Documentation

leticiacremasco

July 2017

1 Introduction

RatSLAM is a SLAM system that has been derived from models of the hippocampal complex in rodents. The RatSLAM system uses an approximate computational model of the hippocampal complex based on competitive attractor networks. The packet (or packets) of activity in the competitive attractor network represent(s) the belief(s) of the robot with regard to its own pose. Movement of the robot modulates the dynamics of the network, causing the activity packet to change and hence update the pose estimate. Sensory cues become associated with activity packets. Once the associations between sensory cues and pose estimates are learned, the sensory cues will influence the position of the activity packet to update the pose estimate of the robot. [2]

RatSLAM, which consists of four nodes:

• Pose Cell Network—This node manages the energy packet that represents pose in response to odometric and local view connections. In this implementation, this node also makes decisions about experience map node and link creation.

• Local View Cells—This node determines whether a scene given by the current view is novel or familiar by using image comparison techniques.

• Experience Map—This node manages graph building, graph relaxation and path planning.

• Visual Odometry—For image only datasets, this node provides an odometric estimate based on changes in the visual scene.

The other nodes are unaware of the source of the odometric information.

In previous descriptions of RatSLAM, the decision process for creation of nodes and links was handled by the experience mapping module. In OpenRatSLAM, the Pose Cell Network node handles the decision on when to create new nodes and links because it requires knowledge of the internal workings of the Pose Cell Network, which is no longer available due to the split into separate nodes.

2 How to use the python version of RatSLAM

To use the python version of RatSLAM, it is necessary to have a video with all the frames that have to be processed to build the map, i.e., all the frames that

the robot sees during the navigation. To change define the video that will be used, it is important to change the line 11 of file main.py.

For each frame, the RatSLAM system has to produce the odometric information, find the visual template that matches the current frame, control the activity in the pose cell network and, finally, update the experience map. To do all of these steps, it is necessary to use only the function call of RatSLAM class, which receives the current frame as input. It also receives as input the information if the frame is in grayscale. Additionally, it is recommended to used frames in grayscale.

If it is also important to plot the real position of the robot, it is necessary to have a file with all the real positions of the robot. Then, it is important to change the line 23 of file main.py.

Finally, after processing all the frames, the last plot will save in the image file as result.jpg (line 128 of main.py).

3 Visual Odometry

The Visual Odometry node determines camera motion by comparing successive images. The node makes implicit assumptions that the camera motion is limited and that it has relatively constant speed. It allows for separate regions in the image to be specified for determining forward translational and rotational speeds.

The rotational velocity is estimated by determining what relative horizontal offset of two consecutive scanline profiles minimizes the mean of absolute differences between the two profiles. Scanline profiles are generated by summing the images in the vertical direction.

The translational velocity is estimated by multiplying the minimum difference by a scaling factor and limited to a maximum value to prevent spurious results from large changes in illumination. [1]

3.1 Attributes

vtrans-profile: 1D numpy array normalized (or intensity profile) that represents the current frame and it is used to calculate the translation velocity

vtrans-prev-profile: 1D numpy array normalized (or intensity profile) that represents the previous frame and it is used to calculate the translation velocity

vrot-profile: 1D numpy array normalized (or intensity profile) that represents the current frame and it is used to calculate rotational velocity

vrot-prev-profile: 1D numpy array normalized (or intensity profile) that represents the previous frame and it is used to calculate rotational velocity

first: controls if it is the first time that this class is being used

3.2 Methods

convert-view-to-view-template(self, current-view, view-rgb, grayscale, X-RANGE-MIN, X-RANGE-MAX, Y-RANGE-MIN, Y-RANGE-MAX):

Purpose:

This routine transforms the current frame in a 1D array normalized that can be used as a profile.

Algorithm:

It first defines the ranges of the current frame (view-rgb) that are going to be used, then sums all its values in the y-axis. Finally, the resulting array is divided by the range used for the y-axis (Y-RANGE-MAX - Y-RANGE-MIN) and by 255, if it is a grayscale frame. In case it is used an RGB frame, the algorithm is the same, however, the current view is also divided by 3.

