UNIVERSITAS INDONESIA

RANCANG BANGUN SISTEM PENGIKUT WARNA MENGGUNAKAN SENSOR KAMERA BERBASIS MIKROKONTROLER 32-BIT

SKRIPSI

Skripsi ini Diajukan Untuk Melengkapi Persyaratan Memperoleh Gelar Sarjana Strata Satu Fisika

ISMAIL
030302043Y

FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM
PROGRAM STUDI FISIKA
PEMINATAN FISIKA INSTRUMENTASI
DEPOK
JUNI 2009

HALAMAN PERNYATAAN ORISINALITAS

Skripsi ini adalah hasil karya saya sendiri,

dan semua sumber baik yang dikutip maupun dirujuk

telah saya nyatakan dengan benar

Nama : Ismail

NPM : 030302043Y

Tanda tangan :

Tanggal : 11 Juni 2009

HALAMAN PENGESAHAN

Skripsi ini diajukan oleh	:
Nama	: Ismail
NPM	: 030302043Y
Program Studi	: Fisika Instrumentasi
Judul Skripsi	: Rancang Bangun Sistem Pengikut Warna Menggunakan Sensor Kamera Berbasis Mikrokontroler 32-Bit

Telah berhasil dipertahankan di hadapan Dewan Penguji dan diterima sebagai bagian persyaratan yang diperlukan untuk memperoleh gelar Sarjana Sains pada Program Studi Fisika, Fakultas Matematika dan Ilmu Pengetahuan Alam, Universitas Indonesia

Dewan Penguji

Pembimbing I : Dr. Prawito (……)

Pembimbing II : Drs. Lingga Hermanto, M.Si (……)

Penguji I : Dr. rer. nat Martarizal (……)

Penguji II : Dr. Sastra Kusumawijaya (……)

Ditetapkan di : Depok

Tanggal : 11 Juni 2009

KATA PENGANTAR

Assalaamu'alaikum Warahmatullahi Wabarakatuh

Alhamdulillah, segala puja bagi Allah SWT Yang Maha Pengasih lagi Maha Penyayang karena dengan izin-Nya, berkah-Nya, pertolongan-Nya dan karunia-Nya, penulis dapat menyelesaikan skripsi ini dengan baik, Insya Allah.

Atas dasar kesadaran penulis tentang keterbatasan pengetahuan, pengalaman, dan kemampuan yang dimiliki, sehingga penulisan skripsi ini masih terdapat kekurangan-kekurangannya. Banyak kesulitan serta hambatan baik secara mental maupun fisik penulis hadapi dalam menyelesaikan skripsi ini. Namun, berkat bantuan dari berbagai pihak akhirnya penulis mampu menyelesaikan skripsi ini sebaik mungkin. Untuk itu penulis ingin mengucapkan terima kasih kepada:

1. Allah SWT Yang telah memberi karunia, rahmah dan hidayah yang tak tergantikan kepada penulis
2. Orang tua penulis, yakni ibu, yang dengan susah payah berjuang sendiri, dan telah bersabar dalam memberikan dukungan materil dan non materil.
3. Rasulullah SAW, yang telah membimbing penulis untuk menapaki jalan yang terjal yang penulis hadapi selama ini.
4. Pak Lingga Hermanto dan Pak Prawito yang telah membimbing penulis, meminjamkan alat ukur dan memberi arahan selama pembuatan skripsi.
5. Pak Marta dan Pak Sastra selaku dosen penguji yang telah memberikan masukan untuk selesainya tulisan ini.
6. Oma Lily Suryajaya,Mrs. Joyce Suryajaya, serta segenap keluarga besarnya yang telah membantu memberikan dukungan materil.
7. Sahabat penulis Agus H, Hanafi, M.Martisa, Oscar dan lainnya yang sudah memberikan sumbangan moril berupa nasihat-nasihat.
8. Mba Ratna selaku staf administrasi, dan staf Departemen Fisika lainnya yang telah membantu.
9. Budi, Sugi, Dony, dan Zamroni dan rekan-rekan fisika 2003 dan 2004 yang tidak dapat penulis sebutkan satu persatu.

10. Allan, Rangga A, dan Haris atas *share* pengalamannya.
11. Semua pihak yang tidak dapat disebutkan satu persatu yang telah banyak membantu secara langsung maupun tidak langsung.

Sebagai penutup, penulis menyadari bahwa hasil yang telah penulis dapatkan masih jauh dari sempurna, namun demikian penulis berharap semoga hasil penelitian ini bisa memberikan manfaat bagi pihak yang membutuhkan. Saran dan kritik yang membangun penulis harapkan untuk pengembangan penelitian selanjutnya.

Wassalamualaikum warahmatullahiwabarakatuh.

Depok, Juni 2009

(Ismail)

HALAMAN PERNYATAAN PERSETUJUAN PUBLIKASI TUGAS AKHIR UNTUK KEPENTINGAN AKADEMIS

Sebagai sivitas akademik Universitas Indonesia, saya yang bertanda tangan di bawah ini:

Nama : Ismail

NPM : 030302043Y

Program Studi : Fisika Instrumentasi Elektronika

Departemen : Fisika

Fakultas : Matematika dan Ilmu Pengetahuan Alam

Jenis Karya : Skripsi

demi pengembangan ilmu pengetahuan, menyetujui untuk memberikan kepada Universitas Indonesia **Hak Bebas Noneksklusif (*NON-exclusif Royalty-Free Right*)** atas karya ilmiah saya yang berjudul:

Rancang Bangun Sistem Pengikut Warna Menggunakan Sensor Kamera Berbasis Mikrokontroler 32-Bit

Beserta perangkat yang ada (jika diperlukan). Dengan Hak Bebas Royalti Noneksklusif ini Universitas Indonesia berhak menyimpan, mengalihmedia/format-kan, mengelola dalam bentuk pangkalan data (*database*), merawat, dan memublikasikan tugas akhir saya selama tetap mencantumkan nama saya sebagai penulis/pencipta dan sebagai pemilik Hak Cipta.

Demikian pernyataan ini saya buat dengan sebenarnya.

Dibuat di : Depok

Pada Tanggal : 11 Juni 2009

Yang menyatakan

(Ismail)

ABSTRAK

Nama : Ismail
Program Studi : Fisika
Judul : Rancang Bangun Sistem Pengikut Warna Menggunakan Sensor Kamera Berbasis Mikrokontroler 32-bit

Dengan menghitung pusat massa dari warna yang spesifik, dan pengontrol servo untuk menggerakkan kamera, telah dibuat suatu sistem pengikut warna dengan parameter warna yang dapat diubah-ubah untuk mengikuti warna RGB. Sistem dibuat dengan menggunakan mikrokontroler 32-bit LPC2106 yang mempunyai kapasitas RAM 64K byte dan ROM sebesar 128K byte dengan kecepatan 60MHz. Sensor kamera CMOS dengan resolusi 352 x 288 digunakan sebagai sumber masukan ke memori *buffer* FIFO dengan kapasitas 1MB sebelum akhirnya diproses pada mikrokontroler. Sebagai masukan parameter warna dan antar muka pengguna, sistem didukung oleh mikrokontroler Atmega16 yang memiliki kapasitas RAM 16K byte dan ROM 512 byte yang bekerja pada frekuensi 16 MHz. Kedua mikrokontroler berkomunikasi secara serial dalam menentukan proses *tracking*. Untuk menangkap citra dan data *tracking*, aplikasi pada komputer dibuat dengan LabVIEW. Jangkauan *tracking* dari sistem yang dibuat adalah 20,9° untuk *pan* dan 15,4° untuk *tilt*. Hasil menunjukkan bahwa tingkat keberhasilan mencapai 80% dengan respon gerak pada jangkauan maksimum kurang dari 1 detik dengan kecepatan gerak maksimum adalah 260° perdetik. Laju keluaran data yang dihasilkan adalah 14 data perdetik.

Kata kunci: Sensor citra CMOS, computer vision, Akuisisi citra, mikrokontroler.

ABSTRACT

Name : Ismail
Study Program: Physics
Topic : Design of Color-based Follower System using Camera and 32 bit Microcontroller

By calculating center of mass of spesific color, and servo controller to move camera, a color tracking system has been made with changeable color parameter to track color within RGB range. This system was build using 32-bit microcontroller LPC2106 which has 64K byte RAM and 128K byte ROM at 60MHz. CMOS image sensor with 352 x 288 resolution used as input of FIFO memory buffer which has 1 MB of capacity before they were processed in. As input parameter and user interface, system was supported by other microcontroller Atmega16 with 16K byte RAM and 512 byte ROM at 16MHz of frequency operation. These both microcontroller communicating serially in order to decide a tracking process. To grab an image and tracked data, computer application was made with LabVIEW. Frame tracking range from the system is about 20,9° for pan tracking and 15,4° for tilt tracking. Result shows success is about 80% with moving response for maximum range less than 1 second and maximum moving velocity is 260° persecond Data rate produced by system is 14 data persecond.

Keywords: CMOS image sensor, computer vision, image acquisition, microcontroller

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

DAFTAR LAMPIRAN

BAB 1
PENDAHULUAN

1.1 Latar Belakang

Minat terhadap bidang pengolahan citra secara digital dimulai pada awal tahun 1921, yaitu pertama kalinya sebuah foto berhasil ditransmisikan secara digital melalui kabel laut dari sebuah kota New York ke kota London (*Bartlane Cable Picture Transmission System*). Keuntungan utama yang dirasakan pada waktu itu adalah pengurangan waktu pengiriman foto dari sekitar 1 minggu menjadi kurang dari 3 jam. Foto tersebut dikirim dalam bentuk kode digital dan kemudian diubah kembali oleh *printer telegraph*.

Sekitar tahun 1960 baru tercatat suatu perkembangan pesat seiring dengan munculnya teknologi komputer yang sanggup memenuhi suatu kecepatan proses dan kapasitas memori yang dibutuhkan oleh berbagai algoritma pengolahan citra. Sejak itu berbagai aplikasi mulai dikembangkan, yang secara umum dapat dikelompokkan ke dalam dua kegiatan:

1. Memperbaiki kualitas suatu gambar (citra) sehingga dapat lebih mudah diinterpretasikan oleh mata manusia.
2. Mengolah informasi yang terdapat pada gambar (citra) untuk keperluan pengenalan obyek secara otomatis oleh suatu mesin.

Bidang ini sangat erat hubungannya dengan ilmu pengenalan pola (*pattern recognition*), yang secara umum bertujuan mengenali suatu obyek dengan cara mengekstraksi informasi penting yang terdapat dalam suatu citra.

Contoh-contoh aplikasi dalam berbagai disiplin ilmu:

- Dalam bidang kedokteran:
  Sistem mendeteksi diagnosis suatu kelainan dalam tubuh manusia melalui citra yang dihasilkan oleh scanner.
- Dalam bidang industri:
  Sistem pemeriksaan suatu produk melalui kamera video
- Dalam bidang perdagangan:
  Sistem untuk mengenali angka/huruf dalam suatu formulir secara otomatis oleh mesin pembaca.

- Dalam bidang militer:

  Sistem pengenalan target peluru kendali melalui sensor visual.

Sejalan dengan hal tersebut diatas, perkembangan teknologi robotika pada saat ini sudah mencapai tahap dimana robot yang diciptakan dibuat semirip mungkin dengan manusia. Baik dari segi penampilan, kemampuan bahkan cara pikir. Salah satu kemampuan manusia yang ingin ditiru adalah penglihatan.

Beberapa tahun lalu, proses pengolahan citra merupakan proses yang hanya dapat dilakukan oleh PC (*Personal Computer*). Pengenalan pola, deteksi objek dan analisa data citra digital memerlukan sejumlah besar daya komputasi, ruang memori, dan peralatan luar seperti *video camera*. Saat ini, sensor optik bahkan telah banyak diaplikasikan pada *mobile phone* dan sensor tersebut dapat dengan mudah diperoleh. Sehingga kita bahkan dapat membuat kamera digital hanya dengan beberapa komponen.

Untuk mengaplikasikan proses citra digital maka diperlukan pemroses yang memiliki kinerja cukup tinggi. Saat ini, mikrokontroler terus menerus berkembang, dan perkembangannya mengarah kepada kemampuan yang dimiliki PC saat ini. Sehingga proses citra digital memungkinkan untuk dikerjakan oleh mikrokontroler.

Salah satu jenis mikrokontroler yang layak digunakan untuk menjalankan proses citra adalah mikrokontroler 32-bit, pilihannya antara lain pada mikrokontroler dengan arsitektur ARM. ARM adalah arsitektur mikroprosesor yang diperuntukkan untuk mengoptimalisasi kombinasi antara performa dan daya. Kriteria inilah yang memungkinkan untuk mengerjakan suatu proses citra digital dengan mikrokontroler. Salah satu jenis prosesor ARM yakni ARM7 bekerja pada frekuensi mulai dari 60 MHz, sehingga masalah daya komputasi untuk proses citra bukan menjadi hambatan bagi mikrokontroler ini.

Dengan mikrokontroler ini dibuat pemroses data citra untuk menggerakkan servo berdasarkan posisi warna suatu objek. Ada 3 komponen utama yang digunakan untuk mengaplikasikan proses citra digital secara *embedded*. Selain mikrokontroler dengan kinerja yang lebih dan sensor kamera sebagai masukan, salah satu yang sangat diperlukan adalah memori *buffer*. Hal ini dikarenakan laju data yang begitu tinggi tidak mampu disinkronisasi dengan baik

oleh *input* mikrokontroler. Mikrokontroler saat ini pada umumnya tidak mampu menampung data yang banyak untuk diproses secara sinkron dengan cepat. Komponen *buffer* yang digunakan harus memiliki kriteria dengan kapasitas memori yang cukup besar untuk menampung data citra yang dihasilkan oleh sensor dan frekuensi kerja yang tinggi untuk mengimbangi kinerja antara sensor kamera dengan mikrokontroler.

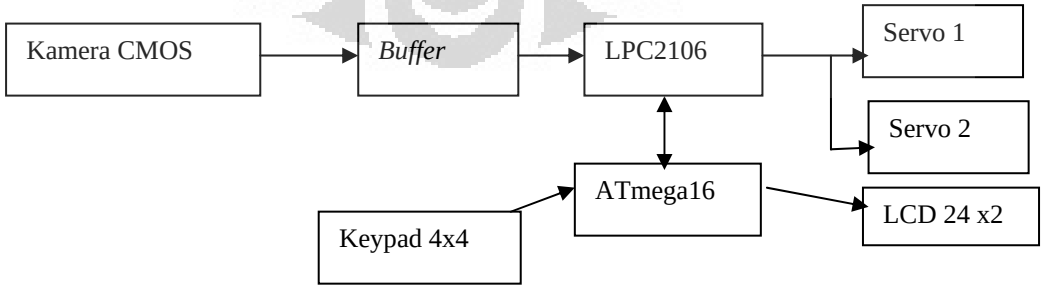
1.2 Tujuan Penelitian

Tujuan dari tugas akhir ini adalah dengan menggunakan sensor kamera, objek dengan warna yang ditentukan bisa dideteksi untuk kemudian diproses agar kamera selalu mengikuti pergerakan warna tersebut. Berdasarkan operasi tersebut, maka akan diuji kemampuan mikrokontroler mampu melakukannya secara *stand-alone*.

1.3 Pembatasan Masalah

Penulis membatasi masalah tugas akhir ini pada pembuatan alat mencakup:

a. Menggunakan perangkat lunak *vision system* CMUcam3.
b. Pemrograman mikrokontroler tambahan (Atmega16) sebagai *user-interface*.
c. Pemrograman dan pengujian sistem berbasis GUI (LabVIEW) pada PC.
d. Jenis sensor kamera yang digunakan adalah CMOS *Image Sensor* OV6620 dengan resolusi maksimal 352 x 288.
e. Servo motor yang digunakan untuk gerakan vertikal dan horizontal adalah *standard servo* 90˚.



Gambar 1.1 Blok fungsi kerja rangkaian sistem

1.4 Metode Penelitian

Metode penelitian yang akan dilakukan terdiri atas beberapa tahap antara lain:

1. Studi Literatur

   Metode Studi Literatur ini digunakan penulis untuk memperoleh teori-teori dasar sebagai sumber dan acuan dalam penulisan skripsi. Informasi dan pustaka yang berkaitan dengan masalah ini diperoleh dari literatur, penjelasan yang diberikan dosen pembimbing, rekan-rekan mahasiswa, internet, *datasheet* dan buku-buku yang berhubungan dengan tugas akhir penulis.

2. Perancangan Alat

   Perancangan alat merupakan tahap awal penulis untuk mencoba, memahami, menerapkan dan menggabungkan semua literatur yang telah diperoleh dan dipelajari untuk melengkapi sistem yang sedang dikembangkan, sehingga untuk selanjutnya penulis dapat merealisasikan sistem sesuai dengan tujuan.

   Perencanaan dan pembuatan sistem, meliputi :

   - Pemahaman cara kerja minsis mikrokontroler 32-bit ARM LPC2106.
   - Perencanaan algoritma dan *software color-tracking* dan pengendali servo, antarmuka keypad dan LCD serta serial *command* untuk *interface* ke *second system* (Atmega16).
   - Perencanaan algoritma dan *software* LabVIEW.
   - Perancangan perangkat keras sistem secara keseluruhan.

3. Pengujian Sistem.

   Pengujian sistem ini berkaitan dengan pengujian alat serta pengambilan data dari alat yang telah dibuat. Pengujian ini dilakukan untuk mengetahui karakteristik dari masing-masing bagian sistem, sehingga dapat diketahui bagaimana kinerja setiap bagian dan sejauh mana tingkat berjalannya setiap bagian sistem untuk dibentuk satu kesatuan sistem.

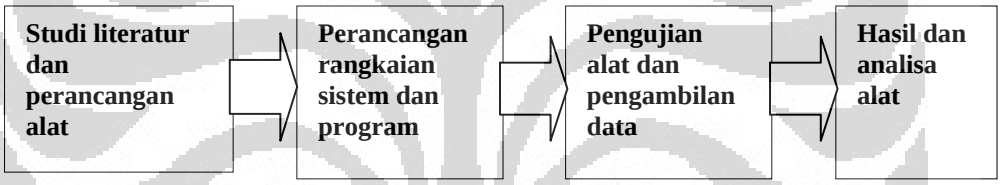
Pengujian terhadap hasil penelitian, meliputi:

- Pengujian titik pusat distribusi warna yang di-*track*. Serta respon *tracking*, relasi jarak dan sudutnya jangkauannya.
- Pengujian bagian servo. Pengujian pergerakan servo dengan memberikan input untuk mengetahui posisi netral dari masing-masing servo melalui GUI LabView pada PC.

4. Metode Analisis.

Metode Analisis data dilakukan dengan mengambil data respon waktu terhadap suatu koordinat pusat obyek berwarna. Setelah itu dilakukan analisa sehingga dapat ditarik kesimpulan dan saran-saran untuk pengembangan lebih lanjut.

Berikut ini adalah diagram langkah-langkah yang yang akan dilakukan dalam penelitian ini:



Gambar 1.2 Diagram langkah-langkah penelitian

1.5 Sistematika Penulisan

Sistematika penulisan skripsi ini terdiri dari bab-bab yang memuat beberapa sub bab dan sub-sub bab. Untuk memudahkan pembacaan dan pemahaman maka penulisan skripsi ini ini terdiri atas 5 bab dan secara garis besar dapat diuraikan sebagai berikut :

BAB I PENDAHULUAN

Pendahuluan berisi latar belakang, permasalahan, batasan masalah, tujuan penulisan, metode penulisan dan sistematika penulisan dari skripsi ini.

BAB II TEORI PENUNJANG

Teori dasar berisi landasan teori sebagai hasil dari studi literatur yang berhubungan dengan perancangan dan perakitan alat (*hardware*) serta pembuatan program (*software*).

BAB III PERANGKAT KERAS DAN PERANGKAT LUNAK

Pada bab ini akan dijelaskan sistem kerja dari semua perangkat keras (*hardware*) dan program (*software*) yang terlibat.

BAB IV HASIL DAN PEMBAHASAN

Bab ini menjelaskan tentang unjuk kerja alat sebagai hasil dari perancangan sistem. Pengujian akhir ini dilakukan dengan menyatukan seluruh bagian dari sistem sehingga dapat diketahui apakah sistem dapat berfungsi dengan baik. Setelah sistem dapat bekerja dengan baik maka dilakukan pengambilan data untuk menentukan kapabilitas dari sistem yang dibangun.

BAB V KESIMPULAN DAN SARAN

Bab penutup ini berisi kesimpulan penulis yang diperoleh berdasarkan pengujian sistem dan pengambilan data selama penelitian berlangsung, selain itu penutup juga berisikan tentang saran-saran dari penulis untuk mendapatkan hasil yang lebih baik dalam pengembangan lebih lanjut dari penelitian ini baik dari perangkat keras (*hardware*) maupun perangkat lunak (*software*).

BAB 2
TEORI PENUNJANG

Color tracking adalah kemampuan untuk mengambil citra, mengisolasi warna tertentu dan mengeluarkan informasi tentang area lokasi dari citra yang memuat warna tersebut. Dalam hal memilih warna, harus didefinisikan nilai minimum dan maksimum yang diperbolehkan untuk masing-masing dari ketiga kanal warna. Setiap warna yang unik direpresentasikan oleh nilai merah, hijau, dan biru yang mengindikasikan seberapa banyak masing-masing kanal tercampur kedalam warna akhir.

2.1 Model Citra Digital

Citra merupakan suatu fungsi intensitas dalam bidang dua dimensi. Karena intensitas yang dimaksudkan berasal dari sumber cahaya, dan cahaya adalah suatu bentuk energi, maka berlaku keadaan dimana fungsi intensitas diantara: $0 \langle f(x,y) \langle \infty$.

Pada dasarnya, citra yang dilihat terdiri atas berkas-berkas cahaya yang dipantulkan oleh benda-benda sekitarnya. Jadi secara alamiah, fungsi intensitas cahaya merupakan fungsi sumber cahaya yang menerangi obyek, serta jumlah cahaya yang dipantulkan oleh obyek, atau ditulis:

$$f(x,y) = i(x,y).r(x,y) \qquad (2.1)$$

yaitu: $0 \langle i(x,y) \langle \infty$ (iluminasi sumber cahaya)

$0 \langle r(x,y) \langle 1$ (koefisien pantul obyek)

Fungsi intensitas *f* pada suatu titik *(x,y)* disebut derajat keabuan atau *gray level (l),* dengan *l* terletak antara $L_{\min} \le 1 \le L_{\max}$. Dengan demikian:

$$L_{\min} = i_{\min}.r_{\min}$$

$$L_{\max} = i_{\max}.r_{\max} \qquad (\text{dalam } \textit{foot-candles}) \qquad (2.2)$$

Selang (L_{min}, L_{max}) sering disebut sebagai skala keabuan. Pada representasi suatu citra hitam putih secara numerik, biasanya selang digeser menjadi: *(0,L)* dengan 0 menyatakan hitam dan *L* menyatakan putih. Semua bilangan yang terletak diantara 0 dan *L* merupakan derajat keabuan.

2.1.1 Pencuplikan dan Kuantisasi

Suatu citra agar dapat direpresentasikan secara numerik, maka citra harus didigitalisasi, baik terhadap ruang koordinat *(x,y)* maupun terhadap skala keabuan *f(x,y).* Proses digitalisasi koordinat *(x,y)* dikenal sebagai "pencuplikan citra" (*image sampling*), sedangkan proses digitalisasi skala keabuan *f(x,y)* disebut sebagai "kuantisasi derajat keabuan" (*gray scale quantization*).

Sebuah citra kontinu *f(x,y)* akan didekati oleh cuplikan-cuplikan yang seragam jaraknya dalam bentuk matriks *N x N*. Nilai elemen-elemen matriks menyatakan derajat keabuan citra, sedangkan posisi elemen tersebut (dalam baris dan kolom) menyatakan koordinat titik-titik *(x,y)* dari citra.

$$f(x,y) = \begin{pmatrix} f(0,0) & f(0,1) & \cdots & f(0,N-1) \\ f(1,0) & f(1,1) & \cdots & f(1,N-1) \\ \vdots & \vdots & & \vdots \\ f(M-1,0) & f(M-1,1) & \cdots & f(M-1,N-1) \end{pmatrix} \quad (2.3)$$

Bentuk matriks ini dikenal sebagai suatu citra digital

Dengan alasan untuk memudahkan implementasi, dalam praktek sebagian besar diambil jumlah pencuplikan pada baris (*M*) dan kolom (*N*) sebagai bilangan pangkat dua dengan jarak cuplikan yang seragam. Jadi diambil:

$$M = 2^n, N = 2^n \quad ; n = \text{bilangan bulat positif} \quad (2.4)$$

Dengan alasan serupa maka lazimnya skala keabuan (0,L) dibagi kedalam G selang dengan panjang selang yang sama:

$$G = 2^m \quad ; m = \text{bilangan bulat positif} \quad (2.5)$$

Bila hal ini diterapkan, maka penyimpanan sebuah citra digital membutuhkan sejumlah *B* bit data

$$B = M \, x \, N \, x \, m \quad (2.6)$$

Makin tinggi nilai *M, N* dan *m*, maka citra kontinu *f(x,y)* akan makin didekati oleh citra digital yang dihasilkan. (Marvin)

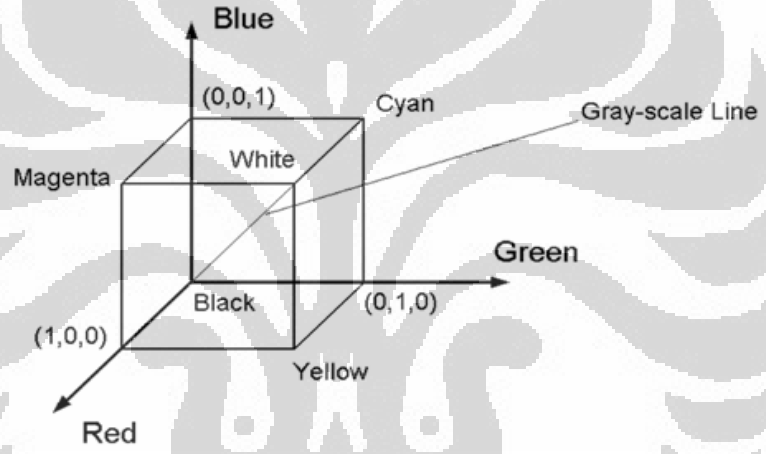
2.2 Warna

Piksel dari citra bitmap biasa terbagi menjadi 3 bagian : merah, hijau, and biru (RGB). Masing-masing komponen ini ter-kode dalam satu byte, 0 berarti bahwa piksel mempunyai nilai minimum, 255 berarti bahwa piksel tersebut

mempunyai nilai maksimumnya, sehingga (0,0,0) merepresentasikan piksel warna hitam sedangkan (255, 255, 255) merepresentasikan warna putih.

Umumnya komputer menampilkan 8-, 16-, atau 24-bit untuk setiap piksel. Banyaknya bit tiap piksel menentukan banyak warna berbeda yang dapat ditampilkan. Dengan 8 bit untuk setiap komponen warna, akan ada sebanyak 256 level untuk setiap komponen warna merah, hijau dan biru. Dari warna-warna tersebut ada 256 level skala keabuan.

Gambar 2.1 menunjukkan ruang warna RGB (*Red, Green, Blue*) dengan 3 sumbu yang mewakili masing-masing kanal. Total dari penjumlahan ketiga warna adalah putih (menggunakan intensitas maksimum). Nilai skala keabuan ditunjukkan pada garis yang dimulai dari hitam ke putih.



Gambar 2.1 Ruang warna RGB.

Sumber : Thomas Klinger.(2003). *Image Processing with LabVIEW™ and IMAQ™ Vision*. New Jersey: Prentice Hall PTR: 60

Intensitas maksimum dari sebuah warna direpresentasikan dengan angka 1, sehingga nilai (1,1,1) adalah warna putih.

