# ENB241 Group\_030 Documentation: C++ Object Tracker Using VXL

## Group 30 Delivery Feature Report

The software system designed, implemented and tested for this project tracks an object as it moves in a sequence of images. The object tracking system implements the following required features:

* Take as input of a list of images, specified as a path and a partial filename (i.e. \*.png to get all png files), and a set of parameters
* Create a distribution field model of an object of interest based on the initial location specified by the input parameters
* Track the object through the rest of the video (sequence of images), updating the model as the video progresses
* Compute performance metrics using a given ground truth image and index, evaluating the performance of the tracker

The system is designed using object oriented programming principles, and the four features listed above are encapsulated by the four main classes UserInput, DistributionField, DFT, and TestCases.

Optional features implemented in this system include:

* Save a set of images showing the tracking output – a square is drawn around the object in each frame
* Enhanced Distribution Field Tracker – better tracking algorithm, replacing distribution field with channel representation and motion estimation
* Track Objects in Colour – use the colour information in the images instead of reducing to greyscale
* Enable the use of a configuration file – for ease of use, a text file saves parameters which the program reads instead of having to input via command line each time.

Usage:

The software can be run from the command line. Parameters can be specified at run time with command line arguments or with a configuration file. The following command structure is used:

ObjectTracker.exe -path “input image path” -glob “image type” -ipx “column pixel of object” -ipy “row pixel of object” -w “width of object in pixels” -h “height of object in pixels” -c “number of channels in the object model” -sb “special blur” -bc “colour blur” -sd “max search distance” -planes “number of colour planes” -lr “model learning rate” -sds “spacial blur standard deviation” -sdc “colour blur standard deviation” –odir “output image directory”

Further help and usage information can be obtained with ObjectTracker.exe -?

An example of proper usage:

ObjectTracker.exe -path ../../Data/bolt -glob .jpg -ipx 336 -ipy 165 -w 25 -h 60 -c 8 -sb 4 -bc 1 -sd 30 -planes 3 -lr 0.05 -sds 1.0 -sdc 0.625 -odir /Output/outputBolt

To run with a correctly formatted configuration file, simply run the executable with no arguments, i.e.:

ObjectTracker.exe

Known issues or shortcomings with the system:

* Only works on a series of images, not actual video
* Not all incorrect inputs are caught, some assumption that the user knows what they are doing
* Test cases do not guarantee infallibility; this is a complex problem that is difficult to exhaustively test.

Extensibility:

Class based design means changes and additions can be made within classes without the rest of the code, for example the extension functionality was all able to be added easily by making the modifications to the relevant class or inheriting from one of the existing classes. However, any changes that require extra input or parameters will need to change several classes to work.

## Initial Work Breakdown Schedule

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Initial Work Breakdown Schedule** | | | | | |
| Wade Jensen | Jordan Laurie | Lachlan Robinson | Saleh Mulla | Omar Alsubaia |
| Initial Project Plan | Initial System Design | Initial System Design | UserInput Class | UserInput Class |
| Final Project Plan | Initial Class Diagram | Initial Class Diagram | Test UserInput Class | Test UserInput Class |
| Initial Class diagrams | Final Systems Design | objectModel Class | EDFT Class | Test Distribution Field Tracker Class |
| Sequence diagrams | Final Class Design | Inline documentation | Test EDFT class | Final Class diagrams |
| Initial System Design | Distribution Field Class | Final Integration | Test Distribution Field Class | Test Colour Tracking |
| Distribution Field Tracker Class | Colour Tracking | Test objectModel Class |  |  |

## Final Work Breakdown Schedule

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Final Work Breakdown Schedule** | | | | | |
| Wade Jensen | Jordan Laurie | Lachlan Robinson | Saleh Mulla | Omar Alsubaia | Non-complete at time of writing |
| Project Plans | Distribution Field Class | Inline documentation | UserInput Class | UserInput Class | Test Distribution Field Class |
| Sequence diagrams | Colour Tracking | Colour Tracking |  |  | Test Colour Tracking |
| Class diagrams | Final Class Design | Final Integration |  |  | Test Distribution Field Tracker Class |
| Initial System Design | Final Systems Design | Stage 1 Integration |  |  | Test EDFT class |
| Final Systems Design | Stage 1 Integration | Debugging and Refactoring |  |  |  |
| Distribution Field Tracker Class | Distribution Field Tracker Class | Delivery report |  |  |  |
| Stage 1 Integration | Final Integration |  |  |  |  |
| Debugging and Refactoring | Debugging and Refactoring |  |  |  |  |
| UserInput Class | EDFT Class |  |  |  |  |
| UserInput Class testing | Channel Representation Class |  |  |  |  |
| Final Integration | Efficiency and optimisation |  |  |  |  |

## Work Breakdown Schedule Notable Differences

Some team members contributed more than others and as a result their contributions were more or less than expected. Names will not be called out here, however the breakdown schedule speaks for itself and a git repository showing the commit history of the project is available on request.

