BAB 3
METODOLOGI PENELITIAN

3.1 Telaah Literatur

Pada tahap telaah literatur, hal yang dilakukan pertama kali adalah menganalisis jurnal penelitian sebelumnya yang dijadikan landasan untuk melakukan penelitian selanjutnya. Tahap analisis dilakukan untuk mempelajari urgensi penelitian sebelumnya, masalah yang dihadapi, solusi yang diberikan, dan batasan yang masih belum diselesaikan pada penelitian tersebut. Salah satu batasan yang ditemukan pada penelitian sebelumnya adalah sistem deteksi belum diuji pada kondisi di mana bagian tubuh dari seseorang yang diawasi terhalang oleh objek atau individu lain [11]. Penelitian ini dilakukan untuk melakukan pengujian batasan masalah tersebut dan mengoptimalkan sistem deteksi jatuh pada kondisi terhalang objek.

3.2 Penentuan Dataset

Dataset yang digunakan pada penelitian ini bersumber dari *Fall Detection Dataset* Le2i [15] yang berisi video simulasi kejadian jatuh dan aktivitas sehari-hari dalam berbagai skenario. Selain *dataset* dari Le2i, terdapat dataset yang dibuat mandiri untuk mendukung pengujian sistem deteksi jatuh pada kondisi terhalang objek.

Penggunaan *dataset* terbagi menjadi beberapa tahapan, yaitu:

1. Replikasi Algoritma

   Pada tahap replikasi algoritma, *dataset* yang digunakan berasal dari Le2i [15] dengan jumlah *dataset* yang dipilih sebanyak 130 video, sesuai dengan pengujian yang dilakukan oleh Tîrziu et al. [11]. *Dataset* digunakan untuk melakukan pengujian ketika replikasi algoritma agar dapat menghasilkan akurasi yang serupa dengan algoritma penelitian sebelumnya. Rincian *dataset* seperti berikut:

   - 99 Video kategori *Fall*.
   - 31 Video kategori *Non Fall*

2. Pengujian Limitasi

   Pada tahap pengujian limitasi dari penelitian sebelumnya, *dataset* yang digunakan merupakan gabungan dari Le2i dan dataset yang dibuat mandiri. *Dataset* yang digunakan sudah dipilih sesuai dengan limitasi, yaitu *dataset* dengan kondisi terhalangi objek. Jumlah *dataset* yang digunakan adalah 100 video dengan rincian sebagai berikut:

   - 58 Video kategori *Fall* dengan 29 *dataset* Le2i dan 29 *dataset* tambahan.
   - 42 Video kategori *Non Fall* dengan 20 *dataset* Le2i dan 22 *dataset* tambahan.

3. Evaluasi Algoritma Optimasi

   Pada tahap evaluasi algoritma yang sudah dioptimalkan, *dataset* yang digunakan sama seperti pada tahap replikasi algoritma untuk menguji performa dan pengaruh dari algoritma deteksi jatuh yang dioptimalkan.

3.3 Replikasi Algoritma

Replikasi algoritma dilakukan dengan menggunakan YOLOv7-W6-Pose mengimplementasikan algoritma, *threshold*, dan *fall decision* sesuai dengan penelitian yang dilakukan oleh Tîrziu et al. [11]. Algoritma yang diimplementasikan menggunakan *keypoints* sebagai informasi utama untuk mendeteksi postur tubuh manusia menggunakan fitur estimasi pose yang disediakan oleh YOLOv7-W6-Pose. Dengan menggunakan bantuan COCO sebagai sumber *pre-trained dataset* dan estimasi pose, algoritma yang terapkan mengekstrak beberapa titik tubuh yang ditetapkan sebagai kunci utama, yaitu terletak pada bahu kiri dan bahu kanan, *torso* kiri dan *torso* kanan, serta kaki kiri dan kaki kanan.

Berdasarkan hasil koordinat yang diekstrak dari sebuah *dataset* tersebut, algoritma akan melakukan perhitungan terhadap beberapa parameter kinematik dan geometrik, yaitu:

1. *Length Factor*: Menghitung rasio antara panjang tubuh bagian atas dan bagian bawah yang digunakan untuk mengukur perubahan postur yang terjadi secara vertikal.

2. *Aspect Ratio* (Rasio Tinggi terhadap Lebar): Digunakan untuk menentukan posisi tubuh seseorang, seperti posisi tegak atau mendatar. Pada kondisi jatuh, rasio ini akan cenderung rendah karena tubuh melebar secara horizontal.

3. Sudut Tubuh (*Angle*) — Dihitung antara bagian atas tubuh (bahu ke *torso*) dan bagian bawah tubuh (*torso* ke kaki). Hasil perhitungan yang menunjukkan bahwa sudut berada di posisi ekstreme dapat mengindikasikan perubahan mendadak pada posisi tubuh.