Untuk mengkonversi warna kedalam skala keabuan, dapat menggunakan persamaan berikut :

$$GS = 0.333R + 0.333G + 0.333B \qquad (2.7)$$

GS = *Gray-Scale*.

2.3 Koordinat Pusat Massa

Pusat massa dari suatu sistem partikel merupakan titik tertentu dimana massa sistem tersebut terkonsentrasi. Pusat massa adalah fungsi yang hanya berisi

posisi dan massa dari partikel yang menyusun sistem. Dalam kasus benda tegar, posisi dari pusat massa adalah tetap. Dan dalam hubungannya dengan medan gravitasi, pusat massa seringkali disebut pusat gravitasi suatu massa.

Pusat massa dari suatu sistem R didefinisikan sebagai rata-rata posisi partikel, $\mathbf{r}_i$, yang diberatkan pada massanya, m_i dengan relasi:

$$R = \frac{\sum m_i \mathbf{r}_i}{\sum m_i} \tag{2.8}$$

Untuk distribusi yang kontinyu dengan densitas massa $\rho(\mathbf{r})$dan total massa M, jumlah tersebut menjadi suatu integral

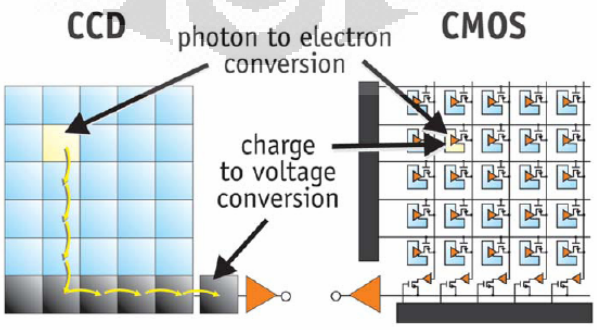
$$R = \frac{1}{M}\int \mathbf{r} dm = \frac{1}{M}\int \rho(\mathbf{r})\mathbf{r} dV = \frac{\int \rho(\mathbf{r})\mathbf{r} dV}{\int \rho(\mathbf{r}) dV} \tag{2.9}$$

Jika objek memiliki densitas yang seragam, maka pusat massa merupakan titik tengah (*centroid*) dari bentuknya.

2.4 Kamera CMOS OV6620

Sensor kamera CMOS adalah sensor *image* yang dibuat berdasarkan teknologi CMOS (*Complementary Metal Oxide Semiconductor*). Sebagaimana CMOS pada chip memori, chip ini terdapat ruang baris dan kolom alamat dekoder. Sensor ini menggunakan konversi *photon-to-electron-to-voltage*, yang diterapkan pada setiap pikselnya. Tiap piksel yang melakukan konversi tersebut disebut *photo-site* atau *photoelement*.

Untuk mendeskripsikan sensor CMOS, akan dibandingkan dengan tipe lain dari sensor kamera diantaranya CCD (*Charged Coupled Device*).



Gambar 2.2 Konversi foton ke tegangan pada CCD dan CMOS.

Sumber : Photonic Spectra. *CCD vsCMOS*: 2

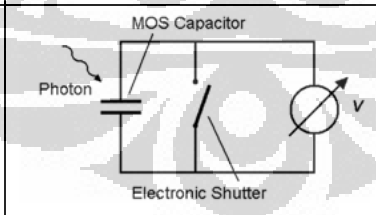
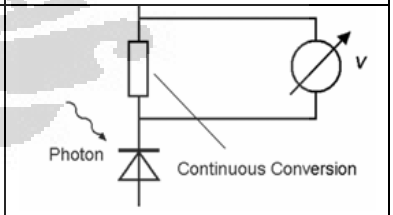
Keuntungan yang didapat menggunakan sensor CMOS ini diantaranya:

- Memungkinkan untuk mengakses tidak hanya gambar keseluruhan, tetapi juga elemen tunggal didalamnya dan beberapa kumpulan piksel, atau ROI (*Region Of Interest*), yang juga dibutuhkan dalam proses dan analisa citra. ROI bisa diatur dengan cara mengalamatkan berdasarkan baris dan kolom.
- Beberapa komponen tambahan seperti *clocking* dan *timing*, diintegrasikan langsung dalam satu chip dengan sensor sehingga membuat harganya semakin murah dan *reliable.*
- Mempunyai efek *blooming* yang alami dan tahan terhadap *smear*.

Satu perbedaan lainnya adalah deskripsi dari kecepatan pencuplikan citra. CCD dideskripsikan dalam istilah *image per second* atau *frame per second*, yang tidak dipakai saat mengakses satu piksel atau ROI. Pada CMOS istilah kecepatan adalah piksel *clock* dalam satuan MHz.

Kekurangan dari CMOS adalah *noise* yang dihasilkan. Konversi yang kontinyu dari *photocurrent*, memberikan efek *noise* terhadap sinyal keluaran. Alasan utamanya adalah pada proses produksi yang memang tidak didesain untuk *photoelement*-nya, sehingga menghasilkan sensitivitas yang buruk. Disisi lain, semakin besar penguatan yang dipakai, semakin besar pula level *noise*-nya.

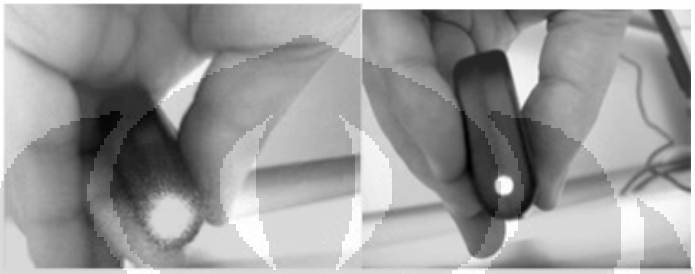
Tabel 2.1 Perbedaan sensor CCD dan CMOS

	CCD	CMOS
Teknologi		
Akses piksel	Gambar penuh	Satu piksel atau ROI
Daya	Relatif lebih tinggi	Relatif rendah
Konversi	Setelah terisi penuh	Setiap piksel
Integrasi	Hanya chip sensor	Beberapa komponen
Noise	*Fixed Pattern Noise*	*Noise* pada masukan
Bloom	Teknik *antiblooming*	Tahan terhadap *blooming*

Smear	Disebabkan oleh transfer kapasitif	Tidak ada

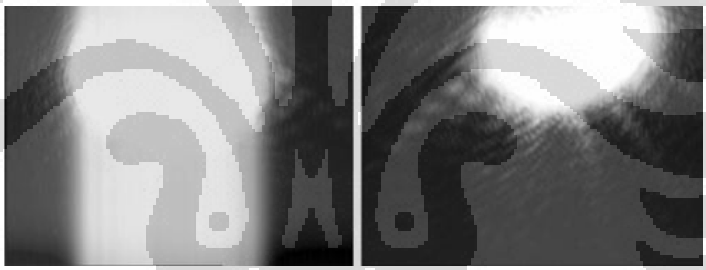
Sumber : Thomas Klinger.(2003). *Image Processing with LabVIEW™ and IMAQ™ Vision*. New Jersey: Prentice Hall PTR: 53 "*telah diolah kembali"*

Efek *blooming* atau kelebihan (*overflow*) elektron sehingga mengalir ke piksel tetangga tidak terjadi pada CMOS karena proses konversi elektron ke tegangan terjadi pada masing-masing sel.



Gambar 2.3 Efek "*blooming*" pada CCD dan CMOS

Sumber : Thomas Klinger.(2003). *Image Processing with LabVIEW™ and IMAQ™ Vision*. New Jersey: Prentice Hall PTR: 53



Gambar 2.4 Efek "*smear*" pada CCD dan CMOS

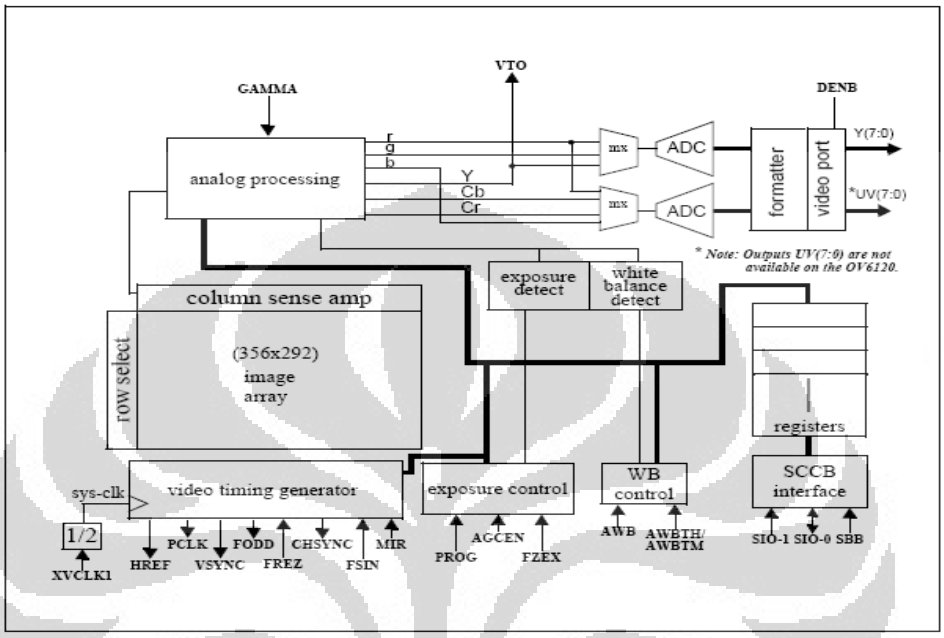
Sumber : Thomas Klinger.(2003). *Image Processing with LabVIEW™ and IMAQ™ Vision*. New Jersey: Prentice Hall PTR: 53

Sensor citra yang digunakan adalah OV6620. Sensor ini mampu beroperasi hingga mencapai 60 *frame per second* serta diterapkan algoritma untuk menghilangkan *Fixed Pattern Noise* (FPN). Semua fungsi kamera termasuk kontrol *exposure*, *gamma*, *gain*, *white balance*, *color matrix*, *windowing*, dan seterusnya, dapat diprogram melalui antarmuka I2C. Alat ini dapat diprogram untuk menghasilkan keluaran 8-bit atau 16-bit. Kamera juga mampu mengeluarkan sinyal analog yang monokrom berstandar PAL yang bisa dihubungkan ke TV.

Format yang digunakan pada penelitian ini adalah keluaran 8-bit RGB atau YCrCb dari sensornya dengan resolusi yang dapat diatur CIF atau QCIF.

Sinkronisasi sinyal *clock* piksel digunakan untuk membaca keluaran data dan mengindikasikan *frame-frame* baru perbaris secara horizontal (*rows-by-rows*).

2.4.1 Deskripsi Fungsi Sensor



Gambar 2.5 Blok diagram OV6620

Sumber : OmniVision. *Datasheet OV6620*: 4

Citra ditangkap pada susunan piksel yang berukuran 356 x 292 dan terhubung pada bagian proses analog dimana sebagian besar proses sinyal terjadi. Blok ini berisi rangkaian yang melakukan separasi warna, matrixing, *Automatic Gain Control* (AGC), koreksi *gamma*, koreksi warna, penyeimbang warna, kalibrasi *black-level*, penghalusan "*knee*", koreksi celah dan kontrol luminasi, kromasi dan filter *anti-alias* citra. Sinyal-sinyal analog video berdasarkan pada rumus untuk format YCrCb:

$$Y = 0{,}59\ G + 0{,}31R + 0{,}11B \quad (2.10)$$

$$Cr = 0{,}713 \times (R - Y) \quad (2.11)$$

$$Cb = 0{,}546 \times (B - Y) \quad (2.12)$$

Data mentah YCrCb/RGB dari bagian proses analog diumpankan ke 2 chip ADC 8-bit: satu untuk kanal Y/RG dan satu lagi terbagi pada kanal CrCb/BG. Data dari ADC kemudian dikondisikan menjadi digital. Sinyal yang telah diproses diteruskan ke port video digital melalui multiplekser yang

terhubung pada pin keluaran yang dapat dipilih oleh pengguna sebagai data video 16-, 8-, atau 4-bit.

Chip ADC 8-bit terintegrasi beroperasi hingga mencapai 9 MHz, tersinkron pada *pixel rate*. *Frame rate* diatur sebagai fungsi laju konversi yang sebenarnya. Kalibrasi konversi *black-level* dihubungkan dengan ketentuan:

- *black level* dari Y/RGB dinormalisasi ke nilai 16
- puncak *white level* terbatas pada 240
- *black level* CrCb adalah 128
- puncak/bawah adalah 240/16
- range keluaran data mentah RGB adalah 16/240

(catatan: nilai 0 dan 255 dipakai untuk sinkronisasi flag)

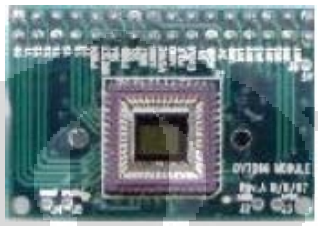
Sensor kamera OV6620 terintegrasi dalam modul C3088. Jika pada sensor kamera terdapat 48 pin, maka pada modul ini hanya digunakan 32 pin yang masing-masing fungsinya dideskripsikan pada Tabel 2.2. *Output* dapat ditampilkan dalam format-format yang berbeda, dengan tipe kanal yang berbeda (RGB, YUV). Salah satunya yang digunakan pada penelitian ini adalah *Video Format.*

Tabel 2.2 Pin modul kamera C3088.

Nomor pin	Nama pin	Deskripsi
1~8	Y0~Y7	*Output digital Y bus*
9	PWDN	*Power down mode*
10	RST	*Reset*
11	SDA	*I2C serial data*
12	FODD	*Odd Field flag*
13	SCL	*I2C serial input clock*
14	HREF	*Horizontal window reference output*
15	AGND	*Analog Ground*
16	VSYN	*Vertical Sync Output*
17	AGND	*Analog Ground*
18	PCLK	*Pixel clock output*
19	EXCLK	*External clock input*

20	VCC	*Power Supply 5VDC*
21	AGND	*Analog Ground*
22	VCC	*Power Supply 5VDC*
23~30	UV0~UV7	*UV bus*
31	GND	*Ground*
32	VTO	*Video Analog Output*

Sumber : COMedia. *Datasheet C3088*:1



Gambar 2.6 C3088

Sumber : COMedia. *Datasheet C3088*:1

2.4.2 *Pixel Mapping*

Peta piksel sensor OV6620 tersusun dalam grid, masing-masing lokasi mampu mendeteksi warna: merah, hijau dan biru. Berikut ini adalah *layout* dari 4 baris yang pertama:

Baris1: B(1,1) G(1,2) B(1,3) G(1,4) B(1,5) G(1,6)….B(1,355) G(1,356)

Baris2: G(2,1) R(2,2) G(2,3) R(2,4) G(2,5) R(2,6)….G(2,355) R(2,356)

Baris3: B(3,1) G(3,2) B(3,3) G(3,4) B(3,5) G(3,6)….B(3,355) G(3,356)

Baris4: G(4,1) R(4,2) G(4,3) R(4,4) G(4,5) R(4,6)….G(4,355) R(4,356)

Layout diatas disebut data mentah citra RGB (RGB *raw data*). Pada mode 8-bit data kamera diperoleh dari dua baris sensor sekaligus untuk menghasilkan keluaran tiap *line* yang keluar pada Y bus, sedangkan UV bus tidak aktif. Modul kamera mengambil data dari dua baris sensor sekaligus hingga menghasilkan keluaran tiap barisnya:

Baris1: B(1,1) G(2,1) R(2,2) G(1,2) B(1,3) G(2,3) R(2,4) G(1,4)…

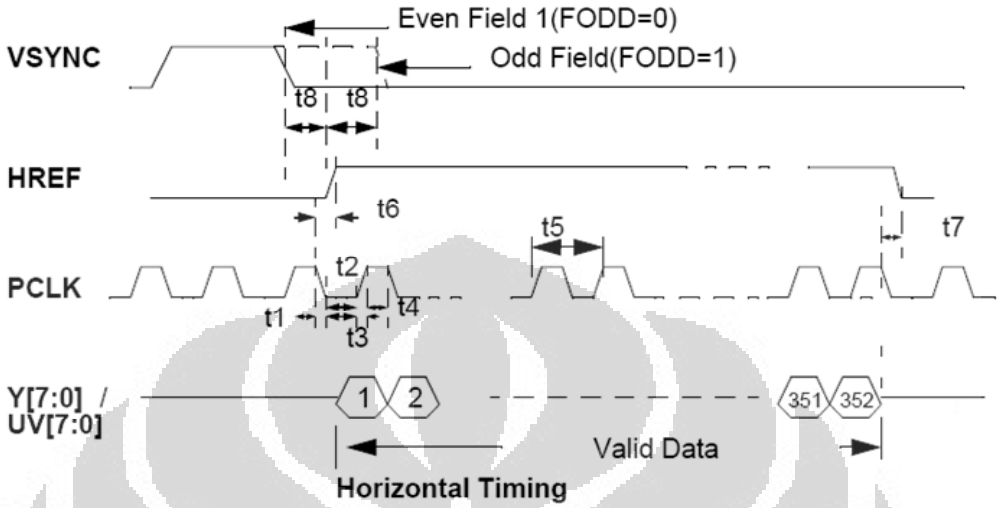
Baris2: B(3,1) G(2,1) R(2,2) G(3,2) B(3,3) G(2,3) R(2,4) G(3,4)…

Sistem yang dibangun kemudian akan mengambil data ini dan mengeluarkan data piksel sebagai berikut:

Baris1:[R(2,2):G(1,1):B(1,1)][R(2,4):G(1,4):B(1,3)]…

Baris2:[R(2,2):G(3,2):B(3,1)][R(2,4):G(3,4):B(3,3)]…

Data citra mentah dikirim secara kontinyu dengan sinkronisasi sinyal pada pin HREF, VSYNC, dan PCLK dengan *timing* seperti yang ditunjukkan pada Gambar 2.7.



Gambar 2.7 *Zoom video port timing*

Sumber : OmniVision. *Datasheet OV6620*: 6

Sinyal VSYNC menandakan adanya *frame* baru, sinyal HREF menandakan bahwa informasinya valid kemudian melakukan sinkronisasi horizontal, dan PCLK adalah *clock* dari piksel yang menunjukkan data valid yaitu pada transisi tinggi. Periode dari PCLK dapat diubah dengan menuliskan register pada kamera.

2.5 Mikrokontroler ARM

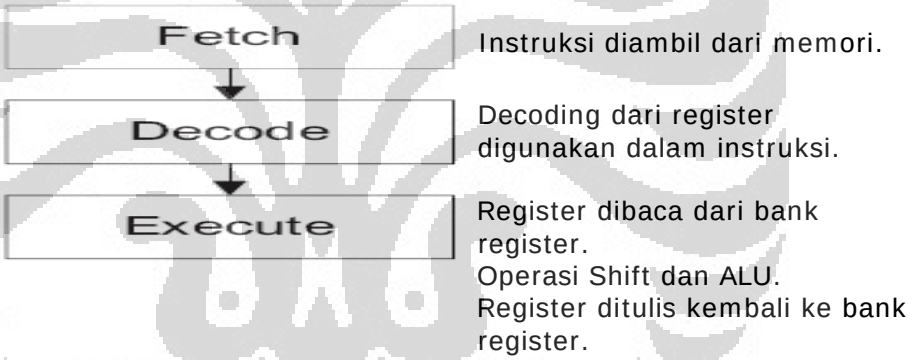
Para pendesain ARM mempunyai perjalanan panjang semenjak adanya ARM1 di tahun 1985. Lebih dari 1 milyar prosesor ARM telah tersebar hingga tahun 2001. Hal yang mendasari keberhasilan ARM ada pada desain yang sederhana dan powerful.

Prosesor ARM tidak hanya satu jenis saja, tetapi sebagian besar desainnya memberikan prinsip desain dan set instruksi yang umum digunakan. Prosesor ARM yang paling sukses adalah ARM7TDMI. Dengan *Dhrystone* hingga 120 MIPS dikenal dengan densitas kodenya yang tinggi, serta konsumsi daya yang rendah.

Arsitektur ARM berdasarkan pada prisip *Reduced Instruction Set Computer* (RISC). Set instruksi RISC sendiri adalah mekanisme yang lebih mudah dari CISC. Kemudahan yang diberikan:

- Instruksi untuk menghasilkan keluaran lebih cepat
- Respon *real-time interrupt* yang tinggi
- Prosesor makrosel yang kecil dan murah

ARM7TDMI menggunakan *pipeline* untuk meningkatkan kecepatan instruksi prosesor. Ini mengakibatkan beberapa operasi mampu berjalan secara simultan, dan proses sistem memori bisa bekerja terus-menerus. *Pipeline* adalah suatu set elemen proses data yang dihubungkan secara seri, sehingga keluaran dari satu elemen adalah masukan dari elemen yang lain. Elemen-elemen dari pipeline biasanya dieksekusi secara paralel atau dalam mode *time-sliced*, pada kasus ini, beberapa jumlah penyimpanan *buffer* dimasukkan diantara elemen.



Gambar 2.8 Instruksi *pipeline*

Selama operasi normal, saat satu instruksi sedang dikerjakan, suksesornya di-*decode*, dan instruksi ketiga diambil dari memori.

ARM7TDMI mengimplementasi arsitektur ARMv4T yang mendukung 2 set instruksi:

- 32-bit ARM
- 16-bit Thumb

2.5.1 Akses Memori

Pada ARM7TDMI, arsitektur yang digunakan adalah *Von Neumann*, dengan data bus 32-bit tunggal membawa data dan instruksi sekaligus. Hanya

instruksi untuk memuat, menyimpan dan menukar yang dapat mengakses data dari memori.

Data bisa berupa:

- 8-bit (*bytes*)
- 16-bit (*halfwords*)
- 32-bit (*words*)

Words harus dijajarkan ke 4-*byte*. *Halfwords* harus dijajarkan ke 2-*byte*.

2.5.2 Antarmuka Memori

Antarmuka prosesor ARM7TDMI membuat potensi performa bisa terwujud selama meminimalisasi penggunaan memori

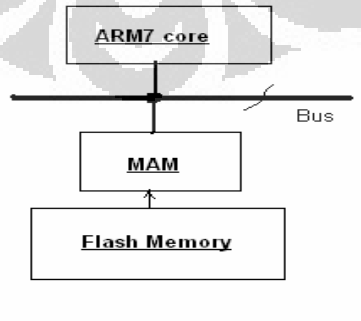
ARM7TDMI memiliki 4 tipe dasar siklus memori :

- Siklus *idle*
- Siklus *nonsequential*
- Siklus *sequential*
- Siklus *transfer register coprosesor*

2.5.3 Modul Pemercepat Memori

Biasanya program pada ARM ditempatkan pada memori *flash,* memori ini hanya mampu bekerja hanya maksimal 20MHz, sementara kecepatan ARM sendiri 60MHz.

Keluarga ARM7 memiliki modul *cache* yang disebut Memory acceleration module (MAM)



Gambar 2.9 Skema MAM

Sumber:http://winarm.scienceprog.com/arm-mcu-types/lpc2000-mcu-memory-acceleration-module.html

MAM pada dasarnya bekerja dengan cara mengisi 4 instruksi ARM pada waktu yang sama dari *flash* (instruksi 8 *Thumb*). Jika ARM berjalan pada 60MHz maka ada 3 siklus akses ke flash dibutuhkan. MAM mengisi instruksi-instruksi ini pada satu siklus. Modul MAM di-*disable* setelah MCU reset. Ada 3 mode kerja dari MAM:

- MAM OFF- semua instruksi dibaca langsung dari memori flash
- *Partially enables* MAM – instruksi berantai diakses menggunakan MAM sementara percabangan dan konstanta diakses langsung dari flash
- *Fully enables* MAM – semua memori diambil langsung dari MAM.

MAM diatur oleh 2 register – *timing*(*MAMTIM*) dan kontrol(*MAMCR*).

2.5.4 Mikrokontroler LPC2106

Mikrokontroler LPC2106 adalah mikrokontroler buatan Philips (NXP) berbasis CPU 16/32-bit ARM7TDMI. Mikrokontroler mendukung emulasi *real-time* dan modul *trace*, dengan 128 kB *flash* memori berkecepatan tinggi. Antarmuka memori selebar 128-bit dan arsitektur pemercepat yang unik menjadikan eksekusi kode 32-bit dilakukan pada rata-rata *clock* yang maksimum. LPC2106 adalah prosesor yang mampu mengendalikan skala frekuensi dengan program dan memiliki modul pemercepat (MAM) yang bisa mengambil data dari *flash* mendekati satu siklus. *Bootloader* yang tertanam membuat proses download file *hex* melalui port serial tanpa perangkat pemrogram luar yang lain.

2.5.5 Konfigurasi Pin LPC2106

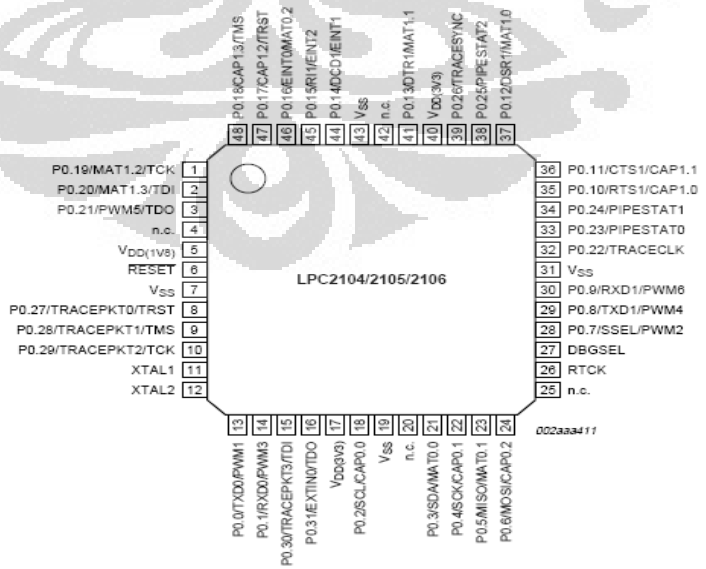
Deskripsi fungsi masing-masing pin dapat dilihat pada Tabel 2.3 dan konfigurasi pin LPC2106 dapat dilihat pada Gambar 2.10:

Tabel 2.3 Deskripsi global fungsi pin LPC2106

Nama Pin	No kaki	Tipe	Deskripsi
RTCK	26	I/O	*Returned Test Clock output*. Sinyal tambahan untuk port JTAG. Pin dua arah dengan *pull-up* internal.
DBGSEL	27	I	*Debug Select*. Saat *low*, beroperasi normalsaat *high* mode *debug* aktif.

			Merupakan pin *input* dengan *pull-up internal*.
/RST	6	I	*External Reset input*. Ketika *low* piranti *reset* semua pin pada keadaan *default* eksekusi prosesor dimulai dari alamat 0
X1	11	I	*Input* rangkaian osilator juga sebagai *internal clock generator*
X2	12	O	Keluaran dari *amplifier* osilator
Vss	7,19,31,43	I	*Ground*
Vdd 1,8	5	I	*Power Supply* CPU
Vdd 3,3	17, 40	I	*Power Supply* I/O
NC	4, 20, 25, 42	-	*Not Connected*
Port0.0 ~ port0.31	1,2,3,8,9,10,13,14,15, 16,18,21,22,23,24,28, 29,30,32,33,34,35,36, 37,38,39,41,44,45,46, 47,48	I/O	Port 0 merupakan I/O port dua arah dengan pengarah individual setiap bit. Operasi I/O diatur oleh *Pin Connect Block.* Penggunaan pin sebagai I/O atau *peripheral*.