Inputs:

current-view: numpy array that is going to store the 1D array normalized, or template produced

view-rgb: current frame captured by the robot

grayscale: True if the current frame is on grayscale, False if the current frame is not on grayscale

X-RANGE-MAX and X-RANGE-MIN: ranges in x-axis of the current frame that will be used

Y-RANGE-MAX and Y-RANGE-MIN: ranges in y-axis of the current frame that will be used

Outputs:

numpy array of 1D (X-RANGE-MAX - X-RANGE-MIN) normalized

visual-odo(self, data, width, olddata)

Purpose:

This routine calculates the translational and rotational velocity between the current frame and the previous frame.

Algorithm:

Compares data to olddata, first shifting only olddata by an offset value, then shifting only data by an offset value. The minimum difference is stored in mindiff and the minimum offset is stored in minoffset. Then, olddata is updated, receving data value. Finally, the rotational velocity and translational velocity are calculated.

Inputs:

data: 1D numpy array normalized (or intensity profile) that represents the current frame

width: size of data

olddata: 1D numpy array normalized (or intensity profile) that represents the previous frame

Outputs:

olddata updated, translational velocity and rotational velocity

on-image(self, data, greyscale)

Purpose:

This routine defines the sequence of steps necessary to calculate the translational velocity and rotational velocity between the current frame (data) and the previous frame.

Algorithm:

First, transforms the current frame in 1D array normalized (template), then calculates the translational velocity by comparing the current profile (vtrans-profile) to the previous profile (vtrans-prev-profile). For rotational velocity, it is the same steps used for translational velocity.

Inputs:

data: current frame

greyscale: True if the current frame is on grayscale, False if the current frame is not on grayscale

Outputs:

translational velocity and rotational velocity

4 Local View Match

The local view match node preprocesses the current image into a visual template representation and then decides if this is a new or previously seen visual template.

1. If required, the image is converted into a mono grayscale format.

2. The image may then be cropped to bias the templates towards visually interesting areas of the camera images. Visual interesting areas are those that will bias towards areas that are visually unique across the environment. For example, this process is used to remove visually bland features such as roads.

3. The cropped region may then be subsampled to defined height and width parameters. The scanline intensity profile is a generalization by subsampling with the height parameter equal to one.

4. The subsampled region may then undergo global and local normalization steps which attempt to alleviate changes in illumination. Global normalization considers the mean and range of the entire image and addresses global changes in illumination. Local normalization preserves contrast in small patch regions by subtracting from each pixel the mean intensity of the surrounding patch region and then dividing by the patch's pixel intensity standard deviation.

After preprocessing, the local view match node compares the visual template that represents the current camera image with all previously learned templates. A similarity measure based on the Sum of Absolute Differences (SAD) between the current visual template and each previously learned visual template is calculated. If the smallest difference is less than a threshold, then the corresponding learned template is selected. Otherwise, the current visual template is added to the database of templates.

For forward facing cameras, the comparison process handles small rotational offsets by finding the minimum SAD while shifting the stored templates relative

to the current view in the horizontal direction by a parameterized amount. For the panoramic camera's views SAD is calculated while shifting the visual templates and current view through a full rotation.

[1]

4.1 Attributes

templates: 1D numpy array that stores all visual templates created

current-view: 1D numpy array that stores the current template relative to the current frame

current-mean: current mean of the template current-view

current-vt: id of the current visual template

vt-error: current error between the current visual template and the closest visual template

prev-vt: id of the previous visual template

vt-relative-rad: relative facing angle between closest visual template and current visual template

view-rgb: store the current frame captured by the robot

greyscale: controls if the image is in grayscale

4.2 Methods

clip-view-x-y(self,x,y):

Purpose:

This routine verifies if the value of x and y coordinates are in a valid range (between 0 and TEMPLATE-X-SIZE or TEMPLATE-Y-SIZE), and if they are not, corrected their value.

Algorithm:

Verifies if the value of x and y coordinates are in a valid range (between 0 and TEMPLATE-X-SIZE or TEMPLATE-Y-SIZE), and if they are not, correct their values to 0 or TEMPLATE-SIZE, depending which one is closer to the current value of x and y.

Inputs:

x: x coordinate

y: y coordinate

Outputs:

x and y updated

convert-view-to-view-template(self, grayscale)

Purpose:

This routine transforms the current frame in a 1D array normalized that can be used as a template.

