EVALUASI KINERJA MINI PC DENGAN ALGORITMA ALPR
STUDI KASUS : OTOMATISASI PERHITUNGAN POTENSI PARKIR SECARA REALTIME

Nur Azizah Novitami[1], Adnan[2], Intan Sari Areni[3]
Departemen Teknik Informatika, Fakultas Teknik, Universitas Hasanuddin
Jl. Poros Malino, Gowa, Sulawesi Selatan, Indonesia
azizahnovitami@gmail.com, adnan.unhas1@gmail.com, intanareni@gmail.com

Abstrak

Pajak Parkir adalah salah satu sumber Pendapatan Asli Daerah yang dibayarkan ke DISPENDA melalui pengelola parkir. Namun salah satu kendala yang dihadapi DISPENDA dalam pemungutan pajak parkir ini adalah DISPENDA tidak dapat memastikan apakah jumlah pajak yang disetor oleh pengelola sesuai dengan jumlah penerimaan yang sebenarnya. Hal ini dikarenakan data pembanding yang dimiliki tidak aktual dan masih menggunakan pengumpulan dengan sistem berbasis *counter* manual. Untuk itu dibutuhkan sebuah sistem otomatisasi *realtime* terhadap perhitungan tersebut. Dalam rangka membantu DISPENDA menghasilkan sistem yang efisien untuk diimplementasikan, penelitian ini mengevaluasi kinerja Mini PC terhadap algoritma ALPR (*Automatic License Plat Recognition)* yang telah ada. Kinerja yang diukur adalah: (1) rata-rata waktu eksekusi satu gambar hingga karakter plat dikenali, dengan skenario pengujian yaitu perubahan nilai kernel *Gaussian Blur* 3x3, 5x5, 7x7 dan 11x11, (2) Kinerja CPU Mini PC dengan skenario pengujian yaitu diamati dalam keadaan startup dan dalam keadaan mengeksekusi program. Hasil yang diperoleh: nilai kernel dengan akurasi terbaik 83% yaitu nilai kernel 7x7, dan nilai kernel dengan rata-rata waktu eksekusi tercepat adalah nilai kernel 11x11 yaitu 23.44806444 detik. Hasil kinerja CPU pada keadaan *startup* yaitu sebesar 0.21% dan dalam keadaan mengeksekusi program sebesar 23.40% , artinya penggunaan CPU naik sebesar 23.19% dari keadaan startup.

Kata Kunci: DISPENDA, ALPR, evaluasi kinerja, *mini* PC.

1. Pendahuluan

1.1. Latar Belakang

Parkir merupakan salah satu hal krusial dalam tata perkotaan. Keberadaan sistem parkir akan mempermudah pengaturan kendaraan-kendaraan yang masuk ke dalam ruang lingkup area tertentu seperti mall, perkantoran dan area kampus. Dengan adanya sistem parkir juga, bisa diperoleh data-data penting seperti jumlah total kendaraan yang berkunjung, lama waktu parkir, dan *manifest* data lainnya yang nantinya bisa dimanfaatkan untuk hal analisis data.

Salah satu hal menarik jika berbicara mengenai parkir adalah pajak parkir itu sendiri. Berdasarkan arsip data dari PD Parkir Makassar Raya, penerimaan pajak parkir tiap tahunnya menunjukkan penetapan target yang hampir selalu lebih tinggi dari realisasi penerimaan tahun sebelumnya bisa dikatakan berhasil. Fenomena tersebut mengindikasikan bahwa pajak sektor parkir menunjukkan progress yang baik dalam tampilan statistiknya. Namun apabila dibenturkan dengan fakta lapangan mengenai kondisi kepadatan penduduk Kota Makassar, kondisi jalan raya protokol yang hampir selalu terlihat *high traffic* di jam berangkat dan pulang kerja, dan psikologi budaya konsumerisme masyarakatnya yang menganjurkan untuk beraktualisasi diri dengan mengunjungi pusat-pusat keramaian. Maka menjadi lumrah untuk mempertanyakan nominal angka penerimaan pajak parkir yang diperoleh. Apakah benar jumlah-jumlah penerimaan yang diperoleh telah menggambarkan potensi sebenarnya dari pajak parkir, mengingat jumlah titik-titik parkir yang cukup banyak tersebar di Kota Makassar.[1]

Salah satu sumber pendapatan pajak parkir yang cukup besar adalah pajak parkir Mall [1] . Sistem parkir Mall di kota Makassar di kelola oleh pihak swasta dimana pajak parkirnya wajib dibayarkan dan dilaporkan kepada DISPENDA (Dinas Pendapatan Daerah). Dalam wawancara

penulis dengan Kepala Bidang DISPENDA kota Makassar pada Februari 2019, Bapak Dahyar menjelaskan, sebagai data pembanding, DISPENDA harus menghitung sendiri potensi parkir Mall tersebut dengan menerjunkan orang-orang dari Laskar Peduli Pajak (LPP) secara bergantian untuk menghitung manual jumlah kendaraan yang masuk dan keluar dalam sehari. Hal inilah yang menjadi kendala DISPENDA, karena perhitungan potensi parkir seperti itu membutuhkan banyak energi dan sumber daya manusia, serta data yang dihasilkan tidaklah aktual. Untuk itu dibutuhkan sebuah model sistem otomatisasi yang aktual (*realtime)* untuk membantu perhitungan potensi parkir tersebut.

Dalam mengimplementasikan sistem sesuai kebutuhan DISPENDA tersebut, dapat menggunakan sistem yang sama yang digunakan pada parkir Mall pada umumnya, yaitu dengan mengambil data plat kendaraan yang masuk dan keluar, kemudian membandingkan waktunya untuk mengetahui lama parkir kendaraan tersebut. Akumulasi jumlah kendaraan yang parkir, lama waktu parkir dan biaya parkirnya akan dikirimkan kepada DISPENDA secara *realtime.* Hanya saja, keseluruhan sistem ini (mulai dari deteksi hingga pengiriman data) haruslah berjalan secara otomatis tanpa bantuan manusia.

Dari segi metode dan algoritma, implementasi deteksi otomatis plat kendaraan sudah banyak dikerjakan di Indonesia, salah satunya adalah dengan algoritma Automatic License Plate Recognition (ALPR) . Algoritma ALPR terdiri atas tiga buah tahapan, yaitu *plate localization, character segmentation* serta *character recognition*. Algoritma ALPR akan bekerja untuk mengenali plat kendaraan tersebut dan menggunakannya untuk menghitung lama parkir kendaraan tersebut.