Sumber : NXP. *User Manual LPC2106*: 73



Gambar 2.10 *Pinout* LPC2106

Sumber : NXP. *Datasheet LPC2104/2105/2106*: 2

2.5.6 Arsitektur LPC2106

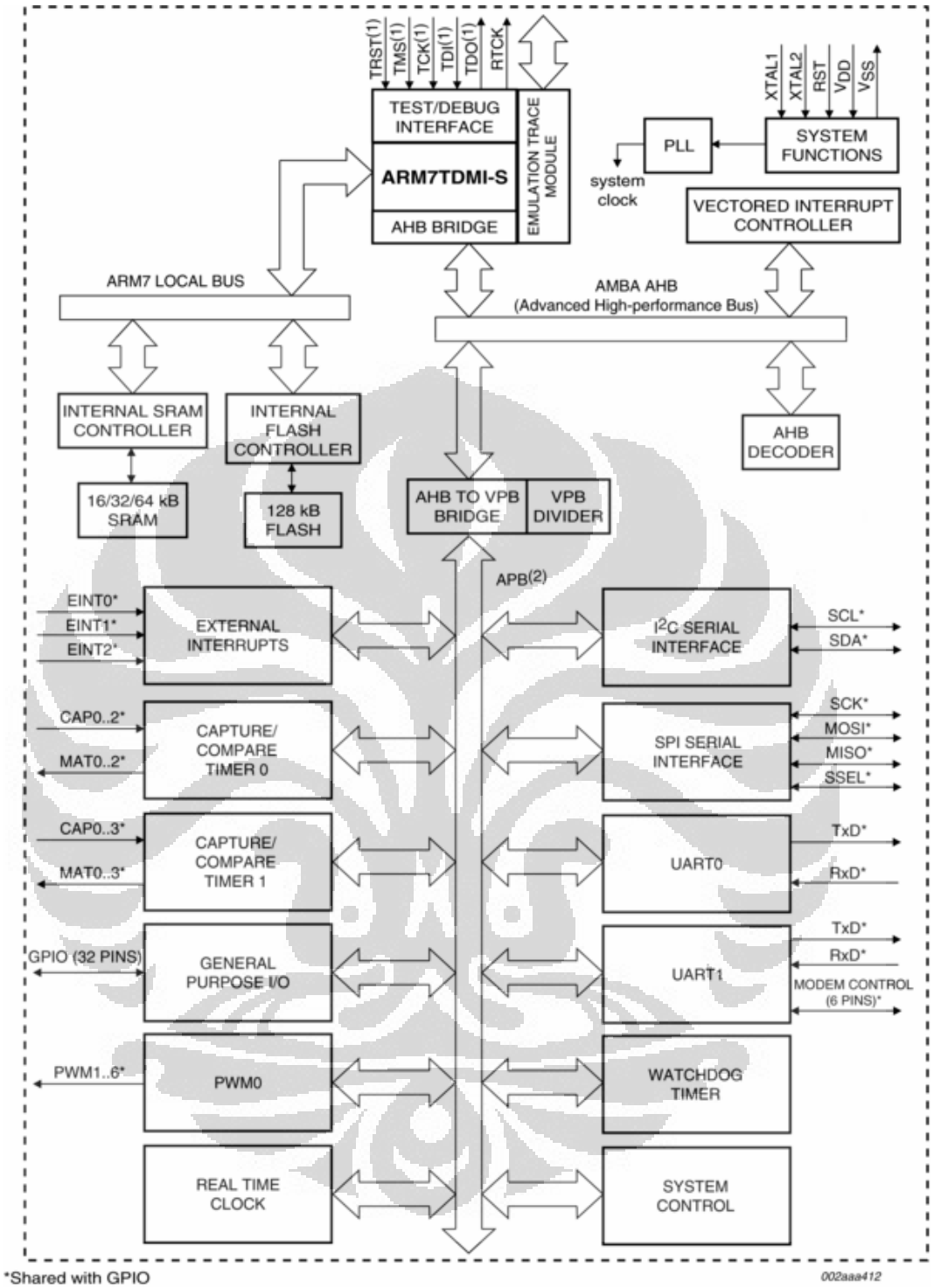
LPC2106 terdiri atas CPU ARM7TDMI-S dengan dukungan emulasi yang merupakan bus lokal ARM7 untuk antarmuka ke kontrol memori, *Advanced High-performance Bus*(AHB) untuk antarmuka kontrol interupt, dan *VLSI Peripheral Bus* (VPB) yang kompatibel dengan *Advanced Peripheral Bus* untuk ARM, sebagai penghubung fungsi-fungsi perangkat *on-chip*. LPC2106 mengkonfigurasi prosesor ARM7TDMI-S dengan mode *little endian byte*.

Perangkat AHB dialokasikan pada jangkauan pengalamatan 2 megabyte pada puncak atas ruang memori 4 gigabyte ARM. Masing-masing AHB dialokasikan pada ruang memori 16kilobyte pada ruang alamat AHB. Perangkat fungsi LPC2106 (selain interupt) terhubung pada bus VPB. Jembatan AHB dan VPB adalah penghubung bus AHB dan bus VPB. Perangkat VPB juga dialokasikan pada jangkauan 2 megabyte pengalamatan, dimulai pada titik 3,5 gigabyte. Masing-masing perangkat VPB dialokasikan pada ruang memori 16kilobyte pada ruang alamat VPB.

Penghubung perangkat *on-chip* dengan pin diatur oleh *Pin Connect Block* yang dikonfigurasi secara *software*. (LPC2106 Datasheet)

2.5.7 Fitur LPC2106

- Prosesor 16/32-bit ARM7TDMI
- 64kB on-chip RAM stastik
- 128 kB *on-chip* memori program. Antarmuka dan pemercepat 128-bit menjadikan operasi kecepatan tinggi hingga 60 MHz.
- *In System Programming* (ISP) dan *In-Application Programming* (IAP) melalui software *on-chip bootloader*. *Programming flash* membutuhkan 1 ms per 512B line. *Single sector* atau menghapus seluruh chip membutuhkan 400 ms.
- *Vectored Interrupt Controller* dengan prioritas yang bisa dikonfigurasikan dan dialamatkan.
- 2 buah timer 32-bit, PWM unit, *Watchdog*, dan RTC.
- *On-chip crystal oscillator* 1-30 MHz.



Gambar 2.11 Arsitektur mikrokontroler LPC2106.

Sumber:http://www.nxp.com/#/pip/pip=[pip=LPC2104_2105_2106_7]|pp=[t=pip,i=LPC2104_2105_2106_7]

2.6 FIFO *Frame Buffer* AL440B

Frame buffer AL440B adalah memori jenis FIFO (*First In First Out*) dengan spesifikasi kerja berkecepatan 50 MHz, kapasitas 1 MB video FIFO. Pada penelitian ini video FIFO digunakan karena dapat menjadikan kamera beroperasi dengan kecepatan penuh dan menjadi penghubung *clock* dari kamera ke CPU. Untuk menjalankan proses banyak *frame*, FIFO bisa melakukan setting ulang *read pointer*. Berikut ini adalah pin-pin yang digunakan:

Tabel 2.4 Jalur penulisan data ke *frame buffer*

Nama pin	Nomor pin	Tipe	Deskripsi
DI[7:0]	9,8,7,6,4,3,2,1	Input	DI adalah input data 8-bit, disinkronisasi oleh clock WCK. Data diterima pada tepi transisi tinggi WCK.
WE	10	Input	Sinyal input yang mengatur operasi penulisan input 8-bit dan penulisan pointer
IE	11	Input	Input yang mengatur aktif tidaknya pin DI. Pointer alamat penulisan internal naik pada tepi kenaikan sinyal WCK dengan mengaktifkan WE tanpa memperhatikan level IE
WCK	13	Input	WCK adalah pin clock input. Data yang ditulis disinkronkan oleh clock ini
WRST	14	Input	WRST adalah sinyal reset input yang me-reset pointer alamat penulisan kembali ke 0
IRDY	15	Output	Status flag output yang melaporkan ruang penyimpanan FIFO yang diperbolehkan.

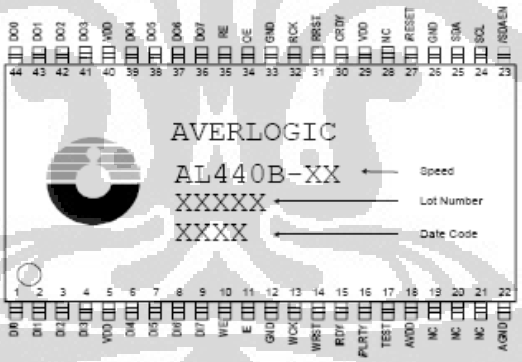
Sumber : Averlogic. *Datasheet AL440B*: 6

Tabel 2.5 Jalur pembacaan data dari *frame buffer*

Nama pin	Nomor pin	Tipe	Deskripsi
DO[7:0]	36, 37, 38, 39, 41, 42, 43, 44	Output	DO adalah output data 8-bit, disinkronisasi oleh clock RCK. Data diterima pada tepi transisi tinggi RCK.
RE	35	Input	Sinyal input yang mengatur operasi

			pembacaac input 8-bit dan pembacaan pointer
OE	34	Input	Input yang mengatur aktif tidaknya pin DO. Pointer alamat pembacaan internal naik pada tepi kenaikan sinyal RCK dengan mengaktifkan RE tanpa memperhatikan level OE
RCK	32	Input	RCK adalah pin clock output. Data yang ditulis disinkronkan oleh clock ini
RRST	31	Input	RRST adalah sinyal reset output yang me-reset pointer alamat pembacaan kembali ke 0
ORDY	30	Output	Status flag output yang melaporkan ruang penyimpanan FIFO yang diperbolehkan.

Sumber : Averlogic. *Datasheet AL440B*: 6



Gambar 2.12 *Pinout* memori FIFO

Sumber : Averlogic. *Datasheet AL440B*: 5

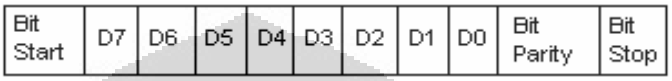
2.7 Komunikasi Serial

Komunikasi serial digunakan pada penelitian ini untuk mengatur dan mengetahui kinerja dari sistem baik melalui PC atau mikrokontroler Atmega16.

Pada komunikasi serial setiap bit dikirimkan satu per satu melalui saluran tunggal. Dalam komunikasi serial terdapat tiga macam mode transmisi, yaitu; sinkron, asinkron dan isinkron. Dalam proyek akhir ini penulis hanya

menggunakan mode transmisi asinkron, sehingga pembahasan hanya sebatas pada *mode* transmisi asinkron saja.

Sinkronisasi yang dilakukan pada transmisi serial asinkron adalah dengan memberikan bit-bit penanda awal dan akhir dari data pada sisi pengirim maupun pada sisi penerima. Bit-bit serial asinkron terdiri atas 1 start bit (selalu *Low*/rendah/0), bit data (karakter), 1 bit paritas, dan 1 atau 2 bit stop (selalu *High*/tinggi/1) (Gambar 2.13).



Gambar 2.13 Bentuk format pegiriman data serial asinkron

Faktor lain yang cukup penting dalam transfer data serial asinkron adalah kecepatan pengiriman. Besaran kecepatan pengiriman data serial ialah bps (*bit per second*), dan biasa disebut *baud rate* atau cps (*character per second*). *Baud rate* yang biasa digunakan adalah 9600, 19200, 38400, dan 115200.

Pada umumnya, setiap peralatan yang dihubungkan dengan menggunakan port serial harus mengubah kembali transmisi serial menjadi seperti bentuk data paralel sebelum digunakan. Proses tersebut dilakukan oleh UART (*Universal Asynchronous Receiver Transmitter*). Disamping itu, komunikasi serial memiliki banyak kelebihan, yaitu:

a. Kabel yang digunakan untuk komunikasi dapat lebih panjang. Port serial mengirimkan level "0" sebagai –3 s/d –25 volt dan level "1" sebagai +3 s/d +25 volt. Dengan kata lain, port serial memiliki *swing* maksimum 50 volts. Sedangkan, port parallel memiliki *swing* maksimum sekitar 5 volt. Hal ini menyebabkan kehilangan tegangan yang disebabkan oleh panjang kabel memiliki pengaruh yang lebih kecil pada komunikasi serial.
b. Kabel yang digunakan lebih sedikit. Komunikasi serial memiliki *null modem*, sehingga hanya memerlukan 3 buah kabel, sedangkan pada komunikasi paralel memerlukan minimal 8 buah kabel.

Peralatan yang menggunakan komunikasi serial dapat berupa DCE (*Data Communication Equipment*) atau DTE (*Data Terminal Eqiupment*). DCE dapat berupa modem, plotter, printer, dan lain-lain. Sedangkan DTE seperti PC dan terminal.

2.7.1 IC MAX232

MAX232 memerlukan 4 kapasitor external 1uF yang masing-masing kaki kapasitor terhubung dengan pin-pin yang telah ditentukan. Kapasitor-kapasitor tersebut digunakan oleh internal *charge pump* untuk menghasilkan nilai tegangan +10 volts dan -10 volts.

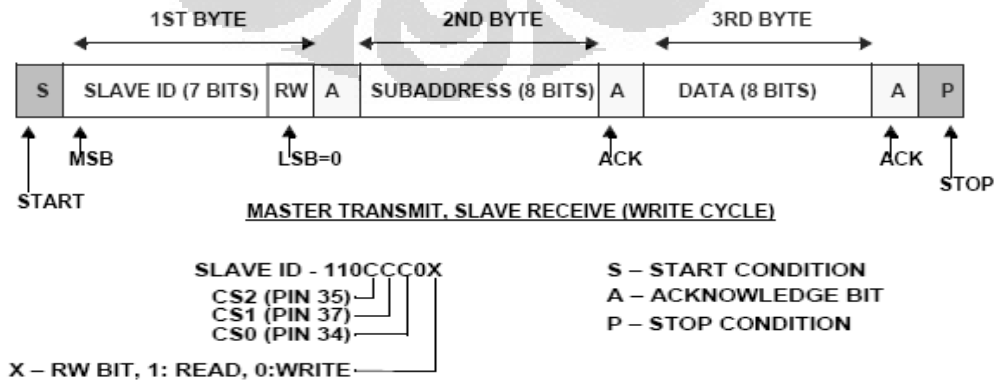
Sebagai suplay MAX232 membutuhkan tegangan sebesar 5 Volt pada pin 16 dan pin 15 sebagai groundnya. Kapasitor sebesar 1uF digunakan untuk mem-*bypass* sinyal tegangan antara kedua pin. Konfigurasi pin pada IC MAX232 digambarkan pada Gambar 2.14. dibawah ini.

Gambar 2.14 *Pinout* IC MAX232

Sumber : Maxim. *Datasheet MAX232*: 2

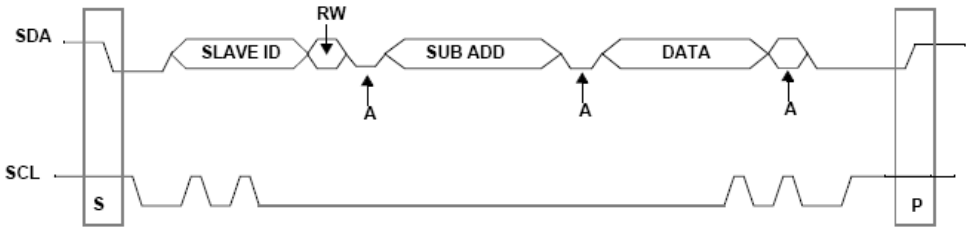
2.7.2 Komunikasi Serial I2C

Konfigurasi sensor kamera dilakukan melalui antarmuka SCCB (*Serial Control Camera Bus*) atau lebih dikenal sebagai I2C. Port I2C diaktifkan dengan menghubungkan pin 12 sensor (I2CB) melalui resistor *pull-up* 10K ohm dengan V_{DD}. Ketika terhubung maka sensor berlaku sebagai piranti *slave* yang mendukung transfer data 400 kbps menggunakan protokol alamat/data 7-bit.



Gambar 2.15 Format protokol I2C

Sumber : OmniVision. *Datasheet OV6620*: 21



Gambar 2.16 Kondisi *start/stop*.

Sumber : OmniVision. *Datasheet OV6620*: 21

Pada operasi I2C ini, mikrokontroler harus melakukan operasi berikut:

- Menghasilkan kondisi *start/stop*
- Menyediakan clock pada pin SCL
- Menempatkan 7-bit alamat *slave*, bit baca/tulis, dan 8-bit sub-alamat pada pin SDA.

Kondisi *start* terjadi saat transisi *low* pin SDA ketika SCL *high*. *Byte* berikutnya yang dikirim terdiri dari 7-bit ID alamat *slave* yang dikendalikan oleh pin CS0:CS2. Pada modul C3088 pin secara *default* menuliskan 3-bit ini bernilai ”000”, sehingga alamat penulisan adalah 0xC0. Kemudian 1-bit sisanya menentukan operasi tulis/baca.

Tabel 2.6 Alamat ID *slave* OV6620

CS[2:0]	000	001	010	011	100	101	110	111
WRITE ID (hex)	C0	C4	C8	CC	D0	D4	D8	DC
READ ID (hex)	C1	C5	C9	CD	D1	D5	D9	DD

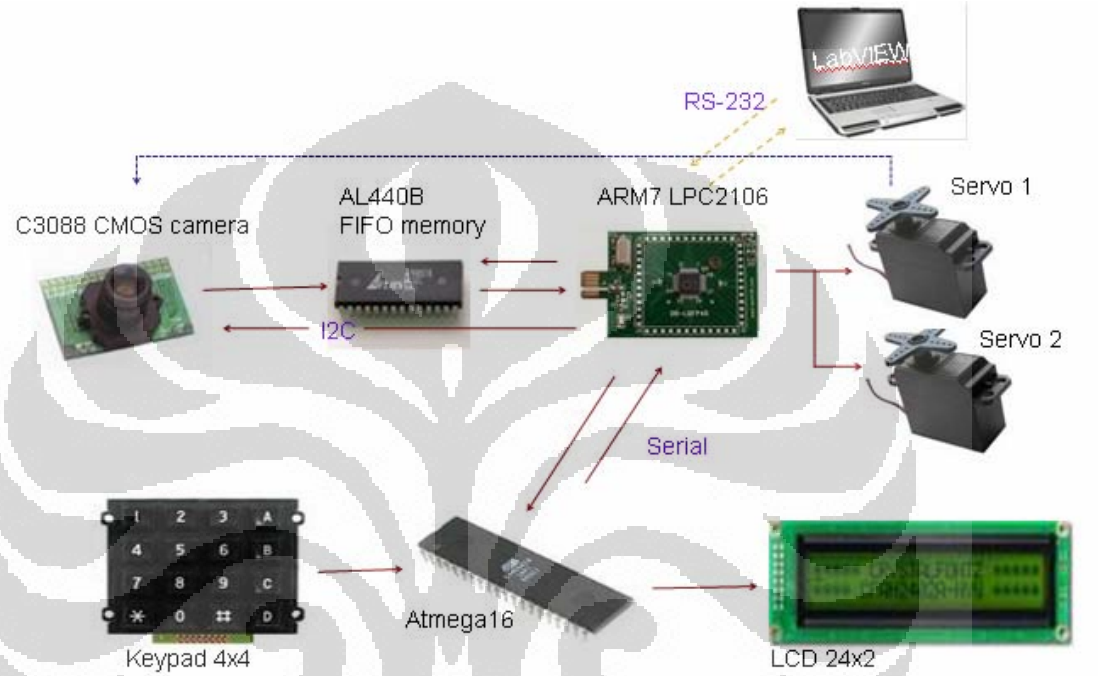
Sumber : OmniVision. *Datasheet OV6620*: 23

Selama operasi tulis, sensor mengirimkan ”*acknowledgement*”. *Byte* yang dikirim selanjutnya adalah sub-alamat register yang direspon ”*acknowledgement*” oleh sensor. *Byte* ketiga yang dikirim oleh mikrokontroler adalah data yang akan dituliskan pada alamat register. Register sub-alamat dan data yang dikirim menentukan keluaran sensor kamera, dan konfigurasi ini dapat dilihat pada lembar data sensor OV6620. Kondisi selesai terjadi saat sinyal *stop* dikirim oleh mikrokontroler, yaitu pada saat terjadi transisi *high* pin SDA sementara SCL sedang *high*. (OV6620 Datasheet

BAB 3
PERANGKAT KERAS DAN PERANGKAT LUNAK

3.1 Perancangan Perangkat Keras

Secara umum perangkat keras yang menyusun sistem *tracking* warna ini dapat ditampilkan sebagai berikut:



Gambar 3.1 Diagram blok sistem *tracking* warna.

Sistem dibagi menjadi 2 bagian, sistem utama dimana proses citra dan pengendalian servo berlangsung dan sistem pendukung sebagai *user-interface* untuk memberikan *input* parameter warna yang akan di-*track*. Komunikasi antara 2 sistem ini terjadi melalui komunikasi serial.

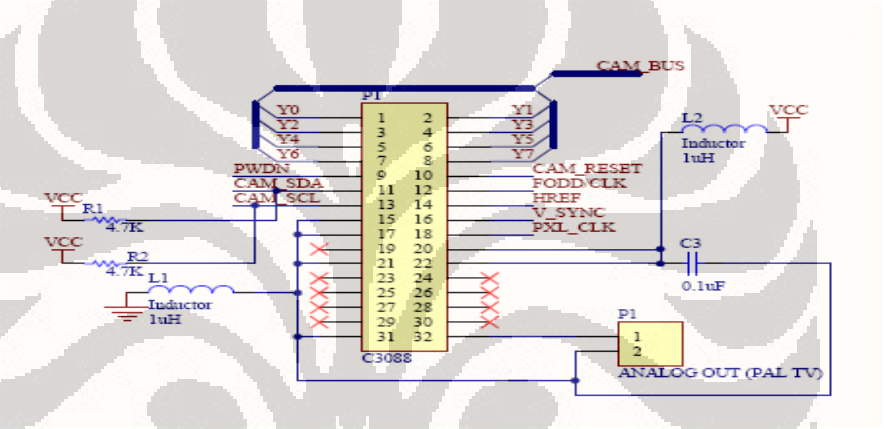
Data citra digital dihasilkan oleh sensor, kemudian diumpankan ke *buffer* sebelum diproses oleh mikrokontroler ARM. Mikrokontroler mengkonfigurasi sensor CMOS menggunakan protokol serial *two-wire*. Transfer sebuah citra dari kamera CMOS ke *frame buffer* di-inisialisasi oleh mikrokontroler. Untuk memasukkan *frame* baru ke dalam *frame buffer*, mikrokontroler harus menunggu terlebih dahulu data *frame* sebelumnya selesai ditransfer. Citra diambil oleh mikrokontroler setelah *frame buffer* terisi sedikitnya 2 blok memori *frame buffer* dengan melakukan *clocking* 8-bit sekaligus secara asinkron untuk mengeluarkan sebuah *frame* dari *buffer*. *Frame* yang terakhir menyulut interupsi mikrokontroler

untuk menghentikan jalur control penulisan dari *frame buffer* ke mikrokontroler hingga sampai pada saat data *frame* dibutuhkan lagi. Data *frame* ini diolah oleh mikrokontroler hingga menghasilkan masukan bagi servo untuk menggerakkan kamera.

Mikrokontroler AVR men-*scan* input *keypad*, setiap penekanan *keypad* akan ditampilkan oleh LCD, sehingga dapat terlihat parameter yang akan dikirimkan melalui serial.

3.1.1 Perancangan Antarmuka Modul C3088

Modul terhubung seperti pada gambar rangkaian berikut:

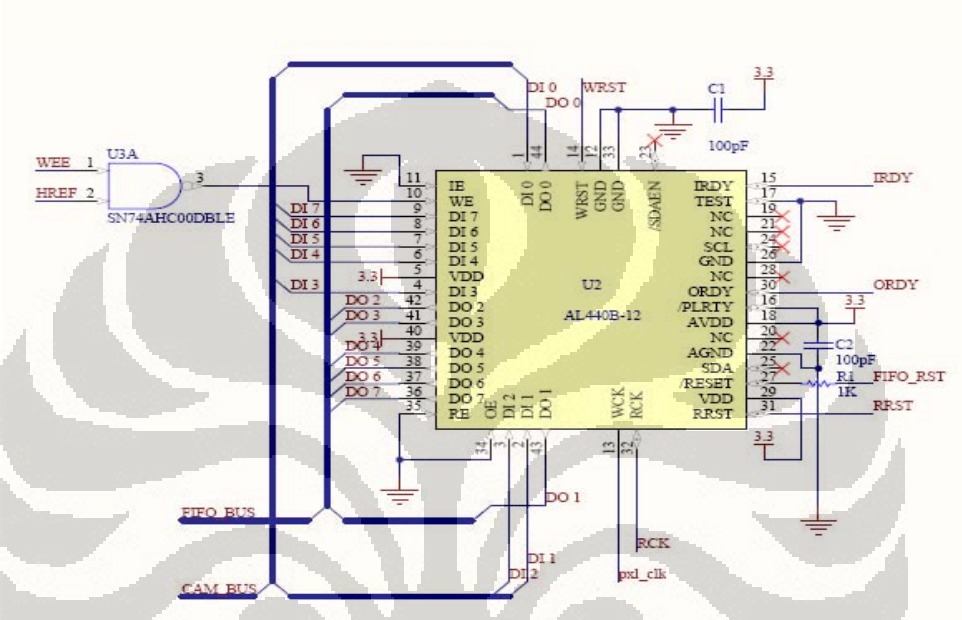


Gambar 3.2 Skema rangkaian antarmuka modul C3088

Pada rangkaian diatas, data citra kanal Y dikirim melalui CAM_BUS yang terhubung pada pin 1-8. Properti kamera diatur oleh mikrokontroler melalui komunikasi I2C (SDA,SCL) pada pin 11 dan 13. PWDN digunakan untuk meminimalisir daya yang digunakan saat kamera beroperasi. Jika PWDN bernilai ”0”, maka kamera beroperasi pada mode normal, sebaliknya jika PWDN bernilai ”1”, maka daya yang digunakan menjadi seminim mungkin. Ini dilakukan bila kamera hanya diperlukan dalam keadaan *standby* dengan tidak mengaktifkan beberapa fitur kamera. Namun dalam penelitian ini, fungsi PWDN tidak dimaksimalkan, karena penggunaan kamera hanya pada waktu yang relatif singkat.

3.1.2 Perancangan Antarmuka *Frame Buffer* AL440B

Pengoperasian *frame buffer* ini cukup sederhana, dengan mengatur jalur penulisan dan pembacaan data yang keluar atau masuk. *Frame buffer* diatur penggunaannya oleh mikrokontroler ARM secara paralel, yaitu dengan mengatur pin-pin kontrol masukan dan keluaran. *Buffer* terhubung pada rangkaian seperti pada Gambar 3.3 :



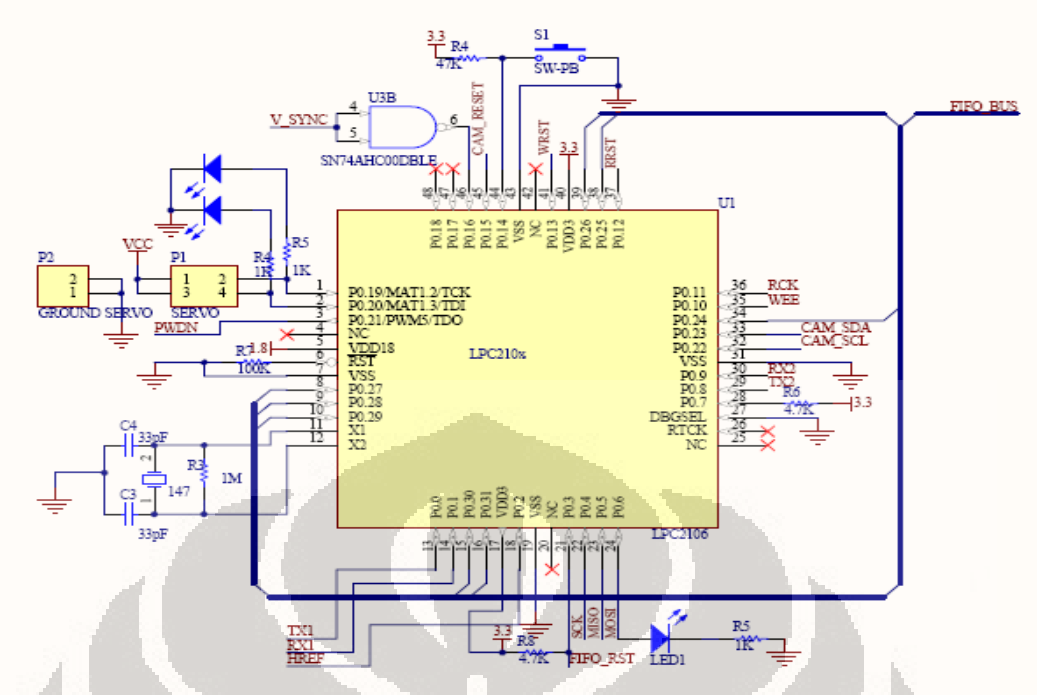
Gambar 3.3 Skema rangkaian antarmuka FIFO *buffer*.

