ДОДАТОК В

ПЕРЕЛІК ДОКУМЕНТІВ НА ОПТИЧНОМУ НОСІЇ

Ім’я файла	**Опис**
Пояснювальні документи	
Диплом_Шишацький.docx	Пояснювальна записка до магістерської роботи. Документ Word.
Диплом_Шишацький.pdf	Пояснювальна записка до магістерської роботи в форматі PDF
Програма	
Program.rar	Архів. Містить коди програм.
Презентація	
Презентація_ Шишацький.pptx	Презентація до магістерської роботи

ДОДАТОК Б

ВІДГУК

керівника економічного розділу

на кваліфікаційну роботу магістра

на тему:

«Розробка програмного забезпечення для управління дорожнім рухом за даними відеоспостереження»

студента групи 121м-19-1 Шишацького Олександра Олександровича

Керівник економічного розділу
доцент каф. ПЕП та ПУ, к.е.н. **Л. В. Касьяненко**

```
        kf, cost_matrix, tracks, detections, track_indices,
detection_indices,
        gated_cost=INFTY_COST, only_position=False):
    """Invalidate infeasible entries in cost matrix based on the
state
    distributions obtained by Kalman filtering.
    Parameters
    ----------
    kf : The Kalman filter.
    cost_matrix : ndarray
        The NxM dimensional cost matrix, where N is the number of
track indices
        and M is the number of detection indices, such that entry
(i, j) is the
        association cost between `tracks[track_indices[i]]` and
        `detections[detection_indices[j]]`.
    tracks : List[track.Track]
        A list of predicted tracks at the current time step.
    detections : List[detection.Detection]
        A list of detections at the current time step.
    track_indices : List[int]
        List of track indices that maps rows in `cost_matrix` to
tracks in
        `tracks` (see description above).
    detection_indices : List[int]
        List of detection indices that maps columns in
`cost_matrix` to
        detections in `detections` (see description above).
    gated_cost : Optional[float]
        Entries in the cost matrix corresponding to infeasible
associations are
        set this value. Defaults to a very large value.
    only_position : Optional[bool]
        If True, only the x, y position of the state distribution
is considered
        during gating. Defaults to False.
    Returns
    -------
    ndarray
        Returns the modified cost matrix.
    """
    gating_dim = 2 if only_position else 4
    gating_threshold = kalman_filter.chi2inv95[gating_dim]
    measurements = np.asarray(
        [detections[i].to_xyah() for i in detection_indices])
    for row, track_idx in enumerate(track_indices):
        track = tracks[track_idx]
        gating_distance = kf.gating_distance(
            track.mean, track.covariance, measurements,
only_position)
        cost_matrix[row, gating_distance > gating_threshold] =
gated_cost
    return cost_matrix
```

```
    tracks : List[track.Track]
        A list of predicted tracks at the current time step.
    detections : List[detection.Detection]
        A list of detections at the current time step.
    track_indices : Optional[List[int]]
        List of track indices that maps rows in `cost_matrix` to
tracks in
        `tracks` (see description above). Defaults to all tracks.
    detection_indices : Optional[List[int]]
        List of detection indices that maps columns in
`cost_matrix` to
        detections in `detections` (see description above).
Defaults to all
        detections.
    Returns
    -------
    (List[(int, int)], List[int], List[int])
        Returns a tuple with the following three entries:
        * A list of matched track and detection indices.
        * A list of unmatched track indices.
        * A list of unmatched detection indices.
    """
    if track_indices is None:
        track_indices = list(range(len(tracks)))
    if detection_indices is None:
        detection_indices = list(range(len(detections)))

    unmatched_detections = detection_indices
    matches = []
    for level in range(cascade_depth):
        if len(unmatched_detections) == 0:  # No detections left
            break

        track_indices_l = [
            k for k in track_indices
            if tracks[k].time_since_update == 1 + level
        ]
        if len(track_indices_l) == 0:  # Nothing to match at this
level
            continue

        matches_l, _, unmatched_detections = \
            min_cost_matching(
                distance_metric, max_distance, tracks, detections,
                track_indices_l, unmatched_detections)
        matches += matches_l
    unmatched_tracks = list(set(track_indices) - set(k for k, _ in
matches))
    return matches, unmatched_tracks, unmatched_detections


def gate_cost_matrix(
```

```
        detection_indices = np.arange(len(detections))

    if len(detection_indices) == 0 or len(track_indices) == 0:
        return [], track_indices, detection_indices  # Nothing to
match.

    cost_matrix = distance_metric(
        tracks, detections, track_indices, detection_indices)
    cost_matrix[cost_matrix > max_distance] = max_distance + 1e-5
    indices = linear_assignment(cost_matrix)

    matches, unmatched_tracks, unmatched_detections = [], [], []
    for col, detection_idx in enumerate(detection_indices):
        if col not in indices[:, 1]:
            unmatched_detections.append(detection_idx)
    for row, track_idx in enumerate(track_indices):
        if row not in indices[:, 0]:
            unmatched_tracks.append(track_idx)
    for row, col in indices:
        track_idx = track_indices[row]
        detection_idx = detection_indices[col]
        if cost_matrix[row, col] > max_distance:
            unmatched_tracks.append(track_idx)
            unmatched_detections.append(detection_idx)
        else:
            matches.append((track_idx, detection_idx))
    return matches, unmatched_tracks, unmatched_detections


def matching_cascade(
        distance_metric, max_distance, cascade_depth, tracks,
detections,
        track_indices=None, detection_indices=None):
    """Run matching cascade.
    Parameters
    ----------
    distance_metric : Callable[List[Track], List[Detection],
List[int], List[int]) -> ndarray
        The distance metric is given a list of tracks and
detections as well as
        a list of N track indices and M detection indices. The
metric should
        return the NxM dimensional cost matrix, where element (i,
j) is the
        association cost between the i-th track in the given track
indices and
        the j-th detection in the given detection indices.
    max_distance : float
        Gating threshold. Associations with cost larger than this
value are
        disregarded.
    cascade_depth: int
        The cascade depth, should be se to the maximum track age.
```

```
import numpy as np
from sklearn.utils.linear_assignment_ import linear_assignment
from . import kalman_filter


INFTY_COST = 1e+5


def min_cost_matching(
        distance_metric, max_distance, tracks, detections,
track_indices=None,
        detection_indices=None):
    """Solve linear assignment problem.
    Parameters
    ----------
    distance_metric : Callable[List[Track], List[Detection],
List[int], List[int]) -> ndarray
        The distance metric is given a list of tracks and
detections as well as
        a list of N track indices and M detection indices. The
metric should
        return the NxM dimensional cost matrix, where element (i,
j) is the
        association cost between the i-th track in the given track
indices and
        the j-th detection in the given detection_indices.
    max_distance : float
        Gating threshold. Associations with cost larger than this
value are
        disregarded.
    tracks : List[track.Track]
        A list of predicted tracks at the current time step.
    detections : List[detection.Detection]
        A list of detections at the current time step.
    track_indices : List[int]
        List of track indices that maps rows in `cost_matrix` to
tracks in
        `tracks` (see description above).
    detection_indices : List[int]
        List of detection indices that maps columns in
`cost_matrix` to
        detections in `detections` (see description above).
    Returns
    -------
    (List[(int, int)], List[int], List[int])
        Returns a tuple with the following three entries:
        * A list of matched track and detection indices.
        * A list of unmatched track indices.
        * A list of unmatched detection indices.
    """
    if track_indices is None:
        track_indices = np.arange(len(tracks))
    if detection_indices is None:
```

```
            assert self.model_image_size[1]%32 == 0, 'Multiples of
32 required'
            boxed_image = letterbox_image(image,
tuple(reversed(self.model_image_size)))
        else:
            new_image_size = (image.width - (image.width % 32),
                              image.height - (image.height % 32))
            boxed_image = letterbox_image(image, new_image_size)
        image_data = np.array(boxed_image, dtype='float32')

        # print(image_data.shape)
        image_data /= 255.
        image_data = np.expand_dims(image_data, 0)  # Add batch
dimension.

        out_boxes, out_scores, out_classes = self.sess.run(
            [self.boxes, self.scores, self.classes],
            feed_dict={
                self.yolo_model.input: image_data,
                self.input_image_shape: [image.size[1],
image.size[0]],
                K.learning_phase(): 0
            })
        return_boxes = []
        return_scores = []
        return_class_names = []
        for i, c in reversed(list(enumerate(out_classes))):
            predicted_class = self.class_names[c]
            if predicted_class != 'person':  # Modify to detect
other classes.
                continue
            box = out_boxes[i]
            score = out_scores[i]
            x = int(box[1])
            y = int(box[0])
            w = int(box[3] - box[1])
            h = int(box[2] - box[0])
            if x < 0:
                w = w + x
                x = 0
            if y < 0:
                h = h + y
                y = 0
            return_boxes.append([x, y, w, h])
            return_scores.append(score)
            return_class_names.append(predicted_class)

        return return_boxes, return_scores, return_class_names

    def close_session(self):
        self.sess.close()

from __future__ import absolute_import
```

```
    def _get_anchors(self):
        anchors_path = os.path.expanduser(self.anchors_path)
        with open(anchors_path) as f:
            anchors = f.readline()
            anchors = [float(x) for x in anchors.split(',')]
            anchors = np.array(anchors).reshape(-1, 2)
        return anchors

    def generate(self):
        model_path = os.path.expanduser(self.model_path)
        assert model_path.endswith('.h5'), 'Keras model or weights
must be a .h5 file.'

        self.yolo_model = load_model(model_path,
custom_objects={'Mish': Mish}, compile=False)

        print('{} model, anchors, and classes
loaded.'.format(model_path))

        # Generate colors for drawing bounding boxes.
        hsv_tuples = [(x / len(self.class_names), 1., 1.)
                      for x in range(len(self.class_names))]
        self.colors = list(map(lambda x: colorsys.hsv_to_rgb(*x),
hsv_tuples))
        self.colors = list(
            map(lambda x: (int(x[0] * 255), int(x[1] * 255),
int(x[2] * 255)),
                self.colors))
        np.random.seed(10101)  # Fixed seed for consistent colors
across runs.
        np.random.shuffle(self.colors)  # Shuffle colors to
decorrelate adjacent classes.
        np.random.seed(None)  # Reset seed to default.

        # Generate output tensor targets for filtered bounding
boxes.
        self.input_image_shape = K.placeholder(shape=(2, ))
        if self.gpu_num>=2:
            self.yolo_model = multi_gpu_model(self.yolo_model,
gpus=self.gpu_num)
        boxes, scores, classes = yolo_eval(self.yolo_model.output,
self.anchors,
                len(self.class_names), self.input_image_shape,
                score_threshold=self.score,
iou_threshold=self.iou)
        return boxes, scores, classes

    def detect_image(self, image):

        if self.is_fixed_size:
            assert self.model_image_size[0]%32 == 0, 'Multiples of
32 required'
```

```
    print(" ")
    print("[Finish]")
    end = time.time()

    if len(pts[track.track_id]) != None:
       print(args["input"][43:57]+": "+ str(count) + " " +
str(class_name) +' Found')

    else:
       print("[No Found]")
     #print("[INFO]: model_image_size = (960, 960)")
    video_capture.release()
    if writeVideo_flag:
        out.release()
        list_file.close()
    cv2.destroyAllWindows()

if __name__ == '__main__':
    main(YOLO())

import colorsys

import numpy as np
from keras import backend as K
from keras.models import load_model

from yolo4.model import yolo_eval, Mish
from yolo4.utils import letterbox_image
import os
from keras.utils import multi_gpu_model

class YOLO(object):
    def __init__(self):
        self.model_path = './model_data/yolo4_weight.h5'
        self.anchors_path = './model_data/yolo_anchors.txt'
        self.classes_path = './model_data/coco_classes.txt'
        self.gpu_num = 1
        self.score = 0.5
        self.iou = 0.5
        self.class_names = self._get_class()
        self.anchors = self._get_anchors()
        self.sess = K.get_session()
        self.model_image_size = (608, 608)  # fixed size or (None,
None)
        self.is_fixed_size = self.model_image_size != (None, None)
        self.boxes, self.scores, self.classes = self.generate()

    def _get_class(self):
        classes_path = os.path.expanduser(self.classes_path)
        with open(classes_path) as f:
            class_names = f.readlines()
        class_names = [c.strip() for c in class_names]
        return class_names
```

```
                cv2.putText(frame,
str(class_names[0]),(int(bbox[0]), int(bbox[1] -20)),0, 5e-3 *
150, (color),2)

            i += 1
            #bbox_center_point(x,y)
            center =
(int(((bbox[0])+(bbox[2]))/2),int(((bbox[1])+(bbox[3]))/2))
            #track_id[center]

            pts[track.track_id].append(center)

            thickness = 5
            #center point
            cv2.circle(frame,  (center), 1, color, thickness)

               # draw motion path
            for j in range(1, len(pts[track.track_id])):
                if pts[track.track_id][j - 1] is None or
pts[track.track_id][j] is None:
                    continue
                thickness = int(np.sqrt(64 / float(j + 1)) * 2)
                cv2.line(frame,(pts[track.track_id][j-1]),
(pts[track.track_id][j]),(color),thickness)
                #cv2.putText(frame,
str(class_names[j]),(int(bbox[0]), int(bbox[1] -20)),0, 5e-3 *
150, (255,255,255),2)

        count = len(set(counter))
        cv2.putText(frame, "Total Objects Counter:
"+str(count),(int(20), int(120)),0, 5e-3 * 200, (0,255,0),2)
        cv2.putText(frame, "Current Pedestrian Counter:
"+str(i),(int(20), int(80)),0, 5e-3 * 200, (0,255,0),2)
        cv2.putText(frame, "FPS: %f"%(fps),(int(20), int(40)),0,
5e-3 * 200, (0,255,0),3)
        cv2.namedWindow("YOLO4_Deep_SORT", 0);
        cv2.resizeWindow('YOLO4_Deep_SORT', 1024, 768);
        cv2.imshow('YOLO4_Deep_SORT', frame)


        if writeVideo_flag:
            # save a frame
            out.write(frame)
            frame_index = frame_index + 1


        fps  = ( fps + (1./(time.time()-t1)) ) / 2
        out.write(frame)
        frame_index = frame_index + 1

        # Press Q to stop!
        if cv2.waitKey(1) & 0xFF == ord('q'):
            break
```

```
        detections = [detections[i] for i in indices]

        # Call the tracker
        tracker.predict()
        tracker.update(detections)

        i = int(0)
        indexIDs = []
        c = []
        boxes = []

        for det in detections:
            bbox = det.to_tlbr()
            cv2.rectangle(frame,(int(bbox[0]), int(bbox[1])),
(int(bbox[2]), int(bbox[3])),(255,255,255), 2)
            #print(class_names)
            #print(class_names[p])

        for track in tracker.tracks:
            if not track.is_confirmed() or track.time_since_update
> 1:
                continue
            #boxes.append([track[0], track[1], track[2],
track[3]])
            indexIDs.append(int(track.track_id))
            counter.append(int(track.track_id))
            bbox = track.to_tlbr()
            color = [int(c) for c in COLORS[indexIDs[i] %
len(COLORS)]]
            #print(frame_index)
            list_file.write(str(frame_index)+',')
            list_file.write(str(track.track_id)+',')
            cv2.rectangle(frame, (int(bbox[0]), int(bbox[1])),
(int(bbox[2]), int(bbox[3])),(color), 3)
            b0 = str(bbox[0])#.split('.')[0] + '.' +
str(bbox[0]).split('.')[0][:1]
            b1 = str(bbox[1])#.split('.')[0] + '.' +
str(bbox[1]).split('.')[0][:1]
            b2 = str(bbox[2]-bbox[0])#.split('.')[0] + '.' +
str(bbox[3]).split('.')[0][:1]
            b3 = str(bbox[3]-bbox[1])

            list_file.write(str(b0) + ','+str(b1) + ','+str(b2) +
','+str(b3))
            #print(str(track.track_id))
            list_file.write('\n')
            #list_file.write(str(track.track_id)+',')
            cv2.putText(frame,str(track.track_id),(int(bbox[0]),
int(bbox[1] -50)),0, 5e-3 * 150, (color),2)
            if len(class_names) > 0:
               class_name = class_names[0]
```

```
def main(yolo):

    start = time.time()
    max_cosine_distance = 0.3
    nn_budget = None
    nms_max_overlap = 1.0

    counter = []
    #deep_sort
    model_filename = 'model_data/market2501.pb'
    encoder = gdet.create_box_encoder(model_filename,batch_size=1)

    find_objects = ['person']
    metric = nn_matching.NearestNeighborDistanceMetric("cosine",
max_cosine_distance, nn_budget)
    tracker = Tracker(metric)

    writeVideo_flag = True
    video_capture = cv2.VideoCapture(args["input"])


    if writeVideo_flag:
    # Define the codec and create VideoWriter object
        w = int(video_capture.get(3))
        h = int(video_capture.get(4))
        fourcc = cv2.VideoWriter_fourcc(*'MJPG')
        out = cv2.VideoWriter('./output/output.avi', fourcc, 15,
(w, h))
        list_file = open('detection_rslt.txt', 'w')
        frame_index = -1

    fps = 0.0

    while True:

        ret, frame = video_capture.read()  # frame shape 640*480*3
        if ret != True:
            break
        t1 = time.time()

        #image = Image.fromarray(frame)
        image = Image.fromarray(frame[...,::-1]) #bgr to rgb
        boxs, confidence, class_names = yolo.detect_image(image)
        features = encoder(frame,boxs)
        # score to 1.0 here).
        detections = [Detection(bbox, 1.0, feature) for bbox,
feature in zip(boxs, features)]
        # Run non-maxima suppression.
        boxes = np.array([d.tlwh for d in detections])
        scores = np.array([d.confidence for d in detections])
        indices = preprocessing.non_max_suppression(boxes,
nms_max_overlap, scores)
```

ДОДАТОК А

ЛИСТИНГ ПРОГРАМИ

```
#! /usr/bin/env python
# -*- coding: utf-8 -*-

from __future__ import division, print_function, absolute_import
import sys
#sys.path.remove('/opt/ros/kinetic/lib/python2.7/dist-packages')
import os
import datetime
from timeit import time
import warnings
import cv2
import numpy as np
import argparse
from PIL import Image
from yolo import YOLO

from deep_sort import preprocessing
from deep_sort import nn_matching
from deep_sort.detection import Detection
from deep_sort.tracker import Tracker
from tools import generate_detections as gdet
from deep_sort.detection import Detection as ddet
from collections import deque
from keras import backend
import tensorflow as tf
from tensorflow.compat.v1 import InteractiveSession
config = tf.ConfigProto()
config.gpu_options.allow_growth = True
session = InteractiveSession(config=config)


ap = argparse.ArgumentParser()
ap.add_argument("-i", "--input",help="path to input video",
default = "./test_video/Dnipro.avi")
ap.add_argument("-c", "--class",help="name of class", default =
"person")
args = vars(ap.parse_args())

pts = [deque(maxlen=30) for _ in range(1000)]
warnings.filterwarnings('ignore')

# initialize a list of colors to represent each possible class
label
np.random.seed(200)
COLORS = np.random.randint(0, 255, size=(200, 3),
     dtype="uint8")
#list = [[] for _ in range(100)]
```

63. Кнут Д. Искусство программирования для ЭВМ. Сортировка и поиск / Д. Кнут. – М.: Вильямс, 2007. – 824 с.
64. Кормен Т., Лейзерсон Ч., Ривест Р. Алгоритмы: построение и анализ. –М.: Вильямс, 2013. – 1328 с.
65. Крюков В. А., Удовиченко Р. В. Отладка DVMпрограмм / В. А. Крюков, Р. В. Удовиченко // Программирование. – 2001, No 3. – С.19–29.
66. Лаврищева Е. М. Методы программирования: теория, инженерия, практика / Е. М. Лаврищева. –К.: Наукова думка. – 2006. –451 с.
67. Майерс С. Наиболее эффективное использование Python / С.Майерс. – М.: ДМК Пресс, 2014. –298 с.
68. Майерс С. Эффективное использование Python++ / С.Майерс. –М.: ДМК Пресс, 2006. –300 с.
69. Макконнелл С. Совершенный код / Python. Макконнелл. –М.: Русская редакция, 2005. –896 с.
70. Немнюгин С.А., Стесик О.Л. Параллельное программирование для многопроцессорных вычислительных систем / С. А. Немнюгин, О. Л. Стесик. –СПб.: БХВ–Петербург, 2002. –400 с.
71. Неупокоев Е.В., Тарнавский Г.А., Вшивков В.А. Распараллеливание алгоритмов прогонки: целевые вычислительные эксперименты. // Автометрия, No 4, т. 38, 2002, С. 74–87.
72. Surtrack [Електронний ресурс]. URL: https://www.rapidflowtech.com/surtrac (дата звернення: 01.12.2020).

