TUGAS AKHIR – TE 141599

PENENTUAN ARAH SUMBER SUARA DENGAN METODE *INTERAURAL TIME DIFFERENCE* MENGGUNAKAN MIKROKONTROLER STM32F4

Mohamad Asfari
NRP. 2214105034

Dosen Pembimbing
Dr. Muhammad Rivai, ST., MT.
Ir. Tasripan, MT.

DEPARTEMEN TEKNIK ELEKTRO
Fakultas Teknologi Elektro
Institut Teknologi Sepuluh Nopember
Surabaya, 2017

Halaman Ini Sengaja Dikosongkan

TUGAS AKHIR – TE 141599

PENENTUAN ARAH SUMBER SUARA DENGAN METODE *INTERAURAL TIME DIFFERENCE* MENGGUNAKAN MIKROKONTROLER STM32F4

Mohamad Asfari
NRP 2214105034

Dosen Pembimbing
Dr. Muhammad Rivai, ST., MT.
Ir. Tasripan, MT.

DEPARTEMEN TEKNIK ELEKTRO
Fakultas Teknologi Elektro
Institut Teknologi Sepuluh Nopember
Surabaya 2017

Halaman Ini Sengaja Dikosongkan

FINAL PROJECT – TE 141599

SOUND SOURCE LOCALIZATION BASED ON INTERAURAL TIME DIFFERENCE METHOD USING MICROCONTROLLER STM32F4

Mohamad Asfari
NRP 2214105034

Advisor Lecturer
Dr. Muhammad Rivai, ST., MT.
Ir. Tasripan, MT.

DEPARTMENT OF ELECTRICAL ENGINEERING
Faculty of Electrical Technology
Institut Teknologi Sepuluh Nopember
Surabaya 2017

Halaman Ini Sengaja Dikosongkan

PERNYATAAN KEASLIAN
TUGAS AKHIR

Dengan ini saya menyatakan bahwa isi sebagian maupun keseluruhan Tugas Akhir saya dengan judul "**Penentuan Arah Sumber Suara dengan Metode *Interaural Time Difference* menggunakan Mikrontroler STM32F4"** adalah benar-benar hasil karya intelektual mandiri, diselesaikan tanpa menggunakan bahan-bahan yang tidak diijinkan dan bukan merupakan karya pihak lain yang saya akui sebagai karya sendiri. Semua referensi yang dikutip maupun dirujuk telah ditulis secara lengkap pada daftar pustaka.

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

Surabaya, Juli 2017

Mohamad Asfari
NRP. 2214105034

Halaman Ini Sengaja Dikosongkan

PENENTUAN ARAH SUMBER SUARA DENGAN METODE *INTERAURAL TIME DIFFERENCE* MENGGUNAKAN MIKROKONTROLER STM32F4

TUGAS AKHIR

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

Menyetujui :

Dosen Pembimbing I	Dosen Pembimbing II
Dr. Muhammad Rivai, ST., MT.	**Ir. Tasripan, MT.**
NIP. 1969 04 26 1994 03 1003	**NIP. 1962 04 18 1990 03 1004**

SURABAYA
JULI, 2017

Halaman Ini Sengaja Dikosongkan

PENENTUAN ARAH SUMBER SUARA DENGAN METODE *INTERAURAL TIME DIFFERENCE* MENGGUNAKAN MIKROKONTROLER STM32F4

Nama : **Mohamad Asfari**
Dosen Pembimbing 1 : **Dr. Muhammad Rivai, ST., MT.**
Dosen Pembimbing 2 : **Ir. Tasripan, MT.**

ABSTRAK

Sistem pendengaran manusia dapat mengestimasi lokasi sumber suara dari pengamatan sinyal suara binaural dengan akurasi yang cukup tinggi. Manusia dapat mengenali lokasi suara secara 3 dimensi dan bahkan dapat mengenali keberadaan sumber suara pada posisi belakang dimana tidak terdapat informasi visual. Sudah banyak dilakukan penelitian mengenai pengenalan lokasi sumber suara menggunakan beberapa *array* mikrofon yang memakai lebih dari 3 buah mikrofon, akan tetapi penelitian sistem pengenalan lokasi sumber suara yang menggunakan 2 buah mikrofon seperti pendengaran manusia masih jarang dilakukan.

Para penelitian ini merancang suatu sistem pendeteksi sumber suara dengan menggunakan 2 buah mikrofon dan mikrokontroler STM32F4 sebagai prosesor utama. Metode *Interaural Time Difference* digunakan untuk menentukan arah dengan cara membandingkan perbedaan waktu sampai sebuah suara yang dipancarkan oleh sumber suara.

Berdasarkan hasil pengujian, sistem ini dapat menentukan arah sumber suara dari nilai antara 0^0 dan 180^o dengan jarak sumber suara di bawah 40 cm dan intensitas suara sebesar 73 dB. Tingkat kesalahan pada penentuan arah tersebut adalah sebesar 38,31%. Diharapkan alat ini dapat diaplikasikan dalam sistem pengusir hama burung untuk menentukan arah suara pengusir.

Kata Kunci: Arah Sumber Suara, *Interaural Time Difference*, STM32F4

Halaman Ini Sengaja Dikosongkan

SOUND SOURCE LOCALIZATION BASED ON INTERAURAL TIME DIFFERENCE METHOD USING MICROCONTROLLER STM32F4

Name	**: Mohamad Asfari**
1st Advisor	**: Dr. Muhammad Rivai, ST., MT.**
2nd Advisor	**: Ir. Tasripan, MT.**

ABSTRACT

Human audition system is capable of estimating the location of sound source from observed binaural audio signals with moderate accuracy. This hearing ability is referred to as the sound source localization. Humans can perform sound localization in three dimensional space and can even recognize the existence of sound source behind their back where no visual information is available. Numerous works have been done to implement audio perception capability to robots, mainly on systems which use multiple microphone array containing more than three microphones. Developments of robotic audition systems which uses only two microphones like humans do, however, are less active as compared to array signal processing.