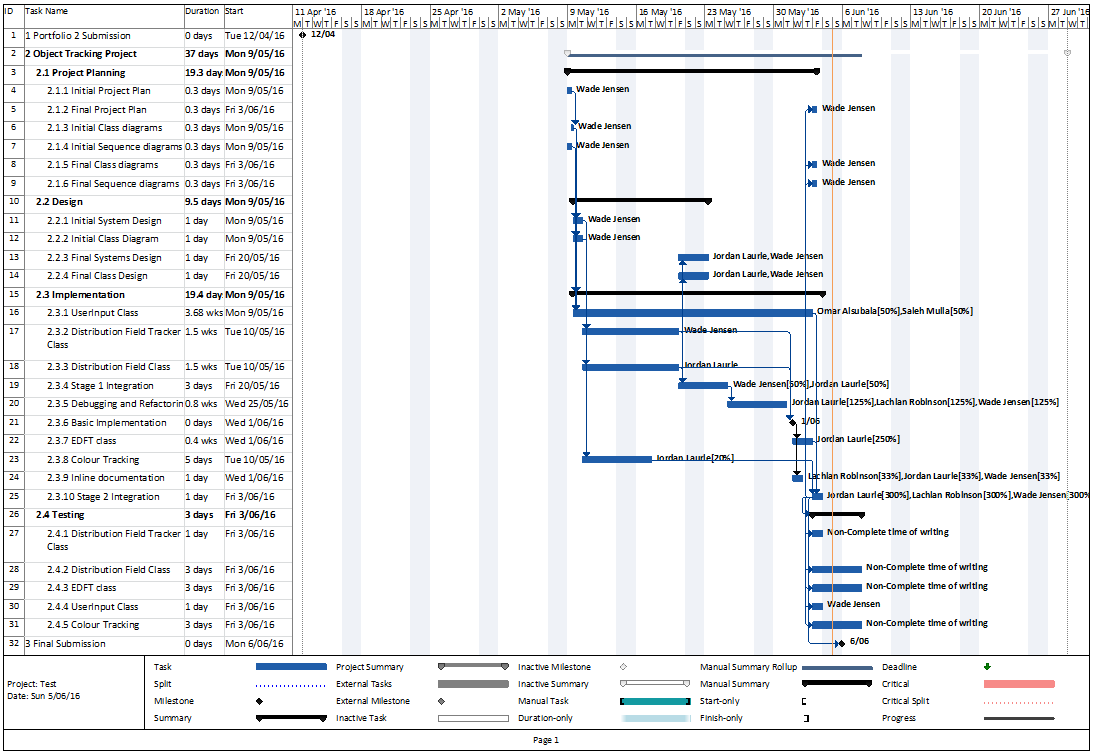
Significant differences arose between tasks originally allocated and the final breakdown of work. New tasks arose, the nature of the design changed and significant development time was spent refactoring and to optimize the code due to slow performance. Wade Jensen primarily worked on documentation, the Distribution Field Tracker class, integration of various classes and debugging. Jordan Laurie primarily worked on a significant portion of the code and technical aspects, optimizing, debugging and most of the extensions. Lachlan Robinson primary worked on the colour tracking extension, inline documentation, the delivery report and integrating classes together into the program. Saleh Mulla and Omar Alsubaia worked jointly on the UserInput class.