Algorithm:

It first defines the ranges of the current frame (self.view-rgb) that are going to be used. Then it sums blocks of pixels of size x-block-size * y-block-size. Then, the resulting array is divided by x-block-size * y-block-size and by 255,

if it is a grayscale frame. In case it is used an RGB frame, the algorithm is the same, however, the current view is also divided by 3. If VT-NORMALISATION ¿ 0, all values will be normalized by this value. Finally, the current mean is calculated.

Inputs:

greyscale: True if the current frame is on grayscale, False if the current frame is not on grayscale.

Outputs:

numpy array of 1D (TEMPLATE-X-SIZE * TEMPLATE-Y-SIZE) normalized

set-current-vt(self, current-vt)

Purpose:

This routine updates the current visual template (current-vt).

Algorithm:

Verifies if the current-vt is different from the input, if it is, updates the prev-vt and current-vt values, if it is not, just updates the current-vt.

Inputs:

current-vt: id of the current visual template

Outputs:

-

create-template(self)

Purpose:

This routine creates a new VisualTemplate (Localview cell) and add this to the collection.

Algorithm:

Create a new VisualTemplate object that stores the current frame and the current mean of this frame. Then, this new VisualTemplate is added to the collection.

Inputs:

-

Outputs:

Id of the VisualTemplate created

compare(self, vt-err, vt-match-id)

Purpose:

This routine compares a visual template to all the stored templates, returning the closest template and the error between these two.

Algorithm:

For each visual template, tries matching the view at different offsets. After finding the smallest offset, calculates the error between these two visual templates, the matching id and the relative angle between them.

Inputs:

vt-err: variable that will store the error between two closest visual templates

vt-match-id: variable that will store the closest template id

Outputs:
the error between the two matching visual templates and the id of the closest template

save(self)
Purpose:
This routine saves all the visual templates stored in the collection.
Algorithm:
Create a file and store first the id, then the mean and the data of visual template.
Inputs:
-
Outputs:
-

load(self)
Purpose:
This routine loads all the visual templates saved and add them to the collection.
Algorithm:
Open a file with all the visual template stored and for each visual template, create a new VisualTemplate object and store it in the collection.
Inputs:
-
Outputs:
-

5 Pose cell network

The pose cell network node responds to two types of input; odometry and view templates. The action on a view template input depends on whether this is a new or existing view template. For new view templates, the id is associated with the centroid of the current peak activity packet in the pose cell network. For existing view templates, activity is injected into the previously associated location in the pose cells. The injected activity for consecutive matches of the same view template decays rapidly but is gradually restored over time. Because RatSLAM has no explicit motion model, this decay process is necessary to avoid potentially incorrect re-localizations when the robot is motionless for long periods of time.

For odometry input, these steps are performed in the following order. Note that the Pose Cell node is unaware of the source of the odometry input, and for example could be from visual odometry or wheel odometry, and either from the dataset file or calculated live. Also, note that each of the flowing steps accounts for the wrapping across each face of the pose cell network rectangular prism.

1. Local excitation where energy is added around each active pose cell.
2. Local inhibition where energy is removed around each active pose cell. These first two steps ensure the stabilization of the energy packets.
3. Global inhibition where energy is removed from all active pose cells but not below zero.
4. Network energy normalization to ensure the total energy in the system is equal to one. This stage ensures the stability of the global pose cell system.
5. Use the odometric information for path integration by shifting the pose cell energy.
6. Identify the centroid of the dominant activity packet in the network.

After performing these steps, the node must determine an action for the experience map's topological graph. The possible actions are: create a new node (which implicitly includes creating an edge from the previous node), create an edge between two existing nodes or set the location to an existing node.

[1]

5.1 Attributes

best_x : $xcoordinateoftheassociatedposecellbest_y$: $ycoordinateoftheassociatedposecellbest_th$: $thcoordinateoftheassociatedposecellvt_delta_pc_th$: $relativeangledifferencebetweenclosestvisualtemplateandc$ $controlsiftheodometrywasanalyzedvt_update$: $controlsifinfluenceofthevisualtemplatewasanalyzed$

posecells: pose cell network activity $\text{visual}_templates$: $collectionofposecellvisualtemplatesexperiences$: $collectionofposecellexperiences$

current_vt : $idofcurrentvisualtemplateprev_vt$: $idofpreviousvisualtemplatecurrent_exp$: $idofcurrentposecellexperienceprev_exp$: $idofpreviousposecellexperience$

5.2 Methods

inject(self, act-x, act-y, act-th, energy):

Purpose:

This routine injects energy into a specific point in the network.