This research has developed a sound source localization system using 2 microphones and microcontroller STM32F4 as main processor unit. The Interaural Time Difference method is used to locate the sound source by comparing the difference in the arrival time between each microphones from the sound source.

Based on the testing result, the sound localization can operate at 0^o to 180^o range with the sound source at below 40 cm distance and the sound intensity at above 73 dB. This system has an average of error at 31,87%. This system is expected to be able to be implemented in the bird's pest repellent system to determine the repellent's direction.

Keywords: Interaural Time Difference, Sound Localization, STM32F4

Halaman Ini Sengaja Dikosongkan

KATA PENGANTAR

Segala puji syukur penulis panjatkan atas kehadirat Allah SWT yang selalu memberikan rahmat serta hidayah-Nya sehingga Tugas Akhir ini dapat terselesaikan dengan baik. Shalawat serta salam selalu dilimpahkan kepada Rasulullah Muhammad SAW, keluarga, sahabat, dan umat muslim yang senantiasa meneladani beliau.

Pada kesempatan ini penulis ingin mengucapkan ucapan terimakasih yang sebesar- besarnya kepada beberapa pihak yang telah memberikan dukungan selama proses pengerjaan tugas akhir ini, antara lain:

1. Dr. Muhammad Rivai, ST., MT. dan Ir. Tasripan, MT. selaku dosen pembimbing pertama atas bimbingan, inspirasi, pengarahan, dan motivasi yang diberikan selama pengerjaan penelitian Tugas Akhir ini.
2. Dr. Tri Arief Sardjono, ST., MT., Ir. Siti Halimah Baki, MT., Rahmad Setiawan, ST., MT. dan Astria Nur Irfansyah, ST., M.Eng. selaku dosen penguji atas pengarahan dan masukan yang diberikan selama pengerjaan buku laporan penelitian Tugas Akhir ini.
3. Dr. Eng. Ardyono Priyadi, ST., M.Eng. Selaku Ketua Departemen Teknik Elektro Fakultas Teknologi Elektro
4. Seluruh dosen bidang studi Elektronika Jurusan Teknik Elektro Fakultas Teknologi Elektro ITS.
5. Keluarga penulis ayahanda Hoetomo Fadli, ibunda Hilyah, beserta seluruh keluarga yang selalu memberikan doa, dukungan, motivasi, semangat, perhatian dan kasih sayangnya.
6. Tri Suliswanto yang bersedia membantu memberi pengarahan serta mengajarkan dasar-dasar dalam pemrograman.
7. Seluruh rekan-rekan yang telah banyak membantu dalam penyelesaian Tugas Akhir ini.

Penulis menyadari bahwa dalam Tugas Akhir ini terdapat banyak kekurangan. Akhir kata semoga melalui tulisan ini dapat bermanfaat dan dapat berbagi ilmu bagi pembacanya. Aamiin.

Surabaya, Juli 2017

Penulis

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

Halaman Ini Sengaja Dikosongkan

BAB I
PENDAHULUAN

1.1 Latar Belakang

Sistem pendengaran manusia dapat mengestimasi lokasi sumber suara dari pengamatan sinyal suara binaural dengan akurasi yang cukup. Manusia dapat mengenali lokasi suara secara 3 dimensi dan bahkan dapat mengenali keberadaan sumber suara pada posisi belakang dimana tidak terdapat informasi visual. Sudah banyak dilakukan penelitian mengenai pengenalan lokasi sumber suara menggunakan beberapa *array* mikrofon yang memakai lebih dari 3 buah mikrofon, akan tetapi penelitian sistem pengenalan lokasi sumber suara yang menggunakan 2 buah mikrofon seperti pendengaran manusia masih jarang dilakukan [1].

Penelitian pada sistem lokalisasi suara pada manusia sudah dilakukan selama beberapa dekade dan mekanisme dari lokalisasi sumber suara pada manusia sudah diketahui, dari penelitian itu terlihat bahwa manusia menggunakan beberapa petunjuk untuk mendapatkan lokasi sumber suara yaitu membedakan level atau intensitas suara dan waktu sampai suara dari sumber ke telinga [2].

Pada tugas akhir ini dirancang suatu sistem untuk menghitung arah sudut dari sumber suara menggunakan metode *Interaural Time Difference* (ITD) dengan cara menangkap suara dari sumber dengan 2 buah mikrofon lalu membedakan waktu sampai suara dari sumber menuju kedua mikrofon dengan posisi yang berbeda.

1.2 Rumusan Masalah

Sehubungan dengan latar belakang yang telah dijelaskan sebelumnya, terdapat beberapa masalah yang akan dibahas antara lain sebagai berikut :

1. Penentuan perbedaan waktu datangnya suara dari sumber suara menuju mikrofon dengan jarak yang berbeda.
2. Penentuan arah sumber suara.

1.3 Batasan Masalah

Batasan masalah yang akan dibahas dalam Tugas Akhir ini adalah :

1. Arah sumber suara yang dideteksi memiliki sudut 0 – 180 derajat.
2. Sumber suara yang digunakan adalah suara stabil dengan frekuensi tertentu.
3. Mikrofon yang digunakan berjumlah 2 buah.

1.4 Tujuan

Tujuan dari tugas akhir ini adalah sebagai berikut:

1. Penggunaan metode *cross-correlation* untuk mendapatkan perbedaan waktu datangnya suara dari sumber suara.
2. Mendapatkan arah lokasi sumber suara menggunakan rumus *Interaural Time Difference*.

