ΠΕΡΙΦΕΡΕΙΑ ΚΕΝΤΡΙΚΗΣ ΜΑΚΕΔΟΝΙΑΣ 2014-2020

ΕΥΔ ΚΕΝΤΡΙΚΗΣ ΜΑΚΕΔΟΝΙΑΣ

Επενδυτικά Σχέδια Καινοτομίας

Κωδικός πράξης: KMP6-0083129

Κωδικός MIS: 5136571

Δικαιούχος: ΑΔΕΛΦΟΙ ΓΕΩΡΓΙΟΥ ΚΟΥΡΤΙΔΗ, ΕΛΚΕ – ΔΙΕΘΝΕΣ ΠΑΝΕΠΙΣΤΗΜΙΟ ΤΗΣ ΕΛΛΑΔΟΣ

ΤΙΤΛΟΣ

«*Βελτιστοποίηση τοποθέτησης και καταμέτρησης εμπορευμάτων σε μεγάλους βιομηχανικούς χώρους με χρήση μη επανδρωμένων αεροσκαφών*»

Τίτλος Πράξης (Αγγλικά) Optimization of placement and counting products in large industrial areas using UAV

Παραδοτέο

Π3.3 Πρόγραμμα σε ελεύθερο λογισμικό αυτόματης καταμέτρησης βιομηχανικών προϊόντων

Αρ. Παραδοτέου	**Π3.3**
Ενότητα Εργασίας	**ΕΕ3. Ανάπτυξη μεθόδων αυτόματης καταμέτρησης και βέλτιστης τοποθέτησης βιομηχανικών προϊόντων**
Υπεύθυνος Φορέας ΕΕ3 / Π3	**Αδελφοί Γεωργίου Κουρτίδη - ΔΙ.ΠΑ.Ε.**
Είδος Παραδοτέου	**Τεχνική Αναφορά**
Παράδοση	**Δεκέμβριος 2023**
Σύντομη Περιγραφή Ενότητας	**Πραγματοποίηση δοκιμαστικών πτήσεων για διαμόρφωση των ΣμηΕΑ και τη συλλογή δεδομένων, ανάπτυξη μεθόδων για την καταγραφή θέσης και ποσότητας προϊόντων, ανάπτυξη μεθόδων βέλτιστης τοποθέτησης προϊόντων, πρόγραμμα σε ελεύθερο λογισμικό.**
Έναρξη Προγράμματος, Λήξη	**Οκτώβριος 2021, Δεκέμβριος 2024**
Φορείς	**ΑΔΕΛΦΟΙ ΓΕΩΡΓΙΟΥ ΚΟΥΡΤΙΔΗ,** **ΕΛΚΕ – ΔΙΕΘΝΕΣ ΠΑΝΕΠΙΣΤΗΜΙΟ ΤΗΣ ΕΛΛΑΔΟΣ**

Λίστα συγγραφέων:

#	***Ονοματεπώνυμο***	***Φορέας***	***email***
1	Γεώργιος Κουρτίδης	ΝΕΚΤΑΡ	info@nektar.gr
2	Γεώργιος Κουρτίδης	ΝΕΚΤΑΡ	info@nektar.gr
3	Ειρήνη Κουρτίδου	ΝΕΚΤΑΡ	info@nektar.gr
4	Αναστάσιος Βαζίκογλου	ΝΕΚΤΑΡ	info@nektar.gr
5	Δημήτριος Βαρσάμης	ΔΙ.ΠΑ.Ε.	dvarsam@ihu.gr
6	Αλκιβιάδης Τσιμπίρης	ΔΙ.ΠΑ.Ε.	atsimpiris@ihu.gr
7	Κωνσταντίνος Χειλάς	ΔΙ.ΠΑ.Ε.	chilas@ihu.gr
8	Πάρις Μαστοροκώστας	ΔΙ.ΠΑ.Ε.	mast@uniwa.gr
9	Δημήτριος Μάνος	ΔΙ.ΠΑ.Ε.	dmanos@ihu.gr
10	Ιορδάνης Ζιώγας	ΔΙ.ΠΑ.Ε.	ziogasi@ihu.gr
11	Ηλίας Πανταζής	ΔΙ.ΠΑ.Ε.	hpant@ihu.gr

Επιμέλεια εγγράφου:

#	***Ονοματεπώνυμο***	***Φορέας***	***email***
	Γεώργιος Κουρτίδης	ΝΕΚΤΑΡ	info@nektar.gr
	Δημήτριος Βαρσάμης	ΔΙ.ΠΑ.Ε.	dvarsam@ihu.gr

Περιεχόμενα

1.Περίληψη έργου

Ο σκοπός του παρόντος έργου είναι η ανάπτυξη ενός ολοκληρωμένου πληροφοριακού συστήματος με στόχο την βέλτιστη τοποθέτηση εμπορευματοκιβωτίων σε μεγάλους βιομηχανικούς χώρους. Για την επίτευξη του αρχικά, θα δημιουργηθεί μία κεντρική σχεσιακή βάση δεδομένων η οποία θα ενοποιεί δεδομένα από διαφορετικές πηγές και χρονικές στιγμές, της επιχείρησης. Έπειτα, θα γίνει καθορισμός των απαιτήσεων και μεθόδων για την ανάλυση των δεδομένων της βάσης μέσω της σκιαγράφησης ενός γενικότερου προτύπου εκτέλεσης όλων των διαδικασιών παραγωγής. Η αποθήκη δεδομένων (warehouse), θα λειτουργήσει ως βάση για την ανάπτυξη και εφαρμογή αλγορίθμων που έχουν ως στόχο την εξαγωγή και παρουσίαση Business Analytics υπό μορφή γραφημάτων, πινάκων και δεικτών.

Ακολούθως, θα πραγματοποιηθούν δοκιμαστικές πτήσεις των ειδικών για το έργο ΣμηΕΑ, πρωτίστως για την διαμόρφωσή τους και εν συνεχεία για την καταγραφή της θέσης και της διαθέσιμης ποσότητας του κάθε προϊόντος. Με βάση τα δεδομένα αυτά, θα αναπτυχθούν υπολογιστικές μέθοδοι οι οποίες θα είναι σε θέση να εντοπίζουν το ζητούμενο προϊόν και να υπολογίζουν την διαθεσιμότητά του. Σύμφωνα με τα δεδομένα αυτά, αλλά και με χρήση business analytics, προκύπτουν συμπεράσματα σχετικά με την ζήτηση και την αναγκαία επάρκεια των προϊόντων.

Στη συνέχεια θα αναπτυχθεί μία ειδική μέθοδος βελτιστοποίησης η οποία με δεδομένα όλα τα παραπάνω στοχεύει στην βέλτιστη τοποθέτηση των παραγόμενων προϊόντων στην αποθήκη με βάση συγκεκριμένες προδιαγραφές.

Τέλος, θα αναπτυχθεί ολοκληρωμένο πληροφοριακό σύστημα το οποίο θα εκτελεί αυτόματη καταμέτρηση της αποθήκης και θα προτείνει την βέλτιστη τοποθέτηση προϊόντων βάση προδιαγραφών που θα δίνονται από τον χρήστη. Το σύστημα αυτό, γίνεται πράξη μέσω της ανάπτυξης ολοκληρωμένης πλατφόρμας αλλά και εφαρμογής για έξυπνες κινητές συσκευές, με τη βοήθεια της οποίας οι εργαζόμενοι της επιχείρησης θα μπορούν να έχουν εικόνα της αποθήκης προϊόντων κάθε στιγμή καθώς και την δυνατότητα μετάδοσης της εικόνας αυτής μεταξύ συνεργατών μέσω ηλεκτρονικού ταχυδρομείου ή κινητής τηλεφωνίας για την επίτευξη της καλύτερης δυνατής επικοινωνίας και συνεργασίας αυτών.

Το έργο λοιπόν, υποβοηθώντας στη διαδικασία λήψης αποφάσεων, θα συνδράμει σημαντικά στη μείωση των εξόδων της εταιρίας, θα μειώσει τον φόρτο εργασίας των εργαζομένων απαλλάσσοντάς τους από τετριμμένες διαδικασίες και θα καταστήσει την επιχείρηση ιδιαιτέρως ανταγωνιστική στον χώρο της παραγωγής. Επιπρόσθετα, η αξία ενός τέτοιου προϊόντος δεν περιορίζεται μόνο στο λειτουργικό του κομμάτι αλλά εστιάζεται και στο εμπορικό, καθιστώντας την επιχείρηση που το διαθέτει άκρως ανταγωνιστική στην αγορά εργασίας, καθώς μία τέτοια εφαρμογή προσδίδει υπεραξία στην επιχείρηση αν επιλεγεί η εμπορική της εκμετάλλευσή της.