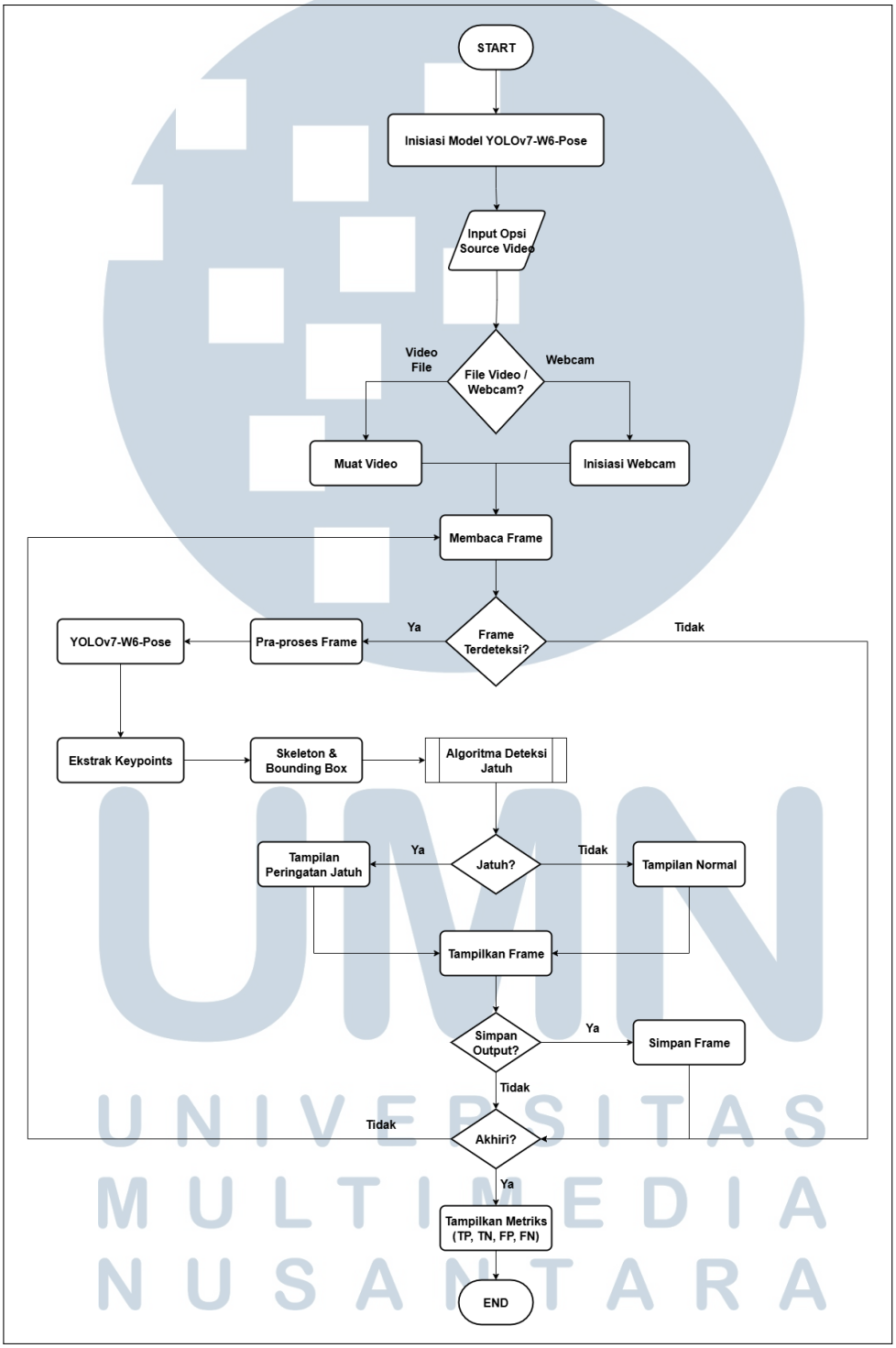
4. Kecepatan Vertikal (*Vertical Velocity*): Mengukur laju perubahan posisi vertikal tubuh yang terjadi antar *frame*. Nilai tinggi mengindikasikan bahwa telah terjadi gerakan jatuh secara tiba-tiba.

Berdasarkan hasil perhitungan dari algoritma yang direplikasi dari Tîrziu et al. [11], logika utama untuk menentukan sebuah *frame* dikategorikan sebagai *fall* adalah sebagai berikut:

- *Length Factor* menurun secara signifikan.
- *Aspect Ratio* menunjukkan tubuh dalam posisi datar atau horizontal.
- Sudut posisi tubuh melewati ambang tertentu.
- Kecepatan vertikal menunjukkan penurunan yang drastis (terjadi secara cepat).

