

IF6083 Vision Assignment - 3

Oleh:

Made Arbi Parameswara

(23522002)



**PROGRAM STUDI MAGISTER INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
2023**

1.Theory Questions

Q1.1: Calculating the Jacobian

Jawab:

Fungsi *warp* $W(u; p)$ mengubah koordinat titik u (pada umumnya dalam format $[x, y, 1]$) berdasarkan parameter transformasi p . Transformasi *affine* dapat direpresentasikan sebagai operasi matriks:

$$\begin{bmatrix} u' \\ v' \\ 1 \end{bmatrix} = W(p) \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}$$

di mana $W(p)$ adalah matriks transformasi *affine* yang bergantung pada parameter p . Adapun representasi matriks $W(p)$ untuk transformasi *affine* dua dimensi adalah:

$$W(p) = \begin{bmatrix} 1 + p_1 & p_3 & p_5 \\ p_2 & 1 + p_4 & p_6 \\ 0 & 0 & 1 \end{bmatrix}$$

Jacobian (J) adalah turunan parsial dari fungsi *warp* $W(u; p)$ terhadap parameter transformasi p . Turunan dari fungsi *warp* $W(u; p)$ untuk mendapatkan nilai (J) adalah sebagai berikut:

$$J = \frac{\partial W(u; p)}{\partial p}$$

Sehingga untuk mencari turunan parsial dari *Jacobian Matrix* (J) berdasarkan operasi matriks diatas adalah sebagai berikut:

$$J = \frac{\partial}{\partial p} \left(\begin{bmatrix} 1 + p_1 & p_3 & p_5 \\ p_2 & 1 + p_4 & p_6 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u' \\ v' \\ 1 \end{bmatrix} \right)$$

$$J = \frac{\partial}{\partial p} \begin{bmatrix} (1 + p_1)u' + p_3v' + p_5 \\ p_2u' + (1 + p_4)v' + p_6 \\ 1 \end{bmatrix}$$

Maka Jacobian Matrix (J) adalah:

$$J = \begin{bmatrix} u' & 0 & v' & 0 & 1 & 0 \\ 0 & u' & 0 & v' & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Jika disederhanakan:

$$\begin{bmatrix} u' & 0 & v' & 0 & 1 & 0 \\ 0 & u' & 0 & v' & 0 & 1 \end{bmatrix}$$

Q1.2: Computational complexity

Menghitung computational cost metode Matthews-Baker

Untuk menemukan kompleksitas komputasi (notasi Big O) dari metode Matthews-Baker, yang juga dikenal sebagai *Inverse Compositional Algorithm* dalam konteks algoritma Lucas-Kanade, kita perlu mempertimbangkan langkah inisialisasi (pra-komputasi matriks Jacobian (J) dan invers Hessian matrix (H^{-1})) dan setiap iterasi *runtime*. Kompleksitas pada analisis ini akan dinyatakan dalam n , m , dan p , di mana:

- n adalah jumlah piksel dalam template T
- m adalah jumlah piksel dalam gambar input I
- p adalah jumlah parameter yang digunakan untuk mendeskripsikan warp W .

Adapun analisa langkah-langkah *computational cost* dari metode ini adalah sebagai berikut:

1. Inisialisasi (Pra-komputasi matriks *Jacobian* dan invers dari matriks *Hessian*):

- *Gradien Template* (T): Kompleksitasnya adalah $O(n)$ karena perlu menghitung gradien pada setiap piksel dalam template.
- *Evaluasi Jacobian* (J): Kompleksitasnya adalah $O(pn)$ karena perlu menghitung Jacobian pada setiap piksel dan setiap parameter.
- *Hessian* (H): Kompleksitasnya adalah $O(p^2n)$ karena merupakan hasil perkalian matriks Jacobian dengan dirinya sendiri (dimensi $p \times n$).
- *Invers Hessian* (H^{-1}): Biasanya memiliki kompleksitas $O(p^3)$ tetapi karena ini hanya dilakukan sekali, dampaknya terhadap kompleksitas keseluruhan mungkin tidak signifikan.