Namun dalam mengimplementasikan sistem sesuai kebutuhan DISPENDA, yang harus dipikirkan bukan hanya keefektifan *software* atau akurasi dari program deteksi plat, tetapi juga keefektifan *hardware* yang akan digunakan (biaya, waktu maupun proses pengiriman data). Karena tantangan yang sering dihadapi adalah bagaimana alat yang dipasang tidak mencolok, hemat daya tetapi proses pengiriman data dapat berjalan lancar dan *realtime,* sistem menggunakan Mini PC bisa menjadi solusinya. Dengan memanfaatkan Mini PC sebagai pengontrol jarak jauh melalui bahasa pemrograman tertentu menjadikan sistem lebih efisien dalam segi ukuran dan daya yang digunakan [2]. Selain itu dengan bantuan jaringan internet, proses *monitoring* bisa berjalan secara *realtime*.

Karena itu, penelitian ini bertujuan untuk mengevaluasi kinerja Mini PC terhadap algoritma deteksi plat agar sistem yang dihasilkan untuk DISPENDA efisien sebelum diimplementasikan.

1.2. Rumusan Masalah

Berdasarkan latar belakang yang telah dipaparkan, maka rumusan masalah pada penelitian ini antara lain:

1. Berapa rata-rata waktu yang dibutuhkan Mini PC dalam memproses algoritma ALPR untuk menghasilkan data plat yang diinginkan?
2. Bagaimana kinerja CPU Mini PC dalam memproses algoritma ALPR?

1.3. Tujuan Penelitian

Tujuan dari penelitian ini antara lain:

1. Untuk mengetahui rata-rata waktu yang dibutuhkan *Mini PC* dalam memproses algoritma ALPR untuk menghasilkan data plat yang diinginkan.
2. Untuk mengetahui kinerja CPU Mini PC dalam memproses algoritma ALPR.

1.4. Manfaat Penelitian

Penelitian ini diharapkan dapat memberi manfaat sebagai berikut:

1. Bagi DISPENDA, dapat digunakan sebagai data pembanding yang aktual dan efisien.
2. Bagi Peneliti, dapat digunakan untuk menambah pengetahuan dan kemampuan di bidang *Internet of Things* dan Pengolahan Citra.
3. Bagi Institusi pendidikan, dapat digunakan sebagai referensi dalam pengembangan penelitian topik terkait.

2. Landasan Teori

2.1. Plat Nomor Kendaraan

Plat nomor kendaran atau biasa disebut juga plat nomor polisi, atau yang dalam bahasa Inggris disebut *vehicle registration plate*, merupakan plat logam yang terdapat di kendaraan dan digunakan untuk tujuan identifikasi. Plat nomor ini berisi karakteristik unik yang berbeda antara satu kendaraan dengan kendaraan yang lain.[3]

Karakteristik plat kendaraan berbeda antara negara satu dan lainnya. Indonesia memiliki karakteristik plat dengan format HH AAA HH atau HH AAAA HHH, dimana H merepresentasikan huruf dan A merepresentasikan angka 0-9. Awalan

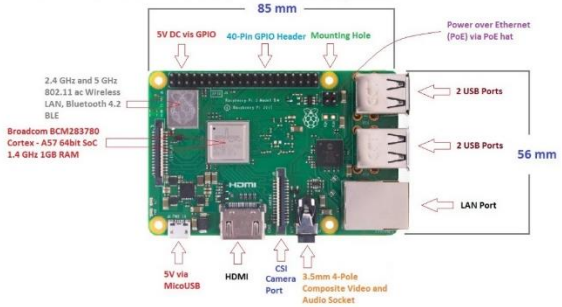
huruf pertama merepresentasikan area dimana kendaraan tersebut didaftarkan. Huruf ini diikuti dengan satu sampai empat angka kemudian diakhiri oleh satu hingga tiga huruf[3]. Misalnya, DD 2173 AK merupakan kendaraan dari daerah Makassar, karena dua huruf pertamanya, DD, merupakan kode untuk Makassar.

Terdapat beberapa kelompok plat nomor yang dibedakan melalui warnanya, yaitu :

1. Warna putih dengan latar hitam, artinya kendaraan pribadi.
2. Warna merah pada latar putih, artinya kendaraan yang belum terdaftar atau kendaraan baru yang belum ada pemiliknya.
3. Warna hitam pada latar kuning, artinya kendaraan untuk transportasi umum seperti bus, taksi, dan angkutan kota.
4. Warna putih pada latar merah, artinya kendaraan milik institusi pemerintahaan.
5. Sedangkan Militer, Polisi dan Pemadam Kebakaran memiliki warna tersendiri dan biasanya terdapat lambing pangkat pemilik kendaraan tersebut.

2.2. Raspberry Pi

Raspberry Pi adalah sebuah komputer papan tunggal (*single-board computer*) atau SBC berukuran kecil.



Gambar 2.1 Struktur Raspberry Pi

Raspberry Pi 3 adalah generasi ketiga dari Raspberry Pi, menggantikan Raspberry Pi 2 Model B pada Februari 2016. Raspberry Pi 3 memiliki bentuk yang identik dengan Raspberry Pi 2 sebelumnya (dan Pi 1 Model B +) dan memiliki kompatibilitas lengkap dengan Raspberry Pi 1 dan 2. Pade Raspberry pi 3 Model B+ pengontrol USB Ethernetnya menawarkan konektivitas gigabit dengan throughput maksimum secara teoritis yaitu 300Mb/s, karena penggunaannya pada channel USB tunggal. Pada perangkat terbarunya ini Raspberry menambahkan fitur built- in wireless dan processor yang lebih bertenaga yang belum pernah dimiliki pada versi sebelumnya[4]. Spesifikasi dapat dilihat pada **tabel 2.1.**

Tabel 2.1 Spesifikasi Rasbperry Pi 3B+

Spesifikasi	Keterangan
SoC	Broadcom BCM2837B0 quad-core A53 (ARMv8) 64-bit @ 1.4GHz
GPU	Broadcom Videocore-IV
RAM	1GB LPDDR2 SDRAM
Networking	Gigabit Ethernet (via USB channel), 2.4GHz and 5GHz 802.11b/g/n/ac Wi-Fi
Bluetooth	Bluetooth 4.2, Bluetooth Low Energy (BLE)
Storage	Micro-SD
GPIO	40-pin GPIO header, populated
Ports	HDMI, 3.5mm analogue audio-video jack, 4x USB 2.0, Ethernet, Camera Serial Interface (CSI), Display Serial Interface (DSI)
Dimensions	82mm x 56mm x 19.5mm, 50g
Power Input	5V/2.5A DC

2.3. Pengolahan Citra

Pengolahan citra adalah metode untuk melakukan beberapa operasi terhadap citra, dengan tujuan adalah untuk meningkatkan kualitas citra maupun untuk mengekstrak informasi yang penting dari citra tersebut. Di dalam mengolah sebuah citra, terdapat berbagai algoritma yang dapat diterapkan untuk menghasilkan keluaran yang lebih baik. Keluaran yang baik akan mempengaruhi hasil dari proses yang akan dilakukan selanjutnya.

