Drone Controller Documentation

Tim Arendes

July 2021

1 Example input

This is just a placeholder for an actual input later on. All coordinates are in AirSim format, meaning that one can just compy the coordinates needed directly out of AirSim. For everything, we have x,y and z coordinates, with z representing height, similar to AirSim. We have:

discretization_factor: Discretisation from AirSim to PRISM. If 100, then a width of 1000 in AirSim becomes a width of 10 in PRISM.

***_0:** AirSim location, where our box begins.

start_*: AirSim location, where the drone starts.

***_max:** AirSim location, where the box stops. Together with *_0, this completely defines our box of operation.

goal_*: AirSim location of the goal.

2 Grid graph

To store the information about the current environment. Saves information about the box of operation and start/goal position. This is saved as AirSim coordinates, as well as PRISM coordinates, so that, whatever is needed, both can be used. Also saves a map that saves all objects in the box of operation. Each objects gets a unique id. We have:

objects: Dictionary from id(int) to object. Used to save all objects

alpha: To save the factor between AirSim coordinates and PRISM. Basically the same as discretisation in the example input

AirSim_*_0: AirSim location, where our box begins.

AirSim_start_*: AirSim location, where the drone starts.

AirSim_max_*: AirSim location, where the box stops. Together with *_0, this completely defines our box of operation.

AirSim_goal_*: AirSim location of the goal.

PRISM_start_x: Starting position for the PRISM model.

PRISM_width/depth/height: Number of points in all directions in the PRISM model.

PRISM_goal_*: Goal position in the PRISM model.

next_obj_id: The id given to the next added object. Increases after adding one object.

2.1 add_object

Adds another object to the grid graph. Returns id of added object.

min_*: Minimal value for a specific direction. Input given as AirSim coordinate.

max_*: Maximal value for a specific direction. Input given as AirSim coordinate.

2.2 get_object_count

returns the number of objects in the grid graph.

3 object

Saves information about an object. Is not directly generated by user, but by add_object method.

min_*: Minimal value for a specific direction. Input given as AirSim coordinate.

max_*: Maximal value for a specific direction. Input given as AirSim coordinate.

id: unique identifier with which object is stored in the grid graph

4 Graph_initializer

Used to initialize the Grid graph. Currently takes operating space information from hard coded Example input, later on could read a file/ ask server. Can either call AirSim to find the objects, or read a file for them.

use_file: Boolean, to indicate, if the objects should be read from file. If not, then AirSim gets called.

4.1 create_blocks_graph

Creates small part of blocks environment as a Grid graph. Either load objects from a file, or calls AirSim to find them. Returns the grid graph with objects.

4.2 read_object_file

Reads the objects from a file. Each line contains one object, with the coordinates in order of: "x_min, y_min, z_min, x_max, y_max, z_max", separated with ", ".

4.3 write_object_file

Saves the objects in the file "objects.txt". Saves each object by itself. Each line contains one object, with the coordinates in order of: "x_min, y_min, z_min, x_max, y_max, z_max", separated with ", ".

5 helper_functions

Smaller helper functions, that get used by other functions.

5.1 PRISM_to_AirSim_API

Converts the given input of PRISM coordinates as coordinates that the AirSim API wants. Returns a list with position 0 being x, position 1 being y and position 2 being z.

PRISM_*: Coordinate inside PRISM

GridGraph: The GridGraph from which the coordinates are computed from. Needed for alpha, startpositions etc.

5.2 get_path

Takes the output of calling PRISM and computes the coordinates of the path PRISM finds to the goal. Assumes, that such a path exists, and the goal is not unreachable. Returns a list of coordinates, where the coordinates are Lists of the form [x_position, y_position, z_position]. All coordinates are PRISM coordinates.

result: String returned from running PRISM on some model.

5.3 model_possible

checks, if the model that was checked was possible. Uses the output of prism and checks for a sign, that the goal can't be reached(False), otherwise, it finds that the goal can be reached(True).

output: String returned from running PRISM on some model.

5.4 model_creator

Contains the functions that create the models that are needed. Saves them in /Files directory.

5.5 create_pm

Creates the basic model that tries to go from the start defined in the GridGraph to the end. Model is named "model.pm"

GridGraph: The graph describing the environment.

