LAMPIRAN

Glosarium

A

Aklimatisasi	:	Proses penyesuaian diri dengan lingkungan atau kondisi baru.
Algoritma	:	Urutan langkah-langkah yang digunakan untuk menyelesaikan suatu masalah atau melakukan tugas tertentu.
Alzheimer	:	Penyakit degeneratif otak yang menyebabkan penurunan daya ingat dan fungsi kognitif.
Anotasi	:	Komentar atau catatan tambahan yang ditulis untuk menjelaskan atau memberi informasi tambahan pada teks atau data.
Array	:	Struktur data yang menyimpan sekumpulan elemen dalam urutan yang teratur.
Artificial Intelligence	:	Kecerdasan buatan yang ditunjukkan oleh mesin, berbeda dari kecerdasan alami yang ditunjukkan oleh manusia dan hewan.
Auditori	:	Berhubungan dengan indra pendengaran.

B

Basal Otak	:	Bagian dasar otak yang terlibat dalam kontrol motorik dan proses lainnya.
Batch	:	Sekumpulan tugas atau pekerjaan yang diproses bersama-sama dalam satu kelompok.

C

C++ *language*	:	Bahasa pemrograman yang dikembangkan sebagai peningkatan dari bahasa C, sering digunakan untuk pengembangan sistem dan aplikasi perangkat lunak.
Confidence	:	Metrik yang menunjukkan keyakinan model bahwa prediksi objek yang terdeteksi dalam sebuah *bounding box* adalah benar

D

Data	:	Informasi yang dikumpulkan untuk dianalisis atau digunakan sebagai dasar pengambilan keputusan.
Dataset	:	Kumpulan data yang biasanya disusun dalam tabel atau format lain untuk dianalisis.
Defisit	:	Kekurangan atau kekurangan sesuatu, sering kali mengacu pada kekurangan dalam fungsi atau sumber daya.

E

Ekologi	:	Ilmu yang mempelajari interaksi antara organisme hidup dan lingkungan.
Ethernet	:	Teknologi jaringan yang digunakan untuk komunikasi data dalam jaringan lokal (LAN).

F

Fisiologi : Ilmu yang mempelajari fungsi dan mekanisme dalam organisme hidup.

Photocell : Alat yang mengubah cahaya menjadi sinyal listrik, sering digunakan dalam sensor cahaya.

Frame-By-Frame : Metode analisis atau pemrosesan video di mana setiap bingkai dianalisis atau diproses secara individual.

G

Genetika : Ilmu yang mempelajari gen, hereditas, dan variasi dalam organisme hidup.

Google Collab : Alat pengembangan berbasis *cloud* yang memungkinkan penulisan dan eksekusi kode Python dalam lingkungan *Jupyter Notebook*.

GPU : Unit pemrosesan grafis, perangkat keras yang dirancang untuk memproses grafik dan tugas komputasi paralel lainnya.

GUI : Antarmuka pengguna grafis, sistem interaksi antara pengguna dan komputer yang menggunakan elemen visual seperti ikon dan jendela.

H

Hardware : Komponen fisik dari sistem komputer, termasuk perangkat keras seperti CPU, memori, dan perangkat input/output.

Hipokampus : Bagian dari otak yang berperan penting dalam pembentukan memori dan navigasi ruang.

Human Eror : Kesalahan yang dilakukan oleh manusia, sering kali mengacu pada kesalahan dalam pengoperasian atau pengambilan keputusan.

I

Iluminasi : Penerangan dengan sinar matahari atau sinar buatan seperti lampu.

Input : Data atau informasi yang dibutuhkan oleh sebuah sistem untuk selanjutnya diproses sesuai dengan ketentuan proses yang telah ditentukan.

Interaktif : Secara umum memiliki arti komunikasi dua arah atau lebih dari komponen-komponen komunikasi.

Interoseptif : Kesadaran interoseptif adalah fokus utama dalam pendekatan ini di mana individu diajarkan kemampuan untuk menyambungkan pikiran dan tubuh dengan meningkatkan kesadaran terhadap sinyal internal.

Interpretasi : Proses pemberian pendapat, kesan, gagasan, serta pandangan secara teoritis pada sebuah objek tertentu yang berasal dari ide yang mendalam dan dipengaruhi oleh latar belakang dari orang yang menciptakan objek tersebut.

J

Javascript : Bahasa pemrograman populer yang digunakan untuk membuat *website* interaktif dan dinamis. Adapun dinamis yang dimaksud di sini berarti konten di dalamnya dapat bergerak dan tampil secara otomatis tanpa harus dimuat ulang manual oleh pengguna.