1. *Grayscale*

   Citra *Grayscale* (skala keabuan) merupakan suatu citra yang memiliki warna putih, abu-abu, dan hitam. Jenis citra ini memiliki intensitas berkisar antara 0 sampai dengan 255. Nilai 0 dinyatakan sebagai hitam dan 255 dinyatakan sebagai putih. Nilai minimum atau maksimum dari suatu citra bergantung pada jumlah bitnya. Misalnya pada skala keabuan 4 bit, maka jumlah

kemungkinan nilainya adalah 2^4 = 16 dan nilai maksimumnya adalah 2^4-1 = 15. Sedangkan untuk skala keabuan 8 bit, maka jumlah kemungkinan nilainya adalah 2^8=256 dimana nilai maksimumnya adalah 2^8-1 = 255[5]. Terdapat tiga metode yang digunakan untuk mengonversi citra RGB ke *grayscale*, yaitu *lightness method*, *average method*, dan *luminosity method.*

2. *Thresholding*

Thresholding merupakan salah satu teknik dasar dalam melakukan segmentasi citra. *Thresholding* digunakan untuk mengonversi citra *grayscale* menjadi citra biner dalam rangka untuk memisahkan beberapa objek target dari latar belakangnya. Citra biner hanya terdiri dari dua intensitas warna yaitu hitam yang memiliki representasi nilai 0 dan putih yang memiliki representasi nilai 1. Sehingga jenis citra in hanya membutuhkan 1 bit memori untuk menyimpan kedua warna ini.

3. *Gaussian Blur*

Gaussian Blur atau yang sering disebut *Gaussian Filter* merupakan sebuah metode untuk *image smoothing* dimana nilai pembobotan untuk setiap piksel berdasarkan pada fungsi *Gaussian*. Filter *Gaussian* dapat digunakan untuk mengurangi derau yang ada pada citra digital.

Gaussian Filter diperoleh dari operasi konvolusi yaitu operasi perkalian yang dilakukan antara matriks kernel dengan matriks citra asli. Nilai kernel *Gaussian* memiliki ukuran yang beragam dan pasti adalah bilangan ganjil, hal ini dikarenakan kernel *Gaussian* memiliki nilai di posisi koordinat tengah. Semakin besar ukuran kernel maka *noise* pada gambar akan semakin hilang. Akan tetapi, ukuran kernel yang terlalu besar juga bisa menyebabkan semakin hilangnya *feature* pada gambar sehingga hasil *pre-processing* menjadi kurang baik [5].

Perkalian antara bobot matriks gambar asli dengan bobot matriks kernel dapat dirumuskan seperti pada persamaan 2.1

$$\frac{1}{K}\sum_{p=0}^{N-1}\left(\sum_{q=0}^{M-1} G(p,q)\ Pixel\ A\left(i+p-\frac{(N-1)}{2}, i+q-\frac{(M-1)}{2}\right)\right) \quad (2.1)$$

Keterangan:

Pixel A = Gambar A (Gambar asli).
N = Jumlah kolom matriks kernel.
M = Jumlah baris matriks kernel.
K = Penjumlahan semua bobot di G
G(p,q) = Elemen matriks kernel pada posisi (p,q).

Gambar yang akan diproses dibagi menjadi dua jenis piksel, yaitu piksel batas dan piksel dalam. Piksel batas yaitu piksel yang posisinya berada pakling luar pada gambar. Piksel dalam yaitu piksel yang posisinya berada di dalam piksel batas. Untuk piksel yang berada di dalam, perkalian dilakukan dengan mencari nilai piksel baru sebagai piksel tengah dan bobotnya dikalikan dengan bobot pada piksel tengan matriks kernel. Hasil dari perkalian tersebut dijumlahkan dengan hasil perkalian antara bobot piksel yang ada pada tetangganya dengan bobot piksel dari matriks kernel. Untuk pisel yang berada pada sudut perbatasan, sebelum melakukan perkalian, maka harus mencari bobot pada piksel-piksel luar (*dummy*). Bobot piksel-piksel ini dicari dengan menggunakan metode interpolasi yaitu melihat kedua piksel yang berdekatan juga searah (*horizontal* atau *vertical*). Jika bobot piksel kurang dari 0 maka bobot tersebut akan dijadikan 0. Apabila ada piksel yang memiliki bobot lebug besar dari 255 maka bobotnya dijadikan 255 [5].

2.4. *Profilling*

Profiling pada suatu program adalah suatu bentuk analisis untuk mengukur hal-hal seperti penggunaan memori, penggunaan waktu, penggunaan instruksi tertentu, atau frekuensi dan durasi pemanggilan fungsi [6]. Ini adalah cara untuk memahami di mana saja jumlah sumber daya terbesar yang digunakan untuk menargetkan optimasi pada bagian tertentu dari program tersebut. Ada dua tipe *profiling*, yaitu [7] :

1. *Profil*ing Deterministik: Semua kejadian/proses akan dimonitor. *Profiling* ini memberikan informasi yang akurat tetapi memiliki dampak besar pada kinerja

(*overhead*), dimana ini berarti kode akan berjalan lebih lambat saat melakukan *profiling*. Jenis *profiling* ini cocok digunakan untuk fungsi-fungsi kecil.

2. *Profiling* statistik: *Profiling* jenis ini akan mengambil sampel kondisi eksekusi secara berkala untuk memperoleh indikatornya. Metode ini kurang akurat, tetapi juga dapat mengurangi overhead.

Melakukan *profiling* pada program Python dapat dilakukan dengan *library* standar yang dimiliki Python, maupun modul dan program pihak ketiga. Python dilengkapi dengan dua modul untuk *profil*ing deterministic yaitu cProfile dan Profile. Keduanya adalah implementasi yang berbeda dari antarmuka yang sama.