Το έργο αποτελείται από έξι επιμέρους ενότητες εργασίας, η ολοκλήρωση των οποίων θα οδηγήσει στην επίτευξη του έργου. Οι ενότητες αυτές είναι:

- ΕΕ1 Σχεδίαση υλοποίησης έργου
- ΕΕ2 Ανάπτυξη μεθόδων και συστημάτων οργάνωσης δεδομένων

- ΕΕ3 Ανάπτυξη μεθόδων αυτόματης καταμέτρησης και βέλτιστης τοποθέτησης βιομηχανικών προϊόντων
- ΕΕ4 Ανάπτυξη διαδικτυακής πλατφόρμας και εφαρμογής σε έξυπνες κινητές συσκευές
- ΕΕ5 Πιλοτική λειτουργία και αξιολόγηση ολοκληρωμένου συστήματος
- ΕΕ6 Υποστήριξη έργου

2. Περίληψη Ενότητας Εργασίας 3 (ΕΕ3)

Κατά τη διάρκεια της ΕΕ3 θα πραγματοποιηθούν δοκιμαστικές πτήσεις για την βέλτιστη διαμόρφωση των ΣμηΕΑ. Παράλληλα, με βάση τα δεδομένα που θα συλλεχθούν, θα αναζητηθούν μέθοδοι και παραμετροποιήσεις έτσι ώστε να πραγματοποιείται σωστή καταγραφή των προϊόντων ως προς την θέση τους και ως προς την ποσότητα τους. Θα αναπτυχθούν εξειδικευμένες υπολογιστικές μέθοδοι οι οποίες με βάση τα δεδομένα που θα συλλέγουν τα ΣμηΕΑ, θα εντοπίζουν το προϊόν, τη θέση του και θα καταμετρούν την ποσότητα ανά θέση και ανά προϊόν. Με βάση τις πληροφορίες από την αποθήκη δεδομένων και τα αποτελέσματα των υπολογιστικών μεθόδων, θα αναπτυχθεί μια ειδική μέθοδος βελτιστοποίησης η οποία έχει σκοπό τη βέλτιστη τοποθέτηση των παραγόμενων προϊόντων στην αποθήκη με βάση συγκεκριμένες προδιαγραφές, για παράδειγμα με βάση τις αναμενόμενες πωλήσεις, τους διαθέσιμους πόρους κ.α.

Παραδοτέα:

- Π3.1 Μέθοδος αυτόματης καταμέτρησης βιομηχανικών προϊόντων
- Π3.2 Μέθοδος βέλτιστης τοποθέτησης παραγόμενων βιομηχανικών προϊόντων
- Π3.3 Πρόγραμμα σε ελεύθερο λογισμικό (π.χ. Python) αυτόματης καταμέτρησης βιομηχανικών προϊόντων
- Π3.4 Πρόγραμμα σε ελεύθερο λογισμικό (π.χ. Python) βέλτιστης τοποθέτησης παραγόμενων βιομηχανικών προϊόντων

3.Πρόγραμμα σε ελεύθερο λογισμικό (π.χ. Python) αυτόματης καταμέτρησης βιομηχανικών προϊόντων