2. Setiap Iterasi *Runtime*:

- Update Warp (W): Setiap iterasi melibatkan *update* pada *warp* yang memiliki kompleksitas $O(pn)$ untuk perhitungan residual dan $O(p^2)$ untuk update parameter menggunakan invers Hessian.

Dengan notasi Big O, kompleksitas inisialisasi metode Matthews-Baker adalah $O(p^2n)$ dan kompleksitas per iterasi runtime adalah $O(pn + p^2)$ sehingga total dari kompleksitas Metode Matthews-Baker adalah $O(p^2n)$.

Menghitung *computational cost* metode Lucas-Kanade

Biaya komputasi dari setiap langkah dalam algoritma Lucas-Kanade, berikut adalah rincian setiap langkah:

1. Lakukan *wrapping input* I terhadap $W(x; p)$ untuk menghitung $I(W(x; p))$, *computational cost* $O(pn)$.
2. Hitung *error image* $T(x) - I(W(x; p))$, *computational cost* $O(n)$.
3. Lakukan *wrapping gradient* ∇I dengan $W(x; p)$, *computational cost* $O(pn)$.
4. Lakukan evaluasi matriks *Jacobian* $\frac{\partial W}{\partial p}$ pada $(x; p)$, *computational cost* $O(pn)$.
5. Lakukan komputasi steepest descent image $\nabla I \frac{\partial W}{\partial p}$, *computational cost* $O(pn)$.
6. Hitung hessian matriks, *computational cost* $O(p^2n)$.
7. Lakukan komputasi terhadap $\sum_x \left[\nabla I \frac{\partial W}{\partial p} \right]^T - [T(x) - I(W(x; p))]$, *computational cost* $O(pn)$.
8. Hitung Δp , *computational cost* $O(p^3)$
9. Lakukan update parameter $p \leftarrow p + \Delta p$ *computational cost* $O(p)$.

Dalam notasi Big O, kompleksitas inisialisasi metode Lucas-Kanade dari semua tahap adalah $O(p^2n + p^3)$.

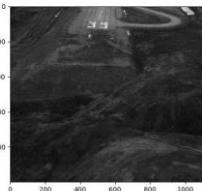
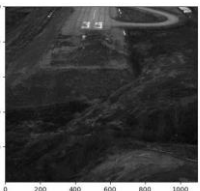
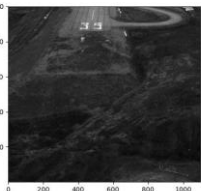
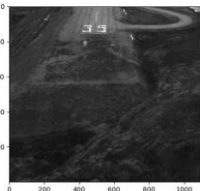
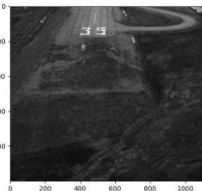
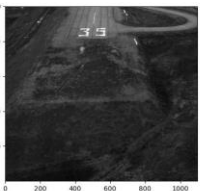
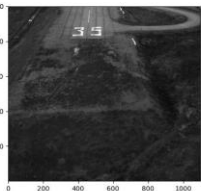
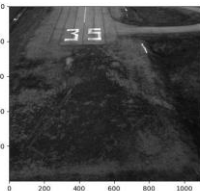
Berdasarkan Analisa diatas perbedaan computational cost antara metode Matthew-Baker dan Lucas-Kanade adalah sebagai berikut $O(p^2n)$ untuk Matthew-Baker dan $O(p^2n + p^3)$ untuk Lucas Kanade. Kompleksitas yang lebih rendah pada metode Matthews-Baker diakibatkan oleh beberapa *variable* sudah dinisasikan pada tahapan inisiasi sehingga tidak perlu dihitung kembali ke dalam langkah iterasi seperti yang dilakukan pada metode Lucas Kanade.

