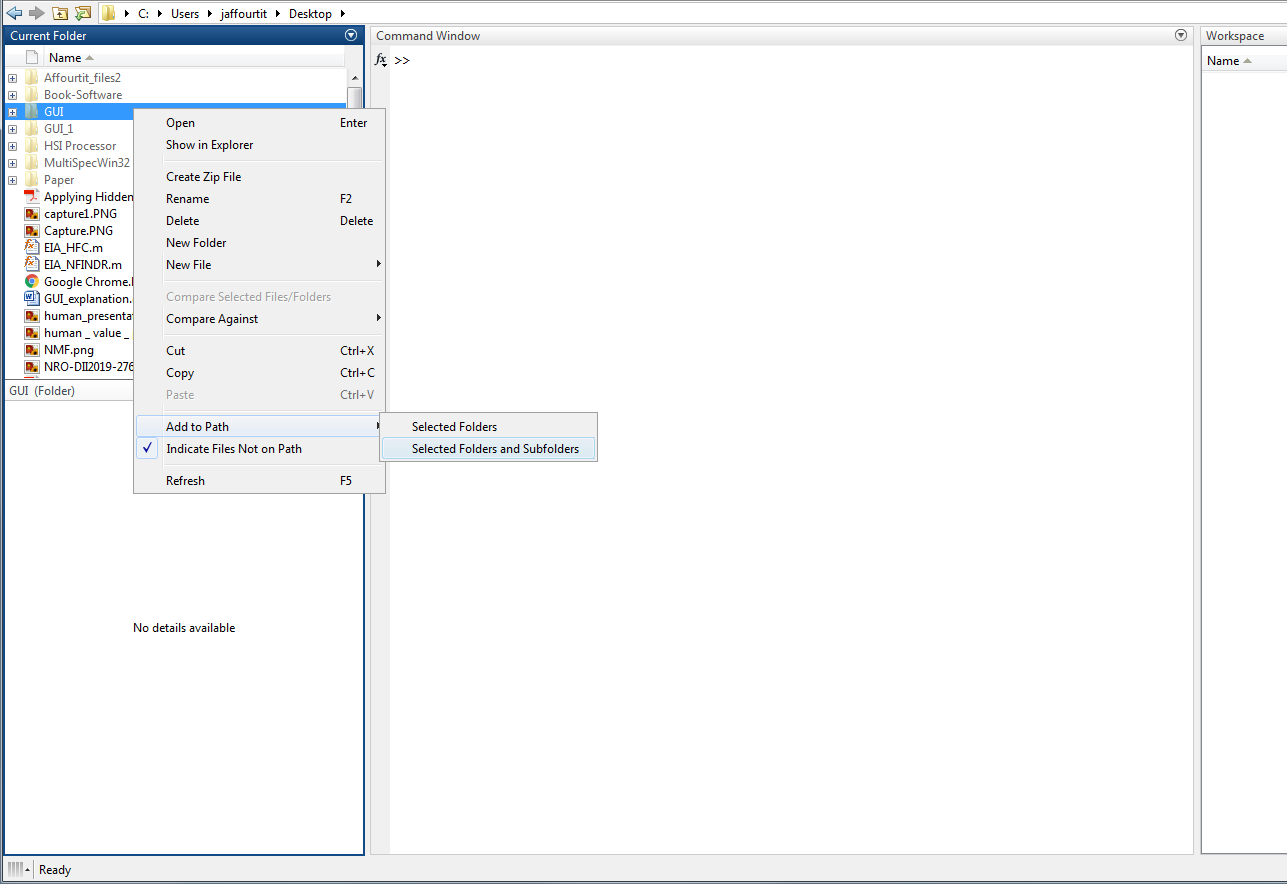
**Guide to operate the GUI and functions used in conjunction with: Optimal Adversarial Location & Pathway Estimation Algorithm**

**Created and edited (August 2018)**: Josef Affourtit and Dr. Nicholas Scott

**Please note:** Everything needed to run the GUI and create the illustrations used in the internship presentation and technical paper are contained in the directory “GUI”. This is a suite of GUI’s and Algorithms which exchange data between figures and functions. The user only needs to run the first GUI by typing “part1”, in order for the HSI image to be processed and Q-learning applied.

**Running the GUI:**

1. The folder must be added to path; otherwise the calls between functions throw errors. *“Selected Folders and Subfolders”*.



