**System Setup: Software and Data Preparation**

**Overview**

A necessity to either continue further work on this system, validate the results we obtained in experimentation and elaborate upon in this report, or produce one’s own results through system use is to setup the required components to run the system. Before being able to modify or run parts of the system, there are several things that must be setup beforehand. This includes:

* Obtaining relevant system resources.
* Downloading the requisite data sets and setting them up in the correct directories.
* Setting up Python and ‘pip’, along with the IDE and necessary packages.
* Running all the setup scripts that are run via the‘setup.cmd’ script.
* Setup of TensorFlow to use a GPU.

Once these steps are all completed, the editing of scripts, building of models, testing of files, and so on can be done by the user. In this part of the report, we shall be going through all the necessary steps to run the system on a different workstation; the hope is that, with the steps completed in this section, any user with the necessary computational resources can build models, reproduce results, and carry out additional experiments using the suit data captured that is used as part of the project.

**Necessary System Resources**

As this system works with large amounts of data and requires a heavy computational workload in order to build and test models, among other tasks, anyone running parts of this system requires a workstation setup with the necessary resources to execute many of the scripts, store the data, and so on. The vast majority of the work done on this project has been undertaken on a ‘Dell XPS 15 (9570)’ laptop with the following specs:

* *CPU*: ‘Intel Core i7-8750H’
* *RAM*: ‘16GB DDR4, 2666MHz’
* *GPU*: ‘Nvidia GeForce GTX 1050Ti with 4GB GDDR5’
* *Storage*: ‘512GB SSD’

A system with similar specs should be adequate to run the system; however, the following is ideal:

* *CPU*: At least a 7th gen Intel i5 or i7 (or AMD equivalent). A lot of the data preparation, computing of statistical values, reading from and writing to ‘.csv’s, and so on are done using the CPU (i.e. anything the system does that’s not including the training, testing, and assessing of models); hence, a good CPU will enable the user to run these scripts in reasonable time.
* *RAM*: Minimum of 10GB needed; some of the larger datasets we look at (e.g. when multiple raw measurements’ data are combined into one data set for a data shape of (~16000, 60, ~180)) require in excess of 8GB of memory just for the data, not including resources required for the IDE and other parts of the script being run. Any less than 10GB, therefore, may risk system instability or limit the user from carrying out certain parts of the experimentation outlined in this report. Additionally, DDR4 is recommended so as to increase the speed at which data is able to be written and read from memory.
* *GPU*: We make heavy use of TensorFlow running using the inbuilt GPU; the alternative would be to use the CPU, which by our estimation is approximately 10x slower to train models than using the GPU. Hence, to realistically build models in this system we need a good GPU. Ideally, the user would use an ‘Nvidia’ card as that is the easiest to setup with TensorFlow and, preferably, the card would have many CUDA cores (the 1050ti has 768) to enable faster parallel processing when training models.
* *Storage*: As of writing, the total storage required for all data sets contained within the ‘local directory’ (including ‘.mat’ files for NSAA assessments, 6-minute walks, natural movement behaviour, and the intermediate data extracted from all files via the data pipeline) amounts to over 170GB, while the system itself contained within the ‘project directory’ requires approximately 725MB of space; hence at least this much storage is required, ideally on an SSD to enable fast read speed from storage (which will happen a lot during the setup scripts). See the chapter on the ‘Project and Local Directories’ for more information about what these specific directories contain.

**Data Sets Setup**

With a workstation obtained with the requisite resources, the next step is obtaining the data sets required by the system. There are two approaches that can be taken: the first involves being able to access the link shown below (which should be possible for anyone with Imperial College London credentials), which contains all the data used as part of this project (i.e. that also contains all the intermediate data constructed via the scripts that make up the data pipeline). Hence, a user that downloads all the data from this link would not need to run the Python scripts contained within the ‘setup.cmd’ script, only the pip installation commands. The second approach should be taken if the user either doesn’t have Imperial College London credentials or otherwise can’t access the files, or if one wishes to run the pipeline ‘from scratch’ (i.e. computing ones own intermediate data files via the data pipeline); this does require the user to fully run the ‘setup.cmd’ script. Note that the below explanations assume the user already has access to the complete ‘project directory’ if one is reading this report; however if not, consult the chapter on ‘Project and Local Directories’ for guidance on how to access this.

The first approach is as follows:

1. Setup a base directory in the user’s storage directory; the default name for this that has been used thus far has been ‘C:\msc\_project\_files\’; however, a more intuitive name can be chosen by the user if desired. This becomes the user’s ‘local directory’ for the project.
2. Download each of the directories contained within the OneDrive link: <https://imperiallondon-my.sharepoint.com/:f:/g/personal/djh18_ic_ac_uk/Euymu00dXG1Cmmeoz3xxq24BekH57ZuDmU9uZtoSr60xfg?e=L5C62Z> and place each directory in the ‘msc\_project\_files’ directory (e.g. ‘left-out’, ‘allmatfiles’, etc.) within the user’s created ‘local directory’.
3. Modify the requisite line within ‘<project directory>\source\settings.py’ to point to this new location. For example, if the user has setup the ‘local directory’ as ‘example’ in ‘C:\’, then they should modify the ‘local\_dir’ variable (line 7) to now contain: ‘local\_dir=”C:\\example\\”. In doing this, it ensures that all other scripts that need to access the data files in ‘example’ are correctly pointed to it.
4. Once the steps outlined in the section below on Python, pip, PyCharm, and the necessary packages have been undertaken, open ‘<project directory>\source\batch\setup.cmd’ for editing in a text editor, comment out line 19 and onward (as these create the intermediate data from the pipeline which we now already have), save the file, and run it to setup the Python packages.