2. Lucas-Kanade Tracker

Sebelum memasuki penjelasan mengenai performa dari setiap algoritma, akan sedikit dibahas mengenai kondisi *tracking* dari setiap video. Hal ini dilakukan untuk mengetahui mengapa sebuah algoritma bekerja lebih baik pada sebuah video dari pada video lainnya. Adapun kondisi tracking dari setiap video dijelaskan sebagai berikut:






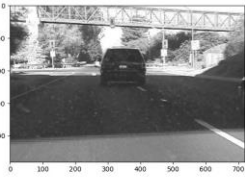


- Landing: Jika diobservasi secara langsung target pada video ini yaitu, “angka 35” mengalami translasi dan scalling.

Tabel 1. Landing *Frame by Frame*

			
<i>Frame: 1</i>	<i>Frame: 7</i>	<i>Frame: 13</i>	<i>Frame: 20</i>
			
<i>Frame: 26</i>	<i>Frame: 32</i>	<i>Frame: 38</i>	<i>Frame: 49</i>









- Car1: Jika diobservasi secara langsung *object tracking* pada video ini yaitu, “mobil hitam” mengalami translasi, scalling, dan perubahan *contrast*. Dalam video ini juga ditunjukkan objek tetangga berupa “2 mobil putih” yang berjalan berdampingan pada waktu tertentu dengan objek *tracking*. Adapun terdapat sedikit perbedaan kecepatan translasi dan *scaling* yang terjadi pada *object tracking* antara *frame* 1 – 186 dengan *frame* 186 – 260.

Tabel 2. Car1 *Frame by Frame*

 <p><i>Frame: 1</i></p>	 <p><i>Frame: 38</i></p>	 <p><i>Frame: 75</i></p>	 <p><i>Frame: 112</i></p>
 <p><i>Frame: 149</i></p>	 <p><i>Frame: 186</i></p>	 <p><i>Frame: 223</i></p>	 <p><i>Frame: 260</i></p>

- Car2: Jika diobservasi secara langsung *object tracking* pada video ini yaitu, “mobil abu-abu” mengalami translasi, sedikit scalling, sedikit perubahan contrast, dan oklusi. Dalam video ini juga ditunjukkan objek tetangga berupa “beberapa mobil lainnya” yang berjalan berdampingan pada waktu tertentu dengan objek tracking. Oklusi yang terjadi diakibatkan mobil terhalang oleh pendanda jalan dan *traffic light*. Adapun terdapat sedikit perbedaan kecepatan translasi yang menyebabkan perbedaan distorsi yang terjadi pada *object tracking* antara *frame* 1 – 168 dengan *frame* 227 – 414.

Tabel 3. Car2 *Frame by Frame*





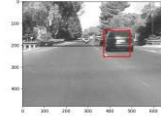
 <p><i>Frame: 1</i></p>	 <p><i>Frame: 60</i></p>	 <p><i>Frame: 134</i></p>	 <p><i>Frame: 168</i></p>
 <p><i>Frame: 227</i></p>	 <p><i>Frame: 286</i></p>	 <p><i>Frame: 345</i></p>	 <p><i>Frame: 414</i></p>

Q2.4 Testing Your Algorithm

Berikut ini adalah analisis performa dari ketiga algoritma Lucas-Kanade pada setiap video dan perbedaan performanya:






Lucas-Kanade Forward Additive Alignment with Translation

Tabel 4. Car1 Frame by Frame with Lucas-Kanade Translation

				
Frame: 1	Frame: 66	Frame: 131	Frame: 196	Frame: 260

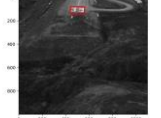
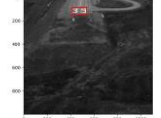
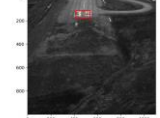
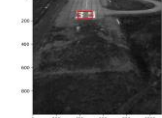
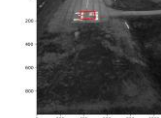
Time taken: 38.57 seconds.

Tabel 5. Car2 Frame by Frame with Lucas-Kanade Translation

				
Frame: 1	Frame: 143	Frame: 234	Frame: 313	Frame: 414

Time taken: 44.23 seconds.