**Figure 1: Add folder to path**

1. There are two files associated with each GUI, the .fig file which contains MATLAB’s GUIDE editor (edit the visuals, add plots or buttons), and the .m file which controls the .fig file and handles user interaction.
2. To run a GUI, find the .m file associated with the .fig file you want to run, and simply call the .m file the way you would run a function. E.g. “run(‘part1.m’)” or just type “part1”.

**Part1:**

Illustrates image chip formation and PCA performed on the chips, and learns target spectral signatures.

1. Click “Start”
2. “Select Pixel” gets a point from the large HSI image, and plots that image chip along with its PCA.
3. This uses a MATLAB function getpts(), click a point and hit enter or right click a point, or double click. Any of these methods will register that point and update the plots.
4. Though not applicable in this GUI, you can click multiple points then hit enter to register multiple points. This is useful in the Human Value GUI.
5. “Get Pixel” displays the PCA signal from a pixel in the image above the button.
6. “Display Spec\_sig” displays the spectral signature (pre-PCA) from a pixel in the image chip above the button.
7. Once finished, click “Next” to move to part2. Or exit the window and in the console run “part2”.

**Part2:**

Illustrates the abundances found from NCLS Matched Filter.

1. “Pause”, “Step”, and “Resume” are buttons that perform their respective actions on the iteration of image chip formation through the large HSI image.
2. The toggle buttons titled “Target Dictionary” cannot be used until the HSI image has been processed. “Display” shows the spectral signatures for the toggled button. I.e., ATGP toggled, press “Display” and the 13 spectral signatures in the ATGP dictionary are displayed.
3. Once finished, click “Next” to move to part3. Or exit the window and in the console run “part3”

**Part3:**

Illustrates the LEK algorithm applied to the image chips. The GUI is baseline functional; there is still much work to be done.

1. Once finished, click “Next” to move to part4. Or exit the window and in the console run “part4”

**Part4:**

Human Value is assigned to image chips in this GUI.

1. “Iterate Image” cuts through the HSI image in the same fashion as the algorithm and rewards can be assigned to each image chip, Low, Med, or High.
   1. There are some bugs not yet fixed.
      1. For example, if a chip is assigned two values (accidentally or otherwise) then the plot will superimpose both values on the HSI image. Though, the value field will store the most recent value for that image chip.
      2. Iterating through the HSI image the algorithm will try to skip over images already assigned value. It works on the first column, and then is buggy for later columns.
2. The value box which is initially set to “5” can be edited to any value. The user can then select a location on the HSI image which corresponds to that value, and the image chip containing that location is given the value from the box.
   1. This uses MATLAB’s getpts() function. Click multiple points or just one point, then hit ‘enter’ to finish, or right click your final location to finish. Also, double clicking a point you already selected will register your points and finish.
3. If the “Gaussian Filter” check box is toggled, then a filter can be applied to the reward field. The slider sets the standard deviation value used in the MATLAB function imgaussfilt(). 1.5 is recommended, because larger values will excessively smooth the reward field.
4. “Illustrate” will illustrate the meshCanopy() of the HSI image in grayscale, along with a smoothed or not smoothed value field above the image.
5. “Perform Machine Learning” checkbox changes the output from “RESULTS”. If the checkbox is not toggled then the “RESULTS” button will only save the value field to the MATLAB root for the user to access outside of the GUI. If it is toggled then the following information is needed:
   1. “Reinforcement Learning”: Q-Learning or SARSA. The toggled button performs the respective reinforcement learning algorithm on the value field.
   2. “Learning Parameter”: a value between 0 and 1, which is the gamma value in the Q-learning or SARSA function.
   3. “Optimal Pathway”, this determines which Instant Reward Matrix is created as well as which algorithm is used to find its respective optimal path.
6. “RESULTS” will begin the Reinforcement Learning, or just save the value field to root. If there is Reinforcement Learning, then a wait bar will appear to illustrate which iteration the learning algorithm is currently on. **Please note:** Using Dijkstra’s algorithm means the instant reward matrix has no restrictions and the reinforcement learning takes significantly longer to process. It may take 10-15 minutes at a maximum.
7. A figure will appear with the optimal pathway and the value field above the grayscale HSI image.

**Functions:**

The folder titled “Functions” in the “GUI” directory contains a multitude of functions used for the GUI as well as creating the illustrations for the Technical Paper and Presentation. Though some are not directly related to the GUI, this is still a useful location for those m files and saved data.

