TUGAS AKHIR - EE 184801

RANCANG BANGUN *QUADCOPTER* PENGHITUNG JUMLAH KENDARAAN

Ladiva Bachrulrachman
NRP 07111540000064

Dosen Pembimbing
Ronny Mardiyanto, S.T., M.T., Ph.D.

DEPARTEMEN TEKNIK ELEKTRO
Fakultas Teknologi Elektro
Institut Teknologi Sepuluh Nopember
Surabaya 2019

TUGAS AKHIR – EE 184801

RANCANG BANGUN *QUADCOPTER* PENGHITUNG JUMLAH KENDARAAN

Ladiva Bachrulrachman
NRP 07111540000064

Dosen Pembimbing
Ronny Mardiyanto, S.T., M.T., Ph.D.

DEPARTEMEN TEKNIK ELEKTRO
Fakultas Teknologi Elektro
Institut Teknologi Sepuluh Nopember
Surabaya 2019

FINAL PROJECT – EE 184801

DESIGN AND REALIZATION OF QUADCOPTER FOR VEHICLE COUNTING

Ladiva Bachrulrachman
NRP 07111540000064

Supervisor
Ronny Mardiyanto, S.T., M.T., Ph.D.

ELECTRICAL ENGINEERING DEPARTMENT
Faculty of Electrical Technology
Sepuluh Nopember Institute of Technology
Surabaya 2019

PERNYATAAN KEASLIAN TUGAS AKHIR

Dengan ini saya menyatakan bahwa isi sebagian maupun keseluruhan Tugas Akhir saya dengan judul "**Rancang Bangun *Quadcopter* Penghitung Jumlah Kendaraan**" adalah benar – benar hasil karya yang dikerjakan secara mandiri, diselesaikan tanpa menggunakan bahan-bahan yang tidak diizinkan dan bukan merupakan karya pihak lain yang saya akui sebagai karya sendiri.

Semua referensi yang dikutip maupun dirujuk telah ditulis secara lengkap pada daftar pustaka.

Apabila ternyata pernyataan ini tidak benar, saya bersedia menerima sanksi sesuai peraturan yang berlaku.

Surabaya, 8 Juli 2019

Ladiva Bachrulrachman
NRP. 07111540000064

[Halaman ini sengaja dikosongkan]

LEMBAR PENGESAHAN

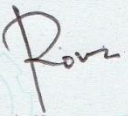
RANCANG BANGUN *QUADCOPTER* PENGHITUNG JUMLAH KENDARAAN

TUGAS AKHIR

Diajukan untuk Memenuhi Sebagian Persyaratan
Untuk Memperoleh Gelar Sarjana Teknik
Pada
Bidang Studi Elektronika
Departemen Teknik Elektro
Fakultas Teknologi Elektro
Institut Teknologi Sepuluh Nopember

Menyetujui,

Dosen Pembimbing I,



Ronny Mardiyanto S.T. ,M.T. ,Ph.D.
NIP. 198101182003121003

SURABAYA
JULI, 2019

[Halaman ini sengaja dikosongkan]

RANCANG BANGUN *QUADCOPTER* PENGHITUNG JUMLAH KENDARAAN

Nama : Ladiva Bachrulrachman
Pembimbing : Ronny Mardiyanto, S.T., M.T., Ph.D.

ABSTRAK

Setiap tahunnya terjadi peningkatan jumlah kendaraan bermotor yang digunakan pada jalan raya dan berpotensi menyebabkan kemacetan. Untuk mengatasi permasalahan kemacetan diperlukan manajemen lalu lintas dan salah satu data yang digunakan adalah perhitungan jumlah kendaraan pada tempo waktu tertentu. Untuk saat ini, informasi tersebut didapatkan dengan menggunakan tangkapan gambar menggunakan CCTV yang terpasang pada jalan tertentu. Namun, CCTV masih bersifat *fixed* yaitu tidak dapat berpindah-pindah. Selain itu, pemasangan diperlukan perencanaan yang matang serta biaya yang lebih untuk dapat diwujudkan. Oleh karena itu, penulis ingin membuat alat *quadcopter* penghitung jumlah kendaraan. *Quadcopter* memiliki beberapa kelebihan yaitu fleksibel, tangkapan gambar yang lebih luas, dan biaya yang lebih murah jika dibandingkan dengan penggunaan CCTV. *Quadcopter* akan membawa alat penghitung jumlah kendaraan ke udara. Saat di udara, *quadcopter* akan mempertahankan posisi dan ketinggiannya. Alat penghitung kendaraan akan menghitung dengan memanfaatkan *computer vision*. Alat ini dilengkapi dengan kamera yang menghadap ke bawah untuk mengambil gambar jalan raya dan memberikan output berupa jumlah kendaraan.

Tahapan metode yang digunakan pada tugas akhir ini adalah *image stabilization*, *background subtraction*, *object tracking*, dan *vehicle counting*. Hasil dari tugas akhir ini memiliki rata-rata akurasi perhitungan sebesar 61% yang dilakukan secara langsung dengan menggunakan kamera dalam 4 pengujian di 2 lokasi. Pada proses perhitungan didapatkan *error* yang disebabkan kendaraan yang tidak terdeteksi, *noise-noise* disekitar tangkapan gambar, dan kondisi lingkungan.

Kata kunci: *Quadcopter, Penghitung Jumlah Kendaraan*

[Halaman ini sengaja dikosongkan]

DESIGN AND REALIZATION OF QUADCOPTER FOR VEHICLE COUNTING

Name : Ladiva Bachrulrachman
Supervisor : Ronny Mardiyanto, S.T., M.T., Ph.D.

ABSTRACT

Number of vehicles are increasing every year. Escalation of vehicle's number can cause traffic congestion in road. Traffic management are needed to prevent traffic congestion. One of data that used for traffic management is count of vehicle in certain periods of time. As for now, that information can be acquired by using an image from CCTV which is placed in specific place. But CCTV can't be place on another place because CCTV need to placed still. Moreover, CCTV's installment needs a thorough planning and high expense. Because of that reason, writer make an innovation which is called quadcopter for vehicle counting. Benefit of using a quadcopter than CCTV are more flexible, wider coverage, and less budget are used. Quadcopter will airborne and carry device's used for vehicle counting. When quadcopter is in airborne, quadcopter will hold its position and altitude. Quadcopter will carry a vehicle counting device and works using computer vision. This device is equipped with camera that face downwards to capture image of road and gives output number of vehicles that are crossed.

Methods that are being used in this final project are image stabilization, background subtraction, object tracking, and vehicle counting. As a result, 61% count's accuracy can be obtained from livestream using camera in four different testing in two different location. Miscalculations can be occurred because vehicles not detected, noises that happens in surrounding image, and environmental conditions.

Keywords : Quadcopter, Vehicle Counting

[Halaman ini sengaja dikosongkan]

KATA PENGANTAR

Dengan mengucapkan puji syukur atas kehadirat Allah SWT atas segala rahmat dan hikmat yang telah diberikan-Nya, penulis dapat menyelesaikan tugas akhir dengan judul "Rancang Bangun *Quadcopter* Penghitung Jumlah Kendaraan". Tugas akhir ini dilakukan sebagai persyaratan untuk menyelesaikan Pendidikan *S*trata 1 di Departemen Teknik Elektro, Fakultas Teknologi Elektro, Institut Teknologi Sepuluh Nopember, Surabaya.

Dalam masa penyelesaian tugas akhir ini, penulis mendapatkan banyak bantuan dan dukungan dari berbagai pihak, Maka dari itu, penulis akan berterima kasih kepada:

1. Keluarga yang dengan setia mendukung dan memberikan perhatian dalam pengerjaan tugas akhir
2. Bapak Ronny Mardiyanto S.T., M.T., Ph.D. sebagai dosen pembimbing atas rekomendasi topik tugas akhir dengan bimbingan serta arahan dalam masa pengerjaan tugas akhir
3. Bapak Dr. Ir. Djoko Purwanto, M.Eng , Bapak Fajar Budiman, S.T., M.Sc., dan Bapak Muhammad Attamimi, B.Eng, M.Eng., Ph.D. sebagai dosen penguji tugas akhir yang memberikan saran serta pendapat untuk memperbaiki tugas akhir ini
4. Rekan-rekan laboratorium A206 dan B202 bidang studi elektronika yang selalu memberikan dukungan dan saran
5. Rekan-rekan mahasiswa angkatan 2015 yang selalu memberikan dukungan dan saran.

Penulis berharap tugas akhir ini dapat memberikan manfaat kepada pembaca. Selain itu, tugas akhir ini masih memiliki banyak kekurangan dan memberikan peluang untuk dapat dikembangkan lebih jauh lagi. Oleh karena itu, penulis bersedia untuk menerima saran yang akan diberikan oleh pembaca. Pemohon meminta maaf apabila memiliki kesalahan dalam penulisan maupun bentuk lainnya dan terima kasih atas waktunya untuk dapat membaca tugas akhir ini.

Surabaya, 8 Juli 2019

Ladiva Bachrulrachman

[Halaman ini sengaja dikosongkan]

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

[Halaman ini sengaja dikosongkan]

BAB 1
PENDAHULUAN

Topik bahasan yang akan dikerjakan pada tugas akhir ini adalah rancang bangun *quadcopter* penghitung jumlah kendaran. Sebelum dilakukan pembahasan secara mendalam metode yang akan digunakan dari judul tugas akhir ini, pada bab ini akan dijelaskan hal-hal yang mendasari pembuatan tugas akhir. Hal-hal mendasar tersebut meliputi latar belakang, perumusan masalah, tujuan penulisan, batasan masalah, metodologi penelitian, sistematika penulisan, dan relevansi

1.1. Latar Belakang

Kendaraan bermotor telah digunakan oleh masyarakat sebagai sistem transportasi untuk menjalankan kehidupan sehari-hari. Jumlah kendaraan bermotor mengalami peningkatan tiap tahunnya. Berdasarkan data tahun 2016, total kendaraan bermotor yang terdiri dari sepeda motor, mobil penumpang, mobil, bis, dan mobil barang di Indonesia mencapai 129.281.079 buah. Untuk tahun 2017, total kendaraan bermotor mencapai 138.556.669 buah[1]. Pertumbuhan jumlah kendaraan bermotor ini dapat meningkatkan potensi kemacetan pada jalan raya. Indonesia tercatat sebagai nomor 2 dari 10 negara dunia dengan tingkat kemacetan terparah versi Inrix[2]. Untuk menurunkan tingkat kemacetan yang terjadi, maka diperlukan manajemen dalam lalu lintas. Manajemen lalu lintas adalah pengelolaan serta pengaturan pada jalan raya untuk memberikan ruang gerak kendaraan secara efisien. Salah satu data yang digunakan pada manajemen lalu lintas adalah jumlah kendaraan pada tempo waktu tertentu. Data statistik berupa jumlah kendaraan bermotor akan dijadikan *Data Annual Average Daily Traffic* (AADT) dan *Average Daily Traffic (ADT)* yang akan dijadikan dasar dalam perencanaan, desain, operasi, dan manajemen sistem transportasi[3]. Data berisi jumlah kendaraan dapat dihitung berdasarkan tempo waktu tertentu mulai dari per jam hingga per tahun.

Di Indonesia salah satunya Surabaya memiliki sistem yang digunakan dalam melakukan manajemen lalu lintas yaitu SITS (Surabaya *Intelligent Transport System*). Dalam melakukan pengawasan lalu lintas, SITS mengandalkan penggunaan CCTV (*Closed Circuit Television*) sebagai bagian dari manajemen lalu lintas. CCTV mampu memberikan informasi serta data mengenai keadaan arus lalu lintas mulai dari tingkat

kepadatan kendaraan pada suatu jalan raya. Namun dalam penggunaan CCTV masih memiliki beberapa kekurangan. Kekurangan penggunaan CCTV yaitu CCTV bersifat *fixed*, yaitu tidak dapat berpindah-pindah. Dikarenakan sifat *fixed* ini, CCTV hanya terpasang pada titik-titik tertentu yang dianggap relevan. Kemudian dalam melakukan pemasangan CCTV diperlukan perencanaan secara matang apabila dilakukan dalam jalan yang kecil.

Untuk mengatasi permasalahan ini, diperlukan inovasi lain yang lebih bersifat fleksibel dalam melakukan perhitungan jumlah kendaraan. *UAV (Unmanned Aerial Vehicle)* dapat dimanfaatkan untuk melakukan perhitungan jumlah kendaraan bermotor. UAV saat ini mengalami perkembangan pesat dalam berbagai aplikasi mulai dari dunia riset, militer, fotografi dan videografi. Jenis UAV yang akan diguanakan adalah jenis *quadcopter*. Jika dibandingkan dengan penggunaan CCTV, *quadcopter* menawarkan beberapa kelebihan yaitu lebih fleksibel, biaya yang lebih murah dan tangkapan lingkup gambar yang lebih luas. *Quadcopter* lebih fleksibel dikarenakan *quadcopter* dapat dibawa ke beberapa tempat dan tidak bergantung pada luas jalan yang akan dihitung. Dikarenakan tidak bergantung pada luas jalan maka dapat dilakukan perhitungan pada jalan yang lebar dan sempit. Penggunaan *quadcopter* juga menghembat pembiayaan dikarenakan tidak membutuhkan biaya untuk pembangunan, pemasangan CCTV dan perlengkapan lainnya seperti kabel, dll. Selain itu, penggunaaan *quadcopter* juga menawarkan tangkapan gambar yang lebih luas dikarenakan tangkapan gambar diambil dari udara dan posisi kendaraan saat di jalan akan bersebelahan. Apabila dibandingkan dengan penggunaan CCTV, kendaraan bisa tertangkap gambar secara bertumpuk satu sama lain. Metode yang akan digunakan pada tugas akhir ini adalah *quadcopter* yang akan membawa alat penghitung jumlah kendaraan. *Quadcopter* akan membawa alat penghitung kendaraan ke udara. Prinsip perhitungan yang akan digunakan adalah berbasis *computer vision*. Dengan menggunakan *computer vision*, alat penghitung kendaraan dapat mengolah gambar untuk menghitung jumlah kendaraan yang berdasarkan tangkapan gambar kamera dan menghsailkan output yaitu jumlah kendaraan. Hasil data perhitungan kendaraan yang telah didapat dapat digunakan oleh pemerintahan pusat atau daerah untuk melakukan manajemen lalu lintas. Dan pada tugas akhir ini akan berfokus dalam menghitung jumlah kendaraan yang lewat pada jalan raya.

1.2. Perumusan Masalah

Berdasar pada latar belakang diatas, dapat dirumuskan permasalahan sebagai berikut:

1. Bagaimana cara mendeteksi pergerakan kendaraan di jalan?
2. Bagaimana cara menghitung jumlah kendaraan yang lewat?
3. Bagaimana cara melakukan stabilisasi gambar dikarenakan pergerakan kecil pada *quadcopter*?

1.3. Tujuan Penelitian

Penelitian pada tugas akhir ini bertujuan sebagai berikut:

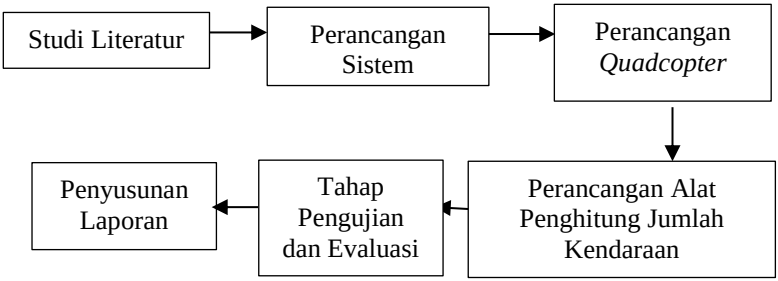
1. Dapat mendeteksi pergerakan kendaraan di jalan
2. Menghitung jumlah kendaraan yang lewat
3. Dapat melakukan stabilisasi gambar pada *quadcopter*

1.4. Batasan Masalah

Batasan masalah dari tugas akhir ini adalah

1. Kendaraan yang akan dihitung secara keseluruhan tanpa membedakan jenis kendaraan
2. Perhitungan dilakukan pada jalan raya yang lurus dan satu arah

1.5. Metodologi Penelitian



Gambar 1.1 Blok Diagram Metodologi

Gambar diatas merupakan alur dari pengerjaan tugas akhir ini. Untuk langkah-langkah yang akan dikerjakan lebih lanjut akan dijelaskan sebagai berikut :

1. Studi Literatur

Studi literatur merupakan tahap dimana dilakukan pengumpulan dan pengkajan teori, data dan penelitian yang terjamin serta relevan untuk mendukung keabsahan tugas akhir ini. Sumber yang diambil dapat berupa jurnal, buku, paper, maupun artikel yang berasal dari badan pemerintahan atau institusi yang terpecaya. Sumber-sumber yang digunakan adalah sumber dengan topik UAV dan pengolahan data dengan menggunakan *computer vision*.

2. Perancangan Sistem

   Perancangan sistem menjelaskan cara melakukan perhitungan jumlah kendaraan dengan menggunakan *quadcopter* yang ada pada jalan raya tertentu. *quadcopter* akan membawa alat yang digunakan untuk menghitung jumlah kendaraan. Saat *quadcopter* berada dalam posisi di udara, alat penghitung kendaraan akan memulai proses perhitungan jumlah kendaraan yang lewat pada jalan raya yang telah ditentukan.

3. Perancangan *Quadcopter*

   Quadcopter akan digunakan untuk membawa alat penghitung jumlah kendaraan ke udara. Pembuatan *quadcopter* akan dibagi menjadi dua bagian yaitu bagian *hardware* dan *software*. Bagian *hardware* dari *quadcopter* merupakan perancangan komponen-komponen yang akan digunakan untuk membuat *quadcopter* supaya dapat terbang. Bagian *software* merupakan perancangan program yang akan menentukan perilaku dari *quadcopter*.

4. Perancangan Alat Penghitung Jumlah Kendaraan

   Tahap ini merupakan perancangan dari alat penghitung jumlah kendaraan. Alat penghitung jumlah kendaraan memiliki tujuan untuk melakukan perhitungan jumlah kendaraan yang lewat pada jalan raya tertentu. Perancangan alat dibagi menjadi dua bagian yaitu bagian *hardware* dan *software*. Bagian *hardware* merupakan perancangan komponen-komponen yang dibutuhkan untuk menyusun alat penghitung jumlah kendaraan. Bagian *software* merupakan bagian perancangan untuk alat dapat menghitung jumlah kendaraan yang lewat. Perancangan *software* berupa perancangan program yang digunakan untuk menghitung jumlah kendaraan. Proses penghitungan akan berbasis *computer vision.* Proses perhitungan dimulai saat *quadcopter* sedang berada di udara.

   Alat penghitung jumlah kendaraan akan dilengkapi dengan kamera. Kamera akan mengambil gambar jalan raya dari udara. Kemudian tangkapan gambar tersebut akan dilanjutkan pada

program. Program yang digunakan akan melakukan *image processing* untuk dapat menghitung jumlah kendaraan. Hasil dari program adalah jumlah kendaraan yang telah terhitung.

5. Tahap Pengujian dan Evaluasi

   Tahap selanjutnya adalah melakukan pengujian terhadap sistem yang telah dirancang. Pengujian ini akan dilakukan pada tiap subsistem yang ada yaitu pengujian *quadcopter* dan pengujian alat penghitung jumlah kendaraan. Tiap pengujian akan menguji tingkat keberhasilan dari rancangan *software* dan *hardware*. Apabila terdapat kesalahan pada rancangan, maka akan dilakukan perbaikan dan evaluasi pada rancangan. Selain itu akan dilakukan analisa untuk mengetahui penyebab terjadi suatu kesalahan yang timbul pada rancangan.

6. Penyusunan Laporan

   Penyusunan laporan dilakukan seiring dengan tahapan lainnya. Laporan berisi proses pembuatan tugas akhir mulai dari awal pembuatan hingga akhir pembuatan tugas akhir.

1.6. Sistematika Penelitian

Penyusunan buku laporan Tugas Akhir disesuaikan dengan peraturan Buku Pedoman Tugas Akhir 2019 Departemen Teknik Elektro, Fakultas Teknologi Elektro, Institut Teknologi Sepuluh Nopember. Laporan ini terdiri dari tiga bagian utama, yaitu bagian awal, bagian inti/pokok, dan bagian akhir. Penyusunan ini menggunakan sistematika sebagai berikut.

- Bagian Awal

  Bagian ini terdiri dari lembar pernyataan keaslian tugas akhir, halaman pengesahan, abstrak, kata pengantar, daftar isi, daftar gambar, dan gambar tabel.

- Bagian Inti / Pokok

  Bagian ini memuat naskah utama dari Tugas Akhir diantaranya pendahuluan, kajian pustaka, metodologi penelitian, Analisa hasil dan pembahasan, dan penutup.

- Bagian Akhir

  Bagian ini memuat bahan-bahan referensi yaitu daftar pustaka dan lampiran.

