MP2: Programmer's Guide to *PhotoPy*

de Guzman, Marie Danielle

Quirim, Coleen

Tablang, Karlo Mark

December 2018

Description

PhotoPy is an application that takes pictures using the device's camera and frames it in photobooth-like manner and is made using Pyglet. Upon running it in the terminal, it opens a window that shows a live stream of what the device's camera can capture. Once the spacebar is pressed, it counts down ten seconds before snapping the each shot out of four. Once four photos have been captured, it displays the framed result.

Modules, Functions, and Lines of Code

- **Main.py**

  This is the file to be run in terminal in order to try out PhotoPy.

  1. **engine.get_images()**

     Uses the `get_images()` function from engine.py to take the four pictures.

  2. **template = config['template']**

     template_config_path = 'resources/' + template + '/'

     path.append(template_config_path)

This line of code gets the template name from config.py and changes file path based on template name.

3. **template_config = import_module(template, package=None)**

   This line of code imports the configuration of the template. It imports the template (or the set frame of PhotoPy) as a module.

4. **frame = pyglet.image.load('resources/' + template + '/frame.png')**

   frame_sprite = pyglet.sprite.Sprite(frame)

   These lines of code sets `frame` as the set template and saves `frame` as a sprite.

5. **output_window = pyglet.window.Window(width=frame.width, height=frame.height, caption='Output')**

   @output_window.event

   This line of code opens the window of PhotoPy. The size of the window is dependent on the size of the frame.

6. **for i in range(4):**

   img.append(pyglet.image.load('tmp/photo_' + '{}'.format(i) + '.png'))

   This line of code loads the four photos one by one on the `tmp` folder created earlier and adds the images to the `img` list.

7. **for i in range(4):**

   sprite.append(pyglet.sprite.Sprite(img[int('{}'.format(i))]))

This line of code makes the four snapshotted pictures into sprites and adds it to the list of sprites.

8. **sprite[0].scale = (frame.width/img[0].width) * template_config.config['scale0']**

   sprite[1].scale = (frame.width/img[1].width) * template_config.config['scale1']

   sprite[2].scale = (frame.width/img[2].width) * template_config.config['scale2']

   sprite[3].scale = (frame.width/img[3].width) * template_config.config['scale3']

   These lines of code scale the four sprites.

9. **sprite[0].position = template_config.config['pos0']**

   sprite[1].position = template_config.config['pos1']

   sprite[2].position = template_config.config['pos2']

   sprite[3].position = template_config.config['pos3']

   These lines of code then position the four sprites.

10. **class OutputWindow(pyglet.window.Window):**

    The class `OutputWindow` is established.

    a. **def on_draw():**
       - **self.clear()**

         This clears data on the function on_draw()

       - **sprite[0].draw()**

         sprite[1].draw()

```
sprite[2].draw()

sprite[3].draw()
```

These lines of codes draws the different sprites saved earlier on list sprite.

- `frame_sprite.draw()`

  This line of code draws the frame.

- `try:`

  `mkdir('output')`

  This line of code makes a folder named output if it doesn't exist already.

- `now = datetime.datetime.now()`

  This line of code saves the current time and date to variable now

- `timestamp = (now.year, now.month, now.day, '__', now.hour, now.minute, now.second)`

  This line of code saves the details of the time it was used to a variable

- `for i in range(len(timestamp)):`

  `if len(timestamp[i]) < 2:`

  `timestamp[i] = '0' + timestamp[i]`

  This line of code gets the time information when it was used and adds zero before it only for the year and month.

- `pyglet.image.get_buffer_manager().get_color_buffer().save('output/output_{}.png'.format(''.join(timestamp)))`

  This line of code gets the details of the picture and saves it to the output folder with the timestamp as its name.

b. `def exit_callback(self):`

```
self.close()
```

These lines of code close the window of the final output.

11. **If __name__== "__main__":**

    a. **output_window = OutputWindow(width=frame.width, height=frame.height, caption='Output')**

       Creates a variable `output_window` that inherits the attributes and methods of `OutputWindow()`.

    b. **pyglet.clock.schedule_once(OutputWindow.exit_callback, int('{}'.format(config['timeout']))**

       Executes `OutputWindow.exit_callback` in the amount of seconds which is based on the `'timeout'` parameter in config.py

    c. **pyglet.app.run()**

       Starts processing the events.