Παρακάτω αναλυτικά ο κώδικας σε γλώσσα Python, για την αυτόματη καταμέτρηση βιομηχανικών προϊόντων:

```
import numpy as np

# ------------------------------------------------------------------
# scan_box
# ------------------------------------------------------------------
def scan_box(
    hits,
    flight_height,
    palette_width,
    palette_height,
    stacks_per_row
):
  # Αρχικοποίηση λίστας αποτελέσματος
  result = []

  # Οι διαστάσεις του πίνακα τιμών του lidar
  rows = hits.shape[0]
  hits_per_row = hits.shape[1]

  # Επανάλαβε για κάθε γραμμή του box
  for row in range(rows):
    row_hits = hits[row]

    # Μετατροπή ύψους σε πλήθος παλετών
    palettes = np.zeros(hits_per_row)
    for i in range(hits_per_row):
      palettes[i] = (flight_height - row_hits[i]) / palette_height

    # Επανέλαβε για κάθε θέση στοίβας στο box (πλάτους palette_width)
    row_result = []
    for i in range(stacks_per_row):
      stack_hits = []
      # Για κάθε μέτρηση κατά πλάτος της στοίβας
      for h in range(palette_width):
        stack_hits.append(palettes[i*palette_width + h])
      # Προσθήκη της μέσης τιμής στον πίνακα αποτελεσμάτων της γραμμής του box
      average = sum(stack_hits) / len(stack_hits)
      row_result.append(round(average))

    # Προσθήκη των αποτελεσμάτων της γραμμής στον τελικό πίνακα
    result.append(row_result)

  # Επιστροφή αποτελέσματος
  return(result)

# ------------------------------------------------------------------
# scan_warehouse
# ------------------------------------------------------------------
def scan_warehouse(
```

```
    hits,
    resolution,
    flight_height,
    palette_width,
    palette_height,
    error_margin,
    accepted_width_pc
):
  # Αρχικοποίηση λίστας αποτελέσματος
  result = []

  # Το πλήθος των γραμμών
  rows = hits.shape[0]

  # Επανάλαβε για κάθε γραμμή του box
  for row in range(rows):
    row_hits = hits[row]
    row_result = []

    # Αρχικοποίηση του group
    current_group=[]
    current_group_average = 0

    # Επανέλαβε για κάθε μέτρηση στην γραμμή
    for hit in row_hits:

      # Αν το group είναι άδειο πρόσθεσέ την
      if len(current_group) == 0:
        current_group.append(hit)
        current_group_average = hit
        continue

      # Έλεγχος αν το hit ανήκει στο current_group
      # Ειδικός έλεγχος για την τιμή 0
      bottom_margin = hit * (1-error_margin)
      top_margin = hit * (1+error_margin)
      if (hit == 0 and current_group_average == 0) or
(current_group_average >=bottom_margin and current_group_average <= top_margin):
        current_group.append(hit)
        current_group_average = sum(current_group) / len(current_group)
        # continue with next hit
      else:
        # Δεν ανήκει στο group - Κλείσιμο και έλεγχος ομάδας
        (stacks,remainder) = divmod(len(current_group) * resolution,
palette_width)
        if remainder / palette_width >= accepted_width_pc:
          stacks += 1
        # Προσθήκη σε κάθε στοίβα του υπολογιζόμενου πλήθους παλετών από την μέση
τιμή της ομάδας
        palettes = round((flight_height - current_group_average) /
palette_height)
        for i in range(stacks):
          row_result.append(palettes)
        # Αρχικοποίση επόμενης ομάδας
        current_group = [hit]
        current_group_average = hit
```

```
    # Έλεγχος του τελευταίου group
    (stacks,remainder) = divmod(len(current_group) * resolution, palette_width)
    if remainder / palette_width >= accepted_width_pc:
      stacks += 1
    palettes = round((flight_height - current_group_average) / palette_height)
    for i in range(stacks):
      row_result.append(palettes)

    # Προσθήκη των αποτελεσμάτων της γραμμής στον τελικό πίνακα
    result.append(row_result)

  # Επιστροφή αποτελέσματος
  return result


# ----------------------------------------------------------------
# Model Grid
# ----------------------------------------------------------------
import numpy as np

class Grid:

  # Constructor
  # Δημιουργία ενός grid διαστάσεων width+1 X length+1 με μηδενικά ύψη
  def __init__(self, width, length):
    self.width = width+1
    self.length = length+1
    self.grid = np.zeros((self.width, self.length), np.uint16)

  # Convert grid to point cloud
  def to_ptCloud(self):
    ptCloud = np.zeros((self.width * self.length), np.uint16)
    i = 0
    for x in range(self.width):
      for y in range(self.length):
        ptCloud[i, 0] = x
        ptCloud[i, 1] = y
        ptCloud[i, 2] = self.grid[x,y]
        i += 1
    return ptCloud

  # Load grid from point cloud
  # Dimensions should match
  def from_ptCloud(self, ptCloud):
    for point in ptCloud:
      self.grid[point[0], point[1]] = point[2]


# ----------------------------------------------------------------
# Model Point
# ----------------------------------------------------------------
class Point:

  # attributes:
  # x: number
  # y: number
  # z: number
```

```
  # color: string of hex notation, i.e. '#FFOODC'

  # ------------------------------------------------------------------
  # constructor
  # ------------------------------------------------------------------
  def __init__(self, x = 0, y = 0, z = 0, color='#000000'):
    self.x = x
    self.y = y
    self.z = z
    self.color = color

  # ------------------------------------------------------------------
  # coordinates to strinct of the form (x,y)
  # ------------------------------------------------------------------
  def coords_xy(self):
    return f"({self.x},{self.y})"


# ------------------------------------------------------------------
# Model PointCloud
# ------------------------------------------------------------------
import tempfile
import open3d as o3d
import numpy as np
import os
from lib.colors import hex2rgb, rgb2hex
from models.Point import Point
import matplotlib.pyplot as plt


class PointCloud:

  # attributes
  # points: list of Point

  # ------------------------------------------------------------------
  # constructor
  # ------------------------------------------------------------------
  def __init__(self):
    self.points = []

  # ------------------------------------------------------------------
  # Save to file
  # ------------------------------------------------------------------
  def to_file(self, filename):
    f = open(filename, 'w')
    for p in self.points:
      rgb = hex2rgb(p.color)
      f.write(f"{p.x} {p.y} {p.z} {rgb[0]} {rgb[1]} {rgb[2]}\n")
    f.close()

  # ------------------------------------------------------------------
  # Load from file (alternative constructor)
  # ------------------------------------------------------------------
  @classmethod
  def from_file(cls, filename) -> 'PointCloud':
    ptc = PointCloud()
```

```
    f = open(filename, 'r')
    lines = f.readlines()
    for line in lines:
      tokens = line.split(' ')
      hex_color = rgb2hex((tokens[3], tokens[4], tokens[5]))
      ptc.add_point(Point(tokens[0], tokens[1], tokens[2], hex_color))
    f.close()
    return ptc

# ---------------------------------------------------------------------
# Keep top point only for each x,y
# Return new point cloud
# ---------------------------------------------------------------------
def prune(self):
    pdict = dict()
    for point in self.points:
      if point.z == 0:  # skip ground points
        continue
      key = point.coords_xy()
      if (key not in pdict) or (pdict[key].z < point.z):
        pdict[key] = point
    pruned_point_cloud = PointCloud()
    for key in pdict:
      pruned_point_cloud.add_point(pdict[key])
    return pruned_point_cloud

# ---------------------------------------------------------------------
# Add a single point to the cloud
# ---------------------------------------------------------------------
def add_point(self, point):
    self.points.append(point)

# ---------------------------------------------------------------------
# Visualize point cloud
# ---------------------------------------------------------------------
def visualize(self):
    temp_filename = next(tempfile._get_candidate_names()) + '.xyz'
    self.to_file(temp_filename)
    xyz_file = o3d.io.read_point_cloud(temp_filename)
    colors = np.loadtxt(temp_filename, usecols=(3,4,5))
    xyz_file.colors = o3d.utility.Vector3dVector(colors)
    o3d.visualization.draw_geometries([xyz_file], 'Point Cloud')
    os.remove(temp_filename)
    # vis = o3d.visualization.Visualizer()
    # vis.create_window()
    # vis.add_geometry(xyz_file)
    # vis.update_geometry(xyz_file)
    # vis.poll_events()
    # vis.update_renderer()


# ---------------------------------------------------------------------
# Histogram of heights
# ---------------------------------------------------------------------
def z_hist(self, bins=10, plot=False):
    z = []
    for p in self.points:
```

```
      z.append(p.z)
    if plot:
      plt.hist(z, bins)
      plt.show()
    return np.histogram(z, bins)

  # ------------------------------------------------------------------
  # Convert points to numpy array
  # ------------------------------------------------------------------
  def to_numpy(self):
    np_points = np.zeros((len(self.points), 6), dtype=float)
    for i in range(len(self.points)):
      p = self.points[i]
      rgb = hex2rgb(p.color)