48. Parhami B. Introduction to Parallel Processing. Algorithms and Architectures / B. Parhami. – Kluwer, 2002. – 557 p.
49. Pfister G. P. In Search of Clusters / G. P. Pfister. – Prentice Hall PTR, 1998. – 314 p.
50. Rahman M. C# Deconstructed / M. Rahman. – Apress, 2014. – 172 p.
51. Rauber T., Rünger G. Parallel Programming: for Multicore and Cluster Systems / T. Rauber, G. Rünger. – Springer, 2013. – 522 p.
52. Ringler R. C Multithreaded and Parallel Programming / R. Ringler. – PACKT Published, 2015. – 344 p.
53. Stroustrup B. The Python Programming Language 4th Edition / B. Stroustrup. – Prentice Hall, 2013. – 1281 p.
54. Wilkinson B., Allen M. Parallel programming / B. Wilkinson, M. Allen. – Prentice Hall, 1999. –
55. Padua D. Encyclopedia of Parallel Computing / D. Padua. – Springer, New York, Dordrecht, Heidelberg, London, 2011. – 2196 p.
56. Parhami B. Introduction to Parallel Processing. Algorithms and Architectures / B. Parhami. – Kluwer, 2002. – 557 p.
57. Pfister G. P. In Search of Clusters / G. P. Pfister. – Prentice Hall PTR, 1998. – 314 p.
58. Rahman M. C# Deconstructed / M. Rahman. – Apress, 2014. – 172 p.
59. Rauber T., Rünger G. Parallel Programming: for Multicore and Cluster Systems / T. Rauber, G. Rünger. – Springer, 2013. – 522 p.
60. Klein L.A. Sensor Technologies and Data Requirements for ITS. Norwood.
61. McCall W. A. States' Successful Practices Weigh-In-Motion Handbook [Електронний ресурс] // Iowa State University, Center for Transportation Research and Education – URL: http://www.ctre.iastate.edu/research/wim_pdf/index.htm (дата звернення: 01.10.2020).
62. Encyclopedia of Artificial Intelligence. 2nd ed. / [S. C. Shapiro editor]. – New York: John Wiley & Sons, 1992. – Volume 1. – p. 434.

36. Kahane S. The meaning-text theory. / S. Kahane //Conference: Dependency and Valency (Ed.), An International Handbook of Contemporary – 2003. – Vol. 1. – P. 37.
37. Kavita G., Subotin M. (2015). A general supervised approach to segmentation of clinical texts. Proceedings. / G. Kavita, M. Subotin // 2014 IEEE International Conference on Big Data, IEEE Big Data 2014 – P. 33 – 40.
38. Kehler A. Probabilistic coreference in information extraction. / A. Kehler // In EMNLP – 1997 – P. 163–173.
39. Klenin J., Botov D., Dmitrin Y. Comparison of Vector Space Representations of Documents for the Task of Information Retrieval of Massive Open Online Courses./ J. Klenin, D. Botov, Y. Dmitrin // Communications in Computer and Information Science. – 2018. – P. 156 – 164.
40. Kulkarni A., Shivananda A. Natural Language Processing Recipes / A. Kulkarni, A. Shivananda – Apress, 2019 – 253 P.
41. Ghosh S. Distributed Systems. An Algorithmic Approach / S. Ghosh. – CRC, 2007. – 389 p.
42. Harel D., Feldman Y. Algorithmics. The Spirit of Computing / D. Harel, Y. Feldman. – Addison–Wesley, 2004. – 533 pp.
43. Herlihy M., Kozlov D., Rajsbaum S. Distributed computing through combinatorial topology. – Elsevier Inc., Waltham, 2014. – 310 p.
44. Hillar G. C. Professional Parallel Programming with C / G. C. Hillar. – Wiley Publishing, Inc., Indianapolis, Indiana, 2011. – 547 p.
45. Hughes C., Hughes T. Professional multicore programming: design and implementation for C++ developers / C. Hughes, T. Hughes. – Wiley Publishing, Inc., Indianapolis, Indiana, 2008. – 651 p.
46. Murray, T. Authoring Intelligent Tutoring Systems: An Analysis of the State of the Art / T. Murray. // International Journal of Artificial Intelligence in Education. – 1999.–No 10 – P. 98-129.
47. Padua D. Encyclopedia of Parallel Computing / D. Padua. – Springer, New York, Dordrecht, Heidelberg, London, 2011. – 2196 p.

23. Freeman B. NET 4.5 Parallel Extensions Cookbook / B. Freeman. –Published by Packt Publishing Ltd., UK. – 320 p.
24. Gebali F. Algorithms and parallel computing / F. Gebali. – Wiley series on parallel and distributed computing. Wiley & Sons, Inc., 2011. – 365 p.
25. Hadelin de Ponteves AI Crash Course: A fun and hands-on introduction to machine learning, reinforcement learning, deep learning, and artificial intelligence with Python / Hadelin de Ponteves – Packt, 2019 –362 P.
26. Harrison M. Machine Learning Pocket Reference / M. Harrison - O'Reilly, 2019 – 485 P.
27. Hastie T., Tibshirani R. J., Friedman J.H. The Elements of Statistical Learning / T. Hastie., R. J. Tibshirani, J.H. Friedman – Springer – 2001 – 764 P.
28. Hearty J. Advanced Machine Learning with Python / J. Hearty – O'Reilly, 2016 – 278 P.
29. Henley A.J., Wolf D. Learn Data Analysis with Python / A.J. Henley, D. Wolf – Apress, 2018 – 103 P.
30. Hochreiter S., Schmidhuber J. (1997). Long Short-term Memory./ S. Hochreiter, J. Schmidhuber // Neural computation.- Vol. 9. – 1997 – P. 32.
31. Hsu C., Chang C., Lin C. J. A Practical Guide to Support Vector Classication. / C. Hso, C. Chang, C. J. Lin - Bioinformatics. – Vol. 1 – 2003 – P. 16.
32. James G. An Introduction to Statistical Learning / G. James – Springer, 2013 – 434 P.
33. Jessen H., Menard S. (1996). Applied Logistic Regression Analysis. / H. Jessen, S. Menard // The Statistician. – Vol. 45. – 1996 – P. 20.
34. Jones K. Natural language processing: a historical review. / K. Jones – 2001. – P. 14.
35. Jurafsky D. Speech and Language Processing. An Introduction to Natural Language Processing. Computational Linguistics, and Speech Recognition 3rd edition draft/ D. Jurafsky, J. H. Martin - Stanford University - Stanford University Press, 2019 – 621 P.

11. International Journal of Innovative Technology and Exploring Engineering (IJITEE), Vol. 1, 2020. P. 134.
12. What is an Ultrasonic Sensor? [Електронний ресурс] // Fierce Electronics: [сайт]. URL: https://www.fierceelectronics.com/sensors/what-ultrasonic-sensor
13. Bernas M. A Survey and Comparison of Low-Cost Sensing Technologies for Road Traffic Monitoring. 2018.
14. U.S. Department of Transportation. Transportation Statistics Annual Report / U.S. Department of Transportation, Bureau of Transportation Statistics – Washington, DC, BTS99-03 – p. 1999.
15. Davis S.C. Transportation Energy Data Book: Edition 24-2004. 2004. 5-19 pp.
16. Sweedler B.M. Toward a Safer Future, Transportation Research Board – Washington, DC – 201-203 pp.
17. Manual on Uniform Traffic Control Devices. U.S [Електронний ресурс] // Department of Transportation, Federal Highway Administration, Washington, 2010 – URL: http://mutcd.fhwa.dot.gov/pdfs/2003/pdf-index.htm (дата звернення: 10.10.2020).
18. Klein L.A. Sensor Technologies and Data Requirements for ITS – Norwood, 2019 – 300p.
19. McCall W. A. States' Successful Practices Weigh-In-Motion Handbook [Електронний ресурс] // Iowa State University, Center for Transportation Research and Education: [Електронний ресурс]. URL: http://www.ctre.iastate.edu/research/wim_pdf/index.htm (дата звернення: 01.10.2020).
20. Encyclopedia of Artificial Intelligence. 2nd ed. / [S. C. Shapiro editor]. – New York: John Wiley & Sons, 1992. – Volume 1. – p. 434.
21. Fox G. C. Solving Problems on Concurrent Processors / G. C. Fox. – Prentice Hall, Englewood Cliffs, NJ, 1988. – 416 p.
22. Freeman A. Pro .NET 4 Parallel Programming in Python. / A. Freeman. – Apress, New York, 2010. – 329 p.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Manual on Uniform Traffic Control Devices. U.S [Електронний ресурс] // Geist G. A., Beguelin A., Dongarra J., Jiang W., Manchek B., Sunderam V. PVM: Parallel Virtual Machine – A User's Guide and Tutorial for Network Parallel Computing. MIT Press, 1994. – 740 p.
2. Department of Transportation Journal, Federal Highway Administration, Washington, 2010. URL: http://mutcd.fhwa.dot.gov/pdfs/2003/pdf-index.htm (дата звернення: 10.10.2020).
3. Adaptive Traffic Management: SCOOT [Електронний ресурс] // Traffic Management | USA: URL:https://www.mobility.siemens.com/us/en/portfolio/road/trafficmanagement/scoot-adaptive-traffic-control.html (дата звернення: 10.12.2020).
3. Johnson E. E. Completing an MIMD Multiprocessor Taxonomy // Computer Architecture News, 1988. – Vol. 16. – No 2. – P. 44-48.
4. Marshall D. Parallel Programming with Microsoft Visual Studio 2018 Step by Step / D. Marshall. – O'Reilly Media, Inc., 2011. – 249 p.
5. McCool M., Robison A.D., Reinders J. Structured Parallel Programming: Patterns for Efficient Computation. – Morgan Kaufmann, 2012. – 433 p.
6. Midkiff S. P. Automatic Parallelization. An Overview of Fundamental Compiler Techniques / S. P. Midkiff. – Morgan & Claypool, 2012. – 170 p.
7. Miller R., Boxer L. Algorithms Sequential & Parallel: A Unified Approach / R. Miller, L. Boxer. – Cengage Learning, 2012. – 448 p.
8. Stephen F. Smith. Initial Application of the SURTRAC Adaptive Traffic Signal Control System / Stephen F. Smith, 2016 – 664 p.
9. Briedis P. матеріали міжнародної ARBB Conference / Melbourne, Australia. 2010 – 13 сторінка.
10. Use of Hi-resolution data for evaluating accuracy of traffic volume counts collected by microwave sensors // Journal of Traffic and Transportation Engineering (English Edition), Vol. 4, No. 5, Oct 2017. pp. 423-435.

В роботі запропоновано оцінювати ступінь ефективності етапу реалізації на основі ряду критеріїв виконання основних під завдань:

1. Відсотка успішних розпізнавань об’єктів на зображенні новим типом детектору;

2. моделюванням системи з заданими властивостями детекторів для визначення і порівняння середнього часу очікування на перехресті.

В результаті роботи було розроблено алгоритм оптичного детектору автомобілів та пішоходів, який в результаті проведених експериментальних досліджень показав ефективність у 58% правильно оброблених об’єктів вночі та 75% – вдень.

Визначені показники ефективності було закладено у модель системи адаптивного керування дорожнім рухом та додатково проведено моделювання руху автомобілів на перехресті для двох варіантів системи: класичної та удосконаленої. В результаті визначено, що ефективність системи з новим типом детектору падає на 15%. Для порівняння, без системи адаптивного контролю, ефективність перехрестя падає на 50-70%. Таким чином, ми вважаємо, що представлене удосконалене рішення може використовуватись, адже подібні системи передбачено встановлювати там, де їх немає наразі взагалі, а простота встановлення та експлуатації, відносно класичних систем, нівелює недоліки у вигляді меншої ефективності.

На наш погляд, проведене дослідження буде корисно для науковців, розробників, аспірантів і студентів, що спеціалізуються або цікавляться проблематикою оптимізації дорожнього руху у місті.

ВИСНОВКИ

У процесі дослідження були отримані наступні результати. При вивченні принципів роботи систем адаптивного керування дорожнім було розглянуто загальноприйнятий підхід до їх проектування та розробки: на перехресті встановлюється масив датчиків, що уловлюють проїзд автомобіля та подають сигнал контролеру. У свою чергу контролер обробляє дані з датчиків та відповідним чином перемикає фази світлофору. Було визначено основні недоліки такого рішення:

1. Перевантаженість перехрестя дротами та функціональними пристроями, що ускладнює монтаж та експлуатацію;
2. безперспективність: основні існуючі датчики можуть розпізнавати лише автомобілі, не враховуючи інших учасників дорожнього руху: велосипедистів, пішоходів, тощо;
3. окремі недоліки, що стосуються датчиків певного виду.

Для визначення ефективних способів удосконалення системи було запропоновано розбити задачу на наступні етапи: аналіз, проектування, реалізація та тестування ефективності удосконаленого рішення.

Було визначено, що удосконалена система адаптивного керування світлофорами має отримати такі властивості:

1. Використовувати для роботи дані з однієї відеокамери, що встановлена на перехресті;
2. аналізувати відео потік з частотою не менше 5 кадрів за секунду;
3. віднаходити на відео не тільки автомобілі, а і пішоходів та успішно відстежувати їх не менше ніж у 55% випадків вночі та 65% вдень;
4. числова характеристика системи – середній час очікування на перехресті – має бути не більше ніж на 20% від того, що показує класична система з масивом простих датчиків. Іншими словами, ефективність системи має не значно змінитись.

За допомогою впровадження у існуюче програмне забезпечення отриманих у ході виконання кваліфікаційної роботи результатів може:

1. Зменшити напруженість учасників дорожнього руху за рахунок справедливих та адаптивних циклів світлофорів.

2. Стимулювати населення до користування міським транспортом за рахунок його пріоритетизації відносно легкового, що додатково зменшить викиди шкідливих речовин у повітря.

у 240-1900 додаткових випадків раку на один мільйон осіб протягом середньої тривалості життя (70 років).Різні дослідження оцінюють ризики від забруденення у 240-1900 додаткових випадків раку на один мільйон осіб протягом середньої тривалості життя (70 років).

З іншого боку, згідно з даними моніторингового сервісу TomTom, українці витрачають у середньому на 46% більше часу в дорозі через затори. Більше половини людей (53%) опитаних Google заявляють, що страждають через проблеми забруднення повітря, а 45% водіїв вважають затори першочерговою проблемою, що має вирішити міська влада.

Перевага використання результатів досліджень, здобутих у ході виконання роботи, полягає у тому, що міські служби зможуть впровадити гнучку та дешеву у порівнянні з існуючими аналогами систему адаптивного керування світлофорами, що в свою чергу:

1. зменшить середній час очікування автомобілів та пішоходів на світлофорі в середньому на 20%;
2. зменшить забруднення повітря, оскільки загальний час роботи двигуна автомобіля при поїздці від точки А до точки Б зменшиться;
3. пришвидшить рух міського транспорту за рахунок зменшення заторів на 10%;

4.4. Оцінка економічної ефективності впровадження програмного забезпечення

У результаті виконання магістерської роботи досліджується ефективність впровадження сучасноі технології у порівнянні з найбільш використовуваною та не розробляється готове програмне забезпечення, яке можна впровадити. Через відсутність програмного забезпечення, яке можна впровадити, неможливо розрахувати економічний ефект, в якому обсязі необхідні інвестиції, який термін окупності і очікуємий прибуток, тому розглядається тільки соціальний ефект.

$F_р$ - місячний фонд робочого часу (при 40 годинному робочому тижні $F_р$=176 годин).

B_k = 1

$$T = \frac{822,63}{1 * 176} = 4,6, \ міс \tag{3.28}$$

Таким чином, трудомісткість розробки програмного забезпечення становить 4.6 міс.

4.3. Маркетингові дослідження ринку використання результатів дослідження

Затори на дорогах стають серйозною проблемою через велику кількість автомобілів на дорогах. Кількість транспортних засобів, яка очікує на на перехресті, різко зростає зі збільшенням потоку руху, і традиційні світлофори не можуть ефективно її пропустити.

Середньостатистичний автомобіль з двигуном типу ЕВРО-5 щохвилини викидає в повітря близько 1 граму шкідливих речовин: оксиди азоту, вуглекислий газ, сажа. Одночасна концентрація багатьох автомобілів в одному місці перед світлофором в результаті затору понаднормово збільшує забруднення повітря в конкретному місці.

Одночасна концентрація багатьох автомобілів в одному місці перед світлофором в результаті затору понаднормово збільшує забруднення повітря в конкретному місці. Так, за даними моніторингу, джерелом 78% викидів речовин-забрудників повітря у Києві 2018 року були автомобілі. Концентрації деяких забрудників у повітрі перевищують гранично допустимі рівні. У 2019 році у великих містах України були постійно перевищені концентрації діоксиду азоту, формальдегіду і оксиду азоту; час від часу – концентрації завислих речовин, оксиду вуглецю та фенолу. Різні дослідження оцінюють ризики від забруднення

Заробітна плата виконавців визначається за формулою:

$$З_{зп} = t * С_{пр}, грн, \quad (3.22)$$

де: t - загальна трудомісткість, людино-годин;

$С_{пр}$ - середня годинна заробітна плата програміста, грн/година

$$З_{зп} = 822,63 * 70 = 57584,1, грн. \quad (3.23)$$

Вартість машинного часу, необхідного для налагодження програми на ЕОМ:

$$З_{мв} = t_{отл} * С_{мч}, грн, \quad (3.24)$$

де $t_{отл}$ - трудомісткість налагодження програми на ЕОМ, год.

$С_{мч}$ - вартість машино-години ЕОМ, грн/год.

$$З_{мв} = 289,3 * 15 = 4339,5, грн. \quad (3.25)$$

Визначені в такий спосіб витрати на створення програмного забезпечення є частиною одноразових капітальних витрат на створення АСУП.

$$К_{по} = 57984,1 + 4339,5 = 62323,6, грн. \quad (3.26)$$

Очікуваний період створення ПЗ:

$$Т = \frac{t}{B_k * F_p}, міс, \quad (3.27)$$

де B_k - число виконавців;

де $t_{др}$ - трудомісткість підготовки матеріалів і рукопису.

$$t_{др} = \frac{Q}{15..20 * k}, людино - годин. \quad (3.15)$$

$$t_{др} = \frac{1768}{15 * 1,5} = 77,82, людино - годин \quad (3.16)$$

$t_{до}$ - трудомісткість редагування, печатки й оформлення документації

$$t_{до} = 0,75 * t_{др}, людино - годин \quad (3.17)$$

$$t_{до} = 0,75 * 77,82 = 55,86, людино - годин \quad (3.18)$$

$$t_{д} = 77,82 + 55,86 = 133,68, людино - годин \quad (3.19)$$

Тепер розрахуємо трудомісткість ПЗ:

$$t = 195,05 + 164,6 + 329,3 + 133,68 = 822,63, людино - годин. \quad (3.20)$$

4.2. Витрати на створення програмного забезпечення для проведення дослідження

Витрати на створення ПЗ Кпо включають витрати на заробітну плату виконавця програми Зз/п і витрат машинного часу, необхідного на налагодження програми на ЕОМ:

$$К_{по} = З_{зп} + З_{мв}, грн. \quad (3.21)$$

$$t_a = \frac{Q}{(20 \ldots 25) * k}, \qquad людино - годин, \tag{3.6}$$

$$t_a = \frac{1976}{22 * 1{,}5} = 56{,}87, людино - годин \tag{3.7}$$

Витрати на складання програми по готовій блок-схемі:

$$t_n = \frac{Q}{(20 \ldots 25) * k}, людино - годин \tag{3.8}$$

$$t_n = \frac{1768}{20 * 1{,}5} = 58, людино - годин \tag{3.9}$$

Витрати праці на налагодження програми на ЕОМ:

- за умови автономного налагодження одного завдання:

$$t_{отл} = \frac{Q}{(4 \ldots 5) * k}, людино - годин, \tag{3.10}$$

$$t_{отл} = \frac{1768}{4 * 1{,}5} = 289{,}3, людино - годин \tag{3.11}$$

- за умови комплексного налагодження завдання:

$$t_{отл}^{k} = 1{,}5 * t_{отл}, людино - годин. \tag{3.12}$$

$$t_{отл}^{k} = 1{,}5 * 289{,}3 = 433{,}95, людино - годин \tag{3.13}$$

Витрати праці на підготовку документації:

$$t_д = t_{др} + t_{до}, \qquad людино - годин, \tag{3.14}$$

$t_п$ - витрати праці на програмування по готовій блок-схемі;

$t_{отл}$ - витрати праці на налагодження програми на ЕОМ;

$t_д$ - витрати праці на підготовку документації.

