BlueSim Documentation

Release 0.0.1

Florian Berlinger
Fritz Lekschas

Nov 20, 2018

CONTENTS:

BlueSim is a simulator for BlueBots.

CHAPTER ONE

FISH

class fish.**Fish**(*id, channel, interaction, lim_neighbors=[0, inf], fish_max_speed=1, clock_freq=1, neighbor_weight=1.0, name='Unnamed', verbose=False*)

This class models each fish robot node in the network from the fish' perspective.

Each fish has an ID, communicates over the channel, and perceives its neighbors and takes actions accordingly. In taking actions, the fish can weight information from neighbors based on their distance. The fish aims to stay between a lower and upper limit of neighbors to maintain a cohesive collective. It can move at a maximal speed and updates its behavior on every clock tick.

communicate()

Broadcast all collected event messages.

This method is called as part of the second clock cycle.

comp_center(*rel_pos*)

Compute the (potentially weighted) centroid of the fish neighbors

Arguments:

rel_pos {dict} – Dictionary of relative positions to the neighboring fish.

Returns: np.array – 3D centroid

eval()

The fish evaluates its state

Currently the fish checks all responses to previous pings and evaluates its relative position to all neighbors. Neighbors are other fish that received the ping element.

homing_handler(*event, pos*)

Homing handler, i.e., make fish aggregated extremely

Arguments: event {Homing} – Homing event pos {np.array} – Position of the homing event initialtor

hop_count_handler(*event*)

Hop count handler

Initialize only of the last hop count event is 4 clocks old. Otherwise update the hop count and resend the new value only if its larger than the previous hop count value.

Arguments: event {HopCount} – Hop count event instance

info_ext_handler(*event*)

External information handler

Always accept the external information and spread the news.

Arguments: event {InfoExternal} – InfoExternal event

info_int_handler(*event*)
Internal information event handler.

Only accept the information of the clock is higher than from the last information

Arguments: event {InfoInternal} – Internal information event instance

leader_election_handler(*event*)
Leader election handler

Arguments: event {LeaderElection} – Leader election event instance

log(*neighbors={}*)
Log current state

move(*neighbors*, *rel_pos*)
Make a cohesion and target-driven move

The move is determined by the relative position of the centroid and a target position and is limited by the maximum fish speed.

Arguments:

neighbors {set} – Set of active neighbors, i.e., other fish that responded to the most recent ping event.

rel_pos {dict} – Relative positions to all neighbors

Returns: np.array – Move direction as a 3D vector

move_handler(*event*)
Handle move events, i.e., update the target position.

Arguments: event {Move} – Event holding an x, y, and z target position

ping_handler(*neighbors*, *rel_pos*, *event*)
Handle ping events

Adds the

Arguments:

neighbors {set} – Set of active neighbors, i.e., nodes from which this fish received a ping event.

rel_pos {dict} – Dictionary of relative positions from this fish to the source of the ping event.

event {Ping} – The ping event instance

run()
Run the process recursively

This method simulates the fish and calls *eval* on every clock tick as long as the fish *is_started*.

start()
Start the process

This sets *is_started* to true and invokes *run()*.

start_hop_count_handler(*event*)
Hop count start handler

Always accept a new start event for a hop count

Arguments: event {StartHopCount} – Hop count start event

start_leader_election_handler(*event*)

Leader election start handler

Always accept a new start event for a leader election

Arguments: event {StartLeaderElection} – Leader election start event

stop()

Stop the process

This sets *is_started* to false.

update_behavior()

Update the fish behavior.

This actively changes the cohesion strategy to either 'wait', i.e, do not care about any neighbors or 'signal_aircraft', i.e., aggregate with as many fish friends as possible.

In robotics 'signal_aircraft' is a secret key word for robo-fish-nerds to gather in a secret lab until some robo fish finds a robo aircraft.

weight_neighbor(*rel_pos_to_neighbor*)

Weight neighbors by the relative position to them

Currently only returns a static value but this could be tweaked in the future to calculate a weighted center point.

Arguments: rel_pos_to_neighbor {np.array} – Relative position to a neighbor

Returns: float – Weight for this neighbor

CHAPTER TWO

ENVIRONMENT

class environment.**Environment**(*node_pos*, *distortion*, *prob_type='quadratic'*, *conn_thres=inf*, *conn_drop=1*, *noise_magnitude=0.1*, *verbose=False*)

The dynamic network of robot nodes in the underwater environment

This class keeps track of the network dynamics by storing the positions of all nodes. It contains functions to derive the distorted position from a target position by adding a distortion and noise, to update the position of a node, to update the distance between nodes, to derive the probability of receiving a message from a node based on that distance, and to get the relative position from one node to another node.

get_distorted_pos(*source_index*, *target_pos*)

Calculate the distorted target position of a node.

This method adds random noise and the position-based distortion onto the ideal target position to calculate the final position of the node.