## Initial Project Plan

## C:\Users\Wade Jensen\Dropbox\01_EN40\YEAR_3_SEM_1\ENB241\CodeBlocks_and_VXL\CodeBlocks-Projects\ObjectTracker\doc\gantt char intitial 4-1.png

**Project: Object Tracker  
Date: Mon 9/05/16**

## Final Project Plan



## Project Plan Notable differences

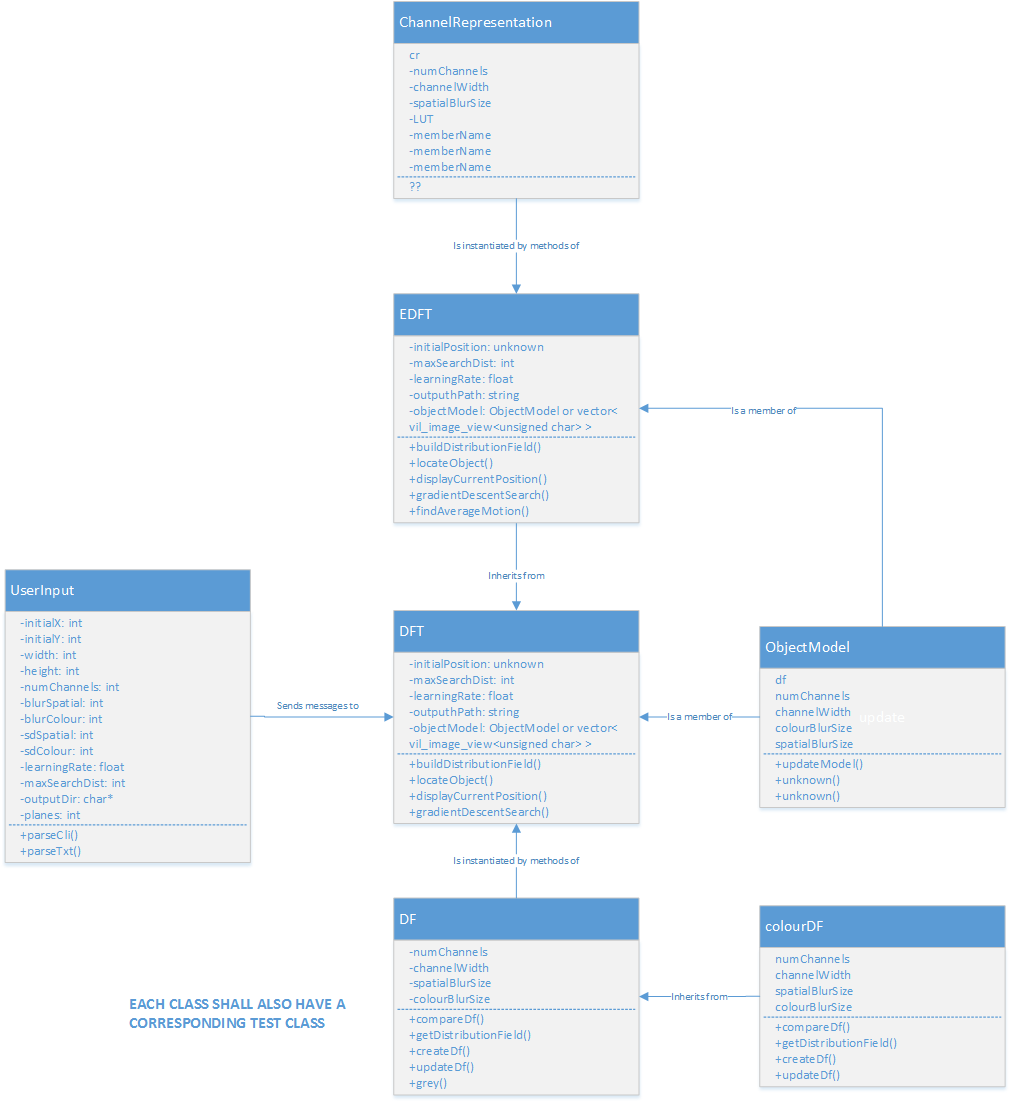
The group was unwilling to come to consensus on a meeting time to work together on the software design for the first several weeks of the project. Because of this the final project timeline does not begin until May 9th. This lead to a significant compression of the timeframe for the project.

The objectModel class assigned to Lachlan Robinson became an obvious component of the distribution field tracker class. In the initial software design it was not fully understood that the object model was merely a distribution field.

While it was originally planned that test cases should be written simultaneously with application code, the team’s unfamiliarity with the algorithm itself made it impossible to test until the code itself code be understood. Had the algorithm been of the team’s own design, this would have been possible, but without a solid understanding of expected outcomes, it was difficult to produce test cases. This lead to testing being left to the final weekend before the due date and in all likelihood reduced the quality of some tests and overall coverage.