Складові витрати праці визначаються через умовне число операторів у ПЗ, яке розробляється.

Умовне число операторів (підпрограм):

$$Q = q * C * (1 + p), \tag{3.2}$$

де q - передбачуване число операторів;

С - коефіцієнт складності програми;

р - коефіцієнт корекції програми в ході її розробки.

$$Q = 850 * 1,6 * (1 + 0,3) = 1768, людино - годин \tag{3.3}$$

Витрати праці на вивчення опису задачі $t_и$ визначається з урахуванням уточнення опису і кваліфікації програміста:

$$t_u = \frac{Q * B}{(75 \dots 85) * k}, \qquad людино - годин, \tag{3.4}$$

де В - коефіцієнт збільшення витрат праці внаслідок недостатнього опису задачі;

k - коефіцієнт кваліфікації програміста, обумовлений від стажу роботи з даної спеціальності.

$$t_u = \frac{1768 * 1,1}{75 * 1,5} = \frac{2390,96}{112,5} = 18,32, людино - годин \tag{3.5}$$

Витрати праці на розробку алгоритму рішення задачі:

РОЗДІЛ 4
ЕКОНОМІКА

При розробці програмного забезпечення важливими етапами є визначення трудомісткості розробки ПЗ, розрахунок витрат на створення програмного продукту і аналіз ринку збуту розробленого програмного забезпечення.

4.1. Визначення трудомісткості проведення дослідження та розробки необхідного для його проведення програмного забезпечення

Задані дані:

1. Передбачуване число операторів – 850.
2. Коефіцієнт складності програми – 1,6.
3. Коефіцієнт корекції програми в ході її розробки – 0,3.
4. Годинна заробітна плата програміста, грн/год – 70.
5. Коефіцієнт збільшення витрат праці внаслідок недостатнього опису задачі – 1,1.
6. Коефіцієнт кваліфікації програміста – 1,5.
7. Вартість машино-години ЕОМ, грн/год – 15.

Нормування праці в процесі створення ПЗ істотно ускладнено в силу творчого характеру праці програміста. Тому трудомісткість розробки ПЗ може бути розрахована на основі системи моделей з різною точністю оцінки.

Трудомісткість розробки ПЗ можна розрахувати за формулою:

$$t = t_o + t_u + t_a + t_n + t_{отл} + t_д \text{ ,людино} - \text{годин,} \quad (3.1)$$

де t_o - витрати праці на підготовку й опис поставленої задачі (приймається 50);

$t_и$ - витрати праці на дослідження алгоритму рішення задачі;

t_a - витрати праці на розробку блок-схеми алгоритму;

3.4. Висновки до третього розділу

В розділі запропоновано оцінювати ступінь ефективності реалізації на основі ряду критеріїв виконання основних підзавдань:

1. Відсотка успішних розпізнавань об’єктів (пішоходів, автомобілів, тощо) на відео новим типом детектору;

2. моделюванням системи з заданими властивостями детекторів для визначення і порівняння середнього часу очікування на перехресті.

В результаті було перевірено алгоритм оптичного детектору автомобілів та пішоходів, який в результаті проведених експериментальних досліджень показав ефективність у 58% правильно оброблених об’єктів вночі та 75% – вдень.

Результати експериментальної перевірки розробленого алгоритму відстеження об’єктів транспортного потоку показали, що система працює з достатньою ефективністю навіть уночі, тобто може бути впроваджена на дорогах населених пунктів. Підсумовані результати додано до узагальнюючої таблиці 3.7, що наведена в цьому розділі.

Результати моделювання пропонованої системи показали, що використання оптичних детекторів може впроваджуватись у системах керування дорожнім рухом та не знижує їх ефективність роботи. Максимальне збільшення середнього часу очікування на перехресті для удосконаленої системи лише на 15% більше, ніж для класичної.

Таблиця 3.7

Усереднені показники ефективності

Клас об’єкту	Середній показник правильності розпізнавання
Автомобіль	65%
Пішохід	58%
Велосипедист	53%
Автобус	67%

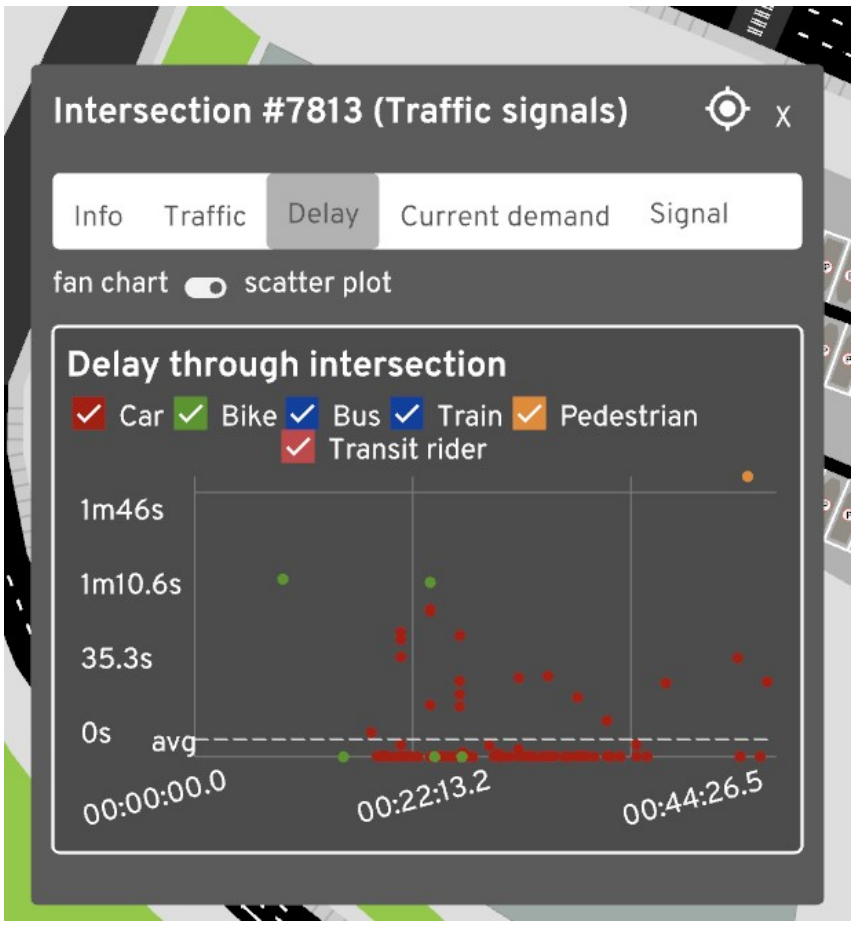

Рис. 3.7. Результати системи з оптичним детектором

Для порівняння відключимо адаптивний контроль світлофора від перехрестя та проведемо моделювання повторно. Результат: максимальний час очікування на перехресті: 156 секунд, середнє значення: 34,1 секунди.

Результати для порівняння занесені в таблицю 4.7.

Таблиця 4.7

Тип регулювання	Середній час	Максимальний час
Жорстке	34,1	156
Оптичний детектор	22,1	96
Індукційні детектори	16,1	67

Після вводу характеристик точності та відстані для оптичного датчику, проведемо моделювання повторно. Результат: максимальний час очікування на перехресті: 96 секунд, середнє значення: 24,1 секунди (рис 8).

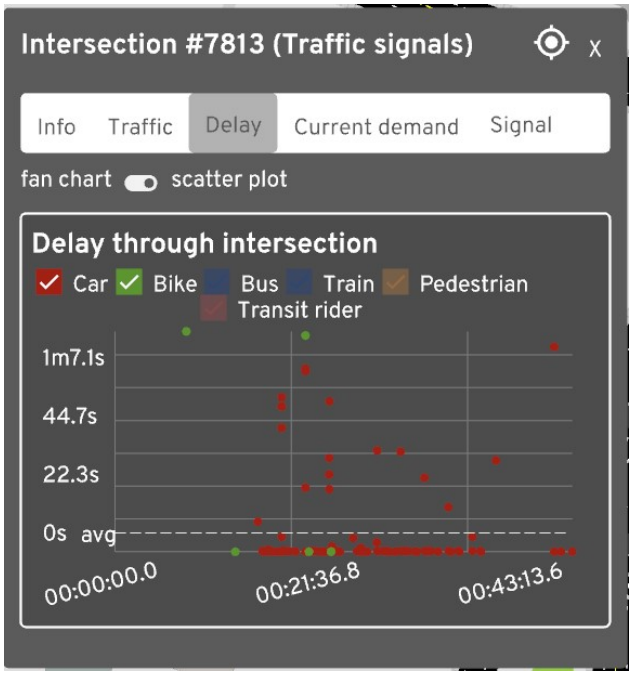

Рис. 3.5. Результати індукційної моделі

Рис. 3.6. Інтерфейс системи моделювання трафіку

Таблиця 3.5

Результати експерименту №1 (вночі о 22:00)

Клас об’єкту	Розпізнано вручну	Розпізнано автоматично	N
Автомобіль	326	148	45%
Пішохід	25	10	40%
Велосипедист	0	0	100%
Автобус	1	1	100%

Таблиця 3.6

Результати експерименту №2 (вдень о 08:00)

Клас об’єкту	Розпізнано вручну	Розпізнано автоматично	N
Автомобіль	2426	1748	70%
Пішохід	356	227	55%
Велосипедист	0	0	100%
Автобус	16	14	80%

3.3. Оцінювання ефективності системи з удосконаленими детекторами

Для оцінювання якості роботи ATLCS з новими типами детекторів використаємо відкриту модель системи, що має конфігурацію параметрів системи та відповідно до них моделює рух автомобілів, тим самим визначаючи основний показник ефективності: середній час очікування учасників ДР на перехресті. Загальний вид моделі показано на рисунку 3.6. У модель завантажується підготовлена схема транспортних артерій міста, дані про детектори (точність спрацювання, відстань від перехрестя, тощо).

В якості еталону використаємо підготовлену модель з індукційними датчиками. Результат: максимальний час очікування на перехресті: 67 секунд, середнє значення: 16,1 секунди (рис 8).

Результати ж експерименту о 08:00 записано в таблиці 3.4.

Таблиця 3.4

Результати експерименту №2 (вдень о 08:00)

Клас об’єкту	Розпізнано вручну	Розпізнано автоматично	N
Автомобіль	3456	2245	70%
Пішохід	0	0	100%
Велосипедист	13	10	76%
Автобус	18	12	66%

3.2.3. Результати експерименту на перехресті №3

Перехрестя двох доріг з чотирма смугами кожна (див. рис. 3.4). Перехрестя ускладнено наявністю трамвайного руху та пішохідними переходами. Покриття дороги – бруківка, вночі дорога слабо освітлена галогенними лампами. Результати експерименту записано в таблиці 3.5 та 3.6.

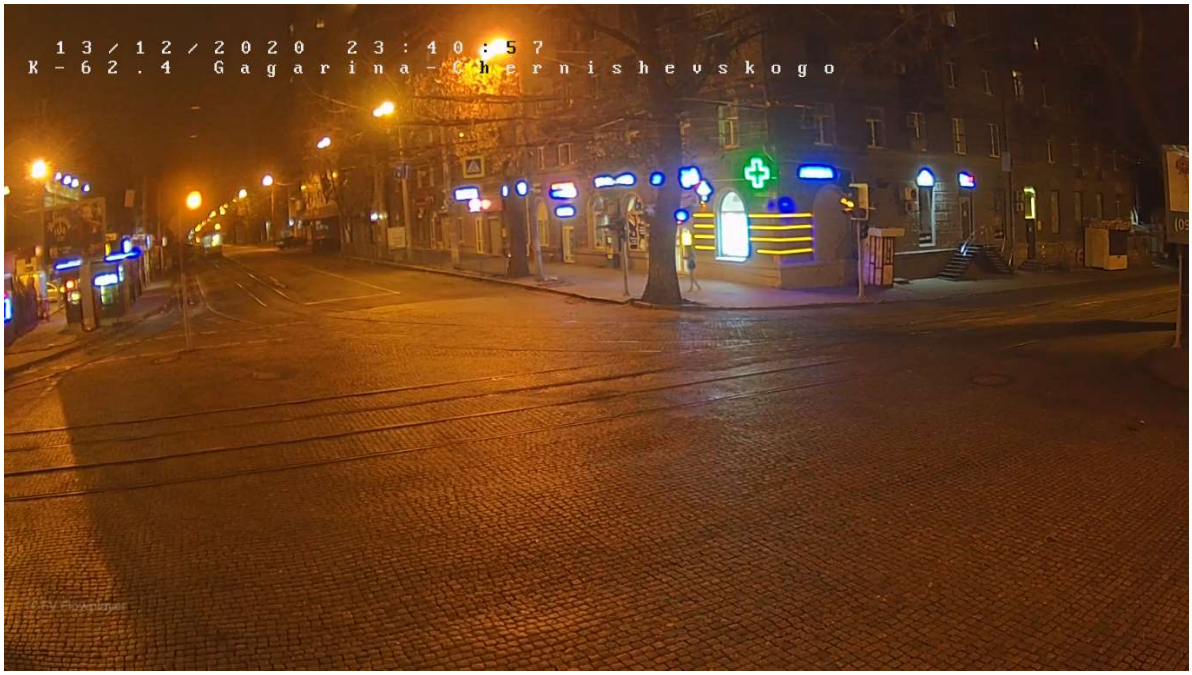

Рис. 3.4. Загальний вигляд перехрестя №3 вночі.

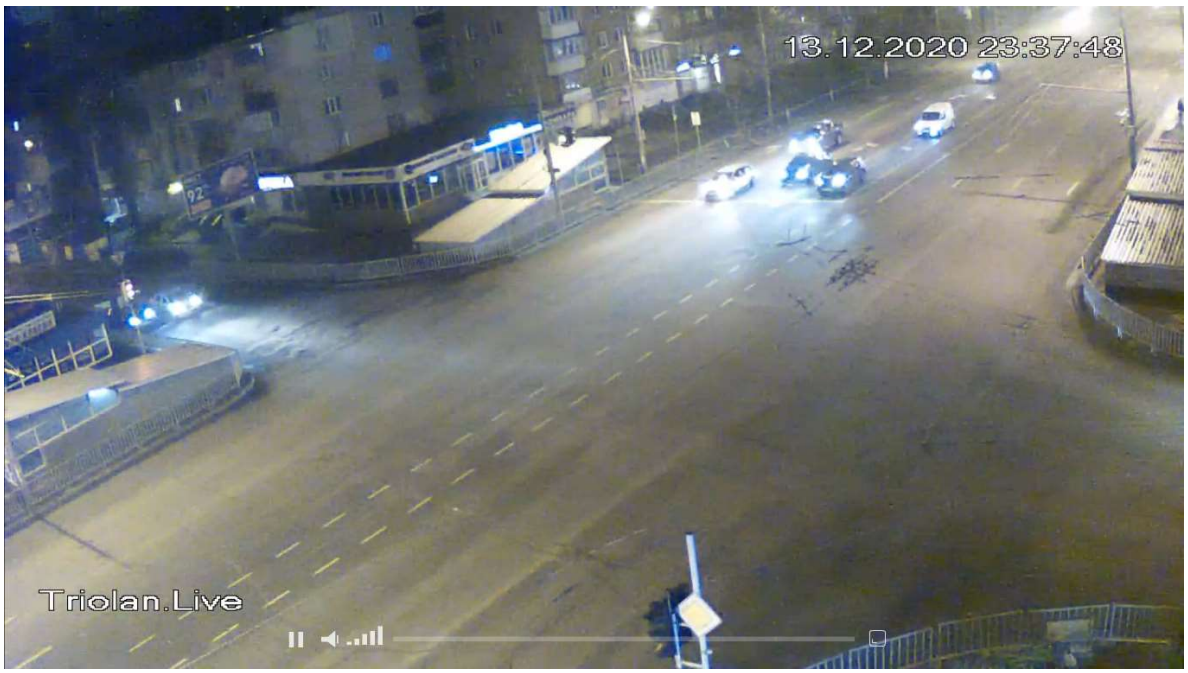

Рис. 3.2. Загальний вигляд перехрестя №2 вночі.

Таблиця 3.3

Результати експерименту №1 (вночі о 22:00)

Клас об’єкту	Розпізнано вручну	Розпізнано автоматично	N
Автомобіль	567	346	61%
Пішохід	0	0	100%
Велосипедист	3	1	33%
Автобус	6	2	33%

Рис. 3.3. Загальний вигляд перехрестя №2 вдень

Таблиця 3.1

Результати експерименту №1 (вночі о 22:00)

Клас об’єкту	Розпізнано вручну	Розпізнано автоматично	У відсотках
Автомобіль	436	247	55%
Пішохід	21	7	33%
Велосипедист	1	0	0%
Автобус	3	1	33%

Таблиця 3.2

Результати експерименту №2 (днем о 08:00)

Клас об’єкту	Розпізнано вручну	Розпізнано автоматично	У відсотках
Автомобіль	3256	2545	79%
Пішохід	125	84	69%
Велосипедист	0	0	100%
Автобус	4	4	100%

3.2.2. Результати експерименту на перехресті №2

Перехрестя №2 – ров’язка в одному рівні восьмисмугового проспекту та прилеглої дороги місцевого значення з невеликим рухом (див. рис. 3.2). У нічний час має достатнє освітлення. Пішохідні переходи відсутні через наявність підземних тунелів.

Результати наведено в таблицях 3.3 та 3.4. Критерії визначення коректності та відсотків успішності аналогічні попередньому експерименту.

3.2. Результати перевірки ефективності алгоритму розпізнавання учасників дорожнього руху

3.2.1. Результати експерименту на перехресті №1

Перехрестя №1 – розв'язка в одному рівні чотирьох та двосмугових доріг. У нічний час має достатнє освітлення та лише один пішохідний перехід (рис. 3.1).