Algorithm:

Check if the point is a valid point in the pose cell network, then add the energy at this specific point.

Inputs:

act_x : $xcoordinatethepointthatwillreceivetheenergyact_y$: $ycoordinatethepointthatwillreceivetheenergyac$ $zcoordinatethepointthatwillreceivetheenergyenergy$: $thevaluethatshouldbeinjectedintheposecellnetwork$

Outputs:

True, if energy is injected otherwise, False

excite(self)

Purpose:

This routine locally excites points in the pose cell network, spreading energy through the network.

Algorithm:

Find which cells in the pose cell have energy. Then spread the energy locally, which the range of cells affected are defined by $\mathrm{PC}_{WED}IMmatrixandtheweightofexcitatoryconnectionsaresto$

Inputs:

-

Outputs:

-

inhibit(self)

Purpose:

This routine locally inhibits points in the pose cell network, compressing through the network.

Algorithm:

Find which cells in the pose cell have energy. Then spread the energy locally, which the range of cells affected are defined by $\mathrm{PC}_{WID}IMmatrixandtheweightofinhibitoryconnectionsareston$

Inputs:

-

Outputs:

-

global-inhibit(self)

Purpose:

This routine is responsible for the global inhibition process.

Algorithm:

For all cells that have more energy than $\mathrm{PC}_{GLOBAL_INHIB}threshold, thisvaluewillbesubtracted.Forthere$

Inputs:

-

Outputs:

-

normalize(self)

Purpose:

This routine normalizes all the energy in the system.

Algorithm:

Divide all values by the total energy in the system

Inputs:

-

Outputs:

-

path-integration(self)

Purpose:

This routine shifts the energy in the system by a translational and rotational velocity.

Algorithm:

First, scale the translational velocity. Then, shift the pose cell network in each th plane given by the th. Rotate the pose cell network instead of

implementing for four quadrants. Extend the pc.Posecells one unit in each direction work out the weight contribution to the NE cell from the SW, NW, SE cells given vtrans and the direction think in terms of NE divided into 4 rectangles with the sides given by vtrans and the angle. Circular shift and multiple by the contributing weight copy those shifted elements for the wrap around. Unrotate the pose cell xy layer. Finally, shift the pose cells +/- theta given by vrot mod to work out the partial shift amount.

Inputs:

vtrans: translational velocity vrot: rotational velocity

Outputs:

-

find-best(self)

Purpose:

This routine finds an approximation of the center of the energy packet.

Algorithm:

First, find the max activated cell. Second, locate de cells that are in the area of $\mathrm{PC}_C ELLS_T O_A VGdistance.Third, getthesumsforeachaxis.Then, findthe(x, y, th)usingpopulationvectord$

Inputs:

-

Outputs:

position of the centre of the energy packet in the pose cell network

get-min-delta(self, d1, d2, maximo)

Purpose:

This routine finds the smallest distance between two specific points in the pose cell network in one of the axis, respecting the wrap connections.

Algorithm:

Calculates the difference between the first two inputs. Then, calculates the other distance between these two cells due to the wrap connections (maximo - absval). Finally, evaluate which one is the smaller.

Inputs:

d1: first coordinate d2: second coordinate maximo: posecell dimension

Outputs:

smaller distance between two specific points in the pose cell network in one of the axis

get-delta-pc(self, x, y, th)

Purpose:

This routine calculates the distance between a specific position and the pose cell with the highest energy.

Algorithm:

Adjusts the orientation of the robot, subtracting the value of the heading direction variation between the current visual template and the previous one. Then, calculates the distance between the between a specific position and the pose cell with the highest energy.

Inputs:

x: x coordinate of a specific position y: y coordinate of a specific position th: th coordinate of a specific position

Outputs:

distance between a specific position and the pose cell with the highest energy

create-experience(self)

Purpose:

This routine creates a new PosecellExperience object and add this to the collection.