      np_points[i] = [p.x, p.y, p.z, rgb[0], rgb[1], rgb[2]]
    return np_points


# ------------------------------------------------------------------
# Model Stack
# ------------------------------------------------------------------
class Stack:

  # ------------------------------------------------------------------
  # constructor
  # ------------------------------------------------------------------
  def __init__(self, palettes, label, color):
    self.palettes = palettes
    self.label = label
    self.color = color  # to hex notation

# ------------------------------------------------------------------
# Model Warehouse
# ------------------------------------------------------------------
import numpy as np
import random
import cv2 as cv

from models.Stack import Stack
from models.PointCloud import PointCloud
from models.Point import Point
from lib.colors import PC_GROUND_COLOR, PC_STACK_COLOR, hex2bgr, hex2rgb


class Warehouse:

    # ------------------------------------------------------------------
    # constructor
    # ------------------------------------------------------------------
    def __init__(
            self,
            products=[('A', '#0000FF')],
            palette_width=50,
            palette_length=50,
            palette_height=30,
            row_groups=4,
            rows_per_group=2,
```

```
            columns=2,
            row_group_gap=20,
            column_gap=20,
            x_margin=20,
            y_margin=20,
            stacks_per_row=4,
            max_palettes_per_stack=5,
    ):

        # Store warehouse parameters
        self.products = products
        self.palette_width = palette_width
        self.palette_length = palette_length
        self.palette_height = palette_height
        self.row_groups = row_groups
        self.rows_per_group = rows_per_group
        self.columns = columns
        self.row_group_gap = row_group_gap
        self.column_gap = column_gap
        self.x_margin = x_margin
        self.y_margin = y_margin
        self.stacks_per_row = stacks_per_row
        self.max_palettes_per_stack = max_palettes_per_stack

        # Calculate warehouse width and length
        self.width = (palette_width * stacks_per_row *
                      columns) + ((columns - 1) * column_gap)
        self.length = (palette_length * row_groups *
                       rows_per_group) + ((row_groups - 1) * row_group_gap)
        self.width += 2 * self.x_margin
        self.length += 2 * self.y_margin

        # Calculate box width and length
        self.box_width = self.palette_width * self.stacks_per_row
        self.box_length = self.palette_length * self.rows_per_group

        # Create numpy array for stacks (row_group, row, col, index)
        self.stacks    =    np.empty((row_groups,    rows_per_group,    columns,
stacks_per_row), np.object)

        # create nparray for warehouse hights and products
        # For product keep index into self.products
        self.mat = np.zeros((self.length, self.width, 2), np.uint16)

        # Create an empty renderer
        self.renderer = None


    # ------------------------------------------------------------------
    # Clone warehouse
    # ------------------------------------------------------------------
    def clone(self):
        wh =  Warehouse(
            products=self.products,
            palette_width=self.palette_width,
            palette_length=self.palette_length,
            palette_height=self.palette_height,
```

```
            row_groups=self.row_groups,
            rows_per_group=self.rows_per_group,
            columns=self.columns,
            row_group_gap=self.row_group_gap,
            column_gap=self.column_gap,
            x_margin=self.x_margin,
            y_margin=self.y_margin,
            stacks_per_row=self.stacks_per_row,
            max_palettes_per_stack=self.max_palettes_per_stack,
        )

        # Create new stacks for self.stacks (deep copy)
        # set_stack_at() updates mat
        for rg in range(self.row_groups):
            for row in range(self.rows_per_group):
                for col in range(self.columns):
                    for index in range(self.stacks_per_row):
                        stack = self.stacks[rg, row, col, index]
                        new_stack = Stack(
                            palettes=stack.palettes,
                            label=stack.label,
                            color=stack.color
                        )
                        wh.set_stack_at(rg,row,col,index,new_stack)

        # Return cloned warehouse
        return wh

    # ------------------------------------------------------------------
    # Extract box as new warehouse
    # ------------------------------------------------------------------
    def extract_box_at(self, row_group, column, x_margin=20, y_margin=20):
        # Create new empty warehouse for single box
        box = Warehouse(
            palette_width=self.palette_width,
            palette_length=self.palette_length,
            palette_height=self.palette_height,
            row_groups=1,
            rows_per_group=self.rows_per_group,
            columns=1,
            row_group_gap=0,
            column_gap=0,
            x_margin=x_margin,
            y_margin=y_margin,
            stacks_per_row=self.stacks_per_row,
            max_palettes_per_stack=self.max_palettes_per_stack,
            products=self.products
        )

        # Copy stacks
        for row in range(self.rows_per_group):
            for index in range(self.stacks_per_row):
                stack = self.stacks[row_group, row, column, index]
                box.set_stack_at(0,row,0,index, stack)

        # return box as warehouse
        return box
```

```
    # ------------------------------------------------------------------
    # Sample all rows at the center
    # Return  a numpy array of z values
    # ------------------------------------------------------------------
    def  sample_rows(self,  flight_height=150,  flight_offset=0,  resolution=1,
error_margin=0.005):

        # total rows to be scanned
        rows = self.row_groups * self.rows_per_group

        # total values across row - take resolution into account
        hits_per_row = int((self.width - 2 * self.x_margin - 2 * flight_offset) /
resolution)

        # init hits array of z values
        hits = np.zeros((rows, hits_per_row))

        # loop for each row
        hits_row = 0
        for rg in range(self.row_groups):
            for row in range(self.rows_per_group):
                start_x = self.x_margin + flight_offset
                end_x = self.width - self.x_margin - flight_offset
                y = rg * self.box_length + row * self.palette_length + rg *
self.row_group_gap + self.y_margin + int(self.palette_length / 2)
                i = 0
                for x in range(start_x, end_x, resolution):
                    error = random.uniform(1-error_margin, 1+error_margin)
                    hits[hits_row, i] = (flight_height - self.mat[y, x, 0]) *
error
                    i += 1
                hits_row += 1

        # return results
        return hits

    # ------------------------------------------------------------------
    # Get x,y stack cordinates from rowgroup, row, col and index
    # ------------------------------------------------------------------
    def get_stack_coordinates(self, row_group, row, col, index):
         box_offset_x  =  col  *  self.box_width  +  col  *  self.column_gap  +
self.x_margin
         box_offset_y = row_group * self.box_length + row * self.palette_length +
row_group * self.row_group_gap + self.y_margin
         x = box_offset_x + index * self.palette_width
         y = box_offset_y
         return x, y

    # ------------------------------------------------------------------
    # Set a given stack at position (rg, row, col, index)
    # ------------------------------------------------------------------
    def set_stack_at(self, rg, row, col, index, stack):
        self.stacks[rg, row, col, index] = stack

        # Update mat
```

```
        x, y = self.get_stack_coordinates(rg,row,col,index)
        for i in range(x, x + self.palette_width):
          for j in range(y, y + self.palette_length):
              self.mat[j,i,0] = stack.palettes * self.palette_height
              product_index = 0
              for product in self.products:
                  if product[0] == stack.label:
                      break
                  product_index += 1
              self.mat[j,i,1] = product_index

    # ------------------------------------------------------------------
    # Create random stacks for all stack positions in warehouse
    # ------------------------------------------------------------------
    def create_ramdom_stacks(self, normalize=True):
        for rg in range(self.row_groups):
            for row in range(self.rows_per_group):
                for col in range(self.columns):
                    for index in range(self.stacks_per_row):
                        palettes = random.randint(0,self.max_palettes_per_stack)
                        product_index = random.randint(0,len(self.products)-1)
                        product = self.products[product_index]
                        new_stack = Stack(