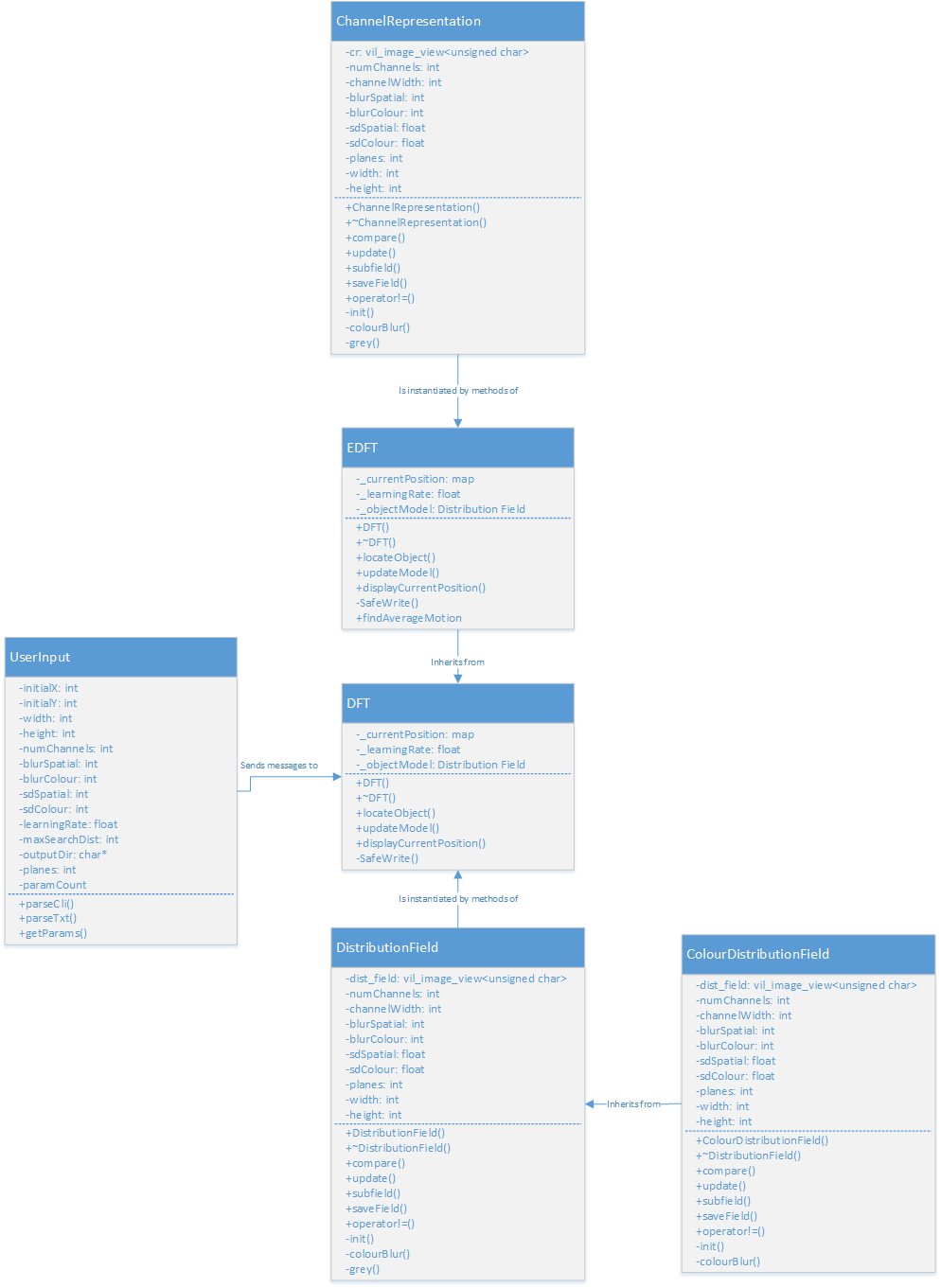
As with any group assignment, some members were not able to contribute at the same level as others, which also lead to slipped deadlines.

## Class Diagram Initial



Note that in comparison to the final class diagram there are several question marks present in some classes. These quite honestly reflect the lack of initial understanding of the algorithm/problem which is inherent to most large software projects. The missing sections should not be regarded as a lack of planning but rather a reflection that designing a system of which you do not possess an understanding is not possible, at least in a manner which would be meaningfully close to the final design.

