This document shows the steps to setup the complete environment for our project. In this project we use various tools for software and hardware implementation. Firstly, a list of all required tools is provided. Secondly, step-by-step procedures are provided to prepare those installed tools for our project in particular. We start with introducing virtual environments through Anaconda, the basic commands, and how to import an existing environment (to maintain compatibility of the project). Then, the Arduino IDE installation for different boards that were used as edge devices.

# Programs and tools:

The following software/tools are the prerequisites:

**Matlab:** briefly to run a short script for data conversion.

**Anaconda:** for managing Python packages and environments.

**Python 3.7.1** (a yaml file will be provided for the full list of required packages) – wait for the provided steps.

**Jupyter Notebook:** to run Python scripts.

**Arduino IDE:** to compile and run codes on micro-controllers – other steps will follow for boards installation.

**C compiler on Anaconda:** to compile the C codes when linking Python to C (Cython). It does not have to have a GUI. It will be used in the terminal only. If the C compiler is not already installed on Anaconda, use “conda install -c msys2 m2w64-gcc”

# Virtual Environments:

**Why?**

Python is great! A very powerful open-source programing language that keeps growing day-to-day. However, since there are many contributors to Python, sometimes this raises some backward-compatibility issues. Thus, your old project may have some trouble working after upgrading your python version and/or any included packages. Saying so, when working on a project, you may want to keep track of the exact versions of Python and its packages so that you can transfer your work to other machines and/or other people (to make sure you all are on the EXACT SAME PAGE!). A smooth way to do so, is to use a “*virtual environment*”.

**What?**

A virtual environment is a set of packages and libraries each with a specific version that you can decide (and handcraft) away from the basic installed version on your devices. It protects your *base* version running on your device from random changes that you may do when working on different projects. The other big advantage is that it ensures the compatibility of your projects on other devices as long as you use the same environment. The reason is, when creating an environment, you can choose the exact version of every package as needed for your project. You can create a virtual environment that is an upgrade or a downgrade (as will be seen) w.r.t. the existing packages on your device while keeping the original installation untouched. Therefore, we will be using virtual environments to complete the setup and make sure we are on the same page.

**How?**

There are different ways to install Python packages in isolated environments. The two most common ways are using: Conda & PIP (however Conda is more powerful). In this work I use Conda (except in one scenario when I had to use PIP). A comparison between both will be introduced later in this document along with the problem when I had to use PIP.

1. Download and Install Anaconda: <https://docs.anaconda.com/anaconda/install/index.html>
2. After installing Anaconda, to make it handy and easy to call, you may want to access it from the CMD/terminal (whether you use Windows or Linux):

- **In windows**, you need to add it to the path:  
Control Panel > System and Security > System > Advanced System Settings > Environment Variables > Edit “Path” > New, select the path where Anaconda is installed [now this will make the system see python], then add another one after you browse to the folder where the conda file exists so that it can see conda.

- **In Linux**, just type in the terminal: export PATH=$PATH:/home/user\_name/anaconda3/bin  
(another reason to love Linux!)

1. Now, get to know each other 😊   
   Try creating new environments with different Python & packages versions and exercise those commands to familiarize yourself with them. Always remember to activate the environment you want to change before making any changes! Otherwise, it may change the base (original) installation on your device instead of the virtual ones.  
     
   **Basic Conda Commands:**

* **Install/uninstall/update a package:**

**conda install/uninstall/update numpy**

or from inside the env use: pip install numpy==1.20.2 (or whatever version you need)

* **Working with Environments:** 
  + **Export and Import Environment:** 
    - **Export: conda env export > new\_env\_name.yaml**

creates a text file (.yaml) with all packages and dependencies along with their versions.

* + - **Import: conda env create -f new\_env\_name.yaml**
  + **Create new: conda create --name its\_name python=3.6**
  + **Create an environment, with python version, and install some additional packages:**

**conda create --name its\_name python=3.6 numpy pandas scipy**

* + **List all env’s:** **conda env list**
  + **Activate one:** conda **activate its\_name**
  + **Leave it: deactivate its\_name**
  + **Remove it: conda env remove --name its\_name**
  + **Display all packages of an env & their versions: pip list**

**Note: you can also check a lib version from python as follows:**

**import numpy; print(numpy.\_\_version\_\_)**

1. Now import the environment we need for our project, use the file: py371\_sk0201.yaml  
   **conda env create -f py371\_sk0201.yaml**

“This will create a new environment called: py371\_sk0201\_base. You can also change the environment name from inside the text file before importing it.”