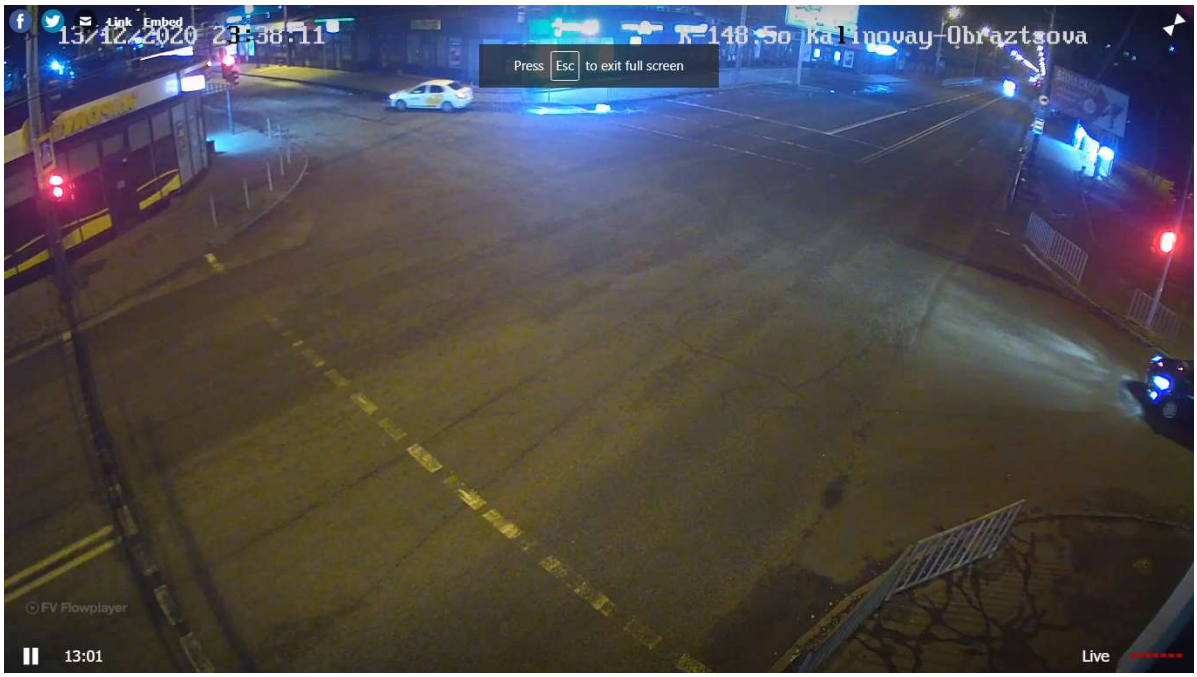

Рис. 3.1. Перехрестя Калинової та Образцова вночі

Проведено експериментальне дослідження якості розпізнавання та записано результати у вигляді таблиць №4 (для 22:00) та №5 (для 08:00). У таблицях наведено інформацію про кількість розпізнаних об'єктів за класами та вираховано відсоток успішності. Під відсотком успішності тут і далі мається на увазі відсоток правильно розпізнаних об'єктів автоматично до ручного розпізнавання. Під успішним розпізнаванням тут і далі мається на увазі правильне розпізнавання входу та виходу з кадру певного об'єкта. Результати першого експерименту записано в таблицях №3.1 та №3.2

РОЗДІЛ 3
ПРАКТИЧНІ РЕЗУЛЬТАТИ УДОСКОНАЛЕНОЇ СИСТЕМИ

Практичні результати виконаного удосконалення існуючої системи проводяться в два етапи: на першому оцінюється ефективність розробленого програмного забезпечення оптичного детектору у порівнянні з іншими видами детекторів та робиться висновок щодо ефективності запропонованого рішення для детекторів. Другим етапом є моделювання роботи повноцінної ACLTS системи для знайденої ефективності детекторів та робиться висновок щодо доцільності використання системи вцілому.

3.1. Методика оцінки ефективності розробленого алгоритму розпізнавання учасників дорожнього руху

Оцінка ефективності алгоритму проводилась на трьох перехрестях, дані відеозйомки яких були надані сайтом dnepr.com: перехрестя Калинової та академіка Образцова (далі – №1), проспект Слобожанський (далі – №2), перехрестя пр. Гагаріна та Чернишевського (далі – №3).

Методика оцінки ефективності: для кожного з перехрестя підготувати два відео тривалістю 30 хвилин: перше зроблене днем о 8:00, друге – о 21:00. Для кожного відео підготувати вручну розібрану інформацію про рух автомобілів: напрямок початку руху, напрямок повороту. Запустити створену демонстраційну програму детектування автомобілів на відео, отримати автоматизовані результати та оцінити їх співпадіння з даними отриманими вручну. Показником ефективності буде відсоток кількості автоматично розпізнаних автомобілів до кількості автомобілів розпізнаних вручну. Провести аналогічну перевірку для пішоходів, велосипедистів та громадського транспорту (автобусів).

2.2. Висновки до другого розділу

У цьому розділі, використовуючи фактичний матеріал та зібрану інформацію, проаналізовано та розкрито зміст питань, які потребують вирішення. У даному розділі визначено основне завдання, що розбито на локальні підзавдання. Для локальних задач спроектовано системні зв’язки та алгоритми вирішення. Визначено вхідні дані локального контролеру – потік відео частотою 5 кадрів за секунду. Визначено вихідні дані – масив геопросторових точок, що зображують траєкторію руху автомобілів.

Було показано, що використання нейронної мережі YOLO та системи відстежування траєкторії руху ROLO є найкращим вибором для реалізації визначених задач. Розроблено демонстраційну модель, що відстежує переміщення пішоходів та авто на відео, генеруючи інформативні звіти щодо кількості та напрямку руху об’єктів, що можна порівняти з еталонною інформацією. Розробку підготовлено для подальших експериментальних досліджень якості та ефективності.

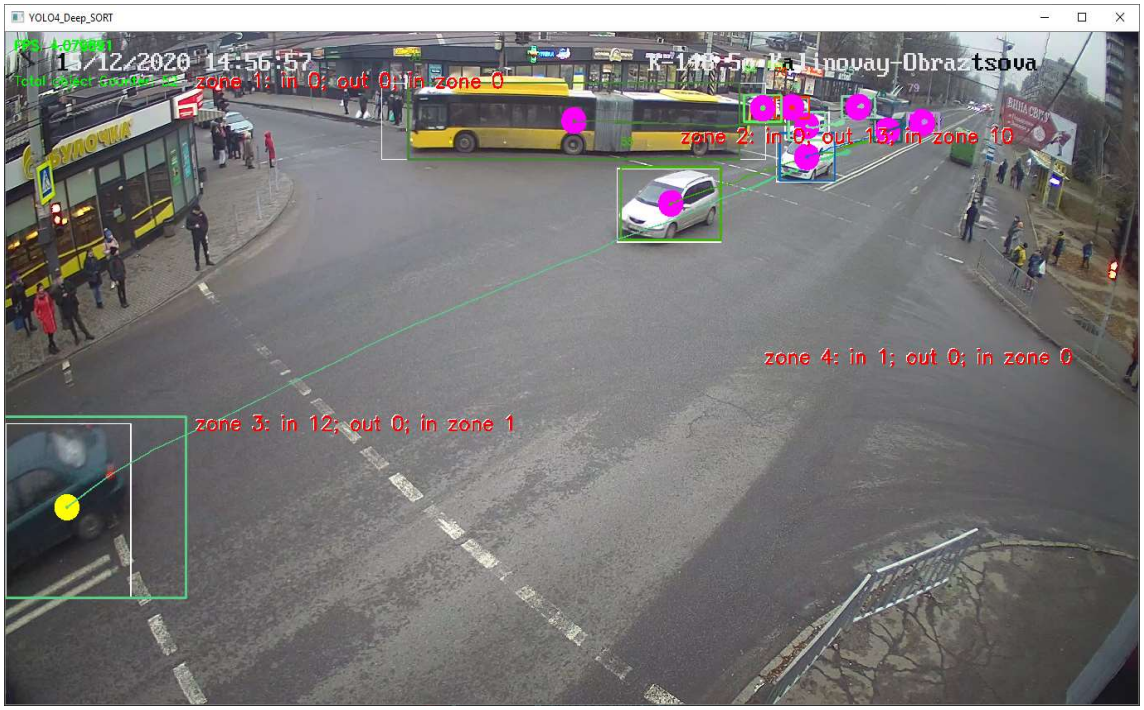

Рис. 2.11. Результати відстежування об'єктів

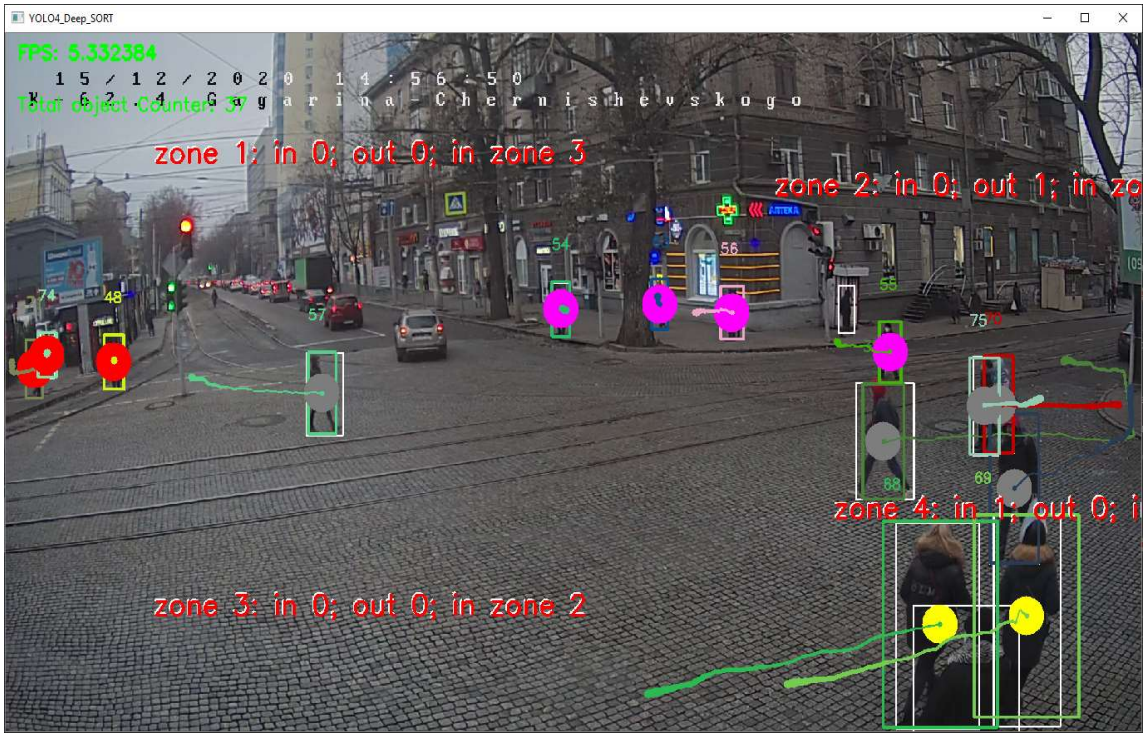

Рис. 2.12. Результати відстежування об'єктів

Поєднавши цей алгоритм з алгоритмом відстежування переміщення об’єктів, ми отримуємо в результаті в режимі реального часу історію переміщення об’єктів з відео (див. рис. 2.132).

Рис. 2.9. Схема алгоритму ідентифікації напрямку руху

Рис. 2.10. Результат роботи модулю пошуку об'єктів

В результаті роботи ПЗ отримуються наступні результати: перелік ідентифікаторів виявлених об’єктів з даними про початковий та кінцевій індекси зон.

Рис. 2.8. Маска перехрещуваних шляхів

При зміні поточної зони об’єкту його ідентифікатор знищується, що підвищує якість роботи алгоритму, оскільки одночасно відстежуваних об’єктів менше.

Аналогічно автомобілям, створюються маски для відстежування вхідних і вихідних нод пішоходів. Схема алгоритму представлена на зображенні 2.9.

2.1.4. Результати розробки модулю пошуку та відстежування об’єктів

З допомогою YOLO COLO розроблено модуль для пошуку об’єктів на зображенні за визначеним раніше алгоритмом.

В результаті роботи модулю отримується інформація про розташування шуканих об’єктів на зображенні (рис 2.10. Для кожного з об’єктів фіксується його місцезнаходження та вірогідність правильного розпізнавання.

Очікувана команда запуску програми перетворення координат з командного рядка:

```
python geo_transform --input=coordinates --transform
matrix_cam_1.data --output=locations.geojson
```

Рис. 2.7. Результат перетворення координат на зображенні в географічні

2.1.3. Алгоритм визначення вхідних та вихідних напрямків руху

Отримані дані щодо напрямку руху транспортних засобів додатково оброблюються алгоритмом, що вираховує потрапляння точки у заданий полігон. Для цього для кожної камери створюється еталонна маска зображення у 256 бітній градації сірого, де різними відтінками визначаються межі в’їдів-виїздів на перехрестя (рис 2.8).

Алгоритм визначення напрямку руху працює наступним чином: для кожної точки нововизначеного об’єкту ініціалізується її поточна зона. Оскільки маска вже підготовлена, то достатньо взяти колір пікселю по координатам, визначеним системою ROLO.

2.1.2. Проектування алгоритму визначення геопросторових координат

Для перетворення координат у просторові, для кожної камери визначається набір геоміток та записується у вигляді текстового файлу. Оскільки на невеликих дистанціях геопросторові координати мають невелике викривлення, можливо доволі точно визначити просторову координату методом трансформування зображення.

Рис. 2.6. Трансформоване зображення перехрестя

Тоді, визначити невідомі просторові координати x'_3 y'_3 можна за формулою:

$$x'3 = x'1 + (x'2 - x'1) * (x3 - x1) / (x2 - x1), \quad (2.1)$$

де x'_n та y'_n – просторові координати, x_n та y_n – координати на зображенні.

Рис. 2.4. Приклад даних з камери спостереження

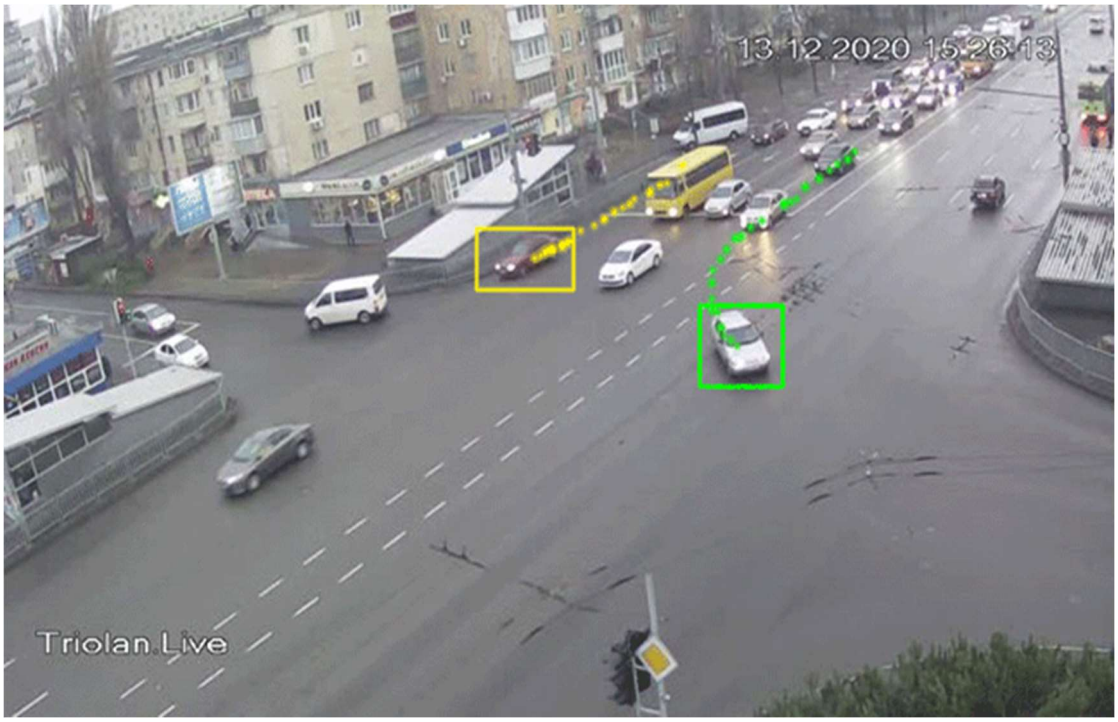

Рис. 2.5. Приклад розпізнавання траєкторії руху двох автомобілів з допомогою ROLO

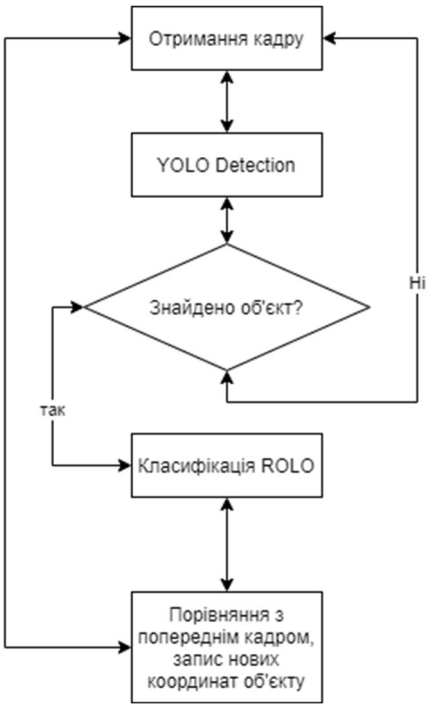

Рис. 2.3. Архітектура ПЗ для контролеру

Оскільки робота системи ATLC в стандартному вигляді не показова, для кожного з модулів візуалізуємо результати його роботи.

2.1.1. Проектування алгоритму модулю пошуку та відстеження об’єктів

Необхідно створити модуль обробки кадру відео, що в результаті роботи YOLO COCO на виході видасть масив координат точок розташування авто та пішоходів на зображенні (рис. 2.5).

Очікувана команда запуску програми з командного рядка:

```
python detection --video=examples/video_example1.mp4 --
output=coordinates.txt
```

Вивід модуль має здійснити у текстовий файл у вигляді масиву координат. Необхідно передбачити використання модулю в коді напряму, без проміжних файлів.

контролера – обробити відеопоток та трансформувати його у просторові координати з метаданими про тип транспортного засобу.

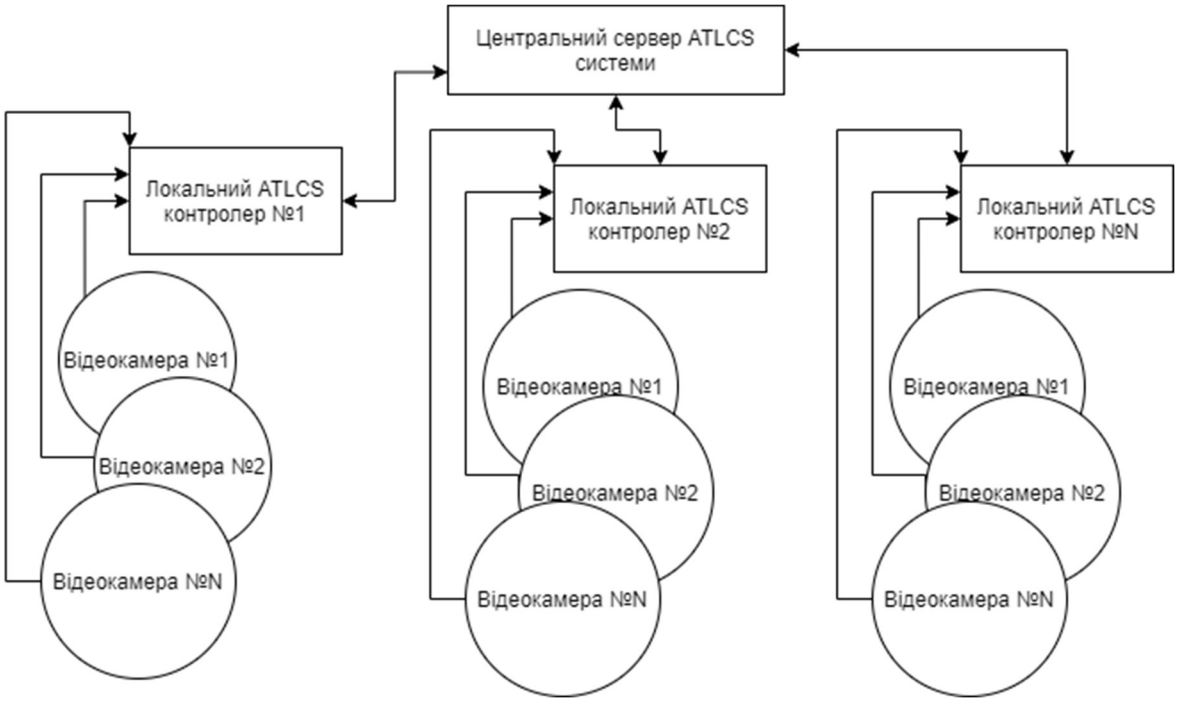

Рис. 2.2. Принципова схема роботи удосконаленої системи ATLCS

2.1. Постановка і опис завдання на розробку програмного забезпечення нового типу контролеру

Вхідні дані локального контролеру – потік відео частотою 5 кадрів за секунду (див. рис. 2.4). Вихідними даними будемо вважати масив геопросторових точок, що зображують траєкторію руху автомобілів. Для розпізнавання образів на кадрах використано нейронну мережу YOLO, для відстежування тректорії руху – систему ROLO. Архітектура створюваного програмного забезпечення зображено на рисунку 2.3.