                            palettes = palettes,
                            label = product[0],
                            color=product[1]
                        )
                        self.set_stack_at(rg,row,col,index,new_stack)

        # Normilize stacks
        if normalize:
            self._normalize_stacks()


    # ------------------------------------------------------------------
    # Normilize stacks
    # Inner stacks should have palettes >= min(above,bellow)
    # (THIS IMPLEMENTATION WORKS WITH MAX 3 ROWS PER GROUP)
    # ------------------------------------------------------------------
    def _normalize_stacks(self):
        for rg in range(self.row_groups):
            for row in range(self.rows_per_group):
                for col in range(self.columns):
                    for index in range(self.stacks_per_row):
                        stack = self.stacks[rg,row,col,index]

                        # Palettes above
                        if row == 0:
                            palettes_above = 0
                        else:
                            stack_above = self.stacks[rg,row-1,col,index]
                            palettes_above = stack_above.palettes

                        # palettes bellow
                        if row == self.rows_per_group - 1:
                            palettes_bellow = 0
                        else:
```

```
                            stack_bellow = self.stacks[rg,row+1,col,index]
                            palettes_bellow = stack_bellow.palettes

                        # normalize
                        min_palettes = min(palettes_above, palettes_bellow)
                        if stack.palettes < min_palettes:
                            stack.palettes     =     random.randint(min_palettes,
self.max_palettes_per_stack)
                            self.set_stack_at(rg,row,col,index,stack)


    # ------------------------------------------------------------------
    # Draw stacks into warehouse - dont' use mat - it's just for display only
    # return a cv mat
    # ------------------------------------------------------------------
    def      draw_stacks(self,      print_labels=False,      print_palettes=True,
print_colors=True):

        # Reset renderer
        self.renderer = np.zeros((self.length, self.width, 3), np.uint8)

        for rg in range(self.row_groups):
            for row in range(self.rows_per_group):
                for col in range(self.columns):
                    for index in range(self.stacks_per_row):
                        stack = self.stacks[rg,row,col,index]
                        x, y = self.get_stack_coordinates(rg,row,col,index)

                        # Draw filled rectangle if self.print_colors - use stack
color
                        # Don't fill stacks with 0 palettes
                        if print_colors and stack.palettes > 0:
                            cv.rectangle(self.renderer,      (x,y),      (x      +
self.palette_width, y + self.palette_length), hex2bgr(stack.color), -1)

                        # Draw outline - white
                        cv.rectangle(self.renderer,      (x,      y),      (x      +
self.palette_width, y + self.palette_length), (255, 255, 255), 1)

                        # Draw text at the middle
                        font = cv.FONT_HERSHEY_SIMPLEX
                        text = ""
                        if print_labels and print_palettes:
                            text = f"{stack.label}:{stack.palettes}"
                        elif print_labels:
                            text = f"{stack.label}"
                        else:
                            text = f"{stack.palettes}"
                        textsize = cv.getTextSize(text, font, 0.3, 1)[0]
                        text_x = x + (self.palette_width - textsize[0]) / 2
                        text_y = y + (self.palette_length + textsize[1]) / 2
                        cv.putText(self.renderer,       text,       (int(text_x),
int(text_y)),
                                   font, 0.3, (255, 255, 255), 1)


    # ------------------------------------------------------------------
```

```
    # Draw empty warehouse with stack places
    # ---------------------------------------------------------------
    def draw_empty(self):

        # Reset renderer
        self.renderer = np.zeros((self.length, self.width, 3), np.uint8)

        for rg in range(self.row_groups):
            for row in range(self.rows_per_group):
                for col in range(self.columns):
                    for index in range(self.stacks_per_row):
                        x, y = self.get_stack_coordinates(rg, row, col, index)
                        cv.rectangle(self.renderer,     (x,     y),     (x     +
self.palette_width, y + self.palette_length), (255, 255, 255), 1)

    # ---------------------------------------------------------------
    # Draws the flight line over a mat given (empty or with stacks)
    # ---------------------------------------------------------------
    def draw_flight(self, flight_offset=0):

        # if renderer is None create empty
        if self.renderer is None:
            self.draw_empty()

        # keep previous y positions and vertical line
        prev_y_offset = 0
        right_side = False

        for rg in range(self.row_groups):
            for row in range(self.rows_per_group):
                x_offset = self.x_margin + flight_offset
                y_offset = rg * self.box_length + row * self.palette_length + rg
* self.row_group_gap + self.y_margin + int(self.palette_length / 2)
                x1 = x_offset
                y1 = y_offset
                x2 = self.width - self.x_margin - flight_offset
                y2 = y_offset
                cv.line(self.renderer, (x1, y1), (x2, y2), (0, 0, 255), 1)

                # draw vertical line at the end if row > 0 and row is odd
                if prev_y_offset > 0:
                    vertical_y1 = prev_y_offset
                    vertical_y2 = y_offset
                    if right_side:
                        vertical_x1 = x2
                    else:
                        vertical_x1 = x1
                    vertical_x2 = vertical_x1
                    cv.line(self.renderer,      (vertical_x1,      vertical_y1),
(vertical_x2, vertical_y2), (0,0,255), 1)

                # Update previous y_offset and vertical side
                prev_y_offset = y_offset
                right_side = not right_side

    # ---------------------------------------------------------------
    # Visualize renderer
```

```
    # ---------------------------------------------------------------------
    def show(self, window_title='Warehouse'):
        cv.imshow(window_title, self.renderer)

    # ---------------------------------------------------------------------
    # Save renderer
    # ---------------------------------------------------------------------
    def save(self, filename='warehouse'):
        cv.imwrite(filename, self.renderer)


    # ---------------------------------------------------------------------
    # Convert to point cloud
    # ---------------------------------------------------------------------
    def to_point_cloud(self, ground=True, full=False, resolution=10):

        pcloud = PointCloud()

        # ground points
        if ground:
            for x in range(self.width):
                for y in range(self.length):
                    pcloud.add_point(Point(x, y, 0, PC_GROUND_COLOR))

        # stack points
        for rg in range(self.row_groups):
            for row in range(self.rows_per_group):
                for col in range(self.columns):
                    for index in range(self.stacks_per_row):
                        stack = self.stacks[rg,row,col,index]
                        x, y = self.get_stack_coordinates(rg, row, col, index)
                        for sx in range(x, x + self.palette_width):
                            for sy in range(y, y + self.palette_length):
                                # set origin to top-left corner to match cv draw
(default is bottom-left)
                                sy = self.length - sy
                                if full:
                                    # Draw all points between 1 and stack height
                                    stack_height     =     stack.palettes     *
self.palette_height
                                    for   sh   in   range(1,   stack_height   +   1,
resolution):
                                        pcloud.add_point(Point(sx,    sy,    sh,
stack.color))
                                else:
                                    pcloud.add_point(Point(sx, sy, stack_height,
stack.color))

        # Return
        return pcloud


    # ---------------------------------------------------------------------
    # Scan warehouse along x axis. Start at (x,y)
    # Return tupples (heigth, product)
    # ---------------------------------------------------------------------
    def scan_x(self, x, y, resolution=1):
```

```
        points = []
        for dx in range(x, self.width, resolution):
            label = '-'
            height = self.mat[y,dx,0]
            if height > 0:
                product = self.products[self.mat[y,dx,1]]
