

## Alpha Particles Random Number Generator

Author: Leo Plese

Contact: [plese.leo@gmail.com](mailto:plese.leo@gmail.com)

Research mentor: prof. Zlatka Markucic

October 17, 2016

### **Abstract**

True randomness is the only way to avoid pattern and is very useful in the computer world. This document presents the process of getting true random number from low activity radioactive source as a source of entropy and camera with CMOS image sensor by using simple algorithms. The program is written in Python programming language and can be used as a service or standalone. The script can be easily customized and included as part of custom user application.

## Table of Contents

Abstract.....	1
Table of Contents.....	2
Introduction.....	4
Motivation.....	4
Randomness and Random Number Generators .....	5
Methodology .....	7
Conclusion. ....	10
Hardware Used in Alpha Random Number Generator Project.....	11
Input Source of Random Number Generator .....	11
Image Sensor – CMOS Image Sensor .....	15
Advantages of CMOS Sensor .....	16
Disadvantages of CMOS Sensor.....	16
Why Have I Chosen CMOS Image Sensor? .....	18
Technical Characteristics of Hardware Used in Alpha Random Project .....	18
Camera. ....	18
Radioactive Source. ....	19
Preparing Hardware .....	19
Additional Modifications Made due to High Image Noise.....	25
Tests Made for Noise Source Determination.....	25
Solving Dark Current Problem Caused by Temperature Increase.....	27
Camera Cooling Final Solution. ....	29
Alpha Random Generator Working Principle .....	34
Pros and Cons of Image Capturing Method Used in Alpha Random Project.....	35
Possible Solution for Disadvantages of Image Capturing Method with CMOS Sensor.....	37
Use of Highly Light-Sensitive CCD Image Sensor. ....	37
Avoiding Dark Current. ....	38
Alpha Random Camera – Calibration Process.....	38
From Picture to Number .....	41
Introduction to Program.....	41
Program Flow.....	41
Algorithms .....	43

"Split and Stack" .....	43
"From Array to 100-Bit Sequence" .....	47
"From 100-Bit Sequence to Float Number" .....	49
"From Float Number to Number 1 / 0 (True / False)" .....	50
"From Float Number to Number from min to max" .....	50
Critical Case .....	50
Inputs and Outputs .....	53
Inputs .....	53
Outputs .....	53
Functions in Program .....	54
main .....	54
fnException .....	54
fnImageCaptureAndTransform .....	54
fnImageArrayToNumber .....	55
fnRandNumGen .....	56
How to Use Program .....	57
Prerequisites for Program Execution .....	57
Program Execution .....	57
Technical Requirements .....	61
Recommended Hardware and OS Requirements .....	61
Software Requirements .....	61
Computer Systems and Software Used for Development and Test .....	62
Conclusions and Future Study .....	63
Acknowledgements .....	64
Glossary of Terms .....	65
References .....	78
Appendix A .....	80
Appendix B .....	84
Appendix C .....	86
Appendix D .....	87
Appendix E .....	89

## **"Alpha Random"**

### Alpha Particles Random Number Generator

## **Introduction**

### **Motivation**

What motivated me to research randomness and build my own random number generator was the next question: "How can a computer generate random numbers?".

Since I am interested in computer science, I know a computer is a machine which works on certain principle, that it is a machine and its output is completely limited by algorithms it uses for different calculations. Moreover, I asked myself: "Is there a different kind of random number generator which would be limited by nothing, which would generate perfectly random number?".

After a thorough research on possible sources of perfectly random numbers and thinking about them for a certain time, I decided to explore one of them more deeply – nuclear radiation. Although there were also some other choices I could have chosen for my random generator, I preferred this one to others. The main reason why I have chosen it is that I am very interested in physics and especially nuclear physics.

Finally, when I learned basics of randomness and random number generators and chose the physical phenomenon I will work with, I started building my own random generator and thinking about an algorithm I would implement into a program I could use for generating something random such as random numbers.

**Randomness and Random Number Generators**

Randomness can be defined as inability to predict further way of occurring of an event due to lack of pattern in its continuity over time. Randomness is a concept which has been present from time immemorial in numerous games of chance as for instance dice rolling. A dice is an excellent example of true random parameter which has random outputs - 1 of 6 possible numbers, so there is, according to probability theory and probability distribution, the probability of  $1/6$  equal for the event of rolling any number on a six-sided dice.

There are two basic types of randomness which are, at the same time, types of random number generators. Random number generator (RNG) is a device designed to generate sequences of random numbers for further use in a particular application. Nowadays randomness has many different applications and each of them has specific requirements which are the base for determining which of the two types of random number generators is to be used. These two types are pseudorandom (PRNG) and true random (TRNG) number generators.

The main difference between these two types is in nature of their input called seed. While true random number generators have true random input such as input from a physical phenomenon e.g. current time in milliseconds or measurement of atmospheric noise, pseudorandom generators like computers do not generate true random numbers but numbers which are relatively random i.e. numbers which can be predicted if the principle of the generator's work is known. This is what makes true random generators nondeterministic and pseudorandom generators deterministic. Deterministic generator is a generator output of which can be predicted exactly if the input and its work principle, for example the algorithm a program works on, are known. This is why pseudorandom generators, which are deterministic, have the prefix "pseudo" which in Greek means fake, meaning these generators generate fake rather than true random numbers. An example of deterministic generator is von

Neumann's generator which uses middle square method, which is not so complex and at the same time of poor quality, since the generator output data have short period (amount of time which passes between 2 successive repeats of data in generator output). Nondeterministic generator's output, however, cannot be predicted on any basis unless by method of guessing, the method which is not based on any algorithm and is perfect. Best example for this is realization of one-time pad (OTP), an encryption method which is the strongest one due to the following two reasons. First reason is a high degree of randomness which is being employed, and it actually implies the second reason which is frequency stability, a property of the one-time pad encryption technique which states there is equal probability that any of the numbers from the given range will be given as output i.e. that there is equal probability distribution for any possible output.

The second important property where the types of random number generators differ is their efficiency, amount of numbers a generator can generate in a reasonable amount of time acceptable for the application the generator is used for. Pseudorandom generators are generally more efficient than true random generators meaning they generate more numbers in a particular amount of time.

The last difference between the two types is periodicity. A true random number generator is aperiodic since its input is true random which means it does not have any pattern which would repeat over time. On the other hand, a pseudorandom number generator has certain pattern of random number output because it is deterministic, which is why sequence of numbers in its output repeats.

According to these three basic and most important differences between the types of random number generators, each of them has its specific purpose. True random generators are used where true random parameters are needed and where a big number of generated numbers is not necessary (they are less efficient than pseudorandom generators). These applications are

those in securing data (in cryptography), games of chance such as gambling games and lotteries, choosing random samples while conducting a research, and others. On the other hand, pseudorandom number generators are used where true random inputs are not inevitable, where a relatively big number of generated numbers is needed or, finally, where it is desirable that the same number sequences appear in generator output. Therefore, they are applied in numerous simulations (e.g. as input of Monte Carlo algorithm which requires sampling number sequences which repeat) and in electronic games (in random generation in computer graphics for creating various three-dimensional models, in slot machines, in video games etc.).

## **Methodology**

This project will guide you through the process of creating hardware for generating true random image from Alpha Random Camera, and will present software solution for generating random number from image generated on Alpha Random Camera.

Alpha Random Camera is based in entropy source, which is the naturally radioactive element thorium-232, contained in natural source thorite mineral. Working with radioactive material, as it is the case with thorite mineral, which is unsealed radioactive source, requires special working conditions such as use of appropriate shielding, area monitoring, personal monitoring (dosimetry), wearing special protective suits and respiratory mask, as well as decontamination procedure after the work is done. Due to lack of working conditions I did not have prerequisites to handle and work with radioactive material, so I decided to base this project on theory. Hardware I used in the project is tested under real working conditions. I used real CMOS camera and instead of putting radioactive source I sealed lens enclosure with black epoxy on inner, and red epoxy on outer side. I made all the tests simulating same conditions as if they were with a radioactive source and, therefore, I gotten real results and

issues except those flashes made by alpha particles. Main issue was the dark current of CMOS image sensor. This issue, as well as the way how I solved it, are described in the project.

Other issues and effects such as blooming effect were described by using documents from references as sources of information.

Data about radioactive source and picture used in the project were taken from Mineralogy Database webpage [webminerals.com](http://webminerals.com) and were used in agreement with authors and owners. To describe more specific case for this project, I used fabricated data such as mass of thorite mineral 1.19 grams, which was used to present calculations of source activity and measurements on radiation meter where activity of background 180 CPM is presented as 18,000 CPM on radioactive source surface.

The base for the project was a real image captured with CMOS camera in resolution 1280x720 (cropped and resized to 631x304 for webpage by author) and Am-241 source of alpha particles. Image was taken from [randombio.com](http://randombio.com) website with the permission of author. See Figure 1.



*Figure 1.* Image generated on CMOS camera with Am-241 radioactive source



To create algorithms for program, test the program and produce all analytical data, I made two simulation programs which are picture generator *AlphaRandom\_picGen.py* and number generator *AlphaRandom\_numGen.py*. You can see the main program *AlphaRandom.py* and simulator programs flow chart on Figure 2.

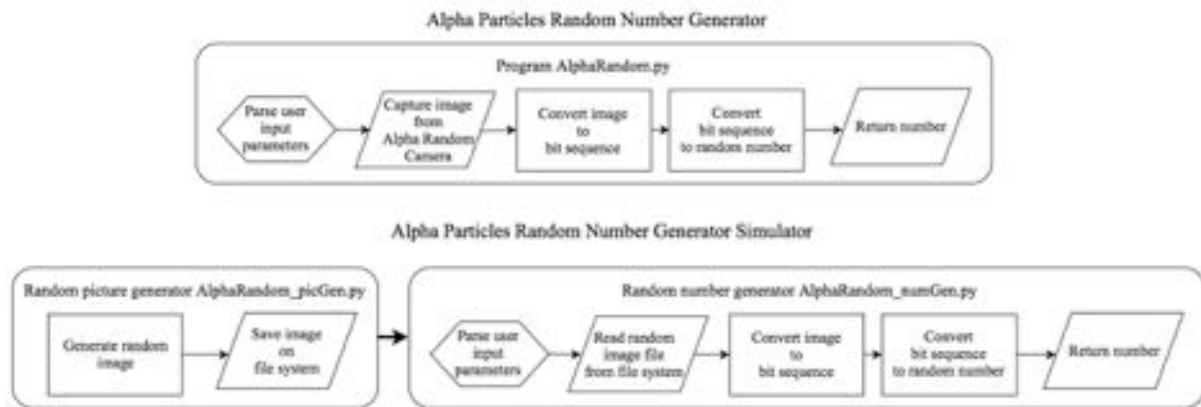


Figure 2. Flow chart of Alpha Particles Random Number Generator program and simulation programs used

Simulation program picture generator *AlphaRandom\_picGen.py* is used to generate pseudorandom pictures on computer without using Alpha Random Camera hardware. Using picture from Figure 1 as template, I created seven shapes of flashes in lower resolution so that they would look like they were produced by Philips camera which has resolution of 640x480. I used these shapes for generating flashes in picture generator program *AlphaRandom\_picGen.py*. The shapes data are stored in constants with names from *shapeOne* to *shapeSeven*. These shapes are shown in Figure 3. Program generates chosen number of pictures with 0 to 35 flashes with shapes which are randomly chosen from the seven aforementioned ones. This program is not described in the project, but in head section of program there are basic instructions on how to use it. The program can be found in Appendix D.



Figure 3. Flashes used for generating random generated picture for simulation

Simulation program number generator *AlphaRandom\_numGen.py* is used to generate random number based on exactly same algorithms in the same way as main program *AlphaRandom.py*, but for input it uses pictures from file system, and they are generated by *AlphaRandom\_picGen.py* rather than by Alpha Random Camera hardware. This program is not described in the project, but in head section of program there are basic instructions on how to use it. This program can be found in Appendix E.

All pictures that show radioactive source thorite mineral were generated and edited in photo editor software to visually demonstrate how it looks like in real world. Example pictures were generated by picture generator *AlphaRandom\_picGen.py* program and eventually edited. Pictures of image noise and dark current are generated on Alpha Random Camera without any source of light to show real conditions and dark current effect, but they were also edited in photo editor software where they were blended with pictures generated by picture generator *AlphaRandom\_picGen.py* program.

**Conclusion.** Results of this project, Alpha Random Camera device and main program *AlphaRandom.py* are designed to work in real world, not only in theory. To present this project, how I created hardware, what issues I gotten and how I solved them, as well as algorithms I created, I am going to present the project as practical work.

## Hardware Used in Alpha Random Number Generator Project

### Input Source of Random Number Generator

Ionizing radiation is generally one of possible sources for input of random generator. According to quantum mechanics, more precisely to quantum chaos (a branch of physics which describes chaos of dynamical systems), subatomic particles behave randomly up to a certain degree. For this generator I chose naturally radioactive source of alpha particles as source of true random input, more precisely natural thorium element, thorium-232 (Th-232) isotope (a source of alpha particles).

Thorium-232 as a naturally occurring isotope is the main part of thorite mineral (Figure 4) that I chose to use as the radioactive source for this project.



*Figure 4.* Thorite mineral

Thorium element was discovered in 1828 by the Swedish chemist Jöns Jakob Berzelius (1779-1848) and was isolated from a silicate mineral  $\text{ThSiO}_4$ . Thorium gotten its name by Thor, a Scandinavian god of war. Radioactivity of thorium was discovered independently by Madame Sklodowska Curie and C. G. Schmidt in 1898. Silicate material

ThSiO<sub>4</sub> found in a pegmatite on an island in the Langesund Fiord, southern Norway, and gotten its name thorite. Thorium has potential importance in the field of atomic energy because its isotope Th-232 can be converted by neutron bombardment to the uranium isotope U-233 that is fissionable.

Thorium-232 is the only primordial isotope of thorium which makes up almost all of natural thorium. Thorium-232 decays by alpha decay with energy of 4.083 MeV and half-life of approximately  $1.41 \times 10^{10}$  years. Its decay product is radium-228 and decay chain ends in lead-208, see Figure 5.

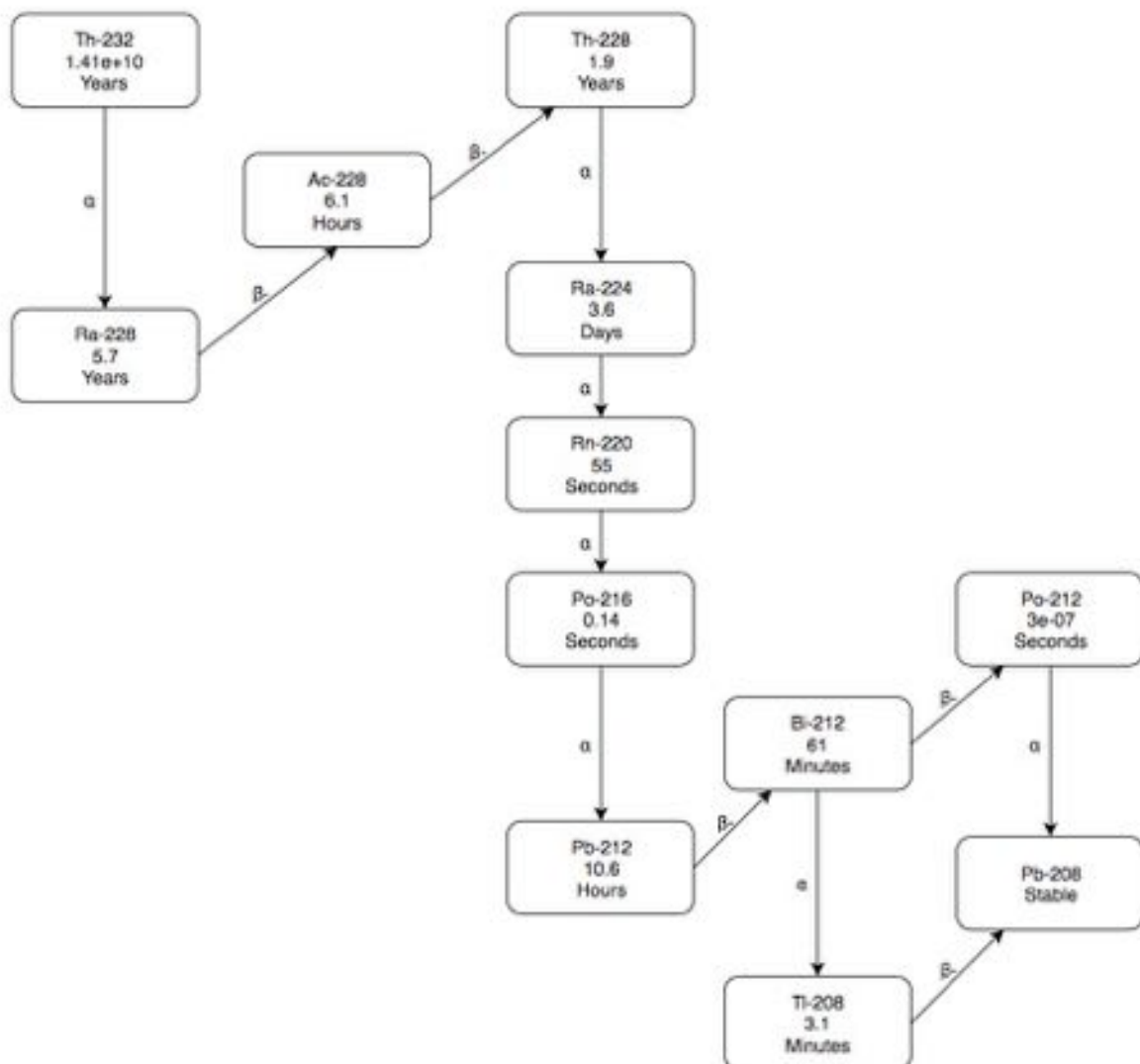


Figure 5. Thorium-232 decay chain

General information of used thorite mineral (taken from Mineralogy Database):

- Chemical formula  $\text{ThSiO}_4$

- Molecular Mass = 324.12 grams

Thorium	71.59 % Th	81.46 % $\text{ThO}_2$
---------	------------	------------------------

Silicon	8.67 % Si	18.54 % $\text{SiO}_2$
---------	-----------	------------------------

Oxygen	19.74 % O	
--------	-----------	--

-----	-----
-------	-------

100.00 %	100.00 % = total oxide
----------	------------------------

- Empirical Formula:  $\text{ThSiO}_4$
- Environment: Augite-syenite rocks
- Locality: Løvøya island, Norway (also known as Island of Lovo)
- Synonym: ICSD 1615, PDF 11-419
- Calculated properties of thorite: specimen size (mass/sphere diameter) 1 gram/7.09 mm
- Radioactivity: thorite is radioactive as defined in 49 CFR 173.403. Greater than 70 Bq / gram. Calculated activity in becquerels is 32,072 Bq per 1 gram.