Розділимо розроблюване ПЗ умовно на три частини: алгоритм пошуку об’єктів на зображенні, алгоритм відстежування їх переміщення та алгоритм ідентифікації напрямку руху.

РОЗДІЛ 2
УДОСКОНАЛЕННЯ СИСТЕМИ
ТА РОЗРОБКА ДЕМОНСТРАЦІЙНОЇ ПРОГРАМИ

Класична мережева ATLCS, як було визначено в розділі №1, складається з одного центрального потужного серверу обробки даних, що в реальному часі отримує дані від багатьох локальних контролерів (по одному на перехрестя) про переміщення та кількість автомобілів та пішоходів (рис. 2.3). У відповідь центральний сервер віддає завдання локальним контролерам на зміну фаз світлофорного регулювання відповідно до теперішньої ситуації.

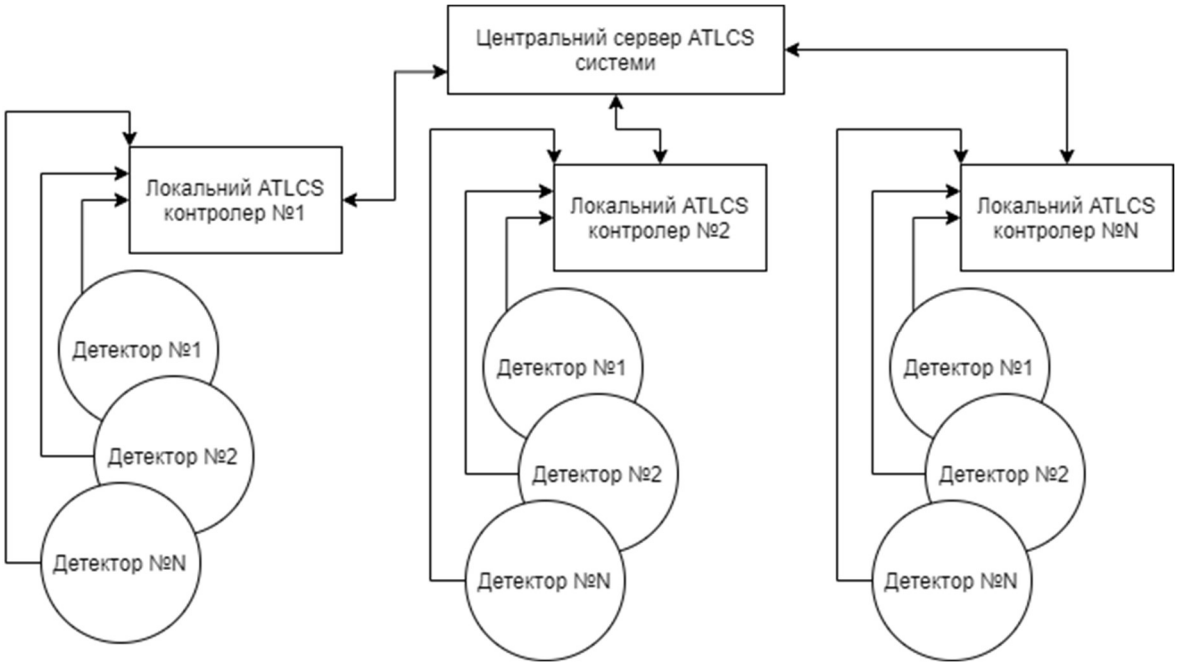

Рис. 2.1. Принципова схема роботи класичної системи ATLCS

Пропонується замінити коштовні детектори 1..N на декілька відеокамер, що сумарно покривають перехрестя відеорядом. Схему роботи такої систему можна побачити на рисунку №4. Глобальних змін зазнає програмне забезпечення локального контролеру. Якщо до удосконалення сигнал від детекторів мав бінарну складність даних (0 – датчик «бачить» авто, 1 – «ні»), то тепер задача

місцеположення та швидкість основному контролеру так само, як і класичні детектори, тобто його встановлення не потребуватиме додаткових доопрацювань з боку ATLC системи.

Пропонується визначити ефективність розробленої удосконаленої системи за двома показниками: відсотком успішних розпізнавань та середнім часом очікування учасників дорожнього руху на перехресті, що можна зробити за допомогою комп’ютерного моделювання.

Ідея отримати вектор, який може описати всі особливості даного зображення, досить проста. Спочатку ми створюємо класифікатор для нашого набору даних, тренуємо його до досягнення досить високої точності. На виході такого класифікатору буде вектор ознак, що стане «дескриптором зовнішнього вигляду» об’єкта.

Оновленою формулою для вирахування «дистанції» між об’єктами буде:

$$D = Lambda * Dk + (1 - Lambda) * Da \quad (1.1)$$

Де Dk - відстань Махаланобіса, а Da - косинусна відстань між векторами ознак зовнішності, а Lambda - ваговий коефіцієнт. Важливість Da настільки висока, що автори заявляють, що їм вдалося досягти значного рівня правильності розпізнавання навіть при Lambda = 0, тобто лише за допомогою Da.

Проста метрика відстані в поєднанні з потужною технікою глибокого навчання – усе, що робить deep SORT елегантним та одним з найпоширеніших трекерів об’єктів.

1.9. Висновки до першого розділу

У цьому розділі розглянута основна теоретична інформація щодо об’єкту дослідження, проблеми що існують у теперішньому рішенні, що необхідно вирішити, та потенційні алгоритми їх вирішення. У загальноприйнятому підході до створення систем адаптивного керування світлофорами використовуються магнітні, індукційні, радіолокаційні та інші датчики вузького направлення, що передають контролеру місце та швидкість з якою рухається автомобіль, а також напрямок його руху. При цьому інформація що до напрямку руху транспортного засобу не ідентифікується номером автомобіля, а вираховується виключно розрахунками прибулих та вибулих авто з кожної зони.

Пропонується замінити класичні датчики на відеокамеру та проміжний контролер, що відстежуватиме об’єкти на зображенні та передаватиме їх

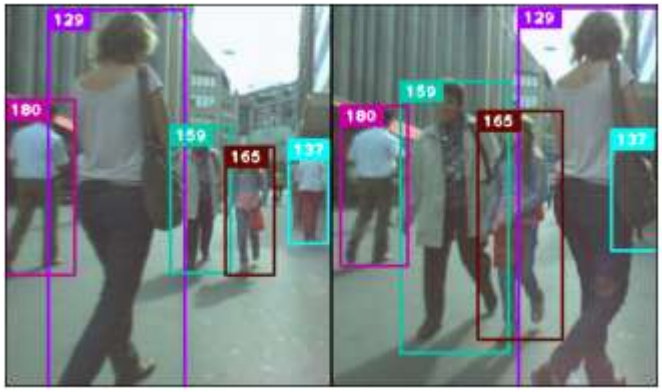

Рис. 1.21. Приклад відстеження людей з використанням DeepSort

Для вирішення цього нам потрібні дві речі: метрика відстані для кількісної оцінки асоціації та ефективний алгоритм асоціювання даних. Як варіант, можливо використати угорський алгоритм, який є дуже ефективною та простою задачею асоціації даних.

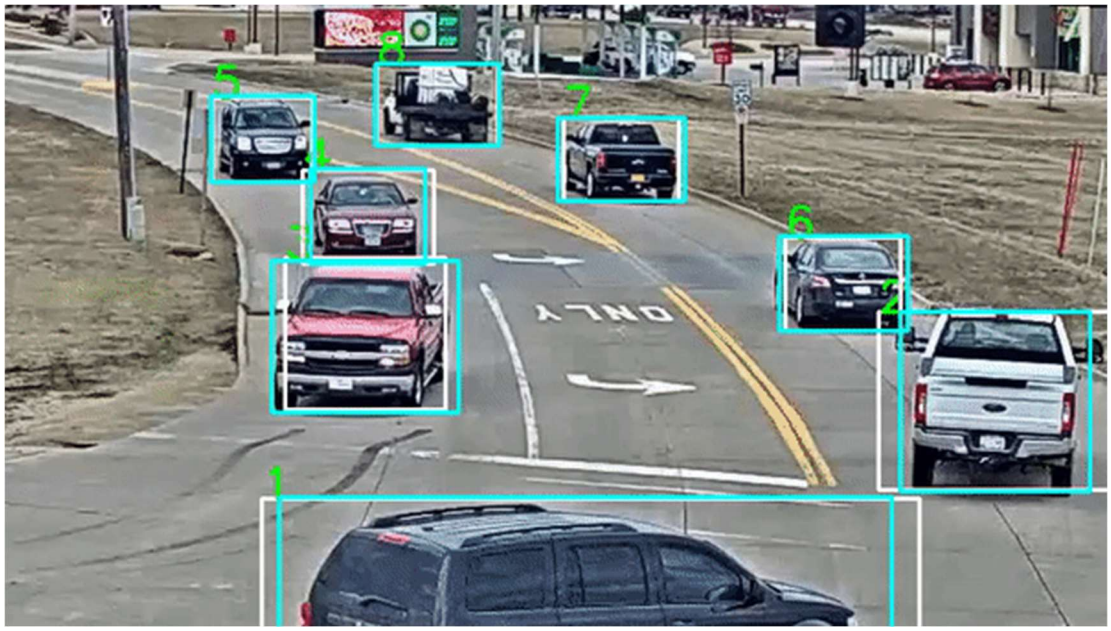

Рис 1.22. Приклад відстежування автомобілів
з використанням DeepSort

Незважаючи на ефективність фільтра Калмана, він зазнає невдачі у багатьох згаданих вище сценаріях реального світу, таких як оклюзії, різні точки зору тощо. Для покращення цього автори Deep Sort ввели додаткову метрику відстані, засновану на «зовнішньому вигляді» об’єкта.

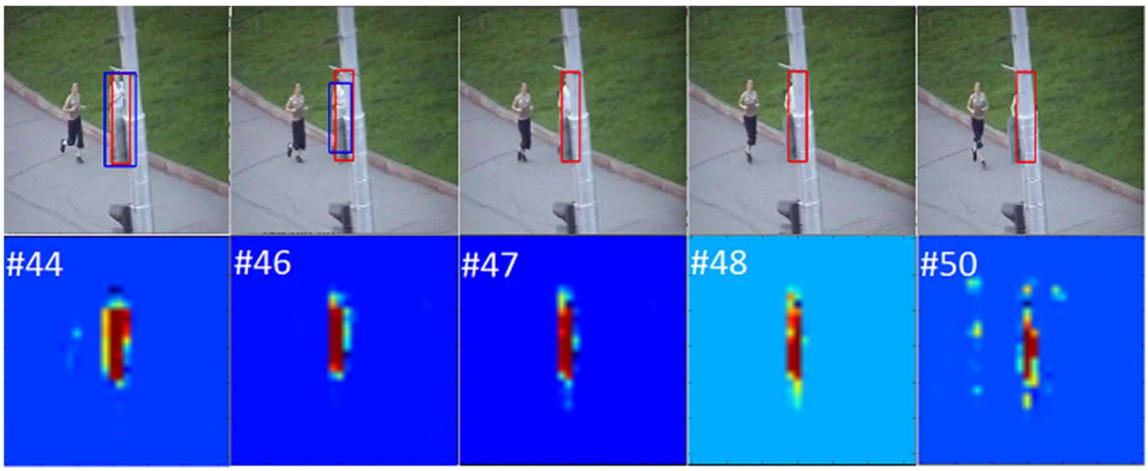

Рис. 1.20. Результат роботи системи ROLO

Іншими словами, за допомогою цього методу вдалось «навчити» алгоритм не втрачати об’єкт, що тимчасово перекривається іншими об’єктами (наприклад, автомобіль, що тимчасово заїхав за дерево).

1.8.4. Deep Sort

Подальшого розвитку ідея не втрачати об’єкт під час його тимчасового перекриття іншим об’єктом отримала в алгоритмі що був названий Deep Sort. Розглянемо короткі концепції та вникнемо у реалізацію.

Фільтр Калмана є найважливішим компонентом алгоритм. Інформація про об’єкт у кожному кадрі містить 8 змінних: (u, v, a, h, u’, v’, a’, h’) де (u, v) - центри обмежувальних рам, a - співвідношення сторін та h – висота зображення.

Для кожного виявленого об’єкту необхідно створити “трек”, який міститиме всю необхідну інформацію про стан об’єкту. У ньому також буде параметр для відстеження та видалення «треків», для яких останнє успішне розпізнавання було здійснено давно, оскільки ці об'єкти найімовірніше покинули сцену.

Тепер, коли у нас є обмежувальні рамки, що відстежуються фільтром Калмана, наступна проблема полягає у пов’язуванні нових виявлень із новими передбаченнями.

1.8.3. ROLO – «Recurrent Yolo»

Невеликі модифікації детектора YOLO та підключення повторюваного блоку LSTM в кінці допомагають відстежувати об’єкти, фіксуючи просторово-часові особливості.

Розпізнані з допомогою YOLO об’єкти (обмежувальні рамки) об'єднуються з вектором ознак на основі CNN (або повторно використаний YOLO, або спеціалізований екстрактор функцій). Об’єднаний вектор ознак, який представляє більшу частину просторової інформації, пов’язаний з поточним об’єктом, разом з інформацією про попередній стан передається в LSTM.

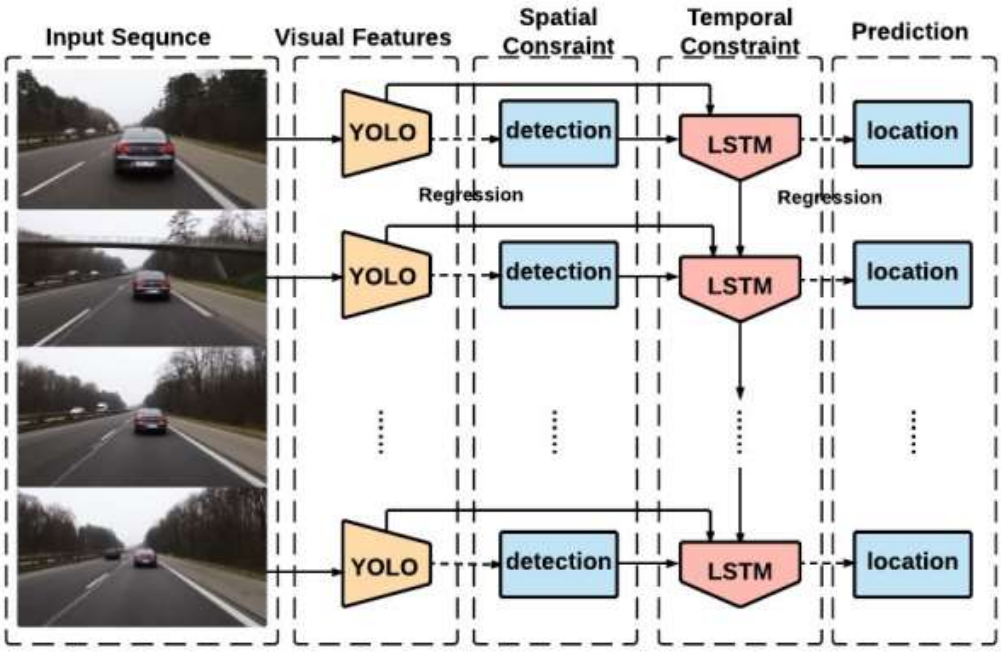

Рис. 1.19. Принцип роботи системи ROLO

На виході LSTM враховується як просторова, так і тимчасова інформація. Цей простий прийом для передбачення обмежувальних коробок дав значні покращення у відстеженні об’єктів.

фізики, враховуючи поточне виявлення, ми можемо зробити здогадку про те, де автомобіль буде в наступному кадрі.

Все це звучить ідеально, в ідеальному світі, але в реальності є проблеми:

1. Автомобілі не завжди їздять з постійною швидкість, а, отже, алгоритм буде помилятись.

2. Оскільки вихід детектора, на основі якого ми робимо прогнози, також не є точним, ця точність також знижує точність відстеження руху.

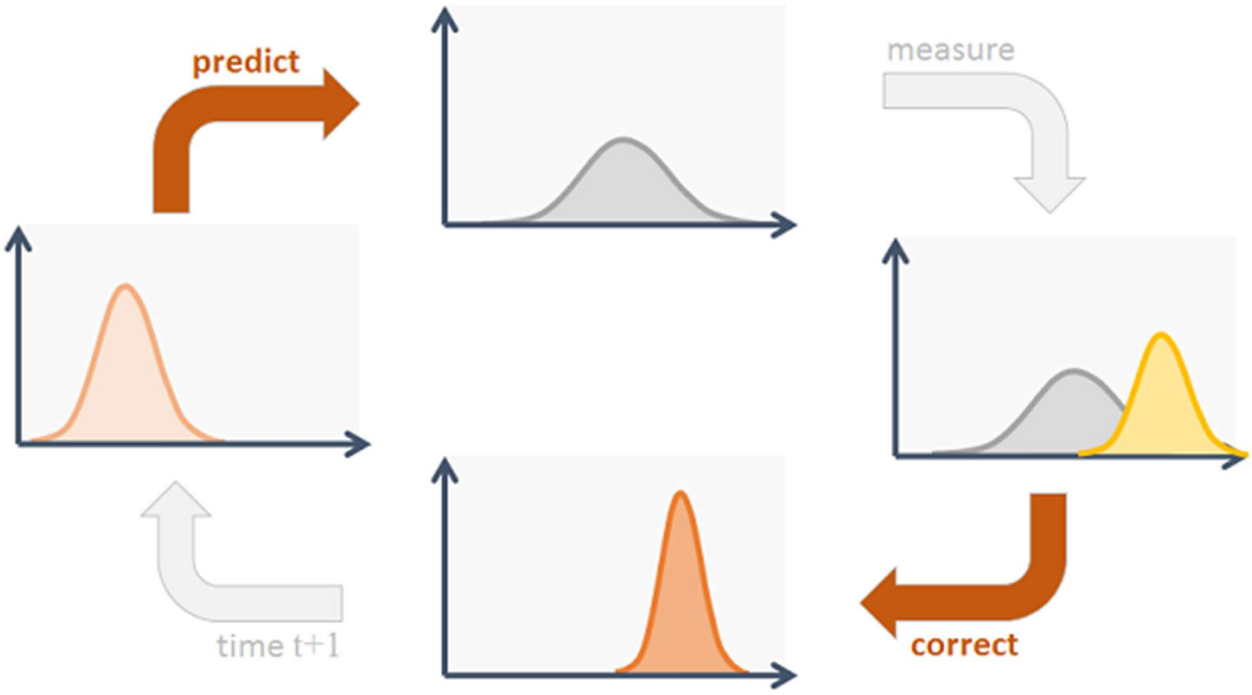

Рис. 1.18. Алгоритм роботи фільтрів Калмана

Як ми бачимо на наведеному малюнку, фільтр Калмана працює рекурсивно, де на першому кроці необхідно зчитати поточні показники, щоб передбачити поточний стан, а потім використати вимірювання та оновити прогнози. По суті, алгоритм зводиться до виведення нового розподілу з попереднього розподілу стану та розподілу вимірювань.

поточному кадрі. Тепер, коли ми маємо наступний кадр, де положення змінилось внаслідок переміщення об'єкта в кадрі, алгоритм MeanShift знаходить нове положення об’єкту з тими ж характеристиками.

1.8.2. Фільтри Калмана

Основна ідея фільтра Калмана полягає у використанні наявних виявлень та попередніх прогнозів для досягнення найкращої здогадки про поточний стан, зберігаючи при цьому можливість помилок у процесі.