Implementasi dari algoritma dan logika utama dilakukan pada setiap *frame*. Untuk mempertahankan stabilitas sistem deteksi, maka *buffer frame* pendek diterapkan untuk memvalidasi bahwa kejadian jatuh benar-benar terjadi, bukan disebabkan oleh fluktuasi pose sesaat. Replikasi sistem deteksi jatuh dilakukan dengan sepenuhnya merujuk pada algoritma yang digunakan oleh Tîrziu et al. [11]. Mengingat informasi yang dimiliki terkait *source code* asli yang digunakan terbatas, maka implementasi algoritma pada saat replikasi dilakukan berdasarkan pemahaman mendalam terkait algoritma pada jurnal tersebut, dengan penyesuaian struktur *output* YOLOv7-W6-Pose. Gambar 3.1 merupakan detail tahapan untuk alur dari replikasi algoritma deteksi jatuh. Alur dimulai dengan menginisiasikan YOLOv7-W6-Pose dan memproses *dataset* yang akan digunakan. Kemudian, sistem menentukan sumber penggunaan *dataset*, yaitu antara *video* atau *webcam* berdasarkan *input* yang diberikan. Selanjutnya, sistem akan memproses video dan membaca *frame* dari video yang sedang diuji. *Frame* yang diproses akan diekstrak oleh sistem agar mendapatkan *keypoints* dan visualisasi *bounding box*. Jika terdapat *frame* yang terdeteksi jatuh, maka sistem akan menampilkan peringatan adanya kejadian jatuh. Setelah video berhasil diproses, sistem akan menentukan pengujian

untuk lanjut atau sudah melalui *input*. Jika *input* adalah berhenti, sistem tidak akan memproses video lainnya dan langsung menghitung matriks (TP, TN, FP, FN).



Gambar 3.1. Detail Tahapan untuk Proses Pengembangan algoritma Replikasi

3.3.1 Implementasi Algoritma

Algoritma dan logika deteksi jatuh yang dibuat oleh Tîrziu et al. [11] kemudian diimplementasikan ke dalam bentuk *source* code. Gambar 3.2 merupakan tahap pertama dalam alur kerja sistem, yaitu tahapan proses inisialisasi. Pada tahap ini, YOLOv7-W6-Pose dimuat ke dalam sistem dengan mengeksekusi fungsi `FallDetector` yang secara otomatis akan memuat bobot model *(weight file)* dan menyiapkan sistem untuk deteksi jatuh.

```python
class FallDetector:
    def __init__(self, poseweights='yolov7-w6-pose.pt', device='0'):
        print(f"Initializing Fall Detector with weights: {poseweights} on device: {device}")

        # Memilih perangkat (CPU atau GPU)
        self.device = select_device(device)
        self.half = self.device.type != 'cpu'

        # Memuat model
        self.model = attempt_load(poseweights, map_location=self.device)
        self.model.eval()

        # Mendefinisikan parameter dan ambang batas
        self.LENGTH_FACTOR_ALPHA = 0.6
        self.VELOCITY_THRESHOLD = 0.8
        self.LEG_ANGLE_THRESHOLD = 50
        self.TORSO_ANGLE_THRESHOLD = 45
        self.ASPECT_RATIO_THRESHOLD = 0.9
        self.CONFIDENCE_THRESHOLD = 0.4
        self.MINIMUM_FALL_FRAMES = 8

        # ... (inisialisasi variabel pelacakan status lainnya)
```

Gambar 3.2. Inisialisasi Sistem dan Pemuatan Model

Pada Gambar 3.3, sistem melanjutkan untuk memilih penggunaan sumber video. Proses ini dilakukan oleh fungsi `cv2.VideoCapture()` dari pustaka OpenCV yang berfungsi untuk membaca sumber video. Sesuai dengan implementasi pada fungsi `run_fall_detection`, `cv2.VideoCapture()` dirancang secara fleksibel untuk menerima *input* berupa berkas video atau ID numerik dari perangkat *webcam* berdasarkan pilihan *input* yang diberikan oleh pengguna melalui fungsi `run_interactive()`.

```python
def run_fall_detection(poseweights='yolov7-w6-pose.pt', source='pose.mp4', ...):
    # ...

    # Parse sumber input
    input_path = source
    if source.isnumeric():
        input_path = int(source) # Menggunakan ID numerik untuk webcam

    # Membuka video capture
    cap = cv2.VideoCapture(input_path)
    if not cap.isOpened():
        # Penanganan error jika video tidak dapat dibuka
        error_msg = f"Error: Could not open video source {source}"
        print(error_msg)
        return
    # ...
```