Thorite mineral used in this project originates from Løvøya island, Norway. These are the properties of mineral:

- Shape and size: elongated rectangular shape of about  $9 \times 5 \times 6$  mm
- Mass: 1.19 grams
- Visual properties: yellow-brown-gray rock with very high surface relief
- Calculated decay activity in becquerels is 38,166 Bq which is gotten by calculation:  $1.19 \text{ g} \times 32,072 \text{ Bq/g}$  (reference value of calculated activity of thorite mineral taken from Mineralogy Database)

The main reason why I chose thorium-232 for this project is that it as an element itself and its decay products can be found everywhere, in dirt and food one consumes every day. Thorium as a natural element is safe in low amounts which are present in nature but can be dangerous if inhaled or eaten.

When handling thorite mineral, despite the fact that it is safe for handling, it is important to follow procedure for handling unsealed radioactive materials, as well as after handling thorite mineral is required decontamination procedure in order to remove radioactive material residue from equipment and working environment.

Since thorium-232 alpha particles have high decay energy (energy it releases during its radioactive decay) of approximately 4.083 MeV, which is its kinetic energy, there can occur a considerable amount of scintillation - a lot of photons are emitted. What is important is that thorium-232 alpha particles energy is sufficiently high to get clearly visible scintillation flashes on CMOS captured image, which can be clearly seen in Figure 6.



*Figure 6.* Part of image captured with Alpha Random Camera

To conclude, thorium-232 with characteristics mentioned above is a great add-on to be used with some of photodetectors. For this purpose I chose CMOS sensors from video camera to get more than a few light sensors (sensor points/pixels) to get more digital bits per acquisition time in the end.

**Image Sensor – CMOS Image Sensor**

In this project CMOS image sensor was used. CMOS (Complementary metal-oxide-semiconductor) is an image sensor which is APS (active-pixel sensor) because it has on-pixel amplifiers instead of having only few amplifiers amplifying signal at the end of the process. It appeared as an alternative to CCD (Charge-coupled device) image sensors which has some advantages over CMOS sensors, but at the same time also some disadvantages.

CMOS consists of three main layers. The two bottom layers consist of semiconductor such as silicon which consists of two parts – P-type and N-type semiconductor connected into P-N junction diode. By connecting such two oppositely charged types of semiconductors, electric current is allowed to pass in one direction which is known as forward bias, but not in the other which is called reverse bias. This phenomenon is the key principle of diode work. What enables this is forming of depletion region, which is region between the 2 semiconductor layers where P-type has mainly holes (lack of electrons – negative charge), while N-type has electrons surplus. Therefore, electrons from N-type move into P-type to fill the holes. In which direction electric current will flow and in which it will not, depends on whether anode or cathode are connected to P-type or N-type, which is called diode biasing. The third, top layer, is metal oxide layer e.g. silicon dioxide ( $\text{SiO}_2$ ) which is used as a dielectric (insulator which can be polarized by an electric field applied onto it).

The upper of two silicon layers is P-type which has pixels on its surface, each of which has its own capacitor and amplifier. When an image is to be taken, a shutter opens and photons hit the top silicon layer. Then photons cause photoelectric effect, which is effect of electrons emission caused by exposure to light (photons); electrons produced in this way are, therefore, called photoelectrons. At the moment electrons emission begins, the shutter closes to confine the electrons and to direct them along wires to parallel bus, which then sends signal

to A/D (analog to digital) converter which converts analog signal (electrons) to digital (binary digits 0 and 1) and, finally, an image is taken and it is stored in the camera's memory.

**Advantages of CMOS Sensor.** Firstly, CMOS requires much less power to produce an image because it does not have to move electrons along the wires row by row. In CCD, electrons are confined along conductor wires (made of e.g. aluminum) and then shifted row by row i.e. from one to the next aluminum wire all the way to serial processing bus which then sends the signal into capacitors that forward it to amplifiers to be amplified and converted into digital. Thus, CCD has much bigger power consumption. The advantage would be to save power in mobile random generator devices (e.g. portable random number generator for OTP) in the future. It is also more eco-friendly due to lower power consumption.

The second advantage of CMOS is faster readout which is also made possible by faster way of moving electrons which move all at once to the processing bus instead of moving row by row all the way to the processing bus. With faster readout more random images can be generated in a unit of time.

Finally, the price of CMOS sensor is also an advantage for low-cost random generators and therefore for mass production.

**Disadvantages of CMOS Sensor.** First of all, there is more noise caused by electric circuitry compared to CCD sensors. Since CMOS has one capacitor and amplifier per pixel, unlike CCD which has only few capacitors and amplifiers, there are more electronic components such as capacitors and diodes which can produce noise, so the total noise level is also higher. In image random generator it can give an additional pattern to an image which is not desirable. This noise is generated by dark current.

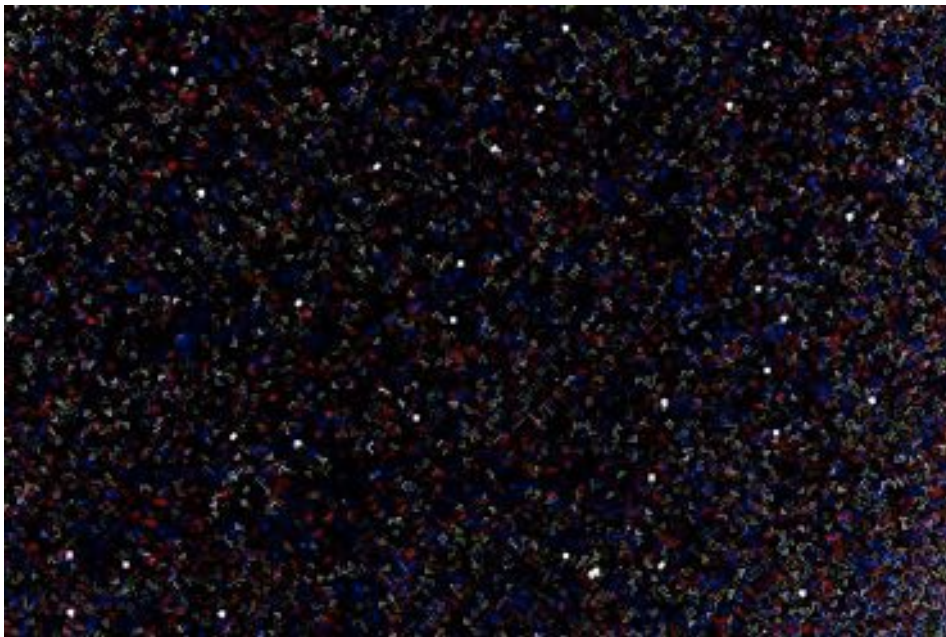
Dark current is electric current which appears in a photosensitive device such as sensor even when there is no light entering the device. In CMOS sensor it is mostly induced in amplifiers and diodes. Dark current is directly dependent on temperature - higher



temperature gives more dark current. The main consequence of dark current is presence of defective pixels, hot and stuck pixels which are consequence of sensor's failure to sense light levels correctly. Hot pixels, also known as sparkles, are those pixels with greater dark current compared to pixels surrounding it which is why they look much brighter than they should. Stuck pixel is a pixel which is "stuck" in a single color (red / green / blue) and always appears as such.

To decrease generation of dark current, industrial camera systems usually use some kind of cooling systems.

Generated flashes are shown as white circles (about 5 pixels in diameter) and noise is shown as small white pixels or colored pixels. In Figure 7, which is an example of an image captured by CMOS sensor, one can see denser noise on the right side of the picture.



*Figure 7.* Example of captured image with noise

Secondly, CMOS has lower light sensitivity compared to CCD. CMOS has a capacitor and amplifier per pixel so area for capturing photons is smaller. However, it is important to

mention there have appeared techniques for enlarging this light-capturing area, and they have overcome this problem by putting a lens in front of a diode with aim of directing light onto the photodiode, or by technique called back-side illumination where photodiode is put closer to light, while metal wiring conducting layer is put at the bottom.

The final disadvantage of CMOS sensor is the use of a shutter called rolling shutter. It captures image line by line by scanning across it in either horizontal or vertical direction. By using rolling shutter, rolling shutter effect or so called jello effect can appear which is the effect when parts of the picture become distorted. Regarding this, the pro of CCD would be a shutter that records a picture at single instant.

**Why Have I Chosen CMOS Image Sensor?** Price was the most important reason for using this image sensor. CMOS sensors are generally less expensive than CCD and, therefore, because of financial reasons I have decided to use a CMOS sensor.

### **Technical Characteristics of Hardware Used in Alpha Random Project**

**Camera.** Here are the properties of the web video camera (Figure 8) used in project.

Vendor: Philips

Model: SPC620NCVGA

Sensor: CMOS

Sensor resolution: VGA ( $640 \times 480$ )

Frame rate: 30 fps

Lens: F2.8, D50°

White balance: 2600 – 7600 k

Min. illuminance: < 5 lux

Color depth: 24 bit



*Figure 8.* Philips web camera model SPC620NCVGA

Note: There are no more relevant detailed information provided by the camera vendor.

**Radioactive Source.** Source of Th-232, thorite mineral used in Alpha Random project has activity of about 38,000 Bq (only alpha particles from Th-232 decay). The used sample of thorite mineral has a mass of 1.19 grams (Figure 9).



*Figure 9.* Thorite mineral weighs 1.19 grams

### Preparing Hardware

Because of thorite's very high surface relief and impurities within the rock, activity is not the same on all spots on source. In order to get the best efficiency of the radioactive source, I had to find the most active spot: the so called "sweet spot", and I was going to put it

so that it faces the CMOS image sensor. Therefore, what was needed was to measure all sides of the rock with Geiger-Müller probe and counter. For this I used radiation meter Eberline ASP-1 and Eberline HP-260 Geiger-Müller probe. The process of measuring activity is shown in Figure 10. At the sweet spot I found activity of alpha particles of 2,400 Bq. It is important to note that the measured alpha particles activity is not completely produced in Th-232 decay but also from Th-232 decay chain elements as Ra-224 and Rn-220.

Eberline HP-260 probe can detect alpha particles with energy more than 3 MeV, so alpha particles below 3 MeV are ignored. To measure beta and gamma only, I used alpha shield made of 0.2 mm thick HDPE (high-density polyethylene). Efficiency of the same probe for alpha particles is ~25%.

Here is the formula I used to get activity on the sweet spot:

$$\text{Activity Bq} = ((\text{CPM without alpha shield} - \text{CPM with alpha shield}) / 60) / 0.25$$

$$\text{Activity Bq} = ((54,000 \text{ CPM} - 18,000 \text{ CPM}) / 60) / 0.25$$

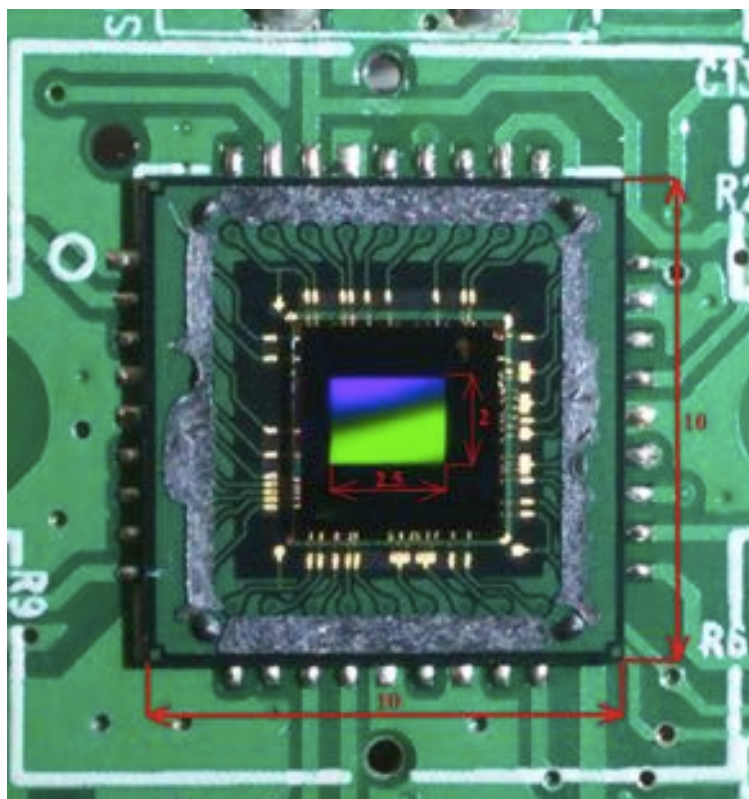
$$\text{Activity Bq} = 600 \text{ CPS} / 0.25 = 2,400 \text{ Bq}$$

Activity of the sweet spot on the thorite mineral is measured twice, firstly without alpha shield when I gotten 54,000 CPM (alpha/beta/gamma), and secondly with HDPE alpha shield when I gotten 18,000 CPM (beta/gamma). To get only detected alpha particles, I subtracted beta/gamma from alpha/beta/gamma in order to get only alpha. The aforementioned result of calculation is divided with 60 to get counts per second (CPS). The last step was division by 0.25 (probe efficiency of ~25% for alpha) to finally get the sweet spot activity of 2,400 Bq.



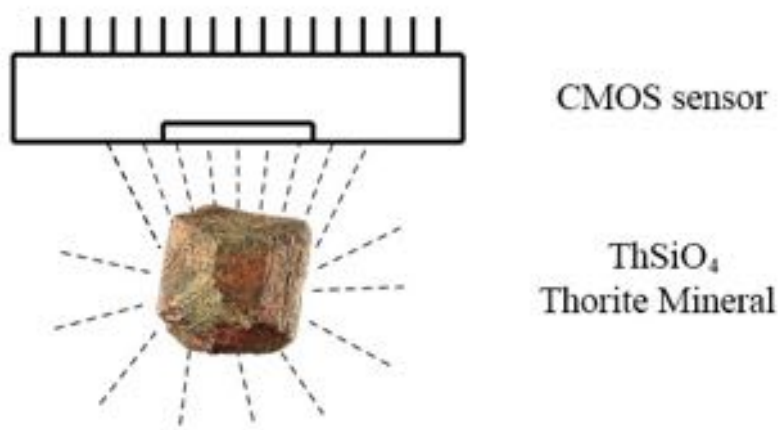
*Figure 10.* Finding most active sweet spot on thorite mineral with Eberline ASP-1 meter and Eberline HP-260 Geiger-Müller probe with HDPE alpha shield

Philips SPC610NC camera uses  $10 \times 10$  mm CMOS sensor with  $2 \times 2.5$  mm active area (Figure 11).



*Figure 11.* CMOS image sensor from Philips SPC610NC camera

Radioactive mineral was mounted 3 mm from CMOS sensor so as to get concentrated irradiated area of about 8 mm in diameter (Figure 12). Thus, center concentrated irradiation i.e. non-irradiated area on CMOS sensor corners was avoided.



*Figure 12.* Sketch of positioning and relation between radioactive thorite mineral and CMOS sensor with indicated direction of alpha particles

Camera lens was taken out from lens enclosure to make place for thorite mineral (Figure 13).



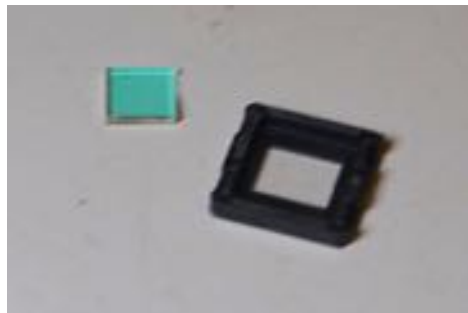
*Figure 13.* Web camera with unmounted lens from CMOS sensor

Thorite mineral was enclosed in light tight lens enclosure for mounting on CMOS sensor and all of that was put into web camera enclosure (Figure 14).



*Figure 14.* Thorite mineral sealed with epoxy glue in camera lens enclosure

A CMOS image sensor which was customized i.e. its cover glass (Figure 15) was removed, was exposed to alpha particles.



*Figure 15.* CMOS sensor cover glass filter

Also, relative position of CMOS sensor active area and size of radiation area of radioactive source had to be set and precalibrated in order to get better results. This setup can be seen on Figure 16.





*Figure 16.* Thorite mineral sealed in lens enclosure and mounted on CMOS sensor

Almost final appearance of camera which was used for Alpha Random project is shown on Figure 17. It was not the final camera appearance due to the problems described in the next chapter.



*Figure 17.* Alpha Random Camera version 1.0



**Additional Modifications Made due to High Image Noise**

**Tests Made for Noise Source Determination.** What is often indicated as possible and main noise source is increased sensor temperature.