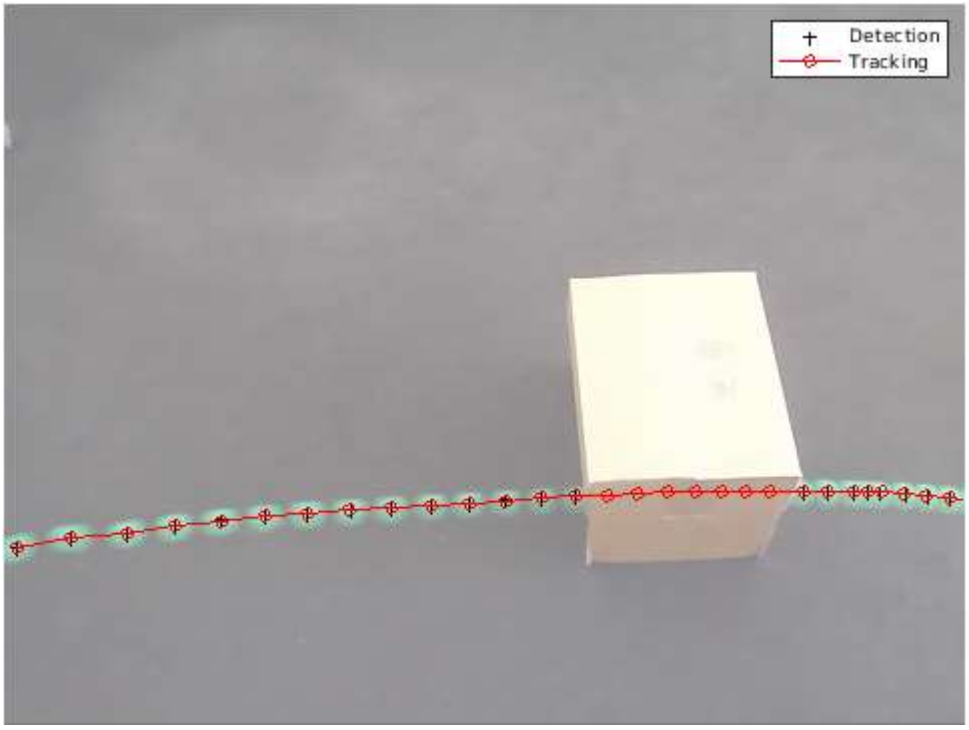

Рис. 1.17. Приклад відстежування об'єкту
з використанням фільтру Калмана

У нашому випадку припустимо, що ми маємо досить хороший детектор об’єктів, який виявляє автомобіль. Але він не на 100% точний, і іноді пропускає кадри, скажімо, в 1 з 10 випадків. Для ефективного відстеження та прогнозування наступного стану автомобіля, припустимо що той рухається з постійною швидкістю. Тепер, як тільки ми визначили просту модель відповідно до законів

1.8. Розпізнавання траєкторії руху об’єктів

Наступна проблема має таке формулювання: для кожного об’єкту одного кадру відеоряду необхідно знайти відповідність на наступному кадрі та отримати вектори траєкторії руху.

1.8.1. Метод Meanshift

Meanshift або «зсув середнього» - це популярний алгоритм, який в основному використовується при кластеризації та інших пов'язаних з ними проблемах. Він схожий на K-Means, але замінює просту центроїдну техніку обчислення центрів кластера на середньозважену, яка надає значення точкам, ближчим до середнього. Мета алгоритму - знайти всі зміщення в заданому розподілі даних. Крім того, цей алгоритм не вимагає оптимального значення “K”, як K-Means.

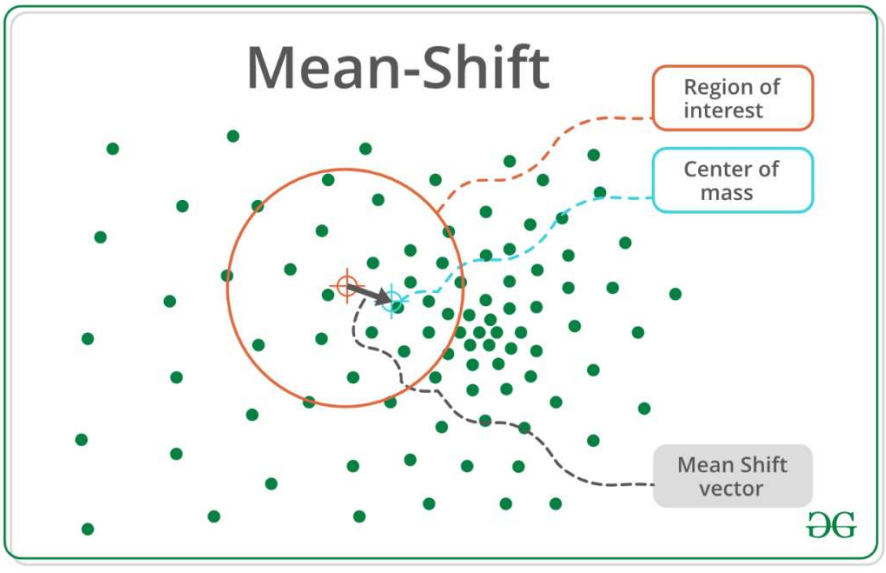

Рис. 1.16. Візуалізація роботи методу MeanShift

Коротко про алгоритм: припустимо, у нас є об’єкт у кадрі, і ми виділяємо з нього певні особливості (колір, текстуру, гістограму тощо). Застосовуючи алгоритм, ми маємо загальне уявлення про те, де подібні ознаки знаходяться у

собою набір складних високоякісних наборів даних для комп’ютерного зору, переважно найсучасніших нейронних мереж.

Рис. 1.15. Приклад розпізнавання на перехресті в м. Дніпро

Це ім'я також використовується для назви формату, що використовується цими наборами даних. COCO – це масштабний набір даних для виявлення, сегментації та титрування. COCO має кілька особливостей:

- Сегментація об’єктів;
- розпізнавання в контексті;
- 330 тис. зображень (> 200 тис. протеговано);
- 1,5 мільйонів екземплярів об'єктів;
- 80 категорій об'єктів.

Формат цього набору даних автоматично розуміється вдосконаленими бібліотеками нейронних мереж, наприклад, Detectron2. Існують навіть інструменти, створені спеціально для роботи з наборами даних у форматі COCO, наприклад COCO-анотатор та COCOapi. Розуміння того, як представлений цей набір даних, допоможе використовувати та модифікувати існуючі набори даних, а також створити власні.

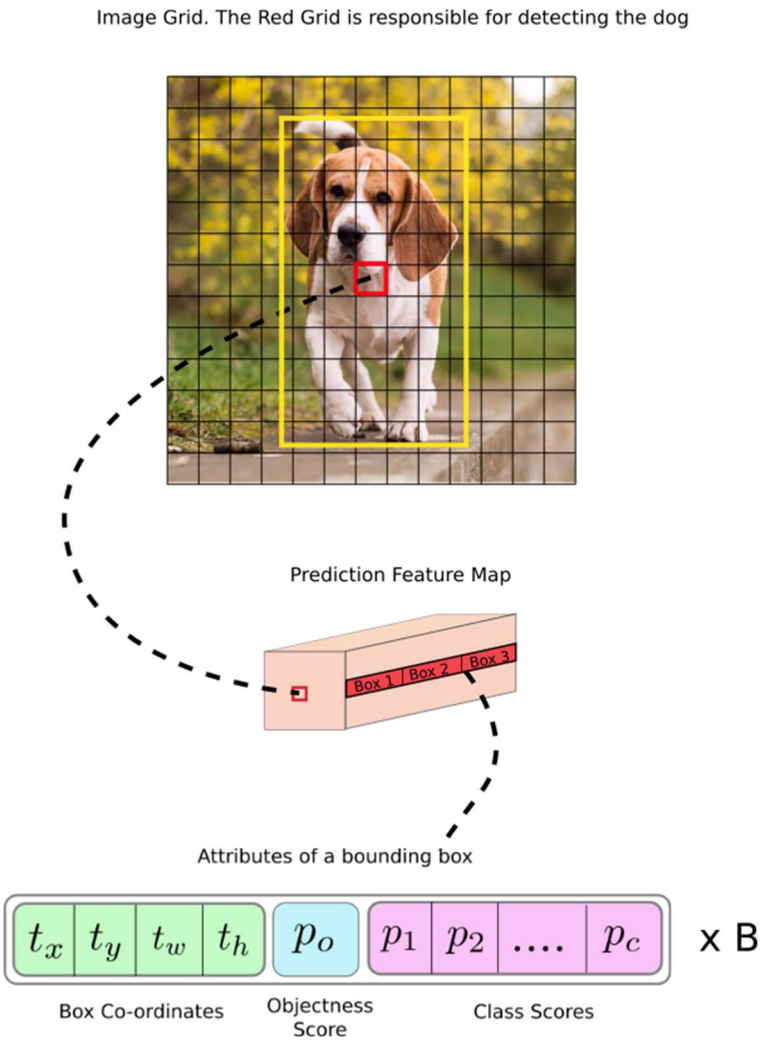

Рис. 1.13. Приклад роботи нейронної YOLO мережі

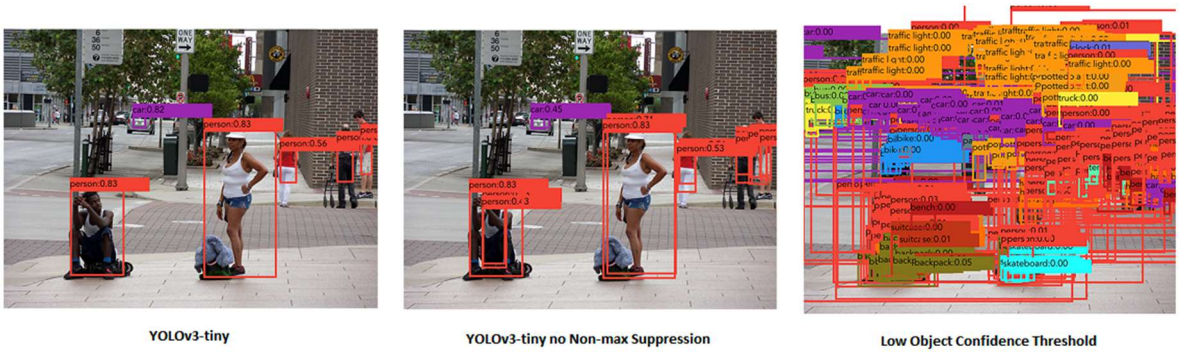

Рис. 1.14. Приклад розпізнавання за різних налаштувань

Один з найкращих датасетів для навчання такої нейронної мережі – COCO від Microsoft. Набір даних COCO, що означає «загальні об’єкти в контексті», являє

кілька прямокутників с досить високими можливостями, використовується досить простий алгоритм під назвою Non maximum suppression.

Порядок дії алгоритму такий:

1. Шукаємо bounding box з найбільшою ймовірністю приналежності до об'єкту.
2. Пробігаємо по всьому bounding box-ам які теж відносяться до цього об'єкту.
3. Видаляємо їх якщо Intersection over Union (IoU) з першим bounding box-му більше заданого порогу.

IoU вираховується за простою формулою (1.1).

$$\text{IoU} = \frac{\text{площа перетину}}{\text{площа об'єднання}} \tag{1.1}$$

На зображенні 15 представлені результати роботи YOLOv3-tiny з різними налаштуваннями IoU.

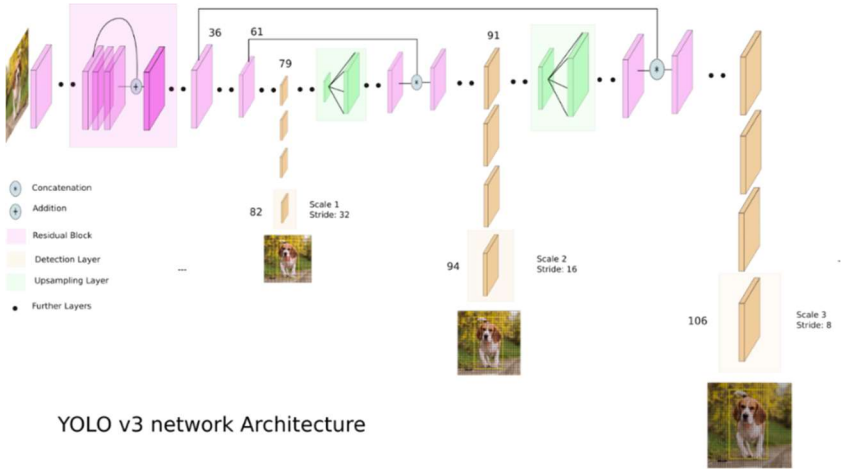

Рис. 1.12. Архітектура третьої версії нейронної YOLO мережі

За результатами тестів, такий тип моделі в 1000 разів швидший за традиційні CNN архітектури, а тому може працювати на досить повільному процессорі, як-от Raspberry PI, що вже підходить для використання в автономних детекторах.

ніс, щоб дізнатися, чи є вони на зображенні; це означає, що ми повинні раніше ідентифікувати лінії, границі, контури або фігури, які схожі на ті, які містять вухо або ніс, які ми бачили раніше. І саме цьому довірені шари згорткової нейронної мережі.

1.7.2.2. You Only Look Once (YOLO) архітектура

Як правило, у R-CNN мережах, з кожною ітерацією класифікатора робиться припущення щодо того, який тип об'єкта знаходиться в теперішній частині зображення. Таким чином створюються тисячі прогнозів для кожного зображення. Це гальмує процес, тому робота по визначенню класів відбувається досить неспішно.

Найбільше досягнення моделі YOLO, власне, віддзеркалено в назві – модель продивляється зображення «в цілому» та лише один раз. Ця модель накладає на зображення сітку, ділячи його на прямокутники. Кожен прямокутник намагається передбачити координати зони виявлення з оцінкою надійності цих полів та ймовірності класів. Оцінка достовірності для кожної зони виявлення значущості за вірогідністю класу, щоб отримати остаточну оцінку.

Рішення цієї проблеми пропонується у моделі YOLOv3, яка використовує три шари для розбиття зображення на різну сітку, розміри осередків цих сіток мають такі значення: 8, 16 і 32. Припустимо на вході у нас є зображення розміром 416x416 пікселів, тоді вихідні матриці (сітки) матимуть розмір 52x52 , 26x26 і 13x13 (416/8 = 52, 416/16 = 26 і 416/32 = 13) (див рис. 1.14).

Для базової мережі навченої на 80-ти класах, для кожного осередку сітки розбиття передбачається 3 bounding box-а, для кожного з них - 80 ймовірностей класів, числове значення ймовірності та 4 числа, що відповідають за положення і розмір прямокутника.

Після того як отримали координати і розміри bounding box-ів і відповідні ймовірності для всіх знайдених об'єктів на зображенні можна починати малювати їх поверх картинки. У випадку коли для одного об'єкта передбачено

об'єктів методом опорних векторів, і визначає, чи є той чи інший об'єкт в переданому регіоні.

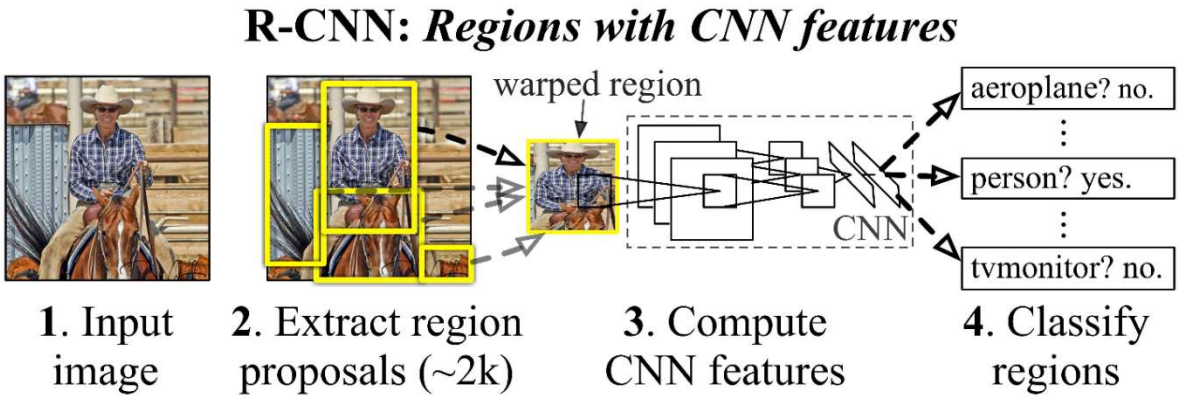

Рис. 1.11. Принцип роботи R-CNN мережі

Потім проводиться класифікація регіону методом лінійної регресії. Параметри, отримані з виходу блоку мережею, аналізуються на предмет того, наскільки даний регіон точно охоплює розпізнається об'єкт і наскільки потрібно його збільшити або зменшити, щоб робити це точніше [6]. Області з високим балом зображення вважаються виявленнями.

Та за точність цього алгоритму доводится поплатитись його швидкодією: потокове відео у 1080p вдасться обробити у 6 fps лише з допомогою потужного відеоприскорювача, що не підходить для автономних детекторів на перехрестях, оскільки значно збільшує їх вартість.

Щоб отримати інтуїтивне уявлення про те, як працюють ці нейронні мережі, варто звернути увагу на те, як люди розпізнають речі. Наприклад, якщо ми бачимо обличчя, ми його розпізнаємо, тому що у нього є вуха, очі, ніс, волосся тощо. Тоді, щоб вирішити, чи є щось обличчям, ми робимо це так, ніби ми маємо деякі уявні зони перевірки характеристик, які ми беремо до уваги. Іноді можна не бачити вуха людини, тому що воно закрите волоссям, але ми також класифікуємо обличчя з певною вірогідністю, тому що ми бачимо очі, ніс і вуста. Власне, саме по такому принципу працюють класифікатори на основі згорткових нейронних мереж. Але насправді, ми повинні спочатку знати, що таке вухо або

тому випадку, якщо шуканий об'єкт не дуже сильно відрізняється від зображень об'єктів, які були використані під час навчання, якщо ж відмінності сильні (кут нахилу, освітлення, щодо) тоді об'єкт взагалі не розпізнається[1].

У випадку задачі розпізнавання транспортних потоків, об’єкти на відеоряді будуть значно змінювати свій напрямок руху та кут відносно оптичного детектору, тому застосування таких алгоритмів розпізнавання не є оптимальним.

1.7.2.1. R-CNN мережі

Ще один спосіб розпізнавання образів, що працює у більш складних умовах – архітектура R-CNN (Regions With CNNs), що була розроблена для вирішення задачі класифікації зображень. R-CNN складається з трьох незалежних один від одного мереж: CNN, бінарного класифікатора (SVM) та лінійної регресії.

На вхід нейронній мережі подається частина вихідного зображення (регіон), яка, ймовірно, містить об'єкт. Ця мережа може обробляти зображення фіксованого розміру, тому всі регіони, які необхідно обробити, підганяються під певний розмір.

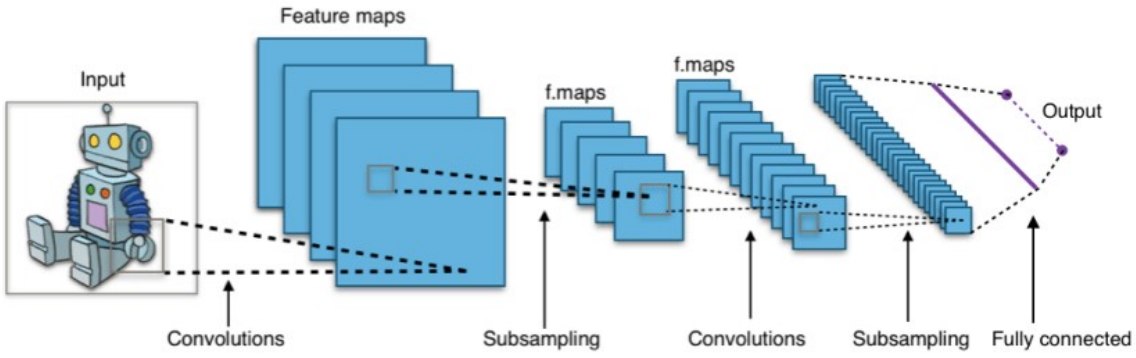

Рис. 1.10. Алгоритм роботи R-CNN мережі

На виході мережі отримується вектор ознак регіону, який подається на вхід SVM-мережі, яка в свою чергу проводить бінарну класифікацію за набором

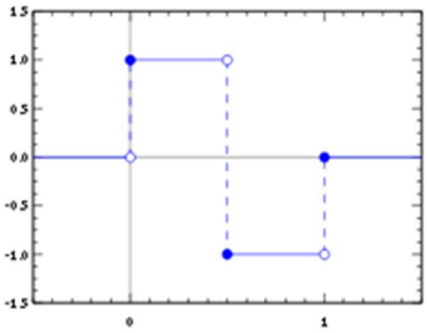

Рис. 1.8. Вейвлет

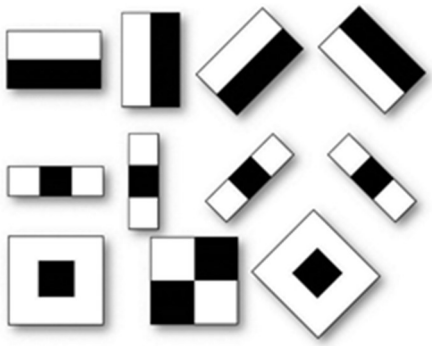

Рис. 1.9. Ознаки Хаара

Висока швидкість обробки зображень досягається за рахунок використання інтегрального представлення зображень: зберігається зображення представлено у вигляді матриці, кожен елемент якої дорівнює сумі всіх елементів, які розташовані вище і лівіше. Інтегральне уявлення корисно тим, що дозволяє швидко обчислювати суму деякого набору осередків матриці, незалежно від розміру обчислюється області, знаходження суми завжди займає константне час. Ознаки Хаара застосовуються до будь-якої області зображення, прямокутник з чорною і білою областю накладається як маска на зображення, знаходяться суми вмісту осередків темної і світлої області. Потім знаходиться різниця між темною і світлою областю, і в результаті виходить число - воно називається значенням ознаки.

