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

DINO C-REx Technical Memorandum

Document ID: DINO_C-REx-Image Generation

CAMERA OBJECT UNIT TESTS

Prepared by	Matt Muszynski

Status: Initial Version
Scope/Contents
Description of unit tests completed for the DINO C-REx camera object. This includes tests 4.1 through 4.7.

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

Contents

1 Overview

This document describes all unit tests written for the camera object by the DINO C-REx camera model team. Each method of the camera object has an associated test, and additional integrated tests are performed where deemed useful by the team. Each test has an entry in this document for status (including date of status change), responsible DINO C-REx team member, a qualitative description of the test performed and a brief technical discussion of the method employed in code to achieve the test.

2 Tests

2.1 Test 4.1: camera.load_all_stars()

Status: Complete. 11.07.17
Responsible Team Member: Matt Muszynski
Description: The method camera.load_all_stars() is used to load stellar data from db/tycho.db into the camera object. This test ensures all data loaded from the db is loaded correctly and that RA/DE coordinates are correctly converted to inertial unit vectors. .
Method: Stellar data is loaded into a camera at runtime and arrays for temperature (T), inertial unit vector coordinates (n_1, n_2, and n_3), right ascension (RA), declination (DE), Tycho visual magnitude (VT), and Tycho color index (BVT) are summed and compared to values computed offline and saved in the file camera_test_support_files/4.1.test_support.npy. The values in the save file are assumed to be correct because they represent a snapshot of the system at the time that all analysis was done and SERs were writted (the Spring 2017 semester).

2.2 Test 4.2: camera.calculate_FOV()

Status: Incomplete. 10.31.17
Responsible Team Member: Beryl Kuo
Description: The method camera.calculate_FOV() calculates the angular field of view of a camera based on focal length and detector dimensions as entered by the user. Three fields of view are computed by hand and compared to the same ones computed at run time.
Method:

2.3 Test 4.3: camera.find_lambda_set()

Status: Incomplete. 10.31.17
Responsible Team Member: Beryl Kuo
Description: camera.find_lambda_set() finds a unified set of wavelength (λ) values at which the planck function will be evaluated and the transmission curve and quantum efficiency curve will be interpolated. Three lambda sets are found manually using three pairs of TC and QE curve. They are then compared to the same sets computed at run time.
Method:

2.4 Test 4.4: camera.interpolate_lambda_dependent()

Status: Incomplete. 10.31.17
Responsible Team Member: Beryl Kuo
Description: camera.interpolate_lambda_dependent() uses the lambda set found by camera.find_lambda_set() to interpolate the TC and QE curves such that they are evaluated at the same wavelength values. This test compares interpolated TC and QE curves calculated at run time to ones found manually using the

TC and QE curves used in test 4.3 and the lambda sets output by test 4.3.
Method:

2.5 Test 4.5: Integrated QE/TC Test

Status: Incomplete. 10.31.17
Responsible Team Member: Beryl Kuo
Description: This test continues the thread of tests 4.3 and 4.4. For each manually calculated data set in those tests, a sensitivity curve is calculated (multiplying the interpolated TC and QE curves). That sensitivity curve is compared to the ones caclulated at run time for each set.
Method:

2.6 Test 4.6: camera.hot_dark()

Status: Incomplete. 10.31.17
Responsible Team Member: Ishaan Patel
Description: Because camera.hot_dark() is inherently random, this test is acheived by running the function (TBD) times and checking that the final distribution of hot/dark pixels is truly random in the way that is expected.
Method:

2.7 Test 4.7: camera.update_state()

Status: Complete. 11.07.17
Responsible Team Member: Matt Muszynski
Description: The camera.update_state() method is responsible for opening, closing, and updating the image object. This test manually sets the take_image message and checks that image objects are handled successfully based on its contents.
Method: All assertions in this test involve counting the number of images present in the camera or the number of scenes present in an image. The following process is used:

1. Assert that there are no images present in the camera object (noStarCam).
2. Run noStarCam.updateState() three times. This adds three attitude states to the image object noStarCam.images[0], but because the image is not yet closed, no scenes are present.
3. Assert that there is one image object in the camera and there are no scenes in that image object.
4. Set takeImage to 0 and run noStarCam.updateState(). This closes noStarCam.images[0] and creates its scenes (one for each attitude state)
5. Assert that there is one image in noStarCam, and that there are three scenes in noStarCam.images[0].
6. Set takeImage to 1 and run noStarCam.updateState(). This creates noStarCam.images[1].
7. Assert that noStarCam now contains two images, that noStarCam.images[0] still only has three scenes, and that noStarCam.images[1] doesn't have any scenes yet (since it hasn't yet been closed)
8. Set takeImage to 0 and run noStarCam.updateState() one last time. This closes noStarCam.images[1] and creates its one scene.
9. Assert that noStarCam has two images, that noStarCam.images[0] still has only three scenes, adn that noStarCam.images[1] has only one scene.