Pada Gambar 3.3, pin DI terhubung pada CAM_BUS dan DO terhubung pada FIFO_BUS. Terdapat NAND yang terhubung dengan WEE pada mikrokontroler dan HREF pada kamera, fungsinya adalah bit tersebut bersama-sama mengatur penulisan *input*, jika HREF set (yang menandakan datanya valid), maka penulisan diatur oleh sinyal WEE. Sebaliknya jika WEE set, maka penulisan *input* dilakukan selama datanya valid. Jika keluaran NAND adalah ”1”, maka proses yang dilakukan adalah penulisan *pointer buffer*. Sehingga hasil akhirnya adalah tiap bit data yang valid tertulis pada *buffer*.

Pada proses pembacaan, proses dilakukan dengan cara terus-menerus dengan aturan sesuai FIFO, yaitu data yang masuk pertama kali adalah data yang keluar pertama kali.

3.1.3 Perancangan Antarmuka Mikrokontroler LPC2106

Antarmuka rangkaian dapat dilihat pada Gambar 3.4:



Gambar 3.4 Skema rangkaian antarmuka mikrokontroler LPC2106

Pada Gambar 3.4 terlihat bahwa mikrokontroler mengambil data hanya melalui buffer yaitu FIFO_BUS yang terhubung pada 8 pin I/O P0.24 - P0.31.Oleh karena itu ke-8 pin tersebut diatur sebagai pin *input*.

Komunikasi I2C digunakan untuk mengatur properti dari kamera. Pin IO hanya terdiri atas 1 port 32 buah, yang semuanya hampir digunakan pada penelitian ini. P0.0 dan P0.1 sebagai penghubung komunikasi serial, sementara P0.2 terhubung dengan HREF pada kamera yang juga terhubung pada HREF dan NAND pada rangkaian *buffer*, pin ini digunakan untuk mengaktifasikan *interrupt* pada mikrokontroler yang akan mengatur jalur penulisan pada FIFO. Jika pin tersebut aktif, maka akan mengaktifkan pin WEE sehingga menghasilkan respon seperti yang dibahas pada subbab sebelumnya.

P0.3 terhubung dengan FIFO_RST yang berfungsi untuk mengaktifkan pin RESET pada FIFO sehingga menghasilkan proses *reset buffer* secara global, yang berarti semua data akan dihapus (kosong). Pin ini di atur sebagai *output* pada mikrokontroler. Sebagai indikator proses digunakan LED yang terhubung pada pin P0.6. Indikator ini berguna untuk mengetahui jalannya program.

Pin 35 dan 36 masing-masing digunakan sebagai *output*. Pin 35 untuk mengatur penulisan *buffer* sebagaimana dijelaskan sebelumnya dan pin 36 sebagai

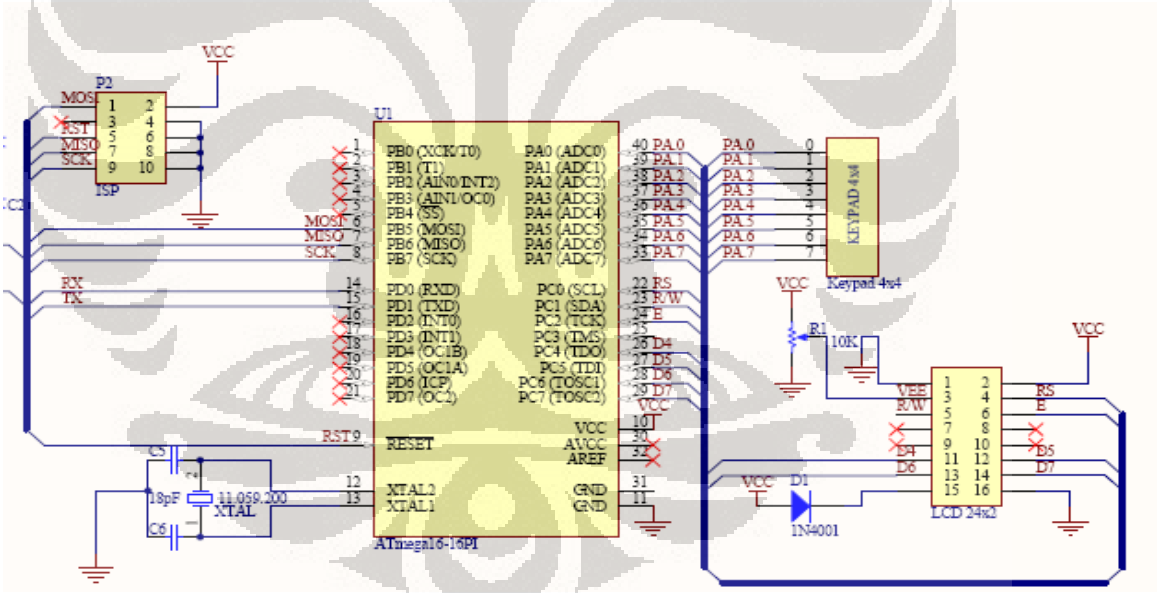
output yang berupa *clock* untuk RCK. Hasil keluaran pin ini adalah berupa sinyal *clock* yang akan digunakan untuk membaca data dari kamera ke memori..

Sebagai inisialisasi awal adanya *frame* baru, maka ditambahkan NAND yang lebih difungsikan sebagai *buffer* terhubung pada V_SYNC kamera dan pin P0.16. Pada mikrokontroler pin ini difungsikan sebagai interupt eksternal, jika ada transisi *low* pada pin ini maka eksternal *interrupt0* akan aktif.

3.1.4 Perancangan Sistem Atmega16

Sebelum proses *tracking* warna dilakukan, *input* dari parameter warna harus dimasukkan terlebih dahulu baik melalui PC atau perangkat lain yang mendukung komunikasi serial. Susunan rangkaian dapat dilihat seperti pada Gambar 3.5

Ada 3 fungsi utama yang digunakan pada mikrokontroler ini: antarmuka *keypad* 4x4, LCD 24x2 dan komunikasi serial.



Gambar 3.5 Skema rangkaian mikrokontroler Atmega16

Pada Gambar 3.5, proses pertama yang dilakukan mikrokontroler ini adalah *scanning keypad* yang terhubung seluruh pin-nya pada portA. MSB (pin 4 - 7) di-set sebagai *output*, sedangkan LSB (pin 0 - 3) di-set sebagai *input*. Untuk mengatur input, pengarah register diberi logika '0', dan sebaliknya untuk *output*. Proses *scanning* terjadi dengan memberikan nilai tertentu pada portA. Sebagai contoh, portA diberi nilai 0b01111111, ketika ada penekanan pada *keypad*,

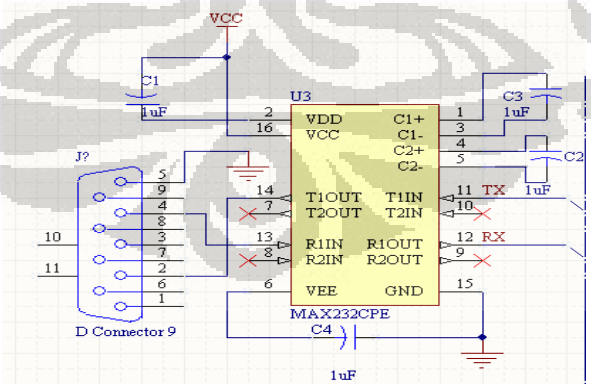
misalnya menekan angka ”1” pada *keypad* yang menghubungkan antara pin 0 dan pin 7, maka akan dicek apakah pin 0 ”*clear*”, jika ya definisikan suatu variabel ”1”, jika tidak dilanjutkan dengan pemberian nilai yang lain pada portA.

Sebelum data yang di-*scan* ditampilkan pada LCD, dibuat variabel *buffer* pada mikrokontroler untuk menyimpan data. Data terakhir yang jadi pembanding, mengindikasikan akhir dari penekanan adalah tombol ” * ”. Setelah tombol ini ditekan, maka data yang tersimpan pada variabel *buffer*, selain ditampilkan di LCD, juga dikirim melalui serial.

LCD yang digunakan adalah LCD dengan jumlah karakter 24x2, cukup panjang mengingat jumlah karakter parameter warna yang panjang harus ditampilkan. LCD terhubung pada port C, yaitu: *Enable* di port C.2; RS di port C.0; DB7 di port C.7; DB6 di port C.6; DB5 di port C.5; dan DB4 di port C.4.

3.1.5 Perancangan Komunikasi Serial RS232

IC MAX232 digunakan sebagai pengubah level tegangan TTL menjadi level tegangan RS-232 ataupun sebaliknya. MAX232 digunakan untuk menjalin komunikasi antara mikrokontroler Atmega16 ataupun LPC2106 dengan komputer maupun komunikasi antara mikrokontroler yang digunakan nantinya untuk pengujian komunikasi serial pada PC. Skema rangkaian IC MAX232 dapat dilihat pada Gambar 3.6:



Gambar 3.6 Skema rangkaian IC MAX232

Komunikasi serial dilakukan dengan pengaturan sebagai berikut untuk kedua perangkat maupun komputer:

- 115200 kbps baud-rate
- 8 data bits

- 1 stop bit
- No parity
- No flow control

3.1.6 Motor Servo

Servo motor yang digunakan adalah produksi dari Futaba tipe S3003. RC servo ini dapat dengan mudah diaplikasikan untuk berbagai kepentingan robotik sederhana. Di samping itu, motor servo ini, memiliki berat yang relatif ringan. Mekaniknya menggunakan *ball bearing* pada output-nya (sehingga gerakannya halus). Meskipun dimensinya kecil dan ringan, RC servo memiliki torsi yang cukup besar (3 kg/cm) dan memiliki kecepatan putaran yang cepat (untuk bergerak sejauh 60° hanya membutuhkan waktu 0,2 detik). Untuk mengoperasikan RC servo ini harus diberi tegangan sebesar 4,8V – 6V. Untuk mengontrol posisi motor harus diberikan pulsa yang berulang-ulang. Motor RC servo memiliki kemampuan mempertahankan posisi sudutnya (jika diberikan pulsa terus-menerus). Karena kemampuannya itu, motor servo sering juga disebut *position locking* motor. Pulsa yang diberikan memiliki logic high sebesar 0,5ms – 2,5ms dengan *duty cycle* 20ms, dimana pergerakan 0,5ms sampai 2,5ms ini setara dengan gerakan mulai dari posisi -45° hingga +45°, maka perubahan 1° diperlukan perubahan lebar *logic high* 22,2µs. Posisi 0° atau posisi *center* bisa didapat dengan memberikan lebar pulsa 1,5ms. Lebar pulsa sebesar 1,5ms itu harus selalu dijaga secara periodik dengan frekuensi sebesar 50Hz. Contoh perhitungan lebar *logic high* untuk mendapatkan posisi servo +45°, sebagai berikut:

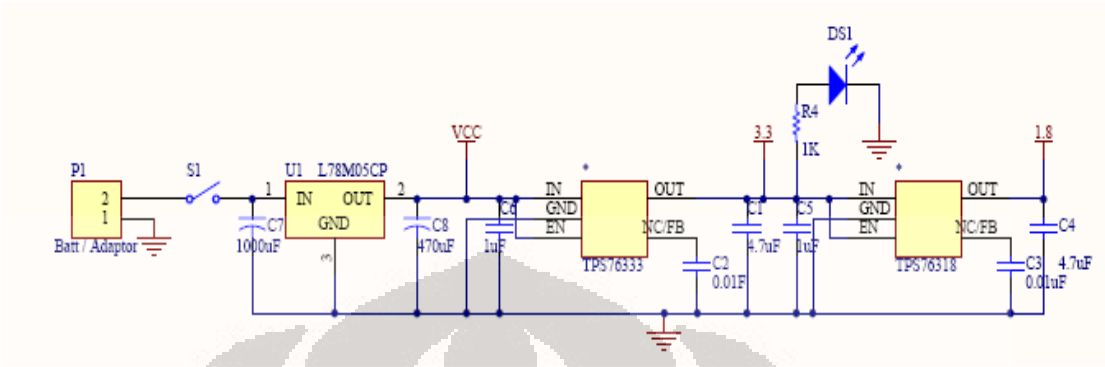
22,2 µs x 45 = 499,5µs (± = 1000 µs atau 1ms)

Karena +45° maka lebar pulsa high = 1,5 ms (0) + 1 ms = 2,5 ms (+45°). Parameter lain yang berbeda antara servo satu dengan servo lainnya adalah kecepatan servo untuk berubah dari posisi satu ke posisi lainnya.

3.1.7 Perancangan *Power Supply*

Ada 3 sumber *power supply* yang digunakan pada keseluruhan sistem, yaitu 1,8 volt, 3,3 volt, dan VCC 5 volt. Mikrokontroler LPC2106 menggunakan

tegangan kerja 1,8 volt dan 3,3 volt untuk *peripheral*nya, AL440B menggunakan tegangan kerja 3,3 volt dan perangkat lainnya menggunakan VCC 5 volt. Rangkaian *power supply* yang dibuat adalah sebagai berikut:



Gambar 3.7 Rangkaian *power supply*.

Regulator 3,3 volt dan 1,8 volt menggunakan IC yang sejenis yaitu TPS763*xx*, dimana *xx* menandakan tegangan keluaran. Ada 5 buah pin, IN, GND, EN, NC/FB,dan OUT. Antara pin OUT dan GND penggunaan kapasitor diperlukan untuk lebih menjaga stabilitas *internal loop* IC. Sedangkan antara IN dan GND, kapasitor digunakan untuk meningkatkan respon transien dan menghindari *noise*. Pin EN sendiri berfungsi untuk mengaktifkan *input* dari IC tersebut.

3.2 Perancangan Perangkat Lunak

Perangkat lunak yang dibuat terbagi menjadi 3 bagian, yaitu:

- ✓ Pemrograman pada mikrokontroler *user-interface* (ATMEGA16).
- ✓ Pemrograman pada mikrokontroler pemroses citra (LPC2106)
- ✓ Pemrograman pada PC untuk menguji fungsi-fungsi sistem (LabVIEW).

3.2.1 Perancangan Perangkat Lunak Atmega16

Untuk mendukung aplikasi sistem *tracking* warna yang *stand-alone*, maka perlu ditambahkan rangkaian lain untuk memberikan *input* sebagai parameter warna yang akan di-*track* pada sistem utama.

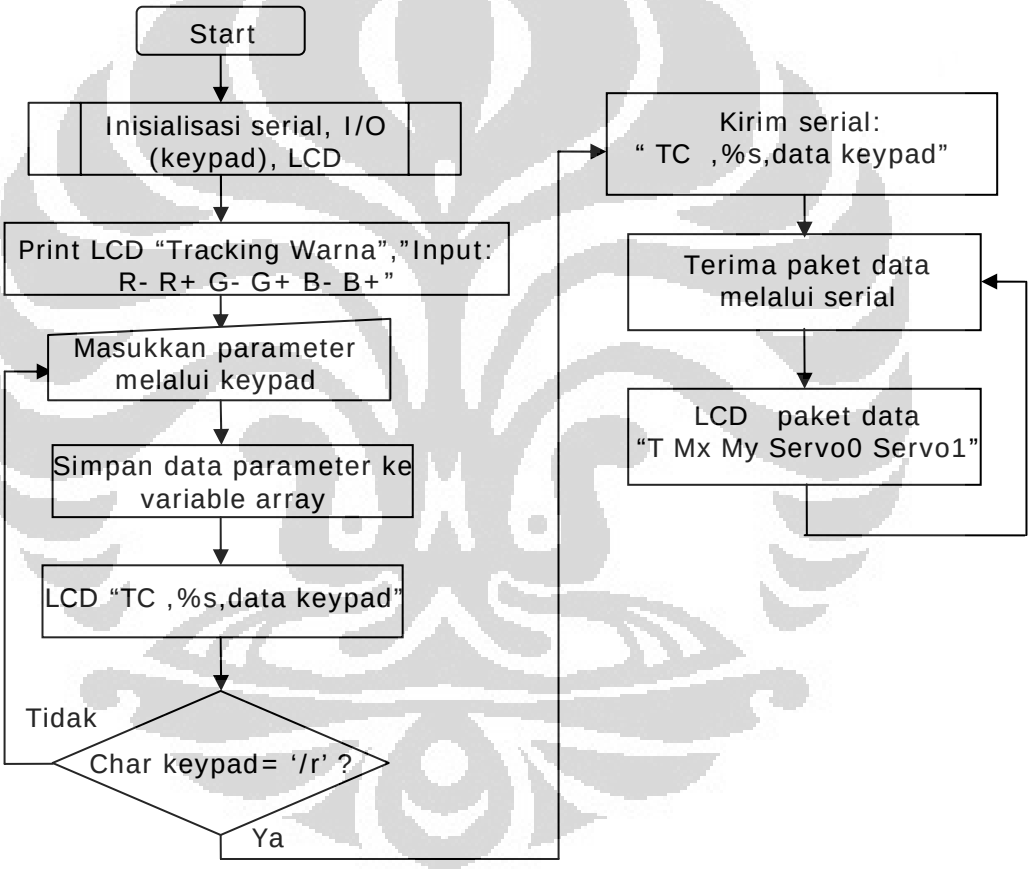
Dalam hal ini diperlukan 3 fungsi utama, yaitu : komunikasi serial, pengaturan *input/output* untuk antarmuka matriks *keypad* 4x4, dan LCD 24x2.

Pemrograman pada mikrokontroler ATMEGA16 dibuat dengan menggunakan *CodeVisionAVR V2.03.4.*

Program ini dibuat untuk mengeluarkan serial *command* untuk memanggil fungsi *tracking* pada mikrokontroler LPC2106. S*tring command* yang akan digunakan yaitu:

- TC *xxx xxx xxx xxx xxx xxx* : command ini yang akan menginisialisasi proses *tracking* warna berdasarkan pada nilai *xxx* yang sudah ditekan pada *keypad*.

Flowchart yang digunakan ditunjukkan pada Gambar 3.8:

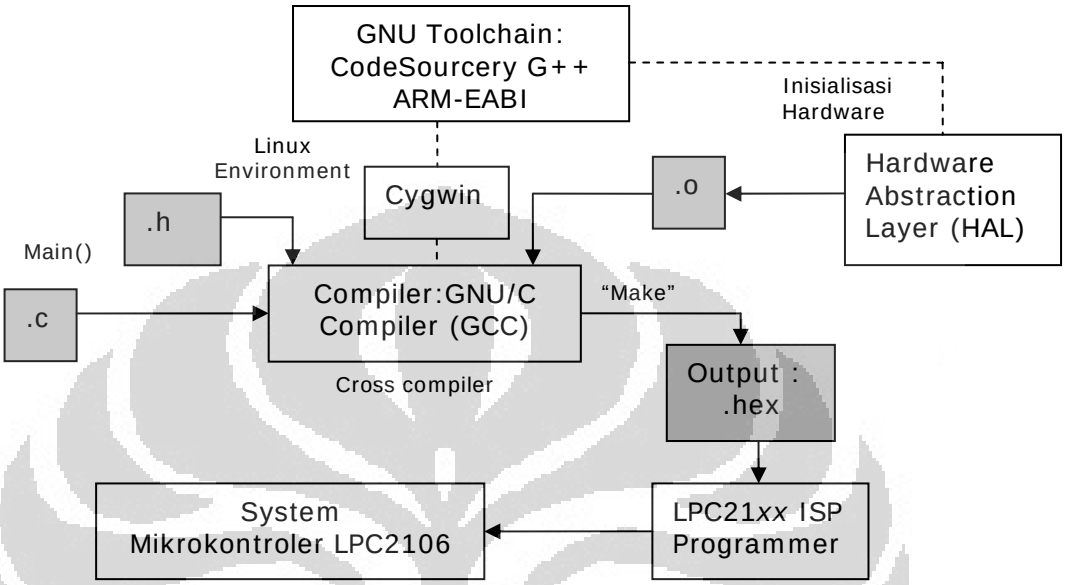


Gambar 3.8 *Flowchart* program Atmega16

3.2.2 Perancangan Perangkat Lunak LPC2106

Pemrograman pada mikrokontroler berbasis ARM dibuat menggunakan bahasa C dengan *platform* kode sistem *open-source* yang dikembangkan pada *platform* perangkat keras CMUcam3. Pembuatan kode dilakukan melalui program

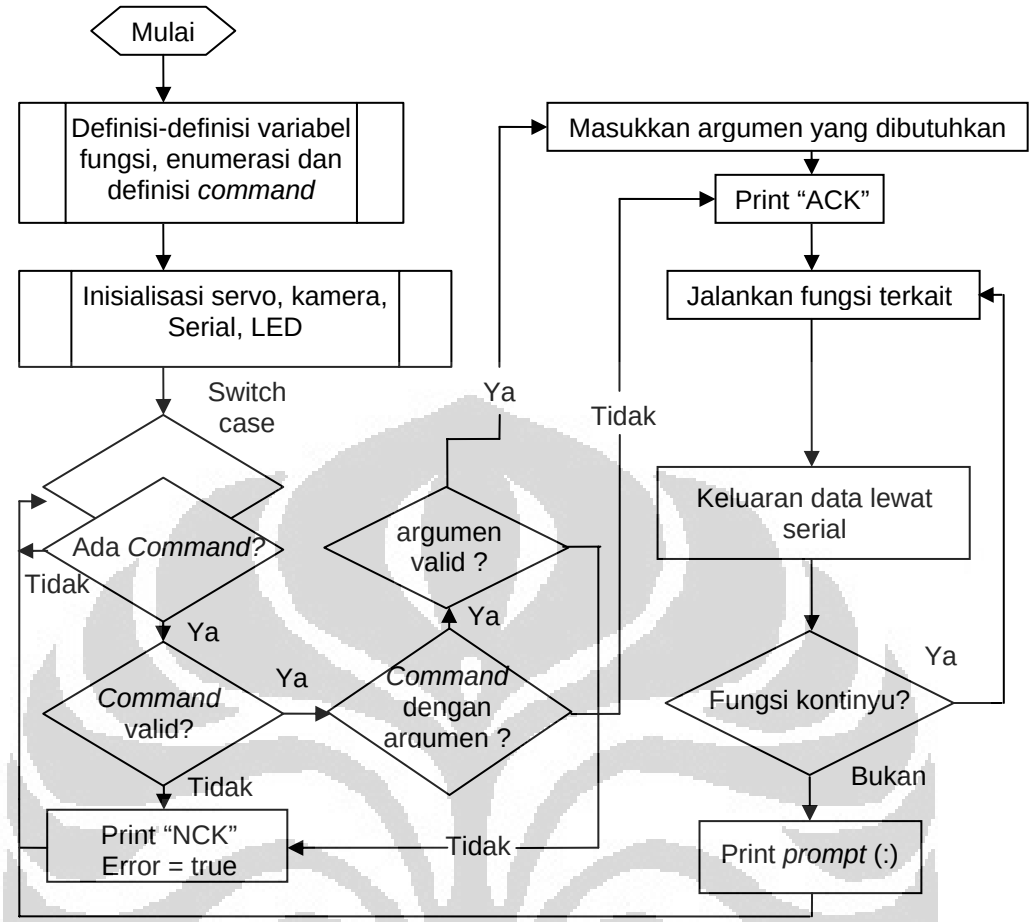
Programmer's Notepad. Setelah kode dibuat, kode di-*compile* menggunakan *cross-compiler* berbasis Linux *environment* pada Windows yaitu Cygwin dengan *compiler* berbasis GNU-ARM CodeSourcery GCC ARM EABI (*Embedded Application Binary Interface*).



Gambar 3.9 Diagram kompilasi pada mikrokontroler CMUcam3-LPC2106.

Pada Gambar 3.9, untuk meng-*compile* file *.c mula-mula dibuat *hardware abstraction layer (hal)* LPC2106 terlebih dahulu untuk menginisialisasi awal mikrokontroler, fitur apa saja yang akan digunakan serta definisi-definisi apa saja yang akan digunakan sehingga akan menghasilkan file *object*. Proses ini harus dilakukan sebelum program rutin main() dibuat. *Hal* yang dibuat mendefinisikan insialisasi I/O, I2C, serial, dan *timer*.

Setelah *hal* dibuat, selanjutnya adalah membuat program "*command*" pada mikrokontroler dengan *flowchart* pada Gambar 3.10:

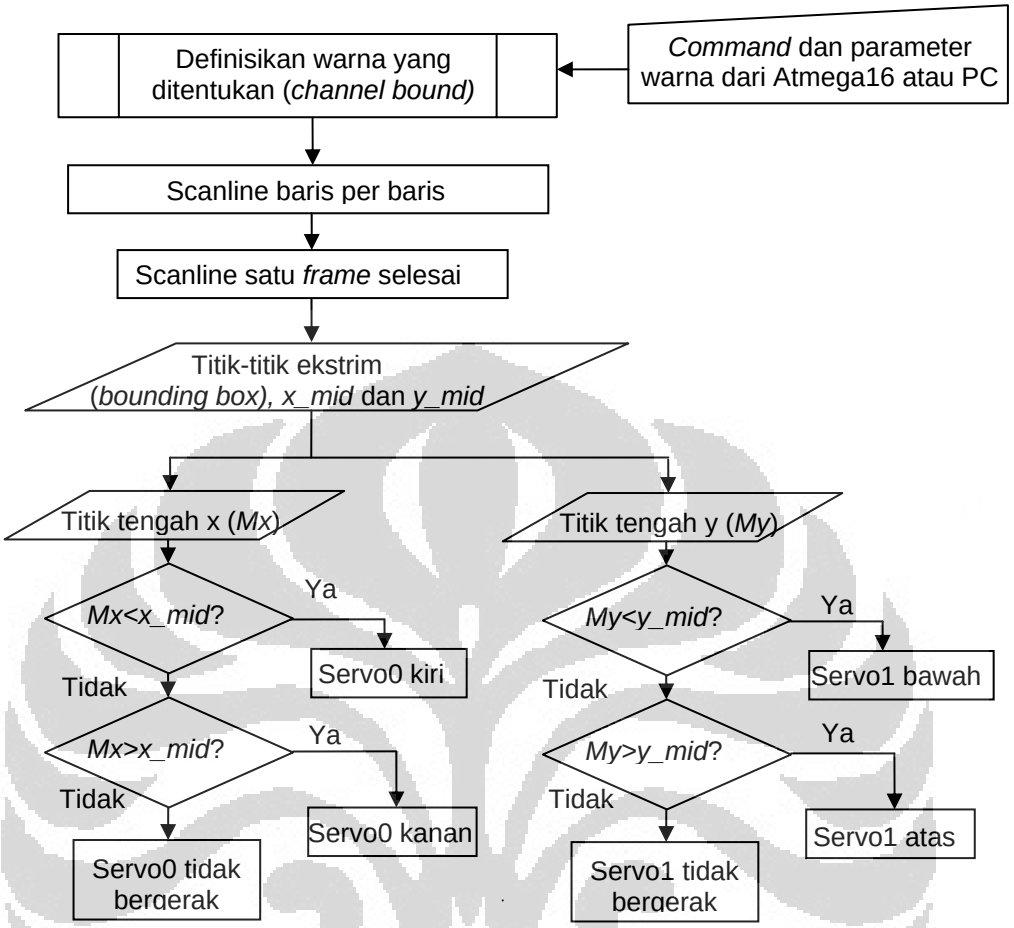


Gambar 3.10 *Flowchart* serial *command*

Pada *Flowchart* diatas, proses utama yang dilakukan adalah mengecek *command* yang masuk melalui serial. Pada keadaan tidak ada *command* maka program akan terus-menerus melakukan *looping*. *Command* dideteksi berdasarkan definisi karakter awal, yang kemudian akan melakukan *switch case* menuju fungsi yang akan dijalankan. Fungsi yang dibuat terbagi menjadi 2 jenis yaitu fungsi dengan argumen dan fungsi tanpa argumen. Jika fungsi tanpa argumen diberi argumen maka program akan menyatakan *command* tidak valid, atau jika ada fungsi dengan argumen tidak diberi argumen maka program juga akan menyatakan *command* tidak valid.