1.5 Metodologi

Metodologi yang digunakan dalam penyusunan tugas akhir ini sebagai berikut :

1. Studi literatur dan diskusi, yaitu studi yang bersumber pada jurnal-jurnal, buku referensi, *paper* dan *datasheet* komponen yang digunakan dalam Tugas Akhir ini serta berdiskusi dengan dosen pembimbing dan narasumber.
2. Perancangan dan realisasi alat, pada tahap ini dimulai dari perancangan mikrofon sebagai sensor suara dan ARM sebagai mikrokontroler untuk memerhitungkan arah sudut dari sumber suara
3. Tahap pengujian sistem dan analisa, pada tahapan ini dimulai dengan memberi suara pada 2 mikrofon dengan jarak – jarak tertentu dan menghitung perbedaan waktu sampai dari sumber suara ke mikrofon, kemudian menggunakan perbedaan waktu tersebut untuk menghitung arah sudut dari sumber suara.
4. Pembuatan laporan mengacu pada perancangan dan realisasi alat, serta hasil dari pengujian alat.

1.6 Sistematika Penulisan

Sistematika penulisan pada tugas akhir ini dibagi menjadi beberapa bab dengan rincian :

BAB I : PENDAHULUAN

Berisi tentang latar belakang, rumusan masalah, batasan masalah, tujuan, manfaat, metodologi dan sistematika penulisan.

BAB II : TEORI PENUNJANG

Berisi tentang teori *Sound Localization*, penjelasan mikrofon, *amplifier*, motor servo, serta STM32F4.

BAB III : PERANCANGAN DAN REALISASI ALAT

Berisi tentang tahap-tahap perancangan perangkat hardware dan software.

BAB IV : PENGUJIAN DAN ANALISA SISTEM

Bab ini membahas mengenai pengujian dari rancangan yang telah diimplementasikan pada sistem pendeteksi arah sumber suara.

BAB V : PENUTUP

Berisi tentang kesimpulan dan saran yang diperoleh dalam Tugas Akhir ini.

1.7 Relevansi

Adapun manfaat yang diharapkan dengan penelitian ini adalah sistem pendeteksi sumber suara yang lebih mendekati sistem pendengaran pada manusia yaitu dengan menggunakan 2 buah mikrofon serta membuktikan teori metode ITD dengan menggunakan mikrokontroler STM32F4.

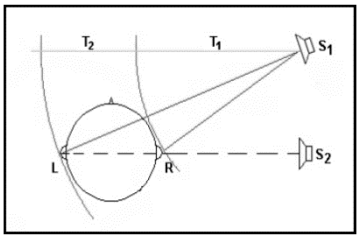
Hasil yang dicapai diharapkan dapat menjadi salah satu referensi dalam pengembangan sistem pendeteksi arah sumber suara.

Halaman Ini Sengaja Dikosongkan

BAB II
TEORI PENUNJANG

2.1 *Sound Localization*

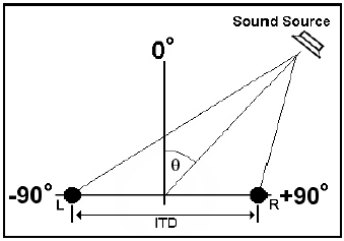
Sound localization adalah proses menentukan lokasi sumber suara berdasarkan beberapa observasi yang diterima pada sinyal suara[3]. Terdapat beberapa metode dalam proses penentuan lokasi pada *sound localization*, salah satunya adalah metode *Time Difference of Arrival* (TDOA). Metode ini terinspirasi dari cara kerja kedua telinga manusia seperti yang diilustrasikan pada gambar 2.1 dalam menerima suara dari posisi tertentu, menggunakan 2 buah atau lebih mikrofon sebagai penerima input suara lalu membedakan waktu datangnya suara dari sumber menuju masing-masing mikrofon. Jika 2 buah mikrofon dipasang pada jarak tertentu, dan saat sumber suara berjarak lebih dekat pada 1 mikrofon daripada mikrofon lainnya, akan terdapat perbedaan waktu sampai antara mikrofon yang lebih dekat dengan yang lebih jauh, perbedaan ini yang selanjutnya digunakan untuk perhitungan sudut arah datangnya sumber suara. Metode ini dapat juga disebut dengan metode *Interaural Time Difference* (ITD)[3].



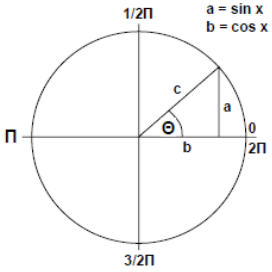
Gambar 2.1 Ilustrasi perbedaan waktu sampai (TDOA) antara sumber suara dengan telinga manusia sebagai sensor suara

Untuk menentukan lokasi sumber suara pada lapangan dibutuhkan perhitungan azimut antara sumber suara dengan sensor suara. Azimut yang dihasilkan disini mempresentasikan perbandingan lokasi antara sumber suara dengan sensor suara dengan sudut 0^o sebagai posisi sumber suara tepat di depan sensor suara sesuai dengan yang ditampilkan pada gambar 2.1. Sudut ini didapat dari perhitungan TDOA dari sinyal yang

diterima oleh kedua sensor suara. Gambar 2.2 menunjukkan orientasi sudut posisi depan sumber suara terhadap sensor suara.



Gambar 2.2 Perhitungan sudut untuk menentukan azimut. Kiri dari 0^o negatif dan kanan dari 0^o positif



Gambar 2.3 Perhitungan sudut menggunakan prinsip trigonometri

Pencarian azimut pada metode ITD ini pada dasarnya menggunakan prinsip trigonometri dengan nilai c pada gambar 2.3 sebagai jarak antara kedua mikrofon dan nilai a sebagai jarak yang didapat dari kecepatan suara yang dikalikan dengan waktu jeda pada kedua mikrofon.