K

Kognisi : Serangkaian proses mental yang berkaitan dengan perolehan, penyimpanan, manipulasi, dan pengambilan informasi.

Kognitif : Suatu proses berpikir, yaitu kemampuan individu untuk menghubungkan, menilai dan mempertimbangkan suatu kejadian atau peristiwa.

Korteks Prefrontal : Bagian otak yang berfungsi untuk mengatur fungsi eksekutif, yaitu kemampuan merencanakan sesuatu, membuat keputusan, memecahkan masalah, mengontrol diri, mengingat instruksi, menimbang konsekuensi, dll.

M

Memori : Kemampuan untuk mengkode, menyimpan, mempertahankan dan mengingat informasi atau pengalaman masa lalu pada otak manusia. Sebagian besar informasi tersebut disimpan untuk kontrol masa yang akan datang pada aktivitas motorik dan untuk dipakai dalam pengolahan berpikir.

Microsd : Kartu memori non-volatile yang dikembangkan oleh SD *Card Association* yang digunakan dalam perangkat portabel.

N

Navigasi : Proses atau kegiatan mengarahkan atau menentukan jalan.

Neurologis : Cabang ilmu yang berhubungan dengan saraf dan sistem saraf.

Neurosains : Ilmu yang mempelajari sistem saraf, termasuk otak.

O

Observasi : Tindakan atau proses mengamati sesuatu dengan cermat.

Open Source : Perangkat lunak yang kode sumbernya tersedia untuk umum dan dapat diubah serta didistribusikan oleh siapa saja.

Operating System : Perangkat lunak yang mengelola perangkat keras komputer dan menyediakan layanan umum untuk program komputer.

P

Pixel	:	Elemen gambar terkecil dalam tampilan digital.
Preferensi	:	Pilihan atau kecenderungan terhadap sesuatu.
Probability	:	Kemungkinan atau peluang terjadinya suatu peristiwa.
Prototype	:	Model awal atau versi awal dari suatu produk yang akan dikembangkan.
Python language	:	Bahasa pemrograman tingkat tinggi yang mudah dibaca dan digunakan, mendukung berbagai paradigma pemrograman, serta memiliki banyak *library* dan framework untuk berbagai keperluan.

R

Real-Time	:	Pemrosesan data atau kejadian seketika saat data diterima.
Rgb	:	Model warna yang terdiri dari tiga warna dasar: Merah (Red), Hijau (Green), dan Biru (Blue).

S

Septum	:	Struktur anatomi yang memisahkan dua ruang atau bagian, sering kali merujuk pada bagian dalam tubuh.
Spasial	:	Berhubungan dengan ruang atau posisi dalam ruang.

T

Taktil	:	Berhubungan dengan indera peraba atau sentuhan.
Tensorflow	:	erangkat lunak open-source yang digunakan untuk machine learning dan pemrosesan data.
Tracker	:	Alat atau perangkat lunak yang digunakan untuk melacak atau memantau posisi atau aktivitas objek.
Transfer Learning	:	Teknik dalam *machine learning* di mana model yang telah dilatih pada satu tugas digunakan kembali untuk tugas lain yang terkait.
Transkripsi	:	Proses mengubah suara atau ucapan menjadi teks tertulis.

V

Virkon	:	Merk disinfektan yang efektif terhadap berbagai mikroorganisme.
Visual Studio Code	:	Editor kode sumber yang dikembangkan oleh Microsoft, digunakan untuk pengembangan perangkat lunak.
Visualisasi	:	Proses membuat gambaran visual dari data atau informasi untuk memudahkan pemahaman.
Vnc Viewer	:	Perangkat lunak yang memungkinkan pengguna untuk mengakses dan mengontrol komputer lain dari jarak jauh melalui jaringan.

W

Web Browser	:	Perangkat lunak yang digunakan untuk mengakses dan menampilkan halaman web di internet.

Website	:	Kumpulan halaman web yang saling terkait dan dapat diakses melalui internet.
Wireless	:	Teknologi komunikasi yang tidak menggunakan kabel fisik, sering kali mengacu pada jaringan nirkabel.