                label = product[0]
            points.append((height, label))
        return points


    # ------------------------------------------------------------------
    # Add noise
    # Alter each point into mat if random float is bellow percentage
    # max_error is the maximum alternation.
    # Get a random direction - up or down
    # ------------------------------------------------------------------
    def add_noise(self, percentage, max_error):
        for i in range(self.width):
            for j in range(self.length):
                height = self.mat[j,i,0]
                if height == 0:
                    continue
                alter = random.random() < percentage
                if not alter:
                    continue
                # alter pixel
                noise = random.randint(0,max_error)
                direction = random.randint(0,1)
                if direction == 0:
                    self.mat[j,i,0] = height - noise
                else:
                    self.mat[j,i,0] = height + noise


    # ------------------------------------------------------------------
    # Remove noise from mat
    # 1. Divide with palette hight
    # 2. If remainder is less than half the divider take quotient
    # 3. else take quotient+1
    # ------------------------------------------------------------------
    def remove_noise(self):
        for i in range(self.width):
            for j in range(self.length):
                height = self.mat[j,i,0]
                if height == 0:
                    continue
                if height % self.palette_height < self.palette_height / 2:
                    self.mat[j,i,0]   =   (height   //   self.palette_height)   *
self.palette_height
                else:
                    self.mat[j,i,0] = ((height // self.palette_height) + 1) *
self.palette_height


# ------------------------------------------------------------------
# Καταμέτρηση ενός box με περιθώρια
```

```
# -----------------------------------------------------------------
import cv2 as cv
from functions.scan_warehouse import scan_warehouse
from models.Stack import Stack
import matplotlib.pyplot as plt

from models.Warehouse import Warehouse

#  Create warehouse
box = Warehouse(
    palette_width=40,
    palette_length=40,
    palette_height=20,
    row_groups=1,
    rows_per_group=2,
    columns=1,
    row_group_gap=60,
    column_gap=60,
    x_margin=20,
    y_margin=5,
    stacks_per_row=8,
    max_palettes_per_stack=5,
    # products=[('A', '#0000FF'), ('B', '#FF0000'), ('C', '#DD44DD')],
    products=[('A', '#0000FF')],
)

# box.create_ramdom_stacks()
box.set_stack_at(0,0,0,0,Stack(3, 'A', '#0000FF'))
box.set_stack_at(0,0,0,1,Stack(5, 'A', '#0000FF'))
box.set_stack_at(0,0,0,2,Stack(0, 'A', '#0000FF'))
box.set_stack_at(0,0,0,3,Stack(5, 'A', '#0000FF'))
box.set_stack_at(0,0,0,4,Stack(1, 'A', '#0000FF'))
box.set_stack_at(0,0,0,5,Stack(4, 'A', '#0000FF'))
box.set_stack_at(0,0,0,6,Stack(4, 'A', '#0000FF'))
box.set_stack_at(0,0,0,7,Stack(2, 'A', '#0000FF'))

box.set_stack_at(0,1,0,0,Stack(1, 'A', '#0000FF'))
box.set_stack_at(0,1,0,1,Stack(3, 'A', '#0000FF'))
box.set_stack_at(0,1,0,2,Stack(5, 'A', '#0000FF'))
box.set_stack_at(0,1,0,3,Stack(4, 'A', '#0000FF'))
box.set_stack_at(0,1,0,4,Stack(0, 'A', '#0000FF'))
box.set_stack_at(0,1,0,5,Stack(4, 'A', '#0000FF'))
box.set_stack_at(0,1,0,6,Stack(3, 'A', '#0000FF'))
box.set_stack_at(0,1,0,7,Stack(5, 'A', '#0000FF'))

# Save empty box
flight_offset = -10
box.draw_empty()
box.show('empty')
# box.draw_stacks()
# box.draw_flight(flight_offset=flight_offset)
# box.save('d:/with_margin/empty_box.png')
box.draw_flight(flight_offset=flight_offset)
box.show('flight')
# box.save('d:/with_margin/empty_box_flight.png')

# Get hits
```

```
flight_height = 120
resolution = 1
hits = box.sample_rows(flight_height=flight_height, flight_offset=flight_offset,
resolution=resolution)

# plot box rows
plt.figure('Box Row 1')
plt.title("Box Row 1")
plt.xlabel("Hits")
plt.ylabel("Lidar measurment")
plt.plot(hits[0,:])
plt.show(block=False)
# plt.savefig('d:/with_margin/box_row1.png')

plt.figure('Box Row 2')
plt.title("Box Row 2")
plt.xlabel("Hits")
plt.ylabel("Lidar measurment")
plt.plot(hits[1,:])
plt.show(block=False)
# plt.savefig('d:/with_margin/box_row2.png')

# draw stacks and save
box.draw_stacks()
box.show('stacks')
# box.save('d:/with_margin/box_stacks.png')

result = scan_warehouse(
  hits=hits,
  resolution=resolution,
  flight_height=flight_height,
  palette_width=box.palette_width,
  palette_height=box.palette_height,
  error_margin=0.008,
  accepted_width_pc=0.9
)
print(result)

input()


# ----------------------------------------------------------------
# Καταμέτρηση ενός box χωρίς περιθώρια
# ----------------------------------------------------------------
import cv2 as cv
from functions.scan_warehouse import scan_warehouse
from models.Stack import Stack
import matplotlib.pyplot as plt

from models.Warehouse import Warehouse

#  Create warehouse
box = Warehouse(
    palette_width=40,
    palette_length=40,
    palette_height=20,
    row_groups=1,
```

```
    rows_per_group=2,
    columns=1,
    row_group_gap=60,
    column_gap=60,
    x_margin=5,
    y_margin=5,
    stacks_per_row=8,
    max_palettes_per_stack=5,
    # products=[('A', '#0000FF'), ('B', '#FF0000'), ('C', '#DD44DD')],
    products=[('A', '#0000FF')],
)

# box.create_ramdom_stacks()
box.set_stack_at(0,0,0,0,Stack(3, 'A', '#0000FF'))
box.set_stack_at(0,0,0,1,Stack(5, 'A', '#0000FF'))
box.set_stack_at(0,0,0,2,Stack(0, 'A', '#0000FF'))
box.set_stack_at(0,0,0,3,Stack(5, 'A', '#0000FF'))
box.set_stack_at(0,0,0,4,Stack(1, 'A', '#0000FF'))
box.set_stack_at(0,0,0,5,Stack(4, 'A', '#0000FF'))
box.set_stack_at(0,0,0,6,Stack(4, 'A', '#0000FF'))
box.set_stack_at(0,0,0,7,Stack(2, 'A', '#0000FF'))

box.set_stack_at(0,1,0,0,Stack(1, 'A', '#0000FF'))
box.set_stack_at(0,1,0,1,Stack(3, 'A', '#0000FF'))
box.set_stack_at(0,1,0,2,Stack(5, 'A', '#0000FF'))
box.set_stack_at(0,1,0,3,Stack(4, 'A', '#0000FF'))
box.set_stack_at(0,1,0,4,Stack(0, 'A', '#0000FF'))
box.set_stack_at(0,1,0,5,Stack(4, 'A', '#0000FF'))
box.set_stack_at(0,1,0,6,Stack(3, 'A', '#0000FF'))
box.set_stack_at(0,1,0,7,Stack(5, 'A', '#0000FF'))

# Save empty box
flight_offset = 0
box.draw_empty()
# box.draw_stacks()
# box.draw_flight(flight_offset=flight_offset)
box.save('d:/without_margin/empty_box.png')
box.draw_flight(flight_offset=flight_offset)
box.save('d:/without_margin/empty_box_flight.png')

# Get hits
flight_height = 120
resolution = 1
hits = box.sample_rows(flight_height=flight_height, flight_offset=flight_offset,
resolution=resolution)

# plot box rows
plt.figure('Box Row 1')
plt.title("Box Row 1")
plt.xlabel("Hits")
plt.ylabel("Lidar measurment")
plt.plot(hits[0,:])
plt.savefig('d:/without_margin/box_row1.png')

plt.figure('Box Row 2')
plt.title("Box Row 2")
plt.xlabel("Hits")
```

```
plt.ylabel("Lidar measurment")
plt.plot(hits[1,:])
plt.savefig('d:/without_margin/box_row2.png')

# draw stacks and save
box.draw_stacks()
box.save('d:/without_margin/box_stacks.png')


# ----------------------------------------------------------------
# Καταμέρτηση αποθήκης με περιθώρια
# ----------------------------------------------------------------
import cv2 as cv
from functions.scan_warehouse import scan_warehouse
from models.Stack import Stack
import matplotlib.pyplot as plt

from models.Warehouse import Warehouse

#  Create warehouse
wh = Warehouse(
    palette_width=40,
    palette_length=40,
    palette_height=20,
    row_groups=2,
    rows_per_group=2,
    columns=2,
    row_group_gap=60,
    column_gap=60,
    x_margin=20,
    y_margin=60,
    stacks_per_row=8,
    max_palettes_per_stack=5,
    # products=[('A', '#0000FF'), ('B', '#FF0000'), ('C', '#DD44DD')],
    products=[('A', '#0000FF')],
)

# wh.create_ramdom_stacks()

# 0,0,0
wh.set_stack_at(0,0,0,0,Stack(4, 'A', '#0000FF'))
wh.set_stack_at(0,0,0,1,Stack(2, 'A', '#0000FF'))
wh.set_stack_at(0,0,0,2,Stack(1, 'A', '#0000FF'))
wh.set_stack_at(0,0,0,3,Stack(1, 'A', '#0000FF'))
wh.set_stack_at(0,0,0,4,Stack(4, 'A', '#0000FF'))
wh.set_stack_at(0,0,0,5,Stack(4, 'A', '#0000FF'))
wh.set_stack_at(0,0,0,6,Stack(3, 'A', '#0000FF'))
wh.set_stack_at(0,0,0,7,Stack(3, 'A', '#0000FF'))

# 0,1,0
wh.set_stack_at(0,1,0,0,Stack(5, 'A', '#0000FF'))
wh.set_stack_at(0,1,0,1,Stack(1, 'A', '#0000FF'))
wh.set_stack_at(0,1,0,2,Stack(2, 'A', '#0000FF'))
wh.set_stack_at(0,1,0,3,Stack(0, 'A', '#0000FF'))
wh.set_stack_at(0,1,0,4,Stack(5, 'A', '#0000FF'))
wh.set_stack_at(0,1,0,5,Stack(0, 'A', '#0000FF'))
wh.set_stack_at(0,1,0,6,Stack(0, 'A', '#0000FF'))
```

```
wh.set_stack_at(0,1,0,7,Stack(1, 'A', '#0000FF'))

# 0,0,1
wh.set_stack_at(0,0,1,0,Stack(2, 'A', '#0000FF'))
wh.set_stack_at(0,0,1,1,Stack(2, 'A', '#0000FF'))
wh.set_stack_at(0,0,1,2,Stack(5, 'A', '#0000FF'))
wh.set_stack_at(0,0,1,3,Stack(1, 'A', '#0000FF'))
wh.set_stack_at(0,0,1,4,Stack(1, 'A', '#0000FF'))
wh.set_stack_at(0,0,1,5,Stack(4, 'A', '#0000FF'))
wh.set_stack_at(0,0,1,6,Stack(1, 'A', '#0000FF'))
wh.set_stack_at(0,0,1,7,Stack(3, 'A', '#0000FF'))

# 0,1,1
wh.set_stack_at(0,1,1,0,Stack(4, 'A', '#0000FF'))
wh.set_stack_at(0,1,1,1,Stack(0, 'A', '#0000FF'))
wh.set_stack_at(0,1,1,2,Stack(0, 'A', '#0000FF'))
