DAFTAR PUSTAKA

Amaluddin, F., Muslim, M. A., & Naba, A. (2015). Klasifikasi Kendaraan Menggunakan Gaussian Mixture Model (GMM) dan Fuzzy Cluster K Means (FCM). *Jurnal EECCIS (Electrics, Electronics, Communications, Controls, Informatics, Systems)*, *9*(1), 19–24.

Andrew, A., Buliali, J., & Wijaya, A. (2017). Deteksi Kecepatan Kendaraan Berjalan di Jalan Menggunakan OpenCV. *Jurnal Teknik ITS*, *6*. https://doi.org/10.12962/j23373539.v6i2.23489

Apostolopoulos, J., Tan, W.-T., & Wee, S. (2002). *Video streaming: Concepts, algorithms, and systems*.

Bovik, A. C. (2010). *Handbook of image and video processing*. Academic press.

Fajriyah, F., & Setiyono, B. (2016). Pengembangan Perangkat Lunak Deteksi Kecepatan kendaraan Bergerak Berbasis Pengolahan Citra Digital. *Skripsi. FMIPA Institut Teknologi Sepuluh Nopember, Surabaya*.

Gibran, M. K. (2012). *Pendeteksian Objek Yang Bergerak Dengan Metode Contour Menggunakan Software Visual Studio Dan Detektor Webcam*.

Gokule, R., & Kulkarni, A. (2014). Video based vehicle speed estimation and stationary vehicle detection. *Int J Adv Foundation Res Comput (IJAFRC)*, *1*(11), 93–99.

Gonzalez, R. C., & Woods, R. E. (2002). Digital image processing. upper saddle River. *J.: Prentice Hall*.

Huang, T. S. (1996). *Computer Vision: Evolution and Promise*.

Karim, M. R., & Dehghani, A. (2010). Vehicle speed detection in video image sequences using CVS method. *International Journal of the Physical Sciences*, *5*(17), 2555–2563.

Mukhopadhyay, J. (2011). *Image and Video Processing in the Compressed Domain*.

Putra, B. C. (2016). *Klasifikasi Kendaraan Bergerak Dengan Logika Fuzzy Berbasis Pengolahan Citra*. https://repository.its.ac.id/76012/

Putro, B., Furqon, M. T., & Wijoyo, S. H. (2018). Prediksi Jumlah Kebutuhan

Pemakaian Air Menggunakan Metode Exponential Smoothing. *Jurnal Pengembangan Teknologi Informasi Dan Ilmu Komputer*, *2*(11), 4679–4686.

Shirai, Y. (2012). *Three-dimensional computer vision*. Springer Science & Business Media.

Wicaksono, D. W. (2017). *Pengembangan Sistem Estimasi Kecepatan pada Kendaraan Bergerak Berbasis Pengolahan Citra Digital*. 127. http://repository.its.ac.id/2054/

Yilmaz, A., Javed, O., & Shah, M. (2006). Object tracking: A survey. *Acm Computing Surveys (CSUR)*, *38*(4), 13-es.

Yuwono, B. (2015). Image Smoothing Menggunakan Mean Filtering, Median Filtering, Modus Filtering Dan Gaussian Filtering. *Telematika*, *7*(1). https://doi.org/10.31315/telematika.v7i1.416

Lampiran 1 Lembar Perbaikan Skripsi

LEMBAR PERBAIKAN SKRIPSI

"SISTEM DETEKSI KECEPATAN KENDARAAN SECARA REAL-TIME DENGAN METODE GAUSSIAN MIXTURE MODELS (GMM)"

OLEH:

MUHAMMAD IKHWAN RAMADHANI
D121171512

Skripsi ini telah dipertahankan pada Ujian Akhir Sarjana pada tanggal 22 Januari 2024. Telah dilakukan perbaikan penulisan dan isi skripsi berdasarkan usulan dari penguji dan pembimbing skripsi.

Persetujuan perbaikan oleh tim penguji:

	Nama	Tanda Tangan
Ketua	Prof. Dr. Ir. Indrabayu, S.T., M.T., M.Bus.Sys., IPM.ASEAN. Eng.	
Sekretaris	Elly Warni, S.T., M.T.	
Anggota	Prof. Dr. Ir. Andani, M.T.	
	Muhammad Alief Fadhal Imran Oemar, S.T., M.Sc.	

Persetujuan perbaikan oleh pembimbing:

Pembimbing	Nama	Tanda Tangan
I	Prof. Dr. Ir. Indrabayu, S.T., M.T., M.Bus.Sys., IPM.ASEAN. Eng.	
II	Elly Warni, S.T., M.T.	