Каскадні класифікатори, засновані на алгоритмі Віоли-Джонса мають як переваги, так і недоліки. Їх можна використовувати для обробки відео в реальному часі, але потрібні потужні системи. Працює алгоритм точно лише в

2. збільшити швидкість виробництва.

Потрібно врахувати, що обладнання може застосовуватись в місцях де складно розмістити оператора, та працювати без вихідних в автоматичному режимі, без втручання людини.

З попереднього пункту очевидно, що єдиним правильним рішенням буде використання алгоритмів на базі машинного навчання – так звані нейронні мережі. На зображенні можливо знайти шуканий елемент з використанням методу машинного навчання – за допомогою каскадного класифікатору і детектору на основі методу опорних векторів. В уже класичному OpenCV каскадний класифікатор побудований на основі дерев і реалізує алгоритм виявлення осіб Віоли-Джонса.

Метод складається з двох частин - навчання і розпізнавання. Перед тим, як почати обробляти цільові зображення, необхідно провести навчання і сформувати певну базу даних, що складається з ознак. В якості ознак використовуються ознаки Хаара, які були так названі через візуальні подібності з вейвлетом Хаара (рис. 1.8 і рис. 1.9).

Найпростішу прямокутну ознаку Хаара можна визначити як різницю між сумою пікселів двох суміжних областей всередині прямокутника, який може займати різні положення і масштаби на зображенні. Такий вид ознак називається 2-прямокутним. Алгоритм Віоли-Джонса визначає 3-прямокутні і 4-прямокутні ознаки[26]. Кожна ознака може показати наявність (або відсутність) будь-якої конкретної характеристики зображення, такий як кордону або зміна текстур. Наприклад, 2-прямокутна ознака може показати, де знаходиться межа між темним і світлим регіонами.

Головна перевага в тому, що каскади Хаара дуже швидко обраховуються через інтегральне представлення зображення. Що дає всього лише 4 звернення до пам'яті і 3 математичні дії для підрахунку суми всіх елементів прямокутника незалежно від його розміру. При розрахунку інших каскадів, відмінних від каскадів примітив Хаара, потрібна кількість дій пропорційна квадрату розміру примітиву.

Рис. 1.7. Метод віднімання фону. Зліва направо:
фон, кадр з об'єктом переднього плану, шукана маска об'єкта.

1.7.2. Методи на базі машинного зору

Машинний зір використовує аналіз зображень, щоб вирішувати промислові завдання. Він дозволяє значно підвищити продуктивність і якість продукції, яка виготовляється. Система здатна обробляти не тільки плоскі, але і об'ємні зображення.

Швидкість життя постійно збільшується, в наслідок чого збільшується швидкість виробництва товарів. Зі збільшенням швидкості виробництва продукції, постає питання якості виробленої продукції.

Системи технічного зору використовуються для:

1. перевірки якості;
2. виявлення браку;
3. правильного позиціонування деталей;
4. сортуванні продукції;
5. зчитування штрихкодів (QR-кодів);
6. робототехніки.

Для правильного підбору обладнання потрібно проаналізувати процес виробництва. Важливим є місце встановлення обладнання, сила вібрації, та зміна освітлення.

Застосування даних систем дає можливість:

1. підвищити якість виробництва (запобігти потраплянню браку на наступний етап виробництва, або до споживача);

руху. Отже, для створення оптичного детектору необхідно дослідити шляхи вирішення двох завдань:

1. Розпізнавання об’єктів на відеоряді та їх класифікація (транспорт, пішоходи, велосипедисти);
2. Моніторинг траєкторії руху об’єктів з одночасним трекінгом об’єктів між кадрами для визначення вектору руху об’єкту.

1.7. Алгоритми розпізнавання руху

Для завдання розпізнавання учасників дорожнього руху у відео потоці ключовим критерієм вибору алгоритму є швидкість обробки даних. Проаналізуємо існуючі методи розпізнавання образів.

1.7.1. Метод віднімання фонового зображення

Методи віднімання фону містять декілька варіацій, але всі вони мають спільну ідею – будь-яким способом виділити фон зображення, а потім, віднімаючи його з кадру з об'єктами, отримувати області, що містять рухомі об'єкти.

Незважаючи на простоту реалізації алгоритмів, вони мають особливість: виділення фону буде коректним, якщо камера, яка формує відеопотік, знаходиться в нерухомому стані [5]. В нашому випадку камера буде закріплена статично на світлофорному об’єкті, тому метод має право на життя.

Серед плюсів цього методу можна виділити простоту реалізації та автоматичне калібрування під зміну зовнішніх умов: освітлення, пори року, тощо. Основний мінус – метод «бачить» увесь рух, навіть у тих об’єктах, які не потрібні для задачі контролю транспортних потоків: деревах, кущах, тощо. Також метод не дозволяє автоматично класифікувати розпізнані об’єкти, тому потрібно все одно мати нейронну мережу, яка пост обробкою розпізнає об’єкти та надасть їх клас.

1.5.4. Акустичні детектори

Розпізнають автомобілі за допомогою чутливих мікрофонів, що вловлюють шум від працюючого двигуна. Основні переваги:

1. Пасивне виявлення об’єктів;
2. нечутливість до атмосферних явищ: опадів, снігу тощо;
3. у деяких моделях доступна багатосмугова експлуатація.

Основні недоліки:

1. Холод може впливати на точність підрахунку автомобілів;
2. неможливість визначення автомобілів, що зупинились.

Відповідно до дослідження, проведеного у 2010 році, точність розпізнавання автомобілів таким типом детектору складала від 85 до 97% для авто та близько 75% для пішоходів.

Висновок. Використання оптичних детекторів руху наразі обмежено, оскільки основу складають детектори що працюють на інших фізичних принципах, що працюють з точністю у межах від 75 до 95%, в залежності від складності оточуючого середовища та трафіку. Отже, використання оптичних методів розпізнавання учасників дорожнього руху може бути доцільним за не набагато меншого відсотку успішності розпізнавання об’єктів аніж у існуючих системах та за умови зниження таких вартості систем адаптивного контролю трафіку. Потенційно новий вид системи може надати можливість операторам дорожніх мереж за меншу суму грошей обладнати більшу кількість перехресть.

1.6. Методика оптичного розпізнавання учасників дорожнього руху

Дана методика є предметом дослідження цієї кваліфікаційної роботи. Для спрощення приймемо як факт такий формат даних, що детектори надсилають для роботи ATLCS системи: перелік географічних координат з метаданими для кожної з них: тип учасника дорожнього руху (авто, пішохід, тощо), його вектор

2. Обмежена класифікація: такий детектор не зможе відрізнити вантажний автомобіль від автобуса приблизно того ж розміру.

Відповідно до дослідження, проведеного у 2010 році, точність розпізнавання автомобілів таким типом детектору складала від 82% при трафіку у 250 авто на годину до 95% для 50 авто на годину [4].

1.5.3. Ультразвукові детектори

Як і мікрохвильові радари, базуються на фізичній властивості хвиль відображатись від об’єктів (рис. 1.6).

Основні переваги:

1. Можливість обробляти дані одночасно з декількох смуг руху [4].

Недоліки:

1. Ускладнено функціонування у місцях, де різниця між мінімальною та максимальною швидкістю автомобіля більше 20км/год.

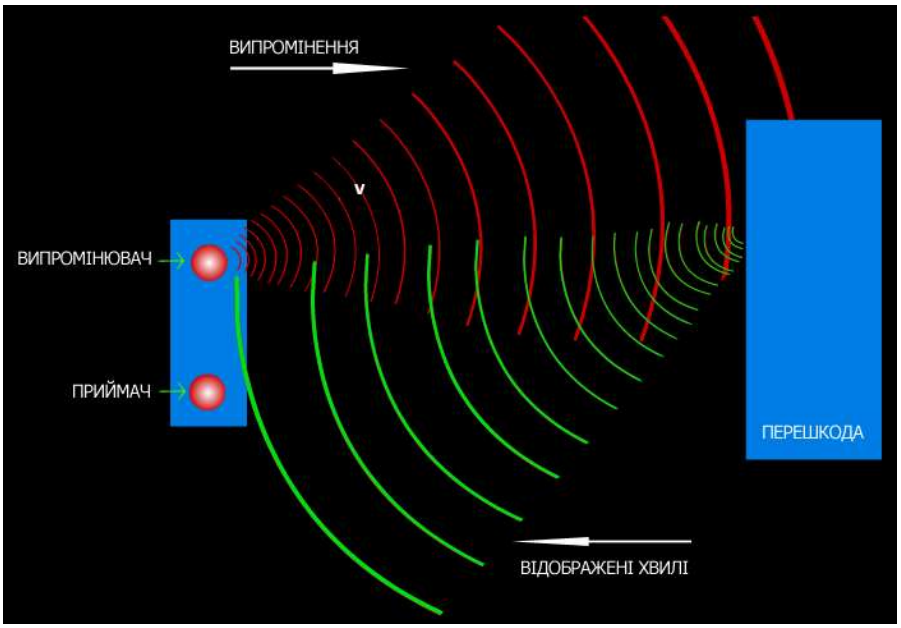

Рис. 1.6. Принцип роботи ультразвукового сенсору

постійне перебування під атмосферним впливом та безпосередній контакт з колесами автомобілів не найкращим чином впливає на тривалість роботи. Встановлення ж датчику у середину дорожнього одягу потребує, щонайменше, середнього ремонту дорожнього покриття, що збільшує вартість та тривалість робіт зі встановлення системи.

1. Фізична природа явища індукції не дозволяє одним датчиком «покрити» усю проїзну частину дороги, необхідно встановлення окремих індукційних петель на кожну смугу руху транспорту.

2. Можливість розпізнавання різних видів автомобілів (автобуси, вантажні та особисті авто) відсутня.

3. Відсутні можливості подальшого розвитку технології.

Помилкове виявлення транспортних засобів поза зоною виявлення називається «розбризкуванням» (англ. spashover). Ця проблема часто виникає, коли довгі петлі працюють на рівні чутливості, необхідному для виявлення невеликих транспортних засобів (наприклад, мотоциклів). За високої чутливості транспортні засоби із сусідніх смуг можуть бути помилково виявлені.

Відповідно до дослідження, проведеного у 2010 році, точність розпізнавання автомобілів таким типом детектору складала від 87% до 92% в залежності від кількості смуг руху [3].

1.5.2. Мікрохвильові радари

Базується на принципі відбиття мікрохвиль від об’єктів. Дозволяє отримувати інформацію про дальність, швидкість та розмір об’єкту.

Основні переваги:

1. Незалежність точності розпізнавання автомобілів від погодних умов: система не реагує на дощ, туман, ожеледицю, тощо.

2. Можливість обробляти дані одночасно з декількох смуг руху.

Недоліки:

1. Неможливість розпізнавати нерухомі об’єкти.

1.5. Основні переваги та недоліки існуючих рішень

Розглянемо декілька найбільш поширених видів детекторів дорожнього руху та сформулюємо проблеми, що існують. В подальшому в ході роботи буде запропоновано методи та підходи до розв’язання цих проблем.

1.5.1. Датчики на основі фізичних властивостей індукційної петлі

Принцип дії індукційного петлі полягає у фізичному явищі зміни індуктивності за рахунок впливу зовнішніх факторів (наприклад металевих предметів).

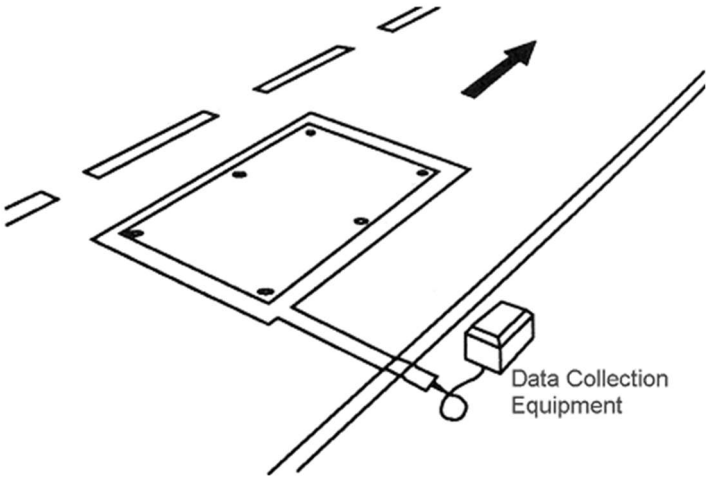

Рис. 1.5. Приклад індукційного детектору

Основними перевагами рішення можна визначити:

1. Дешевизна конструкції.
2. Незалежність точності розпізнавання автомобілів від погодних умов: система не реагує на дощ, туман, ожеледицю, тощо.
3. Наявність світового досвіду використання таких систем.

Але наявні також і недоліки:

1. Встановлення індукційної петлі може бути або поверх дорожнього одягу, або під ним. У першому випадку, хоч встановлення і потребує не так багато ресурсів, проте виникає проблема збереження датчику у робочому стані:

Механічні проблеми з датчиком контактної пластини призвели до введення електропневматичного датчика. Незважаючи на те, що цей пристрій знайшов деяке застосування, його було дорого встановлювати, він міг виявляти лише рухомі об’єкти, а його точність підрахунку була обмежена.

Найочевидніше рішення – детектор на основі ваги автомобіля – не став розповсюджений повсюди через проблеми в експлуатації. Снігоочисники піднімали плиту з проїжджої частини, що призводило до дорогого ремонту. Також були витрати на переустановлення детектора після зміни профілю дороги. Ці проблеми призвели до пошуку датчиків потоку руху на основі більш тонких властивостей, таких як:

1. Звук (акустичні датчики).
2. Непрозорість (оптичні та інфрачервоні датчики).
3. Геомагнетизм (магнітні датчики, магнітометри).
4. Відображення переданої енергії (інфрачервоний лазерний радар, ультразвукові датчики та мікрохвильові радарні датчики).
5. Електромагнітна індукція (індукційні детектори).
6. Вібрація (сейсмічні та датчики інерції).

Сьогодні індуктивний контурний детектор, безумовно, є найбільш широко використовуваним датчиком у сучасних системах управління дорожнім рухом [2]. Магнітометри, магнітні датчики, процесори відеозображень, мікрохвильові та лазерні радіолокаційні датчики, ультразвукові, акустичні та пасивні інфрачервоні датчики також випускаються в комерційних цілях і використовуються для різних програм управління дорожнім рухом. Оптичний датчик знайшов застосування для виявлення пріоритетних та перевантажених автомобілів.

1.3. Дані, що необхідні адаптивним алгоритмам для роботи

Усі ATLCS поєднує наявність датчиків зворотного зв’язку. Простіше кажучи, адаптивному алгоритму необхідно «знати» де і яка кількість автомобілів та пішоходів знаходиться та якими маршрутами рухається, аби в подальшому прогнозувати рух та коригувати фази світлофорних об’єктів. Такі датчики називаються детекторами дорожнього руху та працюють з допомогою багатьох різних фізичних принципів. Основні з них будуть розглянуті нижче.

1.4. Різновиди детекторів дорожнього руху

У 1920-х роках, коли ручну керовану дорожню сигналізацію замінювали на автоматичну, інженери зрозуміли, що їм потрібен метод збору даних про дорожній рух, що раніше отримував черговий на світлофорі. Першою розробкою був датчик, який спрацьовував, коли водій подавав звуковий сигнал на своєму автомобілі в спеціальному місці. Цей пристрій складався з мікрофона, встановленого в невеликій коробці. Вперше встановлений в 1928 році на перехресті Балтимора, пристрій Адлера дозволив вперше автоматичному контролеру визначати перевагу руху за допомогою датчика транспортного засобу.

Адлер продовжив свою роботу зі звуковими детекторами і в 1931 році представив інший звуковий детектор, який використовував порожнисті металеві коробки, вбудовані в дорожній одяг на перехресті. Ці коробки вловлювали звук проїжджаючих коліс, який передавався на мікрофони.

Приблизно в той же час Генрі А. Хо, інженер-електрик, розробив чутливий до тиску датчик з двох металевих пластини, які встановлювались на дорозі та виконували роль електричних контактів. Тиск колеса транспорту зближував пластини, що реєструвалось контролером. Цей тип датчику виявився більш популярним, ніж датчик, що активується гудком та користувався широким попитом понад 30 років як основний засіб виявлення транспортних засобів.

який наближається до перехрестя. Інші системи базують коригування на основі вимірювань від'їжджаючого трафіку, тобто зміни виконуються занадто пізно [22].

2. SCOOT повністю оптимізує тривалість циклу, розбиває та зміщує кожен цикл, роблячи невеликі контрольовані регулювання. Інші системи роблять масштабніші налаштування з набагато більшими інтервалами (до 10 хвилин), а деякі взагалі не оптимізують тривалість циклу. Це робить їх набагато менш чутливими до швидко мінливих умов дорожнього руху [23].

1.2.2.2. Метод SURTRAC

На відміну від інших систем, які можуть годинами реагувати на зміни трафіку, Surtrac адаптується в режимі реального часу до зміни трафіку, оптимізуючи потоки щосекунди. Surtrac координує транспортні потоки на складних мережах, а не лише на артеріях або коридорах з набагато менш динамічними схемами руху.

Доведено, що Surtrac забезпечує значне покращення порівняно зі звичайними системами синхронізації сигналів дорожнього руху та іншими адаптивними системами. Час у дорозі зменшується на 15%, час очікування на сигнали - на 20%, зупинок - на 10% та викидів на 25% відносно інших популярних систем адаптивного керування [52].

Основні переваги:

1. Люди добираються до місця призначення на 25% швидше, менше зупиняються зупинки, скорочується час очікування та збільшується швидкість подорожі;

2. зменшується на 30-40% кількість зупинок у місті, збільшується ресурс шин та зменшується споживання палива;

1.2.2. Мережеві методи

Мережеві, або централізовані методи контролю працюють на центральному сервері, який обробляє дані що надходять з безлічі перехресть міста і надсилає зворотні сигнали для перемикання фаз світлофорів. За рахунок централізації вдається досягти кращих результатів: регулювання з пріоретизацією складних популярних маршрутів (від центра міста до набережної, через міст, тощо) дозволяє влаштовувати «зелені хвилі» для трафіку, що позитивно відображається на пропускній здатності [1]. Найпопулярніші системи:

1. SCOOT від компанії Сіменс;
2. ATLCS від IEEE
3. SURTRAC

1.2.2.1. Метод SCOOT

Split Cycle Offset Optimisation Technique (SCOOT) – це адаптивна система управління дорожнім рухом в режимі реального часу для координації та контролю дорожньої сигналізації через міську дорожню мережу. Була розроблена Науково-дослідною лабораторією транспорту [45] для Департаменту транспорту США в 1980 році, дослідження та розробка SCOOT продовжуються наразі в компанії Sciemens[35]. SCOOT широко використовується всій Великобританії.

SCOOT автоматично регулює синхронізацію сигналів дорожнього руху відповідно до поточних умов дорожнього руху, використовуючи дані потоку від датчиків дорожнього руху. [34] Дані датчики зазвичай отримують з індуктивних петель на проїжджій частині, але все частіше використовуються інші форми виявлення.