In alphabetic order:

**Dijkstra:**

This folder contains the functions used in Dijkstra’s algorithm. The accessories functions are not needed to be understood to use the algorithm. In this algorithm, the \*Q matrix following reinforcement learning is the correct adjacency matrix to use, and the initial and end nodes are arbitrary but usually the starting node is 1 and the end node is the number of image chips (i.e., rows\*cols of HSI image = number of image chips).

\*Dijkstra’s algorithm finds the least distance shortest path. The Q matrix holds large values for very rewarding actions and small values for not so rewarding of an action. This means Dijkstra would find the least rewarding path (since the smaller values are “less distance”). Using invert\_mat() on the Q matrix solves this problem and Dijkstra’s algorithm works as intended.

The input parameters are:

1. An adjacency matrix (the inverted Q matrix)
2. An integer initial state
3. An integer end state

The outputs are:

1. The cost of the actions, or the sum of each value corresponding with the action
2. The pathway

**Create\_reward\_matrix:**

This function creates an adjacency matrix for a graph. For the purpose of this algorithm, this function creates an instant reward matrix from a value field, which is then used for Q-Learning or SARSA to find optimal states and actions for an agent. The function parameters are:

1. A value field (a matrix of m by n dimensions)
2. An integer end state. If the end state is not specified, the function defaults to the node m\*n, the final image chip.

This function allows the agent to have complete freedom of movement within the bounds of the HSI image.

**Create\_reward\_matrix\_1:**

This function is the same as the previous function, except that this function does not allow for complete movement of the agent. The agent cannot move into a row it has already visited and cannot traverse its current row. Exception: The final two rows. For example, if the bottom right image chip is the end state, and the agent reaches the bottom left image chip, then the agent can still find a path to the end state.

**Find\_optimal\_paths:**

This function creates a figure showing the optimal state/action for the agent in an HSI image. The input parameters are:

1. A vector containing the maximum rewarding actions of state, JJ = max(Q’)
2. An integer end state
3. The value field in matrix form

**Invert\_mat:**

This function takes an adjacency matrix following reinforcement learning, and inverts the nonzero values. E.g., inverts the Q matrix. The purpose of this function is for Dijkstra’s Algorithm. For example, a value of 90 which is a very rewarding action will become 1/90 which is very close to 0. Also, a value of 5 which is not very rewarding will become 1/5 which is closer to 1 than 1/90 is to 1.

So for the purpose of Dijkstra’s algorithm, the rewarding actions become ‘least distance’ and the not as rewarding actions become ‘most distant’.

Example function call: **[~,route] = dijkstra(invert\_mat(Q),1,322);**

**MeshCanopy:**

This function creates a figure which canopies MATLAB’s mesh() function over a grayscale image. This function is useful for illustrating a value field over the HSI image. The function parameters are:

1. A grayscale image. MATLAB’s rgb2gray() function can turn the HSI image into a grayscale.
2. A 2d matrix which is used in the mesh() function
3. A string which describes the theme. Not required, defaults to ‘jet’.
4. A double value which sets the initial z value where the mesh is plotted. 20 is default.

Example function call: **meshCanopy(rgb2gray(hsi\_image),imgaussfilt(value\_field.\*(1/8),1.5));**

**PCA:**

This function is used by a GUI, *part1.m line 271*. It takes an image chip and plots the PCA of that image chip, as well as the spatial gradient. Parameters are:

1. The image chip in matrix form (usually 32x32x256),
2. A GUI construct called “handles”, it allows the function to take control of a GUI so that it can update plots.

**Plot\_dijk:**

This function takes the result from Dijkstra’s algorithm which contains the optimal path from an initial node to an end node, and creates a figure which illustrates this path along with the grayscale HSI image and the mesh value field hovering above. The parameters are:

1. The optimal route
2. Value field in matrix form

**Q\_learning:**

This function teaches an agent the most rewarding actions/states through a graph. The input parameters are:

1. An Instant reward matrix, R
2. A learning parameter, value between 0 and 1
3. An integer end state

The outputs are:

1. The Q matrix, Q
2. The optimal actions for each state, JJ

**RandomPermutation:**

This function is used in Q-Learning and is an accessory function. It randomizes the list of possible actions from each state.

**Run\_Algorithm\_1:**

This function controls the first GUI, part1. It is responsible for cutting image chips, performing PCA, target dictionary learning, and plotting on the GUI. Its input parameters are:

1. X, which is the entire HSI image with all the wavelength bands. (E.g., 1480x256x256)
2. Handles, which is the construct used in controlling a GUI

The outputs are:

1. ATGP Spectral Dictionary
2. K-Means Clustering Spectral Dictionary
3. Image\_PCs\_cube, 32x32x15 matrix containing the PCA data for the final image chip.