Gambar 3.3. Pemilihan dan Pemuatan Sumber Video

Setelah sumber video berhasil ditentukan, sistem akan melanjutkan ke *looping* utama untuk memproses setiap *frame* dalam video yang dipilih. Fungsi `cap.read()` akan digunakan untuk memproses satu persatu *frame* dari aliran video. Jika sebuah *frame* berhasil diproses (ditandai dengan memberikan nilai `ret` yang bernilai *True*), maka *frame* tersebut akan diteruskan ke fungsi `detector.process_frame()` untuk menjalani tahap pra-pemrosesan, yang meliputi konversi ruang warna, penyesuaian ukuran, dan transformasi ke format *tensor*. Tahapan dapat dilihat pada Gambar 3.4.

```python
# Di dalam fungsi run_fall_detection()
while cap.isOpened():
    # Membaca Frame
    ret, frame = cap.read()
    if not ret:
        # Jika tidak ada frame lagi, keluar dari loop
        break

    frame_count += 1

    # Pra-Proses Frame (di dalam process_frame)
    # image = cv2.cvtColor(orig_image, cv2.COLOR_BGR2RGB)
    # image = letterbox(image, 640, stride=64, auto=False)[0]
    # image = transforms.ToTensor()(image)
    # ...

    # Memanggil fungsi yang melakukan langkah 4, 5, 6, 7
    processed_frame, is_fall, _, _ = detector.process_frame(frame)

    # ... (lanjutan loop)
```

Gambar 3.4. *Loop*, Pengambilan, dan Pra-Proses Frame

Fungsi `detector.process_frame(frame)` didefinisikan pada Gambar 3.5. Fungsi `process_frame` digunakan untuk memproses *frame* dengan menyalin *frame* asli agar tidak termodifikasi. Warna dari *Frame* yang disalin akan disesuaikan dengan warna standar dari model dengan `image = cv2.cvtColor`, yang kemudian akan dirubah ukuran gambar nya ke 640x640 dengan fungsi `image = letterbox(image, 640, stride=64, auto=False)[0]`.

```python
def process_frame(self, frame):
    # Menyalin frame asli agar tidak termodifikasi
    orig_image = frame.copy()
    # Mengonversi warna dari BGR (standar OpenCV) ke RGB (standar Model)
    image = cv2.cvtColor(orig_image, cv2.COLOR_BGR2RGB)

    # Mengubah ukuran gambar ke 640x640 dengan letterboxing untuk menjaga rasio aspek
    frame_height, frame_width = orig_image.shape[:2]
    image = letterbox(image, 640, stride=64, auto=False)[0]

    # Mengonversi gambar ke format Tensor PyTorch dan normalisasi (0-255 -> 0.0-1.0)
    image = transforms.ToTensor()(image)
    image = torch.tensor(np.array([image.numpy()]))
```

Gambar 3.5. Pra-proses *Frame*

Tahap selanjutnya yang dilakukan merupakan proses inti yang merepresentasikan implementasi algoritma deteksi jatuh pada Gambar 3.1.

Pada tahap ini, fungsi `detect_fall()` yang berfungsi sebagai inti pemrosesan dari sistem akan dieksekusi. Metode ini menerima koordinat *keypoints* dari setiap individu yang terdeteksi pada *frame* yang kemudian akan dianalisis untuk menentukan apakah seseorang telah terjatuh atau tidak berdasarkan postur dan gerakan yang telah didefinisikan sebagai kondisi dan logika algoritma. Proses dapat dilihat pada Gambar 3.6.

```python
def detect_fall(self, keypoints):
    # Definisi indeks keypoint
    NOSE, LEFT_SHOULDER, RIGHT_SHOULDER = 0, 5, 6
    LEFT_HIP, RIGHT_HIP = 11, 12
    LEFT_KNEE, RIGHT_KNEE = 13, 14
    LEFT_ANKLE, RIGHT_ANKLE = 15, 16

    # 1. Ekstraksi dan Validasi Keypoint
    # ... (Kode untuk mengekstrak koordinat dan memeriksa confidence score) ...

    # 2. Kalkulasi Kondisi-Kondisi Deteksi
    # Kondisi Ketinggian (height_cond)
    height_cond = min_shoulder_y >= (max_feet_y - self.LENGTH_FACTOR_ALPHA * Lfactor)

    # Kondisi Kecepatan (speed_cond)
    speed_cond = avg_speed >= self.VELOCITY_THRESHOLD

    # Kondisi Sudut (leg_angle_cond & torso_cond)
    leg_angle_cond = min(left_leg_angle, right_leg_angle) < self.LEG_ANGLE_THRESHOLD
    torso_cond = torso_angle > self.TORSO_ANGLE_THRESHOLD

    # Kondisi Rasio Aspek (aspect_cond)
    aspect_cond = orientation_ratio > self.ASPECT_RATIO_THRESHOLD

    # 3. Agregasi dan Pengambilan Keputusan
    conditions_met = sum([height_cond, speed_cond, leg_angle_cond, torso_cond, aspect_cond])
    is_fall = conditions_met >= 2

    self.fall_buffer.append(is_fall)
    final_detection = sum(self.fall_buffer) >= 2 if len(self.fall_buffer) >= 1 else is_fall

    return final_detection, current_state, conditions_info
```