Основні переваги:

1. SCOOT - це проактивна система, що означає, що вона налаштовує параметри сигналів руху в режимі реального часу для обслуговування трафіку,

максимальне значення. Тривалість періоду фаз вибирається між цими значеннями в залежності від характеру прибуття транспортних засобів до перехрестя. Після закінчення мінімального значення часу горіння зеленого сигналу тривалість може бути збільшена на невеликий додатковий інтервал часу в разі реєстрації нового автомобіля детектором транспорту на підході до перехрестя. При кожному спрацьовуванні детектора транспорту під час горіння зеленого сигналу, збільшує період горіння на додатковий інтервал, але при цьому значення періоду не може перевищити максимальну величину. Якщо жодного транспортного засобу за час чергового подовження зеленого сигналу зареєстровано не буде, то зелений сигнал змінюється на червоний. Додатковий інтервал визначається часом, який потрібен транспортному засобу, після його реєстрації детектором, на подолання відстані від детектора до стоп-лінії і подальшого безпечного проїзду перехрестя.

Принципом описаного методу управління є визначення інтервалів часу між послідовно прибувають до перехрестя транспортними засобами, і перемикання горіння зеленого сигналу світлофора на червоний в тому випадку, якщо цей інтервал перевищить додатковий інтервал. Управління на перехресті відбувається з повною адаптацією до транспортного попиту, якщо даний алгоритм використовується на всіх напрямках руху перехрестя. Якщо спостерігається мала інтенсивності руху на другорядних напрямках, то датчиками можна обладнати тільки ці напрямки і тоді таке управління називають напівадаптивним до транспортного попиту. Мінімальна і максимальна час горіння зеленого сигналу, а також додатковий час, який ще називається екіпажним, є керуючими параметрами, які повинні бути фіксовані в цьому алгоритмі. Їх можна визначити аналітично за допомогою методу Даррoха, але це не просте завдання завдання, ба більше, більш гнучким є алгоритм, за умови залежності керуючих параметрів від умов руху. В такому випадку використовується велика інформація про параметри потоку.

1.2. Різновиди методів адаптивного керування трафіком

1.2.1. Автономні методи керування

Для перехресть, що знаходяться на значному віддаленні один від одного підходить найпримітивніший вид автоматичного регулювання: автономний. Контролери працюють ізольовано один від одного, не обмінюючись інформацією та не оброблюючи дані з інших перехресть. Алгоритми ж керування можна умовно поділити на два види: «жорсткі» та адаптивні.

Цей клас працює на принципі попереднього задання керуючих параметрів: тривалості циклу регулювання та розподілення циклів протягом дня на основі попереднього аналізу характеристик транспортного потоку.

Адаптивні ж алгоритми додатково можна поділити на декілька підкласів.

1.2.1.1. Управління на основі довжини черги.

Для більш ефективного використання періодів горіння зеленого сигналу, можна мінімізувати і час перемикання сигналу світлофора з зеленого сигналу на червоний сигнал, але при цьому необхідно обмежити тривалість цього періоду, щоб оберегти від появи дуже довгих циклів регулювання і гарантувати можливість безпечного переходу доріг пішоходами. Данн і Поттс [6] узагальнили ці поняття і запропонували свій алгоритм управління. У цьому алгоритмі тривалість циклу підтримується відносно малою, збільшення дозволяється тільки при збільшенні інтенсивності руху

1.2.1.2. Управління на основі інтервалів

Інші назви цього методу: «світлофор, що приводиться в дію транспортним потоком» або «алгоритм пошуку розриву». На практиці цей метод використовується з 1930-х років. Його можна застосовувати при середній і низькій інтенсивності руху. Кожна фаза світлофора має мінімально і

У подальшому у Європі та США концепію прискорення руху транспорту довелось змінити. Замість будівництва нових магістралей вирішили максимально оптимізувати існуючі шляхи, збільшивши пропускну здатність. У випадку міського руху, «вузькими місцями», що зменшують пропускну здатність, здебільшого є перехрестя.

Тобто, оптимізувавши цикли світлофорного регулювання, можна досягти збільшення пропусної здатності, для чого використовують системи адаптивного керування світлофорами.

1.1. Історія адаптивних систем керування дорожнім рухом

Системи адаптивного керування світлофорами активно впроваджувались на території США та Європи ще у 1980х роках минулого століття. До цього ще на початку 1920 років в США були розроблені та встановлені перші примітивні контролери-автомати для переключення сигналів світлофорів. Повністю електромеханічні, напочатку вони містили в собі одну примітивну програму регулювання, що працювала протягом усього дня. Між тим очевидно, що інтенсивність руху суттєво змінюється протягом дня і одна програма не могла забезпечити оптимальне регулювання протягом ранішніх та вечірніх піків. Трохи пізніше технології дозволили вмістити в контролері цілих три програми, що дозволило розділити цикли світлофора на ранковий пік, денне плато та вечірній пік, пов’язані з денною робочою міграцією населення.

Багатопрограмне жорстке регулювання зменшує середній час очікування на перехресті, проте не є найоптимальнішим, бо не може обробляти випадкове прибуття транспорту або пішоходів до перехрестя (наприклад, у випадку ДТП, коли трафік перерозподіляється та змінює стандартний напрям руху). Вирішення проблеми стало можливим лише з появою більш потужних контролерів, що могли реалізувати обробку зворотного зв’язку від транспортного потоку.

Ця проблема добре ілюструється на рисунку нижче: трьохсмугова дорога не може пропустити більше автомобілів за одиницю часу аніж найвужче її місце (у прикладі – ремонтні роботи).

У випадку з широкими магістралями, «роль» найвужчого місця брали на себе з’їзди та подальші маршрути міськими дорогами, що проілюстровано на рисунках 1.2 та 1.3.

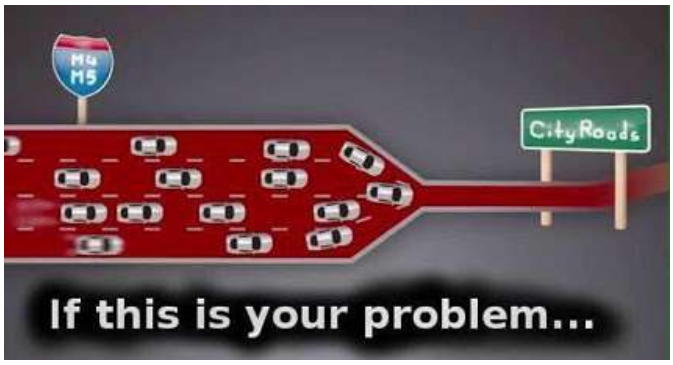

Рис. 1.3. Ілюстрація з'їзду з магістралі на вулиці міста до її розширення

У радянському союзі також існувала схожа концепція: магістралі безперервного руху. Восьмисмугові дороги мали пройти по центру міста. Через брак коштів та об’єктивні причини відсутності місця в центрі міста проект реалізувати не вдалось, а від мережі магістралей залишились окремі об’єкти.

Для Дніпра це пр. Слобожанський, Набережна Заводська, Донецьке шосе. Якщо придивитись, то кожен з цих транспортних коридорів має проблеми з заторами там, де через об’єктивні причини транспортні артерії звужуються: з’їзд з Нового мосту, проспект Свободи, тощо.

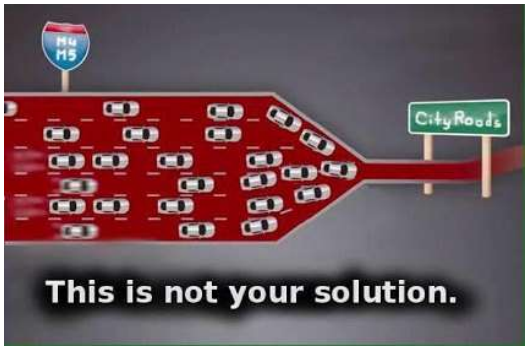

Рис. 1.4. Ілюстрація з'їзду з магістралі на вулиці міста після її розширення

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

Проблема оптимізації потоку трафіку не нова і має базу напрацьованих методів та алгоритмів. Так, найпершою та найочевиднішою ідеєю було розширення доріг – додаткові смуги давали швидкий ефект. Так по всьому світі поступово з’являлись десяти смугові магістралі в центрі міста, чотирьох рівневі розв’язки та підземні переходи.

Рис. 1.1. Розв'язка в Г'юстоні

Та з часом було отримано невтішні результати: розширення доріг призвело до зворотного ефекту: затори у середмісті збільшились. Пояснюється цей ефект простим правилом «вузького місця»: неможливо пропустити маршрутом більше одиниць транспорту за одиницю часу ніж дозволяє найвужче місце усього маршруту.

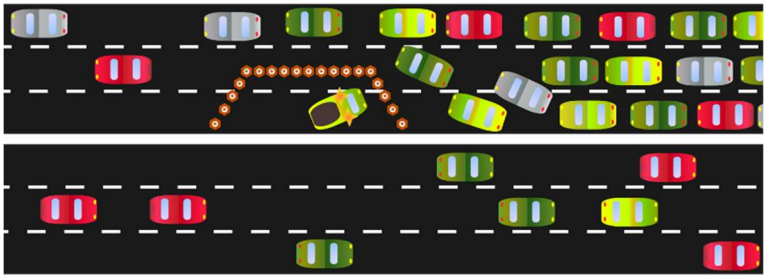

Рис. 1.2. Приклад "вузького місця"

3. розробці теоретичної частини роботи, в якій досліджені і систематизовані знання про існуючі підходи систем адаптивного керування трафіком.

Основні положення і результати магістерської роботи були покладені в основу тезису та обговорені на студентській науковій конференції.

Структура і обсяг роботи. Робота складається з вступу, чотирьох розділів і висновків. Містить 90 сторінок друкованого тексту, в тому числі 60 сторінок тексту основної частини з 34 рисунками, список використаних джерел з 68 найменуваннями на 4 сторінках, 3 додатки на 3 сторінках.

Завдання дослідження. Для виконання поставленої задачі у ході роботи були сформульовані й вирішені наступні завдання:

1. проаналізувати існуючі рішення для детекторів дорожнього руху (далі – ДР), визначити їх слабкі та сильні сторони, відсотковий показник коректності розпізнавання різних класів об’єктів;
2. проаналізувати існуючі технології та алгоритми розпізнавання об’єктів на відео, визначити ті, що підходять для вирішуваної задачі;
3. запроектувати та розробити демонстраційну модель оптичного детектору, що має успішно підраховувати кількість автомобілів та пішоходів, що перетинають перехрестя. Визначити ефективність моделі та порівняти її з існуючими рішеннями;
4. проаналізувати, як заміна детекторів на оптичний впливає на систему ATLC та як змінюється її ефективність.

Об’єктом досліджень є процес адаптивного управління дорожнім рухом.

Предметом досліджень є методи і алгоритми розпізнавання учасників дорожнього руху з використанням оптичних детекторів.

Методи дослідження. При вирішенні поставленої задачі використовувалися наукові досягнення у сферах розробки інформаційних систем, адаптивних алгоритмів і програмного забезпечення

Наукова новизна полягає в удосконаленні методів регулювання дорожнього руху за допомогою нових алгоритмів та принципів роботи детекторів автомобільного та пішохідного руху, заснованих на технології комп’ютерного бачення.

Практична цінність результатів полягає в тому, що запропоноване в роботі удосконалення дозволяє спростити встановлення та експлуатацію систем адаптивного керування трафіком без відчутного зменшення ефективності подібних систем.

Особливий внесок автора полягає у:

1. виборі методів досліджень і технологій реалізації;
2. створення демонстраційної моделі нового типу детекторів;

ВСТУП

Актуальність дослідження. Затори на дорогах поступово стають серйозною проблемою через велику кількість автомобілів. Кількість транспортних засобів, яка очікує на перехресті, різко зростає зі збільшенням потоку руху, і традиційні світлофори не можуть ефективно її пропустити. Неоптимальні цикли світлофорного регулювання викликають затримки міського транспорту, пішоходів, особистого транспорту та дратують учасників дорожнього руху. Також неоптимальне регулювання впливає на екологію: середньостатистичний автомобіль з двигуном типу ЕВРО-5 щохвилини викидає в повітря близько 1 граму шкідливих речовин: оксиди азоту, вуглекислий газ, сажа. Одночасна концентрація багатьох автомобілів в одному місці перед світлофором в результаті затору понаднормово збільшує забруднення повітря в конкретному місці. Так, за даними моніторингу, джерелом 78% викидів речовин-забрудників повітря у Києві 2018 року були автомобілі. Концентрації деяких забрудників у повітрі перевищують гранично допустимі рівні. У 2019 році у великих містах України були постійно перевищені концентрації діоксиду азоту, формальдегіду і оксиду азоту; час від часу – концентрації завислих речовин, оксиду вуглецю та фенолу. Різні дослідження оцінюють ризики від забруднення у 240-1900 додаткових випадків раку на один мільйон осіб протягом середньої тривалості життя (70 років).

Актуальність теми також підкріплюється статистичними опитуваннями. Згідно з даними моніторингового сервісу TomTom, українці витрачають у середньому на 46% більше часу в дорозі через затори. Більше половини людей (53%) TomTom Google заявляють, що страждають через проблеми забруднення повітря, а 45% водіїв вважають затори першочерговою проблемою, що має вирішити міська влада.

Метою дослідження є підвищення ефективності роботи системи керування дорожнім рухом за допомогою детекторів, що працюють на основі даних відеоспостереження.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

ATLCS – Adaptive traffic light control system

Алгоритм – порядок дій для досягнення результату

AI – Artificial intelligence (штучний інтелект)

SVM – support vector machine (метод опорних векторів)

RNN – Recurrent neural network (рекурентна нейронна мережа)

YOLO – You Only Look Once (назва алгоритму, дослівно «ти дивишся лише раз»)

ЗМІСТ

ABSTRACT

Explanatory note: 90 pages., 36 figures, 7 tables, 4 applications, 68 sources.

Object of research: the process of adaptive traffic control.

Subject of research: methods, algorithms, functioning and efficiency of algorithms for recognizing traffic flow.

Purpose of Master's thesis: improving the efficiency of the traffic control system by using detectors based on video surveillance data.

Research methods. Used methods to solve the problems: data analysis, pattern recognition theory in the field of computational intelligence, fuzzy set theory, object-oriented programming.

The scientific novelty of the results of the thesis is determined by the fact that for the first-time improved algorithms and principles of operation of car and pedestrian detectors based on computer vision technology, which significantly reduces the cost of adaptive traffic control systems.

The practical value of the results is that the proposed improvement improves the installation and operation of adaptive traffic management systems without significantly reducing the efficiency of such systems

In the Economics section calculations of the complexity of software development, the cost of creating software and the duration of its development, as well as marketing research of the market for the software product.

Keywords: video surveillance, traffic, traffic jams, cars, adaptive control, traffic lights.

РЕФЕРАТ

Пояснювальна записка: 90 стор., 36 рис., 7 таблиць, 4 додатка, 68 джерел.

Об'єкт дослідження: процес адаптивного управління дорожнім рухом.

Предмет дослідження: методи і алгоритми розпізнавання учасників дорожнього руху з використанням оптичних детекторів.

Мета магістерської роботи: підвищення ефективності роботи системи керування дорожнім рухом за допомогою детекторів, що працюють на основі даних відеоспостереження.

Методи дослідження. Для вирішення поставлених задач використані методи: аналізу даних, теорії розпізнавання образів з області обчислювального інтелекту, теорії нечітких множин, об'єктно-орієнтоване програмування.

Наукова новизна полягає в удосконаленні методів регулювання дорожнього руху за допомогою нових алгоритмів та принципів роботи детекторів автомобільного та пішохідного руху, заснованих на технології комп’ютерного бачення.

Практична цінність результатів полягає в тому, що запропоновані в роботі методи дозволяють спростити встановлення та експлуатацію систем адаптивного керування трафіком без відчутного зменшення ефективності подібних систем.

У розділі «Економіка» проведені розрахунки трудомісткості розробки програмного забезпечення, витрат на створення ПЗ і тривалості його розробки, а також проведені маркетингові дослідження ринку збуту створеного програмного продукту.

Список ключових слів: відеоспостереження, трафік, затори, автомобілі, адаптивне керування, світлофори.

- проведено аналіз ефективності використання штучної нейронної мережі для розпізнавання учасників дорожнього руху;

- проведено моделювання удосконаленої системи та визначено показники її ефективності відносно поточної версії.

4 ЕТАПИ ВИКОНАННЯ РОБІТ

Найменування етапів робіт	**Строки виконання робіт (початок – кінець)**
Збір інформації для дослідження предметної області	03.09.2020 – 30.09.2020
Дослідження методів для вирішення поставленого завдання	01.10.2020 – 31.10.2020
Експериментальні дослідження	01.11.2020 – 06.12.2020

Завдання видав _____ *Мороз Б.І.*
(підпис) (прізвище, ініціали)

Завдання прийняв до виконання _____ *Шишацький О.О.*
(підпис) (прізвище, ініціали)

Дата видачі завдання: *03.09.2020р.*

Термін подання кваліфікаційної роботи до ЕК *10.12.2020*

Міністерство освіти і науки України
Національний технічний університет
«Дніпровська політехніка»

ЗАТВЕРДЖЕНО:
Завідувач кафедри
Програмного забезпечення комп'ютерних систем
(повна назва)

_____ І.М. Удовик
(підпис) (прізвище, ініціали)

« » _____ 20 20 Року

ЗАВДАННЯ
на виконання кваліфікаційної роботи магістра

спеціальності *121 Інженерія програмного забезпечення*
(код і назва спеціальності)

студенту *121М-19-1* *Шишацькому Олександру Олександровичу*
(група) (прізвище та ініціали)

Тема кваліфікаційної роботи *Розробка програмного забезпечення для управління дорожнім рухом за даними відеоспостереження.*

1 ПІДСТАВИ ДЛЯ ПРОВЕДЕННЯ РОБОТИ

Наказ ректора НТУ «Дніпровська політехніка» від 22.10.2020 р. № 888-с

2 МЕТА ТА ВИХІДНІ ДАНІ ДЛЯ ПРОВЕДЕННЯ РОБІТ

Об'єкт досліджень – процес адаптивного управління дорожнім рухом.

Предмет досліджень – методи і алгоритми розпізнавання учасників дорожнього руху з використанням оптичних детекторів.

Мета роботи – підвищення ефективності роботи системи керування дорожнім рухом за допомогою детекторів, що працюють на основі даних відеоспостереження.

3 ОЧІКУВАНІ НАУКОВІ РЕЗУЛЬТАТИ

Наукова новизна полягає в удосконаленні методів регулювання дорожнього руху за допомогою нових алгоритмів та принципів роботи детекторів автомобільного та пішохідного руху, заснованих на технології комп'ютерного бачення.

Практична цінність результатів полягає в тому, що запропоновані в роботі методи дозволяють спростити встановлення та експлуатацію систем адаптивного керування трафіком без відчутного зменшення ефективності подібних систем. Також в ході експериментальних досліджень:

Міністерство освіти і науки України
Національний технічний університет
«Дніпровська політехніка»

Інститут електроенергетики
(інститут)

Факультет інформаційних технологій
(факультет)

Кафедра Програмного забезпечення комп'ютерних систем
(повна назва)

ПОЯСНЮВАЛЬНА ЗАПИСКА
кваліфікаційної роботи ступеня

магістра
(назва освітньо-кваліфікаційного рівня)

студента *Шишацького Олександра Олександровича*
(ПІБ)

академічної групи *121М-19-1*
(шифр)

спеціальності *121 Інженерія програмного забезпечення*
(код і назва спеціальності)

на тему: *Розробка програмного забезпечення для управління дорожнім рухом за даними відеоспостереження.*

_____ *О.О. Шишацький*

Керівники	Прізвище, ініціали	Оцінка за шкалою		Підпис
		рейтинговою	інституційною	
розділів кваліфікаційної роботи				
спеціальний	Проф. Мороз Б.І.			
економічний	Доц. Касьяненко Л.В.			
Рецензент	Проф. Корнієнко В.І.			
Нормоконтролер	Доц. Сироткіна О.І.			

Дніпро
2020