Nilai a disini merupakan hasil kali dari kecepatan suara dengan waktu jeda dari kedua mikrofon sesuai dengan persamaan (2.1).

$$length = t \times V_{sound} = (\Delta \times \sigma) \times V_{sound} \quad (2.1)$$

Nilai t pada persamaan (2.1) adalah waktu jeda yang digunakan untuk mencari nilai sudut dimana Δ adalah frekuensi sampling dan σ adalah hasil dari *delay* yang dihitung oleh metode *cross-correlation*, nilai V_{sound}

disini adalah kecepatan rambat suara pada udara sebesar 384 m/s pada suhu sebesar 24º C. Setelah itu didapat nilai a dan c yang hasilnya dapat digunakan untuk menghitung sudut arah sumber suara dengan hukum sinus sesuai pada persamaan (2.3) dan persamaan (2.4).

$$\sin\theta = \frac{a}{c} \therefore \theta = \sin^{-1}\frac{a}{c} \tag{2.3}$$

$$\theta = \sin^{-1}\frac{(\Delta \times \sigma) \times V_{sound}}{c} \tag{2.4}$$

2.1.1 *Cross-correlation*

Salah satu variabel yang digunakan untuk mencari nilai σ pada persamaan (2.1) didapat menggunakan metode *cross-correlation*. Metode ini adalah cara untuk mencari perbedaan dari kedua sinyal, pada kasus ini yang dicari adalah perbedaan fasanya sehingga didapat jeda atau perbedaan waktu sampai sumber suara menuju kedua mikrofon.

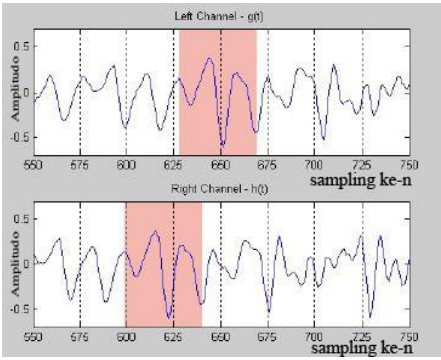
$$Corr(g,h)_j(t) \equiv \sum_{k=0}^{N-1} g_{j+k} h_k \tag{2.5}$$

Fungsi *cross-correlation* disini didefinisikan pada persamaan (2.5), fungsi korelasi *Corr(g,h)(t)* akan memiliki nilai maksimum pada *(t_n)* saat fungsi *g(t)* digeser seiring waktu melalui persamaan *h(t)*.

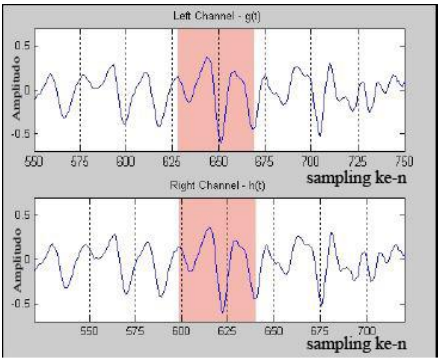
Cross-correlation disini digunakan untuk membandingkan kesamaan 2 buah vektor A dan B (dimana keduanya memiliki nilai untuk sinyal *g(t)* dan *h(t)*). Vektor A dan B disini merupakan input dari *cross-correlation* yang mempresentasikan sinyal suara yang diterima pada mikrofon. Sinyal tersebut lalu akan digeser melalui satu sama lain pada semua poin untuk menghasilkan vektor C dengan panjang sesuai dengan persamaan (2.6).

$$length(\boldsymbol{C}) = \big(length(\boldsymbol{A}) + length(\boldsymbol{B})\big) - 1 \tag{2.6}$$

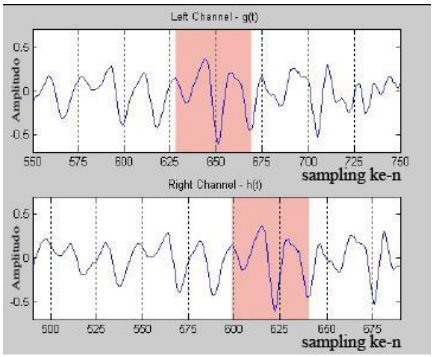
Nilai maksimun dari hasil vektor C menunjukkan korelasi maksimum diantara kedua sinyal *g(t)* dan *h(t)* dengan waktu *delay* σ. Pada plot dari vektor C akan didapat 2 axis x dan y dimana axis y mempresentasikan hasil perkalian dari nilai vektor A dan B dengan waktu *delay* σ, sedangkan axis x menunjukkan waktu atau *step* pada algoritma *cross-correlation* atau disebut juga waktu *delay* σ.



Gambar 2.4 Contoh plot kedua sinyal g(t) dan h(t) saat sebelum berpapasan

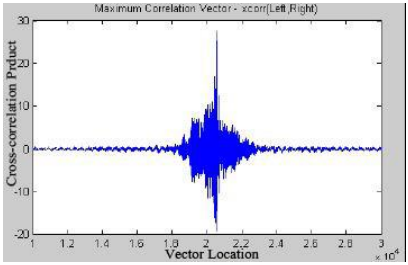


Gambar 2.5 Contoh plot kedua sinyal g(t) dan h(t) saat berpapasan



Gambar 2.6 Contoh plot kedua sinyal g(t) dan h(t) setelah berpapasan

Sebagai contoh pada gambar 2.4 terdapat 2 sinyal dengan bentuk sama dan fasa yang berbeda, lalu salah satu sinyal digeser sehingga fasa kedua sinyal berpapasan seperti gambar 2.5, setelah itu menjauh lagi seperti gambar 2.6. Dari gambar tersebut didapat hasil *cross-correlation* yang ditunjukkan oleh gambar 2.7 dengan waktu sesuai dengan pergeseran kedua sinyal, pada saat kedua sinyal belum berpapasan terlihat hasil dari *cross-correlation* bernilai rendah atau tidak terdapat kesamaan, setelah itu salah satu sinyal digeser sehingga kedua sinyal berpapasan dan terjadi kenaikan nilai pada hasil *cross-correlation* lalu mengecil lagi setelah sinyal terus bergeser dan perbedaannya membesar kembali[3].

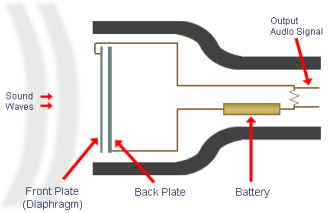


Gambar 2.7 Plot hasil korelasi kedua sinyal

2.2 Mikrofon

Mikrofon adalah alat yang digunakan untuk mengubah besaran suara menjadi besaran listrik. Mikrofon memiliki banyak macam, seperti: mikrofon kristal, mikrofon elektrodinamik, dan mikrofon kondensor seperti yang ditunjukkan pada Gambar 2.2[3].