Gambar 3.6. Eksekusi Algoritma Inti Deteksi Jatuh

Setelah melalui proses ektraksi *keypoints* dan analisis postur, tahap selanjutnya adalah memberikan *feedback* berupa visualisasi dari deteksi jatuh pada setiap *frame* dari video. Visualisasi yang diberikan berupa penggambaran kerangka tubuh (*skeleton*), *bounding box* dengan warna yang sesuai pada kondisi (merah untuk jatuh, hijau untuk normal), dan penyematan teks untuk interpretasi kondisi. Proses dapat dilihat pada Gambar 3.7.

```python
# Di dalam method FallDetector.process_frame(), setelah ekstraksi keypoint
# Loop untuk setiap orang yang terdeteksi
for idx in range(output.shape[0]):
    key_points = output[idx, 7:]

    # Memanggil algoritma deteksi jatuh
    person_fall, person_state, person_conditions = self.detect_fall(key_points)

    # Visualisasi berdasarkan hasil
    if person_fall:
        is_fall = True
        status_text = "FALL DETECTED!"
        cv2.putText(img, status_text, (50, 50), cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 0, 255), 2)
        cv2.rectangle(img, (int(xmin), int(ymin)), (int(xmax), int(ymax)), (0, 0, 255), 2)
    else:
        status_text = f"State: {person_state}"
        cv2.putText(img, status_text, (10, 60), cv2.FONT_HERSHEY_SIMPLEX, 0.6, (255, 255, 255), 1)
        cv2.rectangle(img, (int(xmin), int(ymin)), (int(xmax), int(ymax)), (0, 255, 0), 1)

    # Menggambar skeleton
    plot_skeleton_kpts(img, output[idx, 7:].T, 3)
```

Gambar 3.7. Analisis Jatuh dan Visualisasi Hasil

Setelah proses visualisasi selesai, tampilan visual dari *frame* akan ditampilkan kepada layar pengguna agar dapat melihat hasil dari proses deteksi jatuh. Terdapat opsi untuk menyimpan hasil dari proses visualisasi yang bisa dinonaktifkan. Pengguna juga dapat menekan tombol 'q' untuk memberhentikan proses. Apabila tidak ada interupsi, maka proses akan berlanjut ke video berikutnya dan akan mengulang proses pemrosesan *frame* hingga visualisasi. Proses dapat dilihat pada Gambar 3.8.

```python
# Di dalam loop 'while' pada fungsi run_fall_detection()
    # ... (setelah memanggil detector.process_frame())

    # Menampilkan frame ke layar
    if display:
        cv2.imshow('Fall Detection', processed_frame_resized)

    # Menyimpan frame ke video output
    if save_output and out is not None:
        out.write(processed_frame_resized)

    # Pengecekan kondisi akhir (input 'q' dari user)
    key = cv2.waitKey(1) & 0xFF
    if key == ord('q'):
        break
```

Gambar 3.8. Penayangan, Penyimpanan, dan Iterasi

Setelah seluruh *frame* dari video dan *looping* selesai diproses, sistem

akan menjalankan tahap finalisasi. Pada tahap ini, semua sumber daya yang digunakan untuk memproses video akan diberhentikan dan dilepaskan melalui fungsi `(cap.release()).` Seluruh *interface* yang dibuka oleh OpenCV akan ditutup. Apabila data *ground truth* tersedia, sistem akan menghitung metrik performa secara keseluruhan dan menampilkan hasilnya pada konsol sebelum sistem memberhentikan eksekusi. Tahapan ini dapat dilihat pada Gambar 3.9.

```python
# Setelah loop 'while' pada fungsi run_fall_detection() berakhir
cap.release()
if save_output and out is not None:
    out.release()
cv2.destroyAllWindows()

# ... (Kode untuk menghitung TP, TN, FP, FN, accuracy, precision, recall, f1_score) ...

# Menampilkan statistik akhir
if frame_count > 0 and not batch_mode and not interactive_mode:
    # ...
    if is_le2i:
        print("\\nDetection Metrics:")
        print(f"True Positives: {true_positives}")
        print(f"True Negatives: {true_negatives}")
        # ... (dan metrik lainnya)
```

Gambar 3.9. Agregasi dan Tampilan Metrik Akhir

3.3.2 Pengujian Algoritma Replikasi

Pengujian algoritma replikasi deteksi jatuh dilakukan dengan tiga tahap. Tahap pertama dilakukan dengan melakukan pengujian pada *dataset* Le2i dengan jumlah *dataset* sebanyak 99 video kategori *fall* dan 31 video kategori *non fall*. Pengujian replikasi algoritma menghasilkan nilai akurasi sebesar 92,31% atau 3,39% lebih kecil dari algoritma penelitian Tîrziu et al. [11]. Adapun hasil yang didapatkan menurun karena keterbatasan informasi terkait dengan parameter dan pilihan *dataset* yang digunakan untuk pengujian algoritma oleh Tîrziu et al. [11] sehingga implementasi dapat berbeda. Daftar penggunaan *dataset* dari pengujian algoritma replika dapat dilihat pada Tabel 3.1.