Selain dibedakan berdasarkan argumen, fungsi juga dibedakan berdasarkan kekontinyuan fungsi tersebut. Fungsi seperti *tracking* merupakan fungsi kontinyu, yang artinya setelah fungsi dipanggil, maka fungsi akan melakukan *looping*.

Fungsi *tracking* warna dibuat dengan *flowchart*:



Gambar 3.11 *Flowchart* sistem *tracking* warna

Flowchart tracking warna berlaku jika sebelumnya ada *command* yang memanggil fungsi tersebut. Dari *flowchart* Gambar 3.11 dapat dijelaskan bahwa proses *tracking* menggunakan algoritma yang sederhana dengan memproses setiap *frame* citra yang baru secara independen. Fungsi *tracking* dimulai ketika ada perintah dari Atmega16 atau PC. Proses *scanning* dimulai dari kiri atas dari citra secara berurutan menguji setiap piksel baris demi baris. Jika piksel yang diuji berada dalam *bounding* warna yang terdefinisi, maka ditandai sebagai piksel yang di-*track*. Proses ini juga akan menentukan posisi piksel tersebut berada pada ujung atas, bawah, kanan dan kiri dari keseluruhan piksel yang di-*track* pada citra yang ditemukan pada saat itu, sehingga terdefinisi suatu box yang disebut *bounding box*. Jika piksel lain yang benar berada diluar area *bounding box* yang di-*track*, maka bounding akan melebar supaya piksel baru ini masuk sebagai piksel yang di-*track*. Informasi lain penting yang akan disimpan adalah jumlah dari koordinat

piksel yang di-*track* secara horizontal dan vertikal. Pada akhir proses citra, jumlah kedua koordinat tersebut dibagi dengan jumlah total piksel yang di-*track*, sehingga lokasi nilai tengah (*centroid*) dari piksel yang di-*track* bisa diperoleh. Fungsi API yang dipanggil ketika menjalankan proses *tracking* adalah *"cc3_track_color_scanline"* kemudian data hasil *tracking* disimpan dengan memanggil fungsi API *"cc3_track_color_scanline_finish"*.

Nilai tengah dari 1 *frame* ditentukan sebagai *x_mid* dan *y_mid* berdasarkan rumus:

$$x\_mid = x0 + (width / 2) \quad (3.1)$$

$$y\_mid = y0 + (height / 2) \quad (3.2)$$

dimana:

x0 pada fungsi = cc3_g_pixbuf_frame.x0 = inisial titik x

y0 pada fungsi = cc3_g_pixbuf_frame.y0 = inisial titik y

width pada fungsi = *cc3_g_pixbuf_frame.width* = lebar 1 *frame*

height pada fungsi = *cc3_g_pixbuf_frame.height* = panjang 1 *frame.*

Nilai masing-masing *centroid* berdasarkan rumus:

$$m_x = \frac{\sum_{i=0}^{n} x_i \bullet n}{(x_n - x_0)(y_n - y_0)} \quad (3.3)$$

$$m_y = \frac{\sum_{i=0}^{n} y_i \bullet n}{(x_n - x_0)(y_n - y_0)} \quad (3.4)$$

dimana:

m_x = *centroid x,*

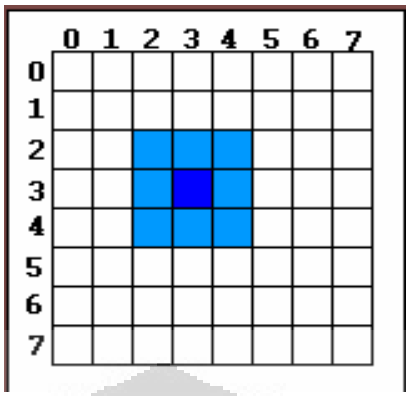
x_0 , x_n = koordinat *x* ekstrem *bounding box*

m_y = *centroid y,*

y_0 , y_n = koordinat *y* ekstrem *bounding box*

Setelah *tracking* 1 *frame* dilakukan, sistem akan mengirimkan laporan data melalui serial berupa data paket yang diinisialisasikan karakter "T". Pada sistem yang dibangun komunikasi yang terjalin akan tampak seperti pada Gambar 3.13.

Contoh ilustrasi mencari *centroid* adalah sebagai berikut:



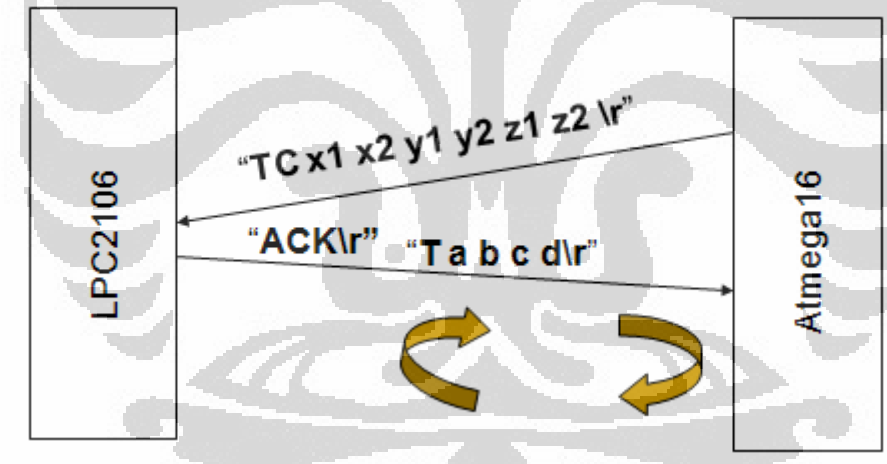
Gambar 3.12 Ilustrasi perhitungan centroid

Sumber:
http://www.cs.uml.edu/~fredm/courses/91.548-spr03/student/tkneelan/CMUCam/CMUCam.html

Pada Gambar 3.12 dapat dihitung posisi tengah dari warna biru :

$$Centroid\_x = (2+3+4)\ x3\ /9 = 3$$

$$Centroid\_y = (2+3+4)\ x3\ /9 = 3$$



Gambar 3.13 Komunikasi antar mikrokontroler.

3.2.3 Perancangan Fungsi *Command* Serial

Untuk berhubungan dengan sistem luar lainnya, maka dibuat program pemanggilan fungsi pada sistem kamera berupa perintah. Beberapa perintah (*command*) yang dibuat antara lain:

- SF\r

Perintah ini akan mengirim *frame* ke komputer lewat serial. Perintah ini mengeluarkan karakter-karakter ASCII yang tidak terbaca berupa data mentah

RGB yang terdiri atas data gambar kolom per kolom dengan *byte* sinkronisasi *frame* dan *byte* sinkronisasi kolom. Data ini dapat ditampilkan oleh program LabVIEW yang dibuat.

- GV\r

  Perintah ini untuk mengetahui versi software dari kamera.

- TC [Rmin Rmax Gmin Gmax Bmin Bmax]\r

  Perintah ini digunakan untuk men-*track* warna ("*Track Color*"). Yang dibutuhkan dalam perintah ini adalah nilai minimum dan maksimum RGB dari warna yang akan di-*track*. Keluaran yang diberikan oleh kamera akan berupa paket data T. Resolusi citra efektif yang diproses adalah 88 x 144 (*default*). Nilai X akan berkisar 1 sampai 88, sedangkan nilai Y berkisar 1 sampai 144. jika data yang diterima berupa angka 0, berarti tidak ada warna yang terlacak. Pemberian perintah ini tanpa argumen akan membuat kamera memakai data sebelumnya.

  Format paket data T:

  T mx my sx sy\r

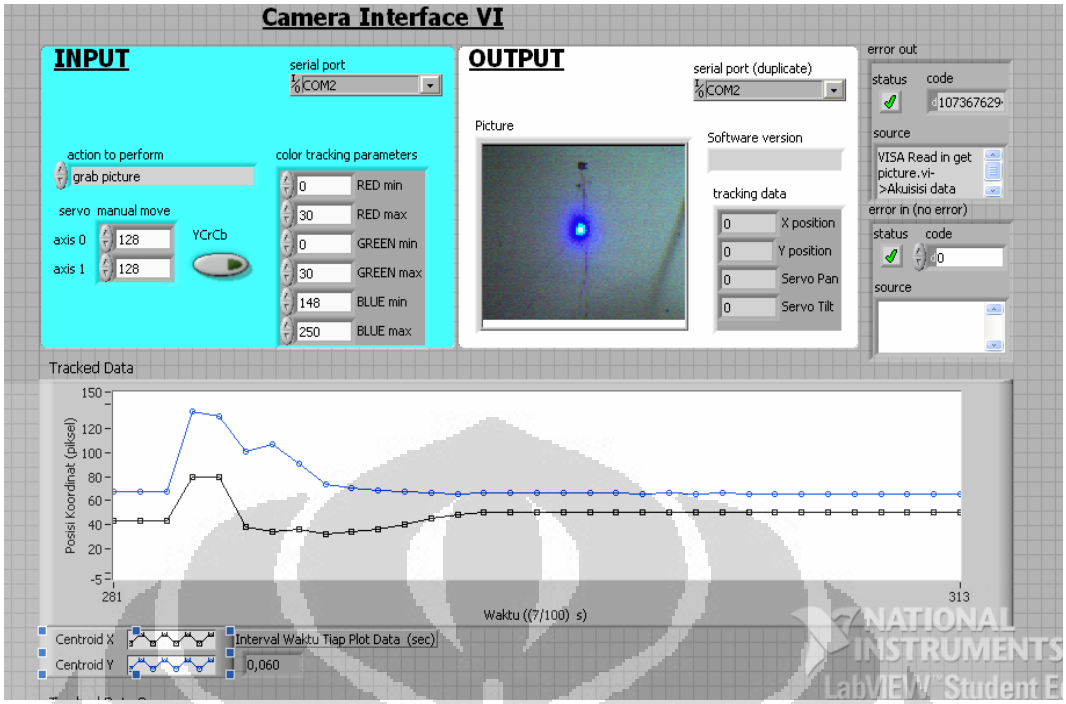
  mx : koordinat pusat massa x
  my : koordinat pusat massa y
  sx : Nilai servo *pan*
  sy : Nilai servo *tilt*

- RS\r

  Perintah ini digunakan untuk me-*reset* kamera. Setelah kamera di-reset karakter pertama yang dikirimkan adalah \r

3.2.4 Perancangan Perangkat Lunak Pada Komputer

Antarmuka pada komputer dibuat dengan menggunakan pemrograman grafik LabVIEW 8.0. Fungsi dari program yang dibuat adalah untuk mengetahui fungsi-fungsi sistem utama dengan konfigurasi melalui serial *command* seperti halnya yang dilakukan oleh Atmega16. Desain tampilannya dapat dilihat pada Gambar 3.14.



Gambar 3.14 Tampilan program LabVIEW.

Keterangan:

- Action to perform:

  Untuk menginformasikan sistem supaya melakukan aksi tertentu yaitu: *grab picture, get version, start tracking, stop tracking, close serial port, initialization serial, get chart, move servo*.

- *Serial Port*:

  Untuk memilih port serial yang ada pada komputer.

- *Color tracking parameter*:

  Parameter warna yang akan di-*track*

- *Servo manual move*:

  Menjalankan servo sesuai parameter yang dimasukkan.

- *Tracked data*:

  Grafik dari keluaran kamera.

- *Picture*:

  Menampilkan citra yang diperoleh setelah aksi "*grab picture*" dijalankan

- *Camera version*:

  Untuk mengetahui versi *software* yang dibuat pada kamera.

BAB 4
HASIL DAN PEMBAHASAN

Pada bab ini akan dibahas tentang pengujian dan analisa sistem yang telah dikerjakan. Pengujian ini dilakukan untuk mengetahui kemampuan sistem apakah telah berfungsi seperti apa yang diharapkan dan menganalisa apabila terjadi kegagalan.

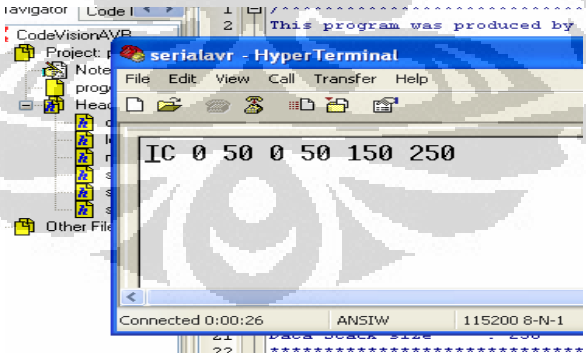
4.1 Pengujian Keluaran Atmega16

Untuk menguji keluaran serial dari Atmega16 sebelumnya dilakukan penekanan *keypad* dengan proses sebagai berikut:

Mula-mula pada layar LCD akan tampil berbagai string kalimat yang diakhiri string seperti pada Gambar 4.1, setelah itu dilakukan penekanan *keypad*, parameter warna yang diinginkan, hasil pada LCD seperti gambar 4.1:

Gambar 4.1 Tampilan LCD

Setelah penekanan selesai (diakhiri tombol ” * ”), kemudian dilihat hasilnya dengan program *Hyperterminal* seperti pada Gambar 4.2:

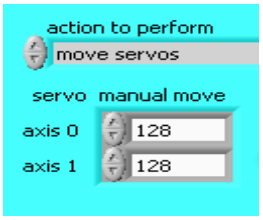


Gambar 4.2 Keluaran Atmega16 pada *Hyperterminal*

4.2 Uji Linearitas Servo

Untuk mengetahui apakah servo motor dapat berfungsi dengan baik maka dilakukan pengujian linearitas posisi servo motor terhadap masukan yang diberikan. Data diambil dengan memberikan parameter *input* pada program

LabVIEW yang dibuat, setelah itu dipilih aksi "*move servos*". Hasilnya dapat dilihat pada Tabel 4.1.



Gambar 4.3 *Input* servo melalui LabVIEW

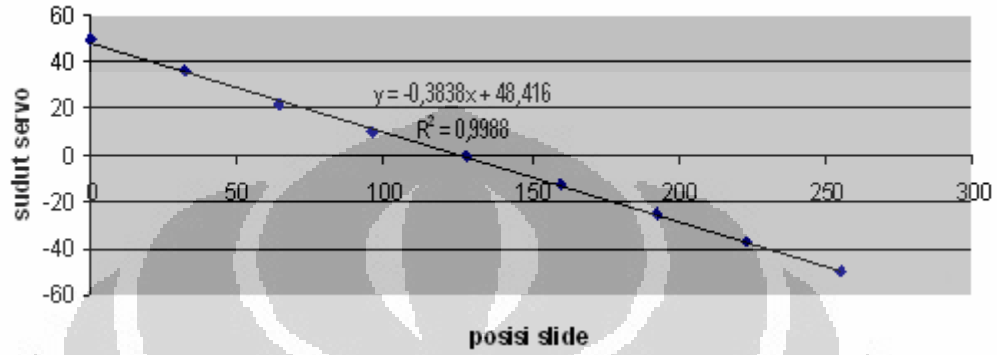
Tabel 4.1 Keluaran port servo untuk *pan*.

Parameter *Input*	Sudut	Tegangan (volt)	Arah putaran
0	50°	0,17	CW
32	36°	0,19	CW
64	22°	0,22	CW
96	10°	0,24	CW
128	0°	0,26	Tidak berputar
160	12°	0,28	CCW
193	25°	0,30	CCW
223	37°	0,33	CCW
255	50°	0,35	CCW

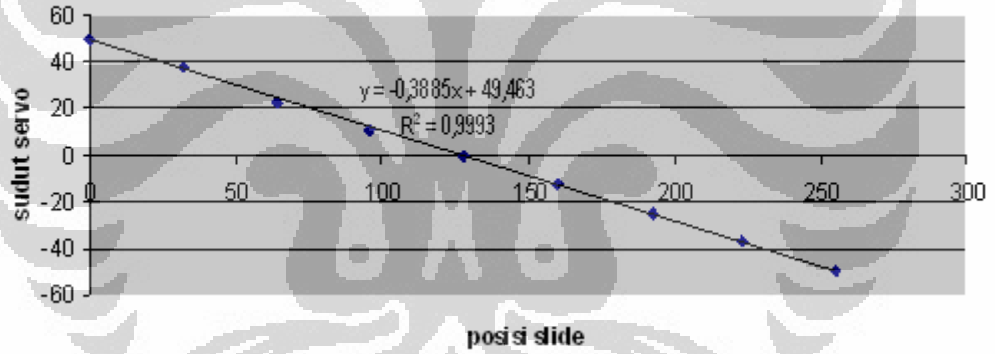
Tabel 4.2 Keluaran port servo untuk *tilt*.

Parameter *Input*	Sudut	Tegangan (volt)	Arah putaran
0	50°	0,17	CW
32	38°	0,19	CW
64	23°	0,21	CW
96	11°	0,23	CW
128	0°	0,26	Tidak berputar
160	12°	0,28	CCW
193	25°	0,31	CCW
223	37°	0,34	CCW
255	50°	0,36	CCW

Servo *pan* yang bergerak CW (*Counter Wise*) berarti servo bergerak ke kanan dari kamera dan sebaliknya. Dan pada servo *tilt* yang bergerak CW berarti servo bergerak ke bawah, demikian sebaliknya. Dari data yang diperoleh dibuat grafik untuk mendapatkan persamaan garis untuk menentukan sudut tertentu yang tidak terdapat pada tabel. Grafik ditunjukkan oleh Gambar 4.1 dan 4.2.



Gambar 4.4 Linearitas posisi servo *pan* terhadap *input*

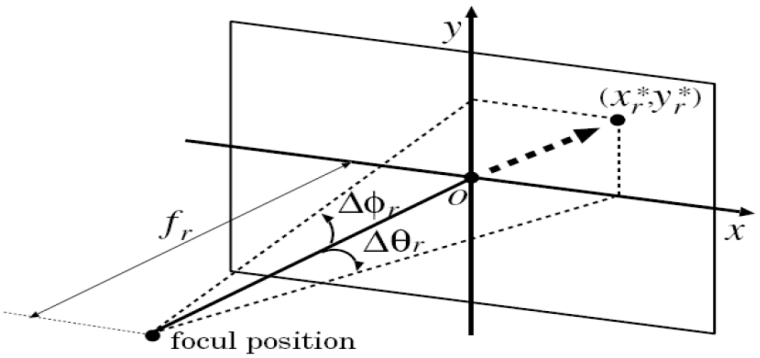


Gambar 4.5 Linearitas posisi servo *tilt* terhadap *input*

Dari dua grafik diatas, linearitas servo terhadap posisi sudut yang dibentuk cukup baik berdasarkan nilai R^2 yang mendekati 1.

4.3 Uji Coba Jangkauan *Frame*

Untuk mengetahui hubungan sudut *pan/tilt* dengan koordinat objek pada layar dapat ditampilkan geometri sederhana seperti Gambar 4.3 :



Gambar 4.6 Sudut *tilt/pan* menurut jarak pada sebuah citra

Sumber: Interstudy.Advanced .Curricula and Mobile Tools for Interdisciplinary Modular Study:122

Dari Gambar 4.6 diformulasikan:

$$\Delta\theta_r = Arc\tan(\frac{x_r}{f_r}) \quad (4.1)$$

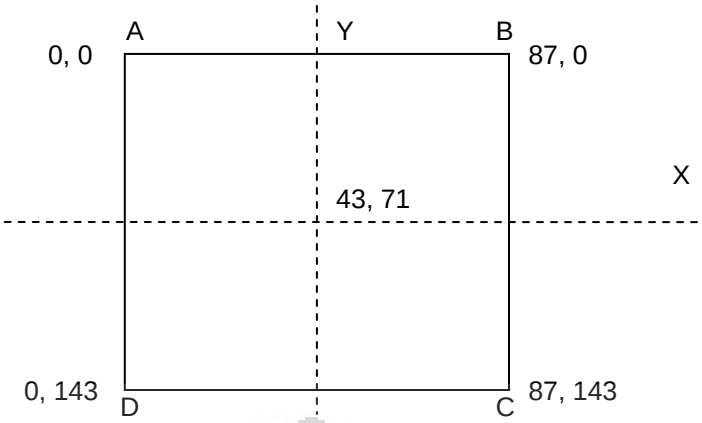
$$\Delta\phi_r = Arc\tan(\frac{y_r}{f_r}) \quad (4.2)$$

Dimana f_r adalah jarak titik fokus dan bidang *tracking*.

4.3.1 Kalibrasi Bidang *Tracking*

Kalibrasi dilakukan untuk menguji kesejajaran bidang *tracking* (dinding) terhadap bidang sensor kamera, sekaligus menguji kestabilan koordinat *tracking*. Data diambil dalam selang waktu 20 detik untuk setiap koordinat, dengan mengamati nilai yang muncul pada program LabVIEW. Posisi sensor dengan titik tengah bidang *tracking* berjarak 150 cm. Data juga diambil dengan pengambilan sampel data yang cenderung valid, yang artinya warna yang sedang di-*track* terdeteksi dengan baik.

Dari data yang terlihat, koordinat piksel pada sistem dapat digambarkan berdasarkan titik-titik ujung bidang. Bidang *tracking* yang terbentuk adalah seperti pada Gambar 4.7:

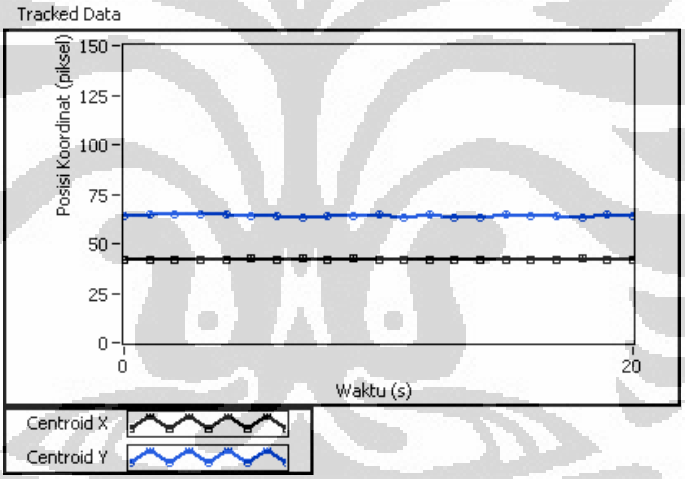


Gambar 4.7 Bidang *tracking*

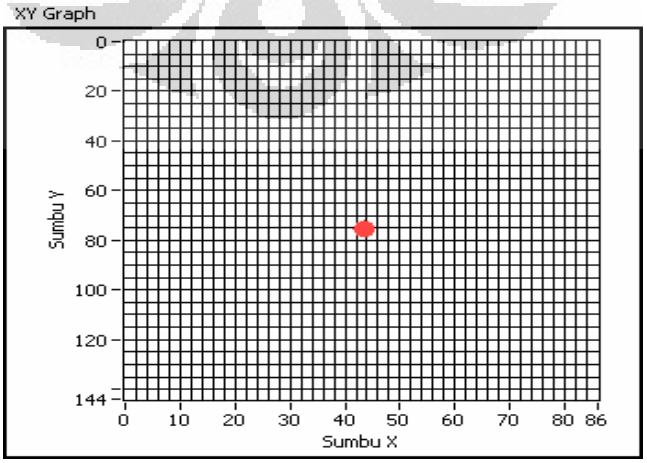
Panjang yang terukur pada bidang *tracking* adalah :

AB = 111 cm, BC = 87 cm, CD = 110 cm, DA = 85 cm.

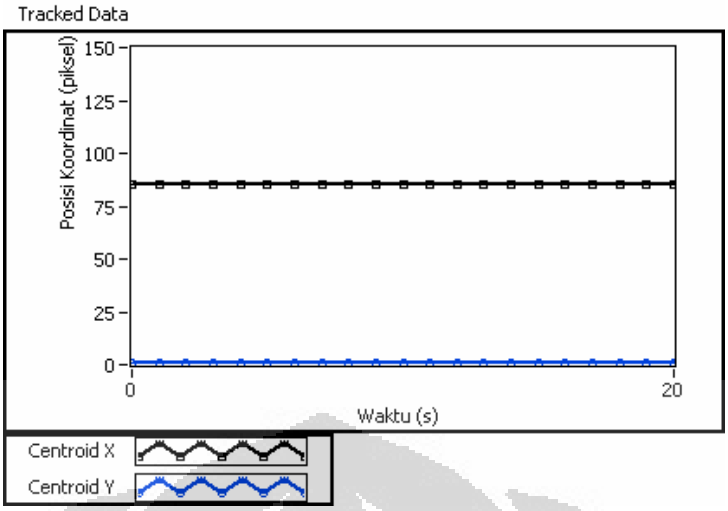
Respon kestabilan titik sampel adalah sebagai berikut :



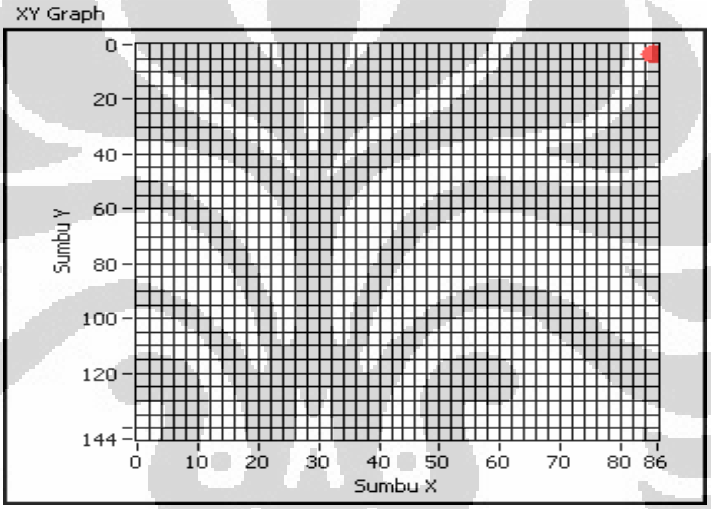
Gambar 4.8 Grafik respon titik tengah bidang *tracking*.