Tabel 6. Landing Frame by Frame with Lucas-Kanade Translation

				
Frame: 1	Frame: 14	Frame: 27	Frame: 40	Frame: 49

Time taken: 17.85 seconds.

Algoritma ini dedesain untuk fokus pada translasi tanpa mempertimbangkan skala atau rotasi.




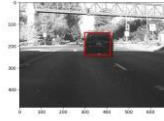
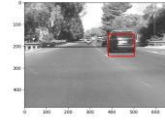
Adapun hasil analisa untuk setiap objek adalah sebagai berikut:

- Car1: Dapat mendeteksi translasi mobil hitam dengan baik, tetapi setelah terjadi perubahan kontras antara frame 196 dan frame 260, akurasi dari deteksi sedikit menurun.

- Car2: Dapat mendeteksi translasi cukup baik namun ketika terjadi oklusi dan translasi yang secepat akurasi deteksi sedikit menurun, hal itu terjadi dari *frame* 234, *frame* 313, dan *frame* 414.
- Landing: Algoritma memiliki kinerja yang cukup baik dalam mendeteksi translasi, tetapi kesulitan dalam menyesuaikan dengan perubahan skala. Dapat terlihat pada *frame* 49 skala pada saat target tracking menderkat ukurannya kurang tepat.






Lucas-Kanade Forward Additive Alignment with Affine Transformation

Tabel 7. Car1 *Frame by Frame with Lucas-Kanade Affine Transformation*

				
<i>Frame: 1</i>	<i>Frame: 66</i>	<i>Frame: 131</i>	<i>Frame: 196</i>	<i>Frame: 260</i>

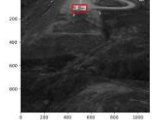
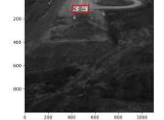
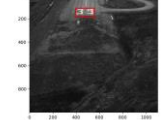
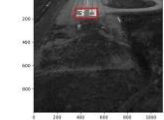
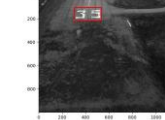
Time taken: 114.79 seconds.

Tabel 8. Car2 *Frame by Frame with Lucas-Kanade Affine Transformation*

				
<i>Frame: 1</i>	<i>Frame: 143</i>	<i>Frame: 234</i>	<i>Frame: 313</i>	<i>Frame: 414</i>

Time taken: 150.44 seconds.

Tabel 9. Landing *Frame by Frame with Lucas-Kanade Affine Transformation*

				
<i>Frame: 1</i>	<i>Frame: 14</i>	<i>Frame: 27</i>	<i>Frame: 40</i>	<i>Frame: 49</i>




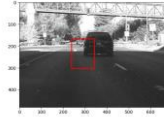
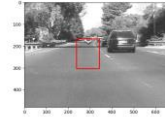
Time taken: 25.68 seconds.

Algoritma ini memperluas kemampuan deteksi sehingga mencakup translasi, rotasi, dan skala dengan lebih baik.

- Car1: Dapat menangani translasi dan skala dengan lebih baik, namun ketika terjadi perubahan contrast (*frame* 260) dan terdapat objek tetangga (*frame* 1 & *frame* 66) akurasi dari *frame* sedikit menurun.
- Car2: Dapat menangani translasi dan skala dengan lebih baik, namun kinerjanya menjadi sangat buruk Ketika terjadi oklusi (*frame* 105), akurasi semakin buruk seiring waktu dan pada *frame* 414, ukuran dari bounding box menjadi sangat besar.
- Landing: Lebih mampu menyesuaikan dengan perubahan skala daripada varian translasi, pada *frame* 49 ukuran target sesuai jika dibandingkan algoritma yang hanya berfokus pada translasi.






