Working With the Lloyd's App: Supplemental Documentation

August 7, 2018

1 Functions

(i) `addToArray:` Adds an (x,y) value to a rx2 matrix while maintaining its order. If (x,y) is already in the matrix, then it leaves the matrix unchanged.

(ii) `arraySearch:` Searches an rx2 matrix for (x,y). If the matrix contains (x,y) then it returns found = 1 and index = the index that contains (x,y). If the matrix does not contain (x,y) then it returns found = 1 and index = the index where (x,y) should be inserted in order to maintain order.

(iii) `assign_points:` Returns a 1xN cell array, where the ith cell contains the ith agent's assigned points. Each agent is assigned the points within its observation radius that are closer to itself than to any other agent. The algorithm first fills the ith cell of *cellPoints* with the (x,y) points that each communication cell can observe. Once cellPoints has been filled, it distributes the points to their closest agents.

(iv) `bump:` Randomly moves an agent up, down, left, or right by $1/Partition_Number$. In the special case that the agent is on the boundary of the arena, the agent is moved towards the center.

(v) `centroid_finder:` Returns an Nx2 matrix, where (i,1) and (i,2) represent the ith agent's centroid's x and y value, respectively. Calculates the centroid using $\bar{x} = 1/M \cdot \Sigma(x \cdot D(x,y))$ and $\bar{y} = 1/M \cdot \Sigma(y \cdot D(x,y))$, the discrete version of the centroid formula taught in APSC 172.

(vi) `Check_Repetition:` Checks if any agents are in the same location and, if so, those agents are moved randomly using *bump*iv. Returns Agent_Positions in its usual format.

(vii) `Com_Graph:` Creates and updates the communication graph in the GUI.

(viii) `communication_fun:` Returns two matrices: first, com_mat is an NxN matrix where (j,i) = (i,j) = 1 when agent i and agent j can communicate with one another. Second, communication is a 1xsomething cell array where each cell is a vector containing the robots that are able to communicate with one another. For example, if one cell is the vector [1 2 4 7], then agents 1, 2, 4, 7 can communicate with one another and no one else.

(ix) `covered_mass_fun:` Returns the total amount of 'mass' covered by all of the agents. This is used to determine percent covered and its corresponding graph in the GUI.

(x) `Density_Position_Generator:` Returns two things: first, it returns Agent_Positions where each of the positions have been randomly generated (but remain within the arena). Second, it returns Density, which is either a randomly generated matrix containing *integer* values on the interval [1, 100], or a randomly generated polynomial function of x and y.

(xi) `distance_between:` Returns a scalar representing the distance between the two input points.

(xii) `Fixing_Starting_Positions:` Moves the agents to their nearest partition lines. Returns Agent_Positions in its usual format.

(xiii) `fun_to_array:` Returns a square matrix with dimensions sides*Partition_Number $\times$ sides*Partition_Number matrix representing the discretized density function.

(xiv) `iteration_variation_fun:` Returns the Density matrix which may change over time, if the user so chooses. Those who would like to have a time-varying density matrix should study and tweak this function. The algorithm uses *delay* to check if it is time to update the density matrix (it is updates every *delay* number of iterations).

(xv) `Lloyd_App.mlapp:` The main app. The central hub for the code, and visual display window. Calls *PlayButtonPushed* xvi.

(xvi) `PlayButtonPushed:` The 'brains' and, in a way, the 'main' of the program. This runs when the Play button is pushed.

(xvii) `mass_calc:` Returns a vector where the ith positions represents the mass that is within the ith agent's radius of observation. Note that this output is used in *centroid_finder*.

(xviii) `move_agents:` Moves the agents towards their centroid by an amount determined by `velocity_fun`. Returns three things: first, it returns an updated `Agent_Positions` in its usual format. Second, it returns `distance_travelled`, a vector that keeps track of the total distance travelled by all of the agents for the purposes of making the plot in the GUI. Third, it returns the energy array of the agents `E`. Those who wish to change the cost function

from `distance_travelled` to something else should study and tweak this function, as well `velocity_fun`.

(xix) `order_com_cell:` Sorts the communication cell array. Sorting this array increases the efficiency of *assign_points*.

(xx) `scomponents:` Open source code created by Stanford University. Determines which agents are able to communicate with one another.

(xxi) `sparse_to_csr:` Open source code created by Stanford University. Compresses a sparse array.

(xxii) `vectorSearch:` Vector version of *arraySearch* ii.

(xxiii) `velocity_fun:` Returns the Δx and Δy used in *move_agents*xviii. These values are determined by the user's choice of velocity, whether it be constant, proportional, or custom. Those who wish to make the velocity more interesting should study and tweak this function.

(xxiv) `saved_file2mat:` Converts a saved (.mat) file into a usable Matlab matrix.