5.6 create_check_pm

Creates a model, that can be used to check, if the given path is still possible. Drone is limited to the path and starts at the beginning of it (first coordinate). Model is named "check_model.pm"

path: The path the drone is supposed to fly in PRISM coordinates.

GridGraph: The graph describing the environment.

5.7 create_new_path_pm

Creates a model to find a new path to the goal. Drone is not limited to the path, but begins at the start of the path(first coordinate). Model is named "model.pm"

path: The path which is used for the starting position(first coordinate)

GridGraph: The graph describing the environment.

6 object

Class to describe one object in the Grid graph. Coordinates are given in PRISM values.

id: Unique id of an object.

***_min:** minimal value in one direction in PRISM coordinates.

***_max:** maximal value in one direction in PRISM coordinates.

7 PRISM_calls

Contains command to call PRISM. Currently only one command.

7.1 execute_PRISM_model

Executes PRISM on some model. The name of the model is given as argument. Returns the output of PRISM. Expects the model and properties to be in the Files directory.

model_name: Name of the model that is supposed to be executed.

8 main

Contains the main threads that are used to coordinate everything and the main that controls these threads. Contains mutexes for variables that get used by multiple threads. To avoid deadlocks, always lock in the same order as the mutexes are given (in the file or in this documentation, same order)

mtx_path: Mutex for the path the drone is supposed to fly.

mtx_step: Mutex for the step the drone is at in the path.

mtx_possible: Mutex for the boolean that states, if it is possible for the drone to fly to the goal with the current path.

mtx_graph: Mutex for the Grid graph describing the environment.

mtx_changes: Mutex for the boolean that states, if changes were made to the Grid graph.

8.1 path_checker

Until the drone is at the goal, checks if changes were made to the path. If changes were made, check if the current path is still possible. If it is not, try to find a new path. If no new path can be found, try this again until one can be found. Needs: mtx_path, mtx_step, mtx_possible, mtx_graph, mtx_changes

path: The path the drone is supposed to fly. Gets changed, if new path needs to be calculated

step: The current step the drone is at on the current path.

possible: Boolean stating, if it is possible for the drone to fly along the current path.

graph: The Grid graph describing the environment.

changes: Boolean stating, if changes were made to the Grid graph.

8.2 graph_updater

Used to update the graph. Later on will be connected to AirSim in some way to update accordingly, at the moment can either place a block that blocks the path(block = True), so that the drone needs to fly above the obstacle, or place one were the drone was, which does not change the behaviour of the drone. Needs: mtx_graph, mtx_changes.

graph: The Grid graph describing the environment.

changes: Boolean stating, if changes were made to the Grid graph.

8.3 AirSim_control

Used to send commands via the AirSim API to make the drone fly along the path. Runs, until the drone is at the goal. Send one coordinate after anothe with "client.moveToPositionAsync" with velocity 5. If Drone is blocked, waits until a new path is found. Needs: mtx_path, mtx_step, mtx_possible, mtx_graph

path: The path the drone is supposed to fly.

step: The current step the drone is at on the current path. Gets changed here.

possible: Boolean stating, if it is possible for the drone to fly along the current path.

graph: The Grid graph describing the environment.

8.4 main

Main function controlling the three threads. Creates the first model and its path, initialized the shared variables and then starts the threads. Shared Booleans and Integer are saved as List, so that we can change the value and keep the reference in the single threads(use step[0] instead of step for example).

9 object_finder

Contains functions, that are used for the automatic detection of objects in the given task.

merging_distance: The maximum distance at which two objects get merged, given in AirSim coordinate length.

box_safety_scaling: A scaling factor which makes the boxes of objects a bit bigger than the objects themselves, to make the Flight more save. Multiplies with the width etc.. Could be changed to a constant safety margin instead of a multiplicative.

9.1 get_objects

Calls AirSim to get every AirSim_object, and creates the objects accordingly. Only creates objects for known AirSim_objects, by comparing their names.

graph: The grid graph, not containing any objects at this time.

9.2 inbounds

Detects, if the given objects is inside the area in which the drone is flying. Used to not include objects, that are outside of this area, as they are not important.

graph: The grid graph, not containing any objects at this time.

obj: The objects, which gets checked, if it is inside the bounds.