Modul Profile lebih sesuai jika ingin memperluas *profiling* dengan cara tertentu. Sedangkan cProfile lebih banyak digunakan untuk program-program yang akan berjalan lama / terus menerus [7].

CProfile dapat dijalankan dari terminal, maupun diimpor sebagai modul dengan Python. CProfile akan memberikan informasi hasil *profiling* dari suatu fungsi, total waktu yang digunakan untuk setiap pemanggilan (tidak termasuk pemanggilan ke fungsi lain), waktu kumulatif fungsi dan jumlah panggilan ke fungsi tersebut. Contoh pemanggilan cProfile dan output yang dihasilkan terlihat pada **gambar 2.2**

```
→ python3 -m cProfile script.py

58 function calls in 9.419 seconds

Ordered by: standard name

ncalls  tottime  percall  cumtime  percall filename:lineno(function)
     1    0.000    0.000    9.419    9.419 part1.py:1(<module>)
    51    9.419    0.185    9.419    0.185 part1.py:1(computation)
     1    0.000    0.000    9.419    9.419 part1.py:10(function1)
     1    0.000    0.000    9.243    9.243 part1.py:15(function2)
     1    0.000    0.000    0.176    0.176 part1.py:20(function3)
     1    0.000    0.000    9.419    9.419 part1.py:24(main)
```

Gambar 2.2 Contoh Pemanggilan CProfile dan Output yang dihasilkan

Hasil **gambar 2.3** bisa diuraikansebagai berikut :

- *ncalls*: Seperti namanya, variabel ini memuat jumlah panggilan terhadap fungsi. Dengan data ini, fungsi yang memiliki banyak panggilan atau menghabiskan terlalu banyak waktu per panggilan dapat dioptimalkan.
- *tottime*: Total waktu yang dihabiskan dalam fungsi itu sendiri, tetapi tidak termasuk sub-panggilan. Pada contoh diatas, dapat dilihat bahwa fungsi 'computation' dipanggil sebanyak 51 kali, dan setiap kali pemanggilan mengkonsumsi 0,185 detik.
- *cumtime*: Waktu kumulatif. Waktu ini termasuk panggilan sub, itulah sebabnya pada contoh di atas, function1 dan function2 memiliki waktu kumulatif yang mirip dengan total waktu perhitungan.
- *percall*: Terdapat 2 jenis variabel 'percall'. Yang pertama berarti total waktu per panggilan, dan yang kedua berarti waktu kumulatif per panggilan.

3. Metodologi Penelitian

3.1. Tahapan Penelitian

Tahapan-tahapan yang akan dilakukan pada penelitian ini dapat dilihat pada **gambar 3.1**.

Gambar 3.1. Tahapan Penelitian

Berdasarkan diagram pada **gambar 3.1.**, tahapan penelitian secara garis besar dapat dijabarkan sebagai berikut.

1. Pengumpulan Studi Literatur. Pada tahap ini, dilakukan pencarian literatur dan dokumentasi penelitian dari berbagai sumber terkait sistem deteksi plat kendaraan secara *realtime* berbasis Mini PC.

2. Identifikasi kebutuhan dan desain sistem. Dalam tahap ini dilakukan identifikasi data yang dibutuhkan untuk mengevaluasi kinerja sistem, alat dan bahan yang diperlukan dalam mendukung pengambilan data tersebut, termasuk pengumpulan dataset gambar plat mobil sebagai data latih pada sistem yang akan dibuat.

3. Perancangan sistem. Dalam tahap ini dilakukan desain dan perancangan sistem otomatisasi perhitungan potensi parkir.

4. Evaluasi kinerja. Setelah sistem diuji coba, maka akan dilakukan evaluasi kinerja dengan parameter rata-rata waktu yang dibutuhkan Mini PC serta kinerja CPU dalam memproses algoritma ALPR untuk menghasilkan data plat yang diinginkan.

5. Penulisan laporan. Tahap ini merupakan tahap penulisan seluruh proses penelitian yang telah dilakukan dalam bentuk skripsi. Laporan ini digunakan sebagai bahan publikasi maupun untuk acuan penelitian selanjutnya.

6. Kesimpulan penelitian. Setelah melakukan tahapan-tahapan di atas, diperolehlah kesimpulan berdasarkan penelitian yang telah dilakukan.

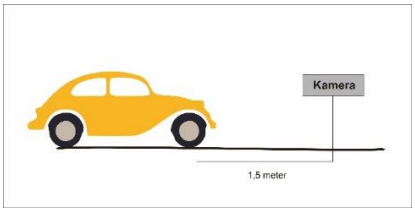
3.2. Waktu dan Lokasi Penelitian

Waktu penelitian dilakukan selama 9 bulan, dimulai sejak bulan Januari 2019 hingga proses pelaporan hasil tugas akhir ini pada bulan September 2019. Pengambilan data dilakukan di Parkiran Kampus Teknik Universitas Hasanuddin, Gowa, sedangkan Evaluasi dan pengolahan data dilakukan di Laboratorium *Internet of Things* dan *Parallel Computing* (IoTPC), Departemen Teknik Informatika, Fakultas Teknik, Universitas Hasanuddin.

3.3. Akuisisi Data

Data yang digunakan pada penelitian ini yaitu data latih berupa citra karakter plat mobil dan data uji berupa citra mobil. Data latih diperoleh dari berbagai sumber yaitu dataset penelitian terkait berupa lebih dari 1000 citra karakter. Dari data tersebut akan diolah dan dipilah kembali untuk menghasilkan satu citra latih dengan berbagai variasi karakter di dalamnya.