1.7. Relevansi

Pengawasan arus lalu lintas dengan menggunakan *quadcopter* merupakan konsep yang masih terbilang baru di Indonesia. Konsep ini diharapkan menjadi metode baru dalam perhitungan kendaraan terutama pada daerah yang memiliki jalan yang kecil sebagai alternatif dari penggunaan CCTV. Hasil dari perhitungan tersebut dapat digunakan untuk dilakukannya manajemen lalu lintas.

BAB 2
TEORI PENUNJANG

Pada bab ini akan membahas beberapa teori dasar yang akan digunakan pada Tugas Akhir ini. Setiap teori dasar akan digunakan untuk menyelesaikan beberapa permasalahan yang akan dialami selama masa pengerjaan. Selain itu juga akan dilakukan tinjauan pustaka dimana akan membahas riset-riset sebelumnya yang hampir sama dengan tugas akhir ini.

2.1. *Quadcopter*

Quadcopter merupakan salah satu jenis dari sebuah UAV (*Unmanned Aerial Vehicle*). UAV (*Unmanned Aerial Vehicle*) adalah pesawat tanpa awak yang dapat dik*end*alikan secara jarak jauh oleh user atau yang sudah bersifat otomatis. Aplikasi UAV telah digunakan secara luas dengan kepentingan yang berbeda-beda diantaranya sistem pengawasan, videografi, fotografi, dll. Berdasarkan konfigurasi pada *frame* UAV, UAV dapat dibedakan menjadi 3 bagian yaitu VTOL (*Vertical Take-off and Landing*), HTOL (*Horizontal Take-off and Landing*), *Hybrid* (Kombinasi VTOL dan HTOL). HTOL merupakan UAV yang membutuhkan melakukan akselerasi secara horizontal pada sebuah runway untuk mendapatkan kecepatan terbang. VTOL merupakan jenis UAV yang melakukan proses take-off dan landing secara vertikal. Untuk konfigurasi pada jenis VTOL, UAV dapat dibedakan kembali menjadi beberapa jenis berdasarkan jumlah rotor yang akan digunakan, yaitu *Single-main rotor, Tandem rotor, Coaxial rotor, Tri-rotor, dan Quad-rotor*[4]. Quad-rotor atau *Quadcopter* adalah UAV yang berbentuk VTOL dan memiliki konfigurasi 4 rotor. Komponen-kompenen yang menyusun dari *quadcopter* adalah sebagai berikut.

1. *Flight Controller*

   Flight Controller merupakan sirkuit board kecil yang memiliki fungsi sebagai otak dari *quadcopter*. Flight Controller akan mengendalikan kecepatan dari motor quadcopoter. Flight controller akan menerima perintah dari operator yang akan dilanjutkan menuju motor untuk mengendalikan kecepatannya sesuai dengan perintah operator. Flight controller dilengkapi dengan sensor-sensor diantarnya barometer, akselerometer, dan gyroscope. Selain itu dapat dilengkapi dengan tambahan lain seperti GPS, kompas, servo, dll.

2. *Electronic Speed Controller (ESC)*
   Electronic Speed Controller (ESC) merupakan sirkuit elektronik yang digunakan untuk mengendalikan kecepatan dari motor. Pada ESC memiliki sebuah rating yang berbeda dimana rating tersebut menunjukkan nilai arus maksimum.
3. Motor
   Motor yang telah terpasang dengan propeller berfungsi untuk memberikan daya dorong pada *quadcopter* untuk dapat terbang. Daya dorong ini menyebabkan *quadcopter* dapat terbang secara vertikal.
4. *Transmitter dan Receiver*
   Pada komunikasi radio terdapat dua komponen utama yaitu transmitter dan receiver. Transmitter akan mengirimkan sinyal sedangkan receiver akan menerima sinyal dari transmitter. Pada *quadcopter*, transmitter akan digunakan oleh operator untuk memberikan perintah kepada *quadcopter* kemudian perintah tersebut akan diterima oleh receiver yang telah terpasang pada *quadcopter*.
5. GPS (*Global Positioning System*)
   GPS merupakan sistem navigasi yang menggunakan satelit dengan tujuan untuk memberikan informasi posisi dari suatu obyek di darat. GPS pada *quadcopter* menggunakan receiver untuk mengambil sinyal dari satelit untuk mendapatkan informasi posisi, kecepatan, dan waktu. GPS pada *quadcopter* dapat digunakan untuk beberapa fungsi diantaranya *position hold, return to home*, dan *autonomous flight*. *Position hold* merupakan fungsi yang digunakan untuk *quadcopter* dalam mempertahankan posisi dan ketinggian tertentu. *Return to home* merupakan fungsi dimana *quadcopter* dapat kembali ke posisi awal take-off secara otomatis. *Autonomous flight* merupakan fungsi *quadcopter* untuk dapat terbang secara otomatis setelah dilakukan penentuan jalur terbang oleh operator.

2.2. Gimbal

Gimbal merupakan alat yang berporos dimana digunakan untuk mempertahankan posisi orientasi dari instrumen yang terpasang pada gimbal. Pada *quadcopter*, gimbal digunakan untuk mempertahankan posisi kamera datar. Gimbal pada *quadcopter* bisa memiliki 2-axis maupun 3-axis stabilisasi dan menggunakan motor brushless pada tiap axis. Untuk gimbal 3-axis, motor tersebut akan mempertahankan kamera

untuk tetap datar dengan bergerak secara 3 dimensi (atas/bawah, kanan/kiri, depan/belakang).

2.3. iNAV

iNav merupakan *software* yang digunakan secara luas dan berfokus pada fitur GPS mulai dari UAV bermodel pesawat dan multirotor. Konfigurasi yang digunakan dilakukan mulai dari flash *firmware* sampai konfigurasi *flight controller* sesuai penggunaan. Konfigurasi ini dapat dilakukan dengan menggunakan *software* yaitu iNav Configurator. Dengan menggunakan iNav Konfigurator dapat dilakukan pengaturan perilaku dari UAV sesuai dengan keinginan pengguna[5].

2.4. Raspberry Pi 3 B+

Raspberry Pi 3 B+ merupakan sebuah *minicomputer* yang dijalankan dengan menggunakan 64-bit *quadprocessor* 1,4 Ghz. Raspberry Pi 3 B+ dilengkapi dengan 1 GB LPDDR2 SDRAM, 2.4GHz dan 5GHz IEEE 802.11.b/g/n/ac *Wireless* LAN, Bluetooth 4.2, BLE, Ethernet, 40 pin GPIO, HDMI, 4 port USB 2.0, port kamera, dan port micro-SD[6]. Raspberrry Pi banyak digunakan dalam beberapa aplikasi dimulai dari IoT (*Internet of Things*), robotik, otomasi rumah, dan otomasi industri.

Gambar 2.1 Raspberry Pi 3 B+

2.5. Raspberry Pi *Camera Module*

Raspberry Pi *Camera Module* merupakan modul kamera yang disediakan oleh Raspberry Pi. Modul ini akan terpasang pada port yang telah tersedia pada minicomputer Raspberry Pi. Modul kamera ini dapat digunakan untuk mengambil video HD dan foto. Modul kamera telah digunakan secara luas mulai dari penggunaan untuk *slow motion, time-lapse*, dan lainnya. Modul kamera dapat digunakan untuk seluruh model Raspberry Pi mulai dari 1,2, dan 3. Pengaksesan dapat dilakukan dari MMAL dan V4L API. Kamera ini juga merupakan kamera 5 MP 1080p *Focal Adjustable*. Spesifikasi dari modul kamera adalah sebagai berikut.

1. Sensor : OV5647
2. Piksel : 5 MP
3. Besar CCD : ¼ inch
4. Panjang *Focal* : 3,6 mm *Adjustable*
5. Sudut Diagonal : 60 derajat
6. Dimensi : 25 mm x 24 mm

2.6. Baterai LiPo

Baterai LiPo merupakan baterai yang dapat diisi ulang yang tersusun dari lapisan polimer yang membungkus baterai lithium-ion. Baterai ini dapat digunakan sebagai suplai tegangan untuk *quadcopter* dan Raspberry Pi. Tegangan pada baterai LiPo bergantung pada jumlah dari sel secara seri. Setiap sel memiliki tegangan sebesar 3,7 V. Label yang menandakan dari jumlah sel dari baterai yaitu 1S, 2S, 3S, dan seterusnya. Tegangan 3,7 V merupakan tegangan ideal dari bekerjanya baterai LiPo. Tengangan maksimum dari satu sel adalah 4,2 V. Baterai LiPo memiliki tegangan minimum yang dikatakan sebagai tegangan cut-off. Tegangan cut-off yang ditetapkan dari pabrik adalah 3 V. Kemudian pada baterai LiPo terdapat nilai kapasitas. Nilai kapasitas diukur dengan satuan Amp-hrs(Ah) atau milliamp-hrs(mAh). Kapasitasi ini menunjukkan jumlah energi yang dapat disimpan pada baterai LiPo. Semakin besar kapasitas dari baterai makan semakin lama baterai akan bertahan.

2.7. UBEC (*Universal* BEC)

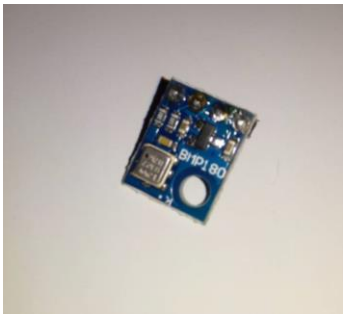
UBEC (*Universal* BEC) merupakan BEC (*Battery Elimination Circuit*) yang digunakan untuk menurunkan tegangan.dari baterai LiPo. BEC merupakan nama lain dari regulator tegangan. Tegangan besar yang dikeluarkan dari baterai LiPo akan diturunkan menjadi tegangan yang lebih kecil dengan menggunakan UBEC.

2.8. BMP180

BMP180 merupakan sensor tekanan barometrik dengan menggunakan komunikasi I2C. Sensor ini akan melakukan kalkulasi pada tekanan absolut udara di lingkungan sekitar. Tekanan ini memiliki nilai yang bervariasi bergantung pada keadaan cuaca dan ketinggian. Dengan nilai tekanan, sensor ini dapat digunakan untuk mencari nilai ketinggian, tekanan, temperatur, dan hal lainnya dengan tingkat akurasi tertentu. Untuk mendapatkan nilai dari ketinggian geopotensial (*geopotential height)*, maka dari itu akan digunakan rumus yang akan ditunjukkan berikut.

$$ketinggian\ (h) = 44330 * \left(1 - \left(\frac{p}{p_o}\right)^{\frac{1}{5.255}}\right),$$

$$dimana\ p = tekanan\ terukur\ , p_o = 1013.25\ hP$$



Gambar 2.2 BMP180

2.9. Filter Eksponensial

Filter eksponensial digunakan sebagai filter sinyal yang memiliki nilai *noise* yang sangat tinggi. Penggunaan filter ini digunakan untuk memperbaiki nilai BMP180 yang memiliki tingkat *noise* yang sangat tinggi. *Noise* yang dikeluarkan dari sensor akan diproses dengan menggunakan filter eksponensial yang ditunjukkan dengan persamaan sebagai berikut

$$altitude\ filter_n = altitude * \alpha + (1 - \alpha) * altitude\ filter_{n-1}$$

Nilai α merupakan nilai konstanta dengan *range* 0 sampai 1. Penentuan nilai α bergantung dengan hasil *output* dari sensor. Apabila nilai *noise* bernilai tinggi maka nilai α lebih optimal mendekati nol. Namun sebaliknya jika *noise* tidak bersifat dominan, maka nilai α lebih optimal jika mendekati satu[7].

2.10. Pengolahan Citra (*Image processing*)

Pengolahan Citra atau *Image processing* merupakan proses pengolahan gambar yang didapat dalam bentuk video/foto dan hasil data olahan tersebut digunakan untuk tujuan tertentu[9, hlm. 3]. Untuk setiap video/foto yang tertangkap oleh kamera, maka akan menghasilkan sebuah gambar atau biasa dikenal dengan kata *frame*. Setiap *frame* ini memiliki deretan nilai-nilai piksel yang mewakili nilai warna yang ada pada *frame*. Kemudian, nilai-nilai pada piksel ini akan diproses untuk dapat diambil data/informasi yang diinginkan.

2.10.1. *Background subtraction*

Background subtraction banyak digunakan pada beberapa aplikasi diantaranya adalah penggunaan keamanan. Cara kerja dari *background subtraction* adalah dengan membandingkan gambar *background* dengan gambar yang ada saat sekarang. Kemudian gambar sekarang akan dikurangi dengan gambar *background*. Obyek yang dihasilkan dari pengurangan tersebut merupakan hasil dari *background subtraction* atau dikenal dengan kata obyek *foreground*. Hasil pengurangan ini dapat digunakan sebagai identifikasi pergerakan benda[9].

2.10.2. Konversi Warna RGB menjadi *GRAY*

Warna RGB (*Red*,*Green*,*Blue*) merupakan warna yang terdiri dari 3 channel warna dimana masing-masing warna (contoh merah) memiliki nilai piksel tersendiri pada suatu gambar. Namun untuk warna *GRAY* hanya memiliki 1 channel warna. Untuk melakukan konversi dari RGB menjadi *GRAY* adalah menggunakan rumus sebagai berikut

$$RGB[A]\ to\ GRAY = Y \leftarrow 0{,}299 \cdot R + 0{,}587 \cdot G + 0{,}114 \cdot B$$

[9]

2.10.3. *Contour*

Contour adalah sekumpulan titik yang mempresentasikan kurva pada gambar yang memiliki warna atau intensitas yang sama. *Contour* merupakan tools yang berguna pada analisis bentuk, deteksi obyek, dan rekognisi obyek. Hasil penyatuan *contour* akan membentuk sebuah area yang dinamakan sebagai *blob* [9].

2.10.4. Filter

Filter adalah algoritma yang dimulai dengan gambar I(x,y) yang dilanjutkan dengan komputasi untuk menghasilkan gambar baru I'(x,y). Komputasi yang dilakukan pada setiap lokasi piksel (x,y) di gambar I' dengan fungsi tertentu di rentang area tertentu. *Template* yang menjelaskan rentang area tertentu ini akan dikombinasikan yang disebut sebagai filter / kernel. Kernel yang banyak digunakan pada pengolahan citra adalah linear kernel dan didefinisikan pada persamaan seperti :

$$I'(x,y) = \sum_{i,j \in kernel} k_{i,j} \cdot I(x+i, y+j)$$

Dari rumus diatas dapat dijelskan perkalian nilai pada $k_{i,j}$ dengan nilai pada piksel di gambar I dengan offset dari x,y oleh nilai i,j. Ukuran dari array I dinamakan *support of kernel*. Kemudian terdapat juga istilah anchor points yang didefinisikan sebagai titik dimana kernel akan diselaraskan dengan sumber gambar[9].

2.10.4.1. *Gaussian Blur*

Blur atau *Smoothing* digunakan untuk meminimalisir *noise* yang ada. Hasil output yang dikeluarkan pada setiap piksel merupakan rata-rata pada semua piksel yang ada pada kernel pada gambar input[9]. Salah satu tipe dari *blur* adalah *Gaussian Blur*. *Gaussian Blur* melakukan konvolusi pada tiap titik di array input yang kemudian dinormalisasi dengan Gaussian kernel dan ditambahkan untuk menghasilkan *array output* [9]. Parameter pada besar kernel memiliki lebar dan panjang. Parameter selanjutnya adalah nilai sigma pada kernel gaussian di dimensi-x. Parameter selanjutnya adalah nilai sigma pada dimensi-y. Apabila yang diatur hanya nilai x dan nilai y bernilai 0 (*default*), maka nilai y dan x akan menjadi sama. Apabila nilai x dan y bernilai 0, maka parameter gaussian akan dilakukan secara otomatis sesuai dengan besaran *window* dengan menggunakan rumus dibawah ini.

$$\sigma_x = \left(\frac{n_x - 1}{2}\right) \cdot 0{,}30 + 0{,}80\,, n_x = ksize.width - 1$$

$$\sigma_y = \left(\frac{n_y - 1}{2}\right) \cdot 0{,}30 + 0{,}80\,, n_y = ksize.height - 1$$

2.10.4.2. *Threshold*

Threshold merupakan operasi konvolusi sederhana dengan menggunakan kernel 1x1 dan melakukan operasi terhadap satu piksel. *Threshold* memiliki beberapa tipe yaitu *THRESH_BINARY*, *THRESH_BINARY_INV*, *THRES_TRUNC*, *THRESH_TOZERO*, dan *THRESH_TOZERO_INV* . Hasil yang didapatkan dari penggunaan threshold adalah nilai biner berupa 1 dan 0.[9].

2.10.4.3. *Dilate*

Dilate merupakan konvolusi gambar dengan kernel dimana nilai apapun yang diberikan akan diganti dengan nilai maksimum lokal yang terdapat pada dalam kernel tersebut. Hasil akhir dari *dilate* adalah perluasan pada area *contour* [9]. Berikut merupakan rumus dari *dilate*

$$dilate(x, y) = \max_{(i,j) \in kernel} src(x + i, y + j)$$

2.10.4.4. *Erode*

Erode merupakan konvolusi gambar dengan kernel dimana nilai apapun yang diberikan akan diganti dengan nilai minimum lokal dari dalam kernel tersebut. Hasil akhir dari *erode* adalah pengecilan pada area *contour* [9]. Berikut merupakan rumus dari *erode*.

$$dilate(x, y) = \min_{(i,j) \in kernel} src(x + i, y + j)$$

2.10.5. Lucas Kanade untuk *Optical Flow*

Lucas Kanade merupakan algoritma yang biasa digunakan untuk *optical flow*[9]. *Optical flow* digunakan untuk mengetahui lokasi banyak titik yang ada pada gambar saat ini bergerak di gambar selanjutnya yang terjadi pada gambar sekuensial di video. *Optical Flow* digunakan sebagai estimasi gerakan sebuah obyek pada gambar. *Output* ideal pada algoritma *optical flow* adalah menghitung perpindahan vektor dari setiap piksel dalam gambar yang mengindikasi lokasi relatif pada gambar selanjutnya.

Peristiwa ini biasa dinamakan *dense optical flow*. Namun ada alternatif lain yaitu *sparse optical flow*. *Sparse optical flow* adalah melacak sebagian subset dari point pada sebuah gambar. Algoritma ini terbilang lebih cepat karena membatasi jumlah point yang harus dilacak sehingga lebih mudah untuk melacak. Algoritma dari Lucas Kanade berdasarkan pada 3 asumsi yaitu

1. Kekonstanan kecerahan(*Brightness Constancy*)
   Piksel obyek pada gambar tidak akan berubah saat berpindah dari*frame*awal ke*frame*selanjutnya.
2. Kegigihan sementara *(Temporal Persistence*) atau pergerakan kecil
   Pergerakan pada gambar akan bergerak secara pelan seiring waktu.
3. Koheren spasial (*Spatial Coherence*)
   Poin yang berdekatan antara satu sama lain di *frame* akan memiliki pergerakan yang sama dan akan mengarah ke pointer terdekat pada *frame* selanjutnya

2.10.6. *Convex Hull*

Convex hull merupakan algoritma yang digunakan untuk memperbaiki *contour* yang telah didapatkan pada sebuah gambar. *Convex hull* dinyatakan untuk mengubah *contour* menjadi bentuk poligonal. Algoritma pada *convex hull* menggunakan algoritma Sklanksky yang memiliki O(N logN)[10].

2.11. OpenCV

OpenCV (*Open Source Computer Vision Library*) adalah *library* yang bersifat *open source* yang digunakan pada *computer vision* dan *machine learning*. OpenCV telah memiliki lebih dari 2500 algrotima mulai dari identifikasi obyek, pelacakan pergerakan, deteksi dan rekognisi muka, dsb. Bahasa pemrograman yang digunakan mulai dari C++, Python, Java, dan MATLAB dan m*end*ukung sistem operaso yaitu Windows, Linux, Android, dan MacOS[8].

2.12. Tinjauan Pustaka

Tinjauan pustaka digunakan untuk menjadi sebuah acuan untuk tugas akhir ini dengan perangkat yang telah ada dan sudah dikembangkan sebelumnya. Berikut merupakan judul-judul *paper* yang akan diajukan sebagai acuan tugas akhir ini.

1. *Vehicle Detection and Counting from Video Frame*[11]

Riset digunakan untuk mengembangkan deteksi kendaraan dan sistem penghitungan dengan menggunakan pengolahan citra. Seluruh pengembangan ini berbasis *software* yang membutuhkan streaming video dan penangkapan pada *frame* video. *Frame* tersebut berisi komponen dimana jalan tanpa ada kendaraan yang bergerak dan *frame* dimana terdapat kendaraan bergerak. Hasil dari pengujian mencapai tingkat presisi 91.98 % dan 96.35%.

2. *UAV Video Processing for Traffic Surveillence with Enhanced Vehicle Detection*[12]

   Untuk menggantikan pengawasan kendaraan dengan menggunakan kamera yang bersifat statis, UAV menjadi terobosan baru untuk melakukan pengawasan lalu lintas. Kerangka kerja terdiri dari empat tahap. Tahap sebelum memasuki kerangka kerja adalah *Adaptive Gamma Correction* untuk peningkatan kontras. Tahap pertama dari kerangka kerja adalah KLT *Tracker* untuk pelacakan pergerakan kendaraan. Tahap kedua adalah klasifikasi kendaraan. Tahap ketiga adalag melakukan perhitungan kendaraan yang telah terdeteksi. Tahap terakhir adalah perhitungan kecepatan rata-rata dan kepadatan kendaraan.