wh.set_stack_at(0,1,1,3,Stack(3, 'A', '#0000FF'))
wh.set_stack_at(0,1,1,4,Stack(4, 'A', '#0000FF'))
wh.set_stack_at(0,1,1,5,Stack(0, 'A', '#0000FF'))
wh.set_stack_at(0,1,1,6,Stack(5, 'A', '#0000FF'))
wh.set_stack_at(0,1,1,7,Stack(4, 'A', '#0000FF'))

# 1,0,0
wh.set_stack_at(1,0,0,0,Stack(3, 'A', '#0000FF'))
wh.set_stack_at(1,0,0,1,Stack(1, 'A', '#0000FF'))
wh.set_stack_at(1,0,0,2,Stack(4, 'A', '#0000FF'))
wh.set_stack_at(1,0,0,3,Stack(1, 'A', '#0000FF'))
wh.set_stack_at(1,0,0,4,Stack(5, 'A', '#0000FF'))
wh.set_stack_at(1,0,0,5,Stack(4, 'A', '#0000FF'))
wh.set_stack_at(1,0,0,6,Stack(0, 'A', '#0000FF'))
wh.set_stack_at(1,0,0,7,Stack(1, 'A', '#0000FF'))

# 1,1,0
wh.set_stack_at(1,1,0,0,Stack(1, 'A', '#0000FF'))
wh.set_stack_at(1,1,0,1,Stack(4, 'A', '#0000FF'))
wh.set_stack_at(1,1,0,2,Stack(5, 'A', '#0000FF'))
wh.set_stack_at(1,1,0,3,Stack(3, 'A', '#0000FF'))
wh.set_stack_at(1,1,0,4,Stack(0, 'A', '#0000FF'))
wh.set_stack_at(1,1,0,5,Stack(1, 'A', '#0000FF'))
wh.set_stack_at(1,1,0,6,Stack(5, 'A', '#0000FF'))
wh.set_stack_at(1,1,0,7,Stack(4, 'A', '#0000FF'))

# 1,0,1
wh.set_stack_at(1,0,1,0,Stack(5, 'A', '#0000FF'))
wh.set_stack_at(1,0,1,1,Stack(0, 'A', '#0000FF'))
wh.set_stack_at(1,0,1,2,Stack(4, 'A', '#0000FF'))
wh.set_stack_at(1,0,1,3,Stack(4, 'A', '#0000FF'))
wh.set_stack_at(1,0,1,4,Stack(2, 'A', '#0000FF'))
wh.set_stack_at(1,0,1,5,Stack(1, 'A', '#0000FF'))
wh.set_stack_at(1,0,1,6,Stack(0, 'A', '#0000FF'))
wh.set_stack_at(1,0,1,7,Stack(3, 'A', '#0000FF'))

# 1,1,1
wh.set_stack_at(1,1,1,0,Stack(4, 'A', '#0000FF'))
wh.set_stack_at(1,1,1,1,Stack(5, 'A', '#0000FF'))
wh.set_stack_at(1,1,1,2,Stack(3, 'A', '#0000FF'))
wh.set_stack_at(1,1,1,3,Stack(2, 'A', '#0000FF'))
```

```
wh.set_stack_at(1,1,1,4,Stack(1, 'A', '#0000FF'))
wh.set_stack_at(1,1,1,5,Stack(3, 'A', '#0000FF'))
wh.set_stack_at(1,1,1,6,Stack(1, 'A', '#0000FF'))
wh.set_stack_at(1,1,1,7,Stack(4, 'A', '#0000FF'))

# Save empty wh
flight_offset = -10
wh.draw_empty()
wh.show("Empty")
# wh.draw_stacks()
# wh.draw_flight(flight_offset=flight_offset)
# wh.save('d:/with_margin/empty_warehouse.png')
wh.draw_flight(flight_offset=flight_offset)
wh.show('flight')
# wh.save('d:/with_margin/empty_warehouse_flight.png')

# Get hits
flight_height = 120
resolution = 1
hits = wh.sample_rows(flight_height=flight_height, flight_offset=flight_offset,
resolution=resolution)

# plot wh rows
plt.figure('Warehouse - Row 1')
plt.title("Warehouse - Row 1")
plt.xlabel("Hits")
plt.ylabel("Lidar measurment")
plt.plot(hits[0,:])
plt.show(block=False)
# plt.savefig('d:/with_margin/warehouse_row1.png')

plt.figure('Warehouse - Row 2')
plt.title("Warehouse - Row 2")
plt.xlabel("Hits")
plt.ylabel("Lidar measurment")
plt.plot(hits[1,:])
plt.show(block=False)
# plt.savefig('d:/with_margin/warehouse_row2.png')

plt.figure('Warehouse - Row 3')
plt.title("Warehouse - Row 3")
plt.xlabel("Hits")
plt.ylabel("Lidar measurment")
plt.plot(hits[2,:])
plt.show(block=False)
# plt.savefig('d:/with_margin/warehouse_row3.png')

plt.figure('Warehouse - Row 4')
plt.title("Warehouse - Row 4")
plt.xlabel("Hits")
plt.ylabel("Lidar measurment")
plt.plot(hits[3,:])
plt.show(block=False)
# plt.savefig('d:/with_margin/warehouse_row4.png')

# draw stacks and save
wh.draw_stacks()
```

```
wh.show("Stacks")
# wh.save('d:/with_margin/warehouse_stacks.png')

result = scan_warehouse(
  hits=hits,
  resolution=resolution,
  flight_height=flight_height,
  palette_width=wh.palette_width,
  palette_height=wh.palette_height,
  error_margin=0.008,
  accepted_width_pc=0.9
)
print(result)

input()


# ----------------------------------------------------------------------
# Καταμέτρηση αποθήκης χωρίς περιθώρια
# ----------------------------------------------------------------------
import cv2 as cv
from functions.scan_warehouse import scan_warehouse
from models.Stack import Stack
import matplotlib.pyplot as plt

from models.Warehouse import Warehouse

#  Create warehouse
wh = Warehouse(
    palette_width=40,
    palette_length=40,
    palette_height=20,
    row_groups=2,
    rows_per_group=2,
    columns=2,
    row_group_gap=60,
    column_gap=60,
    x_margin=5,
    y_margin=60,
    stacks_per_row=8,
    max_palettes_per_stack=5,
    # products=[('A', '#0000FF'), ('B', '#FF0000'), ('C', '#DD44DD')],
    products=[('A', '#0000FF')],
)

# wh.create_ramdom_stacks()

# 0,0,0
wh.set_stack_at(0,0,0,0,Stack(4, 'A', '#0000FF'))
wh.set_stack_at(0,0,0,1,Stack(2, 'A', '#0000FF'))
wh.set_stack_at(0,0,0,2,Stack(1, 'A', '#0000FF'))
wh.set_stack_at(0,0,0,3,Stack(1, 'A', '#0000FF'))
wh.set_stack_at(0,0,0,4,Stack(4, 'A', '#0000FF'))
wh.set_stack_at(0,0,0,5,Stack(4, 'A', '#0000FF'))
wh.set_stack_at(0,0,0,6,Stack(3, 'A', '#0000FF'))
wh.set_stack_at(0,0,0,7,Stack(3, 'A', '#0000FF'))
```

```
# 0,1,0
wh.set_stack_at(0,1,0,0,Stack(5, 'A', '#0000FF'))
wh.set_stack_at(0,1,0,1,Stack(1, 'A', '#0000FF'))
wh.set_stack_at(0,1,0,2,Stack(2, 'A', '#0000FF'))
wh.set_stack_at(0,1,0,3,Stack(0, 'A', '#0000FF'))
wh.set_stack_at(0,1,0,4,Stack(5, 'A', '#0000FF'))
wh.set_stack_at(0,1,0,5,Stack(0, 'A', '#0000FF'))
wh.set_stack_at(0,1,0,6,Stack(0, 'A', '#0000FF'))
wh.set_stack_at(0,1,0,7,Stack(1, 'A', '#0000FF'))

# 0,0,1
wh.set_stack_at(0,0,1,0,Stack(2, 'A', '#0000FF'))
wh.set_stack_at(0,0,1,1,Stack(2, 'A', '#0000FF'))
wh.set_stack_at(0,0,1,2,Stack(5, 'A', '#0000FF'))
wh.set_stack_at(0,0,1,3,Stack(1, 'A', '#0000FF'))
wh.set_stack_at(0,0,1,4,Stack(1, 'A', '#0000FF'))
wh.set_stack_at(0,0,1,5,Stack(4, 'A', '#0000FF'))
wh.set_stack_at(0,0,1,6,Stack(1, 'A', '#0000FF'))
wh.set_stack_at(0,0,1,7,Stack(3, 'A', '#0000FF'))

# 0,1,1
wh.set_stack_at(0,1,1,0,Stack(4, 'A', '#0000FF'))
wh.set_stack_at(0,1,1,1,Stack(0, 'A', '#0000FF'))
wh.set_stack_at(0,1,1,2,Stack(0, 'A', '#0000FF'))
wh.set_stack_at(0,1,1,3,Stack(3, 'A', '#0000FF'))
wh.set_stack_at(0,1,1,4,Stack(4, 'A', '#0000FF'))
wh.set_stack_at(0,1,1,5,Stack(0, 'A', '#0000FF'))
wh.set_stack_at(0,1,1,6,Stack(5, 'A', '#0000FF'))
wh.set_stack_at(0,1,1,7,Stack(4, 'A', '#0000FF'))

# 1,0,0
wh.set_stack_at(1,0,0,0,Stack(3, 'A', '#0000FF'))
wh.set_stack_at(1,0,0,1,Stack(1, 'A', '#0000FF'))
wh.set_stack_at(1,0,0,2,Stack(4, 'A', '#0000FF'))
wh.set_stack_at(1,0,0,3,Stack(1, 'A', '#0000FF'))
wh.set_stack_at(1,0,0,4,Stack(5, 'A', '#0000FF'))
wh.set_stack_at(1,0,0,5,Stack(4, 'A', '#0000FF'))
wh.set_stack_at(1,0,0,6,Stack(0, 'A', '#0000FF'))
wh.set_stack_at(1,0,0,7,Stack(1, 'A', '#0000FF'))

# 1,1,0
wh.set_stack_at(1,1,0,0,Stack(1, 'A', '#0000FF'))
wh.set_stack_at(1,1,0,1,Stack(4, 'A', '#0000FF'))
wh.set_stack_at(1,1,0,2,Stack(5, 'A', '#0000FF'))
wh.set_stack_at(1,1,0,3,Stack(3, 'A', '#0000FF'))
wh.set_stack_at(1,1,0,4,Stack(0, 'A', '#0000FF'))
wh.set_stack_at(1,1,0,5,Stack(1, 'A', '#0000FF'))
wh.set_stack_at(1,1,0,6,Stack(5, 'A', '#0000FF'))
wh.set_stack_at(1,1,0,7,Stack(4, 'A', '#0000FF'))

# 1,0,1
wh.set_stack_at(1,0,1,0,Stack(5, 'A', '#0000FF'))
wh.set_stack_at(1,0,1,1,Stack(0, 'A', '#0000FF'))
wh.set_stack_at(1,0,1,2,Stack(4, 'A', '#0000FF'))
wh.set_stack_at(1,0,1,3,Stack(4, 'A', '#0000FF'))
wh.set_stack_at(1,0,1,4,Stack(2, 'A', '#0000FF'))
wh.set_stack_at(1,0,1,5,Stack(1, 'A', '#0000FF'))
```

```
wh.set_stack_at(1,0,1,6,Stack(0, 'A', '#0000FF'))
wh.set_stack_at(1,0,1,7,Stack(3, 'A', '#0000FF'))

# 1,1,1
wh.set_stack_at(1,1,1,0,Stack(4, 'A', '#0000FF'))
wh.set_stack_at(1,1,1,1,Stack(5, 'A', '#0000FF'))
wh.set_stack_at(1,1,1,2,Stack(3, 'A', '#0000FF'))
wh.set_stack_at(1,1,1,3,Stack(2, 'A', '#0000FF'))
wh.set_stack_at(1,1,1,4,Stack(1, 'A', '#0000FF'))
wh.set_stack_at(1,1,1,5,Stack(3, 'A', '#0000FF'))
wh.set_stack_at(1,1,1,6,Stack(1, 'A', '#0000FF'))
wh.set_stack_at(1,1,1,7,Stack(4, 'A', '#0000FF'))

# Save empty wh
flight_offset = 0
wh.draw_empty()
wh.show('Empty')
# wh.draw_stacks()
# wh.draw_flight(flight_offset=flight_offset)
# wh.save('d:/without_margin/empty_warehouse.png')
wh.draw_flight(flight_offset=flight_offset)
wh.show('flight')
# wh.save('d:/without_margin/empty_warehouse_flight.png')

# Get hits
flight_height = 120
resolution = 1
hits = wh.sample_rows(flight_height=flight_height, flight_offset=flight_offset,
resolution=resolution)

# plot wh rows
plt.figure('Warehouse - Row 1')
plt.title("Warehouse - Row 1")
plt.xlabel("Hits")
plt.ylabel("Lidar measurment")
plt.plot(hits[0,:])
plt.show(block=False)
# plt.savefig('d:/without_margin/warehouse_row1.png')

plt.figure('Warehouse - Row 2')
plt.title("Warehouse - Row 2")
plt.xlabel("Hits")
plt.ylabel("Lidar measurment")
plt.plot(hits[1,:])
plt.show(block=False)
# plt.savefig('d:/without_margin/warehouse_row2.png')

plt.figure('Warehouse - Row 3')
plt.title("Warehouse - Row 3")
plt.xlabel("Hits")
plt.ylabel("Lidar measurment")
plt.plot(hits[2,:])
plt.show(block=False)
# plt.savefig('d:/without_margin/warehouse_row3.png')

plt.figure('Warehouse - Row 4')
plt.title("Warehouse - Row 4")
```

```
plt.xlabel("Hits")
plt.ylabel("Lidar measurment")
plt.plot(hits[3,:])
plt.show(block=False)
# plt.savefig('d:/without_margin/warehouse_row4.png')

# draw stacks and save
wh.draw_stacks()
wh.show("Stacks")
# wh.save('d:/without_margin/warehouse_stacks.png')

result = scan_warehouse(
  hits=hits,
  resolution=resolution,
  flight_height=flight_height,
  palette_width=wh.palette_width,
  palette_height=wh.palette_height,
  error_margin=0.008,
  accepted_width_pc=0.9
)
print(result)

input()
```