Lampiran 2 Source Code 1

```
from turtle import distance, width
import numpy as np
from tracker import *
import cv2
import math

# cap =
cv2.VideoCapture('rtsp://root:unhasvivo1@169.254.202.185:554/live.sd
p')
cap = cv2.VideoCapture("..\Video DSLR\Kecepatan20\Motor2.MOV")

# Tracker
tracker = EuclideanDistTracker()

# Subtractor -- History and varThreshold can change
mog2Subtractor = cv2.createBackgroundSubtractorMOG2(history=None,
varThreshold=None)

#Keeps track of what frame we're on
count = 0
center_points_prev_frame ={}
tracking_objects = {}
track_id = 0
V_dict = {}
# data = {}
alpha = 0.5
beta = 10

#Rumus Hitung Kecepatan
def countVelocity(cp_current, cp_prev , area):
    f = 28 #Motor2, Motor3, Motor4
    # f = 30 #Motor1L, Motor1P
    # f = 24 #Mobil1-Motor1, Mobil1
    rad = 57.2958
    tc = rad*(2*math.atan(35/(2*f))) #radtodegree
    H = 7200 #mm
    res = 720
    t = 1/59.94 #(1/fps)

    # tergantung mobil motor atau truk
    if (area < 10000):
        h = 1000
    elif (area < 30000):
        h = 2000
    else:
        h = 4000

    # Rumus mulai
```

```
    d = H*math.tan(67/rad) #Motor2, Motor3, Motor4
    # d = H*math.tan(60/rad) #Motor1L, Motor1P
    # d = H*math.tan(73/rad) #Mobil1-Motor1, Mobil1
    L = math.sqrt((H-h)**2 + d**2)
    P = 2*math.tan((tc/2)/rad)*L
    K = P/res
    D = math.sqrt(((cp_current[0]-cp_prev[0])**2)+((cp_current[1]-
cp_prev[1])**2))
    V = 0.0036*K*D/t #mm/s ke km/jam

    return V


while(True):
    # Capture frame-by-frame
    ret, frame = cap.read()

    #  Check if a current frame actually exist
    if not ret:
        break

    # Point Current Frame
    center_points_cur_frame = {}

    # Extract Region of Interest
    roi = frame[200:600,300:1100]

    # 1. Object detection
    # 1.1. Background Subtraction
    adjusted = cv2.convertScaleAbs(roi, alpha=alpha , beta=beta)
    mog2Mmask = mog2Subtractor.apply(adjusted)

    #1.2. Preprocessinng (Smoothing, Shadow Removal, Closing)
    kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (5, 5))
    mog2Mmask = cv2.medianBlur(mog2Mmask, 5) #Smoothing
    _, mog2Mmask = cv2.threshold(mog2Mmask, 128, 255,
cv2.THRESH_BINARY ) #Shadow removal
    mog2Mmask = cv2.morphologyEx(mog2Mmask, cv2.MORPH_CLOSE, kernel
,iterations=2) #Closing

    # 1.3. Deteksi dengan Contour
    contours, _ =cv2.findContours(mog2Mmask, cv2.RETR_TREE,
cv2.CHAIN_APPROX_SIMPLE)
    detections = []
    area_fix = 0
    for cnt in contours:
        area = cv2.contourArea(cnt)
        if 50000 > area > 1000:
            area_fix = area
```

```
            x, y, w, h = cv2.boundingRect(cnt)
            cv2.rectangle(roi, (x,y), (x+w, y+h), (0, 255, 0), 3)
            detections.append ([x, y, w, h])

    # 2. Object Tracker
    boxes_ids = tracker.update(detections)
    for box_id in boxes_ids:
        x , y , w , h , id =  box_id
        cx = int ((x + x + w)/2)
        cy = int ((y + y + h)/2)
        center_points_cur_frame[id] = (cx,cy)
        cv2.putText(roi, str(id), (x,y-15), cv2.FONT_HERSHEY_PLAIN,
1, (255,0,0), 2)
        if (count % 1 == 0):
            if center_points_prev_frame.get(id) and
center_points_cur_frame.get(id):
                V = countVelocity(center_points_cur_frame[id],
center_points_prev_frame[id], area_fix)
                V_dict[id] = V
                print (" V dari id(", id, ") adalah ",V)
                cv2.putText(roi, str("{:.2f}".format(round(V, 2))) +
' km/jam', (x,y-15), cv2.FONT_HERSHEY_PLAIN, 1, (255,0,0), 2)
                cv2.rectangle(roi, (x,y), (x+w, y+h), (0, 255, 0),
3)
                cv2.circle(roi, (cx, cy), 5, (0, 0, 255), -1)

    count += 1
    cv2.imshow('Normalisasi', frame)
    cv2.imshow('MOG2', mog2Mmask)
    cv2.imshow("ROI",roi)

    # make a copy of current frame as prev frame
    center_points_prev_frame = center_points_cur_frame

    key = cv2.waitKey(1)
    if key == 27:
        break

# When everything done, release the capture
cap.release()
cv2.destroyAllWindows()
```

Lampiran 3 Source Code 2

```
import math

class EuclideanDistTracker:
    def __init__(self):
        # Store the center positions of the objects
        self.center_points = {}
        # Keep the count of the IDs
        # each time a new object id detected, the count will
increase by one
        self.id_count = 0

    def update(self, objects_rect):
        # Objects boxes and ids
        objects_bbs_ids = []

        # Get center point of new object
        for rect in objects_rect:
            x, y, w, h = rect
            cx = (x + x + w) // 2
            cy = (y + y + h) // 2

            # Find out if that object was detected already
            same_object_detected = False
            for id, pt in self.center_points.items():
                dist = math.hypot(cx - pt[0], cy - pt[1])

                if dist < 25:
                    self.center_points[id] = (cx, cy)
                    print(self.center_points)
                    objects_bbs_ids.append([x, y, w, h, id])
                    same_object_detected = True
                    break

            # New object is detected we assign the ID to that object
            if same_object_detected is False:
                self.center_points[self.id_count] = (cx, cy)
                objects_bbs_ids.append([x, y, w, h, self.id_count])
                self.id_count += 1

        # Clean the dictionary by center points to remove IDS not
used anymore
        new_center_points = {}
        for obj_bb_id in objects_bbs_ids:
            _, _, _, _, object_id = obj_bb_id
            center = self.center_points[object_id]
            new_center_points[object_id] = center

        # Update dictionary with IDs not used removed
```

```
        self.center_points = new_center_points.copy()
        return objects_bbs_ids
```