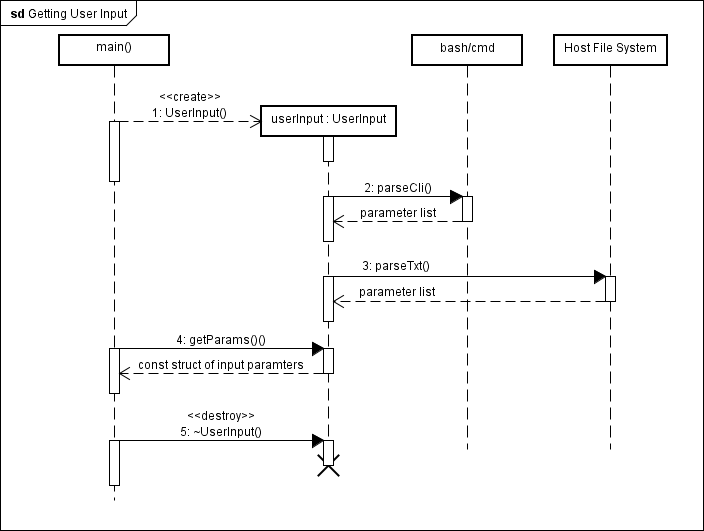
## Class Diagram Final



## Use Case Diagrams

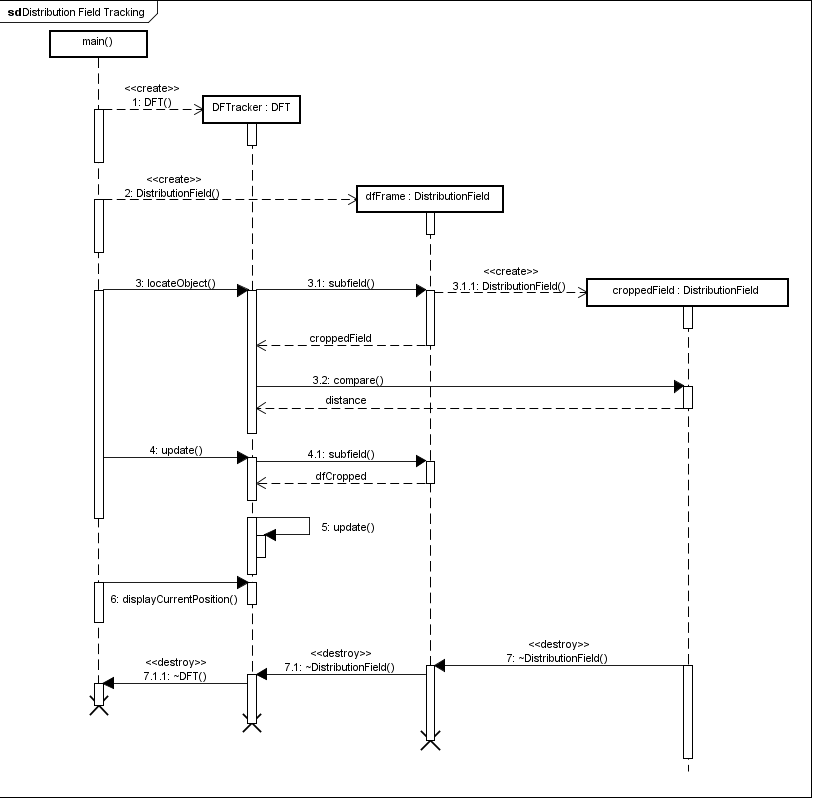
### Getting User Input

The process of getting input from the user, either from command line or a configuration file and returning a read-only structure of parameters for use by the tracking algorithm.



### Distribution Field Tracking

The following sequence diagram describes the main process of the tracker algorithm at a high level. From the instantiation of the DFTracker object to the gradient descent algorithm within DFT::locateObject, to the saving of the final image showing the object position withindisplayCurrentPosition.



## Test Plan

The testing shall be broken up into unit tests. Such that each class has tests for each of its methods, and all code paths shall be tested within each methods. Given the time constraints which the group fell under, it was decided that only the critical public methods need be tested, particularly given that the code was already found to be functioning correctly with the ‘bicycle’, ‘bolt’, ‘cup’, ‘juice’ and ‘jump’ data sets.

