Documentation of Stage 1 of the Trajectory Detector

Stage Description:

This stage involves the converting of the COutlierDetector library and its functionality to be able to be accessed from a python script.

Plan:

There are 2 ways at which I have deduced that this can be carried out.

1. Implementing the COutlierDetector library as a separate application with command-line arguments that can be used to carry out various tasks and can be called from a python script with ease.
2. Wrapping needed C++ functions in a python wrapper and can be called as regular functions in python.

There are advantages and disadvantages to both approaches, these are

Advantages of Option 1:

* It is simpler to implement
* It is less error-prone since there are no type issues between the 2 languages
* It is easier to maintain and scalable
* It is independent of python script and as such will run much faster

Disadvantages of Option 1:

* Could be messed up easily on the python end and result in runtime errors but can be very easily fixed

Advantages of Option 2:

* It is more streamlined into the python script and data transfer is seamless to an extent

Disadvantages of Option 2:

* It is more difficult to implement and more time consuming
* It is error-prone and difficult to debug at times
* It is not as fast as Option 1 since the code is still within python but will be faster than native python

I believe it would be best to implement Option 1 as it is easier to implement with fewer errors and it is very scalable with no need to adjust anything within the python code. Using this method will also provide the ability to change or improve upon the COutlierDetector library.

Implementation:

A program named trajectory\_detector was created. This program when called will expect certain command-line arguments that will dictate what operation it would be doing. This would be called using:

os.system(‘./trajectory\_detector “Arguments” ’)

Where “Arguments” would be the command-line arguments to be passed to the program and it expects no output just the number of trajectories to be printed on the screen and for a file with .eps extension to be created.

The library, trajectory.py contains the functions as described below:

* trajectory\_plot(infile, outfile) –

This function plots the trajectory of the specified infile and produces an output called outfile. The parameters infile and outfile are strings with specific file types, the extension for infile must be a string ending in .tra and that for outfile must be a string ending with .eps, failure to follow this procedure would result in a runtime-error or wrong output.

* trajectory\_convert\_csv(infile, outfile) –

This function is used to convert a .csv with the needed information to a .tra file that can be passed to trajectory\_plot. The parameters infile and outfile are strings with specific file types, the extension for infile must be a string ending in .csv and that for outfile must be a string ending with .tra, failure to follow this procedure would result in a runtime-error or wrong output.

* trajectory\_convert\_plot(infile, outfile) –

This function takes the raw data from a .csv file and converts it to a .eps with the trajectory plotted. The parameters infile and outfile are strings with specific file types, the extension for infile must be a string ending in .csv and that for outfile must be a string ending with .eps, failure to follow this procedure would result in a runtime-error or wrong output.

* trajectory\_help() –

This file outputs basic data about the core backend C++ program and the command-line parameters it takes.

* Parameter functions-

These include:

* set\_g\_fraction(param\_value)
* set\_g\_distance(param\_value)
* set\_g\_min\_outlier(param\_value)
* set\_mdl\_cost(param\_value)
* set\_min\_lineseg(param\_value)
* set\_max\_lineseg(param\_value)
* set\_default\_parameters()

The first 6 functions are used to modify the parameters for trajectory detection by changing the values in Param.h

The last function, set\_default\_parameters(), is used to revert the parameters to their default values which are:

const float g\_FRACTION\_PARAMETER = (float)0.95;

const float g\_DISTANCE\_PARAMETER = (float)82.0;

const float g\_MINIMUM\_OUTLYING\_PROPORTION = (float)0.50;

const int MDL\_COST\_ADVANTAGE = 20;

const float MIN\_LINESEGMENT\_LENGTH = 1.0;

const float MAX\_LINESEGMENT\_LENGTH = 10000.0;

Due to the need to modify constant parameters of the Param.h file a makefile check will be carried out in the core functions so as to update the build with the new parameters.

An example program is shown in application/, this is a basic application that utilizes the trajectory.py and trajectory\_detector files.

Note:

The files trajectory.py and trajectory\_detector must be in the same folder. Also check that all required dependencies are installed before running, if by chance one isn’t installed a prompt will indicate which it is and the proper installation procedure can be found below for the needed file.

Dependencies:

The COutlierDetector library and associated makefile require certain packages to be installed on the system. The needs are divided into compile-time dependencies and runtime dependencies. They are as follows:

Runtime Dependencies:

* gnuplot
  + this can be installed on Linux by running “sudo apt-get install gnuplot”

Compile-Time Dependencies:

* C++ Boost Library
  + This can be installed on Linux by running the command “sudo apt-get install libboost-all-dev”. Version 1.67 is required.
* g++ compiler
  + For Linux run “sudo apt-get install g++”
* GNU Make
  + For Linux run “sudo apt-get install make” then “sudo apt-get install build-essential”