Inverse Compositional Alignment with Affine Transformation

Tabel 10. Car1 *Frame by Frame with Lucas-Kanade Inverse Affine Transformation*

				
<i>Frame: 1</i>	<i>Frame: 66</i>	<i>Frame: 131</i>	<i>Frame: 196</i>	<i>Frame: 260</i>

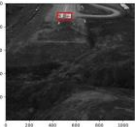
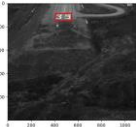
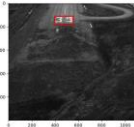
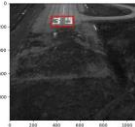
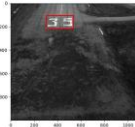
Time taken: 52.53 seconds.

Tabel 11. Car2 *Frame by Frame with Lucas-Kanade Inverse Affine Transformation*

				
<i>Frame: 1</i>	<i>Frame: 143</i>	<i>Frame: 234</i>	<i>Frame: 313</i>	<i>Frame: 414</i>

Time taken: 81.31 seconds.

Tabel 11. Car1 *Frame by Frame with Lucas-Kanade Inverse Affine Transformation*

				
<i>Frame: 1</i>	<i>Frame: 14</i>	<i>Frame: 27</i>	<i>Frame: 40</i>	<i>Frame: 49</i>

Time taken: 17.02 seconds.

Algoritma ini lebih efisien secara komputasi dengan menggunakan pendekatan komposisi terbalik. Namun secara performa algoritma ini tidak terlalu buruk jika dibandingkan dua algoritma lainnya.

- Car1: Dapat mengikuti objek dengan efisien selama perubahan skala, tetapi kontras yang berubah pada *frame* 196 membuat algoritma kehilangan jejak dari target.
- Car2: Lebih efisien dari kedua algoritma lainnya dan baik dalam menangani translasi, namun sama dengan Lucas Kanade Affine oklusi menyebabkan pada *frame* 143 menyebabkan pembesaran box tracer.
- Landing: Performa terbaik dan terefisien dari algoritma ini dikarenakan tidak adanya oklusi dan perubahan kontras.

Perbandingan dan Titik Kegagalan

Perbedaan Performa:

- *Lucas-Kanade Forward Additive Alignment with Translation*: lebih sederhana dan efektif untuk gerakan linear dan oklusi, tetapi kurang fleksibel dalam menangani perubahan kontras dan scalling.
- *Lucas-Kanade Forward Additive Alignment with Affine Transformation*: lebih fleksibel dan dapat menangani gerakan yang lebih kompleks termasuk translasi, skala dan rotasi. Namun memiliki kelemahan pada oklusi.
- *Inverse Compositional Alignment with Affine Transformation*: algoritma dengan komputasional yang paling efisien namun tetapi mungkin kurang akurat dalam translasi, perubahan kontras dan oklusi. tertentu.

Titik Kegagalan:




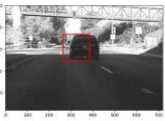
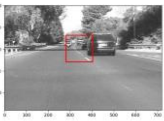
- *Lucas-Kanade Forward Additive Alignment with Translation*: Kegagalan terjadi pada perubahan translasi, skala dan kontras yang signifikan.
- *Lucas-Kanade Forward Additive Alignment with Affine Transformation*: Lebih resisten terhadap kegagalan, tetapi masih bisa terpengaruh oleh perubahan kontras yang cepat dan oklusi berat.
- *Inverse Compositional Alignment with Affine Transformation*: Kegagalan serupa dengan varian affine.

Setiap algoritma memiliki kekuatan dan keterbatasan yang berbeda tergantung pada karakteristik gerakan dan visual dari objek yang dilacak dalam video. Pemilihan algoritma yang paling cocok bergantung pada kebutuhan spesifik dari aplikasi dan kondisi video.

3. Extra Credit






Q3.1x: Adding Illumination Robustness.