Sedangkan untuk citra uji diperoleh dengan melakukan pengukuran jarak antara mobil dan kamera *smartphone*. Jarak yang digunakan adalah 1,5 meter. Setelah sesuai, maka dilakukan pengambilan gambar dengan sudut 90° antara kamera dan permukaan tanah. Tinggi kamera disesuaikan sedemikian rupa sehingga mobil berada di tengah *frame.* Ilustrasi pengambilan data dapat dilihat pada **gambar 3.2.**

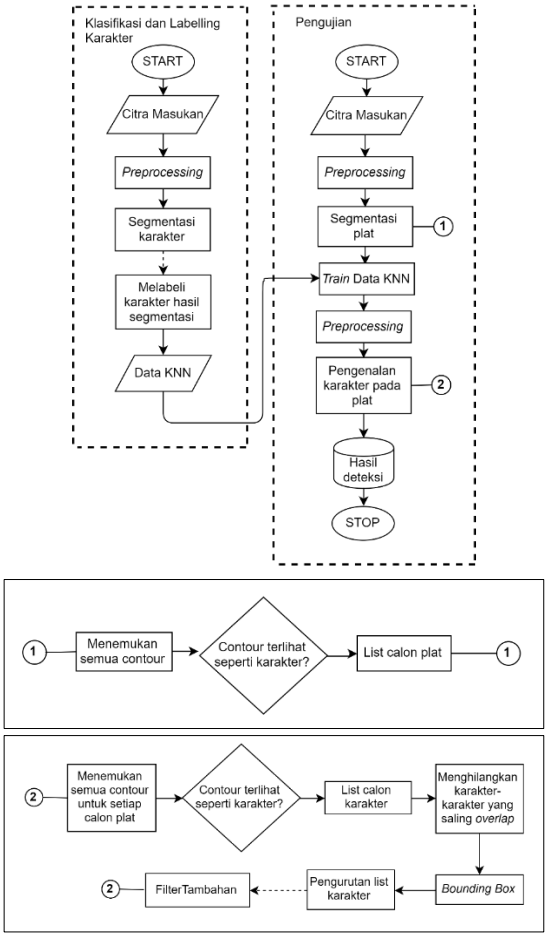


Gambar 3.2. Ilustrasi pengambilan data objek penelitian

3.4. Perancangan dan Implementasi Sistem

Sistem yang dibuat menggunakan sebuah Raspberry Pi 3 Model B+ yang digunakan sebagai *microprosessor* untuk mengolah citra masukan. Untuk bagian program, secara garis besar terbagi atas dua bagian, yaitu proses klasifikasi dan *labelling* serta proses pengujian, seperti yang diperlihatkan pada **gambar 3.3**.

Baik proses klasifikasi maupun pengujian akan melewati tahap *preprocessing* dan segmentasi. Dalam proses klasifikasi, fitur yang diesktrak yaitu berupa bentuk dan jenis karakter 0-9 hingga A-Z. Fitur yang telah diekstrak selanjutnya akan diberi label dan diklasifikasikan, kemudian hasilnya akan disimpan sebagai data untuk digunakan pada proses *training*.



Gambar 3.3. Flowchart Sistem

3.5. Evaluasi Kinerja

Proses evaluasi dilakukan dengan menggunakan parameter pengaruh nilai kernel *Gaussian Blur* terhadap akurasi pendeteksian dan waktu processing serta kinerja Mini PC dalam mengeksekusi algoritma ALPR (dengan menghitung *system time, user time*, dan *idle time*).

Untuk mengetahui waktu yang dibutuhkan program untuk mengenali karakter plat dari suatu gambar, pengujian dilakukan dengan menggunakan parameter pengaruh nilai kernel *Gaussian Blur* terhadap akurasi pendeteksian. Nilai kernel yang digunakan adalah 3x3, 5x5, 7x7, 9x9 dan 11x11. Evaluasi dilakukan dengan mengambil data citra dari database kemudian data diuji sebanyak 5 kali dengan merubah parameter nilai kernel yang digunakan dalam tahap *Gaussian Blur* (*preprocessing*).

Untuk menghitung presentase akurasi dalam mengenali karakter plat digunakan dua kondisi, yaitu pertama kondisi karakter plat dikenali dengan benar oleh sistem, dan kedua kondisi plat gagal dikenali dengan benar oleh sistem. Berikut persamaan yang digunakan berdasarkan dua kondisi tersebut.

$$Akurasi = \frac{JB}{JK}\ x\ 100\% \qquad (3.1)$$

Keterangan:

1. JB = Jumlah citra yang dikenali dengan benar
2. JK = Jumlah citra secara keseluruhan

Dalam penelitian ini, selain diperoleh jumlah citra plat yang berhasil dikenali, juga akan diperoleh lama waktu program mengeksekusi masing-masing citra. untuk mengetahui nilai kernel mana yang membutuhkan waktu eksekusi paling lama dan apakah waktu eksekusi tersebut berpengaruh terhadap tingkat akurasi .

Untuk memperoleh nilai waktu eksekusi digunakan perintah `time.time()` di bagian awal program (setelah berhasil mengakuisi citra dari database) dan dibagian akhir program (setelah *output* pendeteksian ditampilkan). Setelah itu dihitung selisih keduanya. Hal ini dilakukan untuk memperoleh waktu eksekusi yang sebenarnya tanpa dipengaruhi oleh faktor lama waktu citra diakusisi dari database. Berikut penulisan program yang dimaksudkan :

```
//Akuisisi citra dari database
Response =
requests.get('http://localhost/getImage2.php'
).text
print(response)
blnKNNTrainingSuccessful =
DetectChars.loadKNNDataAndTrainKNN()
start = time.time()  //Mulai menghitung waktu
.
.  //perintah-perintah untuk mengenali
karakter
.
.
plateResult = detectPlate(imgOriginalScene)
end = time.time() //Hentikan perhitungan
waktu
print(end-start) // Tampilkan waktu eksekusi
```

Setelah memperoleh waktu eksekusi dari program, selanjutnya dapat dilakukan profilling lebih lanjut mengenai di bagian mana saja program menghabiskan banyak waktu atau fungsi-fungsi apa saja yang paling banyak dipanggil. *Profilling* ini dilakukan sebagai referensi tambahan jika nantinya ingin mengoptimalkan kinerja dari program tersebut. Pada proses ini digunakan *built-in* modul dari python yang bernama cProfile.

Sedangkan untuk mengevaluasi kinerja Mini PC dalam penerapan algoritma pengenalan plat pada citra mobil dilakukan dengan modul *mpstat* pada terminal raspberry. Berikut penulisan perintah yang dimaksudkan :

```
pi@raspberrypi:~ $ mpstat -P ALL
```

Perintah ini akan menghasilkan informasi mengenai kinerja semua CPU dan rata-ratanya. Terdapat banyak variabel waktu, tetapi yang akan menjadi fokus utama dalam penelitian ini yaitu *system time*, *user time* dan *idle time*. *System time* adalah waktu yang dihabiskan CPU untuk menjalankan kode di kernel sistem operasi dengan melakukan tindakan yang diminta oleh proses atas nama program yang dijalankan. Waktu ini dijumlahkan dari semua CPU yang digunakan. *User time* adalah waktu yang dihabiskan oleh CPU untuk menjalankan program atau proses, dijumlahkan dari semua CPU yang digunakan. Sedangkan *idle time* adalah waktu dimana CPU aktif tetapi tidak melakukan aktifitas.