1. Importing the environment may not work because there is a consistency problem with conda ☹ So instead, let’s do the steps manually (this also includes adding the new environment to the Jupyter kernels, so it does the next step as well):

conda create --name env\_test4 python=3.7.1 numpy=1.15.4 pandas=0.23.4 matplotlib=3.1.1 tqdm=4.28.1 scipy=1.1.0 joblib=0.16.0 scikit-learn==0.20.1

conda activate env\_test4

pip install m2cgen==0.9.0 datetime

conda install -n env\_test4 nb\_conda\_kernels

python -m ipykernel install --user --name env\_test4

1. How to show multiple kernels (i.e. different environments on Jupyter notebook) and choose between them?
   * Make sure you have this package “nb\_conda\_kernels” by listing all conda packages using “conda list”. If you don’t have it, install it.
   * Next: associate different Python versions to different Environments and add it to the ipython kernels, ex:
     + **conda create -n env\_1 python=3.6 ipykernel**
     + **conda create -n env\_2 python=2.7 ipykernel**
   * You can double check by opening Jupyter notebook and checking the kernel drop-down list. Make sure that the environment you have imported from py371\_sk0201.yaml shows in the kernel list. This is the one we will be using to run our code in Jupyter Notebooks
2. Some little note about Conda and PIP:

* Sometimes, Conda does not have access to some old versions of packages. Therefore, it is good to know how to use PIP install as an alternative way to install packages. I had to do so to install a certain version of Scikit-Learn along with a newer version of Python. Here is a brief comparison between Conda and PIP that I summarized from the following github repo.

**Conda & PIP:** <http://jakevdp.github.io/blog/2016/08/25/conda-myths-and-misconceptions/>

* Conda is a general-purpose package management system, designed to build and manage software of any type from any language (not only Python). Pip (**P**ip **I**nstalls **P**ackages) is a general-purpose manager for Python packages.
* Conda and pip serve different purposes, and only directly compete in a small subset of tasks: namely installing Python packages in isolated environments.
* pip installs *python* packages within *any* environment; conda installs *any* package within *conda* environments.
* Conda and pip+virtualenv can both be used to create isolated Python environments without modifying your system Python installation.
* If you have an existing system Python installation and you want to install packages in or on it, pip+virtualenv will allow you to install new Python packages and build environments on top of that existing distribution.
* If you want to install Python packages within an Isolated environment, pip+virtualenv and conda+conda-env are mostly interchangeable. However, Conda's uniform, cross-platform, full-stack management of multiple parallel Python environments with robust dependency management has proven to be an *incredible* time-saver.

# Arduino IDE and MCU setup:

1. Install Arduino IDE from:
2. The boards I used are: Arduino uno, ESP32 Wrover Module, and Adafruit PyBadge. Arduino uno is already installed by default, but you need to install more packages for the later two:

To install new boards, you MUST add a link (or comma separated links) in “File>Preferences>Additional Boards Manager URL”. Those links tell the IDE where to find the installation files and/or any supporting packages associated to that board.

For ESP32 installation, add this link:  
<https://dl.espressif.com/dl/package_esp32_index.json>

Tools>Board>Boards Manager. Then search for ESP32 in the Boards Manager wizard and install it. When it is connected, it usually appears on COM9. From Tools>Board, choose ESP32 Wrover Module. Run a LED flasher or something basic from the examples to validate the installation. Depending on the ESP32 board you have, you may need to click a loader button to run the code after it has been compiled and uploaded to the device. Other models require you to click a certain button before connecting it, then upload the code, and finally unplug and plug it so that it runs the uploaded code. [For the ESP32 I had, after uploading the code I needed to click the button “Boot” to get it to start running]

1. For Adafruit PyBadge installation, you need to add the link of their github repo in the Additional Boards Manager URL (comma-separated from other links that you added previously):  
   <https://adafruit.github.io/arduino-board-index/package_adafruit_index.json>

Then, follow the steps in this link: <https://learn.adafruit.com/adafruit-pybadge/setup> to complete the installation for : Adafruit pyBadge M4 Express (SAMD51).

Notes:   
A) When connecting Adafruit PyBadge to the PC, it first appears as COM8, then the PC reads it and recognizes the ports, then it changes it to COM7. Port numbers may differ but the main point it, do not rush into the IDE and select the first port that appears at the beginning. Instead, wait for few seconds and select the correct port associated with it.

The board name is: Adafruit pyBadge M4 Express (SAMD51)

And choose Tools>USB Stack: Arduino.

B) When writing new code to this board, DO NOT RESET it, turn it On then Off instead. Resetting a connected board confuses the IDE of which port it is connected to.  
C) There was a problem that makes the IDE confused between USB & Serial Ports for this board. To solve it, use those few lines at the top of any code that you write to the board:   
#if defined(ARDUINO\_SAMD\_ZERO) && defined(SERIAL\_PORT\_USBVIRTUAL)

// Required for Serial on Zero based boards

#define Serial SERIAL\_PORT\_USBVIRTUAL

#endif