Tabel 12. Car1 Frame by Frame with Lucas-Kanade Adding Illumination Robustness

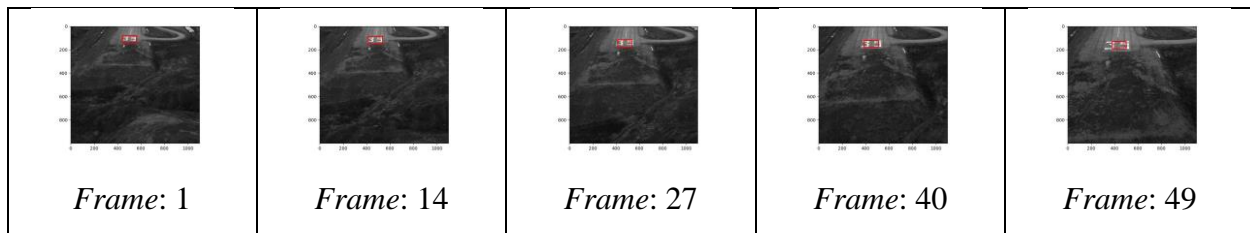
				
Frame: 1	Frame: 66	Frame: 131	Frame: 196	Frame: 260

Time taken: 1672.77 seconds.

Tabel 13. Car2 Frame by Frame with Lucas-Kanade Adding Illumination Robustness

				
Frame: 1	Frame: 143	Frame: 234	Frame: 313	Frame: 414

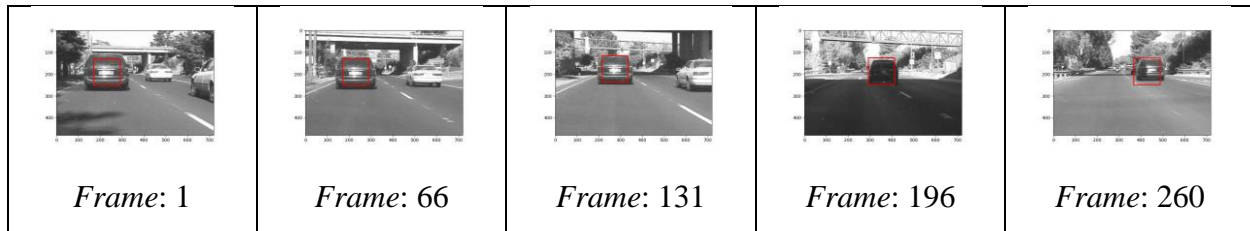
Time taken: 136.08 seconds.

Tabel 14. *Landing Frame by Frame with Lucas-Kanade Adding Illumination Robustness*

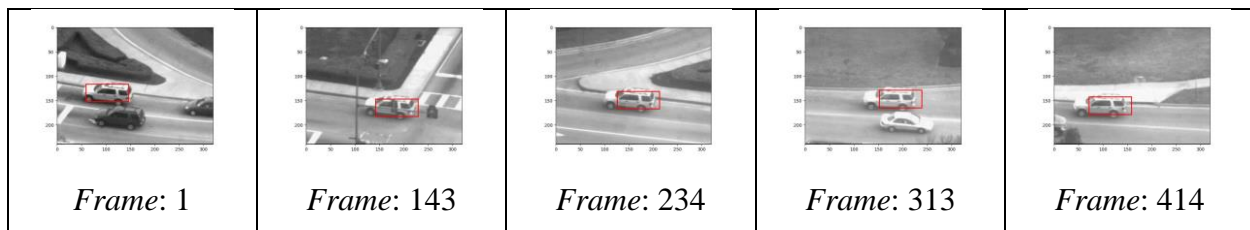
Time taken: 234.58 seconds.

Algoritma ini fokus pada translasi dan illumination terhadap robustness tanpa mempertimbangkan skala atau rotasi pada image. Secara umum algoritma ini bekerja sangat baik namun adanya perubahan contrast yang tinggi membuat algoritma ini kekurangan akurasi. Kelemahan lain dari algoritma ini adalah efisiensinya yang paling buruk diantara algoritma lainnya. Detail performa dari algoritma pada setiap video adalah sebagai berikut:

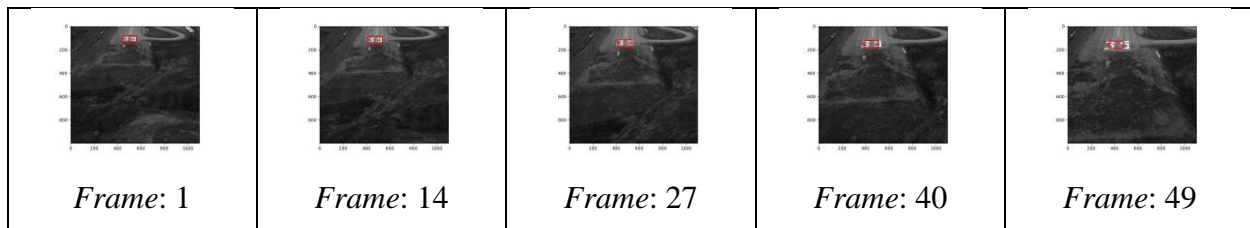
- Car1: Dapat mendeteksi translasi mobil hitam dengan baik, tetapi setelah terjadi perubahan kontras antara *frame* 196 dan *frame* 260, akurasi dari deteksi menjadi menurun. Hal ini menunjukkan perlunya peningkatan optimasi algoritma untuk mengoptimasi akurasi dari *tracer*.
- Car2: Dapat mendeteksi translasi dan oklusi dengan baik, bukan hanya itu algoritma ini juga dapat mendeteksi translasi yang cepat dengan sangat baik. Hal ini menunjukkan *illumination robustness* bekerja dengan baik ketika objek mengalami oklusi dan distorsi yang tinggi antar piksel akibat peningkatan kecepatan Car2.
- Landing: Algoritma memiliki kinerja yang baik dalam mendeteksi translasi, tetapi masih kesulitan dalam menyesuaikan dengan perubahan skala. Dapat terlihat pada *frame* 49 skala pada saat *target tracking* mendekat, ukuran *box* menjadi kurang tepat.

Q3.2x: LK Tracking on Image Pyramid**Tabel 15.** *Car1 Frame by Frame with Lucas-Kanade on Image Pyramid*

Time taken: 35.92 seconds.

Tabel 16. *Car2 Frame by Frame with Lucas-Kanade on Image Pyramid*

Time taken: 39.47 seconds.

Tabel 17. *Landing Frame by Frame with Lucas-Kanade on Image Pyramid*

Time taken: 11.95 seconds.

Algoritma ini berfokus untuk meningkatkan efisiensi dengan mentransformasi image menjadi sebuah pyramid sehingga proses menjadi lebih efisien. Kegagalan transformasi ini terjadi pada saat terjadi perubahan contrast yang tinggi dan kurangnya kemampuan algoritma ini dalam mendeteksi scaling. Algoritma ini merupakan algoritma yang paling efisien dan memiliki performa yang baik. Detail performa dari algoritma pada setiap video adalah sebagai berikut:

- Car1: Dapat mendeteksi translasi mobil hitam dengan baik, tetapi setelah terjadi perubahan kontras antara *frame* 196 dan *frame* 260, akurasi dari deteksi menjadi menurun. Hal ini menunjukkan perlunya peningkatan optimasi algoritma untuk mengoptimasi akurasi dari tracer. Algoritma ini paling efisien dalam melakukan tracing pada video Car1 jika dibandingkan algoritma lainnya.
- Car2: Dapat mendeteksi translasi dengan baik dan ketika terjadi oklusi dan translasi yang

cepat akurasi deteksi sedikit menurun bertahan. Hal ini menunjukkan algoritma ini kurang mampu ketika objek mengalami oklusi dan distorsi yang tinggi antar piksel akibat peningkatan kecepatan dari Car2. Algoritma ini paling efisien dalam melakukan tracing pada video Car2 jika dibandingkan algoritma lainnya.

- Landing: Algoritma memiliki kinerja yang baik dalam mendeteksi translasi, tetapi masih kesulitan dalam menyesuaikan dengan perubahan skala. Dapat terlihat pada *frame* 49 skala pada saat target tracking mendekat, ukuran box menjadi kurang tepat. Algoritma ini paling efisien dalam melakukan tracing pada video Landing jika dibandingkan algoritma lainnya.