9.3 merge_objects

Merges the two given objects into one single one. Takes the maximum/minimum of both objects, so that every part of the old objects is part of the new objects.

obj_*: The objects, which should be merged.

9.4 check_merging

Checks, if the two given objects should be merged. Currently compares the distance of their middle points, but can change other metrics.

obj_*: The objects, which maybe should be merged.

9.5 create_TemplateCube_Rounded_object

Takes an AirSim_objects name, and uses it to find its position and scale, and returns the objects that correspond to these parameters. Currently works for "TemplateCube_Rounded", other AirSim_object types can be added in a similar way.

client_*: The AirSim client, used to make the necessary calls.

graph_*: The graph, with no objects at the moment.

name_*: The name of the AirSim_object, for which the objects should be computed.

10 Example PRISM model basic

mdp

formula goal = (x_pos = 70 & y_pos = 42 & z_pos = 2);

const int max_x = 76;
const int max_y = 53;
const int max_z = 29;

module position

step:[1..2] init 1;

x_pos:[0..76] init 40;
y_pos:[0..53] init 51;
z_pos:[0..29] init 2;

$[change\_pos](step = 1)\&(x\_pos + 1 < max\_x)$
$-> (x\_pos' = x\_pos + 1)\&(step' = 2);$
$[change\_pos](step = 1)\&(y\_pos + 1 < max\_y)$
$-> (y\_pos' = y\_pos + 1)\&(step' = 2);$
$[change\_pos](step = 1)\&(z\_pos + 1 < max\_z)$
$-> (z\_pos' = z\_pos + 1)\&(step' = 2);$

$[change\_pos](step = 1)\&(x\_pos - 1 >= 0) -> (x\_pos' = x\_pos - 1)\&(step' = 2);$
$[change\_pos](step = 1)\&(y\_pos - 1 >= 0) -> (y\_pos' = y\_pos - 1)\&(step' = 2);$
$[change\_pos](step = 1)\&(z\_pos - 1 >= 0) -> (z\_pos' = z\_pos - 1)\&(step' = 2);$

$[check](step = 2) -> (step' = 1);$

endmodule

const int obj_0_min_x = 24;
const int obj_0_min_y = 5;
const int obj_0_min_z = -1;
const int obj_0_max_x = 38;
const int obj_0_max_y = 30;
const int obj_0_max_z = 11;

module obj_0
$[check]!((x\_pos >= obj\_0\_min\_x)\&(y\_pos >= obj\_0\_min\_y)\&(z\_pos >= obj\_0\_min\_z)\&(x\_pos <= obj\_0\_max\_x)\&(y\_pos <= obj\_0\_max\_y)\&(z\_pos <= obj\_0\_max\_z)) -> true;$
endmodule

const int obj_0_min_x = 24;
const int obj_0_min_y = 5;
const int obj_0_min_z = -1;
const int obj_0_max_x = 38;
const int obj_0_max_y = 50;
const int obj_0_max_z = 11;

module obj_0
$[check]!((x\_pos >= obj\_0\_min\_x)\&(y\_pos >= obj\_0\_min\_y)\&(z\_pos >= obj\_0\_min\_z)\&(x\_pos <= obj\_0\_max\_x)\&(y\_pos <= obj\_0\_max\_y)\ \&(z\_pos <= obj\_0\_max\_z)) -> true;$
endmodule

More objects could can be added in the same way.

11 Further additions

More things can be added to the PRISM model. Nothing currently in code, but examples can be seen in Documents/Example_models. In the following, the additions get explained.

11.1 battery

Adds a module, that subtracts one from a battery level for every move. One it reaches zero, position can't be changed, so that a deadlock is reached.

11.2 noop areas

Adds an additional noop action and a noop area. In this area, one has to make multiple noop actions until the next position change. Allows to make areas less desirable to cross. Subtract one battery state for each noop action. More noop areas can be added, noops in the module have to be given an index then.

11.3 other drone(simple)

Adds another drone, that flies along a given path. Only saves the current step of the drone. Checks, if extra drone and main drone collide.

11.4 other drone(too complex)

Similar to the previous part, adds another drone, that the main drone has to avoid. This time, also saves the position of the drone in the module. Could handle more things (i.e. probabilistic positions), but is too complex, so it can't be calculated in PRISM.