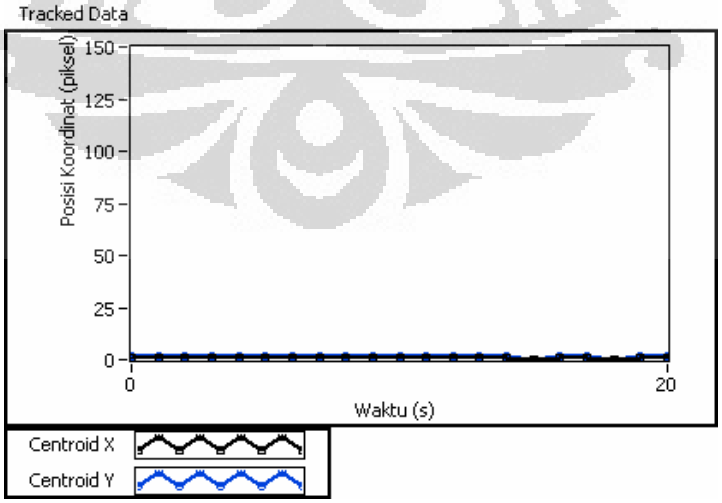
Gambar 4.9. Plot titik tengah pada bidang XY



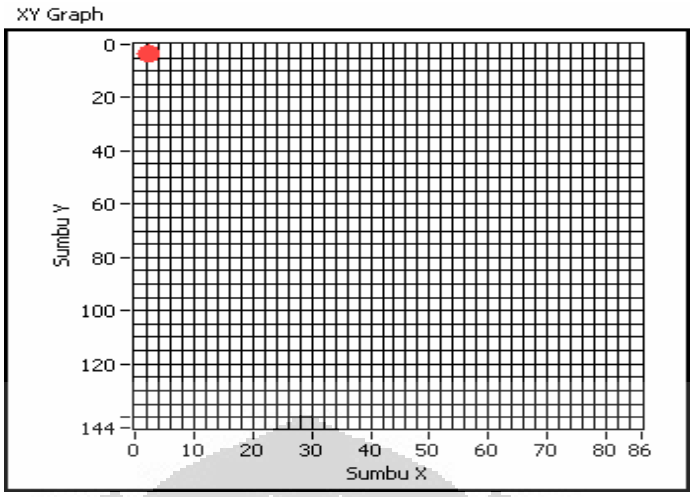
Gambar 4.10 Grafik respon titik B bidang *tracking*



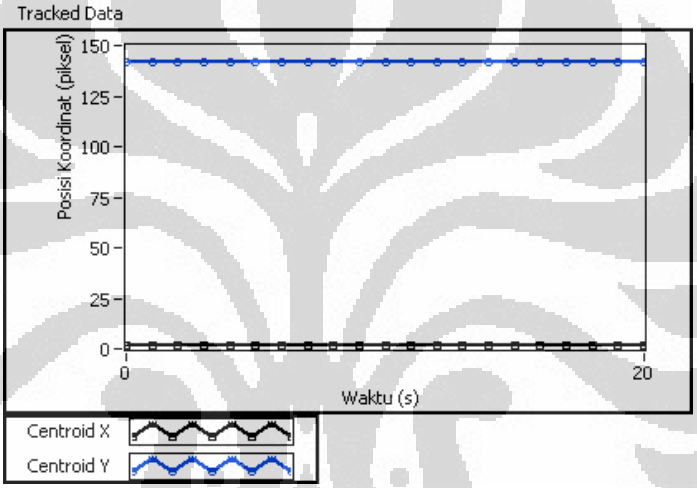
Gambar 4.11 Plot titik B pada bidang XY



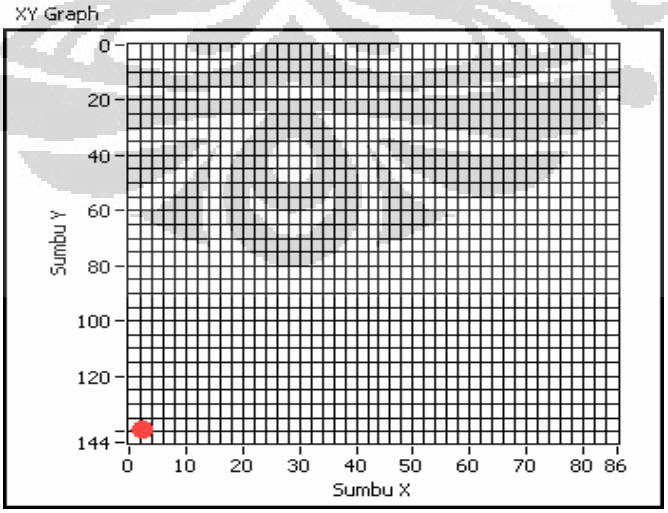
Gambar 4.12 Grafik respon titik A bidang *tracking*



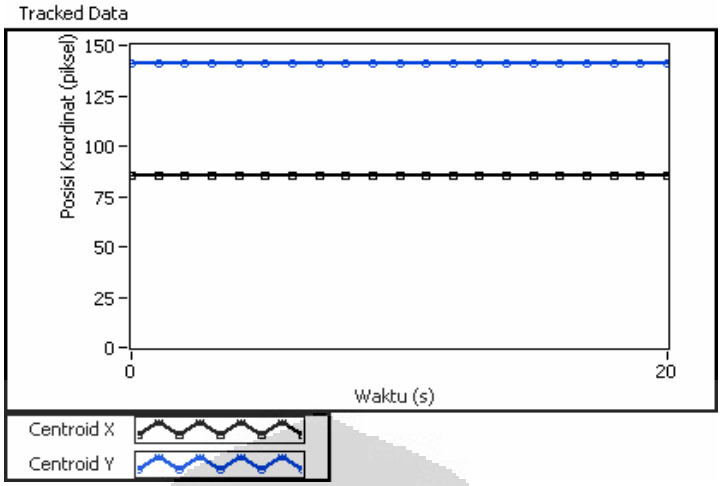
Gambar 4.13 Plot titik A pada bidang XY



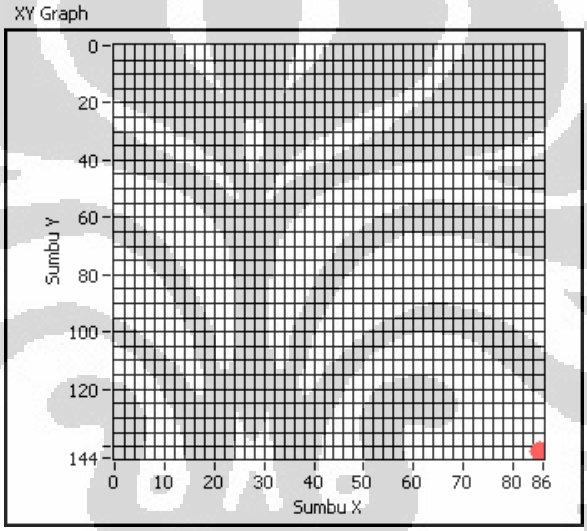
Gambar 4.14 Grafik respon titik D bidang *tracking*



Gambar 4.15 Plot titik D pada bidang XY



Gambar 4.16 Grafik respon titik C bidang *tracking*



Gambar 4.17 Plot titik C pada bidang XY

Dari data yang diperoleh terlihat bahwa kestabilan titik-titik sampel cukup baik, serta bidang *tracking* yang terbentuk juga cukup baik, dimana antara 2 sisi yang sejajar tidak terlalu signifikan perbedaan jaraknya.

4.3.2 Pengujian dengan Variasi Jarak

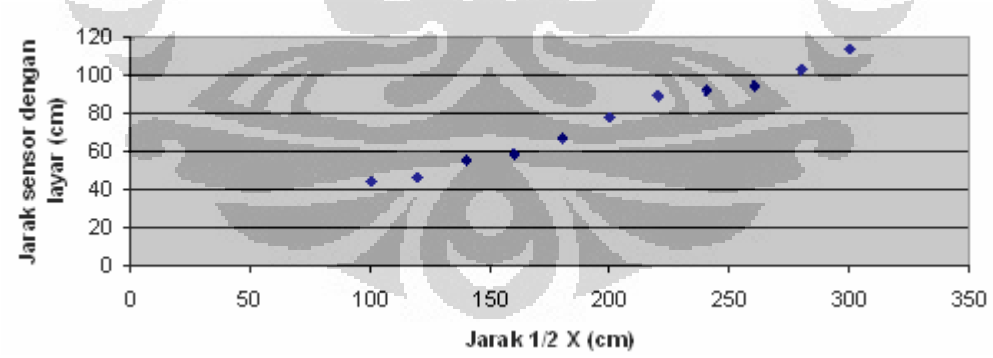
Pada pengujian ini , dibuat variasi jarak antara 1 sampai 3 m dengan selang 0,2 m. Pada jarak lebih dari 3 m LED kurang terdeteksi dengan baik. Tujuan dari pengukuran ini adalah untuk menentukan jangkauan bidang *tracking*, sejauh mana posisi objek dapat terdeteksi. Jangkauan *frame tracking* diukur pada jarak ½ X dan ½ Y dari titik tengah bidang *tracking*, karena posisi awal kamera idealnya adalah pada tengah bidang *tracking*. Pengukuran dilakukan secara

manual dengan acuan posisi LED yang diletakkan pada bidang. Hasilnya dapat dilihat pada Tabel 4.3.

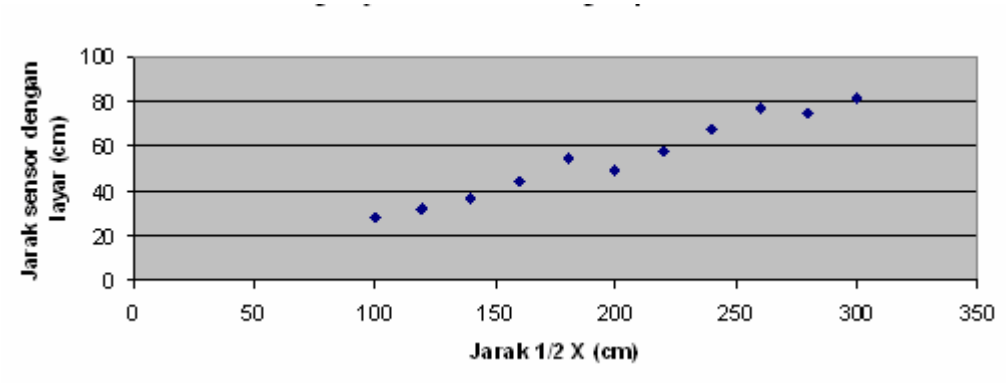
Tabel 4.3 Pengukuran jarak secara manual

Jarak layar dengan sensor (cm)	Jarak ½ *X* pada layar (cm)	Jarak ½ *Y* pada layar (cm)
100	44,0	28,0
120	46,0	32,1
140	54,7	36,9
160	58,5	44,0
180	67,3	55,0
200	78,0	49,8
220	89,3	58,1
240	91,5	68,0
260	94,6	77,0
280	102,4	75,0
300	114,0	81,5

Dari Tabel 4.3, dibuat grafik yang menghubungkan parameter-parameter tersebut. Hasilnya seperti pada Gambar 4.18 dan Gambar 4.19:



Gambar 4.18 Hubungan jarak sensor dengan posisi ½ X



Gambar 4.19 Hubungan jarak sensor dengan posisi ½ Y

Fungsi ideal dari kedua grafik diatas adalah fungsi *tangen*. Hubungan sudut pada tiap masing-masing sumbu koordinat sesuai dengan rumus (4.1) dan (4.2) , dapat dihitung $\Delta\theta_r$ dan $\Delta\Phi_r$ dengan hasil pada Tabel 4.4 :

Tabel 4.4 Perhitungan sudut

Jarak layar dengan sensor (cm)	$\Delta\theta_r$ (°)	$\Delta\Phi_r$ (°)
100	23,0	16,0
120	20,0	15,0
140	21,0	14,8
160	20,8	15,4
180	20,5	17,0
200	21,3	14,0
220	22,1	14,8
240	20,9	15,8
260	20,0	16,5
280	20,1	15,0
300	20,8	15,2

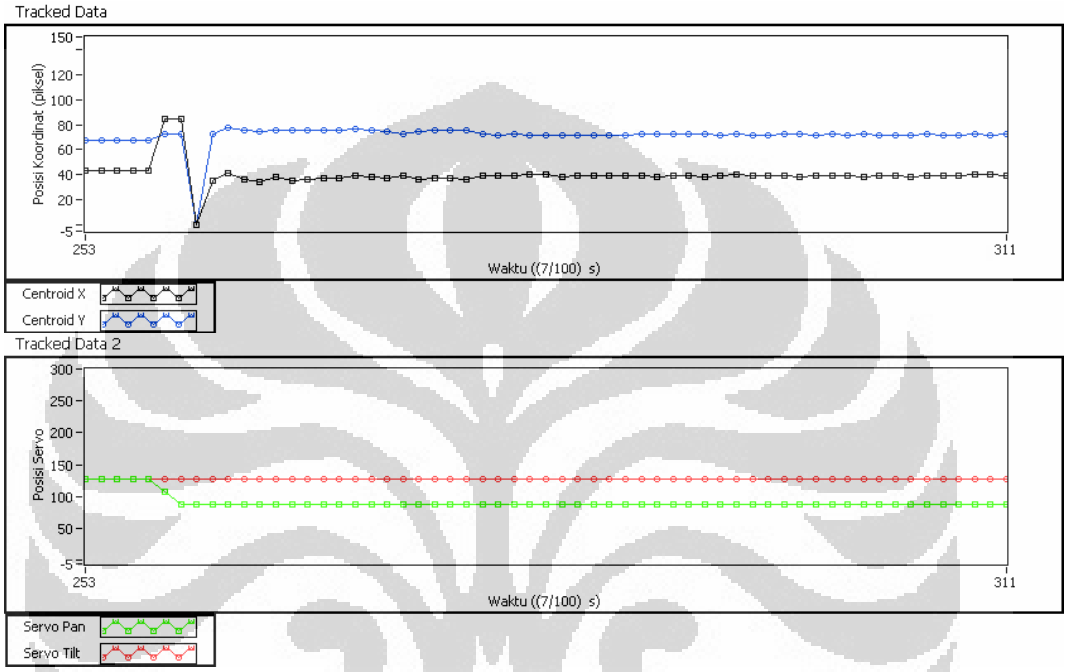
Nilai rata-rata $\Delta\theta_r$ adalah 20,9° dengan kesalahan relatif sebesar 3,58% dan $\Delta\Phi_r$ adalah 15,4° dengan kesalahan relatif sebesar 6,46%.

4.4 Uji Coba *Tracking.*

Uji coba dilakukan untuk mengetahui hubungan antara koordinat piksel dengan nilai servo. Grafik di-*plot* dalam waktu sekitar per 0,07 detik sesuai hasil selisih waktu antar data yang muncul pada LabVIEW. Gerak yang diukur adalah

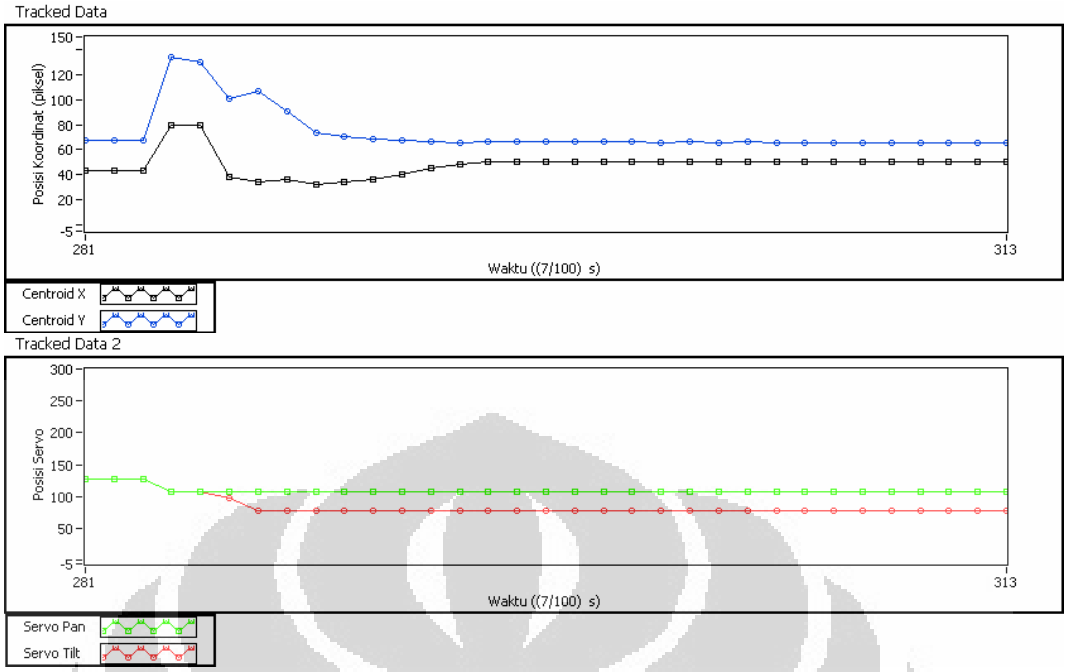
gerakan pada bidang *tracking* dengan jarak bidang (titik tengah) dengan sensor adalah 100 cm dan dengan variasi jarak pergerakan terhadap titik tengah bidang *tracking*. Inisial awal adalah sensor mendeteksi titik tengah bidang *tracking* terlebih dahulu, kemudian dilanjutkan ke titik yang lain. Hal ini dilakukan untuk mempermudah penghitungan sudut gerak dari sistem.

Data yang diperoleh di-*plot* dengan grafik-grafik sebagai berikut:



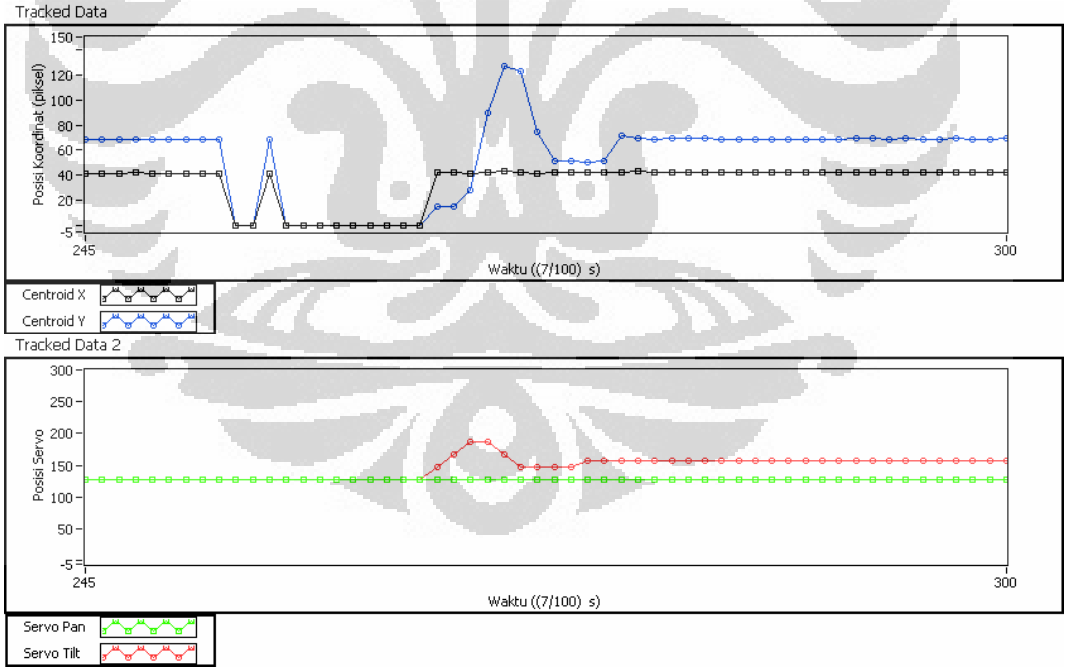
Gambar 4.20 Grafik respon pergerakan sejauh 38 cm.

Pada Gambar 4.20 terlihat adanya pergerakan ditandai perubahan nilai *centroid* dan servo. *Centroid* bergerak dari titik (42,68) ke titik (82,73) dan cenderung stabil setelah servo bergerak yaitu pada kisaran posisi (39,70).



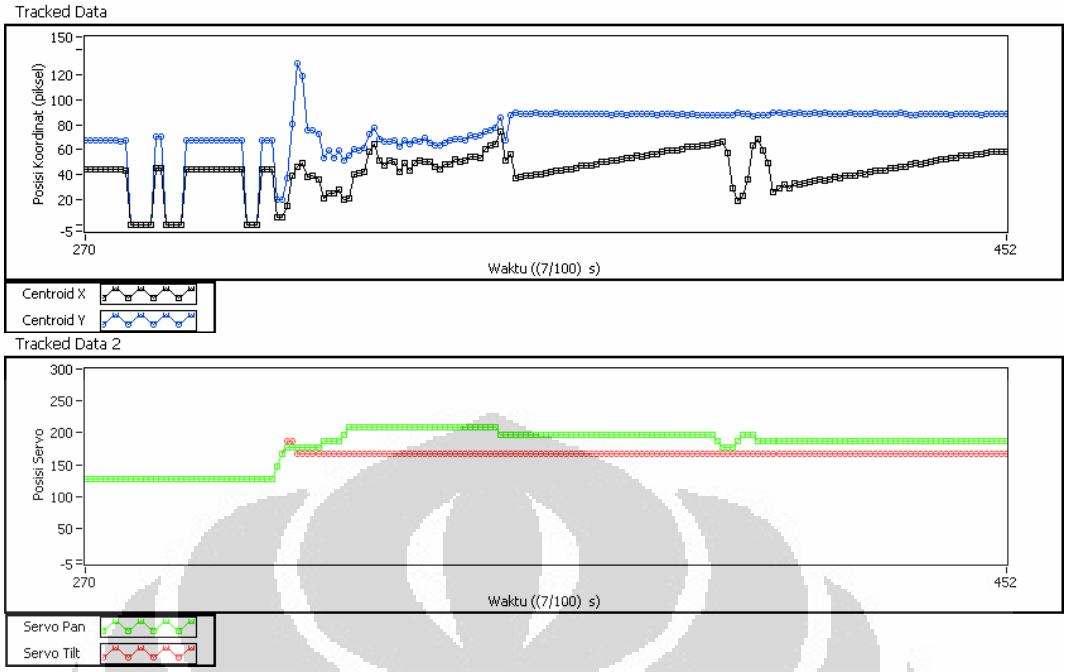
Gambar 4.21 Grafik respon pergerakan sejauh 44 cm.

Centroid bergerak dari titik (42,68) ke titik (78,144) dan cenderung stabil setelah servo bergerak yaitu pada posisi (43,65).



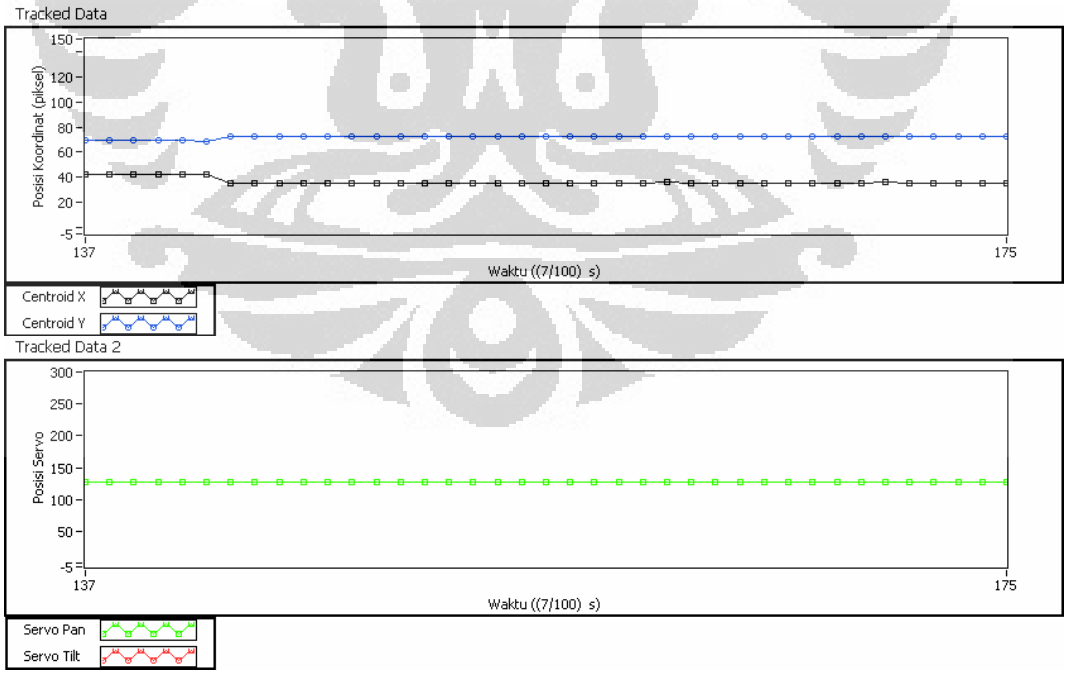
Gambar 4.22 Grafik respon pergerakan sejauh 22 cm.

Centroid bergerak dari titik (42,68) dan cenderung stabil setelah servo bergerak yaitu pada posisi (42,68). Sebelum terjadi pergerakan servo, *centroid* berfluktuasi ke titik (0,0).



Gambar 4.23 Grafik respon pergerakan sejauh 41 cm.

Centroid bergerak dari titik (42,68), berfluktuasi dan cenderung stabil setelah servo bergerak yaitu pada posisi (43,66). Pada kasus ini respon cenderung membutuhkan waktu yang lebih lama, penyebabnya adalah *centroid* yang berfluktuasi tinggi saat proses sebelum akhirnya stabil.

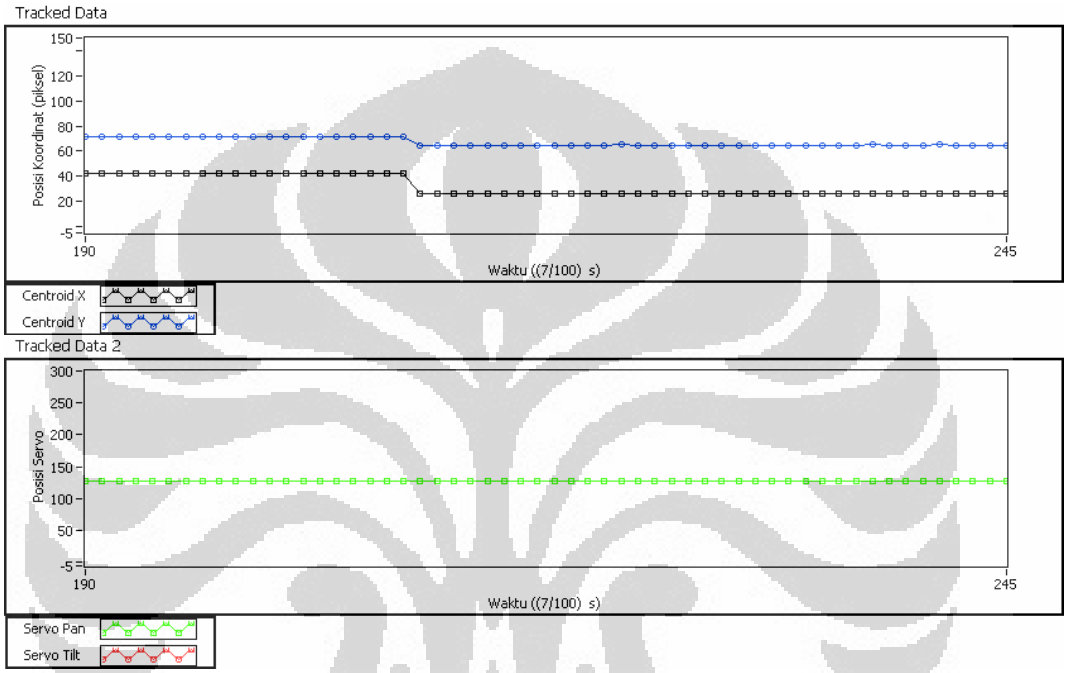


Gambar 4.24 Grafik respon pergerakan 10 cm.