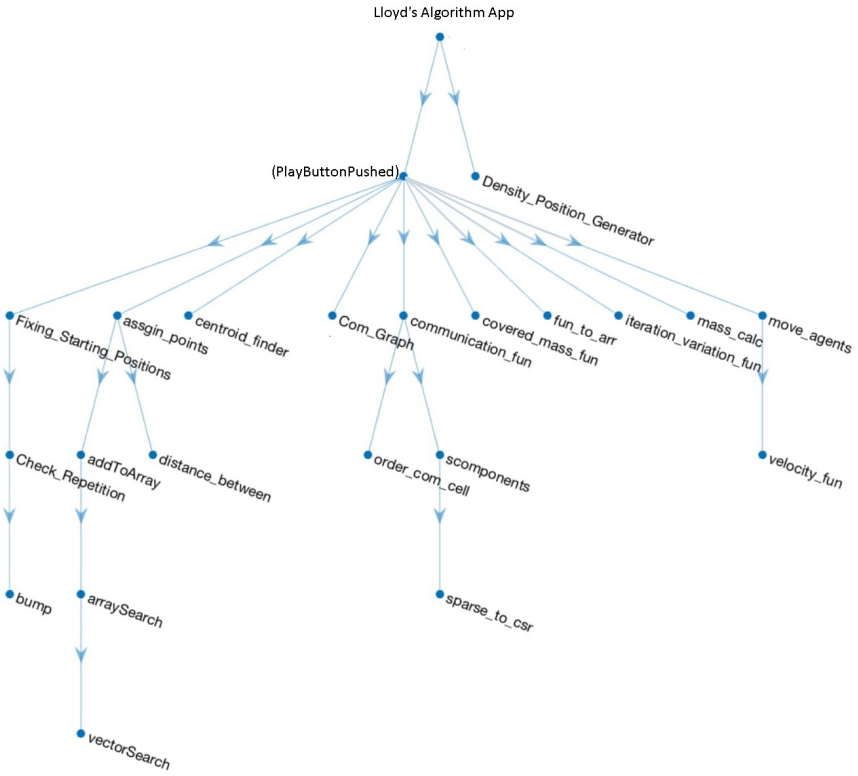
(xxv) `Plot_Assigned_Points:` Plots `Agent_Points` corresponding to each agent in one colour and the that agent's position in the same colour (in a diamond shape).

2 Functions Relationships Table

Function	Calls	Function Called By
addToArray(i)	arraySearch(ii)	assign_points(iii)
arraySearch(ii)	vectorSearch(xxii)	addToArray(i)
assign_points(iii)	addToArray(i), distance_between(xi)	PlayButtonPushed(xvi)
bump(iv)		Check_Repetition(vi)
centroid_finder(v)		PlayButtonPushed(xvi)
Check_Repetition(vi)	bump(iv)	(Fixing_Starting_Positions)(xii)
Com_Graph(vii)		PlayButtonPushed(xvi)
communication_fun(viii)	order_com_cell(xix), scomponents(xx)	PlayButtonPushed(xvi)
covered_mass_fun(ix)		PlayButtonPushed(xvi)
Density_Position_Generator(x)		Lloyd_App.mlapp(xv)
distance_between(xi)		assign_points(iii)
Fixing_Starting_Positions(xii)	Check_Repetition(vi)	PlayButtonPushed(xvi)
fun_to_arr(xiii)		PlayButtonPushed(xvi)
iteration_variation_fun(xiv)		PlayButtonPushed(xvi)
Lloyd_App.mlapp(xv)	PlayButtonPushed(xvi), Density_Position_Generator(x)	RAN BY USER
PlayButtonPushed(xvi)	assign_points(iii), centroid_finder(v), Com_Graph(vii), communication_fun(viii), covered_mass_fun(ix), Fixing_Starting_Positions(xii), fun_to_arr(xiii), iteration_variation_fun(xiv), mass_calc(xvii), move_agents(xviii)	Lloyd_App.mlapp(xv)
mass_calc(xvii)		PlayButtonPushed(xvi)
move_agents(xviii)	velocity_fun(xxiii)	PlayButtonPushed(xvi)
order_com_cell(xix)		communication_fun(viii)
scomponents(xx)	sparse_to_csr(xxi)	communication_fun(viii)
sparse_to_csr(xxi)		scomponents(xx)
vectorSearch(xxii)		arraySearch(ii)
velocity_fun(xxiii)		move_agents(xviii)

Table 1: Function Calls

3 Function Relations Graph



4 Extra Variables

1. `Density_Type`: Used to determine whether a function or a matrix was input as a density. If `Density_Type` = 1, then the input was a symbolic function in terms of x and y. If `Density_Type` = 2, then the input was a matrix,

2. `Velocity_Type`: Used to determine whether the rate at which agents move is a constant velocity, or a proportion of the distance,

3. `History`: Matrix of positions of agents used to save to Excel Sheet. x-position of agent i is in `History(:,2*i-1)`. The y-position of agent i is in `History(:,2*i)`.

4. `Density_variation_type`: Used to determine whether `Density` is *static* or *dynamic*. If `Density_variation_type` = 1, then `Density` is *static*. If `Density` = 2, then `Density` is *dynamic*,

5. `algorithm_type`: Used to determine whether **Regular Algorithm** or **Custom Algorithm** was selected in the GUI. If `algorithm_type` = 1, then **Regular Algorithm** was selected. If `algorithm_type` = 2, then **Custom Algorithm** was selected.