Mikrofon kondensor merupakan komponen elektronik yang menyimpan energi dalam medan elektrostatik. Mikrofon jenis ini menggunakan bahan dasar berupa kapasitor. Diperlukan catu daya bahan berupa baterai untuk pengoperasian mikrofon jenis ini.



Gambar 2.8 Mikrofon Kondensor

Dalam mikrofon kondensor terdapat dua keping plat yang memiliki beda tegangan. Salah satu dari keping plat terbuat dari bahan yang ringan sehingga bertindak sebagai diafragma. Fungsi dari diafragma adalah mendeteksi getaran suara dari lingkungan dengan beresonansi dengan getaran tersebut. Ketika beresonansi dengan getaran suara, jarak antar dua keping plat berubah-ubah sehingga menghasilkan nilai kapasitansi yang berubah-ubah. Nilai kapasitansi berbanding lurus dengan jumlah muatan pada kapasitor. Jumlah muatan berbanding lurus dengan besar arus yang dihasilkan oleh mikrofon kondensor. Pada bagian keluaran mikrofon terdapat resistor yang menyebabkan tegangan berubah ketika mendapatkan perubahan arus.

Mikrofon kondensor lebih sensitif terhadap suara jika dibandingkan dengan mikrofon dinamis. Akibat dari sensitifitas yang tinggi mikrofon kondensor rentan terhadap derau[4].

2.3 Mikrokontroler STM32F4

Mikrokontroler STM32F4 termasuk dalam keluarga mikrokontroler 32 bit dengan arsitektur ARM. Mikrokontroler jenis STM32F4 merupakan seri pertama dalam kelompok STM32 dengan ARM core-M4F. Seri F4 juga merupakan seri pertama dari STM32 yang dapat melakukan *digital signal processing* (DSP). Seri STM32F4 memiliki pin-pin yang sama dengan pin-pin STM32F2 dengan penambahan kecepatan clock yang lebih tinggi dari STM32F2, 64K CCM static RAM, full duplex I2S, dan memiliki kecepatan konversi ADC.

Berdasarkan chip STM32F407VGT6, *board* mikrokontroler ini memiliki *ST-LINK/V2 embedded debug tool*, accelerometer digital, mikrofon digital, satu *audio DAC* dengan driver speaker *class* D terintegrasi, beberapa LED dan *push button*s dan sebuah konektor USB OTG micro-AB.

Ciri khas mikrokontroler jenis STM32F4 dapat diketahui berdasarkan beberapa spesifikasi sebagai berikut:

a. Core : ARM Cortex-M4F dengan *clock* 84/168/180 MHz.
b. Memori
   Static RAM sampai 192KB, 64KB *core coupled memory*(CCM), 4KB *battery-backed.* Flash terdiri dari 512/1024/2048KB, 30KB *system boot*, 512 *byte one-time-programmable*(OTP)
c. Peripheral
   USB 2.0 OTG HS dan FS, dua CAN 2.0B, SPI, dua I^2S, tiga I^2C, empat USART, dua UART, SDIO untuk kartu SD/MMC, dua belas timer 16 bit, dua timer 32 bit, dua timer watchdog, sensor temperatur, 16 atau 24 kanal dalam tiga ADC, dua DAC, 51 sampai 140 GPIO, enam belas DMA, RTC.
   STM32F4x7 memiliki ethernet MAC dan antar muka kamera. STM32F41x/43x memiliki *cryptographic processor* untuk DES / TDES / AES. STM32F4x9 memiliki pengontrol LCD-TFT.
   Osilator terdiri dari internal (16 MHz, 32 kHz) atau eksternal (4 to 26 MHz, 32.768 to 1000 kHz). Tegangan kerja sekitar 1.8 sampai 3.6 volt.

Gambar 2.9 *Board STM32F4-Discovery*

Gambar 2.7 merupakan gambar dari *Board STM32F4-Discovery* yang sudah dilengkapi dengan *downloader* tipe ST-Link untuk memasukkan program dari komputer ke mikrokontroler STM32F4 [4].

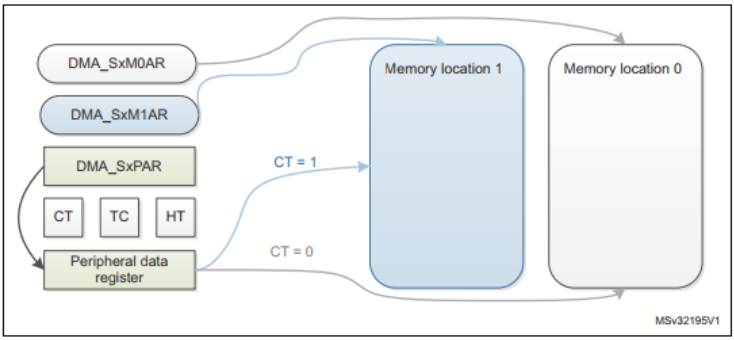
Selain fitur tersebut terdapat juga beberapa kelebihan mikrokontroler STM32F4 dalam operasi pemrosesan sinyal, fitur tersebut adalah *Direct Memory Access* (DMA), DMA adalah sistem yang memiliki arsitektur *bus matrix* dengan beberapa *layer*, dan memori sistem yang berkontribusi dalam memberikan *bandwidth* data yang tinggi serta membuat waktu respon *software* memiliki *latency* atau jeda yang rendah.

Penggunaan DMA memungkinkan untuk proses data secara *background* tanpa intervensi dari prosesor utama Cortex-Mx. Pada operasi ini, prosesor utama dapat menjalankan tugas lain dan hanya dapat di-*interrupt* ketika blok data sudah penuh dan siap untuk diproses.