4. Hasil dan Pembahasan

Bab ini menjelaskan secara keseluruhan hasil pengujian dan evaluasi kinerja dari raspberry pi terhadap sistem pendeteksian dan pengenalan plat mobil yang telah dibuat.

4.1. Hasil Pengujian Perubahan Nilai Kernel *Gaussian Blur* terhadap Akurasi Pengenalan Plat

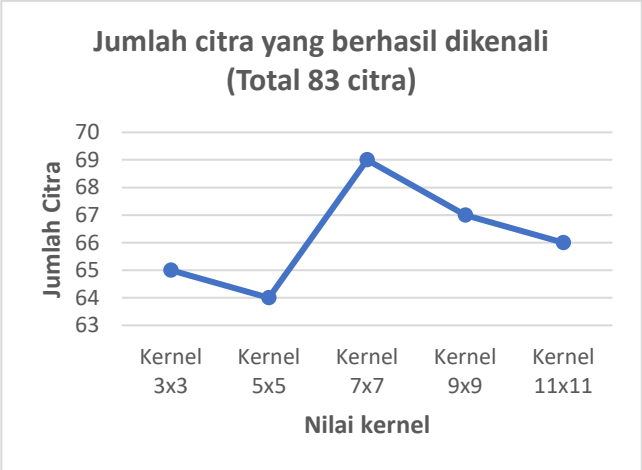
Evaluasi kinerja yang dilakukan pada sistem *processing* meliputi evaluasi kinerja terhadap kemampuan sistem dalam mendeteksi dan

mengenali karakter plat dan tingkat kesalahannya, waktu eksekusi, dan *profilling* lebih lanjut mengenai fungsi-fungsi apa saja yang paling banyak dipanggil selama eksekusi.

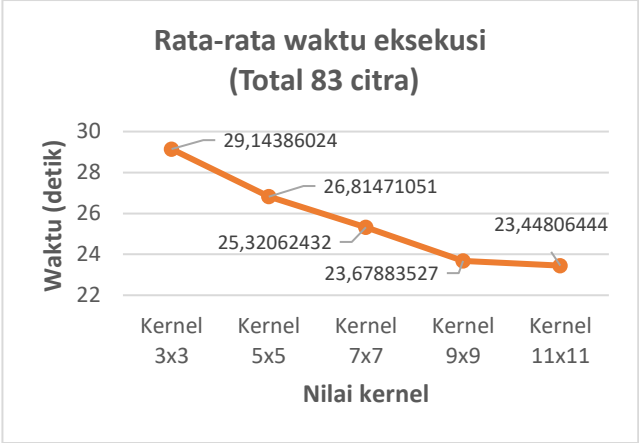
Tabel 4.1 memperlihatkan bahwa terdapat 83 data citra mobil di dalam database (yang disimbolkan dengan label berbentuk angka 1-83) . Citra-citra ini memiliki ukuran 1936 x2581 px.

Tabel 4.1 Hasil Perbandingan Akurasi dan Waktu Eksekusi

	Jumlah gambar terdeteksi dengan benar	Akurasi	Waktu (detik)
Kernel 3x3	65	78%	29.14386024
Kernel 5x5	64	77%	26.8147105
Kernel 7x7	69	83%	25.3206243
Kernel 9x9	67	80%	23,6788352
Kernel 11x11	66	79%	23.4480644



Gambar 4.1 Grafik kernel terhadap jumlah citra yang berhasil dikenali.



Gambar 4.2 Grafik kernel terhadap waktu eksekusi.

Berdasarkan **tabel 4.1**, dari 5 kali hasil pengujian pembacaan karakter plat dengan perubahan nilai kernel pada tahap *Gaussian Blur* diperoleh bahwa hasil paling optimal (plat yang terkenali paling banyak) adalah dengan menggunakan nilai kernel 7x7 yaitu berhasil mengenali 69 citra dengan benar (Akurasi 83%). Hasil ini lebih baik daripada hasil pembacaan karakter plat dengan nilai kernel lainnya. Hal ini terjadi karena untuk kernel 3x3 dan 5x5 , nilai kernel yang semakin kecil mengindikasikan bahwa proses ini membuang semakin sedikit detail atau *noise* pada citra. Akibatnya, masih terdapat bagian-bagian tidak penting pada citra hingga menyebabkan program salah mengenali karakter plat.

Untuk kernel 9x9 dan 11x11, nilai kernel yang besar mengindikasikan bahwa nilai-nilai ini membuang banyak detail [5] pada citra untuk menyatukan atau membuat citra tersebut blur. Akibatnya, banyak citra yang diuji kehilangan detail-detail penting yang seharusnya dipertahankan untuk pembacaan plat, dan menyebabkan program salah mengenali karakter plat atau bahkan tidak mampu mengenali plat sama sekali.

Sedangkan untuk nilai kernel 7x7 dianggap paling optimal karena nilai ini tidak membuang terlalu banyak detail citra juga tidak mempertahankan noise atau bagian-bagian tidak penting (nilai berada ditengah-tengah).

Dari **gambar 4.1** juga bisa dilihat bahwa bentuk grafik tidaklah signifikan, ini artinya bahwa semakin besar atau semakin kecil nilai kernel yang digunakan tidak berarti bahwa akan semakin banyak atau sedikit pula citra yang dikenali.

Dari segi waktu proses seperti pada **gambar 4.2**, diperoleh bahwa waktu eksekusi

program yang paling sedikit adalah dengan menggunakan nilai kernel 11x11 yaitu 23.44806444 detik , sedangkan waktu eksekusi paling lama adalah pada kernel 3x3 yaitu 29.14386024.