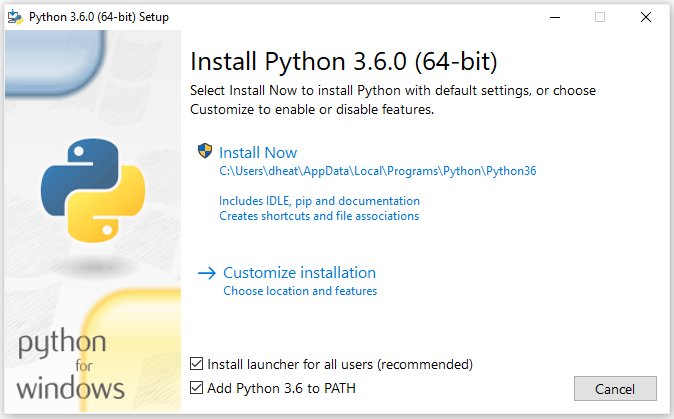
The second approach is as follows:

1. Setup a base directory in the user’s storage directory; the default name for this that has been used thus far has been ‘C:\msc\_project\_files\’; however, a more intuitive name can be chosen by the user if desired. This becomes the user’s ‘local directory’ for the project.
2. Obtain permission from the owners of the data sets used as part of the KineDMD research initiative to access and download the directories.
3. Create another directory within the ‘local directory’ called ‘output\_files’. This shall contain a number of things produced by the scripts and by the models, including the ‘.csv’ files of computed statistical values, the constructed models themselves, among other parts of the system.
4. The user should have links to the following data sets (though if they don’t the requisite permission for each must be obtained by the relevant parties): ‘NSAA’, ‘NMB’, ‘allmatfiles’, ‘6MW-matFiles’, and ‘6minutewalk-matfiles’. Each of these directories should then be downloaded and directly placed within ‘local\_dir’.
5. Once the steps outlined in the section below on Python, pip, PyCharm, and the necessary packages have been undertaken, run the ‘setup.cmd’ in full to obtain the necessary Python packages for the project and compute the intermediate data used as part of the project.

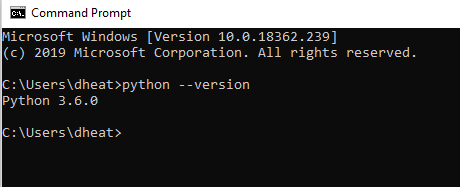
With these steps done, the data should now be in a form in which all the scripts that form the system should be able to access with the necessary directories constructed.

**Setup of Python, Pip, Necessary Packages, and PyCharm**

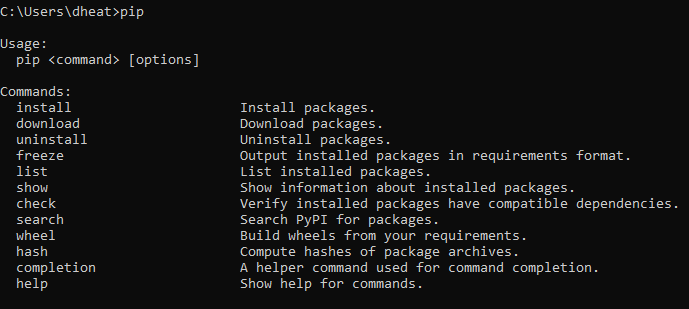
We now turn our attention to the setting up of the language, the ‘pip’ tool for package installation, and the libraries required to run the scripts. The first step is the installation of Python; the version this system runs on is ‘3.6.0’ and, while installing any subsequent versions should be acceptable, we shall install this version to avoid any possible complications to do with the language further down the line. Version ‘3.6.0’ can be downloaded from <https://www.python.org/downloads/release/python-360/> and by selecting the relevant installer from ‘Files’. With the installation window open, ensure that the ‘Add Python 3.6 to PATH’ box is checked; this shall ensure that the user is able to run Python commands from the command line or terminal:



With this installed, we can ensure that Python is setup correctly with the required version by entering ‘python --version’ at the command line:



It should be noted that the installation window for the user should also say that it is installing ‘pip’. This can be asserted to have been installed for the user by entering ‘pip’ at the command line, which should show output of something like this:



If, for whatever reason, ‘pip’ has not been installed, follow the steps outlined at <https://pip.pypa.io/en/stable/installing/>, which includes the modification of the PATH environment variable to ensure that we are able to use ‘pip’ correctly.