3. *Monitoring Road Traffic with a UAV-based System*[13]

   Paper ini membentuk sistem keadaan arus lalu lintas dengan menggunakan beberapa *drone*. Metode dikembangkan dengan membentuk lintasan adaptif UAV dimana berdasarkan pelacakan dari titik yang bergerak dalam pandangan UAV. Selain itu, lintasan ini menggunakan model mobilitas yang biasa digunakan untuk model mobilitas kendaraan. UAV melakukan *monitor* pada jalan kota dan bertugas untuk mengambil dan mengirimkan informasi kendaraan secara *real time*. Hasil informasi tersebut digunakan untuk keperluan regulasi lalu lintas.

4. *A UAV-based Traffic Monitoring System*[14]

   Penelitian ini menggunakan sistem pengawasan arus lalu lintas secara fotografi udara. Kamera pada UAV akan mengambil video arus lalu lintas yang kemudian dikirim dan diproses didalam *cloud*. Implementasi dari rancang bangun dengan menggunakan *quadcopter* yang dilengkapi dengan kamera. Selain itu dilengkapi dengan algoritma proses video dan data, serta aplikasi *web*. Untuk modul proses video berisi deteksi kendaraan berbasis model *Haar cascade* dan pelacakan per *frame*.

5. *Urban Traffic Density Estimation Based on Ultrahigh-Resolution UAV Video and Deep Neural Network* [15]

   Paper mempresentasikan solusi dalam estimasi kepadatan lalu lintas pada jalan raya dengan metode deep learning. Tahap pertama yaitu mengambil rekaman lalu lintas selama satu jam dengan resolusi sangat tinggi pada lima persimpangan jalan raya dengan menggunakan UAV pada jam sibuk kerja. Kemudian dilakukan sampling sebanyak 17.000 buah 512x512 piksel potongan kecil dari *frame* video dan secara manual sebanyak 64.000 kendaraan yang digunakan sebagai dataset. *Deep learning* yang digunakan adalah *Deep Neural Network* (DNN) berbasis deteksi kendaraan dan lokalisasi, rekonginisi tipe (mobil, bis, dan truk), tracking, dan perhitungan kendaraan .
6. *Automatic Car Counting Method for Unmanned Aerial Vehicle Images* [16]

   Paper ini akan melakukan deteksi dan perhitungan mobil dengan menggunakan UAV. Metode yang dilakukan dimulai dengan melakukan proses screening pada zona beraspal untuk membatasi area yang akan dilakukan deteksi kendaraan dan mereduksi kesalahan. Kemudian dilakukan proses ekstraksi fitur dengan *Scalar Invariant Feature Transform* yang akan menjadi sebuah set *keypoints*. Kemudian dilakukan pembedaan antara *keypoints* yang diberikan sebagai kendaraan dari lainnya, yaitu dengan menggunakan *support vector machine classifer*. Tahap terakhir berfokus pada mengelompokkan *keypoint* yang dimiliki oleh satu mobil untuk menjadikan "satu *keypoint* – satu mobil". Hasil output yang dihasilkan adalah jumlah mobil pada suatu gambar.
7. *Car Park Occupancy Analysis using UAV Images*[17]

   Analisa pada hunian pada tempat parkir berguna untuk pemilik yang berwenang untuk membuat sebuah keputusan dalam desain, perencanaan, dan manajemen pada tempat parkir. Dikarenakan perkembangan penggunaan UAV yang populer digunakan untuk pengaplikasian kegiatan sipil, maka dari itu dilakukan pemanfaatan penggunaan UAV untuk mengetahui jumlah mobil yang terparkir pada periode tertentu. Proses perhitungan dilakukan dengan menggunakan estimasi kepadatan. Perhitungan jumlah kendaraan diubah menjadi estimasi densitas pada piksel sebuah gambar.
8. *A Fast Screening Method for Detecting Cars in UAV Images Over Urban Areas*[18]

Paper ini merepresentasikan metode cepat dalam melakukan *screening* untuk mengisolasi area aspal dengan menggunakan gambar yang ditangkap menggunakan UAV. *Screening* merupakan kunci dalam melakukan standar deteksi dan perhitungan kendaraan untuk mengembangkan waktu komputasi dan mengurangi kesalahan. Metode yang digunakan adalah membagi gambar original dari UAV menjadi *tiles*. Setiap *tiles* memiliki informasi warna dari gambar. Kemudian dilakukan ekstraksi dan komparasi pada *training library* untuk mencari *tiles* yang sama.

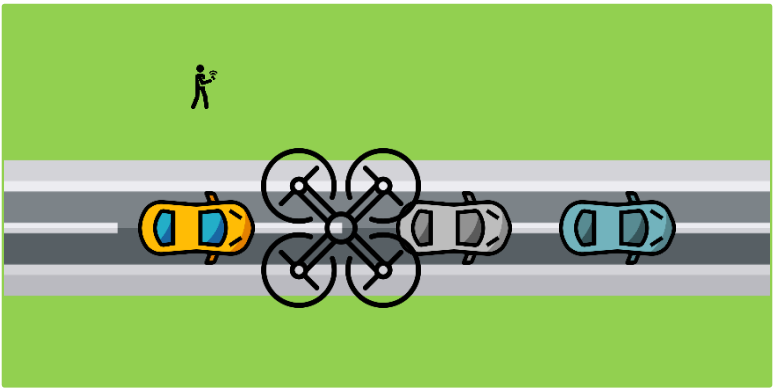
9. *Motion-Vector Clustering for Traffic Speed Detection from UAV Video*[19]

   Paper ini akan merepresentasikan metode untuk deteksi kecepatan rata-rata dari lalu lintas melalui video udara yang tidak stasioner. Metode pertama yaitu melakukan ekstrak *interest points* dari sepasang *frame* dan melakukan pelacakan pada poin tersebut dengan menggunakan algoritma *optical flow*. Output dari *optical flow* adalah sebuah set vektor pergerakan yang merupakan rata-rata *k* pada kelompok kecepatan. Nilai tengah dari kelompok kecepatan tersebut korespon dengan rata-rata kecepatan dari lalu lintas.

10. *Vehicle Detection in UAV Traffic Video Based on Convolutional Neural Network* [20]

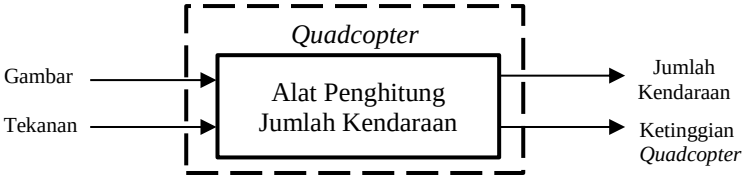
    Teknologi deteksi kendaraan merupakan komponen penting pada *intelligent transportation system*, namun untuk saat ini masih berbasis kamera pengawasan jalan. Jika dibandingkan dengan kamera *fixed*, UAV menawarkan keuntungan yaitu lebih fleksibel, tangkapan gambar yang lebih luas, dan lebih cepa sehingga deteksi kendaraan akan menjadi tantangan. Metode yang digunakan adalah dataset baru dibuat berdasarkan video lalu lintas melalui UAV dan *neural network*. *Network* yang dibuat akan menggabungkan fitur dari beberapa layer. Kemudian layer konvolusi digunakan untuk memperkecil dimensi fitur dan layer dekonvolusi digunakan untuk melakukan *upsampling* dan meningkatkan informasi respon. Tahap terakhir yaitu beberapa layer yang terkoneksi akan digunakan untuk melakukan deteksi.

BAB 3
PERANCANGAN SISTEM



Gambar 3.1 Ilustrasi Rancangan Sistem

Alat yang akan dibuat pada tugas akhir ini adalah *quadcopter* penghitung jumlah kendaraan. Sistem akan dibagi menjadi dua subsistem besar yaitu *quadcopter* dan alat penghitung jumlah kendaraan. Berdasarkan gambar 3.1, *quadcopter* akan membawa alat penghitung jumlah kendaraan ke udara. Kemudian *quadcopter* akan mempertahankan ketinggian serta posisinya saat di udara. Posisi *quadcopter* saat di udara akan berada diatas jalan raya. Kemudian alat penghitung kendaraan yang dibawa bersama *quadcopter* akan memulai proses perhitungan jumlah kendaraan yang lewat. Alat penghitung jumlah kendaraan akan dilengkapi dengan kamera dan sensor tekanan. Kamera digunakan untuk mengambil gambar dari jalan raya dan sensor tekanan digunakan untuk mendapakan nilai ketinggian dari *quadcopter* saat dilakukan proses perhitungan kendaraan. Kamera akan diarahkan ke jalan raya dengan menghadap kebawah. Hasil dari alat penghitung kendaraan adalah jumlah kendaraan yang lewat dan ketinggian *quadcopter* saat melakukan proses perhitungan jumlah kendaraan. Diagram blok keseluruhan rancangan dari sistem ini adalah sebagai berikut



Gambar 3.2 Diagram Blok Keseluruhan Rancangan Sistem

Pada setiap susbsistem akan memiliki komponen masing-masing yang akan menyusun terbentuknya subsistem. Komponen-komponen yang akan menyusun dari *quadcopter* serta alat penghitung jumlah kendaraan adalah sebagai berikut

Tabel 3.1 Komponen *Quadcopter* dan Alat Penghitung Jumlah Kendaraan

Subsistem	Komponen	Nama Komponen
Quadcopter	*Frame*	*Frame* F450
	Flight controller	Airbot Omnibus F4 V6
	Motor	*Brushless* Motor RacerStar BR2212 930 kV
	ESC	FuriBee H32 BLHeli-32 30A *Brushless* ESC
	Transmitter	Taranis QX7
	Receiver	Frsky FX400R 2.4 GHz
	GPS	Ublox NEO M8N GPS
	Propeller	Propeller 10 x 4.5”
	Baterai	Baterai LiPo 3S 1800 mAh
	Gimbal	Feiyu Tech Mini 3D 3-Axis *Brushless* Gimbal
	Frame	*Frame* F450
	Flight controller	Airbot Omnibus F4 V6
	Motor	*Brushless* Motor RacerStar BR2212 930 kV
Alat Penghitung Jumlah Kendaraan	Minicomputer	Raspberry Pi 3 B+
	Kamera	Raspberry Pi Camera
	UBEC	
	Sensor Tekanan	BMP180

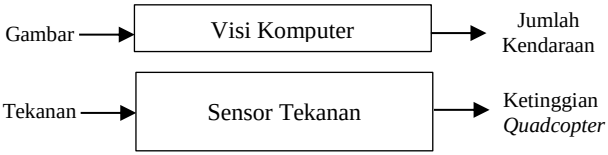
	Baterai	Baterai LiPo 3S 1800 mAh

3.1. Quadcopter

Quadcopter memiliki tujuan untuk membawa alat penghitung jumlah kendaraan ke udara. Kemudian *quadcopter* tersebut perlu ditempatkan pada posisi yaitu diatas jalan raya. Setelah ditempatkan diatas jalan raya, *quadcopter* perlu berada dalam posisi serta ketinggian yang tetap. Ini dikarenakan untuk membantu alat penghitung jumlah kendaraan dalam menghitung jumlah kendaraan yang lewat pada jalan raya. Untuk dapat mempertahankan ketinggian serta posisi dari *quadcopter*, maka perlu ada pengaturan pada flight controller. Pengaturan tersebut dapat dilakukan dengan menggunakan software iNav pada bagian flight modes. *Flight modes* merupakan mode-mode terbang yang dapat diaktifkan. Dikarenakan *quadcopter* diperlukan dapat mempertahankan ketinggian dan posisi yang tetap saat terbang, maka dari itu perlu dilakukan konfigurasi *flight modes*. Mode-mode yang akan digunakan untuk terwujudnya kondisi tersebut adalah mode *altitude hold* dan *position hold*. Mode *altitude hold* merupakan mode yang akan mempertahankan ketinggian dari *quadcopter* sedangkan mode *position hold* merupakan mode yang akan mempertahankan posisi dari *quadcopter*. *Altitude hold* dapat terjadi dengan menggunakan sensor barometer yang terdapat pada *quadcopter*. Kemudian *position hold* dapat dilakukan dengan menggunakan GPS yang telah terpasang pada *quadcopter*. GPS tersebut akan memberikan nilai koordinat posisi *quadcopter* saat terbang.

3.2. Alat Penghitung Jumlah Kendaraan

Alat penghitung kendaraan memiliki tujuan yaitu menghitung jumlah kendaraan yang lewat pada suatu jalan raya. Jalan raya tersebut ditentukan dari posisi *quadcopter* saat berada di udara. Berdasarkan gambar 3.2 telah diketahui bahwa input yang masuk pada alat ini adalah tangkapan gambar jalan raya dan tekanan udara. Kemudian dilakukan proses pada alat penghitung jumlah kendaraan akan menghasilkan output yaitu jumlah kendaraan yang lewat di jalan raya dan ketinggian dari *quadcopter* saat dilakukannya perhitungan jumlah kendaraan. Diagram blok dari alat penghitung jumlah kendaraan adalah sebagai berikut.

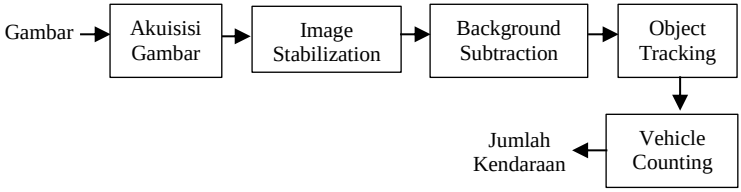


Gambar 3.3 Diagram Blok Alat Penghitung Jumlah Kendaraan

Seluruh proses ini terjadi didalam sebuah minicomputer yaitu Raspberry Pi. Tangkapan gambar didapatkan dari kamera dan tekanan akan didapatkan dengan sensor tekanan yang masing-masing telah terpasang pada minicomputer. Hasil yang didapatkan berupa jumlah kendaraan dan ketinggian *quadcopter*.

3.2.1. Visi Komputer

Visi komputer akan mengolah gambar dari jalan raya yang didapatkan dengan menggunakan kamera. Kemudian dari hasil gambar tersebut akan melewati beberapa tahap image processing supaya didapatkan hasil pengolahan gambar yang menghasilkan jumlah kendaraan yang lewat. Berikut merupakan diagram blok dari visi komputer.



Gambar 3.4 Diagram Blok Visi Komputer

Visi komputer dimulai dengan mengambil tangkapan gambar dari kamera. Kemudian dilanjutkan dengan *image stabilization*. *Image stabilization* digunakan untuk meminimalisir pergerakan kecil pada tangkapan gambar yang diakibatkan oleh *quadcopter*. Hasil *image stabilization* dilanjutkan dengan *background subtraction*. *Background subtraction* digunakan untuk mendeteksi pergerakan yang ada pada

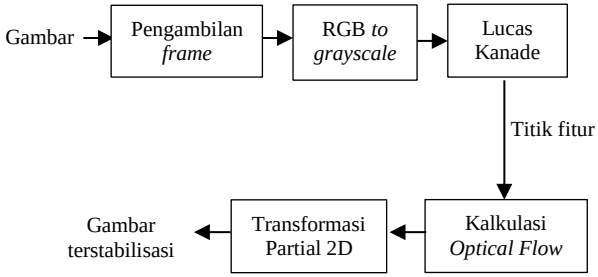
gambar. Deteksi pergerakan yang dianalisa pada gambar adalah pergerakan dari kendaraan yang sedang bergerak di jalan raya. Hasil olah dari *background subtraction* akan dilanjutkan dengan *object tracking*. *Object tracking* yaitu melacak pergerakan obyek yang terjadi pada gambar. Pergerakan obyek yang akan dideteksi adalah pergerakan dari kendaraan setelah melewati proses *background subtraction*. Tahap terakhir yaitu proses perhitungan kendaraan.

3.2.1.1. Akuisisi Gambar

Berdasarkan perancangan dari alat, diketahui bahwa Raspberry Pi akan dikoneksikan dengan sebuah kamera yaitu Raspberry Pi *Camera Module*. Kemudian dari hasil tangkapan gambar dari kamera, ukuran gambar akan diperkecil menjadi ukuran 320x240. Pengecilan dari ukuran gambar ini berguna untuk mempercepat proses dari *image processing.*

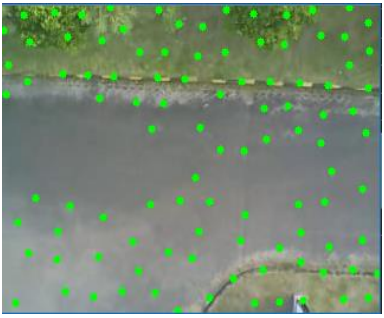
3.2.1.2. Image Stabilization

Image stabilization perlu dilakukan untuk menghilangkan pergerakan kecil yang dialami oleh *quadcopter* saat sedang mengudara. Pergerakan kecil ini dapat mempengaruhi tahap selanjutnya yaitu tahap *background subtraction.* Image stabilization berfungsi untuk melakukan proses stabilisasi gambar yang tertangkap dari kamera. Pergerakan yang terjadi pada kamera akan dihilangkan dengan menggerakan gambar yang telah terstabilisasi ke arah yang sesuai dengan pergerakan tersebut. Stabilisasi gambar akan memanfaatkan *Lucas Kanade Optical Flow.* Berikut merupakan diagram blok dari proses *image stabilization*.



Gambar 3.5 Diagram Blok *Image Stabilization*

Proses pertama kali yang dilakukan adalah pengambilan *frame* yaitu saat t = n dan *frame* saat t=n+1. Kemudian *frame-frame* tersebut dilakukan konversi warna dari RGB menjadi *grayscale*. Hasil *grayscale* pada saat t = n akan diambil titik-titik fitur yang dapat diambil dengan menggunakan Lucas Kanade.



Gambar 3.6 Ilustrasi Pencarian Fitur Lucas Kanade

Hasil dari lucas kanade akan menghasilkan titik koordinat posisi fitur pada *frame*. Namun ada kalanya tidak ditemukan titik fitur pada gambar. Apabila peristiwa tersebut terjadi, maka proses stabilisasi gambar akan dilewati untuk menghindari program *error*. Untuk disaat titik fitur telah ditemukan, maka akan dilakukan perhitungan *optical flow*.

Perhitungan *optical flow* dilakukan dengan cara membandingkan titik fitur pada *frame* saat t = n dengan t = n+1. Hasil dari *optical flow* kemudian akan dilakukan pembuangan pada titik-titik fitur yang tidak ditemukannya nilai *optical flow*. Tahap selanjutnya adalah membuat matriks transformasi dengan menggunakan transformasi matriks partial 2D. Transformasi partial 2D merupakan fungsi yang digunakan untuk mendapatkan nilai rotasi, skala uniform, dan translasi Transformasi partial 2D dapat dilakukan dengan menggunakan syntax *T = estimateRigidTransform(prev_corner2, cur_corner2, false)*. Hasil dari matriks T adalah sebagai berikut.

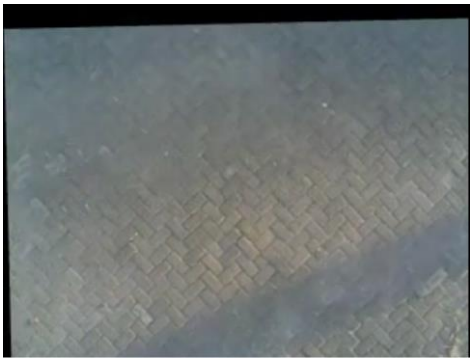
$$T = \begin{bmatrix} \cos\theta & -\sin\theta & t_x \\ \sin\theta & \cos\theta & t_y \end{bmatrix}$$

Dari nilai matriks tersebut dapat diambil nilai dari dx, dy dan da.Nilai dx dapat diambil dari nilai t_x pada matriks, dy didapatkan dari

nilai t_y, dan nilai da dapat diambil dari nilai arctan dari $\cos\theta$ dan $\sin\theta$. Nilai dari dx, dy, dan da yang telah didapat akan disimpan kedalam vektor.

Setelah didapatkan nilai dari dx, dy, dan da maka nilai tersebut dipindahkan sebagai nilai vektor trajektori. Setelah didapatkan nilai trajektori maka tahap selanjutnya adalah melakukan penghalusan pada nilai trajektori. Untuk algoritma yang digunakan untuk melakukan penghalusan adalah dengan menggunakan nilai rata-rata (*average*). Tahap selanjutnya adalah memasukkan nilai dx, dy, dan da hasil penghalusan kedalam data transformasi terbaru.