Jika dilakukan profiling lebih lanjut terhadap 2 nilai kernel tersebut , diperoleh hasil seperti pada **Gambar 4.3** dan **Gambar 4.4** .

```
         4513137 function calls (4500441 primitive calls) in 26.867 seconds

   Ordered by: internal time
   List reduced from 2554 to 15 due to restriction <15>

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
  1999559   13.976    0.000   15.240    0.000 npyio.py:683(floatconv)
     6656    3.691    0.001   18.931    0.003 npyio.py:985(<listcomp>)
  1999559    1.264    0.000    1.264    0.000 {method 'lower' of 'bytes' objects}
        4    1.104    0.276   22.416    5.604 npyio.py:710(loadtxt)
     6660    0.982    0.000    0.982    0.000 {method 'split' of 'bytes' objects}
    18/7     0.675    0.037    1.079    0.154 {built-in method _imp.create_dynamic}
       42    0.540    0.013    0.540    0.013 {built-in method numpy.core.multiarray.array}
        2    0.378    0.189    0.378    0.189 {adaptiveThreshold}
     7024    0.299    0.000    0.299    0.000 {method 'split' of '_sre.SRE_Pattern' objects}
      355    0.232    0.001    0.232    0.001 {built-in method marshal.loads}
        1    0.217    0.217    0.217    0.217 {imread}
       10    0.188    0.019    0.188    0.019 {resize}
9985/6656    0.134    0.000    0.139    0.000 npyio.py:896(pack_items)
1243/1231    0.112    0.000    0.370    0.000 {built-in method builtins.__build_class__}
     6660    0.109    0.000    1.568    0.000 npyio.py:912(split_line)
```

Gambar 4.3 Hasil profiling eksekusi program dengan kernel 3x3.

```
         4507049 function calls (4494353 primitive calls) in 26.910 seconds

   Ordered by: internal time
   List reduced from 2554 to 15 due to restriction <15>

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
  1999559   13.927    0.000   15.215    0.000 npyio.py:683(floatconv)
     6656    3.744    0.001   18.959    0.003 npyio.py:985(<listcomp>)
  1999559    1.288    0.000    1.288    0.000 {method 'lower' of 'bytes' objects}
        4    1.078    0.269   22.429    5.607 npyio.py:710(loadtxt)
     6660    1.005    0.000    1.005    0.000 {method 'split' of 'bytes' objects}
    18/7     0.686    0.038    1.092    0.156 {built-in method _imp.create_dynamic}
       42    0.545    0.013    0.545    0.013 {built-in method numpy.core.multiarray.array}
        2    0.384    0.192    0.384    0.192 {adaptiveThreshold}
     7024    0.305    0.000    0.305    0.000 {method 'split' of '_sre.SRE_Pattern' objects}
      355    0.235    0.001    0.235    0.001 {built-in method marshal.loads}
        1    0.217    0.217    0.217    0.217 {imread}
        2    0.203    0.101    0.203    0.101 {GaussianBlur}
       10    0.187    0.019    0.187    0.019 {resize}
9985/6656    0.127    0.000    0.131    0.000 npyio.py:896(pack_items)
        8    0.111    0.014    0.111    0.014 {method 'findNearest' of 'cv2.ml_KNearest' objects}
```

Gambar 4.4 Hasil profiling eksekusi program dengan kernel 11x11.

Dari **gambar 4.3** dan **gambar 4.4** disajikan 15 fungsi teratas yang paling banyak menghabiskan waktu dari total 2554 daftar fungsi. 15 fungsi ini diurutkan berdasarkan waktu (*tottime*) yang dihabiskan dari yang terbanyak hingga yang paling sedikit. Bisa dilihat bahwa fungsi yang paling banyak dipanggil baik jika menggunakan kernel 3x3 dan 11x11 adalah sama yaitu fungsi *floatconv* pada file *library* npyio.py dengan 1.999.559 pemanggilan dan menghabiskan waktu sebanyak 13,976 dan 13,927 detik. Artinya, ketika program ini dieksekusi, proses yang memakan waktu paling lama dan paling banyak adalah proses untuk mengubah bilangan hex menjadi *decimal / floating point*. Hal ini menjadi wajar jika mengingat bahwa data citra pada umumnya tersimpan dalam bentuk *byte* / heksadesimal, sedangkan untuk mengolah citra tersebut memerlukan perhitungan yang melibatkan *floating* point dan matriks.

4.2 Hasil Evaluasi Kinerja Mini PC dengan Parameter System Time, User Time dan Idle Time

Untuk mengetahui seberapa optimal Mini PC dalam mengeksekusi program yang akan mengenali plat, yang harus diperhatikan adalah waktu yang digunakan. Pada bagian ini akan disajikan hasil evaluasi kinerja Mini PC untuk masing-masing CPU ketika melakukan eksekusi terhadap algoritma ALPR dengan menggunakan nilai kernel (dengan akurasi paling optimal yaitu 7x7).

Proses pengujian dilakukan dengan dua keadaan yaitu mengamati kinerja CPU Mini PC saat tidak melakukan ekseksusi program (hanya startup program yang berjalan) dan kinerja CPU Mini PC saat mengeksekusi program. Masing-masing keadaan diamati setiap 20 detik selama program dieksekusi. Berikut hasil yang diperoleh seperti pada **gambar 4.5.**

Average:	CPU	%usr	%sys	%iowait	%soft	%idle
Average:	all	0.21	0.11	0.01	0.06	99.62
Average:	0	0.00	0.00	0.03	0.18	99.80
Average:	1	0.33	0.28	0.00	0.05	99.35
Average:	2	0.03	0.00	0.00	0.00	99.98
Average:	3	0.48	0.18	0.00	0.00	99.35

Gambar 4.5 Rata-rata kinerja Mini PC dalam keadaan hanya menjalankan *startup program*.

Dari **gambar 4.5** bisa dilihat bahwa selama Mini PC tidak melakukan apa-apa selain menjalankan *startup program*, rata-rata penggunaan keseluruhan CPU untuk *user* adalah sebesar 0.21% , untuk *system* sebesar 0.11% dan *idle* sebesar 99.62%. Untuk pembagian kinerja keempat CPU, bisa dilihat bahwa CPU yang bekerja paling banyak dalam keaadan ini adalah CPU 1 dan 3 dengan waktu *idle* terkecil diantara CPU lainnya yaitu sebesar 99,35%. Masing-masing CPU 1 dan 3 menggunakan waktu dalam mode *user (user time)* sebesar 0.33% dan 0.48%, sedangkan dalam mode kernel (*system time*) sebesar 0.28% dan 0.18%.

Average:	CPU	%usr	%sys	%iowait	%soft	%idle
Average:	all	23.40	0.46	0.01	0.02	76.11
Average:	0	4.30	0.27	0.03	0.06	95.33
Average:	1	16.83	0.61	0.00	0.02	82.54
Average:	2	46.63	0.61	0.00	0.00	52.75
Average:	3	25.66	0.35	0.00	0.00	73.99

Gambar 4.6 Rata-rata kinerja Mini PC dalam keadaan mengeksekusi program ALPR.

