exposure that it represents. Finally, the scene object contains information pertaining only to the objects visible at a single integration time step. The camera and image classes have associated functions that are described qualitatively here. Specific i/o information and more detailed descriptions of functionality can be found in code comments and the DINO C-REx Doxygen code documentation.

2.1 Camera

The camera class is remarkable in that it defines the only objects directly created by the user. All other objects in the camera module are created by functions that cascade through the object model. Upon initialization, camera.__init__() runs to calculate camera characteristics based on user inputs and load stellar information from the SQLite database tycho.db. camera.updateState() is run at each simulation timestep, its behavior controlled by the takeImage message delivered by the navigation executive.

2.1.1 camera.__init__()

The init function for the camera class does five things:

- Uses camera.calculateFOV() to calculate the angular size of the camera field of view from user inputs (focal length and physical size of the detector)
- Loads stellar data from tycho.db using camera.loadAllStars(), including right ascension, declination, estimated temperature, and solid angle subtended by source. camera.loadAllStars() also calculates unit vectors from the spacecraft to each star[1]
- Interpolates quantum efficiency and transmission curve data provided by the user so that both are sampled at the same wavelengths.[2]
- Calculates a solar blackbody curve at the same wavelength values as the quantum efficiency and transmission curves.
- Sets camera attributes based on user input and output of member functions.[3]

2.1.2 camera.updateState()

The camera's update state function is the main controller of the camera module. It is controlled itself by the takeImage message written by the navigation executive, and will take different actions depending on whether takeImage is True or False.
Regardless of the value of takeImage, camera.updateState() begins by counting the number of open images[4] present in the camera. There should never be more than one open image, so if more than one is

[1] See SER 4.1 for details on calculation of estimated temperature and solid angle subtended by source.
[2] See SER 4.13 for details.
[3] For full itemization of camera class attributes, see Doxygen code documentation
[4] images with the imageOpen attribute set to 1

found, the function throws an error and exits. If either 1 or 0 images are open, the function proceeds. If there are no open images, camera.updateState() will instantiate a new image object, place it in the list contained in camera.images, and run the image's updateState() method[1]. If there is one open image, camera.updateState() will run image.updateState() on that open image.

2.2 Image

The image class also has two main functions, an init function and an update state function. But here, the functionality lies completely in image.updateState().

2.2.1 image.__init__()

The image.__init__() function is very simple. It simply initializes attributes in the image object.

2.2.2 image.updateState()

The image class's update state function does most of the heavy lifting in image creation. Like camera.updateState(), it also behaves differently based on the value of the takeImage message:

- If takeImage == 1, image.updateState() simply writes the inertial to camera direction cosine matrix to the list contained in the list image.DCM. It will be used later.
- If takeImage == 1, image.updateState() does the following:
  - Converts all matrices in image.DCM to 3-2-1 Euler angle rotations from the initial attitude of the image.
  - Finds the maximum and minimum right ascension and declination (relative to the camera, not inertial RA and DE) of any frame in the image.
  - Uses those maximum and minimum values to calculatee a new field of view that contains all stars possibly visible during the exposure
  - Uses that field of view with the function image.findStarsInFOV() to find the subset of stars that and beacon facets[2] that are visible at any time during the exposure (this greatly increases the speed of image.findStarsInFOV() when it is used to find the stars and beacons in each scene).
  - Saves data returned by image.findStarsInFOV() for the large FOV into attribtues to be used later.
  - Calculates the number of photons per second incident on the detector using em.planck(), temerature and solid angle data loaded into the camera object from tycho.db, and the sensitivity curve calculated by camera.__init()__.
  - Creates one scene object per attitude in the list image.DCM. For each, the stars and beacon facets found for the large full-image FOV are reduced to only those visible in the current time step. Point spread and rasterization are applied, shot noise and dark current noise added, and hot and dark pixels applied.
  - Finally, all scenes in the image are summed, read noise is applied, and the final array of pixel values is sent to image processing.

2.3 Scene

The scene object is more of a debugging and organizational artifact than a functional piece of the model. It is only used to hold information about stars and beacon facets visible in each integration time step.

[1] See below for details

[2] beacon facets themselves are added by image.findStarsInFOV(), and those stars and beacon facets that are occulted by beacons are also removed by image.findStarsInFOV()

2.3.1 scene.__init__()

The scene class's init function initializes attriutes and sets them as directed by image.updateState()