Due to high dark current at higher temperatures, which subsequently caused additional increase in temperatures of camera electronics, a lot of noise was created.

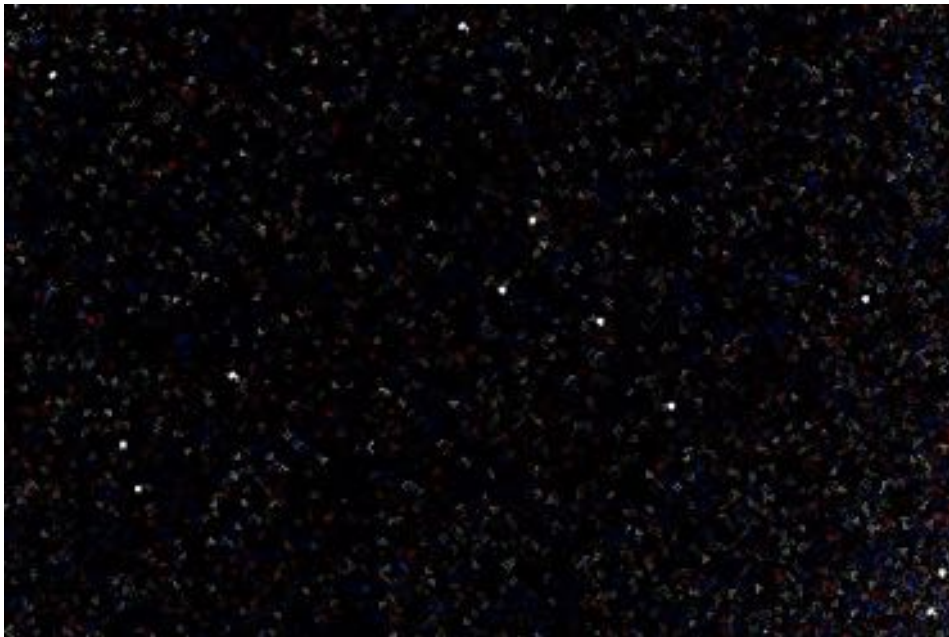
To determine that noise in captured image was caused by noise in heating CMOS sensor and electronics up, I made a temperature test.

I put the camera as well as two packages of silica gel into a waterproof container in order to avoid moisture condensation on the camera electronics. I put the container with the camera into the fridge with working temperature of 280 K for 30 minutes.

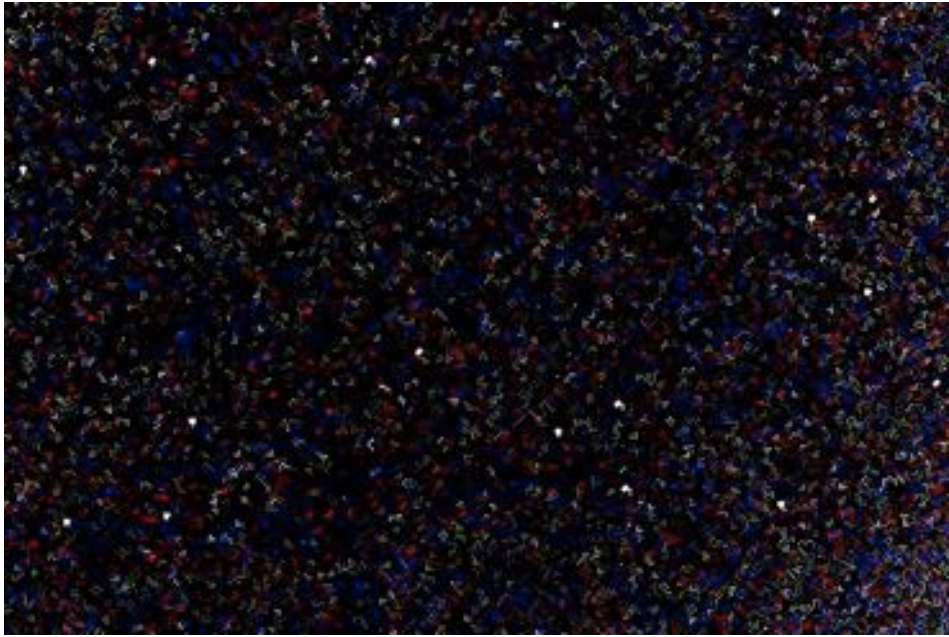
When I took the camera out of the container, I connected it to the computer. I was taking image captures and at the same time measuring temperature of electronics immediately after the CMOS sensor using laser thermometer. Air temperature in room where the tests were made was 297 K. At the moment immediately after capturing the first image, the temperature was 282 K (Figure 18). The next temperatures were measured after every 2 minutes, and these were respectively: 289 K (Figure 19), after 2 minutes 296 K (Figure 20) and after the next 2 minutes 299 K (Figure 21). It can be noticed the CMOS sensor temperature was even higher than the room temperature, and that is so because the sensor electronics heats up while working.



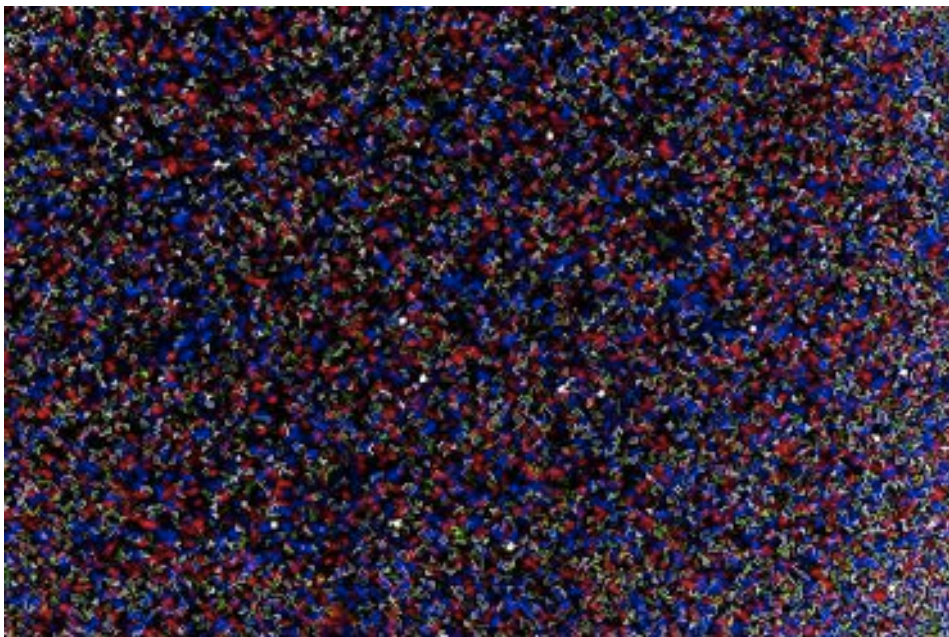
*Figure 18.* Image captured at 282 K



*Figure 19.* Image captured at 289 K



*Figure 20.* Image captured at 296 K



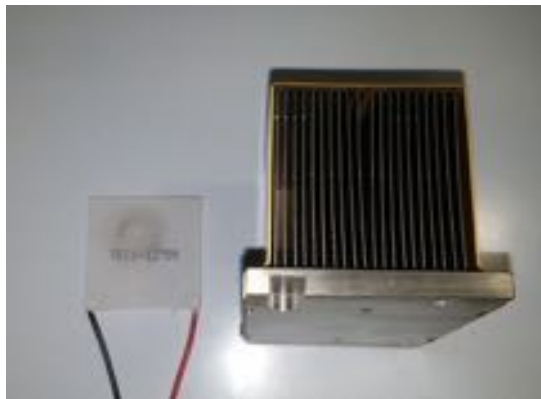
*Figure 21.* Image captured at 299 K

**Solving Dark Current Problem Caused by Temperature Increase.** The initial solution was setting active cooling beneath the CMOS sensor. For this I chose Peltier element, which would cool the CMOS sensor electronics by its work and would be simultaneously

cooled down by high-quality passive cooler which was initially used for cooling server's Intel Xeon central processing unit. Passive cooler (Figure 22 at right-hand side) was chosen so as to avoid causing additional electronic noise by additional ventilating fan.

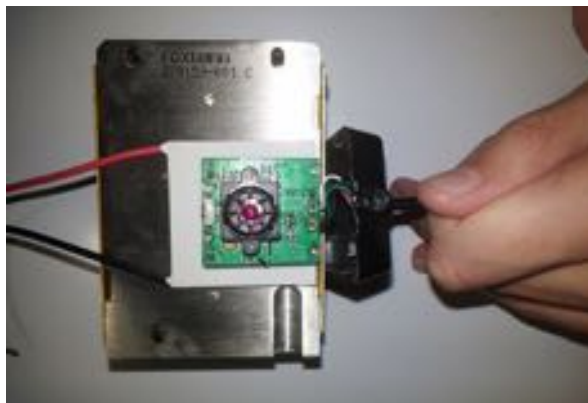
Peltier element (Figure 22 at left-hand side) is thermoelectric cooler or heat pump which transfers heat by conduction from one of its sides to the other one by using electric energy.

Peltier element is made of two different materials. Working principle is that it creates a temperature difference between hot and cold side.



*Figure 22. Peltier element (left) and CPU passive cooler (right)*

Figure 23 shows planned setup for camera with Peltier element and passive cooler.



*Figure 23. Initial plan for camera cooling system*

After all I decided not to use this method because of complicated process of fixing camera electronic to Peltier element. The case with the camera I used is that CMOS sensor is soldered on camera electronic board, and on the other side of the board there are SMD electronic components (Figure 24). If I had put any kind of thermal conductive material like paste or gel, I could have brought additional instability in form of capacitance or conductivity between electric charge carriers.



*Figure 24.* Back side of camera electronic board

**Camera Cooling Final Solution.** Due to the disadvantages of the above mentioned possible method by using Peltier element as cooler, I decided not to use this method, but I made my solution for camera cooler.

For this I used portable coolbox (travel refrigerator) which is shown in Figure 25.



*Figure 25.* Campingaz coolbox



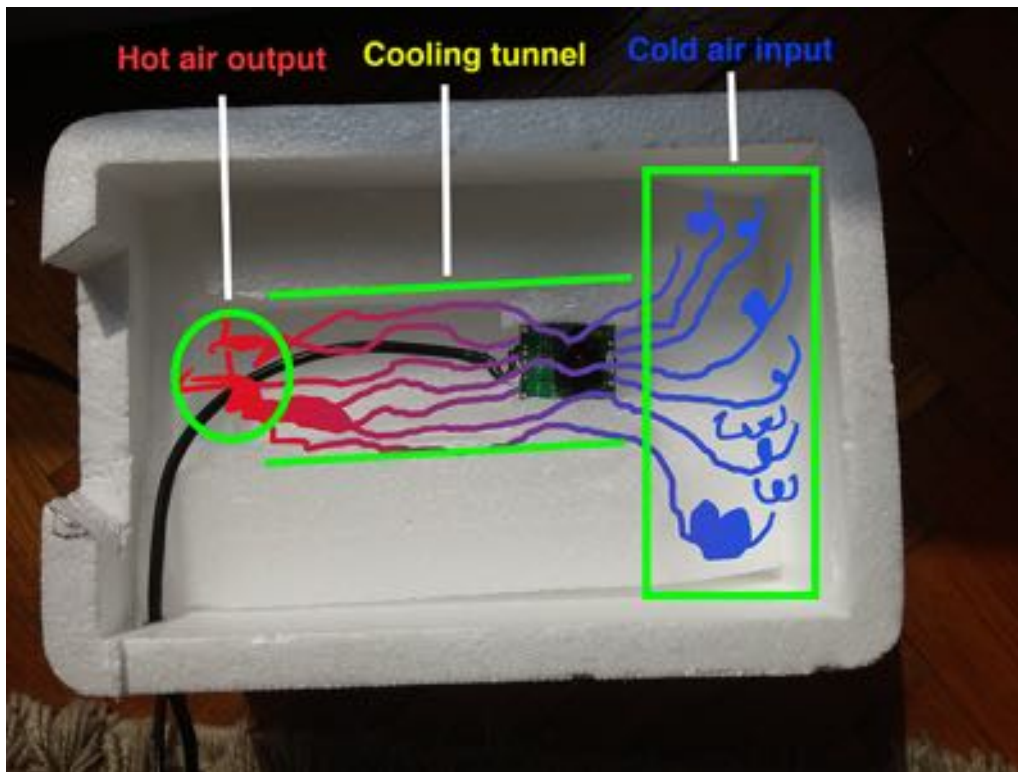
From coolbox I took the lid with active cooler made of Peltier element and coolers with ventilation fan. Ventilating fan in used coolbox is a low noise brushless DC motor which is mounted on outer hot side with double propeller on one axis for inner and outer side of Peltier cooler. For coolbox I used expanded polystyrene foam box to get smaller volume of air space (smaller than original box of the refrigerator) for cooling, and I added additional expanded polyethylene foam to create a narrow tunnel where I was to put my camera. I put camera into the narrow tunnel of the expanded polystyrene foam box (Figure 26).



*Figure 26. Camera fixed in air tunnel*

Another reason I chose expanded polystyrene foam is that it is an efficient thermal insulator so only very little heat from the outside comes into the box.

Camera cooling works on the principle that is shown in Figure 27.



*Figure 27. Sketch of cold air flow in Alpha Random Camera*

Camera is positioned in the narrow tunnel closer to cold air input of expanded polystyrene foam box (chamber) and vertically in the middle of tunnel to get air flow below and above the electronic board. In Figure 27 parts of chamber and air flow from cold air input through the cooling tunnel to the hot air output are shown. In this way, air flow and constant cooling of camera is assured. Polystyrene foam chamber is sealed for coolbox lid with self-adhesive tape to prevent leakage of cold air out of chamber on all sides, other than that side on which there are two depressions on coolbox lid (Figure 28), which are used for collecting water condensed on inner Peltier cooler. Note that coolbox must stand like in the figures – Figure 28, Figure 29 and Figure 30 (opposite than in ordinary use of coolbox) to prevent condensed water from getting in contact with camera.



*Figure 28.* Depression for collecting condensed water

Camera is prepared for use in random generator process. In further text this device with camera and cooling system is called "Alpha Random Camera". Final appearance of Alpha Random Camera, its back and front side can be seen in Figure 29 and Figure 30 respectively.



*Figure 29.* Back side of camera toolbox





*Figure 30.* Front side of camera coolbox

Cooling of camera is the key for getting picture free of undesirable colored pixels which appear due to dark current in sensor. In this way, temperature of camera image sensor falls down and, consequently, dark current is being decreased, producing thus less defective pixels.

### **Alpha Random Generator Working Principle**

Theoretically, in every decay of Th-232 a thorium-232 atom releases one alpha particle and then the atom is transmuted into Ra-228 and further by Th-232 decay chain until it becomes stable Pb-208.

Alpha particles hit the CMOS sensor. When an alpha particle enters the silicon layer, it deposits all of its energy in the layer by handing it over to surrounding atoms and thus ionizing them. Some electrons from the ionized atoms in the layer are released because they have gotten enough energy from the alpha particle, energy which is higher than the band gap energy (energy where no electron states can exist) so they become excited and move from a lower (valence band) to a higher (conduction band) energy band. The freed electrons begin recombination process where they combine with holes (positive charge carriers) in silicon layer and in this way produce electron-hole pairs. Finally, if there are enough frequent and enough dense recombination events made possible by sufficiently dense ionization, some of the ionized atoms will scintillate i.e. produce flashes which may be visible even to human eye. The light energy released in form of photons during scintillation is caused by loss of an electron's energy which happens during recombination process when the electron reoccupies the energy band of hole it is paired with i.e. it returns to the energy band where it was in the beginning by moving from conduction to valence band. These two bands lie on opposite sides of band gap where the so called Fermi level is located (a hypothetical energy level of electron, which in case of semiconductors lies in band gap which electrons and holes can populate).

After the scintillation, photoelectric effect can take place. Photons which were emitted during scintillation of ionized silicon atoms can release additional number of electrons from the atoms of silicon layer.

Finally, camera is now ready to capture an image.

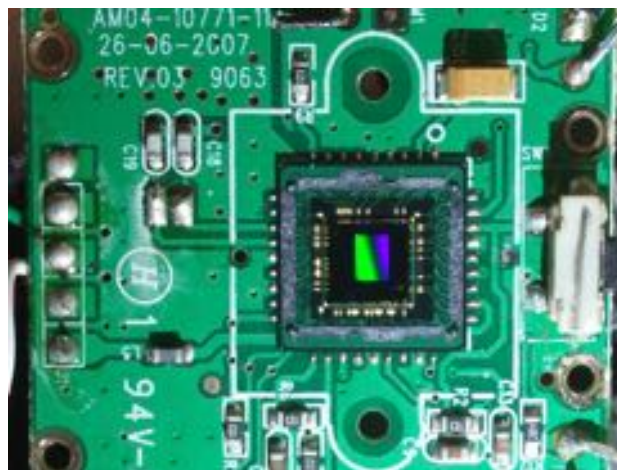
**Pros and Cons of Image Capturing Method Used in Alpha Random Project**

The greatest advantage is that it can be made for a very low price from an old web camera.

Moreover, due to low light sensitivity of used CMOS sensor there is no need to use special dark chamber to avoid additional undesirable light.

Due to use of low ionizing radiation activity and alpha particles which have short range and do not penetrate through thin layer of material or outer layer of human skin, there is no need to use special shielding materials to protect surroundings. Of course, radioactive material requires to be handled with special procedure and also to be labeled as a hazardous material.

The biggest disadvantage of this method for image capture of alpha particles with direct image sensor capture is that image sensor customization is inevitable. The customization implies removing sensor cover glass which is used for sensor protection. This has to be done to remove the main obstacle so as to enable alpha particles to hit the top layer of image sensor and cause scintillation which would produce visible flashes. In this process chances for damaging image sensor are very high. The CMOS from which sensor cover glass filter has been removed is shown in Figure 31.