Once ‘pip’ has been installed and/or asserted to be setup ready to use, we are able to install the required packages. Navigate to the ‘<project directory>\source\batch\_files’ directory and execute the ‘setup.cmd’ script. This will setup the required packages to run the project. Not only that, but it will ensure that the versions of each of the packages installed for this system will also be setup by the user; this should minimize the chances that the user will encounter dependency issues between packages (different versions of ‘tensorflow’ have been shown to interact badly with certain versions of ‘numpy’, for example). It should be noted that ‘setup.cmd’ will also run all the Python scripts within the ‘source’ directory that are used to setup the files for other scripts such as ‘rnn.py’ and ‘model\_predictor.py’, so if the user doesn’t wish to run these at this time, the recommended way is to comment out each of these lines. This can be done by commenting out each of these setup lines (line 19 and onward) by inserting ‘REM ‘ at the beginning of each line. Note that this means that ‘setup.cmd’ must be run later when the user wishes to prepare the data for the system and if the user doesn’t already have the intermediate data.

**Installation of PyCharm (Integrated Development Environment)**

If the user is intending to do any long-term modifications to the system, or if they simply want a more convenient place to launch the scripts from, then it is recommended that they install an IDE for the project; specifically, PyCharm Community Edition. Using this provides several advantages to the user:

* An in-built terminal to run the scripts of the system with the necessary arguments.
* Easy interaction with Git to work from the project code in GitHub/GitLab.
* Syntax assistance when editing any files.
* Multiple tabs to help with editing multiple files simultaneously along with the script variable explorer.

To setup the IDE, the following steps should be undertaken:

1. Download the community edition of PyCharm, the link for which can be found at: <https://www.jetbrains.com/pycharm/download/#section=windows>.
2. In the ‘Installation Options’ window, it’s recommended that the user selects the ‘Add “Open Folder as Project”’ option (to allow opening the ‘project directory’ as a PyCharm project) and the ‘.py’ association (so Python files open in PyCharm as default).
3. Launch PyCharm and select ‘Do not import settings’.
4. From the ‘Customize PyCharm’ window, click ‘Skip Remaining and Set Defaults’.
5. Prior to continuing with PyCharm setup, we first must setup Git if the user doesn’t have it already. The following section applies to Windows, but the equivalent can easily be done for iOS or Linux.
   1. Download Git from <https://git-scm.com/download/win>.
   2. Run the installation, making note of where Git is installed, with the default settings.

With this now completed (or not, depending on whether Git is already installed), the user should launch PyCharm and in the ‘Welcome to PyCharm’ window, click ‘Configure’ and ‘Settings’, followed by navigating to ‘Version Control’ and Git. In the ‘Path to Git executable’, enter the location of the ‘git.exe’ program that was just installed (or previously installed). This can be found within the ‘Git\bin\’ directory within the location where Git was setup. Click ‘Apply’ and ‘OK’.

1. In the ‘Welcome to PyCharm’ window, select ‘Check out from Version Control’ and Git. This is because we will be directly installing the ‘project directory’ directly from ‘GitHub’. Note that it’s recommended doing this even if the source directory has already been downloaded and setup elsewhere, as doing it this way ensures that it is easily to modify and commit to git if any changes to the scripts are to be made. Within this new window, enter the URL: <https://github.com/dan-heaton/MSc_indiv_project> within the URL window and click ‘Clone’ to clone the repository. Alternatively, if one wishes to clone from GitLab instead, swap the URL above with: <https://gitlab.doc.ic.ac.uk/djh18/MSc_indiv_project>.
2. The final step is to associate PyCharm with the Python interpreter we have previously installed. To do so, with the project opened in PyCharm, navigate to ‘File’ and ‘Settings’. Under ‘Project: MSc\_indiv\_project’ and ‘Project Interpreter’, click the settings icon and ‘Add...’. Under ‘System Interpreter’ the Python executable should already be detected within ‘Python36\’ as ‘python.exe’. However, modify this path to the ‘python.exe’ file if has not done so already. Click ‘OK’, ‘Apply’, and ‘OK’. In the main PyCharm window, it may take a few seconds to configure this interpreter but, once done, the user will be able to edit and run the scripts of the system within PyCharm.

**Configuring of TensorFlow to Use the GPU**

The following section works on the assumption that the user is working on a workstation containing a GPU. This is more or less a necessity to build models using ‘rnn.py’: while a reasonable GPU with >700 CUDA cores builds a typical model in 5-15 minutes, building these using an equivalently-priced CPU would take 1-3 hours. While the necessary steps to undertake the complete setup of a GPU are somewhat arduous, there exists a useful guide to doing so at <https://www.tensorflow.org/install/gpu>, along with a CUDA installation guide at <https://docs.nvidia.com/cuda/> for multiple OS’s. This includes details on setting up the CUDA toolkit and the CuDNN (CUDA Deep neural Network library), which are required to run TensorFlow on the GPU. With this setup, TensorFlow with subsequently run as default in all scripts using the GPU to create models as opposed to using the CPU.