4. Ανάλυση Κώδικα Python

Εισαγωγή

Ο κώδικας αυτός υλοποιεί μια μέθοδο καταγραφής και καταμέτρησης παλετών σε αποθήκη, χρησιμοποιώντας δεδομένα από αισθητήρες **LiDAR** και επεξεργασία μεθόδων υπολογιστικής όρασης. Ο σκοπός είναι η δημιουργία ενός αυτοματοποιημένου συστήματος που μπορεί να σαρώσει την αποθήκη και να παρέχει μια εκτίμηση του αριθμού των παλετών μέσω επεξεργασίας σημειακών νεφών και ανάλυσης ύψους.

Σάρωση Αποθήκης και Υπολογισμός Παλετών

Συνάρτηση `scan_box`

Η συνάρτηση αυτή εκτελεί ανάλυση μιας μικρής περιοχής (box) της αποθήκης.

- Υπολογίζει το ύψος κάθε σημείου σε σχέση με το **ύψος πτήσης** του UAV.
- Μετατρέπει το ύψος σε αριθμό παλετών.
- Ομαδοποιεί τα δεδομένα για την καταμέτρηση των παλετών ανά στοίβα.

Συνάρτηση `scan_warehouse`

Αυτή η συνάρτηση εφαρμόζει την καταμέτρηση σε όλη την αποθήκη.

- Διαχειρίζεται ομάδες δεδομένων που έχουν παρόμοιο ύψος, προσπαθώντας να μειώσει τα σφάλματα.
- Υπολογίζει τον αριθμό των παλετών χρησιμοποιώντας **ποσοστό αποδοχής (accepted_width_pc)** και **όριο σφάλματος (error_margin)**.
- Ομαδοποιεί τα δεδομένα σε γραμμές και αναγνωρίζει περιοχές με στοίβες παλετών.

Μοντελοποίηση Αποθήκης και Ανάλυση Δεδομένων

Κλάση Grid

Αντιπροσωπεύει έναν πλέγμα 2D για την αποθήκη, αποθηκεύοντας πληροφορίες σχετικά με το ύψος σε κάθε σημείο.

- Μπορεί να μετατραπεί σε **σημειακό νέφος (point cloud)** για οπτικοποίηση.
- Μπορεί να φορτώσει δεδομένα από ένα σημειακό νέφος.

Κλάση Point

Αντιπροσωπεύει ένα σημείο στο χώρο με **x, y, z συντεταγμένες** και μια χρωματική παράμετρο.

- Χρησιμοποιείται για να καταγράφει δεδομένα ύψους σε κάθε σημείο της αποθήκης.

Κλάση PointCloud

Αντιπροσωπεύει ένα σημειακό νέφος για την απεικόνιση των δεδομένων LiDAR.

- Επιτρέπει τη φόρτωση και αποθήκευση δεδομένων σε αρχεία.
- Παρέχει λειτουργίες **αφαίρεσης περιττών σημείων (prune)** και **οπτικοποίησης** μέσω open3d.

Μοντελοποίηση Αποθήκης και Παλετών

Κλάση Stack

Αντιπροσωπεύει μια στοίβα από παλέτες, περιλαμβάνοντας το ύψος, την ετικέτα και το χρώμα της.

Κλάση Warehouse

Αντιπροσωπεύει την αποθήκη ως σύνολο στοίβων παλετών.

- Χρησιμοποιείται για την **προσομοίωση αποθήκης** με στοίβες παλετών.
- Περιλαμβάνει λειτουργίες:
    - Δημιουργία τυχαίων παλετών.
    - **Κανονικοποίηση** δεδομένων για την αφαίρεση σφαλμάτων.
    - Οπτικοποίηση μέσω OpenCV.
    - Μετατροπή των δεδομένων σε σημειακό νέφος για αναλυτική επεξεργασία.

Υλοποίηση και Ανάλυση Δεδομένων

Καταγραφή Δεδομένων μέσω UAV

Η αποθήκη δημιουργείται με τη χρήση της κλάσης Warehouse και γίνεται προσομοίωση της κατανομής παλετών.

- Εκτελείται **σάρωση με LiDAR** μέσω της `sample_rows()`.
- Αποθηκεύονται τα αποτελέσματα και οπτικοποιούνται μέσω `matplotlib`.

Υπολογισμός και Καταμέτρηση Παλετών

Εκτελείται η scan_warehouse() για να εξάγει τον αριθμό παλετών σε κάθε περιοχή της αποθήκης.

- Αξιολογείται η ακρίβεια των αποτελεσμάτων μέσω σύγκρισης με τα πραγματικά δεδομένα της αποθήκης.

Σύνοψη

Ο κώδικας παρέχει:

- Μια **αυτόνομη μέθοδο** για την ανάλυση αποθήκης μέσω UAV και LiDAR.
- Τη δυνατότητα **δημιουργίας τρισδιάστατης αναπαράστασης** της αποθήκης.
- Βελτιστοποιημένο αλγόριθμο για **ακριβή εκτίμηση** των παλετών μέσω επεξεργασίας δεδομένων LiDAR.

5. Κατάλογος όρων και συντομογραφιών

ΣμηΕΑ	Συστήματα μη Επανδρωμένων Αεροσκαφών
ΔΙ.ΠΑ.Ε.	Διεθνές Πανεπιστήμιο της Ελλάδος
ΠΑ.Δ.Α.	Πανεπιστήμιο Δυτικής Αττικής
Δ.Ε.Π.	Διδακτικό Ερευνητικό Προσωπικό
Ε.ΔΙ.Π.	Εργαστηριακό Διδακτικό Προσωπικό
Α.Ε.Ι.	Ανώτατο Εκπαιδευτικό Ίδρυμα
Τ.Ε.Ι.	Τεχνολογικό Εκπαιδευτικό Ίδρυμα
Δ.Ο.Α.Τ.Α.Π.	Διεπιστημονικός Οργανισμός Αναγνώρισης Τίτλων Ακαδημαϊκών και πληροφόρησης