*Figure 31.* CMOS sensor chip with removed cover glass filter

Another disadvantage is image sensor damages caused by ionizing radiation of high energy alpha particles. These pixels are called "dead" pixels and they are presented as black dots. This can be a big problem because it creates a pattern with more dead pixels over time. These pixels always appear as black dots at the same places on a captured image.

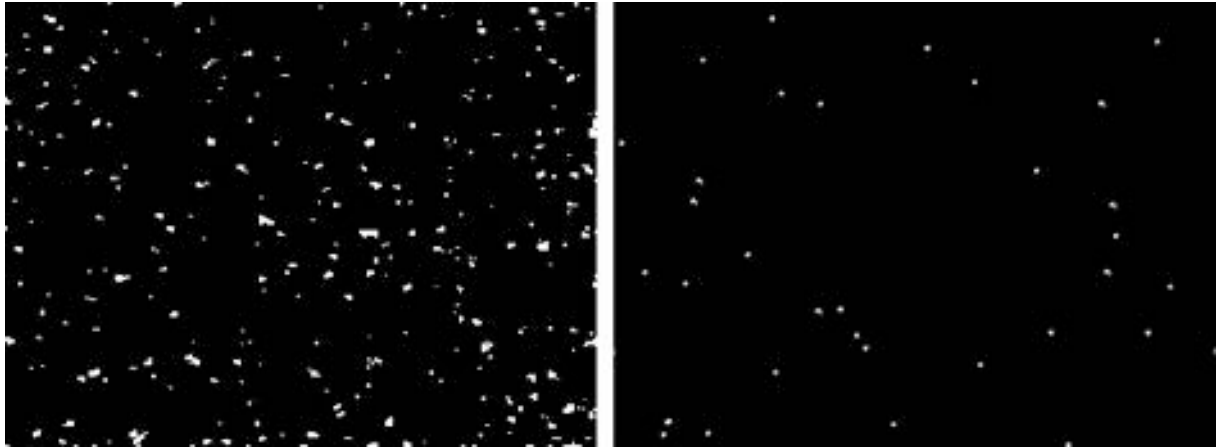
Another con is the blooming effect. Blooming effect is an effect used in computer graphics to make illusion of brighter picture by extending light from the borders of light areas of an image. In this case, it is not desirable to have this effect due to the fact that an alpha particle which hits image sensor produces a light of about 4-6 pixels in diameter for resolution 640x480 instead of only being shone on the hit pixel. The bright pixels which are produced in this way can be of different shapes such as the samples shown in Figure 32.



*Figure 32. Different shapes of captured bright pixels*

The principle of blooming effect is next: the higher the energy a particle has, it will brighten more pixels surrounding the hit pixel, that is turn them from black to white/bright. This happens when a pixel's potential well (a region in a field of force which has lower potential than points immediately outside of it) interacts with a radiation particle (such as alpha particle) which has more charge than it can be stored in the potential well or carried out of the sensor, and then this charge also starts filling potential wells of surrounding pixels.

Finally, there is a big chance to get high noise from dark current if CMOS sensor is not cooled. In later phase of the project, the dark current appeared as a problem where I gotten too much white pixels. The comparison between image with and without dark current noise is presented in Figure 33.



*Figure 33.* Left image with dark current noise, right image without noise

### **Possible Solution for Disadvantages of Image Capturing Method with CMOS Sensor**

**Use of Highly Light-Sensitive CCD Image Sensor.** Possible solution for the above explained disadvantages of the image capturing method with CMOS image sensor used in Alpha Random project is use of highly light-sensitive CCD image sensors with ZnS(Ag) scintillation screen. In this method highly light-sensitive CCD image sensor with protective cover glass on can be used, and alpha particle detector called ZnS(Ag) scintillator fixed onto the image sensor cover glass.

ZnS(Ag) scintillator is a silver activated zinc sulfide screen which gives a very strong scintillation effect when it is excited with alpha particles. Since ZnS(Ag) scintillator is in form of polycrystalline powder fixed on transparent foil, it cannot give very high resolution of captured image because of the size of crystals in scintillator. This appearance is similar to blooming effect mentioned as the disadvantage of the previous method.

Combination of this kind of CCD image sensor and ZnS(Ag) screen is very good because ZnS(Ag) scintillator emits light of 450 nm which is in visible light part of electromagnetic spectrum.

This method also protects image sensor from direct exposure to ionizing radiation and indirectly avoids sensor damage from ionizing radiation.

CCD image sensors are very expensive themselves and also in combination with ZnS(Ag) scintillation screen, which is the main reason why I have not used this method.

**Avoiding Dark Current.** Solution for avoiding dark current can be hot and stuck pixels subtraction method. In this method image is captured at the moment when shutter is closed or image sensor is not exposed to ionizing radiation. Next, another image is captured with image sensor exposed to alpha particles. To get result image, hot and stuck pixels have to be subtracted from image taken at no exposure to radioactive source ionizing radiation from image taken when sensor was exposed to the radiation. By using this method, hot and stuck pixels are not shown but, on the other hand, image quality and some image details may be lost.

In Alpha Random project, CMOS sensor was exposed directly to ionizing radiation and there was no possibility to isolate image sensor. I decided to set up CMOS sensor in the way described in the paper in order to get faster image capture for random generator. After initially used configuration without cooling CMOS sensor where I was getting high noise level on higher temperatures (hot summer days), I decided to make camera cooler.

### **Alpha Random Camera – Calibration Process**

To get better quality of random generated image, very important process is to calibrate the whole system to avoid processing of pixels which originate from dark current, and to emphasize pixels brightened by alpha particles.

Prerequisite for the calibration process is that computer and Alpha Random Camera are connected via USB cable, both of them are turned on, all required software is installed and cooling system of Alpha Random Camera is turned on for at least twenty to thirty minutes (depends on room temperature) to cool down the camera device.

Calibration process is made with Python script *AlphaRandom\_cal.py* from Appendix B. Result of script is image file *CalibImg.png* which is located in calibration script path. First step is to set constant *camera\_port* to value zero (0) if Alpha Random Camera is only camera on system or it is declared as the first camera on computer system, if it is second camera, set value one (1) and so on. If you are not sure which port is used by Alpha Random Camera, set value zero for start and run script and if there are no results, increase value in constant *camera\_port* for 1 and repeat the process until you get output result image *CalibImg.png* and no exception message from Python such as the one shown in Figure 34.

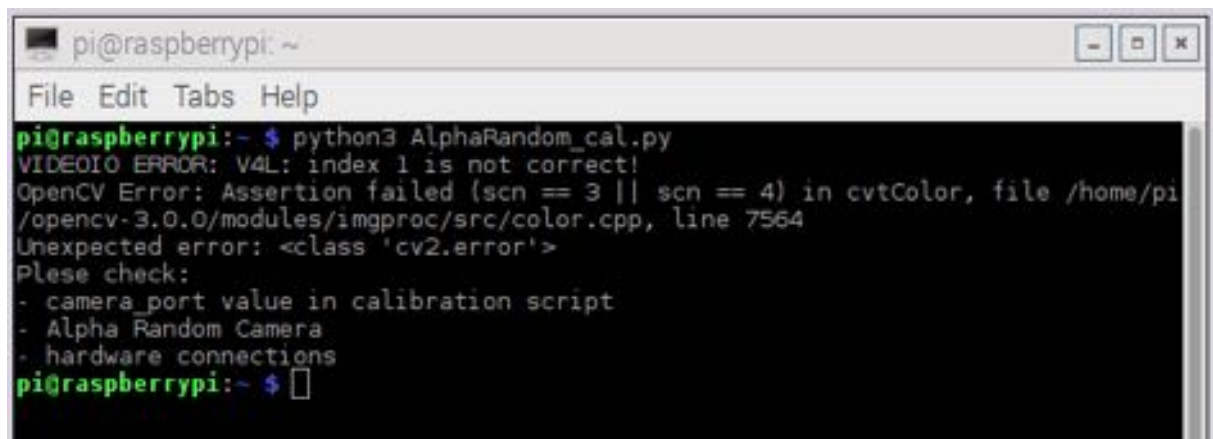


Figure 34. Example of error message from running of calibration script

When right camera port is set, you can start with contrast calibration. Contrast calibration can be set in the next part of program:

```
grayImage[grayImage<5] = 0  
grayImage[grayImage>=5] = 255
```

Value 5 is threshold value and needs to be changed to right value so as to get an image such as the image in the example shown on Figure 35.



*Figure 35.* Example of a random generated image

Image needs to be clear with bright pixels grouped in groups of about 15-20 pixels which are usually circular shaped like in example shown in Figure 35.

In case where image has more bright pixels, set threshold value to higher number, or in case there is no or just few bright pixels in more sampled images set value to lower until you get image as needed.

When you set the threshold to right value and get right image appearance, set the same threshold value and *camera\_port* constant value from calibration script in main Python program script *AlphaRandom.py* from Appendix A.



## From Picture to Number

### Introduction to Program

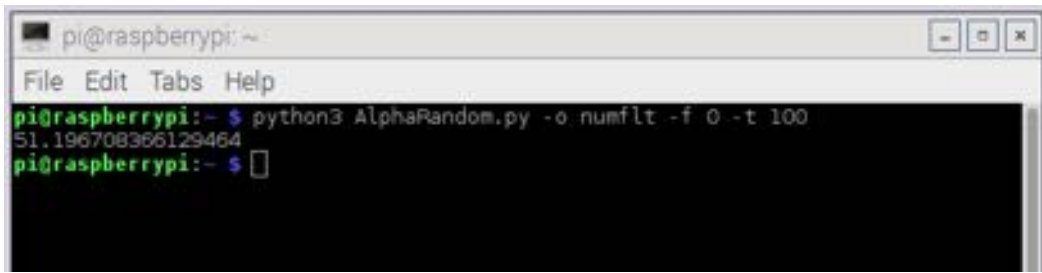
In this chapter program which is used for image capturing process will be described. Through previous part of this document you have gotten the picture of technical requirements for Alpha Particles Random Number Generator, as well as of problems and solutions to the problems related to hardware prerequisites. Program is written in Python programming language and it uses very simple structure in order to get a smaller footprint and a faster execution.

### Program Flow

When Alpha Random Camera system is working and ready to start processing, the computer gets picture from the CMOS image sensor from the web camera with the help of Python script in the following way.

- 1 Python script is executed from command line interface with passed arguments or from another caller script.
- 2 In case that program is called from command line interface, *main* function is called and input arguments are parsed. If program is called from another caller script, *main* function would not be executed and this step is skipped. In case of incorrect options or arguments, exception is raised and program stops with execution and exits.
- 3 Function *fnRandNumGen* is called with parameters *outType* (output type) and optionally *inMin* and *inMax* (minimum and maximum in range). In case of incorrect user input exception is raised and program stops with execution and exits.

- 4 Parameter values are checked and depending on output type *fnRandNumGen* function chooses process case and as the first step calls *fnImageArrayToNumber* function and waits for its result.
- 5 Function *fnImageArrayToNumber* calls *fnImageCaptureAndTransform* function and waits for its result.
- 6 Function *fnImageCaptureAndTransform* creates instance of OpenCV CV2 object called *ARcamera* which is used to capture an image of resolution  $640 \times 480$  pixels (307200 pixels). Image is written to variable *capturedImage*. Image from variable *capturedImage* is converted from BGR to grayscale image which is stored in variable *grayImage*. To get higher contrast picture, array variable *grayImage* is processed with numpy function in the next way: all numbers less than 5 become 0 and all values equal to or greater than 5 become 255. Note: value 5 is determined by calibration of system. As the last step, the function returns high contrast image stored in variable *grayImage* to caller function *fnImageArrayToNumber*.
- 7 Function *fnImageArrayToNumber* uses returned array and converts it with algorithms "Split and Stack", "From Array to 100-Bit Sequence" and "From 100-Bit Sequence to Float Number" to result as floating point number. The result is returned to caller function *fnRandNumGen*.
- 8 Function *fnRandNumGen* processes returned value, converts it to required output type and returns the result as the result of the program script or to the caller script. Figure 36 represents appearance of executing *AlphaRandom.py* program in CLI.



```

pi@raspberrypi:~$ python3 AlphaRandom.py -o numflt -f 0 -t 100
51.196708366129464
pi@raspberrypi:~$

```

Figure 36. Result of *AlphaRandom.py* program execution in command line interface

## Algorithms

**"Split and Stack".** In *fnImageArrayToNumber* function is used "Split and Stack" algorithm. This algorithm is used for transforming picture into 100 sections ( $10 \times 10$ ) which consist of 3072 pixels.

This algorithm is going to be explained on an analogous example of an image with  $16 \times 12$  pixels and 16 sections which has smaller number of pixels so to be easier to visualize the algorithm of separating an image data into a bit sequence.

Figure 37 represents initial state of data gotten from an image. Each column is assigned a letter (from A to P for 16 columns) and each row a number (from 1 to 12 for 12 rows). These data are stored in variable *inputImageArray* which gets processed grayscale image from *fnImageCaptureAndTransform* function.

																16			
12	A1	B1	C1	D1	E1	F1	G1	H1	I1	J1	K1	L1	M1	N1	O1	P1			
	A2	B2	C2	D2	E2	F2	G2	H2	I2	J2	K2	L2	M2	N2	O2	P2			
	A3	B3	C3	D3	E3	F3	G3	H3	I3	J3	K3	L3	M3	N3	O3	P3			
	A4	B4	C4	D4	E4	F4	G4	H4	I4	J4	K4	L4	M4	N4	O4	P4			
	A5	B5	C5	D5	E5	F5	G5	H5	I5	J5	K5	L5	M5	N5	O5	P5			
	A6	B6	C6	D6	E6	F6	G6	H6	I6	J6	K6	L6	M6	N6	O6	P6			
	A7	B7	C7	D7	E7	F7	G7	H7	I7	J7	K7	L7	M7	N7	O7	P7			
	A8	B8	C8	D8	E8	F8	G8	H8	I8	J8	K8	L8	M8	N8	O8	P8			
	A9	B9	C9	D9	E9	F9	G9	H9	I9	J9	K9	L9	M9	N9	O9	P9			
	A10	B10	C10	D10	E10	F10	G10	H10	I10	J10	K10	L10	M10	N10	O10	P10			
	A11	B11	C11	D11	E11	F11	G11	H11	I11	J11	K11	L11	M11	N11	O11	P11			
	A12	B12	C12	D12	E12	F12	G12	H12	I12	J12	K12	L12	M12	N12	O12	P12			

Figure 37. Example of image with size  $16 \times 12$  pixels

Figure 38 represents the desired sections of  $4 \times 3$  pixels. To get them, the initial image array has to be split, stacked and rearranged in the next way.

				16															
12	A1	B1	C1	D1	E1	F1	G1	H1	I1	J1	K1	L1	M1	N1	O1	P1			
	A2	B2	C2	D2	E2	F2	G2	H2	I2	J2	K2	L2	M2	N2	O2	P2			
	A3	B3	C3	D3	E3	F3	G3	H3	I3	J3	K3	L3	M3	N3	O3	P3			
	A4	B4	C4	D4	E4	F4	G4	H4	I4	J4	K4	L4	M4	N4	O4	P4			
	A5	B5	C5	D5	E5	F5	G5	H5	I5	J5	K5	L5	M5	N5	O5	P5			
	A6	B6	C6	D6	E6	F6	G6	H6	I6	J6	K6	L6	M6	N6	O6	P6			
	A7	B7	C7	D7	E7	F7	G7	H7	I7	J7	K7	L7	M7	N7	O7	P7			
	A8	B8	C8	D8	E8	F8	G8	H8	I8	J8	K8	L8	M8	N8	O8	P8			
	A9	B9	C9	D9	E9	F9	G9	H9	I9	J9	K9	L9	M9	N9	O9	P9			
	A10	B10	C10	D10	E10	F10	G10	H10	I10	J10	K10	L10	M10	N10	O10	P10			
	A11	B11	C11	D11	E11	F11	G11	H11	I11	J11	K11	L11	M11	N11	O11	P11			
	A12	B12	C12	D12	E12	F12	G12	H12	I12	J12	K12	L12	M12	N12	O12	P12			

Figure 38. Imagined sections of  $4 \times 3$  pixels

In Figure 39 the first step of an image array transformation is shown. The array *inputImageArray* is split into 4 sections horizontally (along its width) of  $4 \times 12$  pixels to get *dataSplit* numpy array.

																dataSplit			
arr1				arr2				arr3				arr4							
A1	B1	C1	D1	E1	F1	G1	H1	I1	J1	K1	L1	M1	N1	O1	P1				
A2	B2	C2	D2	E2	F2	G2	H2	I2	J2	K2	L2	M2	N2	O2	P2				
A3	B3	C3	D3	E3	F3	G3	H3	I3	J3	K3	L3	M3	N3	O3	P3				
A4	B4	C4	D4	E4	F4	G4	H4	I4	J4	K4	L4	M4	N4	O4	P4				
A5	B5	C5	D5	E5	F5	G5	H5	I5	J5	K5	L5	M5	N5	O5	P5				
A6	B6	C6	D6	E6	F6	G6	H6	I6	J6	K6	L6	M6	N6	O6	P6				
A7	B7	C7	D7	E7	F7	G7	H7	I7	J7	K7	L7	M7	N7	O7	P7				
A8	B8	C8	D8	E8	F8	G8	H8	I8	J8	K8	L8	M8	N8	O8	P8				
A9	B9	C9	D9	E9	F9	G9	H9	I9	J9	K9	L9	M9	N9	O9	P9				
A10	B10	C10	D10	E10	F10	G10	H10	I10	J10	K10	L10	M10	N10	O10	P10				
A11	B11	C11	D11	E11	F11	G11	H11	I11	J11	K11	L11	M11	N11	O11	P11				
A12	B12	C12	D12	E12	F12	G12	H12	I12	J12	K12	L12	M12	N12	O12	P12				

Figure 39. First step of data transformation - data split

The sections from split array *dataSplit* are now stacked in *dataStack* numpy array in the way that the first array from left to right (arr1) gets to the top and the array on the right side (arr4) gets to the bottom of the data stack array. This step is shown in Figure 40.

dataStack			
A1	B1	C1	D1
A2	B2	C2	D2
A3	B3	C3	D3
A4	B4	C4	D4
A5	B5	C5	D5
A6	B6	C6	D6
A7	B7	C7	D7
A8	B8	C8	D8
A9	B9	C9	D9
A10	B10	C10	D10
A11	B11	C11	D11
A12	B12	C12	D12
E1	F1	G1	H1
E2	F2	G2	H2
E3	F3	G3	H3
E4	F4	G4	H4
E5	F5	G5	H5
E6	F6	G6	H6
E7	F7	G7	H7
E8	F8	G8	H8
E9	F9	G9	H9
E10	F10	G10	H10
E11	F11	G11	H11
E12	F12	G12	H12
I1	J1	K1	L1
I2	J2	K2	L2
I3	J3	K3	L3
I4	J4	K4	L4
I5	J5	K5	L5
I6	J6	K6	L6
I7	J7	K7	L7
I8	J8	K8	L8
I9	J9	K9	L9
I10	J10	K10	L10
I11	J11	K11	L11
I12	J12	K12	L12
M1	N1	O1	P1
M2	N2	O2	P2
M3	N3	O3	P3
M4	N4	O4	P4
M5	N5	O5	P5
M6	N6	O6	P6
M7	N7	O7	P7
M8	N8	O8	P8
M9	N9	O9	P9
M10	N10	O10	P10
M11	N11	O11	P11
M12	N12	O12	P12

Figure 40. Second step of data transformation - data stack

The final numpy array which is gotten is *dataResult* that can be seen in Figure 41. The next step is to transform *dataStack* into *dataResult* by reshaping it into 16 arrays with 12 values in the way it takes each 3 successive rows from *dataStack* and puts them into one 1-dimensional array which is within 2-dimensional *dataResult* array. Final shape of *dataResult* array and its sections with 12 elements can be seen in Figure 41.