Data dengan jumlah besar dapat ditransfer tanpa beban besar terhadap perfoma sistem, DMA disini biasanya digunakan untuk implementasi *central data buffer storage* untuk modul periferal berbeda. Solusi ini lebih murah pada bagian silicon dan konsumsi *power* dibanding solusi dimana tiap periferal mengharuskan untuk implementasi *local data storage* sendiri.

Kontroler DMA STM32F4 memanfaatkan kelebihan dari sistem *multi-layer bus* ini untuk dapat memastikan *latency* rendah pada saat transfer DMA dan pada saat proses eksekusi atau deteksi *interrupt* dari prosesor utama.

Salah satu mode transfer pada DMA adalah mode *double buffer* yang memiliki cara kerja mirip seperti transer DMA *single buffer*, dengan perbedaan yaitu pada *pointer* memori yang digunakan sebanyak 2. Ketika mode *double buffer* diaktifkan, *circular mode* secara otomatis aktif dan pada saat akhir dari masing-masing transaksi (DMA_SxNDTR register mencapai 0) *pointer* dari memori akan bergantian. Secara ilustrasi dapat digambarkan prosesnya seperti gambar 2.10



Gambar 2.10 Mode *Double Buffer*

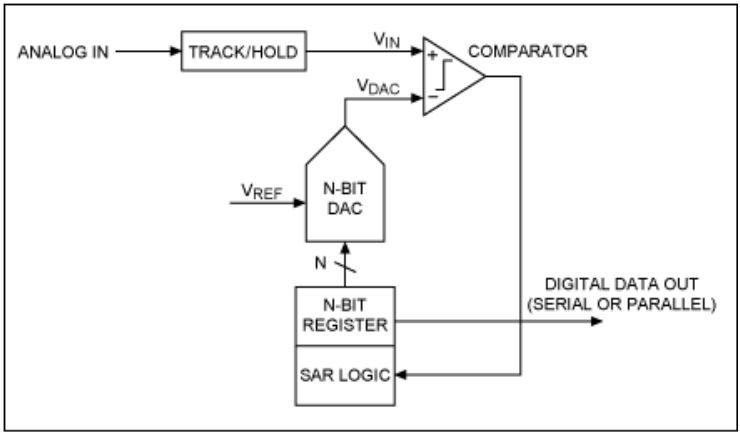
Mode ini memungkinkan *software* untuk melakukan proses pada satu memori area ketika memori area kedua sedang terisi atau terpakai pada transfer DMA

2.3.1 *Analog to Digital* (ADC) STM32F4

ADC merupkan pengubah besaran analog yang berupa tegangan analog menjadi besaran digital supaya dapat diproses secara digital. Mikrokontroler STM32F4xx memiliki 3 ADC dengan masing-masing ADC memiliki 19 kanal. Sembilan belas kanal ADC ini terbagi menjadi 16 kanal ADC eksternal dan 3 kanal ADC internal.

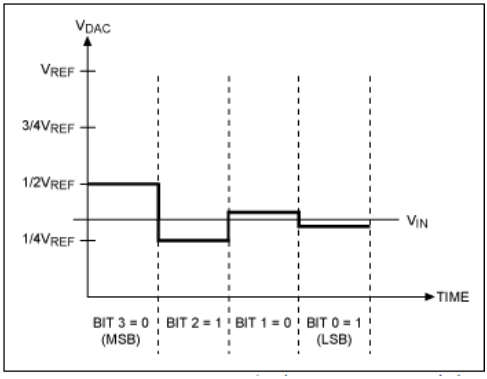
Tipe ADC yang digunakan pada mikrokontroler ini adalah tipe *Successive-approximation-register* (SAR) ADC, tipe ini banyak digunakan di pasaran dikarenakan kemampuannya dalam memproses ADC dengan *sampling rate* tinggi bernilai hingga 5 Megasamples (Msps) dengan resolusi dari 8 ke 18 bits, selain itu penggunaan daya pada ADC ini juga rendah dengan bentuk yang relatif kecil.

Pada intinya tipe ADC ini mengimplementasi algoritma *binary search*. Maka dari itu, ketika sirkuit internal mungkin beroperasi pada kecepatan beberapa megahertz (MHz), *sample rate* pada ADC hanya sebagian dari angka tersebut dikarenakan algoritma *successive-approximation*.



Gambar 2.11 Arsitektur N-bit SAR ADC yang disederhanakan

Gambar 2.11 disini menunjukkan arsitektur ADC tipe SAR, tegangan input analog (V_{IN}) disini ditahan. Untuk implementasi algoritma *binary search*, register N-bit awalnya diset ke skala tengah (yaitu, 100.. .00, dimana *Most Significant Bits* (MSB) diset menjadi 1). Hal ini memaksa output DAC (V_{DAC}) menjadi $V_{REF}/2$, dimana V_{REF} adalah referensi tegangan yang disediakan kepada ADC. Proses pembandingan disini terjadi untuk menentukan nilai V_{IN} memiliki nilai yang lebih besar dari V_{DAC} atau tidak. Jika V_{IN} nilainya lebih tinggi, output dari komparator akan menjadi 1 dan MSB dari register N-bit tetap pada 1. Sebaliknya jika V_{IN} lebih kecil dari V_{DAC}, komparator akan memberi nilai output 0 dan MSB pada register menjadi 0. Kontrol logika SAR lalu menggerakkan ke bit selanjutnya, memaksa nilai bitnya menjadi 1, lalu melakukan komparasi kembali. Urutan ini berjalan secara terus menerus sampai nilai *Least Significant Bits* (LSB). Setelah selesai, proses konversi berakhir dan output digital dengan bit sebanyak N sudah tersedia di register.