Tabel 3.1. Hasil Replika Algoritma dengan *Dataset* Le2i

Nomor	Nama Video	TP	TN	FP	FN
FALL Coffee_room_01					
1	video (1).avi	1	0	0	0
2	video (2).avi	1	0	0	0

Lanjut ke halaman berikutnya

Lanjutan dari Tabel 3.1

Nomor	Nama Video	TP	TN	FP	FN
3	video (3).avi	1	0	0	0
4	video (4).avi	1	0	0	0
5	video (5).avi	1	0	0	0
6	video (6).avi	1	0	0	0
7	video (7).avi	1	0	0	0
8	video (10).avi	1	0	0	0
9	video (11).avi	1	0	0	0
10	video (12).avi	1	0	0	0
11	video (13).avi	1	0	0	0
12	video (16).avi	0	0	0	1
13	video (20).avi	1	0	0	0
14	video (21).avi	1	0	0	0
15	video (22).avi	1	0	0	0
16	video (23).avi	1	0	0	0
17	video (25).avi	1	0	0	0
18	video (26).avi	1	0	0	0
19	video (27).avi	1	0	0	0
20	video (28).avi	1	0	0	0
21	video (31).avi	0	0	0	1
22	video (32).avi	1	0	0	0
23	video (33).avi	1	0	0	0
24	video (35).avi	1	0	0	0
25	video (36).avi	1	0	0	0
26	video (38).avi	1	0	0	0
27	video (39).avi	1	0	0	0
28	video (41).avi	1	0	0	0
29	video (42).avi	1	0	0	0
30	video (43).avi	1	0	0	0
31	video (44).avi	1	0	0	0
32	video (45).avi	1	0	0	0
33	video (46).avi	1	0	0	0
34	video (47).avi	1	0	0	0
35	video (48).avi	1	0	0	0

Lanjut ke halaman berikutnya

Lanjutan dari Tabel 3.1

Nomor	Nama Video	TP	TN	FP	FN
FALL Coffee_room_02					
36	video (49).avi	1	0	0	0
37	video (51).avi	1	0	0	0
38	video (52).avi	1	0	0	0
39	video (53).avi	1	0	0	0
40	video (55).avi	1	0	0	0
41	video (56).avi	1	0	0	0
42	video (57).avi	1	0	0	0
43	video (59).avi	0	0	0	1
44	video (60).avi	1	0	0	0
45	video (62).avi	1	0	0	0
46	video (64).avi	1	0	0	0
FALL Home_01					
47	video (1).avi	1	0	0	0
48	video (2).avi	1	0	0	0
49	video (3).avi	1	0	0	0
50	video (4).avi	1	0	0	0
51	video (5).avi	1	0	0	0
52	video (6).avi	1	0	0	0
53	video (8).avi	1	0	0	0
54	video (9).avi	1	0	0	0
55	video (10).avi	1	0	0	0
56	video (15).avi	0	0	0	1
57	video (18).avi	1	0	0	0
58	video (19).avi	1	0	0	0
59	video (20).avi	1	0	0	0
60	video (21).avi	1	0	0	0
61	video (22).avi	1	0	0	0
62	video (23).avi	1	0	0	0
63	video (24).avi	1	0	0	0
64	video (25).avi	1	0	0	0
65	video (26).avi	1	0	0	0
66	video (27).avi	1	0	0	0

Lanjut ke halaman berikutnya

Lanjutan dari Tabel 3.1

Nomor	Nama Video	TP	TN	FP	FN
67	video (28).avi	1	0	0	0
68	video (29).avi	1	0	0	0
69	video (30).avi	1	0	0	0
FALL Home_02					
70	video (31).avi	1	0	0	0
71	video (32).avi	1	0	0	0
72	video (33).avi	1	0	0	0
73	video (34).avi	1	0	0	0
74	video (35).avi	0	0	0	1
75	video (36).avi	0	0	0	1
76	video (37).avi	1	0	0	0
FALL Lecture_room					
77	video (2).avi	1	0	0	0
78	video (3).avi	1	0	0	0
79	video (4).avi	1	0	0	0
80	video (5).avi	1	0	0	0
81	video (6).avi	1	0	0	0
82	video (7).avi	1	0	0	0
83	video (8).avi	1	0	0	0
84	video (10).avi	0	0	0	1
85	video (11).avi	0	0	0	1
86	video (12).avi	1	0	0	0
87	video (13).avi	1	0	0	0
88	video (14).avi	1	0	0	0
FALL Office					
89	video (2).avi	1	0	0	0
90	video (3).avi	0	0	0	1
91	video (4).avi	1	0	0	0
92	video (6).avi	1	0	0	0
93	video (7).avi	1	0	0	0
94	video (12).avi	1	0	0	0
95	video (13).avi	1	0	0	0
96	video (14).avi	1	0	0	0