Dari **gambar 4.6** bisa dilihat bahwa selama Mini PC mengeksekusi program ALPR, rata-rata penggunaan keseluruhan CPU untuk *user* adalah sebesar 23.40% , untuk *system* sebesar 0.46% dan *idle* sebesar 76.11%. Keadaan ini diamati mulai pukul 07.31:18' hingga 08.18:43'.

Untuk pembagian kinerja keempat CPU, bisa dilihat bahwa CPU yang bekerja paling banyak dalam keaadan ini adalah CPU 2 dengan waktu *idle* terkecil diantara CPU lainnya yaitu sebesar 52.75%. CPU 2 ini menggunakan waktu dalam mode *user* (*user time*) sebesar 46.63% , dan dalam mode kernel (*system time*) sebesar 0.61% .

Sedangkan CPU yang bekerja paling sedikit dalam keadaan ini adalah CPU 0 dengan waktu *idle* paling besar yaitu sebesar 95.33%. Dalam hal penggunaan waktu dalam mode *user* (*user time*), CPU 0 ini menggunakan sebesar 4.30% sedangkan penggunaan *system time* sebesar 0.27%.

Dari nilai-nilai ini bisa dilihat bahwa pembagian kinerja untuk masing-masing CPU tidaklah merata, dimana ada CPU yang sibuk bekerja keras sementara CPU lainnya ada yang hampir tidak bekerja sama sekali. Kemungkinan hal ini terjadi karena program yang dieksekusi adalah program *single thread* , dimana hanya satu instruksi yang dapat dieksekusi dalam satu waktu. Berbeda dengan program *multithread* dimana beberapa proses bisa dijalankan dalam satu waktu.

Namun yang menjadi titik positifnya adalah jika membandingkan rata-rata kinerja CPU pada dua keadaan tersebut yaitu pada keadaan *startup* sebesar 0.21% dan dalam keadaan mengeksekusi program sebesar 23.40% , bisa dilihat bahwa penggunaan CPU hanya naik sebesar 23.19% dari keadaan *startup*. Artinya masih ada sekitar 76,6% dari *resource* Mini PC yang belum terpakai atau dimanfaatkan. Hal ini menjadi sisi positif karena artinya Mini PC bisa menghandle lebih banyak proses pendeteksian dalam satu waktu (jika dimanfaatkan dengan maksimal).

5. Penutup

5.1. Kesimpulan

Dari hasil analisis yang telah dilakukan dalam evaluasi kinerja Mini PC dalam menerapkan algoritma ALPR, dapat disimpulkan bahwa:

1. Perubahan nilai kernel pada tahap *preprocessing* (*Gaussian Blur*) berpengaruh terhadap tingkat akurasi pendeteksian dan pengenalan yang diperoleh, namun tidak secara linear. Berdasarkan 5 skenario nilai kernel yang digunakan akurasi terbaik sebesar 83% didapatkan dengan menggunakan nilai kernel 7x7. Nilai ini dianggap paling optimal karena nilai ini tidak membuang terlalu banyak detail citra juga tidak banyak mempertahankan noise atau bagian-bagian tidak penting (nilai berada ditengah-tengah).
2. Perubahan nilai kernel juga berpengaruh terhadap rata-rata waktu eksekusi program. Semakin besar nilai kernel yang digunakan, maka semakin cepat rata-rata waktu eksekusi. Rata-rata waktu eksekusi tercepat yang diperoleh adalah sebesar 23.44806444 detik dengan menggunakan nilai kernel 11x11.
3. Dari hasil evaluasi kinerja CPU Mini PC diperoleh bahwa rata-rata penggunaan keseluruhan CPU untuk mengeksekusi program pada mode *user* adalah sebesar 0.21% , untuk *system* sebesar 0.11% dan *idle* sebesar 99.62%. Untuk pembagian kinerja keempat CPU, pembagiannya tidak merata dimana yang bekerja paling banyak dalam keaadan ini adalah CPU 2 dengan waktu *idle* sebesar 52.75, *user time* sebesar 46.63% , *dan system time* sebesar 0.61% . Sedangkan CPU yang bekerja paling sedikit adalah CPU 0 dengan waktu *idle* sebesar 95.33%, *user time* sebesar 4.30% dan *system time* sebesar 0.27%.
4. Untuk keseluruhan pengeksekusian program ALPR, Mini PC hanya menggunakan *resource* sebanyak 23,40%, artinya ada sekitar 76,6% yang belum terpakai atau dimanfaatkan. Hal ini menjadi sisi positif karena artinya Mini PC bisa menghandle lebih banyak proses pendeteksian dalam satu waktu (jika dimanfaatkan dengan maksimal).

5.2. Saran

Sehubungan dengan selesainya penulisan skripsi ini, penulis bermaksud menyampaikan saran kepada pembaca untuk pengembangan lebih lanjut terhadap penelitian ini:

1. Untuk memaksimalkan penggunaan Mini PC dalam mengimplementasikan algoritma ALPR di lapangan sebaiknya memanfaatkan *multithreading-program* agar semua *resource* Mini PC dapat dimanfaatkan untuk mengerjakan beberapa proses sekaligus dalam satu waktu.

6. Daftar Pustaka

[1] Alim, Syahirul. 2016. Potensi Pajak Parkir Terhadap Pendapatan Asli Daerah Kota Makassar (Studi Pada PD. Parkir Makassar Raya). Skripsi Fakultas Ekonomi dan Bisnis Islam UIN Alauddin Makassar

[2] Permana, Tauriq Djasa. 2014. "Sistem Monitoring Menggunakan Mini PC Raspberry Pi". *Jurnal Teknik Komputer UNIKOM*. Vol 3 Nomor 1 (April).

[3] Tahir, Ahwan Azhari. 2017. "Deteksi Nomor Pelat Kendaraan Bergerak Menggunakan Metode *Local Binary Pattern* dan *Optical Character Recognition*". Makassar : Universitas Hasanuddin

[4] Raspberry Pi Foundation. "*About Raspberry Pi*". Diakses pada 2 Maret 2019. http://raspberrypi.org.

[5] Pradana, Rizqi Yudi. 2015. "Uji Kuantitatif Efektifitas Filter dalan Perbaikan Kualitas Citra Permukaan Jalan Raya". Yogyakarta: Universitas Muhammadiyah Yogyakarta.

[6] Reputation Datascience. 2017. *Finding Optimizations in Python with Program Profiling*. 31 Maret. Diakses 14 Agustus 2019. https://medium.com

[7] Molner, Antonio. 2019. *High-Performance Python: Part 1. Profiling*. 15 April. Diakses 14 Agustus 2019. https://medium.com