Simulation: External Control of Swarm of Copters

Code Documentation

Asema Hassan Johannes Gätjen

March 11, 2016

1 Introduction

This documentation is solely to explain the code of the project. We present a short overview of important methods and dependencies between Lua scripts.

Firstly, the scripts in V-Rep are written in Lua. V-Rep is quite well documented and a good first step is to look at sample projects and the documentation at the following link: http://www.coppeliarobotics.com/helpFiles/en/writingCode.htm

1.1 Scenes

A basic building block of this framework are scene files with the extension .ttt. They can contain all models and scripts in a single binary file. To create a simulation a new scene is created in V-Rep. By default when you open V-Rep there is an empty scene with some basic models and features.

To load a scene from another project you need to have it your local directory. Then Go to Menu → File → Open Scene → Browse.

1.2 External Scripts:

V-Rep scene files are not suitable for source control systems such as git. Consequently, most of the scripts are moved to external files. An external script is one with a .lua extension and is imported in a V-Rep script with the `require "scriptname"` statement.

1.3 External Scripts path:

As external scripts are not a part of the V-Rep framework you need to make sure that the scripts that are to be imported can be found by Lua/V-Rep. Under Windows, there are basically three options to do this:

1. Manually copy/move files into the Lua folder in the V-Rep installation folder.

2. Create or edit the LUA_PATH environment variable and add the path to your script files in your repository.

3. Create a hard link with administrator command line such as: mklink /H

All three options work but the best one is Option 2 for Windows. For MAC Option 1 can work well copy and paste data into the Application → vrep → Right-Click → Open Contents → MacOS. Option 3 can also work well but it breaks when the branches are switched or the folder structure is changed. For details see the "hints_and_tricks.txt" file in the repository.

1.4 Open Internal Script:

To open the scripts in V-Rep go to Menu → Tools → Scripts.
OR Click Page/Script icon that appears next to associated object in scene hierarchy.

1.5 Change Texture:

To change texture of an object that is in the scene hierarchy.

- Double-click on the shape that appears with the object in scene hierarchy.
- In Object properties dialog go to → Adjust Texture → Load new texture
- Then if you want to change UV's can do that by → Adjust along U and along V values.

1.6 Vision Sensor:

How to create a vision sensor with visualization?

- Right-click in 3D View → Add → Floating View.
- Then select vision sensor → Right-click in floating window → View → Associate view with selected vision sensor.

1.7 User Interface:

The custom UI can be created from the icon of UI at the left most tool bar. OR it can be created from Menu → Tools →custom user interface edition.

2 Code Overview

The basic core functionality of FINken model was already done. We included the files from the previous group into our project directory "Simulation/vrep/". This folder has all the necessary scripts to control FINken flying including the PID controller. These scripts are external scripts and they need to be copied into the installation folder as described above.

The project code is divided into two parts one is in the finken_control/legacy folder which has previous scenes that were implemented including the local map gradient following behavior of single FINken, whereas the finken_control/ folder has the latest scenes and swarm behavior implementation. In order to make both parts work properly copy the scripts given with scene at the time of testing. See the info.txt file for more details.

2.1 Simple Gradient:

We started with simple gradient following behavior of FINken. Which can be found in finken_control/legacy folder with the scene titled "simpleGradient.ttt" and an external script "finken_simpleGradient.lua". In this file the following tasks are controlled. First of all importing another class as:

```
finkenCore = require('finkenCore')
```

A `customRun()` method is called in actuation method fron FINken internal script. Which will be called each time step. This function basically reads the color value from the vision sensor using **simReadVisionSensor(simGetObjectHandle('Floor_camera'))** method and passing the handle of the floor camera.

The target position is then set to the values computed from the color sensor using the method `simSetObjectPosition()`. It takes as parameter the handle of an object whose position needs to be changed, second parameter as -1 to set absolute position and third parameter as new position.

2.2 Position Sensor

The position sensor can also be found in finken_control/legacy folder positionSensor.ttt and it depends on external scripts "finken_simpleGradient.lua" and "imu.lua".