- **Engine.py**

1. **def draw_text(frame, text, x, y, color=(255,255,255), thickness=5, size=4):**

```
if x is not None and y is not None:
    cv2.putText(frame, text, (int(x), int(y)),
    cv2.FONT_HERSHEY_DUPLEX, size, color, thickness)
```

These lines of code create a function to draw text over the camera overlay.

2. **def get_images()**

    a. **tmpdir = 'tmp'**

       try:

```
        mkdir(tmpdir)
    except FileExistsError:
        pass
```

This line of code makes a folder named `tmp` if it does not exist beforehand.

b. `cam = cv2.VideoCapture(config['source'])`

This line of code sets `cam` to the cv2 module function VideoCapture using data, which returns 0 from config.py. `VideoCapture(0)` which determines the video source.

c.
```
img_counter = 0
tick = 'resources/sounds/tick.wav'
shutter = 'resources/sounds/shutter.wav'
```

These lines of code sets the location of tick and shutter sounds.

d.
```
init_time = time.time()
test_timeout = init_time + config['timer']
final_timeout = init_time + config['timer']
counter_timeout_text = init_time + 1
counter_timeout = init_time + 1
counter = config['timer']
```

These lines of code initialize a timer based on the configuration.

e. `while img_counter < 4:`

This line of code ensures that four images are being shot.

- **`ret, frame = cam.read()`**

  This line of code captures what the camera "sees" frame-by-frame. It returns `True` if frame is read correctly. Otherwise, it returns `False`.

- **`x = int(frame.shape[0] * (3/4))`**

  `y = int(frame.shape[0] / 4)`

  `if (time.time() > counter_timeout_text and time.time() < test_timeout):`

  `draw_text(frame, str(counter), x, y)`

  `counter_timeout_text += 0.03333`

  `if (time.time() > counter_timeout and time.time() < test_timeout):`

  `counter-=1`

  `counter_timeout+=1`

  `playsound.playsound(tick)`

  These lines of code display the timer counting down and play a ticking sound every second.

- **`cv2.imshow("Camera", frame)`**

  This line of code displays the camera view in a window.

- **`k = cv2.waitKey(1)`**

  This line of code sets k to `cv2.waitKey(1)`, which infinitely waits for a key event. It sets k to the code of the pressed key.

- **`if k%256 == 27:`**

  `print("Escape hit, closing…")`

```
            break
```

These lines of code are executed when the escape key is pressed. It prints out `print("Escape hit, closing…")` on the terminal before breaking off from the loop.

```
    - elif k%256 == 32:

            img_name = "tmp/" +
            "photo_{}.png".format(img_counter)

            cv2.imwrite(img_name, frame)

            print("{} written!".format(img_name))

            img_counter +=1
```

These lines of code are executed once the spacebar is pressed. It gives the image a name depending on the image number and saves the file using the image name.

f. cam.release()

This function closes the device's camera.

g. cv2.destroyAllWindows()

This function closes the windows which OpenCV opened on the device's screen.

- **config.py**

1. config = {

```
        'source' : 0;

        'template' : 'floral'

        'timer' : 10
```

```
}
```

This code is loaded in both engine.py and main.py. In main.py, when asked for `source`, it returns a `0` and when asked for `template` (or the frame of the PhotoPy), it returns `floral`. Other default choices for templates are hearts and pastel. Additionally, when loaded in main.py, `timer` returns the value `10`.

Final Result:

References

- **python3**
  https://docs.python.org/3/
- **pyglet**
  https://bitbucket.org/pyglet/pyglet/wiki/Documentation

- **OpenCV**
  https://docs.opencv.org/
- **Photobooth templates were based from the following links:**
    - https://www.roaronentertainment.com/templates/
    - https://photoboothpixel.com/standard-design-4x6-templates/
- **Audio were obtained from the following links and modified:**
    - https://www.youtube.com/watch?v=iMI6yys_yKo
    - https://www.youtube.com/watch?v=ZJkamsEwato

All works obtained from the internet are used for academic purpose and not for commercial purposes. All credits go to the works' respective owners.