Dokumentasi

Gambar 1 Konsultasi bersama Ketua Lab Anatomi

Gambar 2 Konsultasi dengan pembimbing pertama

Gambar 3 Diskusi bersama kelompok MBKM terkait sistem monitoring tikus

Gambar 4 Pengujian pada hamster

Algoritma Perhitungan

```
import os
import cv2
import torch
import numpy as np
from ultralytics import YOLO
import time as t

class Tracking:
    def __init__(self, model_path, labels_path):
        self.device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
        self.model = YOLO(model_path)  # Load the model
        self.model.to(self.device)  # Move the model to the correct device
        self.labels = self.load_labels(labels_path)
        self.active = False  # Initialize active attribute
        self.tracking_data = []
        self.previous_positions = []
        self.total_distance = 0
        self.start_time = None
        self.num_left = 0
        self.num_right = 0
        self.arm_left_time = 0
        self.arm_right_time = 0
        self.prob_left_arm = 0
        self.prob_right_arm = 0
        self.pixel_to_meter_scale = 0.35  # Misalnya, 1 unit video = 35 cm = 0.35
meter

    def load_labels(self, labels_path):
        if not os.path.exists(labels_path):
            raise FileNotFoundError(f"Labels file not found: {labels_path}")
        with open(labels_path, 'r') as file:
            labels = file.read().strip().split('\n')
        return labels

    def start_tracking(self):
        self.tracking_data = []
        self.start_time = t.time()
        self.num_left = 0
        self.num_right = 0
        self.arm_left_time = 0
        self.arm_right_time = 0
        self.prob_left_arm = 0
        self.prob_right_arm = 0

    def stop_tracking(self):
        self.start_time = None

    def reset_tracking(self):
        self.tracking_data = []
        self.previous_positions = []
        self.total_distance = 0
```

```
        self.start_time = None
        self.num_left = 0
        self.num_right = 0
        self.arm_left_time = 0
        self.arm_right_time = 0
        self.prob_left_arm = 0
        self.prob_right_arm = 0

    def update_tracking_data(self, frame):
        results = self.model(frame)
        frame_wid = 480
        frame_hyt = 240

        # Define boundaries for left and right arms of the T
        top_left = (int(frame_wid * 0.3), int(frame_hyt * 0.4))
        top_right = (int(frame_wid * 0.7), int(frame_hyt * 0.4))
        bottom_center = (int(frame_wid * 0.5), int(frame_hyt * 0.4))

        # Draw boundaries on the frame
        cv2.rectangle(frame, (0, 0), (top_left[0], top_left[1]), (255, 0, 0), 2)  # Left arm
        cv2.rectangle(frame, (top_right[0], 0), (frame_wid, top_right[1]), (0, 0, 255), 2)  # Right arm
        cv2.line(frame, (0, bottom_center[1]), (frame_wid, bottom_center[1]), (0, 255, 0), 2)  # Horizontal line in the center

        if isinstance(results, list):
            results = results[0]

        predictions = results.boxes
        current_positions = []
        tracking_data = []
        for prediction in predictions:
            x1, y1, x2, y2 = prediction.xyxy[0].tolist()
            conf = prediction.conf[0].item()
            cls = prediction.cls[0].item()

            if conf < 0.6:
                continue

            centroid_x = (x1 + x2) / 2
            centroid_y = (y1 + y2) / 2

            if centroid_y > bottom_center[1]:
                continue

            class_name = self.labels[int(cls)]
            tracking_data.append({
                "class": class_name,
                "confidence": conf,
                "bounding_box": [x1, y1, x2, y2],
                "color": (0, 255, 0) if class_name == "right" else (255, 0, 0)
            })
```

```
                current_positions.append((centroid_x, centroid_y))

            # Update previous positions and total distance only if there is a change in
position
            if len(self.previous_positions) > 0 and len(current_positions) ==
len(self.previous_positions):
                for i in range(len(current_positions)):
                    distance = np.linalg.norm(np.array(current_positions[i]) -
np.array(self.previous_positions[i]))
                    # Check if the distance exceeds a threshold to consider it as
movement
                    if distance > 5:  # Adjust this threshold based on your specific
scenario
                        self.total_distance += distance * self.pixel_to_meter_scale

            self.previous_positions = current_positions
            self.tracking_data.extend(tracking_data)

            for prev_data in self.tracking_data:
                prev_center_x = prev_data["bounding_box"][0] +
(prev_data["bounding_box"][2] - prev_data["bounding_box"][0]) / 2
                prev_center_y = prev_data["bounding_box"][1] +
(prev_data["bounding_box"][3] - prev_data["bounding_box"][1]) / 2
                prev_cls = prev_data["class"]
                prev_prob = prev_data["confidence"]

                for curr_data in self.tracking_data:
                    curr_center_x = curr_data["bounding_box"][0] +