Lanjut ke halaman berikutnya

Lanjutan dari Tabel 3.1

Nomor	Nama Video	TP	TN	FP	FN
97	video (15).avi	1	0	0	0
98	video (16).avi	1	0	0	0
99	video (17).avi	1	0	0	0
NON FALL Coffee_room_02					
100	video (63).avi	0	1	0	0
101	video (66).avi	0	1	0	0
102	video (67).avi	0	1	0	0
103	video (69).avi	0	1	0	0
NON FALL Home_02					
104	video (38).avi	0	1	0	0
105	video (39).avi	0	1	0	0
106	video (40).avi	0	1	0	0
107	video (41).avi	0	1	0	0
108	video (42).avi	0	1	0	0
109	video (43).avi	0	1	0	0
110	video (44).avi	0	1	0	0
111	video (45).avi	0	1	0	0
112	video (46).avi	0	1	0	0
113	video (47).avi	0	1	0	0
NON FALL Lecture_room					
114	video (15).avi	0	1	0	0
115	video (16).avi	0	1	0	0
116	video (18).avi	0	1	0	0
117	video (19).avi	0	1	0	0
118	video (20).avi	0	1	0	0
119	video (21).avi	0	1	0	0
120	video (22).avi	0	0	1	0
121	video (23).avi	0	1	0	0
122	video (25).avi	0	1	0	0
123	video (26).avi	0	1	0	0
NON FALL Office					
124	video (18).avi	0	1	0	0
125	video (21).avi	0	1	0	0

Lanjut ke halaman berikutnya

Lanjutan dari Tabel 3.1

Nomor	Nama Video	TP	TN	FP	FN
126	video (22).avi	0	1	0	0
127	video (25).avi	0	1	0	0
128	video (26).avi	0	1	0	0
129	video (28).avi	0	1	0	0
130	video (29).avi	0	1	0	0

Matriks evaluasi dari pengujian replikasi algoritma menunjukkan tingkat akurasi sebesar 92,31% dengan penggunaan *dataset* sebanyak 130 video. Tingkat *precision* dari pengujian mendapatkan hasil sebesar 98,90%, *recall* sebesar 90,91%, dan *F1-Score* sebesar 94,74%. Hasil matriks evaluasi dapat dilihat pada Tabel 3.2.

Tabel 3.2. Matriks Evaluasi Pengujian Replikasi Algoritma

Hasil	Nilai
Total Video	130
True Positive (TP)	90
True Negative (TN)	30
False Positive (FP)	1
False Negative (FN)	9
Accuracy	92,31%
Precision	98,90%
Recall	90,91%
F1 Score	94,74%

3.4 Optimasi Algoritma

Bagian ini menjelaskan langkah-langkah yang dilakukan sebelum dan saat melakukan optimasi algoritma untuk deteksi jatuh. Tahap pertama yang dilakukan adalah dengan melakukan pengujian algoritma replikasi dengan *dataset* sesuai dengan topik limitasi, yaitu kondisi terhalang objek. Setelah melakukan pengujian pada algoritma replikasi, langkah selanjutnya adalah mengoptimalkan algoritma deteksi jatuh untuk mengurangi hasil *dataset* yang terevaluasi sebagai *False Positives* dan *False Negatives*. Setelah algoritma dioptimalkan, pengujian dilakukan kembali terhadap *dataset* limitasi, yang diakhiri dengan pengujian algoritma yang sudah dioptimalkan dengan *dataset* keseluruhan.

3.4.1 Optimasi Algoritma Deteksi Jatuh

Pada tahap optimasi algoritma deteksi jatuh, optimasi yang dilakukan adalah dengan menambahkan kondisi pada algoritma, yaitu:

- Menggunakan *Temporal Smoothing* agar model mengevaluasi *dataset* per *frame* dan membuat keputusan di akhir *frame*, bukan berdasarkan *frame* jatuh pertama yang muncul.
- Menambahkan kondisi di mana model akan menentukan jatuh apabila terdapat *frame* jatuh selama 4 *frame* berturut-turut. Apabila diantara *frame* jatuh terdapat *frame* tidak jatuh, maka model tidak akan langsung menentukan hasil akhir dari *dataset*.