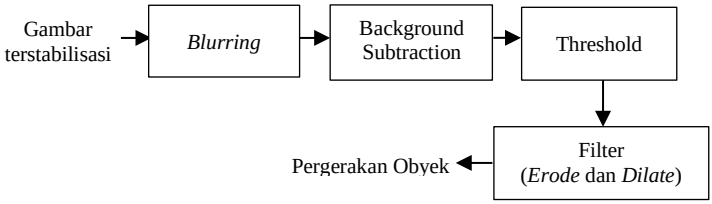
Proses dilanjutkan dengan melanjutkan transformasi partial 2D. Matriks dari T akan diperbarui dengan nilai dx,dy, dan da yang baru. Hasil setelah melakukan proses ini adalah gambar yang telah terstabilisasi. Untuk kondisi dimana saat melakukan pencarian titik fitur dilakukan dan hasilnya tidak mendapatkan titik fitur, maka *proses optical flow* akan dilewati dan hanya akan menjalankan fungsi hasil *output* berupa *frame* saat t = n adalah *frame* saat t=n+1. Hasil dari *image stabilization* adalah gambar frame yang akan menyeseaikan dengan pergerakan dari kamera. Dimisalkan apabila kamera dari *quadcopter* bergerak ke atas, maka hasil yang didapatkan dari hasil stabilisasi adalah gambar akan bergerak ke bawah.



Gambar 3.7 Hasil *Image Stabilization*

3.2.1.3. Background Subtraction

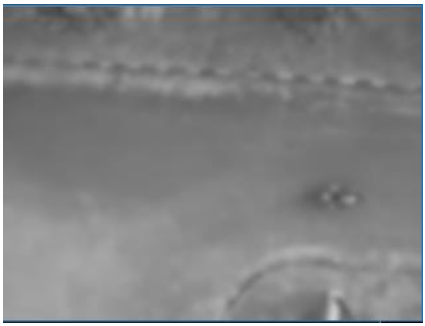
cSetelah dilakukan *image stabilization*, maka tahap selanjutnya adalah *background subtraction*. *Background subtraction* digunakan untuk mencari pergerakan yang terjadi pada *frame*. Untuk tugas akhir ini maka pergerakan yang akan dicari adalah pergerakan dari kendaraan yang sedang melewati jalan raya. Berikut merupakan diagram blok dari *background subtraction*.



Gambar 3.8 Diagram Blok Background Subtraction

Sebelum melakukan *background subtraction*, gambar *output* dari *image stabilization* dilanjutkan dengan proses *blurring*. *Blurring* ini dilakukan untuk meminimalisir *noise* yang akan tercipta setelah dilakukan *background subtraction*. Metode *blur* yang digunakan adalah metode Gaussian *Blur*.

Gambar 3.9 Gambar Input Asli

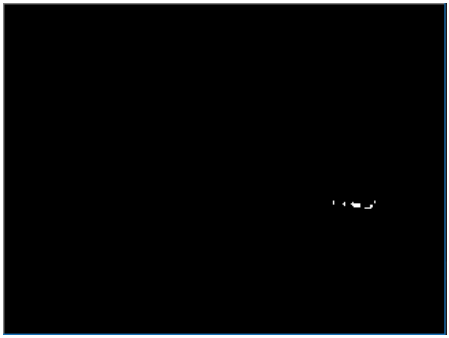


Gambar 3.10 Gambar Hasil Blurring

Hasil *blur* dilanjutkan dengan *background subtraction*. Metode yang akan digunakan dalam melakukan *background subtraction* adalah metode *absdiff()*. Metode *absdiff()* terbilang mudah yaitu dengan cara membandingkan *frame* sebelumnya dengan *frame* selanjutnya. Kedua *frame* tersebut akan dilakukan pengurangan sehingga dapat diketahui posisi piksel yang bernilai beda antara *frame* sebelumnya dan selanjutnya. Hasil dari *background subtraction* akan dilakukan *threshold*. Hasil dari *threshold* akan mengeluarkan piksel yang bernilai 0 atau 1 pada tiap piksel. Tipe *threshold* yang digunakan adalah THRESH_BINARY atau *threshold* binary. *Threshold* binary merupakan tipe *threshold* dimana nilai piksel yang memiliki nilai dibawah nilai *threshold* akan diubah menjadi bernilai minimum yaitu 0, begitupun sebaliknya nilai piksel yang memiliki nilai piksel diatas *threshold* akan menjadi bernilai maksimum yaitu 1.

Gambar 3.11 Hasil *Absdiff()*

Hasil gambar yang dihasilkan dari *background subtraction* pasti akan memiliki *noise*. Sebagai contoh *noise* yang terjadi dapat diakibatkan oleh pergerakan dari pohon di sekitar jalan raya dan pergerakan dari *quadcopter* saat sedang di udara. Filter yang digunakan pada *background subtraction* adalah filter *erode* dan *dilate*. Filter *erode* digunakan untuk menghilangkan *noise-noise* kecil yang muncul setelah melakukan *background subtraction*. Berikut merupakan hasil dari melakukan *erode* pertama.



Gambar 3.12 Hasil *Erode*

Filter selanjutnya adalah filter *dilate*. *Dilate* digunakan untuk memperluas kembali ukuran dari hasil *background subtraction* yang telah melewati filter *erode* dan memperluas kembali area piksel yang bernilai 1 yang akan digunakan dalam *object tracking*. Gambar dibawah ini merupakan hasil dari filter *dilate*.

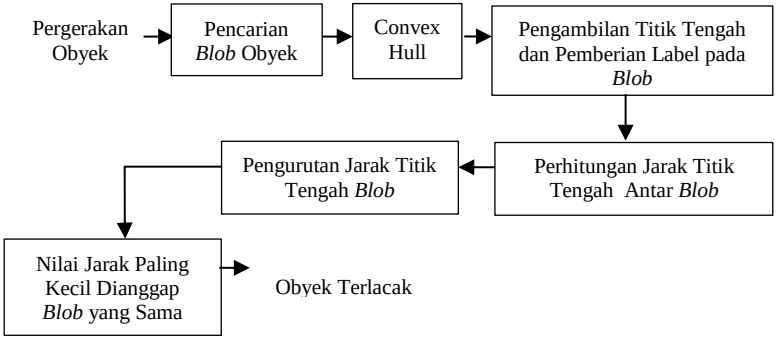
Gambar 3.13 Hasil *Dilate*

Untuk metode filter yang akan digunakan akan melewati beberapa tahap. Metode filter yang akan digunakan adalah sebagai berikut

Erode → *Dilate* → *Erode* → *Dilate*

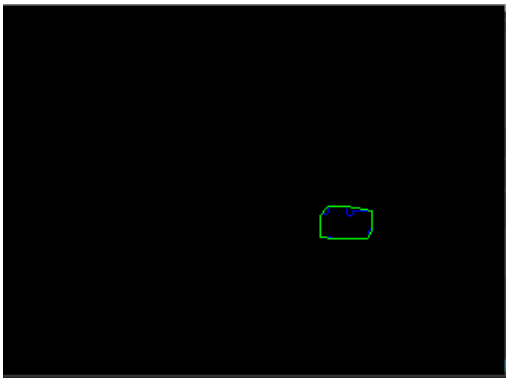
3.2.1.4. Object Tracking

Object tracking dilakukan untuk melakukan pelacakan pada gambar dari *frame* awal menuju *frame* berikutnya. Pelacakan yang dilakukan merupakan pergerakan dari kendaraan yang telah dilakukan pada *background subtraction*. Diagram blok dari *object tracking* akan digambarkan pada gambar dibawah ini



Gambar 3.14 Diagram Blok *Object Tracking*

Hasil dari *background subtraction* merupakan piksel-piksel yang bernilai 0 dan 1. Nilai piksel yang akan dijadikan sebagai nilai *contour* merupakan nilai maksimum dari piksel yaitu 1. *Contour* yang memiliki nilai maksimum beserta berada berdekatan akan disatukan menjadi sebuah area yang dinamakan sebagai *blob*. Setelah dilakukan penemuan *blob* yang terdapat pada *frame* maka dilanjutkan dengan perbaikan *blob* yang telah didapat. Perbaikan ini dilakukan dengan menggunakan metode *convex hull*.



Gambar 3.15 Hasil Pencarian *Blob* (Biru) dan Hasil *Convex Hull* (Hijau)

Setelah dilakukan pencarian *blob*, tahap selanjutnya adalah mengambil nilai dari area dan titik tengah dari *contour* tersebut. Nilai-nilai ini akan disimpan sebagai *blob* saat *frame* t = n. Kemudian dilanjutkan pada *frame* t = n+1 akan dilakukan kembali proses pencarian, perbaikan dengan *convex hull*, dan pengambilan nilai titik tangah dan area. Tahap selanjutnya adalah melakukan perhitungan antara nilai tengah *blob* yang ada pada *frame* t = n dan t = n+1. Proses perhitungan ini akan dibagi menjadi dua proses. Proses yang pertama yaitu menghitung beberapa *blob* yang ditemukan pada *frame* t = n dengan salah satu *blob* yang ditemukan pada *frame* t = n + 1. Perhitungan jarak antar *blob* dilakukan dengan menggunakan rumus *Euclidean distance* dibawah ini.

$$\Delta x = blob_{t=n+1}.x - blob_{t=n}.x$$

$$\Delta y = blob_{t=n+1}.y - blob_{t=n}.y$$

$$distance = \sqrt{(\Delta x)^2 + (\Delta y)^2}$$

Tahap selanjutnya adalah melakukan pengurutan nilai *distance* dari masing-masing *blob* yang telah ditemukan pada *frame*. Nilai *distance* yang paling kecil akan dianggap sebagai *blob* yang sama dan akan diberikan label yang sama. Hasil output yang diberikan adalah obyek

yang telah terlacak. Hasil gambar yang telah dilacak adalah sebagai berikut.

Gambar 3.16 Kendaraan Terlacak

Berdasarkan gambar 3.26 dapat dilihat kendaraan pada tangkapan gambar telah terlacak. Tanda telah terlacak adalah terbuatnya kotak biru yang mengelilingi gambar kendaraan dan tulisan *tracked (0)* di tengah kotak biru. Besaran kotak biru tersebut merupakan *blob* hasil dari *background subtraction*. Nilai 0 merupakan tanda label pada kendaraan tersebut

3.2.1.5. Vehicle Counting

Proses dari penghitungan kendaraan menjadi tahap terakhir dari algoritma ini. Proses perhitungan dilakukan dengan memasang sebuah garis yang melintang secara horizontal pada *frame* output. Garis ini akan menjadi acuan kendaraan tersebut telah melewati garis tersebut atau tidak. Apabila telah melewati garis, maka jumlah kendaraan akan bertambah. Dikarenakan bergantung dengan posisi perspektif dari *quadcopter* saat sedang mengudara, maka metode penghitungan akan dibedakan menjadi dua cara, kendaraan mengarah dari kiri *quadcopter* menuju kanan *quadcopter* atau sebaliknya. Penghitungan ini juga akan memanfaatkan hasil dari *object tracking*.

Hasil *object tracking* akan memiliki nilai posisi dari *blob* yang telah disimpan. Untuk proses penghitungan dilakukan dengan membandingkan posisi dari *frame* t = n dengan *frame* t = n+1. Jika *blob* saat di *frame* t = n berada di posisi sebelum mengenai garis dan *blob* saat di *frame* t = n+1 berada di posisi setelah mengenai garis, maka akan

menambah total dari jumlah kendaraan. Posisi *blob* dapat diambil dari posisi titik tengah dari *blob* tersebut. Syarat dari terjadinya perhitungan ini adalah label antara sebelum dan setelah mengenai garis harus sama. Berikut merupakan hasil perhitungan jumlah kendaraan yang lewat.

Gambar 3.17 Kendaraan saat t = n

Gambar 3.18 Kendaraan saat t = n+1

Gambar 3.19 Hasil Perhitungan Kendaraan

Pada gambar 3.19 terdapat garis horizontal merah yang melintang pada gambar. Garis ini akan dijadikan sebagai acuan dalam melakukan proses perhitungan. Posisi kendaraan berada di sebelah kanan gambar menandakan belum melewati garis. Pada gambar 3.28 kendaraan telah melewati garis sehingga kendaraan tersebut terhitung dan ditandakan dengan pertambahan nilai pada kotak hitam pada pinggir kanan atas gambar dan warna garis menjadi hijau.

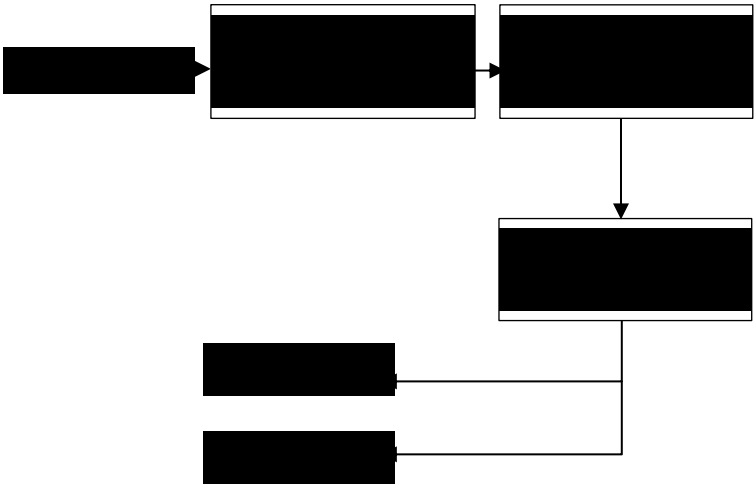
Pada akhir program akan ditambahkan penyimpanan rekaman video dari hasil perhitungan dan rekaman video asli tangkapan kamera. Selain itu akan juga ditambahkan tombol keyboard yang berfungsi untuk melakukan reset perhitungan untuk kembali menjadi 0.

3.2.2. Sensor Tekanan

Selain visi komputer, minicomputer juga terpasang sebuah sensor tekanan. Sensor tekanan tersebut yaitu BMP180. Pembacaan tekanan dari BMP180 akan dilakukan konversi menjadi nilai ketinggian. Nilai ketinggian tersebut digunakan untuk mendapatkan ketinggian dari *quadcopter* saat melakukan proses perhitungan. Hasil output dari data ketinggian BMP180 masih berupa ketinggian geopotensial. Tinggi geopotensial adalah tinggi yang diukur berdasarkan konstanta tekanan laut (p_0 = 1013,25 hPa) dengan satuan h (1 h = 0,98 m). Dikarenakan data ketinggian dari BMP180 masih memiliki *noise* yang banyak, maka akan dilakukan filter pada hasil pembacaan. Filter ini digunakan untuk memperbagus hasil pembacaan ketinggian BMP180. Filter yang akan digunakan adalah filter eksponensial.

Untuk menghitung ketinggian dari *quadcopter* saat sedang berada di udara, maka diperlukan perhitungan dengan memanfaatkan nilai

ketinggian geopotensial yang didapatkan dari BMP180. Perhitungan yang akan dilakukan adalah pengurangan nilai ketinggian geopotensial dari *quadcopter* saat berada di udara dengan nilai ketinggian geopotensial dari *quadcopter* saat berada di darat. Gambar dibawah ini merupakan blok diagram dari perhitungan ketinggian dari *quadcopter*.



Gambar 3.20 Diagram Blok Ketinggian *Quadcopter*

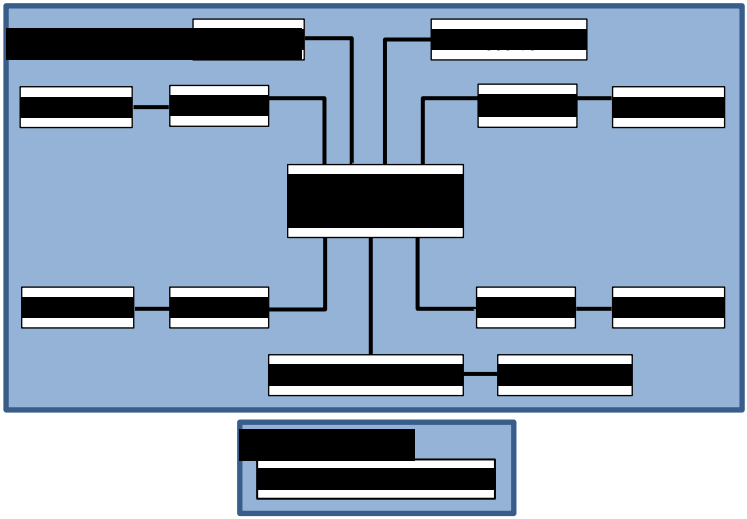
Proses pertama yang dilakukan adalah penyimpanan nilai ketinggian geopotensial dari *quadcopter* saat berada di darat sebagai ketinggian inisiasi (*altitude_init*). Kemudian saat *quadcopter* mulai terbang akan dilakukan penyimpanan nilai ketinggian geopotensial kembali saat di udara sebagai ketinggian terbang (*altitude*). Dari dua nilai ketinggian tersebut akan dilakukan pengurangan antara ketinggian inisiasi dengan ketinggian terbang. Hasil pengurangan tersebut akan didapatkan nilai ketinggian *quadcopter* terbang. Namun hasil ini merupakan hasil tanpa dilakukan filter. Untuk mendapatkan nilai ketinggian dengan menggunakan filter, proses yang dilakukan akan sedikit berbeda. Pada penggunaan filter akan digunakan tiga nilai sebagai ketinggian filter sekarang (*altitude_filter_current*), ketinggian filter sebelumnya (*altitude_filter_previous*) dan konstanta alpha (α). Untuk mendapatkan ketinggian dengan filter maka digunakan rumus sebagai berikut.

$$altitude\_filter\_current = altitude * \alpha + (1 - \alpha) * altitude\_filter\_previous$$

Setelah didapatkan nilai ketinggian filter sekarang (*altitude_filter_current*), maka dimulai perhitungan ketinggian sebenarnya dari *quadcopter* di udara yaitu dengan melakukan pengurangan antara nilai ketinggian filter sekarang (*altitude_filter_current*) dengan ketinggian inisiasi (*altitude_init*). Kedua hasil berupa ketinggian *quadcopter* tanpa filter dan ketinggian *quadcopter* dengan filter akan disimpan pada sebuah 2 file yang berbeda.

3.3. Perancangan *Hardware Quadcopter*

Perancangan *hardware* merupakan perancangan penggunaan dan pemasangan komponen-komponen yang akan dilakukan dalam pembuatan *quadcopter*. *Quadcopter* memiliki fungsi untuk dapat membawa alat penghitung jumlah kendaraan ke udara. Komponen-komponen yang akan digunakan pada *hardware quadcopter* adalah *frame*, *flight controller*, motor, ESC, *transmitter*, *receiver*, GPS, propeller, baterai, dan gimbal. Berikut merupakan sistem *wiring* pada *quadcopter* .



Gambar 3.21 Sistem *Wiring Quadcopter*

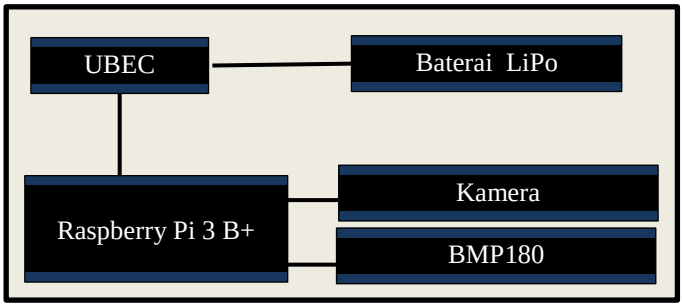
Tabel 3.2 Sistem *Wiring Quadcopter*

Komponen	Bagian Komponen	Port
Receiver	SBUS	SBUS *Flight controller*
	VCC	+5 V *Flight controller*
	GND	GND *Flight controller*
ESC	VCC	+Baterai LiPo
	GND	-Baterai LiPo
	Signal	S1 $^{s}/_{d}$ S4 *Flight controller*
Baterai LiPo	VCC	+VBAT *Flight controller*
	GND	-VBAT *Flight controller*
GPS	VCC	+Baterai LiPo
	GND	-Baterai LiPo
	RX	TX3 *Flight controller*
	TX	RX3 *Flight controller*
	SDA	SDA *Flight controller*
	SCL	SCL *Flight controller*
Gimbal	VCC	+Baterai LiPo
	GND	-Baterai LiPo

Seluruh komponen ini akan diletakkan pada bagian frame dari *quadcopter*. *Flight controller* digunakan sebagai otak dari *quadcopter* yang akan mengendalikan komponen-komponen lainnya yang terhubung dengan *flight controller*. *Flight controller* akan dihubungkan ke 4 ESC yang masing-masing dari ESC akan mengendalikan kecepatan putaran dari motor. *Flight controller* akan juga dihubungkan dengan receiver dimana akan menerima sinyal dari *transmitter* yang digunakan oleh operator untuk menerima perintah dari operator terhadap *quadcopter*. Selain itu terdapat GPS yang digunakan untuk mendapatkan posisi koordinat dari *quadcopter* saat terbang yang nantinya akan berfungsi untuk menjaga posisi *quadcopter* saat di udara. Dikarenakan posisi kamera dari alat penghitung kendaraan diperlukan untuk tetap menghadap ke bawah, maka akan digunakan gimbal. Gimbal akan mempertahankan posisi kamera kebawah disaat *quadcopter* akan melakukan manuver *pitch* dan *roll*. Baterai LiPo digunakan sebagai tegangan suplai dari komponen-komponen.