Arguments:

source_index {int} – Index of the source node which position is to be distorted.

target_pos {np.array} – Ideal target position to be distorted

Returns: np.array – Final position of the node.

get_rel_pos(*source_index*, *target_index*)

Calculate the relative position of two nodes

Calculate the vector pointing from the source node to the target node.

Arguments:

source_index {int} – Index of the source node, i.e., the node for which the relative position to target is specified.

target_index {int} – Index of the target node, i.e., the node to which source is relatively positioned to.

Returns: np,array – Vector pointing from source to target

prob(*node_a_index*, *node_b_index*)

Calculate the probability of connectivity of two points based on their Eucledian distance.

Arguments: node_a_index {int} – Node A index node_b_index {int} – Node B index

Returns: float – probability of connectivity

prob_binary(*distance*)

Simulate binary connectivity probability

This function either returns 1 or 0 if the distance of two nodes is smaller (or larger) than the user defined threshold.

Arguments: distance {float} – Eucledian distance

Returns:

float – probability of connectivity. The probability is either 1 or 0 depending on the distance threshold.

prob_dist (*distance*)

Calls the approriate probability functions

The returned probability depends on prob_type

Arguments: distance {float} – Eucledian distance

Returns: float – probability of connectivity

prob_quadratic (*distance*)

Simulate quadradic connectivity probability

Arguments: distance {float} – Eucledian distance

Returns:

float – probability of connectivity as a function of the distance. The probability drops quadratically.

prob_sigmoid (*distance*)

Simulate sigmoid connectivity probability

Arguments: distance {float} – Eucledian distance

Returns:

float – probability of connectivity as a sigmoid function of the distance.

set_pos (*source_index*, *new_pos*)

Set the new position

Save the new position into the positions array.

Arguments: source_index {int} – Index of the node position to be set new_pos {np.array} – New node position ([x, y, z]) to be set.

update_distance ()

Calculate pairwise distances of every node

Calculate and saves the pairwise distance of every node.

CHAPTER

THREE

INTERACTION

class interaction.**Interaction**(*environment*, *verbose=False*)

Underwater interactions

This class models interactions of the fish with their environment, e.g., to perceive other fish or to change their position.

move(*source_id*, *target_direction*)

Move a fish

Moves the fish relatively into the given direction and adds target-based distortion to the fish position.

Arguments: source_id {int} – Fish identifier target_direction {np.array} – Relative direction to move to

perceive_object(*source_id*, *pos*)

Perceive the relative position to an object

This simulates the fish's perception of external sources and targets.

Arguments:

source_id {int} – Index of the fish that wants to know its location

pos {np.array} – X, Y, and Z position of the object

perceive_pos(*source_id*, *target_id*)

Perceive the relative position to another fish

This simulates the fish's perception of neighbors.

Arguments: source_id {int} – Index of the fish to be perceived target_id {int} – Index of the fish to be perceived

CHAPTER FOUR

CHANNEL

class channel.**Channel**(*environment*, *verbose=False*)

Underwater wireless communication channel

This class models the underwater communication between fish instances and connects fish to the environmental network.

intercept(*observer*)

Let an observer intercept all messages.

It's really unfortunate but there are not just holes in Swiss cheese. Our channel is no exception and a god-like observer is able to listen to all transmitted messages in the name of research. Please don't tell anyone.

Arguments: observer {Observer} – The all mighty observer

set_nodes(*nodes*)

This method just stores a references to all nodes

Arguments: nodes {list} – List of node instances

transmit(*source*, *event*, *pos=array([0., 0., 0.])*, *is_observer=False*)

Transmit a broadcasted event to node instances

This method gets the probability of connectedness between two nodes from the environment and adds the events on the node instances given that probability.

Arguments: source {*} – Node instance event {Event} – Some event to be broadcasted

CHAPTER FIVE

OBSERVER

class observer.**Observer**(*environment*, *fish*, *channel*, *clock_freq=1*, *fish_pos=None*, *verbose=False*)

The god-like observer keeps track of the fish movement for analysis.

activate_reset()

Activate automatic resetting of the fish positions on a new instruction.

check_info_consistency()

Check consistency of a tracked information

check_instructions()

Check external instructions to be broadcasted.

If we reach the clock cycle in which they should be broadcasted, send them out.

check_transmissions()

Check intercepted transmission from the channel

deactivate_reset()

Deactivate automatic resetting of the fish positions on a new instruction.

eval()

Save the position and connectivity status of the fish.

instruct(*event*, *rel_clock=0*, *fish_id=None*, *pos=array([0., 0., 0.])*, *fish_all=False*)

Make the observer instruct the fish swarm.

This will effectively trigger an event in the fish environment, like an instruction or some kind of obstacle.

Arguments: event {*} – Some event instance.