Centroid bergerak dari titik (42,69) ke titik (38,76) dan cenderung stabil pada posisi tersebut dengan tidak terjadinya gerakan pada servo. Kondisi ini diakibatkan karena pada program terdefinisi:

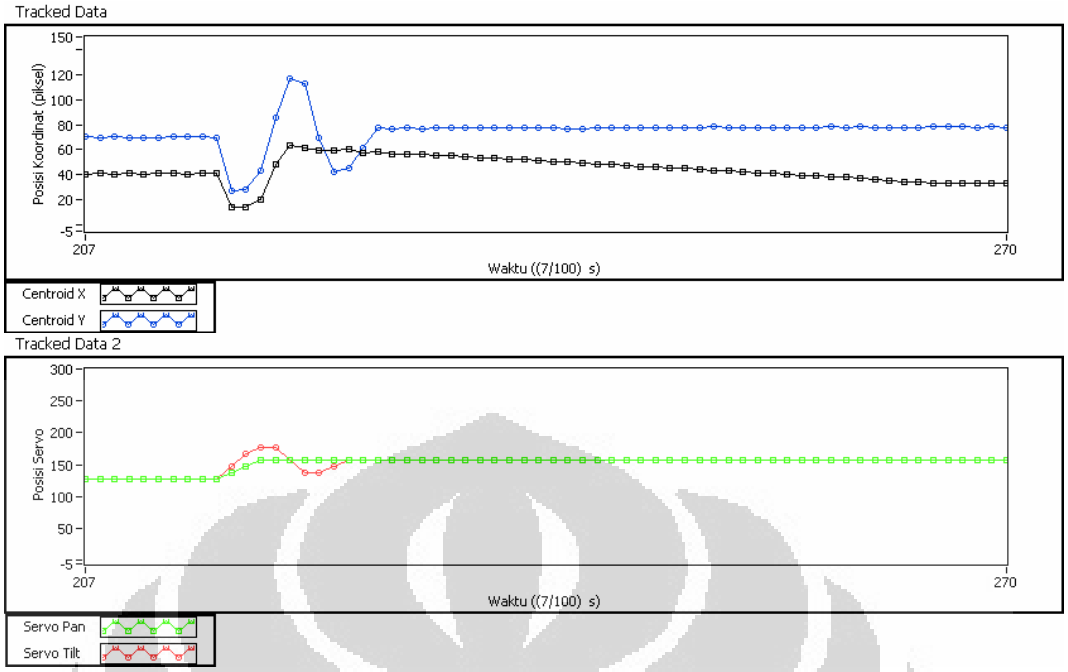
```
servo_settings.pan_range_near = 20;
servo_settings.tilt_range_near = 20;
```

yang menunjukkan bahwa jangkauan terdekat untuk menggerakkan servo adalah 20 piksel. Sehingga transisi diatas belum bisa menggerakkan servo.



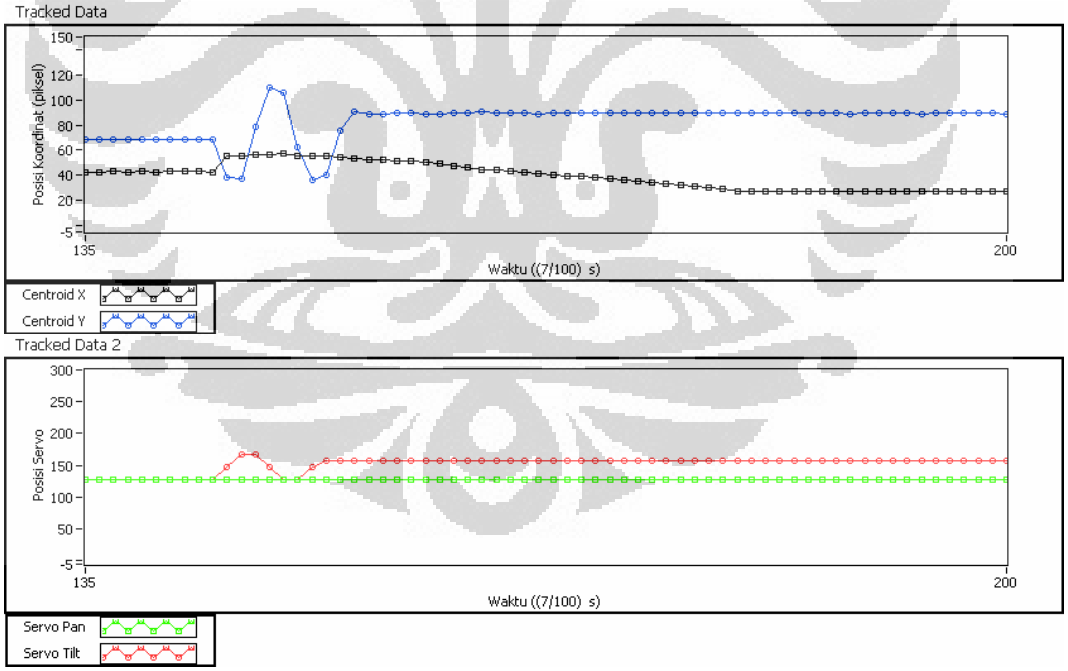
Gambar 4.25 Grafik respon pergerakan sejauh 16 cm.

Centroid bergerak dari titik (42,68) ke titik (20,61) dan cenderung stabil pada posisi tersebut. Sama halnya dengan data sebelumnya, perubahan nilai *centroid* belum dapat menggerakkan servo.



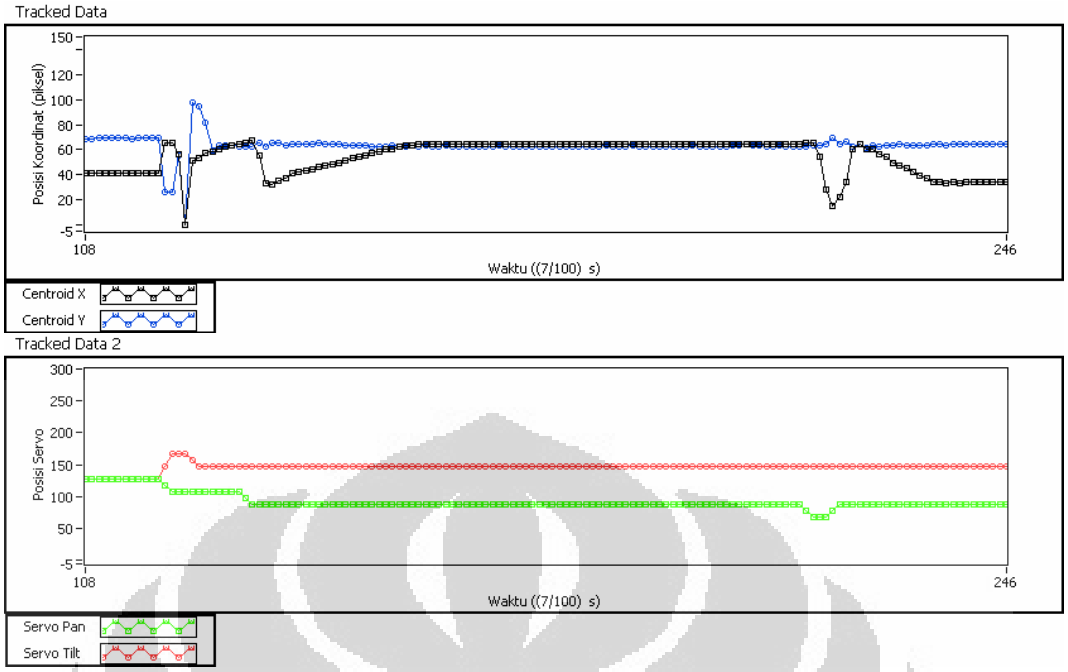
Gambar 4.26 Grafik respon pergerakan 30 cm.

Centroid bergerak dari titik (42,68) ,berfluktuasi dan cenderung stabil setelah servo bergerak yaitu pada posisi (40,74).



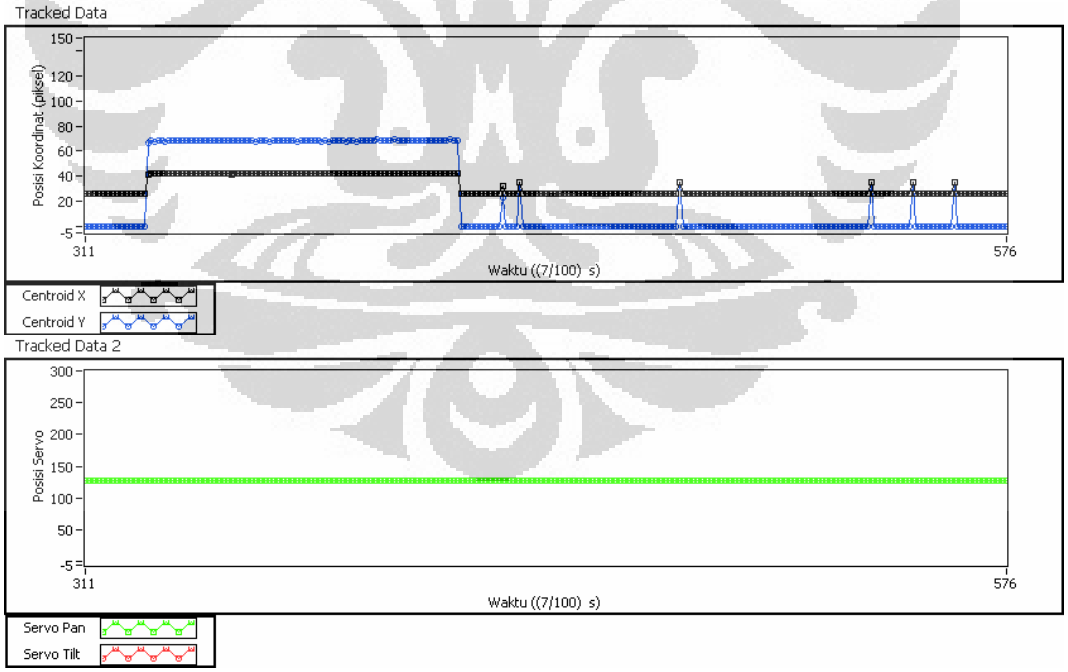
Gambar 4.27 Grafik respon pergerakan sejauh 19 cm.

Centroid bergerak dari titik (41,68), berfluktuasi dan cenderung stabil setelah servo *tilt* bergerak yaitu pada posisi (22,84). Respon servo *pan* cenderung tetap dikarenakan perubahan *centroid* x yang belum mampu menggerakkan servo *pan*.



Gambar 4.28 Grafik respon pergerakan sejauh 27 cm.

Centroid bergerak dari titik (42,68) ke titik (43,65) dan cenderung stabil setelah servo bergerak yaitu pada posisi (23,61). Pergerakan kedua servo terjadi ketika ada transisi >20 piksel.



Gambar 4.29 Grafik respon pergerakan sejauh 34 cm.

Pada awal *tracking*, *centroid* jatuh pada nilai yang rendah sebelum akhirnya pada titik awal. Ini diakibatkan karena sistem sempat mengalami kegagalan deteksi akibat pengaruh kondisi lingkungan seperti gangguan warna

yang berasal dari refleksi cahaya. Proses uji *tracking* ini mengalami kegagalan karena tidak ada pergerakan servo, meskipun transisi *centroid* idealnya mampu menggerakkan servo.

Proses *tracking* yang valid ditandai oleh kestabilan nilai servo atau kestabilan nilai *centroid* dalam waktu yang relatif lebih lama. Sehingga data grafik dapat disederhanakan pada tabel berikut:

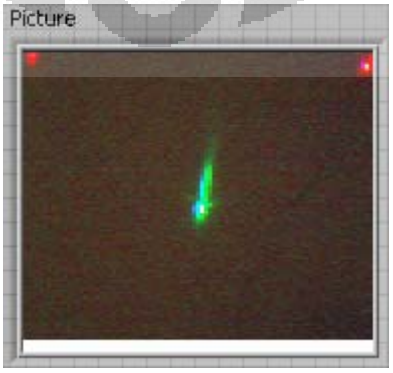
Tabel 4.5 Data *Tracking*

Jarak objek terhadap titik tengah bidang (cm)	Waktu antara 2 kestabilan servo (s)	Waktu antara 2 kestabilan *centroid* (s)	Sudut gerak (°)
10	0 (tidak bergerak)	0,14	5,71
16	0 (tidak bergerak)	0,14	9,09
19	0,63	2,73	10,75
22	0,77	1,82	12,40
27	0,98	9,52	15,10
30	0,70	3,29	16,69
34	0 (tidak bergerak)	16,24	18,77
38	0,21	0,42	20,80
41	5,67	12,8	22,29
44	0,35	0,91	23,74

Dari 10 percobaan lainnya terjadi 1 sampai 3 kegagalan, hingga rata-rata persentase keberhasilan sistem ini berkisar pada 80%.

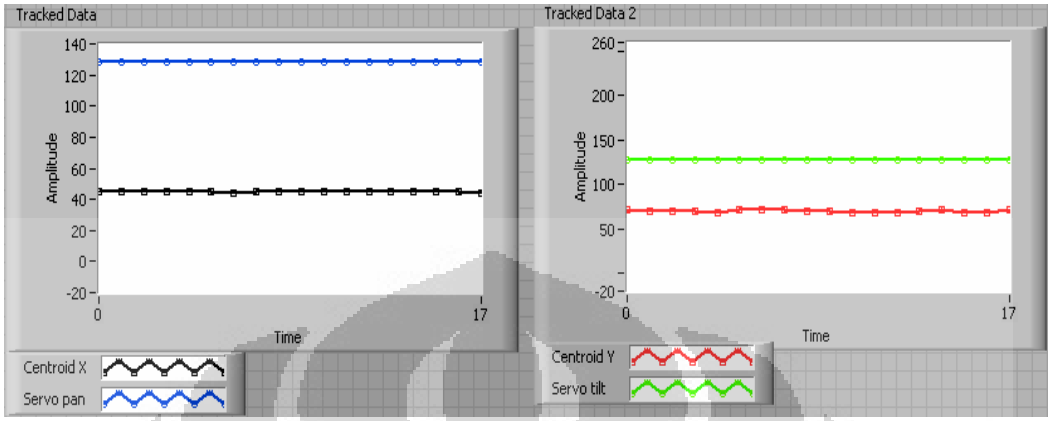
4.5 Uji Coba *Tracking* dengan 2 Warna Berbeda

Uji coba sistem dilakukan dengan meletakkan 2 jenis LED, merah dan hijau, kemudian dilakukan *capture* seperti yang terlihat pada Gambar 4.30:



Gambar 4.30 Uji coba dengan 2 jenis LED.

Warna yang ingin di-*track* adalah warna hijau dengan parameter hasil dari program *Hex Color Finder 3.0* adalah: R 16-45, G 154 – 240, B 48 – 91. Hasilnya ditunjukkan pada Gambar 4.31:



Gambar 4.31 Grafik percobaan 2 jenis LED.

Dari hasil tersebut dapat disimpulkan bahwa sistem berjalan seperti yang diinginkan. LED merah tidak terdefinisi untuk dilakukan *tracking*.

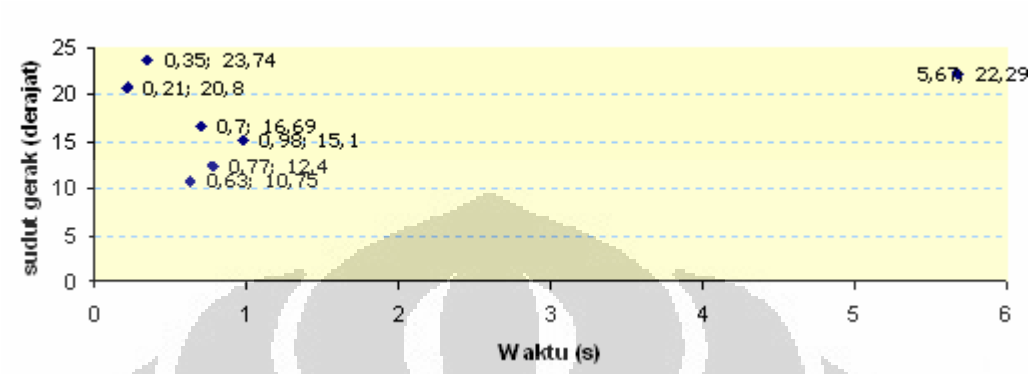
Pada sistem juga dilakukan pangambilan gambar pada tingkat luminasi yang berbeda. Pada LED, tingkat luminasi ruangan yang kecil meningkatkan pendeteksian warna dan sebaliknya luminasi yang tinggi akan menyebabkan warna sejenis terpantulkan menuju bidang *tracking*. Akan tetapi untuk benda warna yang pasif (tidak menghasilkan cahaya sendiri) akan berlaku kebalikan. Perbedaannya dapat dilihat pada Gambar 4.32:

Gambar 4.32 Foto pada luminasi 35 FC dan 2,45 FC

4.6 Analisa Hasil

Selisih waktu pergerakan servo dengan awal perubahan *centroid* berdasarkan data yang didapat mendekati nilai 0, hal ini menandakan respon servo cukup baik. Kecepatan *tracking* diperoleh melalui hubungan antara sudut *tracking*

dengan waktu yang diperlukan servo bergerak mencapai keadaan stabil dengan pengecualian pada tidak bergeraknya servo. Hubungan tersebut dapat diperlihatkan pada Gambar 4.33:



Gambar 4.33 Grafik respon sudut gerak terhadap waktu.

Dari gambar diatas dapat dijelaskan bahwa pergerakan hampir tidak memiliki hubungan linier terhadap waktu, akan tetapi memiliki kecenderungan waktu berkisar dibawah 1 detik dan hanya ada 1 data diluar 1 detik. Kecepatan servo sendiri berdasarkan lembar data adalah 0,23 detik / 60° atau sekitar 260° perdetik. Berdasarkan hal tersebut, kecepatan gerak maksimum servo mampu menangani bidang jangkauan *tracking* dalam waktu < 1 detik. Pembatas kinerja ditentukan pada ada tidaknya objek yang terdeteksi dalam jangkauan bidang *tracking.*

Kesalahan atau kegagalan yang terjadi merupakan akibat dari:

- Pengukuran pada bidang *tracking* yang tidak akurat
- Perubahan distribusi warna yang dihasilkan LED yang tidak stabil saat terjadi pergerakan
- Perubahan distribusi warna yang tidak stabil akibat faktor lingkungan seperti warna-warna sejenis yang masuk dalam bidang *tracking*.
- Geometri yang tidak sejajar antara bidang *tracking* dengan bidang sensor setelah terjadi pergerakan.

BAB 5
KESIMPULAN DAN SARAN

5.1 Kesimpulan

Berdasarkan sistem yang telah dibuat secara keseluruhan, dapat diambil kesimpulan sebagai berikut:

- Mikrokontroler LPC2106 mampu menjalankan proses citra dengan variasi warna sebanyak (2^8-32) pada tiap kanalnya. Karena *black level* pada sensor CMOS adalah 16/240.
- Komunikasi serial berhasil dilakukan antar mikrokontroler LPC2106 dan Atmega16 pada baud rate 115,200.
- Keluaran servo cukup baik pada pengujian servo, akan tetapi pada hasil akhir masih kurang baik karena servo terkadang tidak stabil. Tingkat keberhasilan berkisar 80%.
- Waktu penyesuaian centroid ketika centroid berpindah maksimum < 1s dengan maksimum kecepatan gerak sesuai lembar data servo yaitu 260°/s
- Jangkauan bidang *tracking* untuk sumbu X dan Y, $\Delta\theta_r$ adalah 20,9° dengan kesalahan relatif sebesar 3,58% dan $\Delta\Phi_r$ adalah 15,4° dengan kesalahan relatif sebesar 6,46%.
- Laju keluaran data adalah 14 data tiap detik.

5.2 Saran

Saran-saran yang dapat disampaikan untuk pengembangan penelitian berikutnya mengenai proses citra digital pada mikrokontroler selanjutnya adalah:

- Penggunaan servo dengan jangkauan 360˚ untuk meningkatkan jangkauan pemantauan.
- Penggunaan kamera dengan resolusi yang lebih tinggi untuk meningkatkan performa sistem
- Pengembangan *software* untuk aplikasi lain, seperti *face-detection*, *edge-detection*, *witness-camera* dan sebagainya.
- Pengembangan *software* untuk mengaplikasikan sistem pada satu chip mikrokontroler.

DAFTAR REFERENSI

Administrator. (16 Januari 2007). *LPC2000 MCU Memory Acceleration Module*. 27 November 2007. http://winarm.scienceprog.com/arm-mcu-types/lpc2000-mcu-memory-acceleration-module.html

Averlogic. (28 November 2001). *AL440B Datasheet*. 27 November 2008. http://www.cmucam.org/attachment/wiki/Documentation/al440B.pdf

Bejo, Agus. (2008). *C dan AVR rahasia kemudahan bahasa C dalam mikrokontroler ATMega8535*. Yogyakarta: GRAHA ILMU.

Christopher, G Relf. (2004). *Image Acquisition and Processing with LabVIEW*. Florida : CRC Press.

Danny Diaz. (31 Januari 2006). *LabVIEW CMUCam2 Application*. 27 November 2008. http://www.chiefdelphi.com/forums/showthread.php?t=41214.

Doxygen. (19 Mei 2007). *Cc3 Reference Manual*. 27 November 2008. http://cmucam.org/attachment/wiki/Documentation/refman.pdf

Ian, T.Young. (1998). *Fundamental of Image Processing*. Netherlands:Delft University of Technology.

Inaki Navarro Oiza.(Mei 2004). *Digital Camera Interface.* 19 November 2008. http://www.robozes.com/inaki/dproject/report.pdf

NXP Philips. *LPC2104/2105/2106 Datasheet.* 24 Februari 2009. http://www.nxp.com/acrobat/datasheets/LPC2104_2105_2106_7.pdf

Rowe, Anthony (2003). *CMUCam2 Vision Sensor User Guide*. 27 November 2008.http://cmucam2.org/attachment/wiki/Documentation/CMUcam2_manual.pdf

Rowe, Anthony (22 September 2007). *CMUCam3 Datasheet*. 27 November 2008.http://cmucam.org/attachment/wiki/Documentation/CMUcam3_datasheet.pdf

Trevor, Martin. (2005). *The Insider's Guide To The Philips ARM7-Based Microcontrollers*. Coventory: Hitex (UK) Ltd.

Wijaya, Marvin. (2007). *Pengolahan Citra Digital Menggunakan Matlab*. Bandung: Informatika.

Winoto, Ardi. (2008). *Mikrokontroler AVR dan Pemrogramannya dengan Bahasa C pada WinAVR*. Bandung: Informatika.