(curr_data["bounding_box"][2] - curr_data["bounding_box"][0]) / 2
                    curr_center_y = curr_data["bounding_box"][1] +
(curr_data["bounding_box"][3] - curr_data["bounding_box"][1]) / 2
                    curr_cls = curr_data["class"]

                    if prev_cls == curr_cls:
                        if prev_center_y > bottom_center[1] and curr_center_y <=
top_left[1]:
                            if curr_center_x < frame_wid / 2:
                                self.num_left += 1
                                self.arm_left_time += t.time() - self.start_time
                                self.prob_left_arm += prev_prob
                            elif curr_center_x > frame_wid / 2:
                                self.num_right += 1
                                self.arm_right_time += t.time() - self.start_time
                                self.prob_right_arm += prev_prob

            return tracking_data

        def calculate_distance(self, pos1, pos2):
            return np.linalg.norm(np.array(pos1) - np.array(pos2)) *
self.pixel_to_meter_scale

        def calculate_speed(self, distance, time_elapsed):
```

```
            return distance / time_elapsed if time_elapsed > 0 else 0

        def get_tracking_data(self):
            left_confidences = [data["confidence"] for data in self.tracking_data if data["class"] == "left"]
            right_confidences = [data["confidence"] for data in self.tracking_data if data["class"] == "right"]

            left_mean = np.mean(left_confidences) if left_confidences else 0
            right_mean = np.mean(right_confidences) if right_confidences else 0

            return {
                "left": left_mean,
                "right": right_mean,
            }

        def get_summary_details(self):
            total_time = t.time() - self.start_time if self.start_time else 0
            total_time = round(total_time, 2)

            avg_speed = self.total_distance / total_time if total_time > 0 else 0

            arm_left_time = self.arm_left_time
            arm_right_time = self.arm_right_time

            arm_left_time_seconds = round(arm_left_time, 2)
            arm_right_time_seconds = round(arm_right_time, 2)

            left_confidences = [data["confidence"] for data in self.tracking_data if data["class"] == "left"]
            right_confidences = [data["confidence"] for data in self.tracking_data if data["class"] == "right"]

            avg_confidence_left = np.mean(left_confidences) * 100 if left_confidences else 0
            avg_confidence_right = np.mean(right_confidences) * 100 if right_confidences else 0
            avg_speed = self.total_distance / total_time if total_time > 0 else 0

            return {
                "total_time": f"{total_time:.2f} seconds",
                "num_left_entries": self.num_left,
                "num_right_entries": self.num_right,
                "avg_confidence_left": f"{avg_confidence_left:.2f}%",
                "avg_confidence_right": f"{avg_confidence_right:.2f}%",
                #"arm_left_time": f"{arm_left_time_seconds:.2f} seconds",
                #"arm_right_time": f"{arm_right_time_seconds:.2f} seconds",
                "avg_speed": f"{avg_speed:.2f} m/s",
                "total_distance": f"{self.total_distance:.2f} meters"
            }

    def convert_frames_to_video(frame_list, output_file_path, frame_width, frame_height, frame_rate):
```

```
        out = cv2.VideoWriter(output_file_path, cv2.VideoWriter_fourcc(*'mp4v'), frame_rate, (frame_width, frame_height))
        for frame in frame_list:
            out.write(frame)
        out.release()

    def process_video(input_video_path, model_path, labels_path, output_video_path):
        tracker = Tracking(model_path, labels_path)
        cap = cv2.VideoCapture(input_video_path)

        frame_width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
        frame_height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
        frame_rate = int(cap.get(cv2.CAP_PROP_FPS))

        frame_list = []
        tracker.start_tracking()
        while cap.isOpened():
            ret, frame = cap.read()
            if not ret:
                break
            frame = cv2.resize(frame, (480, 240))
            tracker.update_tracking_data(frame)
            frame_list.append(frame)
        tracker.stop_tracking()

        convert_frames_to_video(frame_list, output_video_path, 480, 240, frame_rate)
        summary_details = tracker.get_summary_details()
        print("Summary Details:")
        for key, value in summary_details.items():
            print(f"{key}: {value}")

        cap.release()
        return summary_details

    if __name__ == "__main__":
        input_video_path = "sample_video.mp4"
        model_path = "best.pt"
        labels_path = "labels.txt"
        output_video_path = "output_video.mp4"

        summary_details = process_video(input_video_path, model_path, labels_path, output_video_path)
print(summary_details)
```