To explain how this has been achieved, reshape method is going to be explained. Reshape is method on numpy array which for parameters takes numpy array which is to be reshaped and an integer or a tuple of integers which determine shape of a new array. In this case, the input array of the method is *dataStack* numpy array and a tuple of integers (12, 16) where the first number (12) determines length of 1-dimensional array within 2-dimensional (*dataResult*) array, while the second number (16) is length of 2-dimensional (*dataResult*) array.

dataResult											
A1	B1	C1	D1	A2	B2	C2	D2	A3	B3	C3	D3
A4	B4	C4	D4	A5	B5	C5	D5	A6	B6	C6	D6
A7	B7	C7	D7	A8	B8	C8	D8	A9	B9	C9	D9
A10	B10	C10	D10	A11	B11	C11	D11	A12	B12	C12	D12
E1	F1	G1	H1	E2	F2	G2	H2	E3	F3	G3	H3
E4	F4	G4	H4	E5	F5	G5	H5	E6	F6	G6	H6
E7	F7	G7	H7	E8	F8	G8	H8	E9	F9	G9	H9
E10	F10	G10	H10	E11	F11	G11	H11	E12	F12	G12	H12
I1	J1	K1	L1	I2	J2	K2	L2	I3	J3	K3	L3
I4	J4	K4	L4	I5	J5	K5	L5	I6	J6	K6	L6
I7	J7	K7	L7	I8	J8	K8	L8	I9	J9	K9	L9
I10	J10	K10	L10	I11	J11	K11	L11	I12	J12	K12	L12
M1	N1	O1	P1	M2	N2	O2	P2	M3	N3	O3	P3
M4	N4	O4	P4	M5	N5	O5	P5	M6	N6	O6	P6
M7	N7	O7	P7	M8	N8	O8	P8	M9	N9	O9	P9
M10	N10	O10	P10	M11	N11	O11	P11	M12	N12	O12	P12

Figure 41. Result of data transformation

**"From Array to 100-Bit Sequence"**. The sections are then transformed into bit sequence in the way that each section gets one matching bit value in bit sequence. If there is a high value bit (1), then the entire section is matched with bit of value 1 in the sequence.

Only 1 bit of value 1 in a 3072-pixels section is enough to match a section with a bit of value 1 in the bit sequence because there are many more bits in the image than average number of bright pixels, which are in proportion 307200:420, which means there is in average 420 bright pixels (the average of bright pixels per image is calculated from the sample of 441 captured images and numbers of their bright pixels are shown in the Figure 42) on the image of 307200 pixels (25 scintillation flashes per image in average). This gives relatively low possibility that just every flash appears on an intersection of four surrounding sections so it is enabled that a section with only 1 bright pixel is matched with bit of value 1 in the bit sequence. In this way is gotten average of about 29 bits of value 1 in a bit sequence of 100 bits, which can be seen in the Figure 43.

Pixels are mainly grouped in groups of 17 pixels and they can appear on area of 1-4 sections. For example, in the case that no flash covers more than one section, for 25 flashes there will be 25 sections with bright pixels i.e. 25 ones in a sequence.

In the case that number of flashes is greater than usually and/or that more flashes appear on intersections of sections, it is possible to get the maximal number of 100 bits of value 1 in a row, that is random generated number 1 (after processing by "From 100-Bit Sequence to Float Number" algorithm).

In the next two figures, Figure 42 and Figure 43, there are graphically presented data and average values of number of bright pixels per image (Figure 42), and average number of bits of value 1 in bit sequence of 100 bits (Figure 43) respectively. These data are calculated from the sample of 441 images captured with the aim of collecting these statistical data.



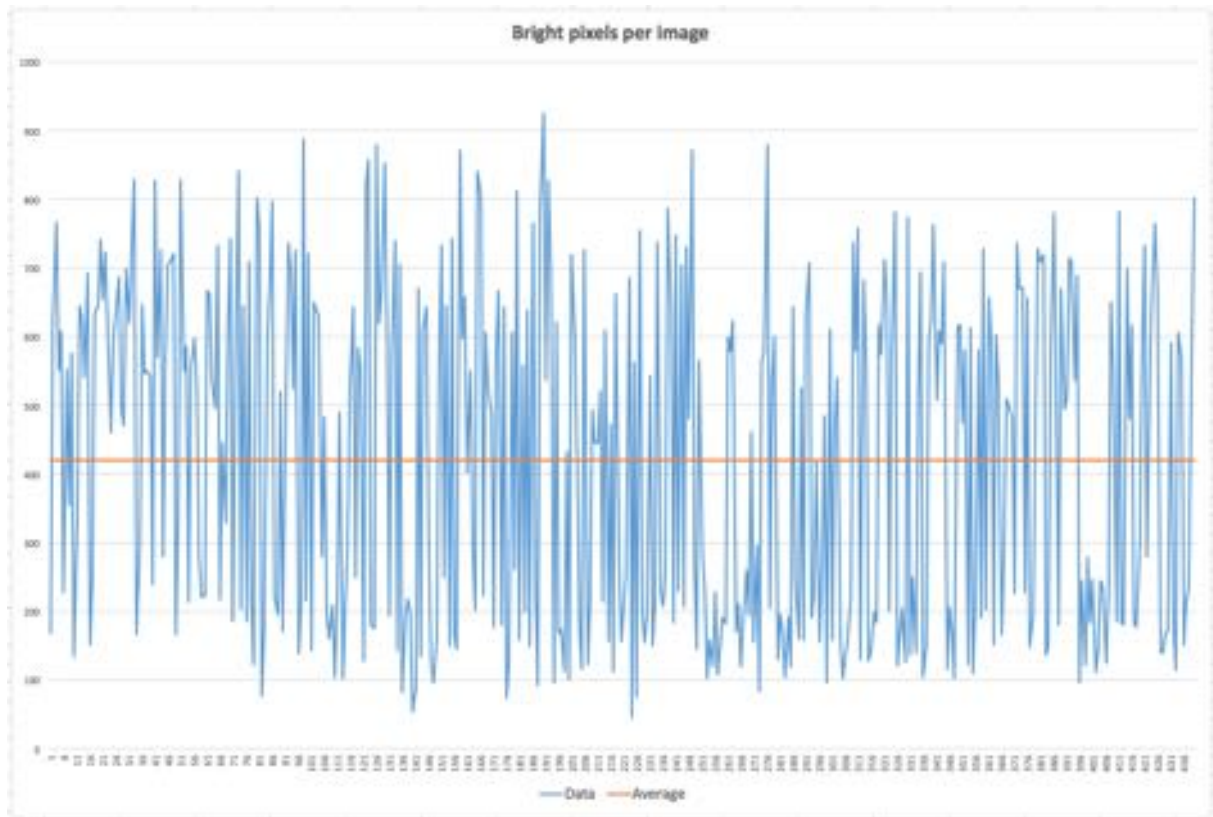


Figure 42. Number of bright pixels per image

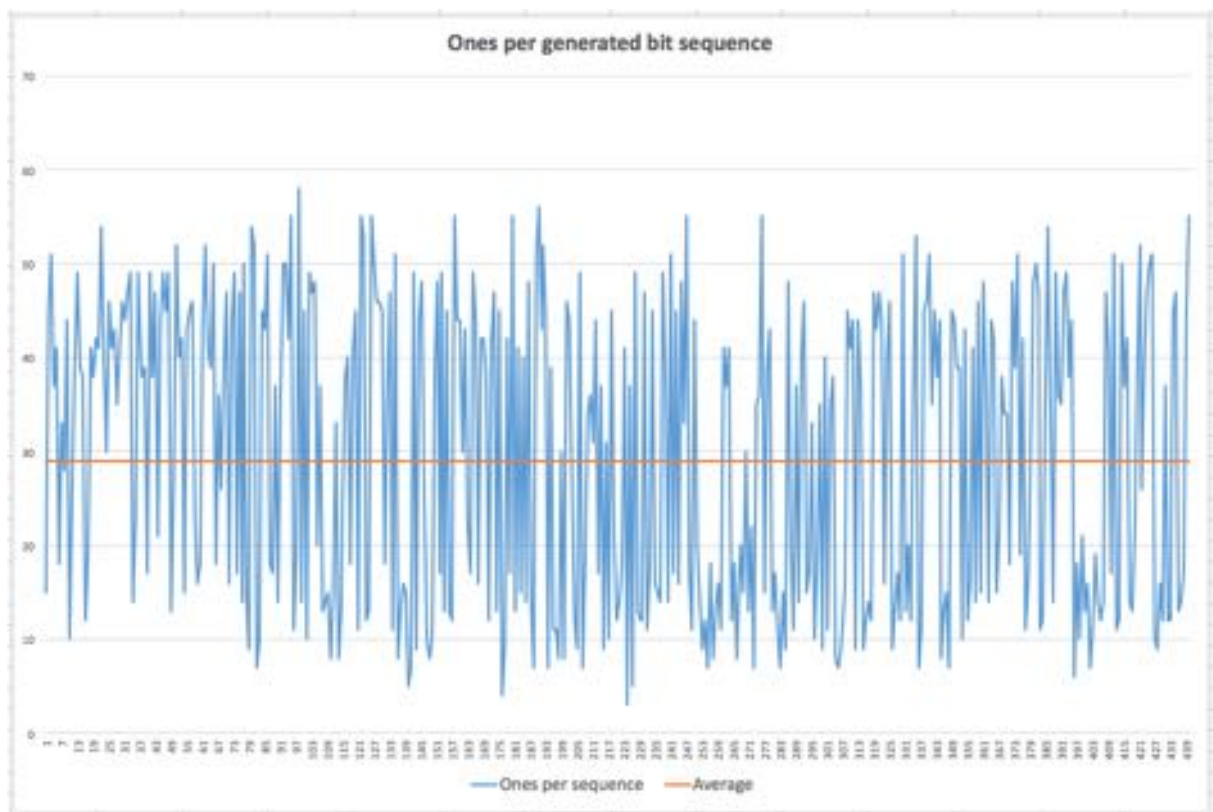


Figure 43. Number of bits of value 1 per 100-bit bit sequence



**"From 100-Bit Sequence to Float Number"**. In *fnImageArrayToNumber* function this algorithm is used. Its input is a 100-bit bit sequence of zeros and ones i.e. bits with values 0 and 1. The first step is to remove leading zeroes from the beginning of *bitSeq* string. Then *bitSeq* is reversed to get *bitSeqRev* string. The values of these two variables are strings which represent binary numbers, which are then written in their decimal form and converted into integers. The final step is to calculate quotient of the smaller of these numbers divided by the other, greater or equal number. The gotten quotient is a number from range  $[0,1]$  which is base for calculating further wanted types of output number such as 1/0 (true/false) and number from range  $[\min, \max]$ , while it is already number from 0 to 1.

There are several issues that are important to point out.

Firstly, the importance of removing leading zeroes from the beginning of initial bit sequence is to avoid very big differences between the two numbers that would give very small number very close to 0 when divided. For example, if there was a bit sequence with 50 zeroes in front of the first "1" in the sequence and with "1" in the end, then its reversed string would be string with 1 in the beginning, which means that the reverse string would have 100 characters while the initial string had 50 characters. Therefore, the integer from initial string would be probably much less than the integer gotten from reversed string, so their quotient will be small number.

Secondly, when dividing one number with the other, it is important to always divide the smaller number by the other number, which is greater than or equal to it. Thus, gotten numbers are from wanted range from 0 to 1 inclusive. However, division by 0 is impossible so this condition is also considered, as well as the condition where two numbers have to be compared with each other prior to their division.

Finally, it is important to say that this algorithm cannot give 0 in output as a result of integer division. To overcome this problem, there are cases where output can actually be 0

and in this way the entire range of numbers  $[0,1]$  is fulfilled. Zero is gotten when the divisor in the division of the numbers is equal to 0. It is impossible to divide by 0 so in that case algorithm chooses the last case where output is 0. To conclude, by introducing this last case of algorithm output, the problem of division by zero was solved and at the same time gotten possibility for output to be 0 which would otherwise be impossible.

**"From Float Number to Number 1 / 0 (True / False)".** Algorithm which gives as its output either 1 or 0 equivalent to true or false has in its input number from range  $[0,1]$  given as output of *fnImageArrayToNumber* function. It simply takes the number and rounds it to 0 or 1 which is its desired output.

**"From Float Number to Number from min to max".** Algorithm of converting a float number to a number from range  $[\min, \max]$  where min and max are given minimum and maximum values respectively uses the following formula:

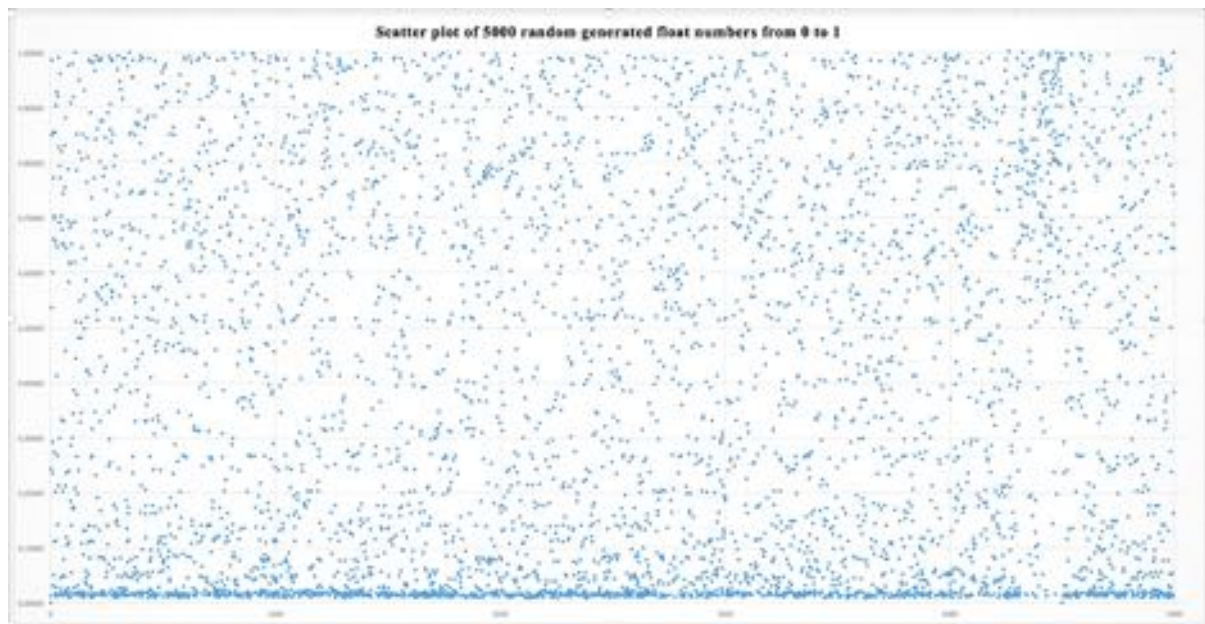
$$number = (max - min) \times numTo1 + min$$

This formula is designed to have uniform distribution of numbers in the range from minimum to maximum including. To get this, the first thing that has to be calculated is the difference between min and max numbers. Next, to get a number from the range  $[0, \max - \min]$ , the product of  $(\max - \min)$  difference and the number from range  $[0,1]$  is calculated. In the end, the number gotten as product has to be added to minimum to get float number from  $[\min, \max]$  range in output. There are 2 choices of this number's form – integer or float. If integer is asked for, then the number is rounded to the nearest integer. Otherwise, output is float number.