3.4. Perancangan *Hardware* Alat Penghitung Jumlah Kendaraan

Perancangan *hardware* alat penghitung kendaraan adalah perancangan yang akan membahas komponen-komponen dan alat pendukung lainnya penyusun pembuatan alat penghitung jumlah kendaraan. Berikut merupakan diagram dari sistem *wiring* alat penghitung jumlah kendaraan.



Gambar 3.22 Sistem *Wiring* Alat Penghitung Jumlah Kendaraan

Tabel 3.3 Sistem *Wiring* Penghitung Jumlah Kendaraan

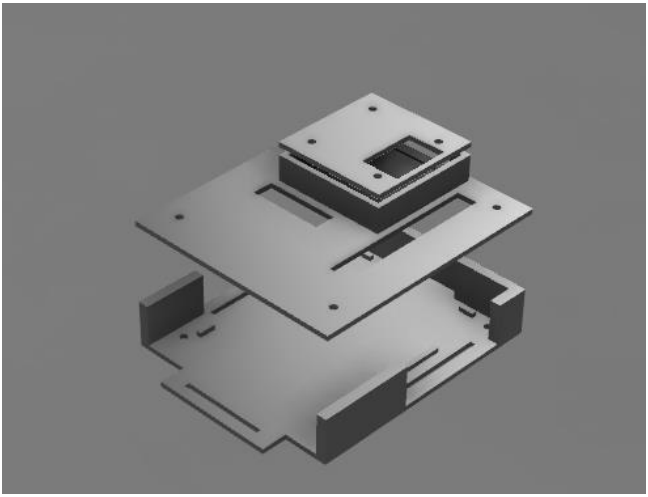
Komponen	Bagian Komponen	Port
UBEC	VCC	+Baterai LiPo
	GND	-Baterai LiPo
Raspberry Pi	VCC (Pin 4)	+UBEC 5V
	GND (Pin 6)	-UBEC 5V
Kamera	*Camera Port*	Raspberry Pi *Camera Module*
BMP180	VCC	+3,3 V Raspberry Pi (Pin 1)
	GND	GND Raspberry Pi (Pin 9)
	SDA	SDA Raspberry Pi (Pin 3)
	SCL	SCL Raspberry Pi (Pin 5)

Setelah dilakukan perancangan terhadap komponen-komponen yang akan digunakan dalam pembuatan alat penghitung jumlah kendaraan, maka diperlukan pembuatan eksterior sebagai pelindung dari komponen-

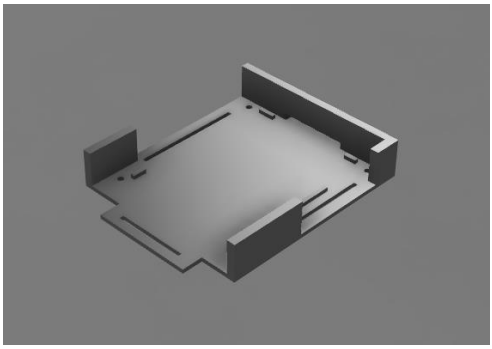
komponen tersebut. Dalam perancangan desain eksterior, desain dilakukan dengan perhitungan supaya tidak menggangu kinerja dari sistem kerja *quadcopter*.

Desain eksterior yang akan dibuat akan digunakan akan dibagi menjadi dua bagian yaitu eksterior Raspberry Pi dan eksterior kamera. Perancangan eksterior dibuat dengan cetak 3D.

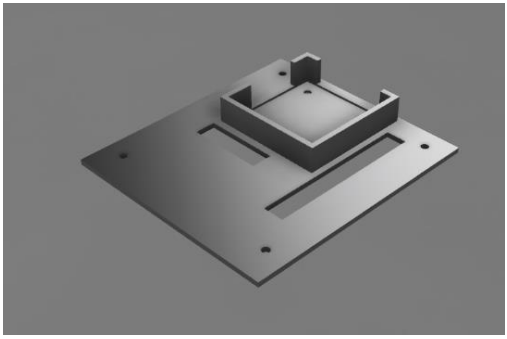
Untuk eksterior Raspberry Pi akan digabungkan dengan *flight controller*. Desain ini akan terbagi menjadi 3 bagian, yaitu bagian dasar untuk Raspberry Pi, bagian tutup untuk Raspberry Pi, dan bagian tutup untuk *flight controller*. Hasil desain dari eksterior Raspberry Pi akan ditunjukan pada gambar dibawah ini.



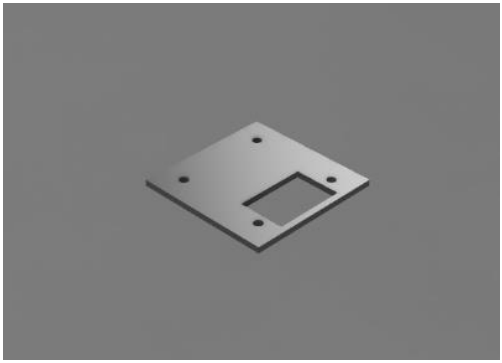
Gambar 3.23 Keseluruhan Eksterior Raspberry Pi



Gambar 3.24 Bagian Dasar Raspberry Pi



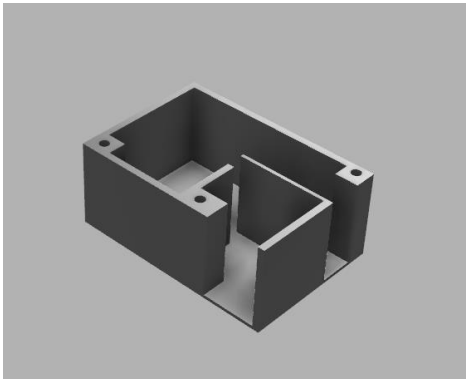
Gambar 3.25 Bagian Tutup Raspberry Pi



Gambar 3.26 Bagian Tutup *Flight controller*

Selain perancangan eksterior untuk Raspberry Pi, pembuatan eksterior lainnya adalah perancangan eksterior untuk kamera yaitu Raspberry Pi *Camera Module*. Eksterior ini akan diletakkan pada bagian gimbal *quadcopter*. Berikut merupakan hasil desain perancangan eksterior gambar.

Gambar 3.27 Keseluruhan Eksterior Kamera



Gambar 3.28 Bagian Dasar Eksterior Kamera

Gambar 3.29 Bagian Tutup Eksterior Kamera

[Halaman ini sengaja dikosongkan]

BAB 4
HASIL DAN ANALISA DATA

4.1. Realisasi Alat

Gambar 4.1 Realisasi Alat Tampak Samping

Gambar 4.2 Realisasi Alat Tampak Atas

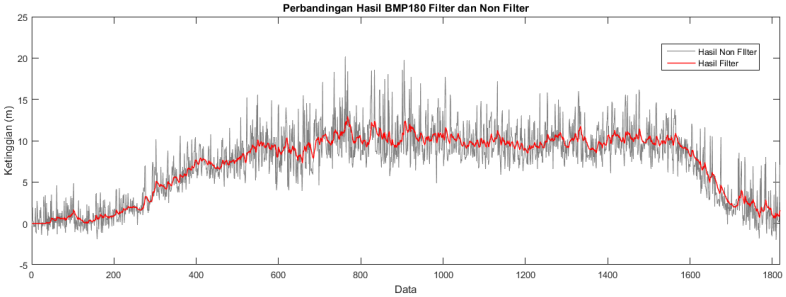
Tabel 4.1 Spesifikasi Alat

Spesifikasi Alat	
Tinggi	26,5 cm
Panjang	32 cm
Lebar	33 cm
Berat	1278 gram
Waktu Lama Terbang	± 2 menit

4.2. Pengujian BMP180

Pengujian BMP180 memiliki tujuan yaitu melakukan perbandingan nilai ketinggian sebelum dan sesudah penggunaan filter eksponensial serta menguji tingkat akurasi nilai ketinggian dari BMP180.

Pengujian pertama yaitu pengujian hasil penggunaan filter eksponensial pada BMP180. Gambar dibawah ini merupakan hasil pengujian dari BMP180 dan melakukan perbandingan antara hasil sebelum dan sesudah penggunaan filter.

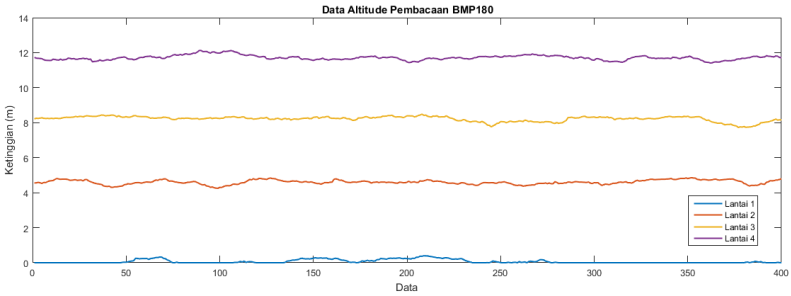


Gambar 4.3 Hasil Pengujian Filter BMP180

Garis abu-abu merupakan garis hasil pembacaan ketinggian dari BMP180 tanpa filter sedangkan garis merah merupakan hasil pembacaan ketinggian dari BMP180 dengan filter. Dari gambar 4.5 dapat dilihat bahwa hasil pembacaan dengan menggunakan filter terlihat lebih *smooth* dibandingkan tanpa menggunakan filter.

Pengujian kedua adalah pengujian tingkat akurasi dari nilai pembacaan ketinggian BMP180. Peralatan yang digunakan pada percobaan ini adalah meteran. Pengujian tingkat akurasi dilakukan dengan perbandingan nilai pengukuran asli dengan meteran dan nilai

pembacaan ketinggian dari BMP180. Pengujian dilakukan di Gedung B Departemen Teknik Elektro, Fakultas Teknologi Elektro, Institut Teknologi Sepuluh Nopember, Surabaya. Meteran digunakan untuk mengukur ketinggian dari tiap lantai dari gedung. Kemudian dilakukan perbandingan nilai dengan pembacaan BMP180. Untuk hasil pembacaan ketinggian dari BMP180 akan diambil 400 sampel data ketinggian pada tiap lantai. Hasil 400 sampel tersebut kemudian akan dilakukan perhitungan rata-rata nilai ketinggian yang didapatkan. Gambar dibawah ini merupakan grafik hasil pembacaan 400 sampel data dari BMP180 pada tiap lantainya.



Gambar 4.4 Hasil Pembacaan Ketinggian BMP180 Tiap Lantai Gedung

Setelah dilakukan pembacaan ketinggian dengan BMP180, 400 data sampel yang telah didapatkan pada tiap lantai akan dilakukan perhitungan rata-rata nilai ketinggian pada tiap lantai. Setelah didapatkan nilai rata-rata tersebut akan dilanjutkan dengan perbandingan nilai hasil pengukuran dengan menggunakan meteran. Tabel berikut merupakan hasil perbandingan nilai yang didapatkan dari BMP180 dan pengukuran dengan meteran.

Tabel 4.2 Hasil Perbandingan Ketinggian

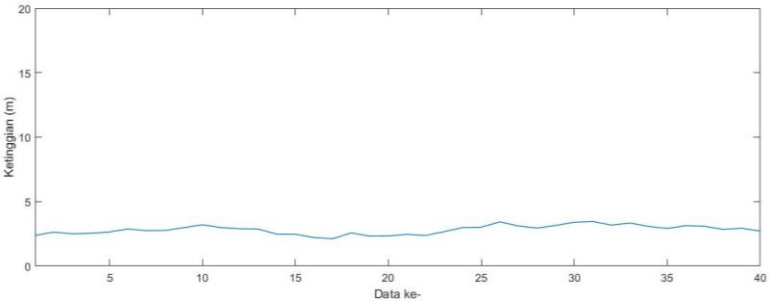
Lantai	Pengukuran Meteran (m)	Rata- Rata Pembacaan BMP180 (m)	*Error* (m)
Lantai 1	0	0,0709	+0,0709
Lantai 2	4	4,6041	+0,6041
Lantai 3	8	8,2185	+0,2185
Lantai 4	12	11,7119	-0,2881
Rata- Rata			+0,1514

Dari tabel hasil perbandingan diatas, pembacaan ketinggian pada lantai 1 memiliki *error* +0,0709 m. Lantai 2 memiliki *error* sebesar +0,6041 m. Lantai 3 memiliki *error* sebesar +0,2185 m. Lantai 4 memiliki *error* sebesar -0,2881. Dari keempat dari hasil *error* pada tiap lantai, nilai *error* ketinggian pada BMP180 adalah +0,1514 m.

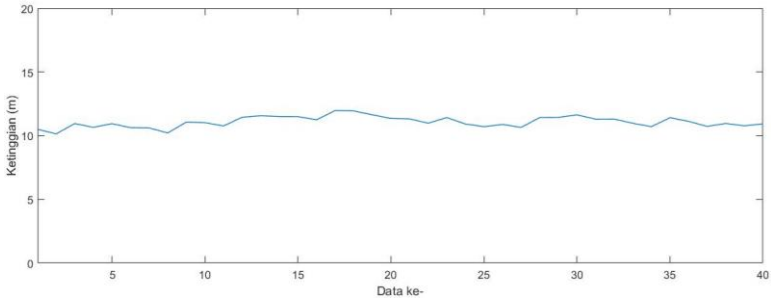
4.3. Pengujian Tingkat Kestabilan *Quadcopter*

Tujuan dari pengujian tingkat kestabilan *quadcopter* adalah pengujian tingkat kestabilan *quadcopter* dalam mempertahankan ketinggian serta posisi dari *quadcopter* saat di udara. Cara pengujian dimulai dengan *quadcopter* dalam posisi terbang manual dengan menggunakan *transmitter* kemudian saat sedang di udara *quadcopter* akan diubah menjadi mode *altitude hold* dan *position hold*. Saat berada di udara akan dilakukan pengambilan nilai altitude dari *quadcopter* yang berasal dari pembacaan BMP180. Berikut merupakan hasil dari pengujian tingkat kestabilan *quadcopter*.

Gambar 4.5 Pengujian Tingkat Kestabilan *Quadcopter*



Gambar 4.6 Ketinggian Pengujian 1 Tingkat Kestabilan *Quadcopter*



Gambar 4.7 Ketinggian Pengujian 2 Tingkat Kestabilan *Quadcopter*

Pada gambar 4.5 *quadcopter* sedang berada dalam posisi *altitude hold* dan *position hold*. Dari gambar tersebut dapat dilihat bahwa operator tidak mengendalikan *quadcopter* secara manual. Data grafik ketinggian *quadcopter* ditunjukkan pada gambar 4.6 dan 4.7. Pengujian dilakukan pada dua ketinggian yang berbeda. Dari hasil kedua ketinggian tersebut dapat dilihat bahwa *quadcopter* dapat mempertahankan ketinggiannya.

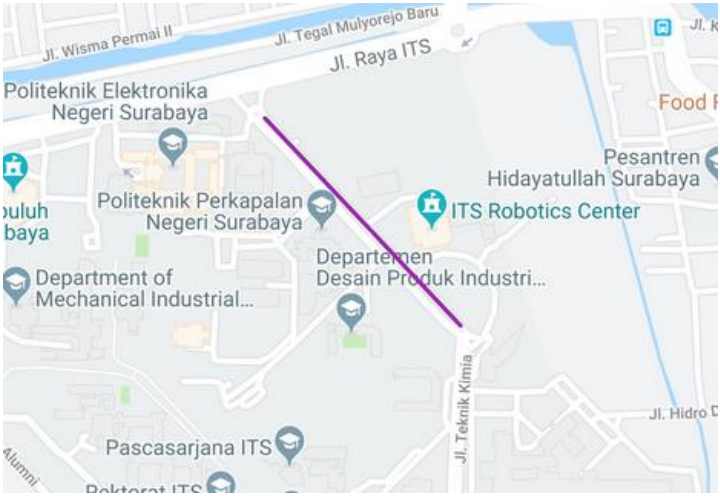
Kemudian kestabilan *quadcopter* akan diuji dengan faktor eksternal yaitu keadaan angin. Angin dapat mengganggu proses menjaga posisi dan ketinggian *quadcopter* apabila angin bertiup sangat kencang hingga menyebabka *quadcopter* dapat terbang tidak stabil. Proses pengujian ini membutuhkan sebuah anemometer untuk melakukan pengukuran dari kecepatan angin

Gambar 4.8 Pengukuran Kecepatan Angin

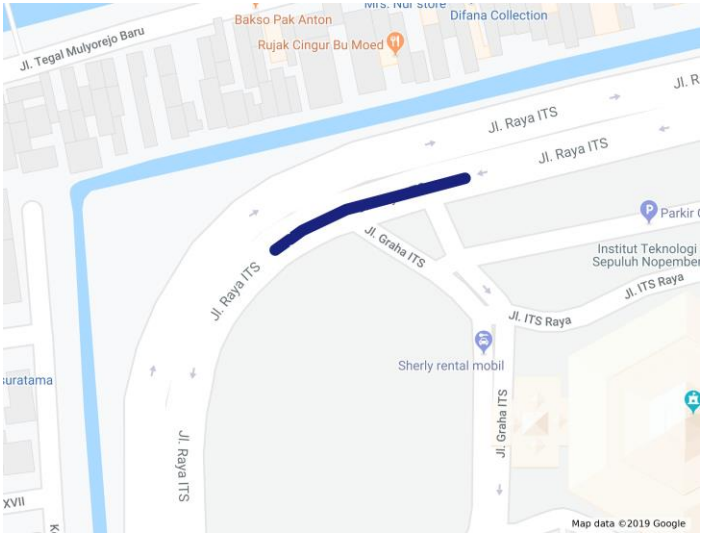
Berdasarkan hasil pengujian kestabilan dengan melakukan pengukuran kecepatan angin, *quadcopter* akan mengalami drift pada kecepatan angin 1,7 $^{m}/_{s}$. Namun *quadcopter* masih dapat melakukan koreksi untuk mempertahankan posisi. Sedangkan untuk kecepatan angin yang mencapai 2 $^{m}/_{s}$, *quadcopter* akan mengalami ketidakstabilan sehingga perlu diambil alih operator supaya kembali stabil.

4.4. Pengujian Perhitungan Jumlah Kendaraan

Pengujian program merupakan pengujian terhadap program yang telah dirancang pada bab sebelumnya. Tujuan dari pengujian ini adalah untuk mengetahui hasil perhitungan jumlah kendaraan yang ada di jalan raya. Pengujian program dilakukan secara *online*. *Online* merupakan pengujian program penghitungan kendaraan secara langsung saat *quadcopter* berada di udara. Lokasi pengujian dilakukan pada dua tempat yang berbeda yaitu di sepanjang Jalan Teknik Kimia depan PPNS dan Jalan Raya ITS depan Graha ITS. Kedua lokasi ini memiliki lebar jalan yang berbeda dimana lebar jalan pada Jalan Raya ITS lebih besar dari Jalan Teknik Kimia yaitu sekitar 10 m dan 5 m. Lokasi ditunjukkan dengan garis berwarna ungu dan biru pada gambar 4.9 dan 4.10.



Gambar 4.9 Lokasi Pengujian Jalan Teknik Kimia



Gambar 4.10 Lokasi Pengujian Jalan Raya ITS

Sebelum dilakukan pengujian diperlukan penentuan nilai dari ukuran kernel pada filter yang digunakan pada bagian *background subtraction*. Ukuran kernel yang digunakan adalah sebagai berikut.

Erode (kernel 3x3) -> Dilate (kernel 25x25) -> Erode (kernel 29x29) -> Dilate (kernel 33x33)

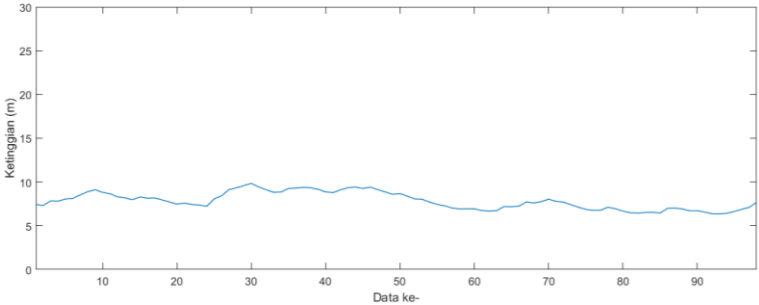
Untuk nilai *erode* pertama ditentukan nilai kernel berukuran 3x3. Alasan dipilihnya ukuran tersebut dikarenakan untuk menghilangkan *noise* kecil yang disebabkan saat terbang dan tidak terlalu mengurangi ukuran *blob* yang mewakili pergerakan kendaraan bermotor. Kemudian hasil tersebut dilanjutkan dengan *dilate* yang berukuran besar untuk memperbesar kembali ukuran *blob* pergerakan kendaraan yang telah berkurang akibat dari filter *erode* dan meminimalisir kemungkinan dua *blob* yang berbeda pada satu kendaraan.

Selain penentuan nilai ukuran dari kernel filter, arah perhitungan kendaraan perlu ditentukan juga. Untuk pengujian ini arah perhitungan untuk kendaraan yang bergerak dari kanan tangkapan kamera *quadcopter* menuju ke kiri tangkapan kamera *quadcopter*.