**Run\_Algorithm\_2:**

This function controls the second GUI, part2. It plots spectral abundances, and spectral signatures. The input parameters are:

1. The struct “Data”, which is created following the completion of Run\_Algorithm\_1
2. Handles, which is the construct used in controlling a GUI

**Run\_Algortihm\_3:**

This function controls the third GUI, part3. It plots the first 6 Laplacian Eigenfaces for each image chip. The input parameters are:

1. The struct “Data”, which is created following the completion of Run\_Algorithm\_1
2. Handles, which is the construct used in controlling a GUI

**Run\_Algorithm\_4:**

This function is called following the Human Value GUI, or part4. It takes data from the GUI and applies Reinforcement Learning to the value field. The input parameters are:

1. The Value Field, in matrix form
2. Handles, which is the construct used in controlling a GUI

**SARSA:**

This function performs the State-Action-Reward-State-Action algorithm, and returns the “agent’s brain” or a Q matrix. The input parameters are:

1. An Instant Reward Matrix, R
2. A learning parameter, value between 0 and 1
3. An integer end state

**Data:**

**Machinelearning.mat** is a mat file containing the results of several SARSA and Q-Learning runs.

**Hsi\_image.mat** is a mat file containing the HSI image in rgb dimensions: 1480x256x3.

Example: load(“hsi\_image.mat”);

**Examples:**

Create a value field, specify Q-learning parameters, and produce a meshCanopy plot with the optimal path all from the part4 GUI. Or the process can be done via code in the following ways.

**Q-Learning – Dijkstra’s Algorithm:**

1. Run “part4” and label image chips with values.
2. Click “RESULTS” to create a value field. Or simply use a different value field (OMP, MF, etc.).
3. The code using a human value field from part4 GUI:



1. Q matrix now contains the agent’s brain, with complete freedom of movement.
2. Apply Dijkstra’s Algorithm to Q matrix to find an optimal path:



1. A new figure will be created with the illustration used in technical paper and presentation to illustrate the path and value field

**Q-Learning – Maximum Action Policy:**

1. Run “part4” and label image chips with values.
2. Click “RESULTS” to create a value field. Or simply use a different value field (OMP, MF, etc.).
3. The code using a human value field from part4 GUI:



1. Q matrix now contains the agent’s brain, with constrained movement
2. Apply maximal action policy to Q matrix to find an optimal path:



1. A new figure will be created with the illustration used in technical paper and presentation to illustrate the path and value field

**SARSA – Dijkstra’s Algorithm:**

Create a value field, specify SARSA parameters, and produce a meshCanopy plot with the optimal path all from the part4 GUI. Or the process can be done via code:

1. Run “part4” and label image chips with values.
2. Click “RESULTS” to create a value field. Or simply use a different value field (OMP, MF, etc.).
3. The code using a human value field from part4 GUI:



1. Q matrix now contains the agent’s brain, with complete freedom of movement.
2. Apply Dijkstra’s Algorithm to Q matrix to find an optimal path:



1. A new figure will be created with the illustration used in technical paper and presentation to illustrate the path and value field

**SARSA – Maximum Action Policy:**

Create a value field, specify SARSA parameters, and produce a meshCanopy plot with the optimal path all from the part4 GUI. Or the process can be done via code:

1. Run “part4” and label image chips with values.
2. Click “RESULTS” to create a value field. Or simply use a different value field (OMP, MF, etc.).
3. The code using a human value field from part4 GUI:



1. Q matrix now contains the agent’s brain, with constrained movement
2. Apply maximal action policy to Q matrix to find an optimal path:



1. A new figure will be created with the illustration used in technical paper and presentation to illustrate the path and value field

**MeshCanopy:**

Run the following commands:



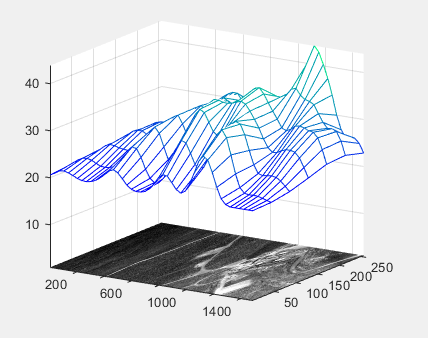


Figure 2: Example MeshCanopy