### Critical Case

The critical case which is going to be explained is proved by the statistics of the sample of 5000 frames captured in sequence with 0.2 seconds of pause between each 2

successive frame captures. In the Figure 44 the random generated numbers from the sample are graphically shown on the scatter plot where x-axis represents number of captured frame and y-axis random generated float number from 0 to 1. The average number of the sample is approximately 0.426408886 which is proof that the critical case is relatively well-solved. However, it can be noticed the average is less than 0.5 (ideal average) and the cause for that is the greater number of generated numbers which are equal to or relatively very close to 0, which can be seen as the denser area at the bottom of the scatter plot (Figure 44). This is the critical case.



*Figure 44.* Scatter plot of 5000 random generated float numbers

This critical case is noticed where there is a very big difference between initial and reversed number which have one or more bits of value 1 on one end, and bits of values 0 on the other end. In this case, the number from reversed string will be either much greater or much smaller than the first number. The algorithm output in this case will be number very close to 0 i.e. very small number. For instance, let 100-bit number be such that its first digit is

1 and all others are 0. Its reversed number is number with the first 99 digits equal to 0 and the last digit 1. The same case is with opposite case because the algorithm always divides the smaller number of the two by the other number which is greater than or equal to it. In the division of the numbers from the given example gotten number is  $1.5777218104420236 \times 10^{-30}$ . Then, if it is required to get an integer number from a range e.g. [1, 10] from the gotten number, the "From Float Number to Number from min to max" algorithm will give output equal to minimum number. This will generally happen if required minimum and maximum are relatively small numbers or/and if their difference is relatively small. In this case of range [1, 10] there are both of these things. On the other hand, this case makes generating a random number close to the minimum of a given range possible, which is especially important for getting numbers close to minimum in the cases of relatively big numbers or/and numbers with relatively big difference.

It is useful to mention that in the process of thinking about critical cases of the "From 100-Bit Sequence to Float Number" algorithm, there was considered also another critical case – the case when a bit sequence is symmetric such as 100...001 or 00...1111...00 which when reversed gives the same sequence. In this way the numbers gotten from the initial and reversed bit sequence are also same, so when one of them is divided by the other, quotient is 1 because the number divided by itself gives 1. Since there are more symmetric bit sequences, there is greater possibility the generated number will be 1. However, this case, which was in the beginning considered critical, has actually turned out to be very efficient for increasing the average of generated numbers which is less than it should be and it also contributes to better numbers distribution. Therefore, it is an advantage rather than a disadvantage of the algorithm.

## Inputs and Outputs

### Inputs.

- System input

System input is captured image. Image capture is done by

*fnImageCaptureAndTransform* function.

- User input through command line interface

Options and arguments from user input are parsed in main function and passed to *fnRandNumGen* function for further process.

Options and arguments are:

*--out* : type of output with arguments : *tf* - 1=true/0=false, *tol* - float number to 1, *numint* – integer number from range [from,to], *numflt* – float number from range [from,to]

*--from* : argument is integer number as minimum in generated range

*--to* : argument is integer number as maximum in generated range

*--help* : help for how to use script

- User input when program is executed by call from another program script

Parameters which are passed directly to *fnRandNumGen* function are:

*outType* – string type; one of *<tf|tol|numint|numflt>* – required parameter

*from* – integer type – optional parameter

*to* – integer type – optional parameter

### Outputs.

- Random number

Random number of required format. The number is also from wanted range [min, max] if integer or float number type was chosen.

- Exception information

Possible error or exception information is raised from the script in case of wrong input, which is done by calling *fnException* function.

### Functions in Program

**main.** Main function (*main*) does 2 major things. Firstly, it is given user input arguments. Secondly, input is subsequently verified so that program ends in the case of incorrect input or outputs a random number in the case of correct input.

If any input option or argument is incorrect, then *main* function calls *fnException* function and passes it a particular exception message and then the function prints the message and ends the program.

If the input is correct, main function calls *fnRandNumGen* function with an output number type (and optionally min and max numbers) and then the function outputs random generated number of the forwarded type.

**fnException.** Function *fnException* is called every time when exception is required to inform user about an error or wrong input. It prints the appropriate message it receives as input from caller function as the first parameter and as the second parameter it receives number one if print of the options menu is required.

**fnImageCaptureAndTransform.** Function *fnImageCaptureAndTransform* captures a random image from Alpha Random camera, transforms it into a grayscale image and processes the grayscale image to get higher contrast.

The first part of the function creates an instance of CV2 object named *ARcamera* and sets settings for better captured image quality. The very first setting is to set value of *camera\_port* constant which stores number of a system port for Alpha Random camera. Another settings of *ARcamera* object are set in order to get better quality of a captured image.

When process of an image capture begins, the image is captured using *ARcamera* object as BGR image and written into variable *capturedImage*. The next step is to close camera by deleting object *camera* so as to stop image capture process. Image data written into variable *capturedImage* are converted from the BGR image into the grayscale image in variable *grayImage* using CV2 function *cvtColor*. The final step is used to give a higher contrast to picture, which is done by setting all the values of the grayscale image which are less than 5 to 0 and others, which are greater than or equal to 5, to 255. The value of 5 is subject to changes due to a contrast needed regarding dark current effect and it has to be corrected in computer/Alpha Random camera calibration process. In the end, the function returns value of *grayImage* variable for further process.

In case of error, an exception is raised to inform user about problem with Alpha Random Camera (Figure 45) and system calibration is needed if the problem persists.

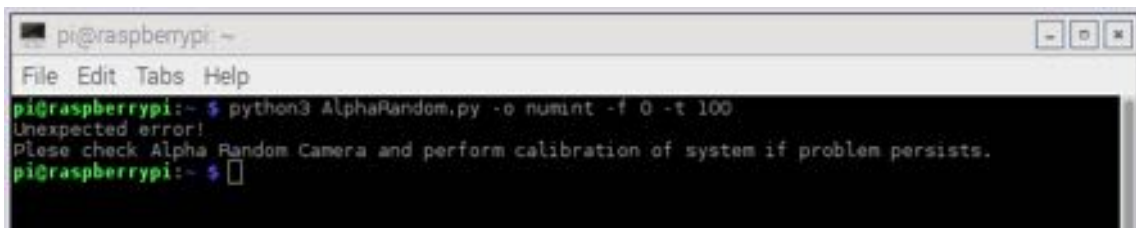


Figure 45. Example of error message gotten from *AlphaRandom.py* program when it is executed from CLI

**fnImageArrayToNumber.** Function *fnImageArrayToNumber* calls *fnImageCaptureAndTransform* function to get image array *inputImageArray* which is going to be transformed. This array is processed using "Split and Stack" algorithm. It is firstly horizontally (along width) split in 64 sections and as such stored into *dataSplit* numpy array. Array *dataSplit* is then stacked and stored into *dataStack* numpy array which is reshaped to get, finally, 2-dimensional numpy array *dataResult* with 3072 sections each of which contains

100 pixels. The next step is to create 100-bit sequence (string) *bitSeq*, which is done using "From Array to 100-Bit Sequence" algorithm. Finally, 100-bit string *bitSeq* is converted to float number *randNum* using "From 100-Bit Sequence to Float Number" algorithm.

Randomly generated number *randNum* is passed to *fnRandNumGen* function.

**fnRandNumGen.** Function *fnRandNumGen* is called from main function or directly from another caller script as a service. It gets 1-3 arguments from main function: output type (*outType*) and optionally *inMin* and *inMax* which are required in case that output type is *numint* or *numflt*. In case that either of parameters *inMin* or *inMax* is not defined, values *inMin* = 0 and *inMax* = 100 will be used as default values. If *numint* or *numflt* output type is used, the function tries to convert value of *inMin* and *inMax* parameters to integer type if it is possible. Converting to integer means that only its integer part is taken. If any parameter cannot be converted into integer or if parameter *inMax* is less than or equal to parameter *inMin*, an exception will be raised. In case that *outType* is either true/false (*tf*) or number to 1 (*to1*), the function is passed only 1 argument – output type, while the other 2 arguments are initialized to their default value of 0 and they are not used further in the function. However, if the choice of output type is integer or float from given range (*numint* or *numflt*), then the other 2 arguments (minimum and maximum number of range) are needed, so they are passed as well.

In the case that chosen output type is true/false, "From Float Number to Number 1 / 0 (True / False)" algorithm is used to get either 1 (true) or 0 (false) value from number which is passed from *fnImageArrayToNumber* function. The value of 1 / 0 is then written into variable *outputResult*.

If a user has chosen number to 1 as the wanted output type, the number which is passed from *fnImageArrayToNumber* function is directly stored into variable *outputResult* without any processing since it is already number to 1 (from range [0,1]).



The last possible case is that chosen output type is either *numflt* or *numint*. Firstly, random number is passed from *fnImageArrayToNumber* function and stored into variable *numMinMax*. Number *numMinMax* as well as minimum *inMin* and maximum *inMax* numbers are used in "From Float Number to Number from min to max" algorithm to get the number stored into variable *outputResult*. If a user wants a float number, it is immediately returned as *outputResult*. However, if a user has chosen an integer number, then *outputResult* is rounded and returned to caller function.

## How to Use Program

### Prerequisites for Program Execution.

- Drivers for camera used in Alpha Random Camera must be installed on operating system and work properly.
- Alpha Random Camera needs to be connected with USB cable to computer system.
- Computer system and Alpha Random Camera must be in running state.
- Alpha Random Camera cooling system must be running at least twenty to thirty minutes (depends on room temperature where system resides) to get an image that can be used in process.
- Alpha Random system needs to be calibrated for use. See "Alpha Random Camera – Calibration Process" chapter.

**Program Execution.** Main program script *AlphaRandom.py* is intended to be used from command line or as a service when it is called from another script as a module.

This script will generate a random number of one of the next types:

- `true(1)/false(0)`
- number from 0 to 1 (range `[0,1]`)
- integer number from input range `[min,max]`

- floating point number from input range [min,max].

**Execution of Program in Command Line Interface.** Program script should be copied to directory on file system and run from command line interface. The script should be run with parameters and output will be returned as according to passed arguments. After the program script execution, number in requested format will be printed out in command line as a result.

If an option output type is not defined, output will be `--out=tol` by default. If option `--from` and/or `--to` values are not defined for output types *numint* and *numflt*, used values will be `--from=0` and/or `--to=100` as default.

Examples of use:

*AlphaRandom.py -o tf*

*AlphaRandom.py --out=tol*

*AlphaRandom.py -o numint -f 100 -t 200*

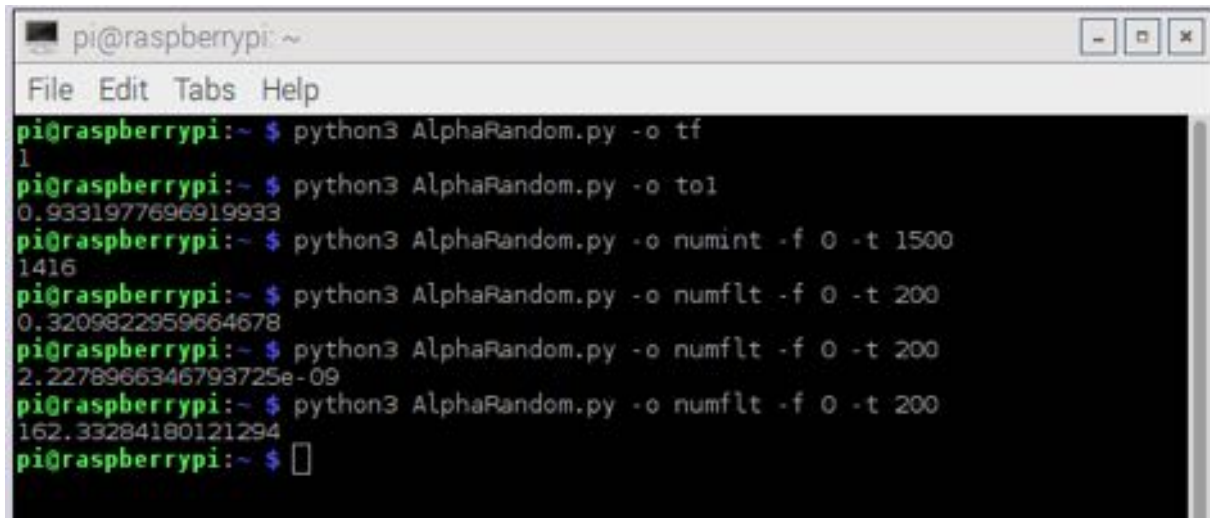
*AlphaRandom.py --out=numflt --from=100 --to=200*

Command line execution example:

*Python3 /Users/lplese/AlphaRandom.py -o numint -f 100 -t 200*

In Figure 46 there are more examples of *AlphaRandom.py* program execution in command line interface.

Note: Path */Users/lplese/* depends on script path so it has to be changed to matching script path. If program is called from script path directory, relative path *./AlphaRandom.py* can be used.



```

pi@raspberrypi: ~
File Edit Tabs Help
pi@raspberrypi:~ $ python3 AlphaRandom.py -o tf
1
pi@raspberrypi:~ $ python3 AlphaRandom.py -o to1
0.9331977696919933
pi@raspberrypi:~ $ python3 AlphaRandom.py -o numint -f 0 -t 1500
1416
pi@raspberrypi:~ $ python3 AlphaRandom.py -o numflt -f 0 -t 200
0.3209822959664678
pi@raspberrypi:~ $ python3 AlphaRandom.py -o numflt -f 0 -t 200
2.2278966346793725e-09
pi@raspberrypi:~ $ python3 AlphaRandom.py -o numflt -f 0 -t 200
162.33284180121294
pi@raspberrypi:~ $

```

Figure 46. Example of *AlphaRandom.py* program execution in CLI

Program options and arguments:

*-o* or *--out*= following argument *<tf|to1|numint|numflt>*

Choose output: true(1)/false(0)|number to 1|integer|float

*-f* or *--from*= following argument *<fromnumber>*

Integer min in range. Required for output *numint/numflt*.

*-t* or *--to*= following argument *<tonumber>*

Integer max in range. Required for output *numint/numflt*.

*-h* or *-help*

Show help.

Defaults:

*--out=to1*

*--from=0*

*--to=100*

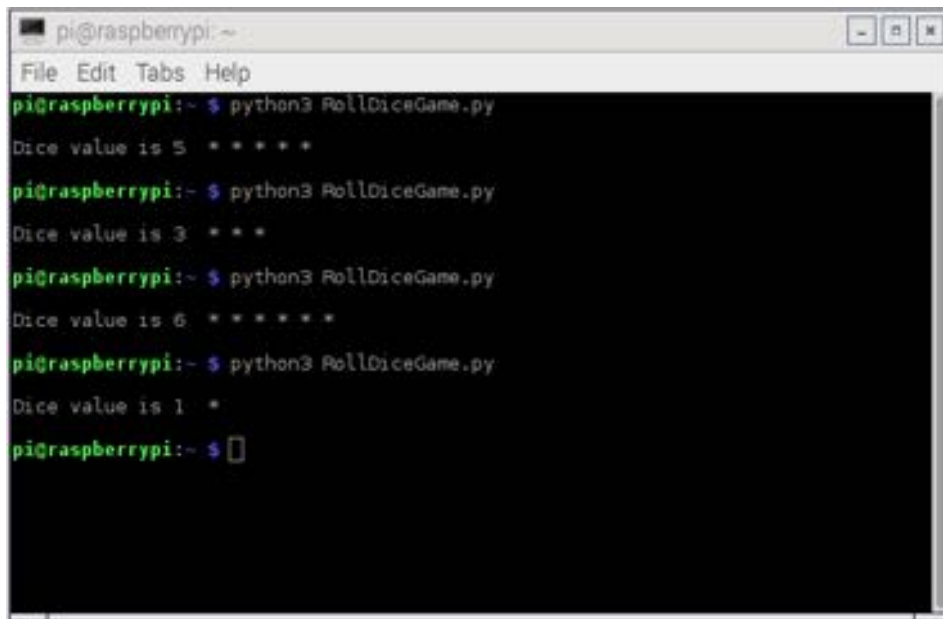
***Execution of Program as Imported Module from Another Script.*** Program

*AlphaRandom.py* can be also used as a service for another program as for instance for a web service for an RNG or as a source of random generated number for a game or some program for encryption. To be used as a service program, it needs to be imported as a module to another script (caller script). As prerequisite both scripts must be located in same directory. When external program script retrieves random generated number from *AlphaRandom.py*, program directly calls *AlphaRandom.fnRandNumGen(outType,[inMin],[inMax])* function and stores the returned value to local variable for use in caller script. Parameters passed to *fnRandNumGen* function are:

- *outType* – string type; one of *<tf|toI|numint|numflt>* – required parameter
- *inMin* – integer type - optional parameter / required for *outType numint* and *numflt*
- *inMax* – integer type - optional parameter / required for *outType numint* and *numflt*

For *numint* and *numflt* output types, parameters *inMin* and *inMax* are required. If option *inMin* and/or *inMax* values are not defined, the values used will be *inMin=0* and/or *inMax=100* .