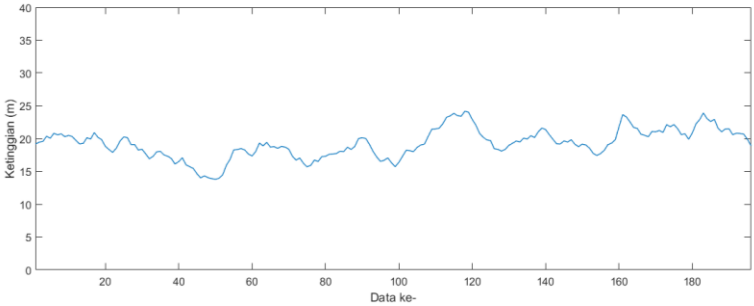
Hasil dari pengujian yang akan diambil adalah jumlah kendaraan yanh terhitung pada rentang waktu tertentu dan ketinggian *quadcopter* saat sedang melakukan proses perhitungan. Untuk nilai tingkat akurasi perhitungan ditentukan dari jumlah kendaraan yang terhitung dengan program dibagi dengan total kendaraan yang lewat dari *quadcopter*. Berikur merupakan hasil perhitungan pada dua lokasi tersebut.

Tabel 4.3 Hasil Pengujian Perhitungan Jumlah Kendaraan *Online*

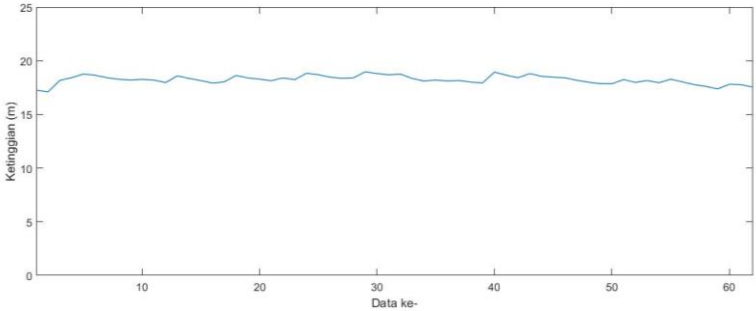
Lokasi	Pengujian	Kendaraan yang lewat	Perhitungan kendaraan	Akurasi	*Output*
Jalan Teknik Kimia	1	7	6	85,71%	10 kendaraan / 10 detik
	2	22	11	50%	5 kendaraan / 10 detik
Jalan Raya ITS	3	24	14	58,3%	17 kendaraan / 10 detik
	4	16	8	50%	16 kendaraan / 10 detik



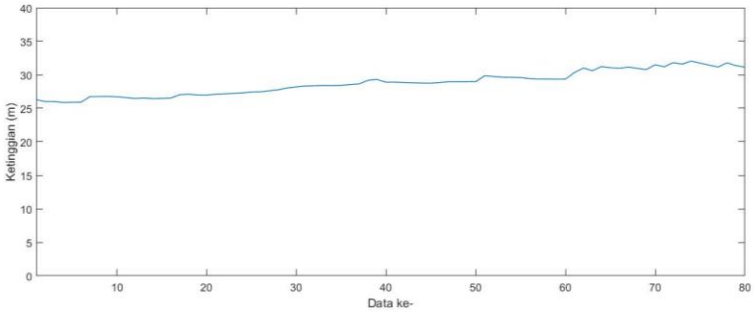
Gambar 4.11 Ketinggian Pengujian 1



Gambar 4.12 Ketinggian Pengujian 2



Gambar 4.13 Ketinggian Pengujian 3



Gambar 4.14 Ketinggian Pengujian 4

Pengujian dilakukan 4 kali dimana pengujian 1 dan 2 dilakukan pada Jalan Teknik Kimia serta pengujian 3 dan 4 pada Jalan Raya ITS. Berdasarkan hasil yang ditunjukkan pada tabel 4.3 dapat dibuktikan bahwa perhitungan dapat dilakukan pada kondisi lebar jalan serta lokasi yang berbeda. Pernyataan ini mendukung bahwa *quadcopter* dapat bersifat fleksibel untuk dapat dilakukan perhitungan jumlah kendaraan pada lokasi yang berbeda dan lebar jalan yang berbeda. Output pada tabel 4.3 menunjukkan data yang akan digunakan pada manajemen lalu lintas. Nilai output didapatkan melalui pembagian jumlah kendaraan yang terhitung pada alat penghitung jumlah kendaran dengan waktu dilakukannya perhitungan.

Pada pengujian 1 didapatkan nilai ketinggian dan data perhitungan kendaraan bermotor yang ditunjukkan pada gambar 4.11 dan tabel 4.3. Dalam proses perhitungan kendaraan, *quadcopter* berada pada ketinggian sekitar 8 m seperti yang ditunjukkan pada gambar 4.11. Kemudian dari hasil perhitungan jumlah kendaraan, jumlah yang terhitung sebanyak 6 dari 7 kendaraan yang lewat dari *quadcopter*. Dari hasil ini didapatkan *error* -1 kendaraan dan tingkat akurasi sebesar 85,71%. Kemudian output yang didapatkan dalam proses perhitungan adalah 10 kendaraan / 10 detik. Pengujian 2 memiliki ketinggian *quadcopter* yang berbeda dengan pengujian 1. Dari gambar 4.12 didapatkan nilai ketinggian sekitar 20 m namun memiliki perubahan tingkat ketinggian yang lebih banyak jika dibandingkan dengan pengujian 1. Ini disebabkan kondisi saat dilakukan perhitungan, kondisi lingkungan angin bertiup kencang dan mengganggu kestabilan *quadcopter*. Untuk hasil pengujian perhitungan jumlah kendaraan didapatkan perhitungan

kendaraan sebanyak 11 dari 22 kendaraan yang lewat *quadcopter*. Hasil ini menimbulkan *error* sebesar -11 kendaraan dan tingkat akurasi sebesar 50 %. Dari hasil pengujian 2 akan didapatkan output yaitu 5 kendaraan / 10 detik.

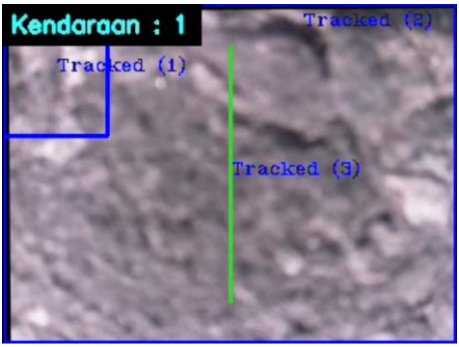
Kemudian pada pengujian 3 memiliki ketinggian *quadcopter* yaitu sekitar 18 m berdasarkan gambar 4.13. Dari hasil perhitungan didapatkan 14 dari 24 kendaraan dapat terhitung. Berdasarkan jumlah yang terhitung didapatkan *error* perhitungan sebanyak -10 kendaraan. Oleh karena itu, tingkat akurasi yang didapatkan pada pengujian 3 adalah 58,3%. Output yang dihasilkan pada pengujian 3 adalah 17 kendaraan / 10 detik. Kemudian dilanjutkna dengan pengujian 4. Pada pengujian 4 didapatkan nilai ketinggian *quadcopter* di sekitar 28 m berdasarkan gamabr 4.14. Dari hasil perhitungan didapatkan 8 dari 16 kendaraan dapat terhitung. Berdasarkan jumlah yang terhitung didapatkan *error* perhitungan sebanyak -8 kendaraan . Hasil tersebut akan mendaaptkan tingkat akurasi mencapai 50%. Output yang dihasilkan adalah 16 kendaraan / 10 detik.

Rata-rata akurasi yang dihasilkan dari keempat pengujian tersebut adalah 61%. Dari nilai output pada tabel 4.3 diketahui bahwa volume kendaraan yang melintas pada Jalan Raya ITS lebih ramai jika dibandingkan dengan Jalan Teknik Kimia. Setelah dilakukan pengujian yang dilakukan secara *online*, data-data yang telah didapatkan pada pengujian akan dilakukan analisa. Analisa dilakukan dengan melihat hasil perekaman video perhitungan kendaraan dan data pembacaan ketinggian. Dari hasil analisa pada kedua pengujian tersebut, didapatkan beberapa poin :

1. Perhitungan kendaraan diakibatkan proses *take off* dan *landing* dari *quadcopter*

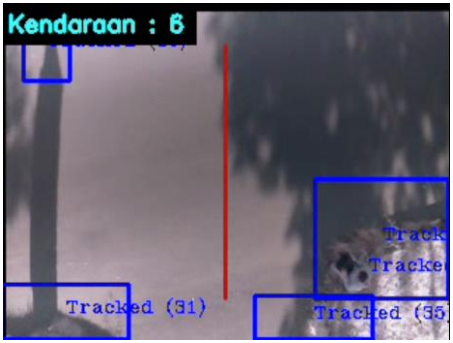
   Pada saat dilakukan pengujian terdapat perhitungan yang tidak diinginkan diakibatkan oleh *noise* yang terjadi. Program dijalankan saat berada di darat. Saat dilakukan *take off* muncul *noise-noise* yang diakibatkan pergerakan dari *quadcopter* untuk menuju titik yang pas untuk melakukan perhitungan. *Noise-noise* yang muncul secara random mengakibatkan terjadi perhitungan yang tidak diinginkan. Begitupun juga terjadi saat *quadcopter* akan landing. Untuk menghindari peristiwa ini, operator akan melakukan reset perhitungan dan akan memulai perhitungan saat *quadcopter* berada pada posisi yang tepat saat melakukan perhitungan. Gambar 4.15

menunjukkan bukti terjadinya salah perhitungan diakibatkan saat *quadcopter* akan *take off*.



Gambar 4.15 Perhitungan saat *Take off*

2. *Noise-noise blob* muncul pada gambar



Gambar 4.16 *Noise* pada Gambar

Saat dilakukan pengujian muncul *noise-noise* yang ditunjukkan dari kotak biru dan tulisan *tracked* pada gambar 4.16. Ini terjadi dikarenakan filter pada *background subtraction* tepatnya pada filter *erode*. Nilai ukuran kernel filter *erode* kurang besar untuk menghilangkan *noise* yang terjadi diluar pergerakan kendaraan. Apabila nilai *erode* terlalu kecil, maka *noise* kecil tersebut akan dilanjutkan menuju filter *dilate*. Filter *dilate* tersebut akan memperbesar kembali *blob noise-noise* kecil tersebut. *Noise-noise* ini

akan meningkatkan potensi terjadinya perhitungan tanpa adanya pergerakan kendaraan pada gambar dibawah ini.

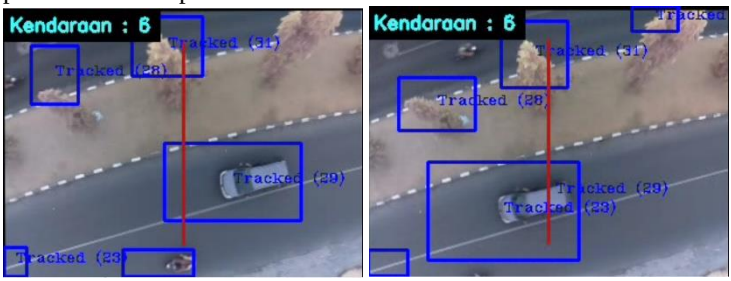
3. Kendaraan tidak terhitung

Kendaraan tidak dapat terhitung dikarenakan dua alasan. Alasan pertama yaitu *blob* kendaraan tidak mengenai garis tengah. Pada metode perhitungan kendaraan ditentukan akan dilakukan perhitungan jika titik tengah dari *blob* kendaraan telah melewati garis. Namun terdapat keadaan dimana titik tengah tersebut tidak melewati garis sehingga menyebabkan kendaraan yang lewat tidak terhitung. Ini merupakan pengaruh dari posisi *quadcopter* yang belum pada posisi yang tepat dalam melakukan perhitungan. Gambar dibawah ini merupakan salah satu contoh dari kendaraan yang tidak terhitung diakibatkan titik tengah tidak mengenai garis.

Gambar 4.17 Kendaraan Tidak Mengenai Garis

Kemudian alasan kedua yaitu *blob* kendaraan tergabung dengan *blob* dari *noise*. Berdasarkan analisa ketiga bahwa masih terdapat *blob-blob* yang terbentuk dikarenakan dari *noise* yang timbul dari pergerakan *quadcopter*. *Blob* diketahui telah terdeteksi pada *frame* t=n. Namun saat ingin melewati garis tersebut, *blob* kendaraan bergabung dengan *blob noise* atau *blob* kendaraan lain sehingga label yang akan dipakai akan berganti. Pergantian label ini menyebabkan

tidak terhitungnya kendaraan. Gambar dibawah ini menunjukkan perubahan label pada kendaraan

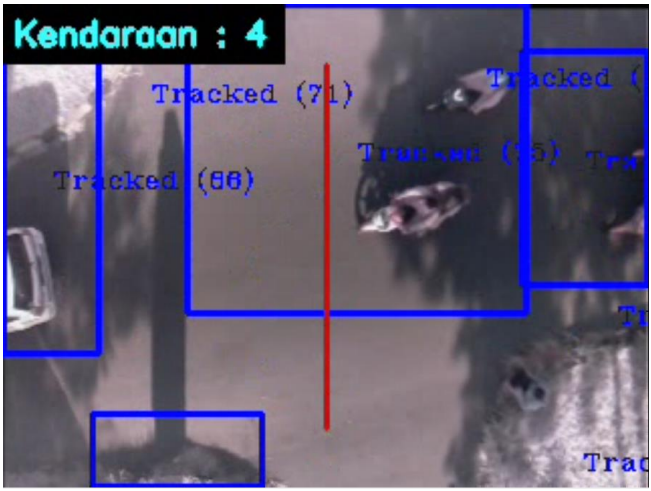


Gambar 4.18 Pergantian Label Kendaraan

Selain itu terdapat hipotesa lain yaitu kecepatan kendaraan yang terlalu cepat sehingga menyebabkan tidak terhitungnya kendaraan. Untuk dapat mengetahui pengaruh kecepatan kendaraan pada proses perhitungan akan dilakukan pengujian selanjutnya yang akan dibahas pada subbab selanjutnya.

4. Kendaraan yang berdekatan menyebabkan *blob* hasil filter *background subtraction* menjadi satu

   Pada saat dilakukan pengujian terdapat kendaraan yang letaknya berdekatan satu sama lain. Dikarenakan posisi yang berdekatan tersbeut, *blob* dari masing-masing kendaraan akan menjadi satu kesatuan. Kemudian disaat kendaraan-kendaraan tersebut akan melewati garis, kendaraan-kendaraan yang berdekatan tersebut akan terhitung menjadi satu . Alasan menjadi 1 perhitungan dikarenakan 2 kendaraan tersebut menjadi 1 *blob* yang besar. *Blob* yang besar ini terjadi diakibatkan filter pada *background subtraction*. Pada *background subtraction* terdapat filter *dilate* yang digunakan untuk memperbesar kembali *blob* yang telah melewati filter *erode*. Dikarenakan besar kernel *dilate* yang terlalu besar, 2 kendaraan yang memiliki *blob* yang berbeda akan menjadi satu *blob* yang besar seperti yang ditunjukkan pada gambar 4.19. Akibat dari peristiwa ini, metode yang digunakan masih kurang cocok apabila digunakan perhitungan pada kondisi macet di jalan.



Gambar 4.19 Dua Kendaraan Satu *Blob*

4.4.1. Pengujian Pengaruh Ukuran Kernel Filter *Erode* dan *Dilate*

Berdasarkan hasil analisa secara *online*, diketahui bahwa permasalahan yang muncul diakibatkan oleh filter dari *background subtraction*. Maka dari itu akan dilakukan pengujian secara *offline*. Pengujian secara *offline* adalah pengujian dengan menggunakan rekaman video tanpa dilakukannya *image processing* pada alat penghitung jumlah kendaraan. Proses analisa dilakukan dengan cara memasukkan video kedalam program *image processing*. Analisa yang akan dilakukan adalah pengaruh yang diberikan dari ukuran kernel dari filter *erode* dan *dilate* pada program.

Pada pengujian 1,2,3, dan 4 akan dilakukan pengujian secara *offline* dengan menggunakan rekaman video. Tabel dibawah ini merupakan hasil tiap pengujian dengan menggunakan ukuran kernel yang berbeda pada tiap filter.

Tabel 4.4 Hasil Pengujian 1 *Offline*

No.	Ukuran Kernel				Total		
	***Erode* 1**	***Dilate* 1**	***Erode* 2**	***Dilate* 2**	**Perhitungan kendaraan**	**Perhitungan *noise***	**Akurasi**
1	3x3	25x25	29x29	33x33	6	0	85,71%
2	5x5	25x25	29x29	33x33	5	0	71,43%
3	7x7	25x25	29x29	33x33	4	0	57,14%

4	3x3	21x21	29x29	33x33	5	0	71,43%
5	3x3	29x29	29x29	33x33	4	0	57,14%
6	3x3	25x25	25x25	33x33	4	0	57,14%
7	3x3	25x25	33x33	33x33	5	0	71,43%
8	3x3	25x25	29x29	29x29	6	0	85,71%
9	3x3	25x25	29x29	37x37	6	0	85,71%

Tabel 4.5 Hasil Pengujian 2 *Offline*

No.	Ukuran Kernel				Total		
	***Erode* 1**	***Dilate* 1**	***Erode* 2**	***Dilate* 2**	**Perhitungan kendaraan**	**Perhitungan *noise***	**Akurasi**
1	3x3	25x25	29x29	33x33	10	0	45,45%
2	5x5	25x25	29x29	33x33	8	0	36,36%
3	7x7	25x25	29x29	33x33	5	0	22,72%
4	3x3	21x21	29x29	33x33	10	1	45,45%
5	3x3	29x29	29x29	33x33	14	0	63,63%
6	3x3	25x25	25x25	33x33	14	0	63,63%
7	3x3	25x25	33x33	33x33	9	1	40,90%
8	3x3	25x25	29x29	29x29	11	0	50%
9	3x3	25x25	29x29	37x37	10	0	45,45%

Tabel 4.6 Hasil Pengujian 3 *Offline*

No.	Ukuran Kernel				Total		
	***Erode* 1**	***Dilate* 1**	***Erode* 2**	***Dilate* 2**	**Perhitungan kendaraan**	**Perhitungan *noise***	**Akurasi**
1	3x3	25x25	29x29	33x33	15	0	62,5%
2	5x5	25x25	29x29	33x33	13	0	54,16%
3	7x7	25x25	29x29	33x33	9	0	37,5%
4	3x3	21x21	29x29	33x33	10	0	41,6%
5	3x3	29x29	29x29	33x33	15	0	62,5%
6	3x3	25x25	25x25	33x33	15	0	62,5%
7	3x3	25x25	33x33	33x33	12	0	50%
8	3x3	25x25	29x29	29x29	15	0	62,5%
9	3x3	25x25	29x29	37x37	15	0	62,5%

Tabel 4.7 Hasil Pengujian 4 *Offline*

No.	Ukuran Kernel				Total		
	***Erode* 1**	***Dilate* 1**	***Erode* 2**	***Dilate* 2**	**Perhitungan kendaraan**	**Perhitungan *noise***	**Akurasi**
1	3x3	25x25	29x29	33x33	8	0	50%
2	5x5	25x25	29x29	33x33	6	0	37,5%
3	7x7	25x25	29x29	33x33	6	0	37,5%
4	3x3	21x21	29x29	33x33	7	0	43,75%
5	3x3	29x29	29x29	33x33	7	4	43,75%
6	3x3	25x25	25x25	33x33	7	5	43,75%

7	3x3	25x25	33x33	33x33	8	0	50%
8	3x3	25x25	29x29	29x29	9	0	56,25%
9	3x3	25x25	29x29	37x37	8	0	50%

Berdasarkan tabel 4.4, 4.5, 4.6, dan 4.7 didapatkan nilai akurasi pada tiap ukuran kernel yang berbeda pada masing-masing filter dengan menggunakan video pengujian yang berbeda. Pada nomor 1 dari tabel menunjukkan nilai *default* dari ukuran kernel yang digunakan pada pengujian secara *online*. Untuk nomor 2 dan 3 dari tabel dilakukan pembedaan pada ukuran kernel *erode* 1. Kemudian pada nomor 4 dan 5 dari tabel dilakukan pembedaan pada ukuran kernel *dilate* 1. Untuk nomor 6 dan 7 dari tabel dilakukan pembedaan pada ukuran kernel *erode* 2. Dan terakhir pada nomor 8 dan 9 dari tabel dilakukan pembedaan pada ukuran kernel *dilate* 2. Sebelum dilakukan pembedaan nilai ukuran filter akan dilakukan perhitungan kembali dengan menggunakan nilai kernel *default*

Hasil dari tabel didapatkan pembedaan ukuran kernel filter dapat mempengaruhi tingkat akurasi perhitungan kendaraan. Pada nilai default juga mendapatkan nilai berbeda dengan nilai perhitungan saat online dikarenakan hasil video yang terekam akan dilakukan stabilisasi gambar kembali saat memasuki program. Pada pengujian 1, tingkat akurasi terbaik yang didapatkan adalah pada nomor 1, 8, dan 9 dengan tingkat akurasi sebesar 85,71%. Untuk pengujian 2 didapatkan tingkat akurasi terbaik pada nomor 5 dan 6 dengan tingkat akurasi sebesar 63,63%. Pada pengujian 3, tingkat akurasi terbaik yang didapatkan adalah pada nomor 1, 5, 6, 8 dan 9 dengan tingkat akurasi sebesar 62,5%. Dari hasil pengujian 3 tidak didapatkan perbaikan pada tingkat akurasi perhitungan. Dan pada pengujian 4, tingkat akurasi terbaik yang didapatkan adalah pada nomor 8 dengan tingkat akurasi sebesar 56,25%.