Gambar 2.12 Contoh ADC 4-bit operasi SAR

Selanjutnya pada gambar 2.12 ditunjukkan contoh konversi ADC 4 bit. Axis Y (dan garis tebal pada gambar) menunjukkan tegangan output DAC. Pada contoh diatas komparasi pertama menunjukkan bahwa V_{IN} lebih kecil dari V_{DAC}, maka bit 3 diset menjadi 0 dan nilai DAC diset menjadi 0100 lalu komparasi kedua dilakukan. Saat V_{IN} lebih besar dari V_{DAC}, bit 2 tetap bernilai 1 lalu nilai DAC diset kembali menjadi 0110, komparasi ketiga selanjutnya dilakukan, bit 1 diset menjadi 0 dan nilai DAC diset menjadi 0101 untuk komparasi terakhir. Akhirnya nilai bit 0 tetap 1 karena V_{IN} lebih besar dari V_{DAC}.

Dapat dilihat bahwa 4 periode komparasi dibutuhkan untuk ADC 4 bit. Maka dari itu SAR ADC N-bit akan membutuhkan komparasi sebanyak N periode dan tidak akan siap untuk komparasi berikutnya sampai komparasi yang sedang dijalankan selesai. Hal ini menjelaskan kenapa SAR ADC memiliki konsumsi daya yang rendah, tetapi kecepatan yang dilakukan tidak bisa lebih dari *sampling* pada frekuensi lebih dari MHz pada resolusi 14 ke 16 bits.

Pada mikrokontroler STM32F4 terdapat 16 kanal ADC yang dapat digunakan, dengan konfigurasi port seperti yang ditunjukkan pada tabal 2.1.

Tabel 2.1 Kanal ADC pada STM32F4

Channel APB	ADC1 2	ADC2 2	ADC3 2
ADC Channel 0	PA0	PA0	PA0
ADC Channel 1	PA1	PA1	PA1
ADC Channel 2	PA2	PA2	PA2
ADC Channel 3	PA3	PA3	PA3
ADC Channel 4	PA4	PA4	PF6
ADC Channel 5	PA5	PA5	PF7
ADC Channel 6	PA6	PA6	PF8
ADC Channel 7	PA7	PA7	PF9
ADC Channel 8	PB0	PB0	PF10
ADC Channel 9	PB1	PB1	PF3
ADC Channel 10	PC0	PC0	PC0
ADC Channel 11	PC1	PC1	PC1
ADC Channel 12	PC2	PC2	PC2
ADC Channel 13	PC3	PC3	PC3
ADC Channel 14	PC4	PC4	PF4
ADC Channel 15	PC5	PC5	PF5

2.3.2 *Timer* STM32F4

Timer merupakan fitur untuk pewaktuan. STM32F4 sebagai mikrokontroler juga memiliki timer sebagai pewaktuannya. STM32F4 memiliki sampai 14 timer. Timer juga dapat dimanfaatkan untuk menghasilkan sinyal kotak PWM.

Tabel 2.2 Macam-macam *timer*, resolusi, dan *clock timer* STM32F4

Timer type	Timer	Counter resolution	Counter type	Prescaler factor	DMA request generation	Capture/ compare channels	Complementary output	Max interface clock (MHz)	Max timer clock (MHz)
Advanced-control	TIM1, TIM8	16-bit	Up, Down, Up/down	Any integer between 1 and 65536	Yes	4	Yes	84	168
General purpose	TIM2, TIM5	32-bit	Up, Down, Up/down	Any integer between 1 and 65536	Yes	4	No	42	84
	TIM3, TIM4	16-bit	Up, Down, Up/down	Any integer between 1 and 65536	Yes	4	No	42	84
	TIM9	16-bit	Up	Any integer between 1 and 65536	No	2	No	84	168
	TIM10, TIM11	16-bit	Up	Any integer between 1 and 65536	No	1	No	84	168
	TIM12	16-bit	Up	Any integer between 1 and 65536	No	2	No	42	84
	TIM13, TIM14	16-bit	Up	Any integer between 1 and 65536	No	1	No	42	84
Basic	TIM6, TIM7	16-bit	Up	Any integer between 1 and 65536	Yes	0	No	42	84

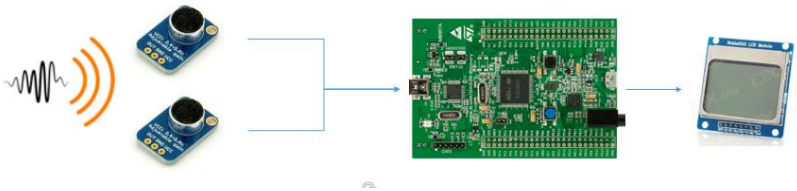
Halaman Ini Sengaja Dikosongkan

BAB III
PERANCANGAN DAN REALISASI ALAT

Pada bab ini menjelaskan tentang perancangan dan implementasi sistem meliputi arsitektur sistem, perancangan *hardware*, *software*, dan realisasi alat.

3.1 Perancangan Sistem

Sistem terdiri dari 2 buah mikrofon sebagai input sensor suara dan *amplifier* sebagai penguat sinyal suara serta mikrokontroler sebagai penghitung input yang diterima oleh sensor, lalu mikrokontroler memberi output berupa tampilan pada LCD. Berikut adalah ilustrasi diagram sistem yang dibuat:



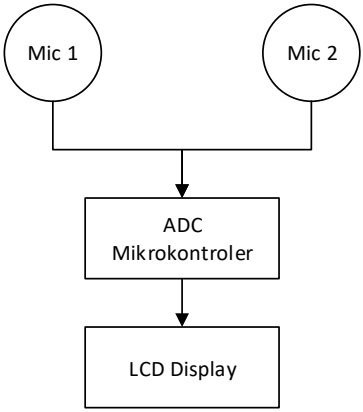
Gambar 3.1 Skema Urutan Sistem Pendeteksi Arah Sumber Suara

Pada Gambar 3.1 ditunjukkan bahwa suara dari sumber merambat melalui udara yang akan ditangkap oleh kedua mikrofon dengan jarak tertentu, lalu akan terdapat perbedaan waktu sampai dari sumber suara menuju kedua mikrofon karena jarak dari sumber suara dengan kedua mikrofon berbeda, jika mikrofon 1 lebih dekat dari sumber suara daripada mikrofon 2, maka waktu sampainya suara dari sumber suara menuju mikrofon 1 akan lebih cepat dibanding mikrofon 2, begitu juga sebaliknya, setelah perbedaan waktunya didapat, rumus perhitungan sudut arah sumber suara juga dapat dihitung.