An example of program execution from caller script is Roll Dice Game from Appendix C. Program *RollDiceGame.py* imports module *AlphaRandom* and executes *AlphaRandom.fnRandNumGen(numint,1,6)* function with parameters *numint*, 1 and 6 to generate number between 1 and 6 (inclusive) and retrieve the result into local variable *diceValue*. As the last step, the game shows result as a number which is value of variable *diceValue* and same number of asterisks symbols (\*). Example of Roll Dice Game program execution in command line interface is presented in Figure 47.



```
pi@raspberrypi: ~  
File Edit Tabs Help  
pi@raspberrypi:~ $ python3 RollDiceGame.py  
Dice value is 5 * * * * *  
pi@raspberrypi:~ $ python3 RollDiceGame.py  
Dice value is 3 * * *  
pi@raspberrypi:~ $ python3 RollDiceGame.py  
Dice value is 6 * * * * *  
pi@raspberrypi:~ $ python3 RollDiceGame.py  
Dice value is 1 *  
pi@raspberrypi:~ $
```

Figure 47. Example of *RollDiceGame.py* program execution in command line interface

## Technical Requirements

### Recommended Hardware and OS Requirements.

- Any computer and operating system which can run Python 3.4.2
- Operating system with supported driver for camera used in Alpha Random Camera
- Computer with USB 2.0 interface

### Software Requirements.

- Python 3.4.2 or higher
- OpenCV 3.0 integrated with Python - open source computer vision and machine learning software library for Python
- Python modules not included in Python standard library:

NumPy v1.11.1

CV2 – installed with OpenCV 3.0

**Computer Systems and Software Used for Development and Test.*****Computer systems.***

- Macbook White (Uni/Late 2009), CPU Intel Core 2 Duo 2.26GHz, RAM 4GB, OS X Mavericks 10.9
- Macbook Pro (Mid 2015), CPU Intel Core i7 2.5GHz, RAM 16GB, OS X El Capitan 10.11.6
- Raspberry Pi 2b, CPU ARM v7 Cortex-A7 900MHz, RAM 1GB, Raspbian Jessie
- Raspberry Pi 3b, CPU ARM v8 Cortex-A53 1.2GHz, RAM 1GB, Raspbian Jessie
- PC, CPU Intel Core i3 2.5GHz, RAM 3GB, MS Windows 7

***Software.***

- Python 3.4.2 and 3.5.2 with development environment IDLE3
- OpenCV 3.0
- Python modules NumPy v1.11.1 and CV2 (CV2 module installed with OpenCV v3.0)

### **Conclusions and Future Study**

This projects shows a simple and a relatively low-cost system for generating true random numbers. Entropy source, radioactive source used in this project, is widely available, and CMOS image sensor is cheap and can be found in consumer electronics.

Due to simple algorithms and small footprint, it is ideal for use as an add-on to programs where true randomness is required. Program can work as a service, for example as a web service on an Internet available web server to serve different kinds of programs, or can be used as a standalone script to generate one of the offered output types. Finally, scripts and algorithms can be modified and included into custom user application.

Program which is used in this project uses a simple algorithm which creates sections of captured screen image and converts them to bits in a bit sequence. Using 100-bit sequence is correlated to size, imagined sections of an image and number of scintillation flash events generated from alpha particles in 1/30 sec (30 fps is camera frame rate). This calculation has been made from statistics created from decay of radioactive source activity used in this project.

This project can be compared to the patent "Random number generator based on the spontaneous alpha-decay" (patent publication number: US6745217 B2) which also uses alpha particles source as input of random number generator. However, this patent uses digital logic and time intervals between two successive electric pulses gotten from alpha particles registered by alpha particles detector. The detector output stream goes into a binary generator that converts the signal into a bit sequence, and that requires relatively more time than the program from Alpha Random project to create the bit sequence, because the binary generator has to wait for a certain time interval to match it with a bit in the bit sequence, while the

Alpha Random program creates 100-bit sequence instantly from only one captured random image.

Program can be modified in part of algorithms to get a desired distribution of generated numbers in certain range or can be adjusted to different radioactive source activity by modifying number of sections of an image.

For hardware, there are more options for further improvements. One option is to use a radioactive source with alpha particles of higher energy than Th-232 which was used here, for example using Am-241 with alpha particles energy of 5.5 MeV. Furthermore, different types of cameras or image sensors can be used in combination with a scintillation screen which can detect a type of radiation such as alpha or beta particles or gamma rays. Also, there are many different, more stable cooling options as a direct cooling of sensor by using Peltier element or using liquid nitrogen coolers to get more stability in work of the image sensor. One more improvement which can be made is to put the image sensor and the radioactive source in a vacuumed vacuum chamber, which is how collisions of alpha particles in the air in the gap between the radioactive source and the image sensor can be avoided so as to get greater efficiency of device.

### **Acknowledgements**

I gratefully acknowledge the support of my research mentor prof. Zlatka Markucic for teaching me about programming in Python and giving me valuable guidelines that have helped me work on this project.



## Glossary of Terms

### A

active area

area of an image sensor which can receive photons and convert them in an image, 21, 23

active cooling

cooling which requires energy to cool something, in context of the project active cooler is

Peltier element which uses electric energy for cooling, 27

activity

in document, number of unstable atomic nuclei which decay per second, 1, 8, 13, 19, 20,

35, 63, 64

algorithm

method for solving a problem, it is unambiguously defined, with clearly defined both

initial/start and result states of input objects, 4, 5, 6, 7, 43, 47, 49, 50, 51, 52, 55, 56, 57,

63

Alpha Random Camera

in this project, final setup of camera with its cooling system, 2, 7, 9, 10, 14, 24, 31, 32, 38,

39, 41, 55, 57, 61, 80

APS

active-pixel sensor - an image sensor which contains an array of on-pixel sensors i.e each

pixel has its photodetector and amplifier, 15, 72

### B

back-side illumination

a method used to increase light sensitivity of an image sensor i.e. to increase its active area by putting photodiodes nearer to top part of an image sensor, 18

band gap energy

energy range in a solid-state body where no electron states can exist in a solid body, term used in solid-state physics, 34

BGR

Blue/Green/Red color space of image, it carries information both about luminosity and intensity of each of the colors (red, green, blue) for each pixel, 42, 55

bit sequence

string sequence of binary values 0 and 1 which represent low and high bit values respectively, 43, 47, 48, 49, 52, 56, 63, 64

blooming effect

in context of the project it is an undesirable phenomenon, when an alpha particle hits image sensor it produces more bright pixels, instead of only 1 bright pixel, 8, 36, 37

## C

calibration

process in which system parameters are adjusted so that the system works properly, 2, 38, 39, 57

caller function

a function which calls another function (calling function), 42, 54, 57

caller script

a script which calls another script (calling script), 41, 42, 56, 60

CCD image sensor

charge-coupled device - an image sensor which has useful properties such as low image noise and high light sensitivity, 2, 37

charge carriers

freely moving particles which carry electric charges, such as electrons and holes, 29, 34

CLI

see command line interface, 42, 55, 59

CMOS image sensor

complementary metal-oxide-semiconductor APS image sensor, it has useful properties -

low power consumption and faster readout, 1, 8, 15, 20, 21, 23, 37, 41, 63

color depth

also known as bit depth, is number of bits used to indicate each color component of a pixel,

18

command line interface

also known as command language interpreter or user interface, a way of interacting

between computer program and user, 41, 43, 53, 58, 60, 61

conduction band

the lowest energy range of the vacant electrons energy bands (unfilled bands) which can be populated by electrons, 34

cover glass

a thin piece of a transparent material put onto an image sensor, it acts both as a protective

shield for image sensor and as a filter which only transmits light, 23, 35, 37

cryptography

a branch of mathematics and computer science that studies ways of securing computer data

e.g. confidential information, 7

CV2

see OpenCV, 42, 54, 55, 61, 62

**D**

## dark current

electric current which appears in a photosensitive/photoelectric device such as image sensor even when there is no light entering the device, 8, 10, 16, 17, 25, 33, 36, 37, 38, 55

## decay

see radioactive decay, 12, 13, 14, 19, 20, 34, 63, 71

## defective pixels

pixels on a liquid crystal display which do not perform in the way they should, 2 types of defective pixels are hot and stuck pixels, 17, 33

## depletion region

also known as junction or depletion layer, a region of doped semiconductor with both p-type and n-type region where there are no mobile charge carriers, 15

## dielectric

an electrical insulation material which can be polarized by an electric field applied, it is used in capacitors as a medium for storing charge, 15

## dynamical systems

systems in which a function describes how a point in a geometrical space depends on time, 11

**E**

## efficiency

in document, number of random numbers that can be generated by an RNG in a certain amount of time, 6, 19, 20, 64

## electric current

a flow of electric charge e.g. electrons in an electric conductor, 15, 16

electric field

a vector field which is proportion of Coulomb's force exerted onto a charged particle and charge of the charged particle, 15

electromagnetic spectrum

term which includes radiation of certain frequency and wavelength, 37

electron-hole pairs

result particles of recombination process, 34

encryption

process of translating plaintext into ciphertext, 6, 60

energy band

a set of available energy states as opposed to using general discrete energy levels, 34

entropy

thermodynamical quantity which represents measure of disorder of a system, 63

## F

Fermi level

a hypothetical energy level of electron, which in case of semiconductors lies in band gap which electrons and electron holes can populate, 34

forward bias

bias in which a p-n junction conducts electric current, 15

fps

frames per second, rate at which frames at an imaging device are taken, see frame rate, 18,

63

frame rate

a rate at which frames (consecutive images) on an imaging device are displayed, 18

## G

### grayscale

a color space of an image whose pixels carry only information about luminous intensity which gives information about human visual perception of brightness, 42, 43, 54, 55

## H

### holes

electron holes, positive charge carriers which present lack of electrons where an electron could be located in a crystal lattice, 15, 34

### hot pixels

also known as sparkles, pixels with greater dark current compared to pixels surrounding them, which is why they look much brighter than they should, 17

## I

### image noise

a variation of information about colors or brightness between taken and real image, an aspect of electronic noise, 2, 25

### ionization

a process in which an atom becomes positively or negatively charged when losing or gaining electrons respectively, 34

### ionizing radiation

radiation which has enough high energy per quantum to ionize atoms i.e. to free electrons from them, 35, 36, 37, 38

### irradiated area

area which is exposed to radiation, 22

### isotope

an atom of a chemical element with same proton but different nucleon number from other atoms of the same element, 11, 12

## J

jello effect

see rolling shutter effect, 18

## K

kinetic energy

energy of motion, 14

## M

MeV

electron volt - measuring unit of amount of energy gained or lost by charge of an electron

which goes through electric potential difference of 1 volt,  $1 \text{ MeV} = 1000000 \text{ eV}$ , 12, 14, 20, 64

middle square method

a method for generating pseudorandom numbers invented by John von Neumann, 6

Monte Carlo algorithm

an algorithm from a group of algorithms which gives random and not always correct output, 7

## N

numpy

extension to Python which adds a range of multi-dimensional arrays and matrices and a library of mathematical functions useful for scientific computing, 42, 44, 45, 46, 55

**O**

one-time pad

ideal encryption technique characterized by true random input and property of frequency stability, 6

OpenCV

Open Source Computer Vision, a library of programming functions used for real-time computer vision with efficient computations, 42, 61, 62

OTP

see one-time pad, 6, 16

**P**

passive cooler

a cooler with a special design which enables cooling without use of an amount of energy, 28

Peltier element

also known as Peltier cooler/heater, a solid-state thermoelectric cooler which transfers heat by conduction from one to the other side of it using electric energy, 27, 28, 29, 30, 64

periodicity

time between two beginnings of a number sequence gotten from an RNG, a deterministic RNG is periodic, while a nondeterministic RNG is aperiodic, 6

photodetectors

also known as photosensors, sensors which detect electromagnetic energy such as light, 14, 71

photoelectric effect

also known as photoemission, release of charge carriers such as electrons from a material, it is consequence of photons hitting a surface of the material, 15, 34



## photoelectrons

electrons produced in photoelectric effect, 15

## photons

quanta of electromagnetic radiation such as light, massless particles, force carriers of

electromagnetic force, 14, 15, 17, 34, 71

## photosensitive device

device which reacts to receiving photons, 16

## potential well

a region where an atomic nucleus is situated and that has a lower potential than points

immediately outside of it, 36

## primordial

primordial isotope, nuclide which was formed before formation of Earth and since then has

existed in the same form, 12

## PRNG

see pseudorandom number generator, 5

## probability theory

branch of mathematics which studies probability (possibility that an event will occur) i.e. it

analyzes randomly-behaving phenomena, 5

## pseudorandom number generator

deterministic random number generator (PRNG) which generates pseudorandom i.e. not

true random numbers by using an algorithm, 6

## Python

an interpreting programming language with relatively simple syntax, it enables quick and

efficient program testing, 1, 39, 40, 41, 61, 62, 64, 73, 77, 79, 80, 82

**Q**

quantum

smallest amount of physical entity a field consists of (e.g. quantization of electromagnetic fields where a photon is the quantum), 11

quantum chaos

a branch of physics which studies ways of how to describe dynamical systems in terms of quantum physics, 11

quantum mechanics

a branch of physics that studies processes which occur at atomic and subatomic scale and which involve subatomic particles, 11

**R**

random generated picture

a picture which is, in context of the project, generated by the TRNG i.e. with true random input, 10

random number

number which is gotten as output of a random number generator, 1, 2, 4, 5, 9, 11, 41

random number generator

device (hardware) or algorithm (software) which is used for generating random numbers, 1, 2, 4, 9, 11, 41

random samples

samples of random elements such as random numbers, 7

randomness

lack of pattern, inability to predict further flow of an event's occurring, 1, 4, 5, 6, 63

recombination

process where electrons combine with electron holes to get electron-hole pairs, which occurs when an electron goes from valence to conduction band, 34

reverse bias

bias in which there is no electric current flow through a p-n junction, 15

RNG

see random number generator, 5, 60

rolling shutter effect

effect of image distortion which happens when rolling shutter is used, which is case with CMOS sensor, 18

## S

scintillation

luminescence, of which primary radiation is ionizing radiation, it produces visible scintillation flashes, 14, 34, 35, 37, 38, 47, 63, 64

seed

a number which is used to initialize a pseudorandom number generator, 5

shutter

shutter (of an image sensor), a method of image capturing, it can be global or rolling, 15, 18, 38

silica gel

a drying agent which is in context of the project used to decrease humidity within expanded polystyrene foam box so as to keep camera electronics dry, 25

silicon

in context of the project, a semiconductor which is used in CMOS chip in 2 types (P-type and N-type), 15, 34

SiO<sub>2</sub>

chemical compound the top layer of CMOS image sensor consists of where it is used as a dielectric, 13, 15

stuck pixel

a pixel which is “stuck” in a single color (red / green / blue) and always appears as such, 17

## T

thermoelectric cooler

heat pump (e.g. Peltier element), 28

TRNG

see True Random Number Generator, 5

true random number generator

nondeterministic random number generator (TRNG) which generates true random random numbers by using true random input such as alpha particles, 6

## U

USB

Universal Serial Bus, name for cables and connectors used for connecting and communicating between electronic devices e.g. computers, 38, 57, 61

## V

valence band

the highest energy range of the energy bands at which electrons normally are (filled bands), 34

## W

white balance

adjusting image by removing unrealistic color casts to render specific color where it is important that neutral colors such as white stay neutral, 18

## **Z**

### **ZnS(Ag)**

silver activated zinc sulfide screen - scintillating material in form of polycrystalline powder fixed on a transparent foil, it enables strong scintillation flashes, 37, 38

### References

Pierre Magnan. Detection of visible photons in CCD and CMOS: A comparative view, Nuclear Instruments and Methods in Physics Research A 504 (2003), 199–212, doi:10.1016/S0168-9002(03)00792-7

Albert H. Titus, Maurice C-K. Cheung and Vamsy P. Chodavarapu (2011). CMOS Photodetectors, Photodiodes - World Activities in 2011, Prof. Jeong Woo Park (Ed.), ISBN: 978-953-307-530-3, InTech, Available from:  
<http://www.intechopen.com/books/photodiodes-world-activities-in-2011/cmos-photodetectors>

Axis Communications. CCD and CMOS sensor technology (2010), Available from:  
[http://www.axis.com/files/whitepaper/wp\\_ccd\\_cmos\\_40722\\_en\\_1010\\_lo.pdf](http://www.axis.com/files/whitepaper/wp_ccd_cmos_40722_en_1010_lo.pdf)

Philips. Philips Webcam SPC620NC VGA CMOS (2014), Available from:  
[http://download.p4c.philips.com/files/s/spc620nc\\_00/spc620nc\\_00\\_pss\\_.pdf](http://download.p4c.philips.com/files/s/spc620nc_00/spc620nc_00_pss_.pdf)

Nick Waltham. CCD and CMOS sensors, Available from: <http://www.issibern.ch/forads/sr-009-23.pdf>

Patent Random number generator based on the spontaneous alpha-decay, publication number: US 6745217 B2, Available from: <http://www.google.com/patents/US6745217>

Clifford Frondel. Systematic Mineralogy Of Uranium And Thorium (1958), Available from:

<https://pubs.usgs.gov/bul/1064/report.pdf>

Mineralogy Database. Thorite Mineral Data, Available from:

[http://webmineral.com/data/Thorite.shtml#.V-Flmjs\\_eWw](http://webmineral.com/data/Thorite.shtml#.V-Flmjs_eWw)

Michał Gumieła, Rafał Kozik. Studies of the applicability of CMOS and CCD sensors for detection, dosimetry and imaging of alpha, beta, gamma, X-ray and proton beam spots (2012), Available from:

[http://student.agh.edu.pl/~kozikr/CCD\\_APS\\_image\\_sensor\\_reaction\\_for\\_ionizing\\_radiatio.pdf](http://student.agh.edu.pl/~kozikr/CCD_APS_image_sensor_reaction_for_ionizing_radiatio.pdf)

John Betts – Fine Minerals. Image "Thorite", Available from:

<http://webmineral.com/specimens/picshow.php?id=1453&target=Thorite#.V-5lIjJh2Uk>

Tom Nelson – Randombio. Image "Single frame (1/30 sec) of a sample of americium-241 showing pure alpha particles", Available from:

<http://www.randombio.com/webcam.html>

## Appendix A

### Main Python program script "AlphaRandom.py"

```
# -*- coding: utf-8 -*-

"""

Project: Alpha Random - Alpha Particles Random Number Generator
Version: 1.0
Author: Leo Plese
e-mail: plese.leo@gmail.com
Web: alpharandom.info

Abstract:
This script is used to generate a random number of one of the next types:
true(1)/false(0), number from 0 to 1 (range [0,1]), integer number from input range
[min,max] or floating point number from input range [min,max].

Note:
If option outtype is not defined, output will be --out=to1 by default.
If option from and/or option to values are not defined, used values will be --from=0 and --
to=100

Options:
-o or --out= following argument <tf|to1|numint|numflt> Choose output:
true(1)/false(0)|number to 1|integer|float
-f or --from= following argument <fromnumber> Integer min in range. Required for
output numint/numflt
-t or --to= following argument <tonumber> Integer max in range. Required for
output numint/numflt
-h or --help Show help.

Defaults:
--out=to1
--from=0
--to=100

Example of use:
AlphaRandom.py -o tf
AlphaRandom.py --out=to1
AlphaRandom.py -o numint -f 100 -t 200
AlphaRandom.py --out=numflt --from=100 --to=200

"""

import time
import cv2
```



```

import numpy as np
import sys
import getopt

def fnImageCaptureAndTransform():
    """ Capture image from Alpha Random camera """
    try:
        camera_port = 0
        camera = cv2.VideoCapture(camera_port)

        # camera settings
        ARcamera.set(3, 640) # width
        ARcamera.set(4, 480) # height
        ARcamera.set(12, 0) # saturation
        ARcamera.set(11, 1) # contrast
        ARcamera.set(10, 0) # brightness

        time.sleep(0.1) # wait for camera to stabilize itself
        isOk, capturedImage = ARcamera.read()
        del(ARcamera)

        """ Transform image to grayscale and give high contrast """
        # transformation to grayscale image
        grayImage = cv2.cvtColor(capturedImage, cv2.COLOR_BGR2GRAY)
        # high contrast image - set with treshold number to treshold hot and stuck pixels
        grayImage[grayImage<5] = 0
        grayImage[grayImage>=5] = 255
        return grayImage
    except:
        fnException("Unexpected error!\nPlease check Alpha Random Camera and perform
calibration of system if problem persists.", 0)

def fnImageArrayToNumber():
    """ Image data transformation to 100 rectangular sections in 2-D array """
    # retrieve image from fnImageCaptureAndTransform()
    inputImageArray = fnImageCaptureAndTransform()
    # reorder values of images into rectangles 64x48 pixels and then into lists with 100x3072
    values
    dataSplit = np.hsplit(inputImageArray, 64)
    dataStack = np.stack(dataSplit, axis=0)
    dataResult = np.reshape(dataStack, (100,3072))

    """ Conversion 100 sections to 100 bit sequence """
    # give value 0 or 1 to image rectangle section and add to bitseq variable
    bitSeq = "" # 100-bit sequence
    for i in range(100):
        if np.count_nonzero(dataResult[i]) > 1:
            bitSeq += "1"

```

```

    else:
        bitSeq += "0"

    """ Convert 100-bit string to float """
    bitSeq = bitSeq.lstrip("0") # remove leading zeros
    bitSeqRev = bitSeq[::-1] # reverse bitSeq sequence

    intBitSeq = int(bitSeq, 2)
    intBitSeqRev = int(bitSeqRev, 2)
    if intBitSeq <= intBitSeqRev != 0:
        randNum = intBitSeq / intBitSeqRev
    elif intBitSeqRev < intBitSeq != 0:
        randNum = intBitSeqRev / intBitSeq
    else:
        randNum = 0

    return randNum

def fnRandNumGen(outType, inMin=0, inMax=100):
    """ Generate number of requested type """
    if outType == "tf":
        outputResult = round(fnImageArrayToNumber())
    elif outType == "to1":
        outputResult = fnImageArrayToNumber()
    elif outType == "numflt" or outType == "numint":
        try:
            inMin = int(inMin)
            inMax = int(inMax)
        except:
            fnException("Incorrect min or/and max number!", 1)

        if inMax <= inMin:
            fnException("Min number is greater than or equal to max number!", 1)

        numMinMax = fnImageArrayToNumber()
        outputResult = ((inMax - inMin) * numMinMax) + inMin

        if outType == "numint":
            outputResult = round(outputResult)
    else:
        fnException("Incorrect output option in the script argument!", 1)

    return outputResult

def fnException(excMsg, prtHelp):
    """ Exception print with short help manual """
    print(excMsg)
    if prtHelp == 1:

```

```

print("""AlphaRandom.py -o <tf|to1|numint|numflt> [-f <fromnumber> -t
<tonumber>]
Options:
-h --help          Show help.
-o --out=<tf|to1|numint|numflt> Choose output: 1=true/0=false|number to
1|integer|float
-f --from=<fromnumber> Integer min in range. Required for output
numint/numflt
-t --to=<tonumber> Integer max in range. Required for output
numint/numflt""")
sys.exit(1)

def main():
    try:
        opts, args = getopt.getopt(sys.argv[1:], "ho:f:t:", ["help", "out=", "from=", "to="])
    except getopt.GetoptError:
        fnException("Correct use of script is:", 1)

    # set default values
    outType = "to1"
    fromNum = 0
    toNum = 100

    for opt, arg in opts:
        if opt in ("-h", "--help"):
            fnException("Correct use of script is:", 1)
        elif opt in ("-o", "--out"):
            outType = arg
        elif opt in ("-f", "--from"):
            fromNum = arg
        elif opt in ("-t", "--to"):
            toNum = arg

    if outType == "tf":
        outValue = fnRandNumGen(outType)
    elif outType == "to1":
        outValue = fnRandNumGen(outType)
    elif outType == "numint" or outType == "numflt":
        outValue = fnRandNumGen(outType, fromNum, toNum)
    else:
        fnException("Incorrect output option in the script argument!", 1)

    print(outValue)

if __name__ == "__main__":
    main()

```

## Appendix B

Python program script for calibration process "AlphaRandom\_cal.py"

```
# -*- coding: utf-8 -*-

"""

Project: Alpha Random - Alpha Particles Random Number Generator
Version: 1.0
Author: Leo Plese
e-mail: plese.leo@gmail.com
Web: alpharandom.info

Abstract:
This script is used to calibrate system for Alpha Particles Random Number Generator.

Note:
Calibration reference image is saved in script path under name CalibImg.png

"""

import time
import cv2
import sys

def fnImageCaptureAndTransform():
    try:
        """ Capture image from Alpha Random camera """
        camera_port = 0
        ARcamera = cv2.VideoCapture(camera_port)

        # camera settings
        ARcamera.set(3, 640) # width
        ARcamera.set(4, 480) # height
        ARcamera.set(12, 0) # saturation
        ARcamera.set(11, 1) # contrast
        ARcamera.set(10, 0) # brightness

        time.sleep(0.1) # wait for camera to stabilize itself
        isOk, capturedImage = ARcamera.read()
        del(ARcamera)

        """ Transform image to grayscale and give high contrast """
        # transformation to grayscale image
        grayImage = cv2.cvtColor(capturedImage, cv2.COLOR_BGR2GRAY)

        """ CALIBRATION POINT - Change value 5 to lower to get brighter image or higher
```

```
to get darker image""
    grayImage[grayImage<5] = 0
    grayImage[grayImage>=5] = 255

    # save image to script path under name
    cv2.imwrite("./CalibImg.png", grayImage)
except:
    print ("Unexpected error:", sys.exc_info()[0], "\nPlease check:\n- camera_port value in
calibration script\n- Alpha Random Camera\n- hardware connections")

fnImageCaptureAndTransform()
```

## Appendix C

Python program script for Roll Dice Game which uses *AlphaRandom.py* program as a source of a random generated number. Script name is "RollDiceGame.py"

```
# -*- coding: utf-8 -*-

"""

Program: Roll Dice Game
Version: 1.0
Author: Leo Plese
e-mail: plese.leo@gmail.com
Web: alpharandom.info

Abstract:
This game uses Alpha Particles Random Number Generator for generating random number
from one to six to simulate rolling six-sided dice.

Note:
For use this program its location need to be in same directory where AlphaRandom.py is
located.

"""

import AlphaRandom

diceValue = AlphaRandom.fnRandNumGen("numint", 1, 6)

print("\nDice value is " + str(diceValue) + " " + " *" * diceValue + "\n")
```

## Appendix D

Python program script for generating random picture on file system. Simulation of Alpha

Random Camera. Script name is "AlphaRandom\_picGen.py"

```
# -*- coding: utf-8 -*-

"""

Project: Alpha Random - Alpha Particles Random Number Generator - Random Picture
Generator
Version: 1.0
Author: Leo Plese
e-mail: plese.leo@gmail.com
Web: alpharandom.info

Abstract:
This script is used for generating random picture which are used by Alpha Particles
Random Number Generator Simulation program (AlphaRandom_numGen.py).

Note:
Set number of generated pictures in variable with name numberOfPictures.
Before using simulation script set absolute file path to your image folder in variable
imagePath and number of images in variable numberOfPictures.
In the program is example of file path /home/pi/Image/ and ordinal number of last image
501.

"""

import numpy as np
from random import randint
import cv2

def main():
    # here set number (>0) of generated pictures in next variable (here: 501)
    numberOfPictures = 501

    # flash pictures
    shapeOne = np.array([1, 1, 638, 1, 1, 1, 636, 1, 1, 1, 1, 636, 1, 1, 1, 638, 1])
    shapeTwo = np.array([1, 1, 1, 1, 636, 1, 1, 1, 1, 636, 1, 1, 1, 1, 637, 1, 1, 1, 638, 1])
    shapeThree = np.array([1, 1, 1, 1, 637, 1, 1, 1, 1, 637, 1, 1, 1, 1, 637, 1, 1, 1])
    shapeFour = np.array([1, 1, 1, 637, 1, 1, 1, 1, 636, 1, 1, 1, 1, 637, 1, 1, 638, 1, 1, 639])
    shapeFive = np.array([1, 1, 638, 1, 1, 1, 636, 1, 1, 1, 1, 636, 1, 1, 1, 1, 1, 1, 635, 1, 3, 1,
1279, 1])
    shapeSix = np.array([1, 1, 1, 1, 1, 636, 1, 1, 1, 1, 637, 1, 1, 637, 1, 1, 1, 638, 1])
    shapeSeven = np.array([1, 1, 1, 1, 637, 1, 1, 1, 637, 1, 1, 1, 637, 1, 1, 1, 637, 3, 638])

    for nop in range(numberOfPictures):
```

```
randPicture = np.zeros(307200, dtype=int)
flashNum = randint(0, 35)

for x in range(0, flashNum):
    positionX = randint(0, 307199)
    shapeNumber = randint(1, 7)
    dicX = {1:shapeOne, 2:shapeTwo, 3:shapeThree, 4:shapeFour, 5:shapeFive,
6:shapeSix, 7:shapeSeven}
    shapeFlash = dicX[shapeNumber]
    for y in range(0, shapeFlash.size):
        positionX += shapeFlash[y]
        if positionX > 307199:
            y = shapeFlash.size
        else:
            randPicture[positionX] = 255

randPicture = randPicture.reshape(480, 640)
# here set absolute file path to images (here: /home/pi/Image/)
imagePath = "/home/pi/Image/"
cv2.imwrite(imagePath + "Img_" + str(nop+1) + ".png", randPicture)

main()
```



## Appendix E

Python program script for generating random number from picture on file system. Simulation of program *AlphaRandom.py*. Script name is "AlphaRandom\_numGen.py"

```
# -*- coding: utf-8 -*-
```

```
"""
```

Project: Alpha Random - Alpha Particles Random Number Generator - Simulation

Version: 1.0

Author: Leo Plese

e-mail: plese.leo@gmail.com

Web: alpharandom.info

Abstract:

This script is used to generate a random number of one of the next types: true(1)/false(0), number from 0 to 1 (range [0,1]), integer number from input range [min,max] or floating point number from input range [min,max].

This script is simulation script used for simulating AlphaRandom.py script by using pregenerated pictures on file system instead of capturing image from Alpha Random Camera.

Only function fnImageCaptureAndTransform is modified and all other functions and algorithms are same as in AlphaRandom.py program.

Before use, please, read Note.

Note:

To use this program, it is important to generate images with program AlphaRandom\_picGen.py.

Before using simulation script set absolute file path to your image folder in variable imagePath as well as number of images in variable numberOfPictures in function fnImageCaptureAndTransform.

In the function is example of file path /home/pi/Image/ and ordinal number of last image 501.

If option outtype is not defined, output will be --out=to1 by default.

If option from and/or option to values are not defined, used values will be --from=0 and --to=100

Options:

-o or --out= following argument <tf|to1|numint|numflt> Choose output:

true(1)/false(0)|number to 1|integer|float

-f or --from= following argument <fromnumber> Integer min in range. Required for output numint/numflt

-t or --to= following argument <tonumber> Integer max in range. Required for output numint/numflt

`-h` or `--help` Show help.

Defaults:

`--out=tol`

`--from=0`

`--to=100`

Example of use:

`AlphaRandom.py -o tf`

`AlphaRandom.py --out=tol`

`AlphaRandom.py -o numint -f 100 -t 200`

`AlphaRandom.py --out=numflt --from=100 --to=200`

"""

`import time`

`import cv2`

`import numpy as np`

`import sys`

`import getopt`

`from random import randint`

`import os.path`

`def fnImageCaptureAndTransform():`

`""" Reading image from file system """`

`try:`

`# here set absolute file path to images (here: /home/pi/Image/) and number of generated files (here: 501)`

`imagePath = "/home/pi/Image/"`

`numberOfPictures = 501`

`picForUse = imagePath + "Img_" + str(randint(1, numberOfPictures)) + ".png"`

`if os.path.exists(picForUse):`

`capturedImage = cv2.imread(picForUse, -1)`

`grayImage = capturedImage`

`return grayImage`

`else:`

`raise NameError("Nonexistent image!")`

`except:`

`fnException("Unexpected error!\nPlease check file path in variable imagePath, number of images in variable numberOfPictures and existence of images in file path.", 0)`

`def fnImageArrayToNumber():`

`""" Image data transformation to 100 rectangular sections in 2-D array """`

`# retrieve image from fnImageCaptureAndTransform()`

`inputImageArray = fnImageCaptureAndTransform()`

`# reorder values of images into rectangles 64x48 pixels and then into lists with 100x3072 values`

```

dataSplit = np.hsplit(inputImageArray, 64)
dataStack = np.stack(dataSplit, axis=0)
dataResult = np.reshape(dataStack, (100,3072))

""" Conversion 100 sections to 100 bit sequence """
# give value 0 or 1 to image rectangle section and add to bitseq variable
bitSeq = "" # 100-bit sequence
for i in range(100):
    if np.count_nonzero(dataResult[i]) > 1:
        bitSeq += "1"
    else:
        bitSeq += "0"

""" Convert 100-bit string to float """
bitSeq = bitSeq.lstrip("0") # remove leading zeros
bitSeqRev = bitSeq[::-1] # reverse bitSeq sequence

intBitSeq = int(bitSeq, 2)
intBitSeqRev = int(bitSeqRev, 2)
if intBitSeq <= intBitSeqRev != 0:
    randNum = intBitSeq / intBitSeqRev
elif intBitSeqRev < intBitSeq != 0:
    randNum = intBitSeqRev / intBitSeq
else:
    randNum = 0

return randNum

def fnRandNumGen(outType, inMin = 0, inMax = 100):
    """ Generate number of requested type """
    if outType == "tf":
        outputResult = round(fnImageArrayToNumber())
    elif outType == "to1":
        outputResult = fnImageArrayToNumber()
    elif outType == "numflt" or outType == "numint":
        try:
            inMin = int(inMin)
            inMax = int(inMax)
        except:
            fnException("Incorrect min or/and max number!", 1)

        if inMax <= inMin:
            fnException("Min number is greater than or equal to max number!", 1)

        numMinMax = fnImageArrayToNumber()
        outputResult = ((inMax - inMin) * numMinMax) + inMin
        if outType == "numint":
            outputResult = round(outputResult)
    else:

```

```

        fnException("Incorrect output option in the script argument!", 1)

    return outputResult

def fnException(excMsg, prtHelp):
    """ Exception print with short help manual """
    print(excMsg)
    if prtHelp == 1:
        print("""AlphaRandom.py -o <tf|to1|numint|numflt> [-f <fromnumber> -t
<tonumber>]
    Options:
    -h --help                Show help.
    -o --out=<tf|to1|numint|numflt> Choose output: 1=true/0=false|number to
1|integer|float
    -f --from=<fromnumber>      Integer min in range. Required for output
numint/numflt
    -t --to=<tonumber>          Integer max in range. Required for output
numint/numflt""")
        sys.exit(1)

def main():
    try:
        opts, args = getopt.getopt(sys.argv[1:], "ho:ft:", ["help", "out=", "from=", "to="])
    except getopt.GetoptError:
        fnException("Correct use of script is:", 1)

    # set default values
    outType = "to1"
    fromNum = 0
    toNum = 100

    for opt, arg in opts:
        if opt in ("-h", "--help"):
            fnException("Correct use of script is:", 1)
        elif opt in ("-o", "--out"):
            outType = arg
        elif opt in ("-f", "--from"):
            fromNum = arg
        elif opt in ("-t", "--to"):
            toNum = arg

    if outType == "tf":
        outValue = fnRandNumGen(outType)
    elif outType == "to1":
        outValue = fnRandNumGen(outType)
    elif outType == "numint" or outType=="numflt":
        outValue = fnRandNumGen(outType, fromNum, toNum)
    else:

```

```
    fnException("Incorrect output option in the script argument!", 1)

    print(outValue)

if __name__ == "__main__":
    main()
```