Dari keempat pengujian didapatkan kesamaan yaitu peningkatan nilai ukuran kernel pada *erode* 1 dapat menyebabkan penurunan nilai akurasi. Selain itu perubahan yang dilakukan juga dapat menimbulkan perhitungan dikarenakan *noise* sehingga dapat terjadi miskalkulasi.

4.4.2. Pengujian Pengaruh Pencahayaan

Pengujian ini dilakukan untuk melihat pengaruh pencahayaan terhadap program yang telah dibuat. Pengujian dilakukan pada kondisi pencahayaan lingkungan yang berbeda, yaitu saat siang hari dan malam hari. Siang hari menunjukkan pencahayaan yang berasal dari matahari dalam keadaan terang sedangkan pada malam hari mendapatkan

pencahayaan dari lampu-lampu yang menyinari di sepanjang jalan namun menunjukkan kondisi cahaya yang lebih redup. Pengujian pada tiap kondisi pencahayaan akan dilakukan dua kali. Pengujian dilakukan dalam kondisi ideal yaitu *quadcopter* dalam keadaan tidak terbang kemudian *quadcopter* beserta alat penghitung jumlah kendaraan akan ditempatkan pada suatu ketinggian. Kamera pada alat penghitung kendaraan akan diarahkan menghadap bawah. Pengujian dilakukan dalam kondisi ideal supaya hasil perhitungan hanya untuk melihat pengaruh dari pencahayaan tanpa melihat faktor lain yang dapat menyebabkan tidak terhitungnya kendaraan. Dari hasil pengujian didapatkan hasil sebagai berikut.

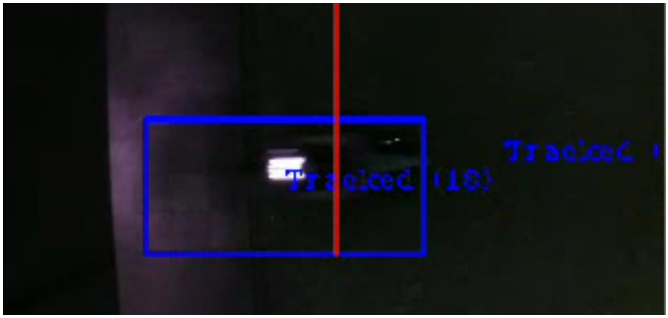
Tabel 4.8 Hasil Pengujian Pengaruh Pencahayaan

Kondisi	Pengujian	Kendaraan yang lewat	Perhitungan kendaraan	Akurasi
Siang hari	1	8	8	100%
	2	10	10	100%
Malam hari	1	6	5	83,3 %
	2	7	4	57,143%

Berdasarkan tabel 4.8 didapatkan akurasi mencapai 100% untuk kondisi siang hari dalam dua pengujian. Namun terjadi penurunan akurasi pada kondisi malam hari menjadi 83,3% dan 57,143% pada dua pengujian. Hasil akurasi pengujian dua kondisi pencahayaan tersebut didapatkan rata-rata akurasi perhitungan 100% pada siang hari dan 70, 22% pada malam hari. Dari hasil analisa pada malam hari, penurunan pada tingkat akurasi dikarenakan kendaraan tidak terhitung diakibatkan label dari *blob* kendaraan saat sebelum melewati garis dan setelah melewati garis berbeda label. Perbedaan ini dikarenakan dari efek *background subtraction* yang terjadi pada gambar. Saat keadaan redup, maka pembacaan pergerakan dari *background subtraction* adalah lampu kendaraan serta pantulan lampu dari kendaraan seperti yang ditunjukkan pada gambar dibawah ini.

Gambar 4.20 Dua *Blob* pada Malam Hari

Kemudian setelah melewati garis, *blob* antara lampu kendaraan dan pantulan lampu kendaraan akan menjadi satu *blob* sehingga menyebabkan tidak terhitungnya kendaraan tersebut. Gambar dibawah ini akan menunjukkan penggabungan dua *blob* antara lampu kendaraan dan pantulan kendaraan.



Gambar 4.21 Penggabungan *Blob* pada Malam Hari

4.4.3. Pengujian Pengaruh Kecepatan Kendaraan

Pengujian pengaruh kecepatan kendaraan merupakan pengujian terhadap respon dari program apabila kendaraan bergerak pada kecepatan tinggi dan kecepatan rendah. Proses pengujian dilakukan dengan menggunakan motor yang akan bergerak pada kecepatan yang berbeda dimulai dari kecepatan rendah hingga kecepatan tinggi. Hasil pengujian yaitu terdeteksi tidaknya motor yang bergerak saat dilakukan proses

perhitungan jumlah kendaraan. Berikut merupakan hasil pengujian pengaruh kecepatan kendaraan pada proses perhitungan

Tabel 4.9 Hasil Pengujian Pengaruh Kecepatan Kendaraan

No.	Kecepatan	*Blob* Terdeteksi	*Blob* Terlacak	*Blob* Terhitung
1	20 $^{km}/_{jam}$	✓	✓	✓
2	30 $^{km}/_{jam}$	✓	✓	✓
3	40 $^{km}/_{jam}$	✓	✓	✓
4	50 $^{km}/_{jam}$	✓	✓	✓
5	60 $^{km}/_{jam}$	✓	×	×

Dari tabel 4.9 didapatkan kendaraan yang bergerak dengan kecepatan 60 $^{km}/_{jam}$ tidak mengalam proses pelacakan dan proses perhitungan namun *blob* hasil dari *background subtraction* dapat terlihat. Sedangkan untuk kendaraan yang bergerak dengan kecepatan 20 $^{km}/_{jam}$ sampai 40 $^{km}/_{jam}$ masih dapat terhitung. Ini dikarenakan *frame rate* yang didapatkan saat dilakukan *image processing* hanya mencapai 8 fps sehingga program tidak sempat untuk melakukan proses perhitungan.

BAB 5
PENUTUP

5.1. Kesimpulan

Berdasarkan proses analisa data yang didapatkan pada tugas akhir ini, dapat disimpulkan bahwa

1. Pembacaan ketinggian dari BMP180 memiliki nilai *error* sebesar +0,1514 m
2. Filter *erode* dan *dilate* yang digunakan pada *background subtraction* memiliki pengaruh besar dalam melakukan perhitungan kendaraan.
3. *Quadcopter* masih dapat menjaga posisi *quadcopter* dalam keaadan stabil dalam kondisi kecepatan angin sampai 1,7 $^{m}/_{s}$ dan akan tidak stabil jika kecepatan angin mencapai 2 $^{m}/_{s}$
4. Perhitungan secara *online* memiliki tingkat akurasi sebesar 61%
5. Kondisi pencahayaan memiliki pengaruh dalam perhitungan kendaraan. Perhitungan saat malam hari memiliki akurasi yang lebih rendah dibandingkan saat siang hari
6. Perhitungan kendaraan tidak dapat dilakukan jika kendaraan melaju melebihi 60 $^{km}/_{jam}$

5.2. Saran

Dikarenakan tugas akhir ini masih memiliki banyak kekurangan dan diperlukannya pengembangan lebih lanjut, maka penulis akan memberikan saran perbaikan yaitu sebagai berikut.

1. Dalam melakukan tracking kendaraan bermotor dapat digunakan metode *deep learning* dikarenakan memiliki akurasi yang lebih baik pada riset lain [15]. Dalam paper ini dijelaskan bahwa digunakan berbagai macam metode *deep learning* dimana YOLO mencapai kebenaran mencapai 0.994.
2. Perancangan ulang *quadcopter* yang lebih stabil terutama bagian baterai supaya dapat terbang lebih lama.

[Halaman ini sengaja dikosongkan]

DAFTAR PUSTAKA

[1] "Badan Pusat Statistik." [Daring]. Tersedia pada: https://www.bps.go.id/linkTableDinamis/view/id/1133. [Diakses: 12-Jun-2019].

[2] G. Cookson, "INRIX Global Traffic Scorecard," hlm. 44.

[3] L. Zongfan dan Z. Ming, "Annual average daily traffic estimation from short traffic counts," dalam *2015 International Conference on Transportation Information and Safety (ICTIS)*, 2015, hlm. 250–252.

[4] R. Austin, *Unmanned aircraft systems: UAVs design, development and deployment*. Reston, Va. : Chichester: American Institute of Aeronautics and Astronautics ; Wiley, 2010.

[5] *INAV: Navigation-enabled flight control software. Contribute to iNavFlight/inav development by creating an account on GitHub*. INAVFlight, 2019.

[6] "Raspberry Pi 3 Model B+," *Raspberry Pi*. .

[7] H. Mayditia, "PERANCANGAN DAN IMPLEMENTASI PENGENDALIAN SISTEM ATTITUDE INERSIAL SATU SUMBU MENGGUNAKAN REACTION WHEEL DAN GIROSKOP MEMS," hlm. 105, 2011.

[8] "OpenCV." [Daring]. Tersedia pada: https://opencv.org/. [Diakses: 14-Mei-2019].

[9] A. Kaehler dan G. R. Bradski, *Learning OpenCV 3: computer vision in C++ with the OpenCV library*, First edition, Second release. Sebastopol, CA: O'Reilly Media, 2017.

[10] J. Sklansky, "Finding the convex hull of a simple polygon," *Pattern Recognit. Lett.*, vol. 1, no. 2, hlm. 79–83, Des 1982.

[11] C. Pornpanomchai, T. Liamsanguan, dan V. Vannakosit, "Vehicle detection and counting from a video frame," dalam *2008 International Conference on Wavelet Analysis and Pattern Recognition*, 2008, vol. 1, hlm. 356–361.

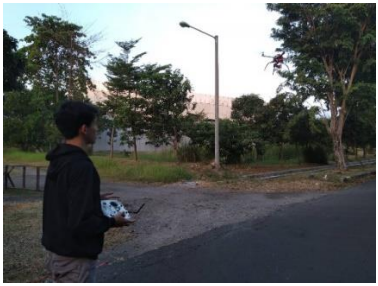
[12] K. V. Najiya dan M. Archana, "UAV Video Processing for Traffic Surveillence with Enhanced Vehicle Detection," dalam *2018 Second International Conference on Inventive Communication and Computational Technologies (ICICCT)*, 2018, hlm. 662–668.

[13] M. Elloumi, R. Dhaou, B. Escrig, H. Idoudi, dan L. A. Saidane, "Monitoring road traffic with a UAV-based system," dalam *2018 IEEE Wireless Communications and Networking Conference (WCNC)*, 2018, hlm. 1–6.

[14] H. Niu, N. Gonzalez-Prelcic, dan R. W. Heath, “A UAV-Based Traffic Monitoring System - Invited Paper,” dalam *2018 IEEE 87th Vehicular Technology Conference (VTC Spring)*, 2018, hlm. 1–5.
[15] J. Zhu *dkk.*, “Urban Traffic Density Estimation Based on Ultrahigh-Resolution UAV Video and Deep Neural Network,” *IEEE J. Sel. Top. Appl. Earth Obs. Remote Sens.*, vol. 11, no. 12, hlm. 4968–4981, Des 2018.
[16] T. Moranduzzo dan F. Melgani, “Automatic Car Counting Method for Unmanned Aerial Vehicle Images,” *IEEE Trans. Geosci. Remote Sens.*, vol. 52, no. 3, hlm. 1635–1647, Mar 2014.
[17] H. Zhou, L. Wei, M. Fielding, D. Creighton, S. Deshpande, dan S. Nahavandi, “Car park occupancy analysis using UAV images,” dalam *2017 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, 2017, hlm. 3261–3265.
[18] T. Moranduzzo, A. Zeggada, dan F. Melgani, “A fast screening method for detecting cars in UAV images over urban areas,” dalam *2015 Joint Urban Remote Sensing Event (JURSE)*, 2015, hlm. 1–4.
[19] R. Ke, S. Kim, Z. Li, dan Y. Wang, “Motion-vector clustering for traffic speed detection from UAV video,” dalam *2015 IEEE First International Smart Cities Conference (ISC2)*, 2015, hlm. 1–5.
[20] S. Li, W. Zhang, G. Li, L. Su, dan Q. Huang, “Vehicle Detection in UAV Traffic Video Based on Convolution Neural Network,” dalam *2018 IEEE Conference on Multimedia Information Processing and Retrieval (MIPR)*, 2018, hlm. 1–6.

LAMPIRAN A

Dokumentasi



[Halaman ini sengaja dikosongkan]