Algorithm:

Find the current PosecellVisualTemplate, update current$_e xp with the id of the new Posecell Experience and cre$

Inputs:

-

Outputs:

id of the new PosecellExperience object

get-action(self)

Purpose:

This routine determines an action for the experience map's topological graph.

Algorithm:

First, check if odometry and visual template inputs were processed. Then, go through all the experiences associated with the current view and find the one closest to the current center of activity packet in the pose cell network. If an experience is closer than the threshold, creates a link. If there is an experience matching the current and exceeds the threshold, then the current experience should set to the previous one. Otherwise, creates a new experience.

Inputs:

-

Outputs:

action and matched experience

on-odo(self, vtrans, vrot, time-diff-s)

Purpose:

This routine process the odometry information and start the dynamic in the pose cell network (excitation, inhibition, global inhibition and path integration processes).

Algorithm:

First, the pose cell network is locally excited, where energy is added around each active pose cell. Second, the pose cell network is locally inhibited, where energy is removed around each active pose cell. These first two steps ensure the stabilization of the energy packets. Third, global inhibition process happens, where energy is removed from all active pose cells but not below zero. Then, network energy normalization occurs to ensure the total energy in the system is equal to one. This stage ensures the stability of the global pose cell system.

Then, path integration occurs, by shifting the pose cell energy. Finally, the centroid of the dominant activity packet in the network is identified.

Inputs:

vtrans: translational velocity vrot: rotational velocity time$_diffs_s$: $timedifferencebetweenthepreviousposi$

Outputs:

-

create-view-template(self)

Purpose:

This routine creates a new PosecellVisualTemplate object and add this to the collection.

Algorithm:

Create a new PosecellVisualTemplate object and then add this to the collection.

Inputs:

-

Outputs:

-

on-view-template(self, vt, vt-rad)

Purpose:

This routine decides which action on a view template will be taken; inject energy or associate the current peak of activity to the view template.

Algorithm:

The action on a view template input depends on whether this is a new or existing view template. For new view templates, the id is associated with the centroid of the current peak activity packet in the pose cell network. For existing view templates, activity is injected into the previously associated location in the pose cells. The injected activity for consecutive matches of the same view template decays rapidly but is gradually restored over time. Because RatSLAM has no explicit motion model, this decay process is necessary to avoid potentially incorrect re-localizations when the robot is motionless for long periods of time.

Inputs:

vt: id of the current visual template vt$_rad$: $relativeanglebetweenclosestvisualtemplateandcurrentvisualte$

Outputs:

-

get$_relative_rad(self)$

Purpose:

This routine returns the relative angle between closest visual template and current visual template in radians.

Algorithm:

Calculates the relative angle between the closest visual template and current visual template in radians.

Inputs:

-

Outputs:
the relative angle in radians

save(self)
Purpose:
This routine saves all the pose cell visual templates stored in the collection and all the pose cell experiences stored in the collection.
Algorithm:
Create a file to store information about all pose cell visual templates (id, position in the pose cell network, decay), then create a file to store information about all pose cell experiences (position in the pose cell network, the id of the visual template associated).
Inputs:
-
Outputs:
-

load(self)
Purpose:
This routine loads all the visual templates saved and add them to the collection
Algorithm:
Open a file with all the pose cell visual template stored and for each pose cell visual template, create a new PosecellVisualTemplate object and store it in the collection. Open a file with all the pose cell experiences stored and for each pose cell experience, create a new PosecellExperience object and store it in the collection.
Inputs:
-
Outputs:
-

6 Experience map

The Experience Map node uses the received actions from Pose cell network to create nodes and links or to set the current node. In this implementation, each experience has an associated position and orientation. Creating a new node also creates a link to the previously active node. Graph relaxation is performed on each action interaction. A link encapsulates the pose transformation and time between nodes based the odometric information. The agent's state is given by the current experience map node and the agent's rotation relative to the node's orientation.
[1]

6.1 Attributes

experiences: collection of Experience objects links: collection of Link objects

current-exp-id: id of the current experience prev-exp-id: id of the previous experience

accum-delta-facing: orientation movement due to the rotation accum-delta-x: space traveled in x axis due to translation accum-delta-y: space traveled in y axis due to translation accum-delta-time-s: time interval from the last position to the current

relative-rad: relative angle difference between closest visual template and current visual template

6.2 Methods

on-create-experience(self, id, vt-id):

Purpose:

This routine creates a new experience in the experience map. This experience will be associated with a visual template with vt-id as id.