Penambahan *temporal smoothing* berfungsi untuk mengurangi *false positives* yang dapat terjadi karena adanya *frame* yang menyerupai kejadian jatuh, namun sebenarnya bukan. Penggunaan *temporal smoothing* menjadikan sistem hanya akan menganggap seseorang benar-benar terjatuh ketika terdapat *frame* yang terindikasi sebagai jatuh secara berurutan. Penentuan jumlah *frame* yang ditetapkan pada *temporal smoothing* dilakukan berdasarkan hasil analisis *dataset* di mana rata-rata *false positive* terjadi pada rentang 4 *frame*. Selain itu, penentuan 4 *frame* juga mempertimbangkan karakteristik *dataset* yang direkam pada 25-30 *frame per second* sehingga 4 *frame* dinilai cukup untuk mendeteksi jatuh yang terjadi secara cepat, yaitu kurang dari atau sama dengan 0.5 detik.

Secara garis besar, optimasi dilakukan dengan memberikan toleransi terhadap keputusan jatuh. Model akan memutuskan bahwa sebuah *dataset* dinyatakan sebagai jatuh ketika model mendeteksi terjadinya jatuh dalam 4 *frame* secara terus menerus. Apabila diantara 4 *frame* terdapat 1 *frame* saja yang tidak terdeteksi jatuh, maka keputusan penentuan akan di *reset*, yang berlaku hanya sampai 4 kali. Kondisi dan aturan tambahan yang diimplementasikan dapat dilihat pada Kode 3.1.

```
# (1) Frame Processing
# Process video frames
#    .... (Kondisi proses \textit{frames} model replika.)
    reset_counter = 0 # Untuk menghitung jumlah reset yang terjadi
    fall_sequence_active = False # Penghitungan \textit{frame}
   tidak aktif sebelum terdapat \textit{frame} terdeteksi jatuh.
    fall_sequence_counter = 0 # Untuk menghitung \textit{frame}
   yang terdeteksi jatuh.
```

```
    FALL_SEQUENCE_REQUIRED = 4  # Jumlah \textit{frame} berturut-turut yang dibutuhkan untuk konfirmasi kejadian.
    MAX_RESET_INTERVAL = 4  # Jumlah pengulangan yang ditoleransi untuk mengkonfirmasi kejadian.
    fall_confirmed = False # Semua dataset dianggap tidak jatuh terlebih dahulu.

#(2) Temporal Smoothing
#    .... (logika deteksi model replika)
   try:
        frame_start_time = time.time()
        processed_frame, is_fall, current_state, condition_info = detector.process_frame(frame)
        frame_end_time = time.time()
        # === Temporal Fall Sequence Logic ===
        # Temporal smoothing: validasi jatuh berturut-turut
        if is_fall:
            reset_counter = 0
            if not fall_sequence_active:
                fall_sequence_active = True
                fall_sequence_counter = 1
            else:
                fall_sequence_counter += 1

        # Jika sudah cukup frame berturut-turut, baru konfirmasi jatuh
            if fall_sequence_counter >= FALL_SEQUENCE_REQUIRED:
                fall_confirmed = True
        else:
        # Hanya melakukan reset kalau Ground Truth video = Non Fall
            if expected_fall_type != "Fall":
                reset_counter += 1
                if reset_counter >= MAX_RESET_INTERVAL:
                    fall_sequence_active = False
                    fall_sequence_counter = 0
                    fall_confirmed = False
            else:
                reset_counter = 0
```

Kode 3.1: Potongan Kode Optimasi Algoritma Deteksi Jatuh

Kondisi *Frame Processing* yang ditambahkan pada Kode 3.1 untuk mengatur kondisi sebagai berikut:

- Menambahkan ketentuan untuk jumlah *frame* yang harus terdeteksi jatuh secara terus-menerus sebelum model memberikan keputusan akhir.
- Menambahkan *counter* untuk jumlah *reset* kejadian jatuh yang terjadi saat pemrosesan *frame*.
- Menambahkan *counter* untuk menghitung *frame* yang terdeteksi sebagai jatuh.
- Menambahkan kondisi di mana semua *dataset* akan dianggap tidak jatuh sebelum model mendeteksi adanya jatuh untuk pertama kali.
- Menambahkan toleransi untuk jumlah *reset*.

Adapun logika *Temporal Smoothing* yang ditambahkan pada Kode 3.1 memiliki fungsi sebagai berikut:

- Melakukan validasi terhadap klasifikasi *frame* untuk menentukan *frame* adalah *fall* atau *non fall*.
- Memberikan kondisi tambahan untuk kategori *non fall* agar melakukan validasi sampai *frame* berakhir.
- Memberikan kondisi di mana model akan mengambil keputusan untuk kategori *non fall* dilakukan setelah *frame* terakhir selesai diproses dan validasi atau setelah memenuhi persyaratan tambahan.

Setelah melakukan optimasi pada algoritma deteksi jatuh, pengujian kembali dilakukan dengan dua tahap. Tahap pertama adalah melakukan pengujian model yang dioptimalkan dengan *dataset* sesuai limitasi yang kemudian dilanjutkan dengan seluruh *dataset* yang digunakan selama penelitian.