LAMPIRAN B

Program Raspberry Pi

```
#include <opencv2/opencv.hpp>
#include <iostream>
#include <string>
#include <fstream>
#include <math.h>
#include "bmp180.h"
#include <wiringPi.h>
#include <wiringPiI2C.h>
#include <raspicam/raspicam_cv.h>

#define OSS BMP180_STANDARD

using namespace std;
using namespace cv;
//=========================================FIRST
INIT=============================================
=============
int video_width = 320;
int video_height = 240;
bool firstFrame;
int countRight = 0;
int countLeft = 0;
int frms = 0;
int total = 0;
char str[200];
Point lineA(video_width / 2, video_height / 2 - 90);
Point lineB(video_width / 2, video_height / 2 + 90);
ofstream file4("Total.txt");
ofstream file5("FlightAltitude(Unfiltered).txt");
ofstream file6("CountAltitude.txt");
ofstream file7("FlightAltitude(Filtered).txt");

//============================BMP180============
================================================
============
```

```
short AC1,AC2,AC3,B1,B2,MB,MC,MD;
unsigned short AC4,AC5,AC6;
int fd;
float alt_filter_curr, alt_filter_prev, alt_filter;
float real_altitude, altitude; //dalam satuan h
char I2C_readByte(int reg)
{
   return (char)wiringPiI2CReadReg8(fd,reg);
}

unsigned short I2C_readU16(int reg)
{
   int MSB,LSB;
   MSB = I2C_readByte(reg);
   LSB = I2C_readByte(reg + 1);
   int value = (MSB << 8) +LSB;
   return (unsigned short)value;
}

short I2C_readS16(int reg)
{
   int result;
   result = I2C_readU16(reg);
   if (result > 32767)result -= 65536;
   return (short)result;
}
void I2C_writeByte(int reg,int val)
{
   wiringPiI2CWriteReg8(fd,reg,val);
}

void load_calibration()
{
   AC1 = I2C_readS16(BMP180_CAL_AC1);
   AC2 = I2C_readS16(BMP180_CAL_AC2);
   AC3 = I2C_readS16(BMP180_CAL_AC3);
   AC4 = I2C_readU16(BMP180_CAL_AC4);
   AC5 = I2C_readU16(BMP180_CAL_AC5);
   AC6 = I2C_readU16(BMP180_CAL_AC6);
```

```
   B1  = I2C_readS16(BMP180_CAL_B1);
   B2  = I2C_readS16(BMP180_CAL_B2);
   MB  = I2C_readS16(BMP180_CAL_MB);
   MC  = I2C_readS16(BMP180_CAL_MC);
   MD  = I2C_readS16(BMP180_CAL_MD);
}
int read_raw_temp()
{
   int raw;
   I2C_writeByte(BMP180_CONTROL,BMP180_READTEMPCMD);
   delay(5);  //5ms;
   raw = I2C_readByte(BMP180_TEMPDATA) << 8;
   raw += I2C_readByte(BMP180_TEMPDATA+1);
   return raw;

}
int read_raw_pressure()
{
   int MSB,LSB,XLSB,raw;

I2C_writeByte(BMP180_CONTROL,BMP180_READPRESSURECM
D +(OSS << 6));
   switch(OSS)
   {
      case BMP180_ULTRALOWPOWER:
         delay(5);break;
      case BMP180_HIGHRES:
         delay(14);break;
      case BMP180_ULTRAHIGHRES:
         delay(26);break;
      default :
         delay(8);
   }
   MSB  = I2C_readByte(BMP180_PRESSUREDATA);
   LSB  = I2C_readByte(BMP180_PRESSUREDATA + 1);
   XLSB = I2C_readByte(BMP180_PRESSUREDATA + 2);
   raw = ((MSB << 16) + (LSB << 8) + XLSB) >> (8 - OSS);
   return raw;
}
```

```
float read_temperature()
{
   float T;
   int UT,X1,X2,B5;
   UT = read_raw_temp();
   X1 = ((UT - AC6)*AC5) >> 15;
   X2 = (MC << 11) / (X1 + MD);
   B5 = X1 + X2;
   T = ((B5 + 8) >> 4) /10.0;
   return T;
}

int read_pressure()
{
   int P;
   int UT,UP,X1,X2,X3,B3,B5,B6;
   unsigned int B4;
   int B7;
   UT = read_raw_temp();
   UP = read_raw_pressure();

   X1 = ((UT - AC6)*AC5) >> 15;
   X2 = (MC << 11) / (X1 + MD);
   B5 = X1 + X2;

   //Pressure Calculations
   B6 = B5 - 4000;
   X1 = (B2 * (B6 * B6) >> 12) >> 11;
   X2 = (AC2 * B6) >> 11;
   X3 = X1 + X2;
   B3 = (((AC1 * 4 + X3) << OSS) + 2) / 4;
   X1 = (AC3 * B6) >> 13;
   X2 = (B1 * ((B6 * B6) >> 12)) >> 16;
   X3 = ((X1 + X2) + 2) >> 2;
   B4 = (AC4 * (X3 + 32768)) >> 15;
   B7 = (UP - B3) * (50000 >> OSS);
   if (B7 < 0x80000000){P = (B7 * 2) / B4;}
   else {P = (B7 / B4) * 2;}
   X1 = (P >> 8) * (P >> 8);
```

```
  X1 = (X1 * 3038) >> 16;
  X2 = (-7357 * P) >> 16;
  P = P + ((X1 + X2 + 3791) >> 4);
  return P;

}
float read_altitude()
{
  float pressure,altitude;
  float sealevel_hpa = 1013.25;
  pressure = (float)read_pressure();
        pressure /= 100;
  altitude = 44330.0 * (1.0 - pow(pressure / sealevel_hpa,(1.0/5.255)));
  return altitude;
}
//=========================================OBJECT
TRACKING=========================================
==============
class Blob
{
public:
        vector<Point> currentContour;
        vector<Point> center_now;

        Rect currentBoundingRect;
        Point center;
        int deltax, deltay;
        int numofConsecutiveWithoutMatch;

        Blob(vector<Point> _contour);
};

Blob::Blob(vector<Point>_contour)
{
        currentContour = _contour;
        currentBoundingRect = boundingRect(currentContour);
        center.x = currentBoundingRect.x +
(currentBoundingRect.width / 2);
```

```
		center.y = currentBoundingRect.y + (currentBoundingRect.height / 2);
	}

	class centroidTracker {
	public:
		int nextObjID = 0;
		int maxDisappeared = 5; //max hilang di frame

		vector< pair <int, Point> > objects;
		vector<int> disappeared;

		void regist(Point centroid)
		{
			objects.push_back(make_pair(nextObjID, centroid));
			disappeared.push_back(0);
			nextObjID++;
		}

		void deregist(int objID)
		{
			objects.erase(objects.begin() + objID);
			disappeared.erase(disappeared.begin() + objID);
		}

		vector<pair<int, Point> > update(vector<Point> inputCentroids)
		{
			if (inputCentroids.size() == 0)
			{
				for (int i = 0; i < objects.size(); i++)
				{
					disappeared[i] += 1;
					if (disappeared[i] > maxDisappeared)
					{
						deregist(i);
					}
				}
				return objects;
```

```
		}
		if (objects.size() == 0)
		{
			for (int i = 0; i < inputCentroids.size(); i++)
			{
				regist(inputCentroids[i]);
			}
		}
		else
		{
			vector< pair <float, int> > rowsSorted;
			vector<int> storeArgmin;
			vector<int> realRows;
			vector<int> realCols;

			for (int i = 0; i < objects.size(); i++) // object size sebelumnya
			{
				vector<float> rows;

				for (int j = 0; j < inputCentroids.size(); j++) // input centroid object sekarang
				{
					Point diff = objects[i].second - inputCentroids[j];
					float mag = sqrt(diff.x*diff.x + diff.y*diff.y);
					rows.push_back(mag);
				}
				int argMin = distance(rows.begin(), min_element(rows.begin(), rows.end())); //ngambil label yang nilai paling kecil
				storeArgmin.push_back(argMin); // masukin ke store

	rowsSorted.push_back(make_pair(rows[argMin], i)); //diberi label baru
			}
```

```
                        //for (int i = 0; i < objects.size(); i++) // object
size sebelumnya
                        //{
                        //      cout << objects[i].second << " ";
                        //}
                        //cout << "obj center" << endl;

                        //for (int i = 0; i < inputCentroids.size(); i++)
// object size sebelumnya
                        //{
                        //      cout << inputCentroids[i] << " ";
                        //}
                        //cout << "input center" << endl;

                        sort(rowsSorted.begin(), rowsSorted.end());
                        for (int i = 0; i < rowsSorted.size(); i++)
                        {

        realRows.push_back(rowsSorted[i].second);

        realCols.push_back(storeArgmin[realRows[i]]);
                        }
                        /*for (int i = 0; i < realRows.size(); i++)
                        {
                        cout << realRows[i] << " " ;
                        }
                        cout << " realRows" << endl;
                        for (int i = 0; i < realCols.size(); i++)
                        {
                        cout << realCols[i] << " " ;
                        }
                        cout << " realCols" << endl;*/
                        vector<int> usedRows;
                        vector<int> usedCols;

                        for (int i = 0; i < realCols.size(); i++)
                        {
                                int _row = realRows[i];
                                int _col = realCols[i];
```

```
                                        if ((find(usedRows.begin(), usedRows.end(), _row) != usedRows.end()) || (find(usedCols.begin(), usedCols.end(), _col) != usedCols.end()))
                                        {
                                                continue;
                                        }
                                        //int objectID = objects[_row].first;
                                        objects[_row].second = inputCentroids[_col];
                                        disappeared[_row] = 0;

                                        usedRows.push_back(_row);
                                        usedCols.push_back(_col);
                                }

                                /*for (int i = 0; i < usedRows.size(); i++)
                                {
                                cout << usedRows[i] << " ";
                                }
                                cout << " usedRows" << endl;

                                for (int i = 0; i < usedCols.size(); i++)
                                {
                                cout << usedCols[i] << " ";
                                }
                                cout << " usedCols" << endl;*/

                                vector<int> unusedRows;
                                vector<int> unusedCols;

                                for (int i = 0; i < realCols.size(); i++)
                                {
                                        if (find(usedRows.begin(), usedRows.end(), i) != usedRows.end())
                                        {
                                                continue;
                                        }
                                        unusedRows.push_back(i);
```

```
                        }

                        int b = 0;
                        for (int i = 0; i < realCols.size(); i++)
                        {
                                if              (find(usedCols.begin(),
usedCols.end(), i) != usedCols.end())
                                {
                                        continue;
                                }
                                b++;
                                unusedCols.push_back(i);
                        }

                        if (inputCentroids.size() > objects.size() && b
== 0)
                        {

        unusedCols.push_back(inputCentroids.size() - 1);
                        }
                        /*for (int i = 0; i < unusedRows.size(); i++)
                        {
                        cout << unusedRows[i] << " unused rows" <<
endl;

                        }

                        for (int i = 0; i < unusedCols.size(); i++)
                        {
                        cout << unusedCols[i] << " unused cols" <<
endl;
                        }*/

                        sort(unusedRows.begin(), unusedRows.end(),
greater<int>());
                        if (objects.size() >= inputCentroids.size())
                        {
                                for (int i = 0; i < unusedRows.size();
i++)
```

```
                                        {
                                                int _row = unusedRows[i];
                                                disappeared[_row] += 1;
                                                if    (disappeared[_row]    >
maxDisappeared)
                                                {
                                                        deregist(_row);
                                                }
                                        }
                                }
                                else
                                {
                                        for (int i = 0; i < unusedCols.size();
i++)
                                        {
                                                int _col = unusedCols[i];

        regist(inputCentroids[_col]);
                                        }
                                }
                        }
                        return objects;
                }
};
//============================================IMAGE
STABILIZATION=====================================
=============
const int SMOOTHING_RADIUS = 30;
double a = 0;
double x = 0;
double y = 0;

Mat current, cur_gray;
Mat previous, prev_gray;
Mat last_T, T;
Mat stabilize, stabilize_gray;

double avg_x = 0;
double avg_y = 0;
```

```
double avg_a = 0;

struct TransformParam
{
	TransformParam() {}
	TransformParam(double _dx, double _dy, double _da) {
		dx = _dx;
		dy = _dy;
		da = _da;
	}
	double dx;
	double dy;
	double da; // angle
};

struct Trajectory {
	Trajectory() {}
	Trajectory(double _x, double _y, double _a) {
		x = _x;
		y = _y;
		a = _a;
	}
	double x;
	double y;
	double a; // angle
};
//==========================================BACKGR
OUND
SUBSTRACTION=====================================
==========
Mat MaskMOG2;
Mat OutputMaskMOG2;
int kernel_size1 = 6; //DEFAULT BLUR
int kernel_size2 = 1; //DEFAULT ERODE 1
int kernel_size3 = 14; //DEFAULT ERODE 2
int kernel_size4 = 12; //DEFAULT DILATE 1
int kernel_size5 = 16 ; //DEFAULT DILATE 2
const int kernel_size_max = 30;
```

```
//================================================
=VEHICLE
COUNTING=======================================
=====
int countingHorizontal(vector< pair <int, Point> > prevObj, vector< pair
<int, Point> > currentObj, Mat _output)
{
	int total = 0;
	for (int i = 0; i < prevObj.size(); i++)
	{
		for (int j = 0; j < currentObj.size(); j++)
		{
			if (prevObj[i].first != currentObj[j].first)
			{
				continue;
			}
			Point prevPoint = prevObj[i].second;
			Point currentPoint = currentObj[j].second;

			if (prevPoint.x > lineA.x && currentPoint.x
<= lineA.x && currentPoint.y <= lineB.y && currentPoint.y >= lineA.y)
//flow to left image
			{
				total++;
				line(_output, lineA, lineB, Scalar(0,
255, 0), 2);
				if (file6.is_open())
				{
					file6 << alt_filter << endl;
				}
				else cout << "Unable to open file6"
<< endl;
			}

			//if (prevPoint.x <= lineA.x &&
currentPoint.x > lineA.x && currentPoint.y <= lineB.y &&
currentPoint.y >= lineA.y) //flow to right image
			//{
			//	total++;
```

```
                                //          line(_output, lineA, lineB, Scalar(0,
255, 0), 2);
                                //}
                        }
                }
                return total;
}
//================================================
LIST                                OF                                VOID
FUNCTIONS=============================================
=
void ImageStab();
void BackgroundSubs(Mat _input);
void cal Erode 1(int, void*);
void cal Erode 2(int, void*);
void cal Dilate 1(int, void*);
void cal Dilate 2(int, void*);
void calBlur(int, void*);

int main()
{
        cout << "Vehicle Counting Program by Ladiva
Bachrulrachman" << endl;
        if(wiringPiSetup() < 0) return 1;
   fd = wiringPiI2CSetup(BMP180_Address);
   load_calibration();
   float altitude_init = read_altitude();
        centroidTracker ct;
        //namedWindow("Callibration", WINDOW_KEEPRATIO);
        VideoWriter countVid("HasilCount.avi", CV_FOURCC('M',
'P', 'E', 'G'), 13, Size(video_width, video_height));
        VideoWriter oriVid("Original.avi", CV_FOURCC('M', 'P', 'E',
'G'), 13, Size(video_width, video_height));
        firstFrame = true;
        //VideoCapture cap("Data8.avi");
        //cap >> previous;
        //cout << "Opening Camera" << endl;
        raspicam::RaspiCam_Cv cap;
        cout << "Opening Camera" << endl;
```

```
	if ( !cap.open()) {cerr<<"Error Opening camera"<<endl;return -
1;}
	cap.grab();
	cap.retrieve(previous);
	if (previous.empty())
	{
		cout << "Check camera (Previous)" << endl;
		return 0;
	}
	resize(previous, previous, Size(video_width, video_height));
	cvtColor(previous, prev_gray, CV_BGR2GRAY);
	createTrackbar("Blur", "Callibration", &kernel_size1,
kernel_size_max, calBlur);
	createTrackbar("Erode 1", "Callibration", &kernel_size2,
kernel_size_max, calErode1);
	createTrackbar("Erode 2", "Callibration", &kernel_size3,
kernel_size_max, calErode2);
	createTrackbar("Dilate 1", "Callibration", &kernel_size4,
kernel_size_max, calDilate1);
	createTrackbar("Dilate 2", "Callibration", &kernel_size5,
kernel_size_max, calDilate2);
	int total = 0;
	vector< pair <int, Point> > objectsPrev;
	while (true)
	{
		vector<Point> centers;

		altitude = read_altitude();
    real_altitude = altitude - altitude_init;
    //printf("Altitude: %.2f h\n",real_alt);
		if (file5.is_open())
		{
			file5 << real_altitude << " " << altitude <<
endl;
		}
		else cout << "Unable to open file5" << endl;
		//printf("Altitude : %.2f h (%.2f h)\n",real_altitude,
read_altitude());
		float a = 0.1;
```

```
alt_filter_curr = altitude*a+(1-a)*alt_filter_prev;
alt_filter = alt_filter_curr - altitude_init;
if(alt_filter < 0) alt_filter = 0 ;
if (file7.is_open())
{
	file7 << alt_filter << " " << alt_filter_curr << endl;
}
else cout << "Unable to open file7" << endl;

cap.grab();
cap.retrieve(current);
//cap >> current;
if (current.empty())
{
	cout << "Check Camera (Current)" << endl;
	return 0;
}
resize(current, current, Size(video_width, video_height));
cvtColor(current, cur_gray, COLOR_BGR2GRAY);

//==================================STABILIZE==========================================
ImageStab();
cvtColor(stabilize, stabilize_gray, COLOR_BGR2GRAY);

//==================================BACKGROUND SUBS====================================
BackgroundSubs(stabilize_gray);

//==================================OBJECT TRACKING====================================
Mat OutputMaskMOG2 = MaskMOG2.clone();
Mat OutputCount = stabilize.clone();
vector<vector<Point>>contours;
findContours(OutputMaskMOG2, contours, RETR_TREE, CHAIN_APPROX_SIMPLE, Point(0, 0));
```

```
vector<vector<Point>>hull(contours.size());
vector<Rect> bound(contours.size());
for (unsigned int i = 0; i < contours.size(); i++)
{
	convexHull(contours[i], hull[i]);
	bound[i] = boundingRect(hull[i]);
	rectangle(OutputCount, bound[i], Scalar(255, 0, 0), 2, 8, 0);
}
for (auto convexHull : hull)
{
	Blob possibleBlob(convexHull);
	Point center;
	center = possibleBlob.center;
	centers.insert(centers.begin(), center);
}
ct.update(centers);
for (int i = 0; i < ct.objects.size(); i++)
{
	char txt[50];
	snprintf(txt, 50, "Tracked (%d)", ct.objects[i].first);
	putText(OutputCount, txt, ct.objects[i].second, FONT_HERSHEY_COMPLEX_SMALL, 0.65, Scalar(255, 0, 0), 1);
}

	//==================================VEHICLE COUNTING==================================
	line(OutputCount, lineA, lineB, Scalar(0, 0, 238), 2);
	if (firstFrame == true)
	{
		objectsPrev = ct.objects;
		firstFrame = false;
	}
	else
	{
		total += countingHorizontal(objectsPrev, ct.objects, OutputCount);
```

```
                    if (file4.is_open())
                    {
                            file4 << total << endl;
                    }
                    else cout << "Unable to open file4" << endl;
                    objectsPrev.clear();
                    objectsPrev = ct.objects;
            }
            sprintf(str, "Kendaraan : %d", total);
            rectangle(OutputCount, Rect(0, 0, 140, 30), Scalar(0, 0,
0), FILLED);
            putText(OutputCount, str, Point(5, 20),
FONT_HERSHEY_SIMPLEX, 0.55, Scalar(255, 255, 50), 2, 8);
            imshow("Output Count", OutputCount);
            countVid.write(OutputCount);
            oriVid.write(current);

            char change = waitKey(1);
            if (change == 27)
            {
                    break; //ESC
            }
            else if (change == 'p')
            {
                    waitKey(0);
            }
            else if (change == 'm')
            {
                    waitKey(1);
            }
            else if (change == 'r') // 'r' for RESET COUNT
            {
                    total = 0;
                    cout << "Count reset to 0\t" << endl;
            }
            else if (change == 'w') //'w' for move line up
            {
                    lineA.y--;
                    lineB.y--;
```

```
		}
		else if (change == 'a') // 'a' for move right up
		{
			lineA.x--;
			lineB.x--;
		}
		else if (change == 's') //'s' for move line down
		{
			lineA.y++;
			lineB.y++;
		}
		else if (change == 'd') //'d' for move line right
		{
			lineA.x++;
			lineB.x++;
		}
		else if (change == '2') //shorten line
		{
			lineA.y++;
			lineB.y--;
		}
		else if (change == '3') //lengthen line
		{
			lineA.y--;
			lineB.y++;
		}
		else if (change == 'u')
		{
			//add area_thresh_up by 100
			area_thresh_up += 100;
			cout	<<	"area_thresh_up=	"	<<
area_thresh_up << endl;
		}
		else if (change == 'j')
		{
			//reduce area_thresh_up by 100
			area_thresh_up -= 100;
			cout	<<	"area_thresh_up=	"	<<
area_thresh_up << endl;
```

```
                }
                else if (change == 'i')
                {
                        //add area_thresh_bottom by 100
                        area_thresh_bottom += 100;
                        cout    <<    "area_thresh_bottom=    "    <<
area_thresh_bottom << endl;
                }
                else if (change == 'k')
                {
                        //reduce area_thresh_bottom by 100
                        area_thresh_bottom -= 100;
                        cout    <<    "area_thresh_bottom=    "    <<
area_thresh_bottom << endl;
                }
                current.copyTo(previous);
                cur_gray.copyTo(prev_gray);
                firstFrame = false;
                alt_filter_prev = alt_filter_curr;
                //frms++;
                //cout << " " << frms << endl;
        }
        //file1.close();
        //file2.close();
        //file3.close();
        file4.close();
        file5.close();
        file6.close();
        file7.close();
        return 0;
}

void ImageStab()
{
        vector <Point2f> prev_corner, cur_corner;
        vector <Point2f> prev_corner2, cur_corner2;
        vector <uchar> status;
        vector <float> err;
        vector <Trajectory> trajectory; // trajectory at all frames
```

```
	vector <Trajectory> smoothed_trajectory; //trajectory at all frames
	vector <TransformParam> new_prev_to_cur_transform;
	vector <TransformParam> prev_to_cur_transform; // previous to current

	goodFeaturesToTrack(prev_gray, prev_corner, 500, 0.001, 20);
	//for (int i = 0; i < prev_corner.size(); i++)
	//{
	//	circle(current, prev_corner[i], 3, Scalar(0, 255, 0), -1, 8);
	//}
	if (prev_corner.empty())
	{
		current.copyTo(previous);
		cur_gray.copyTo(prev_gray);
	}
	else
	{
		calcOpticalFlowPyrLK(prev_gray, cur_gray, prev_corner, cur_corner, status, err);
		//cout << "corner 1: " << prev_corner.size() << " " << cur_corner.size() << endl;
		for (size_t i = 0; i < status.size(); i++)
		{
			if (status[i]) {

	prev_corner2.push_back(prev_corner[i]);

	cur_corner2.push_back(cur_corner[i]);
			}
		}
		//cout << "corner 2: " << prev_corner2.size() << " " << cur_corner2.size() << endl;
		if (cur_corner2.empty() || prev_corner2.empty())
		{
			current.copyTo(stabilize);
			resize(stabilize, stabilize, current.size());
		}
```

```
                else
                {
                        T = c;
                        //cout << T << endl;
                        if (T.data == NULL)
                        {
                                current.copyTo(stabilize);
                        }
                        else
                        {
                                assert(T.data != NULL);
                                T.copyTo(last_T);
                                // decompose T
                                double dx = T.at<double>(0, 2);
                                double dy = T.at<double>(1, 2);
                                double da = atan2(T.at<double>(1,
0), T.at<double>(0, 0));


        prev_to_cur_transform.push_back(TransformParam(dx, dy,
da));

                                for (size_t i = 0; i <
prev_to_cur_transform.size(); i++)
                                {
                                        x                       +=
prev_to_cur_transform[i].dx;
                                        y                       +=
prev_to_cur_transform[i].dy;
                                        a                       +=
prev_to_cur_transform[i].da;

        trajectory.push_back(Trajectory(x, y, a));
                                }
                                for (size_t i = 0; i < trajectory.size();
i++)
                                {
                                        double sum_x = 0;
                                        double sum_y = 0;
```

```
                                        double sum_a = 0;
                                        int count = 0;

                                        for (int j = -
SMOOTHING_RADIUS; j <= SMOOTHING_RADIUS; j++) {
                                            if (i + j >= 0 && i
+ j < trajectory.size()) {
                                                sum_x +=
trajectory[i + j].x;
                                                sum_y +=
trajectory[i + j].y;
                                                sum_a +=
trajectory[i + j].a;
                                                count++;
                                            }
                                        }
                                        avg_a = sum_a / count;
                                        avg_x = sum_x / count;
                                        avg_y = sum_y / count;

            smoothed_trajectory.push_back(Trajectory(avg_x,      avg_y,
avg_a));
                                    }
                                    for (size_t i = 0; i <
prev_to_cur_transform.size(); i++)
                                    {
                                        x                    +=
prev_to_cur_transform[i].dx;
                                        y                    +=
prev_to_cur_transform[i].dy;
                                        a                    +=
prev_to_cur_transform[i].da;

                                        double      diff_x      =
smoothed_trajectory[i].x - x;
                                        double      diff_y      =
smoothed_trajectory[i].y - y;
                                        double      diff_a      =
smoothed_trajectory[i].a - a;
```

```
                                        double dx =
prev_to_cur_transform[i].dx + diff_x;
                                        double dy =
prev_to_cur_transform[i].dy + diff_y;
                                        double da =
prev_to_cur_transform[i].da + diff_a;


        new_prev_to_cur_transform.push_back(TransformParam(dx,
dy, da));
                                    }
                                    T.at<double>(0, 0) = cos(da);
                                    T.at<double>(0, 1) = -sin(da);
                                    T.at<double>(1, 0) = sin(da);
                                    T.at<double>(1, 1) = cos(da);

                                    T.at<double>(0, 2) = dx;
                                    T.at<double>(1, 2) = dy;
                                    warpAffine(previous, stabilize, T,
current.size());
                                    resize(stabilize, stabilize,
current.size());
                            }
                        }
        }
}
void BackgroundSubs(Mat _input)
{
        _input.copyTo(MaskMOG2);
        calBlur(kernel_size1, 0);
        //imshow("Hasil Blur", MaskMOG2);
        absdiff(prev_gray, cur_gray, MaskMOG2);
        threshold(MaskMOG2, MaskMOG2, 30, 255,
THRESH_BINARY);
        //imshow("MOG2 sebelum erode", MaskMOG2);
        calErode1(kernel_size2, 0);
        //imshow("After erode 1", MaskMOG2);
        calDilate1(kernel_size4, 0);
```

```
		//imshow("After dilate 1", MaskMOG2);
		calErode2(kernel_size3, 0);
		//imshow("After erode 2", MaskMOG2);
		calDilate2(kernel_size5, 0);
		//imshow("After dilate 2", MaskMOG2);
}
void calErode1(int, void*)
{
		Mat erode_subs = getStructuringElement(MORPH_RECT,
Size(2 * kernel_size2 + 1, 2 * kernel_size2 + 1), Point(-1, -1));
		erode(MaskMOG2, MaskMOG2, erode_subs);
}
void calErode2(int, void*)
{
		Mat erode_subs2 = getStructuringElement(MORPH_RECT,
Size(2 * kernel_size3 + 1, 2 * kernel_size3 + 1), Point(-1, -1));
		erode(MaskMOG2, MaskMOG2, erode_subs2);
}
void calDilate1(int, void*)
{
		Mat dilate_subs = getStructuringElement(MORPH_RECT,
Size(2 * kernel_size4 + 1, 2 * kernel_size4 + 1), Point(-1, -1));
		dilate(MaskMOG2, MaskMOG2, dilate_subs);
}
void calDilate2(int, void*)
{
		Mat dilate_subs = getStructuringElement(MORPH_RECT,
Size(2 * kernel_size5 + 1, 2 * kernel_size5 + 1), Point(-1, -1));
		dilate(MaskMOG2, MaskMOG2, dilate_subs);
}
void calBlur(int, void*)
{
		Size blur(2 * kernel_size1 + 1, 2 * kernel_size1 + 1);
		GaussianBlur(MaskMOG2, MaskMOG2, blur, 0, 0, 4);
}
```

[Halaman ini sengaja dikosongkan]

BIODATA PENULIS

Penulis merupakan kelahiran dari Jakarta tanggal 10 Oktober 1997. Penulis merupakan anak kedua dari tiga bersaudara. Penulis menjalankan studi Strata-1 di Departemen Teknik Elektro, Insitut Teknologi Sepuluh Nopember, Surabaya pada tahun 2015 setelah lulus dari SMA di SMA Labschool Kebayoran, Jakarta. Penulis mengambil bidang studi elektronika sebagai bidang studi pilihan.

Email : adif.ladiva@gmail.com
HP : 082129129224
Line : adifrachman