LAMPIRAN

A.1 Program cmucam2.c

```
#include <cc3.h>
#include <cc3_ilp.h>
#include <cc3_color_track.h>
#include <cc3_color_info.h>
#include <math.h>
#include <stdbool.h>
#include <stdio.h>
#include <string.h>
#include <ctype.h>
#include <stdlib.h>
#include <jpeglib.h>
#include <cc3_math.h>
#include <cc3_hsv.h>

// Uncomment line below to reverse servo
direction for auto-servo and demo mode
#define SERVO_REVERSE_DIRECTION_PAN
#define SERVO_REVERSE_DIRECTION_TILT

#define SERIAL_BAUD_RATE
CC3_UART_RATE_115200

#define SERVO_MIN 0
#define SERVO_MID 128
#define SERVO_MAX 255
// Define a jitter guard such that more than
SERVO_GUARD pixels are required
// for the servo to move.
#define DEFAULT_COLOR 0
#define HSV_COLOR     1

static const unsigned int MAX_ARGS = 10;
static const unsigned int MAX_LINE = 128;
static const char VERSION_BANNER[] =
"CMUcam modified";

typedef struct {
  uint8_t pan_step, tilt_step;
  uint8_t pan_range_near, tilt_range_near;
  uint8_t pan_range_far, tilt_range_far;
  int16_t x;
  int16_t y;
  bool y_control;
  bool x_control;
  bool y_report;
  bool x_report;
} cmucam2_servo_t;

typedef enum {

RESET,TRACK_COLOR,SEND_FRAME,GET_
VERSION,SET_SERVO,COLOR_SPACE,
  RETURN,
  CMUCAM2_CMDS_COUNT
} cmucam2_command_t;

static const char
cmucam2_cmds[CMUCAM2_CMDS_CO
UNT][3] = {
  [RETURN] = "",[SET_SERVO] =
"SV",[SEND_FRAME] = "SF",
  [TRACK_COLOR] = "TC",[RESET] =
"RS",[GET_VERSION] =
"GV",[COLOR_SPACE] = "CS"
};
static void cmucam2_track_color
(cc3_track_pkt_t * t_pkt,bool auto_led,
                          cmucam2_servo_t *
servo_settings, uint8_t sw_color_space,
bool quiet);
static int32_t cmucam2_get_command
(cmucam2_command_t * cmd,
                               uint32_t arg_list[]);
static int32_t
cmucam2_get_command_raw
(cmucam2_command_t * cmd,
                                 uint32_t
arg_list[]);
static void print_ACK (void);
static void print_NCK (void);
static void print_prompt (void);
static void print_cr (void);
static void cmucam2_write_t_packet
(cc3_track_pkt_t * pkt,
                                 cmucam2_servo_t *
servo_settings);
static void cmucam2_send_image_direct
(bool auto_led,uint8_t sw_color_space);
static void raw_print (uint8_t val);
static bool packet_filter_flag;
static uint8_t t_pkt_mask;
//static uint8_t s_pkt_mask;
static bool raw_mode_output;
static bool raw_mode_no_confirmations;
static bool raw_mode_input;

int main (void)
{

  cmucam2_command_t command;
  int32_t val, n;
  uint32_t arg_list[MAX_ARGS];
  uint8_t sw_color_space;
```

```
  bool error,  auto_led;
  cc3_track_pkt_t t_pkt;
  cmucam2_servo_t servo_settings;
  cc3_uart_init (0,
                 SERIAL_BAUD_RATE,
                 CC3_UART_MODE_8N1,
CC3_UART_BINMODE_BINARY);
  val = setvbuf (stdout, NULL, _IONBF, 0);
  if (!cc3_camera_init ()) {
    cc3_led_set_state (0, true);
    exit (1);
  }
  servo_settings.x_control = true;
  servo_settings.y_control = true;
  servo_settings.x_report = true;
  servo_settings.y_report = true;
start:
  sw_color_space=DEFAULT_COLOR;
  auto_led = true;
  packet_filter_flag = false;
  t_pkt_mask = 3;
  t_pkt.track_invert = false;
  t_pkt.noise_filter = 0;

  t_pkt.lower_bound.channel[0] = 0;
  t_pkt.upper_bound.channel[0] = 0;
  t_pkt.lower_bound.channel[1] = 0;
  t_pkt.upper_bound.channel[1] = 0;
  t_pkt.lower_bound.channel[2] = 0;
  t_pkt.upper_bound.channel[2] = 0;

  raw_mode_output = false;
  raw_mode_no_confirmations = false;
  raw_mode_input = false;

  servo_settings.x = SERVO_MID;
  servo_settings.y = SERVO_MID;
  servo_settings.pan_range_far = 32;
  servo_settings.pan_range_near = 20;
  servo_settings.pan_step = 20;
  servo_settings.tilt_range_far = 32;
  servo_settings.tilt_range_near = 20;
  servo_settings.tilt_step = 20;

  cc3_camera_set_power_state (true);
  cc3_camera_init ();
  cc3_filesystem_init ();
  cc3_camera_set_resolution
(CC3_CAMERA_RESOLUTION_LOW);

  printf ("%s\r", VERSION_BANNER);

  cc3_gpio_set_mode (0,
CC3_GPIO_MODE_SERVO);
  cc3_gpio_set_mode (1,
CC3_GPIO_MODE_SERVO);
  cc3_gpio_set_servo_position (0, SERVO_MID);
  cc3_gpio_set_servo_position (1, SERVO_MID);
  cc3_pixbuf_frame_set_subsample
(CC3_SUBSAMPLE_NEAREST, 2, 1);

  while (true) {
    cc3_channel_t old_coi;

    print_prompt ();
    error = false;
   if (raw_mode_input) {
      n = cmucam2_get_command_raw
(&command, arg_list);
    }
    else {
      n = cmucam2_get_command
(&command, arg_list);
    }
    if (n != -1) {
      switch (command) {
      case RESET:
        if (n != 0) {
          error = true;
          break;
        }
        print_ACK ();
        print_cr ();
        goto start;
        break;

      case GET_VERSION:
        if (n != 0) {
          error = true;
          break;
        }
        print_ACK ();
        // no different in raw mode
        printf ("%s", VERSION_BANNER);
        break;


      case COLOR_SPACE:
        if (n != 1) {
          error = true;
          break;
        }
        print_ACK ();
        sw_color_space= arg_list[0];
        break;

      case SEND_FRAME:
        old_coi = cc3_g_pixbuf_frame.coi;
        if (n == 1) {
          if (arg_list[0] > 4) {
            error = true;
            break;
```

```
        }
        cc3_pixbuf_frame_set_coi (arg_list[0]);
       }
       else if (n > 1) {
        error = true;
        break;
       }
       print_ACK ();
       cmucam2_send_image_direct
(auto_led,sw_color_space);
       cc3_pixbuf_frame_set_coi (old_coi);
       break;

     case TRACK_COLOR:
       if (n != 0 && n != 6) {
        error = true;
        break;
       }
       print_ACK ();
       if (n == 6) {
        t_pkt.lower_bound.channel[0] = arg_list[0];
        t_pkt.upper_bound.channel[0] = arg_list[1];
        t_pkt.lower_bound.channel[1] = arg_list[2];
        t_pkt.upper_bound.channel[1] = arg_list[3];
        t_pkt.lower_bound.channel[2] = arg_list[4];
        t_pkt.upper_bound.channel[2] = arg_list[5];
       }
       cmucam2_track_color (&t_pkt,  auto_led,
//poll_mode,//buf_mode, p //line_mode,
                        &servo_settings,
sw_color_space, 0);

       break;

     case SET_SERVO:
       if (n != 2 || arg_list[0] > 4) {
        error = true;
        break;
       }
       print_ACK ();
       cc3_gpio_set_mode (arg_list[0],
CC3_GPIO_MODE_SERVO);
       cc3_gpio_set_servo_position (arg_list[0],
arg_list[1]);
       if (arg_list[0] == 0)
        servo_settings.x = arg_list[1];
       if (arg_list[0] == 1)
        servo_settings.y = arg_list[1];
       break;

     }
    }
    else
     error = true;
    if (error)
     print_NCK ();
  }
  return 0;
}
void cmucam2_send_image_direct (bool
auto_led, uint8_t sw_color_space)
{
  cc3_pixbuf_load ();
  uint32_t x, y;
  uint32_t size_x, size_y;
  uint8_t *row = cc3_malloc_rows (1);
  uint8_t num_channels =
cc3_g_pixbuf_frame.coi ==
CC3_CHANNEL_ALL ? 3 : 1;

  size_x = cc3_g_pixbuf_frame.width;
  size_y = cc3_g_pixbuf_frame.height;

  putchar (1);
  putchar (size_x);
  if (size_y > 255)
   size_y = 255;
  putchar (size_y);
  for (y = 0; y < size_y; y++) {
   putchar (2);
   if (auto_led) {
    if (y % 4 == 0)
     cc3_led_set_state (0, true);
    else
     cc3_led_set_state (0, false);
   }
   cc3_pixbuf_read_rows (row, 1);
   if(sw_color_space==HSV_COLOR &&
num_channels==CC3_CHANNEL_ALL )
cc3_rgb2hsv_row(row,size_x);
   for (x = 0; x < size_x * num_channels;
x++) {
    uint8_t p = row[x];
    // avoid confusion from FIFO
corruptions
    if (p < 16) {
     p = 16;
    }
    else if (p > 240) {
     p = 240;
    }
    putchar (p);
   }
  }
  putchar (3);

  cc3_led_set_state (0, false);
  free (row);
}

void cmucam2_track_color
(cc3_track_pkt_t * t_pkt,

                    bool auto_led, //bool
poll_mode,//bool selisih//buf_mode,bool
//int8_t line_mode,bool
```

```
                cmucam2_servo_t *
servo_settings,  uint8_t sw_color_space,
                bool quiet)
{
 cc3_image_t img;

 uint16_t x_mid, y_mid;
 int8_t t_step;

 bool prev_packet_empty = true;
 img.channels = 3;
 img.width = cc3_g_pixbuf_frame.width;
 img.height = 1;            // image will hold just 1
row for scanline processing
 img.pix = cc3_malloc_rows (1);
 if (img.pix == NULL) {
  return;
 }
 x_mid = cc3_g_pixbuf_frame.x0 +
(cc3_g_pixbuf_frame.width / 2);
 y_mid = cc3_g_pixbuf_frame.y0 +
(cc3_g_pixbuf_frame.height / 2);



 do {

   cc3_pixbuf_load ();

  if (cc3_track_color_scanline_start (t_pkt) != 0)
{

   while (cc3_pixbuf_read_rows (img.pix, 1)) {

        if(sw_color_space==HSV_COLOR &&
img.channels==CC3_CHANNEL_ALL)
{ cc3_rgb2hsv_row(img.pix,img.width);}

                cc3_track_color_scanline (&img,
t_pkt);}
            }

   cc3_track_color_scanline_finish (t_pkt);

   if (auto_led) {
    if (t_pkt->int_density > 2)
      cc3_led_set_state (0, true);
    else
      cc3_led_set_state (0, false);
   }
   if (t_pkt->int_density > 5 &&
servo_settings != NULL) {
     if (servo_settings->x_control) {
      t_step = 0;
      if (t_pkt->centroid_x > x_mid +
servo_settings->pan_range_far)
        t_step = servo_settings->pan_step;
      else if (t_pkt->centroid_x > x_mid +
servo_settings->pan_range_near)
        t_step = (servo_settings->pan_step
/ 2);

      if (t_pkt->centroid_x < x_mid -
servo_settings->pan_range_far)
        t_step = -(servo_settings-
>pan_step);
      else if (t_pkt->centroid_x < x_mid -
servo_settings->pan_range_near)
        t_step = -(servo_settings->pan_step
/ 2);

#ifdef
SERVO_REVERSE_DIRECTION_PAN
      servo_settings->x -= t_step;
#else
      servo_settings->x += t_step;
#endif
      t_step = 0;
      if (servo_settings->x >
SERVO_MAX)
        servo_settings->x =
SERVO_MAX;
      if (servo_settings->x <
SERVO_MIN)
        servo_settings->x = SERVO_MIN;
      cc3_gpio_set_servo_position (0,
servo_settings->x);
     }
     if (servo_settings->y_control) {
      if (t_pkt->centroid_y > y_mid +
servo_settings->tilt_range_far)
        t_step = servo_settings->tilt_step;
      else if (t_pkt->centroid_y >
            y_mid + servo_settings-
>tilt_range_near)
        t_step = servo_settings->tilt_step /
2;

      if (t_pkt->centroid_y < y_mid -
servo_settings->tilt_range_far)
        t_step = -(servo_settings-
>tilt_step);
      else if (t_pkt->centroid_y <
            y_mid - servo_settings-
>tilt_range_near)
        t_step = -(servo_settings->tilt_step
/ 2);

#ifdef
SERVO_REVERSE_DIRECTION_TILT
      servo_settings->y -= t_step;
#else
      servo_settings->y += t_step;
#endif
```

```
        if (servo_settings->y > SERVO_MAX)
          servo_settings->y = SERVO_MAX;
        if (servo_settings->y < SERVO_MIN)
          servo_settings->y = SERVO_MIN;
        cc3_gpio_set_servo_position (1,
servo_settings->y);
       }
     }

     if (!quiet) {
       if (!(packet_filter_flag &&
           t_pkt->num_pixels == 0 &&
prev_packet_empty)) {
         cmucam2_write_t_packet (t_pkt,
servo_settings);
       }

     prev_packet_empty = t_pkt->num_pixels ==
0;
    }
    else
     return;

    while (!cc3_uart_has_data (0)) {
     if (getchar () == '\r')
      break;
    }


  }while (true);
          free(img.pix);
          return ;
          }


void cmucam2_write_t_packet (cc3_track_pkt_t *
pkt,
                     cmucam2_servo_t *
servo_settings)
{

  // cap at 255
  if (pkt->centroid_x > 255)
    pkt->centroid_x = 255;
  if (pkt->centroid_y > 255)
    pkt->centroid_y = 255;
  if (pkt->x0 > 255)
    pkt->x0 = 255;
  if (pkt->x1 > 255)
    pkt->x1 = 255;
  if (pkt->y1 > 255)
    pkt->y1 = 255;
  if (pkt->y0 > 255)
    pkt->y0 = 255;
  if (pkt->num_pixels > 255)
    pkt->num_pixels = 255;
  if (pkt->int_density > 255)
    pkt->int_density = 255;
  // values to print
  uint8_t p[8];

  if (pkt->num_pixels == 0) {
    p[0] = p[1] = p[2] = p[3] = p[4] = p[5] =
p[6] = p[7] = 0;
  }
  else {
    p[0] = pkt->centroid_x;
    p[1] = pkt->centroid_y;
    p[2] = pkt->x0;
    p[3] = pkt->y0;
    p[4] = pkt->x1;
    p[5] = pkt->y1;
    p[6] = pkt->num_pixels;
    p[7] = pkt->int_density;
  }

  uint8_t mask = t_pkt_mask;
  if (raw_mode_output) {
    putchar (255);
  }
  printf ("T");

  // print out fields using mask
  for (int i = 0; i < 8; i++) {
    if (mask & 0x1) {
      if (raw_mode_output) {
        raw_print (p[i]);
      }
      else {
        printf (" %d", p[i]);
      }
    }
    mask >>= 1;
  }

  // print servo settings?
  if (servo_settings != NULL) {
    uint8_t sx = servo_settings->x;
    uint8_t sy = servo_settings->y;

    if (servo_settings->x_report) {
      if (raw_mode_output) {
        raw_print (sx);
      }
      else {
        printf (" %d", sx);
      }
    }
    if (servo_settings->y_report) {
      if (raw_mode_output) {
        raw_print (sy);
      }
      else {
        printf (" %d", sy);
      }
```

```
    }
  }
  print_cr ();
}

void print_ACK ()
{
  if (!raw_mode_no_confirmations)
    printf ("ACK\r");
}

void print_NCK ()
{
  if (!raw_mode_no_confirmations)
    printf ("NCK\r");
}

void print_prompt ()
{
  printf (":");
}

void print_cr ()
{
  if (!raw_mode_output) {
    printf ("\r");
  }
}

int32_t cmucam2_get_command
(cmucam2_command_t * cmd, uint32_t * arg_list)
{
  char line_buf[MAX_LINE];
  int c;
  char *token;
  bool fail = false;
  uint32_t length, argc;
  uint32_t i;
  length = 0;
  c = 0;
  while (c != '\r') {
    c = getchar ();
    if (length < (MAX_LINE - 1) && c != EOF) {
      line_buf[length] = c;
      length++;
    }
    else {
      // too long or EOF
      return -1;
    }
  }
  // null terminate
  line_buf[length] = '\0';
  // check for empty command
  if (line_buf[0] == '\r' || line_buf[0] == '\n') {
    *cmd = RETURN;
    return 0;
  }
  // start looking for command
  token = strtok (line_buf, " \r\n");

  if (token == NULL) {
    // no command ?
    return -1;
  }
  // get name of the command
  for (i = 0; i < strlen (token); i++) {
    token[i] = toupper (token[i]);
  }
  // do lookup of command
  fail = true;
  for (i = 0; i <
CMUCAM2_CMDS_COUNT; i++) {
    if (strcmp (token, cmucam2_cmds[i]) ==
0) {
      fail = false;
      *cmd = i;
      break;
    }
  }
  if (fail) {
    return -1;
  }
  // now get the arguments
  argc = 0;
  while (true) {
    // extract string from string sequence
    token = strtok (NULL, " \r\n");
    // check if there is nothing else to extract
    if (token == NULL) {
      // printf("Tokenizing complete\n");
      return argc;
    }

    // make sure the argument is fully
numeric
    for (i = 0; i < strlen (token); i++) {
      if (!isdigit (token[i]))
        return -1;
    }
    // we have a valid token, add it
    arg_list[argc] = atoi (token);
    argc++;
  }
  return -1;
}

int32_t cmucam2_get_command_raw
(cmucam2_command_t * cmd, uint32_t *
arg_list)
{
  bool fail;
  int c;
  unsigned int i;
  uint32_t argc;
```

```
  char cmd_str[3];
  cmd_str[2] = '\0';
  // read characters
  for (i = 0; i < 2; i++) {
    c = getchar ();
    if (c == EOF) {
      return -1;
    }
    cmd_str[i] = c;
  }
  // do lookup of command
  fail = true;
  for (i = 0; i < CMUCAM2_CMDS_COUNT;
i++) {
    if (strcmp (cmd_str, cmucam2_cmds[i]) == 0) {
      fail = false;
      *cmd = i;
      break;
    }
  }
  if (fail) {
    return -1;
  }
  // read argc
  c = getchar ();
  if (c == EOF) {
    return -1;
  }
  argc = c;
  if (argc > MAX_ARGS) {
    return -1;
  }
  // read args
  for (i = 0; i < argc; i++) {
    c = getchar ();
    if (c == EOF) {
      return -1;
    }
    arg_list[i] = toupper (c);
  }
  // done
  return argc;
}

void raw_print (uint8_t val)
{
  if (val == 255) {
    putchar (254);              // avoid confusion
  }
  else {
    putchar (val);
  }
}
```

A.2 Program Atmega16.c

```
/*****************************************************
Chip type           : ATmega16
Program type        : Application
Clock frequency     : 11,059200 MHz
Memory model        : Small
External RAM size   : 0
Data Stack size     : 256
Aplikasi input parameter warna untuk sistem tracking
warna CMUcam3
*****************************************************/
#include <mega16.h>
#include <delay.h>
// Standard Input/Output functions
#include <stdio.h>
#include <string.h>

// Alphanumeric LCD Module functions
#asm
   .equ __lcd_port=0x15 ;PORTC
#endasm
#include <lcd.h>

#define RXB8 1
#define TXB8 0
#define UPE 2
#define OVR 3
#define FE 4
#define UDRE 5
#define RXC 7

#define FRAMING_ERROR (1<<FE)
#define PARITY_ERROR (1<<UPE)
#define DATA_OVERRUN (1<<OVR)
#define DATA_REGISTER_EMPTY
(1<<UDRE)
#define RX_COMPLETE (1<<RXC)

// USART Receiver buffer
#define RX_BUFFER_SIZE 20
//int RX_BUFFER_SIZE2= strlen(rx_buffer[]-
1);
char rx_buffer[RX_BUFFER_SIZE];
//char buffer1[20];
//int i;
#if RX_BUFFER_SIZE<256
unsigned char
rx_wr_index,rx_rd_index,rx_counter;
#else
unsigned int
rx_wr_index,rx_rd_index,rx_counter;
#endif

// This flag is set on USART Receiver buffer
overflow
bit rx_buffer_overflow;

// USART Receiver interrupt service routine
interrupt [USART_RXC] void
usart_rx_isr(void)
{
//delay_ms(500);

char status,data;
status=UCSRA;
data=UDR;

if ((status & (FRAMING_ERROR |
PARITY_ERROR |
DATA_OVERRUN))==0)
  {
  rx_buffer[rx_wr_index]=data;
  if (++rx_wr_index ==
RX_BUFFER_SIZE) rx_wr_index=0;
  if (++rx_counter == RX_BUFFER_SIZE)
    {
    rx_counter=0;
    rx_buffer_overflow=1;
    };
  };
if(data=='\r'){lcd_clear();
      lcd_puts(rx_buffer);
      delay_ms(100);
      rx_wr_index =0;
      rx_counter=0;
      rx_buffer_overflow=1;
      memset(rx_buffer,0,20);
      }
}

#ifndef _DEBUG_TERMINAL_IO_
// Get a character from the USART Receiver
buffer
#define _ALTERNATE_GETCHAR_
#pragma used+
char getchar(void)
{
char data;
while (rx_counter==0);
data=rx_buffer[rx_rd_index];
```

```
if (++rx_rd_index == RX_BUFFER_SIZE)
rx_rd_index=0;
#asm("cli")
--rx_counter;
#asm("sei")
return data;
}
#pragma used-
#endif


// Declare your global variables here
unsigned int xu;
unsigned char w;

void uart_puts(char*str)
{       unsigned char x3=0;
        while (str[x3]!=0)
{putchar(str[x3]);
        x3++;}
}

void in_string(char*in)
{ in[xu]=w;
xu++;}

void main(void)
{
   char buf[24];
   xu=3;
// USART initialization
// Communication Parameters: 8 Data, 1 Stop,
No Parity
// USART Receiver: On
// USART Transmitter: On
// USART Mode: Asynchronous
// USART Baud Rate: 115200
UCSRA=0x00;
UCSRB=0x98;
UCSRC=0x86;
UBRRH=0x00;
UBRRL=0x05;
// LCD module initialization
lcd_init(20);
// Global enable interrupts
#asm("sei")
   lcd_clear();
   lcd_putsf("== Tracking Warna ==");
   delay_ms(2000);
   lcd_clear();
   lcd_putsf("masukkan parameter");
   lcd_gotoxy(0,1);
   lcd_putsf("  min= 0, max= 255");
   delay_ms(2000);
   lcd_clear();
   lcd_putsf("TC R- R+ G- G+ B- B+");

   DDRA=0xf0;
   while (1){

   PORTA=0b01111111;delay_ms(1);
   if(PINA.0==0){delay_ms(50); w='S'; };
   if(PINA.1==0){delay_ms(50); w='3'; };
   if(PINA.2==0){delay_ms(50); w='2'; };
   if(PINA.3==0){delay_ms(50); w='1'; };
        PORTA=0b10111111;delay_ms(1);
        if(PINA.0==0){delay_ms(50);
w='T'; };
        if(PINA.1==0){delay_ms(50);
w='6'; };
        if(PINA.2==0){delay_ms(50);
w='5'; };
        if(PINA.3==0){delay_ms(50);
w='4'; };
        PORTA=0b11011111;delay_ms(1);
        if(PINA.0==0){delay_ms(50);
w='C'; };
        if(PINA.1==0){delay_ms(50);
w='9'; };
        if(PINA.2==0){delay_ms(50);
w='8'; };
        if(PINA.3==0){delay_ms(50);
w='7'; };
        PORTA=0b11101111;delay_ms(1);
        if(PINA.0==0){delay_ms(50);
w='M'; };
        if(PINA.1==0){delay_ms(50); w='
'; };
        if(PINA.2==0){delay_ms(50);
w='0'; };
        if(PINA.3==0){delay_ms(50);
w='\r'; };
        delay_ms(200);

      if(w!=0x00)
      {
        buf[0]='T',buf[1]='C',buf[2]=' ';
        in_string(buf);
        lcd_gotoxy(0,1);
        lcd_puts(buf);
        delay_ms(100);
      if (w=='\r')
      {

      lcd_clear();
      lcd_putsf("sending your
command.....");
      delay_ms(1500);
      uart_puts(buf);
      delay_ms(1500);
      lcd_clear();
      lcd_putsf("command has sent");
      delay_ms(500);
      lcd_clear();
```

```
        lcd_putsf("your color \n is being
tracked...");
        delay_ms(1500);
        lcd_clear();

        do{
        delay_ms(1000);
        } while(1);
        }
           w=0x00;
         }

    }
}
```

A.3 Blok Diagram LabVIEW

Diagram penampil grafik:

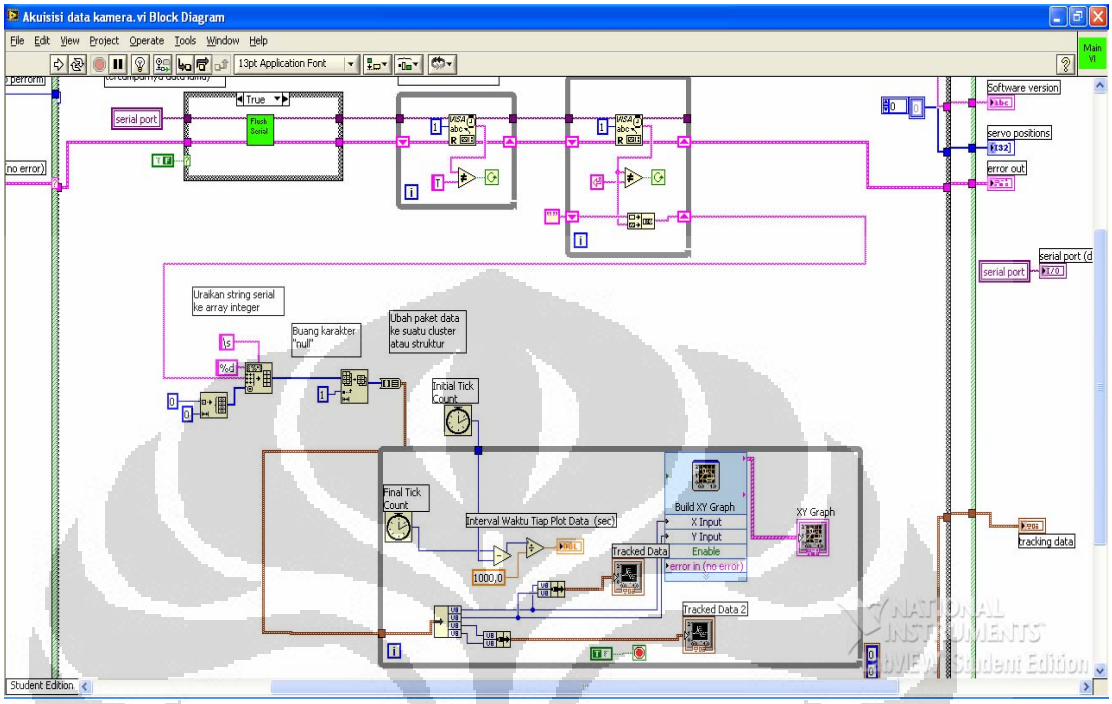


Diagram penampil gambar:

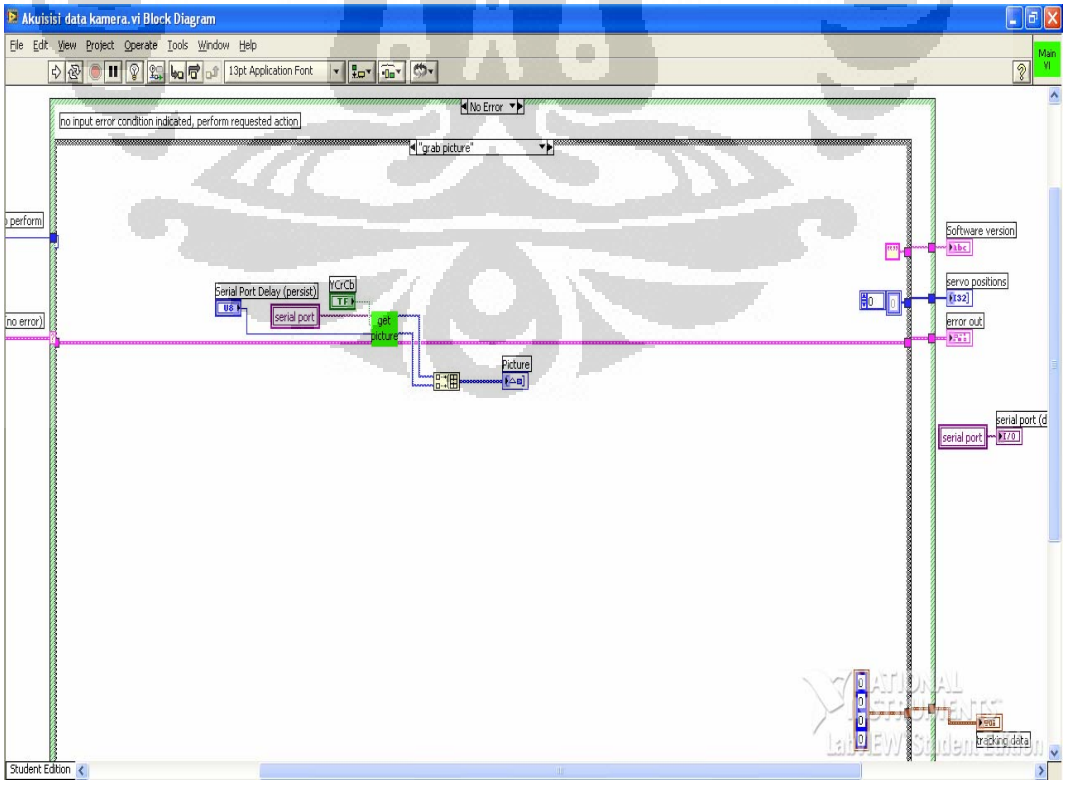
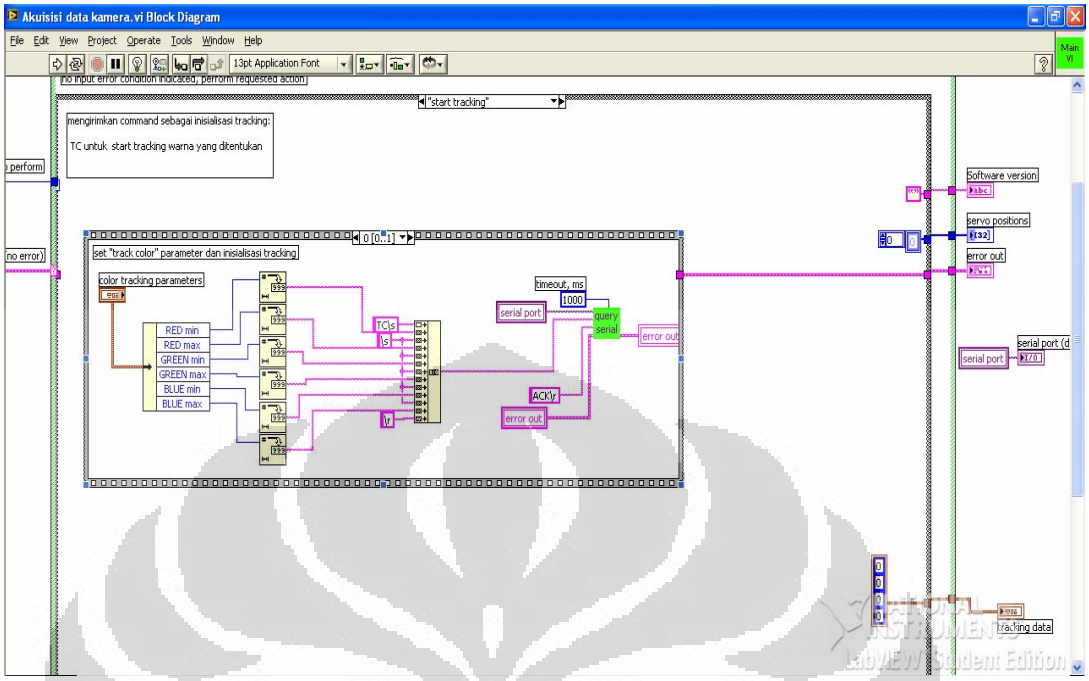


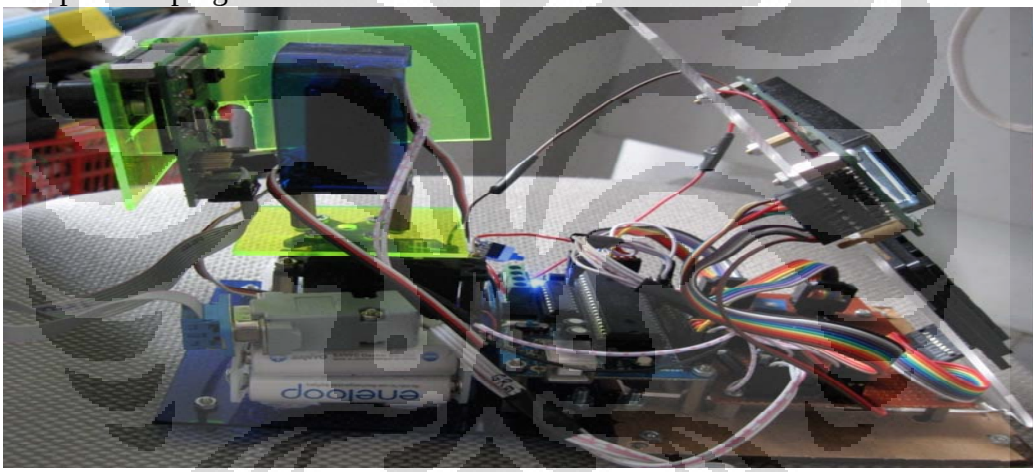
Diagram memulai proses *tracking* :



A.4 Gambar Alat

Tampak Belakang

Tampak Samping



Tampak Depan