**UserInput Class**

The UserInput class has three primary code pathways which are further broken down into success or failure states:

* parseCli – parses the command line for input parameters
  + Valid inputs found -> Success - > return true
  + Invalid command line arguments found -> Failure -> return false.
* parseTxt – if the command line arguments are not valid, then search for the default configuration text file
  + Text file is present and inputs valid -> Success.
  + Text file is not present or inputs are invalid -> Failure -> End program.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **User Input Class Test Scenarios** | | |
| **Test Number** | **Method** | **Scenario** | **Expected Result** |
| 1 | parseCli | Valid command line inputs | Matching member variable values in UserInput object. |
| 2 | parseCli | Invalid command line inputs | return false |
| 3 | parseTxt | Valid text file | Matching member variable values in UserInput object. |
| 4 | parseTxt | Missing text file | return false |
| 5 | parseTxt | Invalid text file values | return false |
| 6 | parseTxt | Valid text file values | Matching member variable values in UserInput object. |

DFT Class (Distribution Field Tracker)

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Distribution Field Class Test Scenarios** | | |
| **Test Number** | **Method** | **Scenario** | **Expected Result** |
| 7 | DistributionField(const vil\_image\_view<unsigned char>&, DF\_params&) | Inputs are a 3 plane colour image and a field parameter object. Number of planes in the parameter object is one. Number of channels is 8. | * A distribution field, ie. a vector of vil\_image\_views. * Assert that the length of the vector equals the number of channels * Assert that the width and height of each field should be the same as the original image * Assert that the number of planes in each field is one (greyscale) |
| 8 | DistributionField(const vil\_image\_view<unsigned char>&, DF\_params&) | Inputs are a 3 plane colour image and a field parameter object. Number of planes in parameter object is three. Number of channels is 8. | * A distribution field, ie. a vector of vil\_image\_views. * Assert that the length of the vector equals the number of channels * Assert that the width and height of each field should be the same as the original image * Assert that the number of planes in each field is three |
| 9 | DistributionField subfield(int, int, int ,int) const; | A distribution field object is initialised and the cropping subfield method is called to reduce it to a desired size. (greyscale) | * Assert that the size of the new distribution field is correct * Assert each field has one plane * Assert that the pixels it contains correspond to those of the original distribution field |
| 10 | DistributionField subfield(int, int, int ,int) const; | A 3 plane distribution field object is initialised with a colour image and the cropping subfield method is called to reduce it to a desired size. | * Assert that the size of the new distribution field is correct * Assert each field has three planes * Assert that the pixels it contains correspond to those of the original distribution field |
|  |  |  |  |
|  |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Distribution Field Tracker Class Test Scenarios** | | |
| **Test Number** | **Method** | **Scenario** | **Expected Result** |
| 11 | DFT(const DistributionField& initialFrameDF , int x, int y, int width, int height, float learningRate, int maxSearchDist); | Inputs are a 3 plane colour image (first frame), the values for the initial position of the object, the learning rate and maxSearch distance. | * A distribution field tracker object * Assert that the currentPosition internal map is set correctly * Assert that the dimensions of the objectModel distribution field are the correct width and height * Assert that the member variable value of learning rate matches the input * Assert that the member variable value of maxSearchDist matches the input |
| 12 | map<vcl\_string,int> DFT::locateObject(void) | The locate object method is called with no input. | * Assert that the parameters inside the map correspond to the given inputs |
| 13 | map<vcl\_string,int> locateObject(const DistributionField& df); | Applying gradient descent search method from a single frame to another subsequent frame. | * Assert that the difference between the groundtruth and resulting currentPosition is less than 4 pixels absolute distance. |
| 14 | void displayCurrentPosition ( vil\_image\_view<unsigned char> image, vcl\_string outputPath, int frameNum ); | The displayCurrentPosition method is given an image and the currentPosition is stored in the DFT object. A rectangle is draw in the current frame and saved. | * Load the saved image and assert that the correct pixels have been set to black to form a border. |
|  |  |  |  |
|  |  |  |  |

|  |  |
| --- | --- |
| Section | List of Contributors |
| **Documentation** |  |
| Project Plan | Wade Jensen |
| Software Design | Wade Jensen |
| Sequence Diagrams | Wade Jensen |
| Test Plan | Wade Jensen, Lachlan Robinson |
| Delivery Feature Report | Lachlan Robinson |
| **Software** |  |
| DF class | Jordan Laurie, Lachlan Robinson |
| DFT class | Wade Jensen, Jordan Laurie |
| EDFT class | Jordan Laurie |
| Channel Representation | Jordan Laurie |
| Debugging and optimisation | Jordan Laurie, Wade Jensen, Lachlan Robinson |
| Input classes | Saleh Mulla, Wade Jensen, Omar Alsubaia, Lachlan Robinson |
| Testing | Wade Jensen, Lachlan Robinson |
| Signatures of all group members: | |