Algorithm:

Creates a new experience, sets the position of this experience due to the translational movement, adds the new experience to the collection, then creates a link between the current experience and the previous one.

Inputs:

id: id of the new experience

vt-id: id of the visual template associated with this experience

Outputs:

the id of the new experience

on-odo(self, vtrans, vrot, time-diff-s)

Purpose:

This routine updates the current position of the experience map since the last experience.

Algorithm:

Calculates the traveled distance due to translational movement and the rotation. Then, updates the variables that control the position traveled in the experience map.

Inputs:

vtrans: translational velocity

vrot: rotational velocity time-diff-s: time interval between the current experience and the previous

Outputs:

-

iterate(self)

Purpose:

This routine iterates the experience map. Perform a graph relaxing algorithm to allow the map to partially converge.

Algorithm:

Corrects the position of all experiences in the experience map based on the stored link information. A 0.5 correction parameter means that e0 and e1 will be fully corrected based on e0's link information.

Inputs:

-

Outputs:

-

on-create-link(self, exp-id-from, exp-id-to, rel-rad)

Purpose:

This routine creates a new experience in the experience map. This experience will be associated with a visual template with vt-id as id.

Algorithm:

Checks if the current link already exists, then creates the new link and add this to the collection. Finally, adds the id of the new link to the collection of the experiences involved.

Inputs:

exp-id-from: the link starts at this experience
exp-id-to: the link ends at this experience
rel-rad: relative angle between these two experiences

Outputs:

-

on-set-experience(self, new-exp-id, rel-rad)

Purpose:

This routine changes the current experience.

Algorithm:

Check if it is necessary to make this change or if it is possible, then, update the previous and current experiences id. Finally, update the parameters of this class.

Inputs:

new-exp-id: id of the current experience
rel-rad: relative head direction between the two experiences

Outputs:

0, if the new experience was not changed to new-exp-id, 1, otherwise

get-status(self)

Purpose:

This routine returns the experiences that the robot can achieve from the current experience.

Algorithm:

First, check if there is more than one experience. Second, find all the links from the current experience and identify the experiences where these links end. Finally, find all the links that end at the current experience and identify the experiences where these links start.

Inputs:

-

Outputs:

id of the current experience and a list of experiences that the robot can achieve

get-distance-and-direction(self)

Purpose:

This routine returns distance and direction to achieve the exp-id from current experience.

Algorithm:

First, find the links that the two experiences have in common. Finally, find the link with the smallest distance.

Inputs:

exp-id: id of the experience that the robot wants to achieve

Outputs:

distance and direction to achieve the exp-id from the current position

save(self)

Purpose:

This routine saves all the experiences, links and goals stored in the collection.

Algorithm:

Create a file to store information about all experiences (id, position in the experience map, id of the visual template associated, collection of links to, collection of links from, time from current, goal to current, and current to goal), then create a file to store information about all links (distance, heading direction, facing direction, experience to, experience from, time between these two experiences), finally create a file to save the list of goals.

Inputs:

-

Outputs:

-

load(self)

Purpose:

This routine loads all the experiences, links and goals saved and add them to the collection.

Algorithm:

Opens a file with information about all experiences (id, position in the experience map, id of the visual template associated, collection of links to, collection of links from, time from current, goal to current, and current to goal) and, for each experience, create an Experience object and add this to the collection. It does the same for links. For goals, adds each goal stored to a list of goals.

Inputs:

-

Outputs:

-

7 RatSLAM

This class controls all the other classes. It defines the sequence of steps necessary to process each frame and build the experience map. The main procedure in this class starts calculating the odometry information. After this step, the next one is to find the matching visual template. Then, the activity in the pose cell network is updated, the center of the energy packet is found and the action for the experience map's topological graph is determined.