In the "imu.lua" script in the step() method, the velocity and orientation of object is read from the simulation at each time step and normally distributed random values (white noise) are added to it. Also the signal values are set here are for velocity, orientation and angular rate.

2.3 Local Map Gradient

The local map implementation with the FINken can be found in finken_control/legacy folder as "LocalMapGradient.ttt". It has dependency on following scripts.

- imu.lua

- landscape.lua
- LocalMap.lua
- finken_localMapGradient.lua

The landscape.lua script is controlling the dynamic landscape parameters such as image path, file name, xScale, yScale and xSpeed, ySpeed. These parameters can all be set via UI or remote API.

The method `customInit()` gets the handle of the UI attached with the landscape object and calls `initializeUI()`. Also this method checks if the parameter values are coming from RemoteAPI or not. If they are not coming from RemoteAPI then the UI will be updated otherwise it will `CreateTexture()` for the dynamic landscape.

The method `CheckUIButton()` checks if the update button was clicked during the simulation. If it was clicked then the signal values will be updated with the new values from the UI.

The LocalMap.lua script is handling the creation for local map from the resolution specified in UI or from RemoteAPI.

The method `LocalMap.new(totalSize, fieldSize)` initializes the local map with the given data and creates a temporary image in the directory with the same resolution as the local map by the method `TempSaveImage()`.The method `mapToByteString()` is used to get the correct texture map data from the saved image.

For the local map visualization a vision sensor pointing towards the saved local map texture was needed. The method `CreateTextureLocalMapDataTable()` creates a texture from the data of the local map and sets it as the texture of an object handle called 'LocalMapVisual', which is placed on some random position away from arena with a vision sensor pointing towards it. And by creating a floating view of vision sensor it can be seen in UI.

The method `updateMap( xDistance, yDistance, value, average, alpha, replace)` is the main method which is storing the color sensor value in the local map data table at each time step. New values are always stored at the center of the table. When the FINken moves `shiftLocalMap(xOffset, yOffset)` updates and shifts the local map table accordingly. To calculate more accurate gradient estimates the relative position of a height value within a field is also stored in the local map. In principle any values can be stored in the local map.

The finken_localMapGradient.lua is the main controller script of the project which gives control to each script.

The method `customInit()` initializes all methods, such as `initializeUI()` of Finken local map if the remote API is not enabled. `CopterPositionSetToCenterOfMap()` sets the copter position to the center of the map at the start of the simulation. It creates a virtual Box around the Finken to represent the local map size using the method `CreateAVirtualBoxAroundFinken`. It sets all the signal values of the parameters such as gradient speed, explore speed etc. It also checks for the flying modes of the copter and changes parameter values according to it.

In the `customRun()` method the color value from the vision sensor is read and is passed to the `LocalMap.updateMap()` method. Also the log data for the height is updated each time step.

The target position of the FINken is also updated in this script using methods like **updateTarget()** which sets the random target position if mode is random represented by 2 in code. Else it checks if FINken has reached the target position, if its true it will call `setNewTarget()`. Otherwise it will check the distance to the target position.

In the `setNewTarget()` method, the neighboring data is checked if that is filled then the gradient will be computed. And `setTargetToGradient(xGrad, yGrad)` will be called if not then the target will be set to explore to fill the local map with `setTarget-ToExplore(neighborArr)`.

At the end of the simulation `customClean()` is called which will call `SaveLogData-ToFile()` to write all the log data. The log data saves the height value at each time step. It also writes all parameter values into the file as a header once.

2.4 Swarm Gradient:

For the swarm behavior, the main scene is "SwarmGradient.ttt" and it has dependency on the external scripts:

- imu.lua
- landscape.lua
- LocalMap.lua
- finkenSwarmGradient.lua

The functionality is mostly the same. Only a few things needed to be changed for the swarm behavior implementation. The signals needed to be set for each FINken using `simGetNameSuffix()`. Also the suffix value is passed while writing log data.

The data in the local map visualization is only displayed for the last FINken.The floor camera is also a child of the FINken so the script controlling position of camera is now moved to the FINken associated script.

Likewise, the landscape movement is now controlled by the average speed of all copters.