# Python Software Project

|  |  |
| --- | --- |
| *Name:* | Lance Conry |
| *Student Number:* | 216176 |
| *Email Address:* | lconry@utas.edu.au |
| *Course Coordinator:* | Dr Anya Reading |

# Platform

This software uses Python 3.3.3, and was developed on, and tested under Windows 8.1.

# Purpose of the software

The intention is to allow the selection and processing of images in the context of a UAV collecting aerial imagery. The structure of the software is intended to allow a plugin type system, where steps in the processing chain are able to be simply configured via a configuration file.

The software consists of four components, namely Triggers, Actors, Agents and the Control. With the exception of the control, the other components are created as modules, with the potential for multiple classes of each type in the module. I will describe each component, and then detail how they all fit together.

# Components

## Triggers

A trigger is a class that causes a process to start. Two triggers have been created, Interval and All-Images-In-Directory.

**Interval**. This trigger allows for the configuration to specify a delay, say 1 second. In this case the trigger will start a process every second. In the scenario of a UAV taking aerial images, this will allow the system to sample images from a high frequency image stream. As an example, consider an onboard camera which is capturing images at 5Hz. We may only want to process a sample at 1Hz, so the Interval trigger is useful.

**All-Images-In-Directory.** This trigger will start a process against every file in a given directory. Consider a directory where all files need to have an image processing algorithm applied. This trigger will allow this to happen.

## Actors

An actor is a class that performs an action. Actors are started by a *Trigger* triggering an action. Three actors have been created, Copy-Latest, Run-Agent-Then-Delete.

**Copy-Latest.** This actor will be initialized with a source folder. It will look for the most recent file in the directory, and copy it to a target folder, also specified in the constructor. Looking back at the scenario where the interval trigger was used, this actor may be used to copy the sample photo from the image stream to a new folder.

**Run-Agent-Then-Delete.** This actor will run an agent (which will be discussed later), then delete the source file that it uses. Used in conjunction with the All-Images-In-Directory trigger, it is used to run an image processing agent against every file in the folder. After each image has been processed, the source image is deleted so that it isn’t processed again.

## Agents

Agents are specific pieces of software that an actor could make use of. Only one agent has been created here, the Sobel agent.

**Sobel.** This is an edge detection image processing agent. Used in conjunction with the Run-Agent-Then-Delete actor, it will apply a Sobel filter to the source image.

## Config

The configuration file, “Config.py” is the definition that puts the triggers, actors and agents to a meaningful use. Looking at it, note the processList list. You will see it has a trigger and actor for both items.

**CaptureImage**

The captureImage item, will perform the scenario outlined above where sample images are taken from the camera stream, and added to a sample folder. Notice that the trigger specifies to use the Interval trigger, and also the rate at which it should fire.

Below the trigger, an actor has been defined. So every time the trigger triggers, the action will run. Note that the CopyLatest actor is defined with both a source and target path. So when the Interval trigger fires, the latest images will be copied from the source folder to the target folder.

**ProcessImage**

The processImage item specifies the AllImagesInDirectory trigger. Note that the source folder is the same as the output of the captureImage task. So it will take each sample file from the sample directory and apply the RunAgentThenDelete actor. We can see that we have specified the Sobel agent to run, and the folder in which to copy the output file.

**Misc**

The config file also specifies a server address, as well as a commsPort for each process. This will be explained later, but are used for socket based inter-process communication.

## Utilities.Communicator.py

Because we run our processes in separate machine processes, we need a way for the controller to keep track of what is happening in each of its child processes. The communicator implements a socket based inter-process communication system, which allows the children to pass messages back to the controller.

## Controller.py

The controller performs several functions. The primary being to run and coordinate the system.

### Chart

A chart is maintained showing the status of the different processes. Rather than display after the process has completed, it shows a live view of the number of sample images taken, as well as the number of files waiting to be processed. Because a system such as this running for a long time will cause the chart data to be overwhelming, the chart is limited to only displaying the most recent 100 seconds worth of data, providing an EKG monitor type effect.

The chart is run it its own thread so as not to block the controller processing.

### Communicator

The controller creates listening servers for each process, using the port as specified in the config file, which allows the child process to connect to and send data back to the controller. This data is used for reporting the state of each process, and plotting to the chart.

### Process spawning

The controller also is responsible for spawning a new system process for each process defined in the config file. Part of this is starting a new communicator listener for each process.

## ProcessRunner.py

When the controller creates a new process, it does so my using the processRunner. This will load the specific section of the config file to configure the process. It is responsible for dynamically importing and loading Actors, Agents and Triggers as defined in the config file.

## But why all the complication?

Consider that we want to be able to easily extend this system as new image processing requirements etc are identified. Rather than re-create a whole new program, all which is required is to create a new Agent, which exposes agreed upon functions and constructor, and modify the config file. As the library of Actors, Agents and Triggers grow, it will be easy for operators to define a custom workflow, while only having to edit a config file.

# Testing

## Controller.py

This file is run without any command line arguments, as all configuration is contained in the congfig.py file.

## CameraSimulator.py

In order to facilitate testing, this script has been included to simulate a camera taking photos. It will take a source image file, and copy it into the target folder at the sample rate specified. It is run from the command line as…

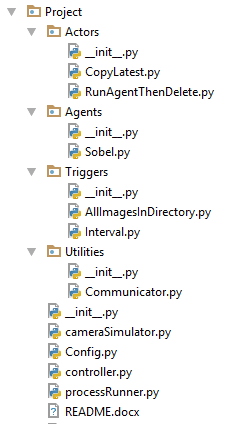
cameraSimulator.py <source image file> <target folder> <sample rate in milliseconds>

eg.

cameraSimulator.py c:\\temp\\test.jpg c:\\temp\\camera 1000

# Requirements

## Folder Structure



## Modules

|  |  |
| --- | --- |
| os | Built-in: Operating System |
| re | Built-in: Regular Expression |
| glob | Pattern matching for file filtering |
| shutil | Shell commands for copying, deleting, moving files |
| time | Built-in: Time |
| scipy | Scipy array support |
| skimage | Scikit-image, image processing |
| threading | Built-in: multi-threading support |
| timer | Interval based execution |
| multiprocessing.connection | Socket based networking support |
| atexit | Support for catching process exiting |
| subprocess | Support for launching child processes |
| matplotlib | Chart plotting |