7.1 Attributes

visual-odometry: object of VisualOdometry class that controls odometry information visual-templates: object of LocalViewMatch class that controls the matching scenes network: object of PoseCellNetwork class that controls activity in the pose cell network and the actions in the experience map map: object of ExperienceMap class vtrans: translational velocity of the robot vrot: rotational velocity of the robot prev-vt: id of the previous visual template time-diff: the time difference between frames

7.2 Methods

call(self, img, greyscale):

Purpose: **This routine updates the current position of the experience map since the last experience.**

Algorithm:

First, calculates the translational velocity and rotational velocity between the current frame (data) and the previous frame. Second, finds the matching template and the angle between the current and previous template. Third, determines an action for the experience map's topological graph. Finally, updates the current position of the

experience map since the last experience, performs the action from pose cell network and graph relaxation.

Inputs:

img: current frame greyscale: True if the current frame is on grayscale, False if the current frame is not on grayscale

Outputs:

-

save(angle)

Purpose:

This routine saves all the visual template, pose cell and experience map information.

Algorithm:

Creates files for each object and stores all the information.

Inputs:

-

Outputs:

-

load(self)

Purpose:

This routine loads all the visual template, pose cell and experience map information.

Algorithm:

Opens files and loads all the information.

Inputs:

-

Outputs:

-

8 Utils

This is not a class. It is just a file with accessory functions that perform some basic operations with angles.

8.1 Functions

clip-rad-360(angle):

Purpose:

This routine returns the corresponding angle in the range between 0 and 2 * pi.

Algorithm:

Calculates the corresponding angle in the range between 0 and 2 * pi.

Inputs:
angle: any angle in radians
Outputs:
an angle between 0 and 2 * pi

clip-rad-18(angle)
Purpose:
This routine returns the corresponding angle in the range between - pi and pi.
Algorithm:
Calculates the corresponding angle in the range between - pi and pi.
Inputs:
angle: any angle in radians
Outputs:
an angle between - pi and pi

get-signed-delta-rad(angle1, angle2)
Purpose:
This routine calculates the delta between two angles in radians in the range between 0 and 2 * pi.
Algorithm:
First, calculates if the delta is positive or negative. Then, calculates the smallest delta between the two angles in the range between 0 and 2 * pi.
Inputs:
angle1: any angle in radians
angle2: any angle in radians
Outputs:
the delta between two angles in radians in the range between 0 and 2 * pi

9 RatSLAM parameters

Parameters that you should change:
IMAGE-WIDTH
IMAGE-HEIGHT

VTRANS-IMAGE-X-MIN
VTRANS-IMAGE-X-MAX
VTRANS-IMAGE-Y-MIN
VTRANS-IMAGE-Y-MAX

VROT-IMAGE-X-MIN

VROT-IMAGE-X-MAX
VROT-IMAGE-Y-MIN
VROT-IMAGE-Y-MAX

VT-MATCH-THRESHOLD

TEMPLATE-X-SIZE
TEMPLATE-Y-SIZE

IMAGE-VT-X-RANGE-MIN
IMAGE-VT-X-RANGE-MAX
IMAGE-VT-Y-RANGE-MIN
IMAGE-VT-Y-RANGE-MAX

PC-VT-INJECT-ENERGY

The current RatSLAM implementation is the result of years of tuning, especially of the pose cell network parameters, in order to generate stable system dynamics that appropriately filtered ambiguous visual sensory input. The parameters given for the experiments performed in the OpenRatSLAM paper serve as a good basis for experiments in large outdoor and smaller indoor environments. Most importantly, the core pose cell parameters do not need to be changed unless one is interested in investigating the effect of varying network dynamics. In general, the only parameters that may need tuning from the defaults given are the vt-match-threshold and, in some circumstances, pc-vt-inject-energy. Please check the parameter tuning process at page 13 of the OpenRatSLAM paper. [1]

References

[1] **D. Ball, S. Heath, J. Wiles, G. Wyeth, P. Corke, M. Milford. OpenRatSLAM: An open source brain-based SLAM system. *Autonomous Robots*, pages 1–28, 2013.**

[2] **M. J. Milford, G. F. Wyeth, D. Prasser. RatSLAM: A Hippocampal Model for Simultaneous Localization and Mapping. *presented at IEEE International Conference on Robotics and Automation, New Orleans, USA*, 2004.**