Keyword Arguments:

rel_clock {number} – Number of relative clock cycles from now when to broadcast the event (default: {0})

fish_id {int} – If not *None* directly put the event on the fish with this id. (default: {None})

pos {np.array} – Imaginary event position. Used to determine the probability that fish will hear the event. (default: {np.zeros(2,)})

fish_all {bool} – If *true* all fish will immediately receive the event, i.e., no probabilistic event anymore. (default: {False})

plot(*dark=False*, *white_axis=False*, *no_legend=False*, *show_bar_chart=False*, *no_star=False*)

Plot the fish movement

run()

Run the process recursively

This method simulates the fish and calls *eval* on every clock tick as long as the fish *is_started*.

start()

Start the process

This sets *is_started* to true and invokes *run()*.

stop()

Stop the process

This sets *is_started* to false.

CHAPTER SIX

EVENTS

class events.**Homing**
Homing towards an external source

class events.**HopCount**(*id*, *clock*, *hops=0*)
Broadcast hop counts

A funny side note: in Germany distributed and DNA-based organisms (often called humans) shout "Hop Hop rin in Kopp", which is a similar but slightly different event type that makes other human instances to instantly enjoy a whole glass of juicy beer in just a single hop! Highly efficient!

class events.**InfoExternal**(*message*, *track=False*)
Share external information with fish

class events.**InfoInternal**(*id*, *clock*, *message*, *hops=0*)
Share information internally with other fish

class events.**LeaderElection**(*id*, *max_id*)
Broadcast a leader election

class events.**Move**(*x=0*, *y=0*, *z=0*)
Make the fish move to a target direction

class events.**Ping**(*id*)
Ping your beloved neighbor fish

class events.**StartHopCount**
Initialize a hop count.

class events.**StartLeaderElection**
Initialize a leader election

CHAPTER

SEVEN

EVENTCODES

CHAPTER

EIGHT

UTILS

Helper methods to run the experiment

`utils.`**`generate_distortion`**(*type='linear'*, *magnitude=1*, *n=10*, *show=False*)
Generates a distortion model represented as a vector field

`utils.`**`generate_fish`**(*n*, *channel*, *interaction*, *lim_neighbors*, *neighbor_weights=None*, *fish_max_speeds=None*, *clock_freqs=None*, *verbose=False*, *names=None*)
Generate some fish

Arguments: n {int} – Number of fish to generate channel {Channel} – Channel instance interaction {Interaction} – Interaction instance lim_neighbors {list} – Tuple of min and max neighbors neighbor_weight {float|list} – List of neighbor weights fish_max_speeds {float|list} – List of max speeds clock_freqs {int|list} – List of clock speeds names {list} – List of names for your fish

`utils.`**`init_simulation`**(*clock_freq*, *single_time*, *offset_time*, *num_trials*, *final_buffer*, *run_time*, *num_fish*, *size_dist*, *center*, *spread*, *fish_pos*, *lim_neighbors*, *neighbor_weights*, *fish_max_speeds*, *noise_magnitude*, *conn_thres*, *prob_type*, *dist_type*, *verbose*, *conn_drop=1.0*)
Initialize all the instances needed for a simulation

Arguments: clock_freq {int} – Clock frequency for each fish. single_time {float} – Number clock cycles per individual run. offset_time {float} – Initial clock offset time num_trials {int} – Number of trials per experiment. final_buffer {float} – Final clock buffer (because the clocks don't

sync perfectly).

run_time {float} – Total run time in seconds. num_fish {int} – Number of fish. size_dist {int} – Distortion field size. center {float} – Distortion field center. spread {float} – Initial fish position spread. fish_pos {np.array} – Initial fish position. lim_neighbors {list} – Min. and max. desired neighbors. If too few

neighbors start aggregation, if too many neighbors disperse!

neighbor_weights {float} – Distance-depending neighbor weight. fish_max_speeds {float} – Max fish speed. noise_magnitude {float} – Amount of white noise added to each move. conn_thres {float} – Distance at which the connection either cuts off

or starts dropping severely.

prob_type {str} – Probability type. Can be *binary*, *quadratic*, or *sigmoid*.

dist_type {str} – Position distortion type verbose {bool} – If *true* print a lot of stuff

Keyword Arguments:

conn_drop {number} – Defined the connection drop for the sigmoid (default: {1.0})

Returns:

tuple – Quintuple holding the *channel*, *environment*, '*fish*, *interaction*, and *observer*

`utils.`**`run_simulation`**(*fish*, *observer*, *run_time=10*, *dark=False*, *white_axis=False*, *no_legend=False*, *no_star=False*)

Start the simulation.

Arguments: fish {list} – List of fish instances observer {Observer} – Observer instance

Keyword Arguments: run_time {number} – Total run time in seconds (default: {10}) dark {bool} – If *True* plot a dark chart (default: {False}) white_axis {bool} – If *True* plot white axes (default: {False}) no_legend {bool} – If *True* do not plot a legend (default: {False}) no_star {bool} – If *True* do not plot a star (default: {False})

PYTHON MODULE INDEX