Pemrosesan sistem setelah menerma input suara menggunakan mikrokontroler STM32F4, suara input dari mikrofon dibaca oleh ADC pada mikrokontroler sehingga sinyal analog suara diubah menjadi nilai digital yang dapat digunakan untuk proses perhitungan perbedaan waktu dan arah sumber suaranya, setelah sudutnya didapat mikrokontroler akan

mengeluarkan output berupa tampilan hasil sudut akhir dari perhitungan pada LCD.

3.2 Diagram Blok



Gambar 3.2 Diagram Blok Pendeteksi Arah Sumber Suara

Gambar 3.2 merupakan diagram blok dari alat pendeteksi arah sumber suara. Setelah mikrofon menerima suara dari sumber, sinyal suara akan diperkuat oleh pengondisi sinyal sehingga dapat dibaca oleh ADC pada mikrokontroler.

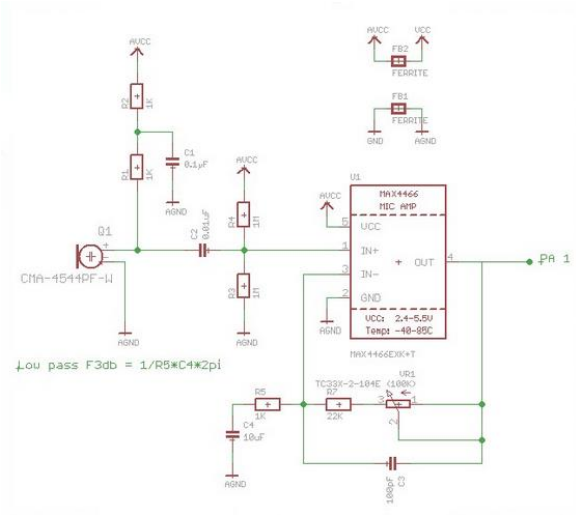
Setelah itu proses yang dilakukan pada mikrokontroler adalah proses *cross corelation* dan proses perhitungan arah sudut suara. *Cross corelation* disini digunakan untuk mendapatkan nilai perbedaan waktu sampai dari sumber suara menuju kedua mikrofon. Lalu arah sudut suara dapat dihitung dengan metode *Interaural Time Difference* (ITD) yang menggunakan perbedaan waktu sampai dan jarak antar mikrofon sebagai variabel penghitungnya. Setelah arah sudut dari sumber suara didapat, mikrokontroler menggerakan motor servo ke arah sudut tersebut sehingga terlihat seperti motor servo bergerak mengikuti posisi sumber suara.

3.3 Perancangan *Hardware*

Pada proyek kali ini hardware yang digunakan adalah 2 buah mikrofon, mikrokontroler STM32F4, serta LCD sebagai *display* hasilnya, perancangan *hardware* yang dilakukan meliputi perancangan mikrofon serta hubungan antara mikrokontroler dengan input dan output

3.3.1 Mikrofon MAX4466

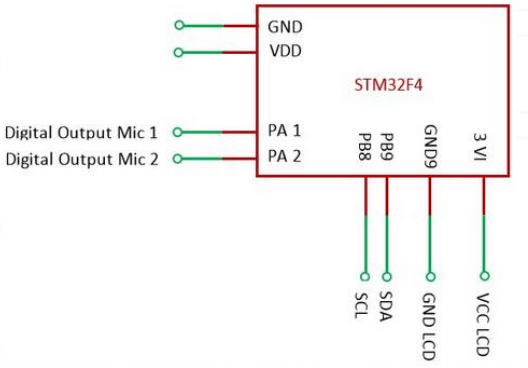
Mikrofon yang digunakan pada proyek ini adalah mikrofon MAX4466, mikrofon ini adalah mikrofon kondensor dengan operasi tegangan yang rendah sehingga cukup ditenagai hanya dari mikrokontroler dengan tegangan sebesar 3,3 v. Mikrofon ini juga memiliki pengondisi sinyal tersendiri dengan skematik seperti gambar 3.3.



Gambar 3.3 Skematik rangkaian mikrofon MAX4466

3.3.2 Mikrokontroler STM32F4

Channel ADC yang digunakan pada mikrokontroler STM32F4 adalah ADC1 dan ADC2 yang terhubung langsung dengan output pada mikrofon, selain itu *power supply* untuk mikrofon dihubungkan dengan VCC dan GND pada mikrokontroler. Lalu output dari program akan ditampilkan pada LCD berupa angka, LCD ini akan terhubung dengan VCC dan GND sebagai *power supply* untuk LCD, serta SDA dan SCL yang terhubung dengan mikrokontroler sebagai komunikasi datanya dengan rangkaian seperti gambar 3.4.



Gambar 3.4 Rancangan *hardware* alat pendeteksi suara

3.4 Perancangan *Software*

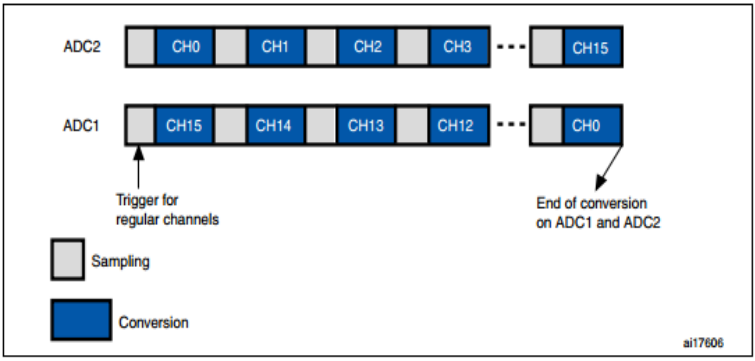
Mikrokontroler STM32F4 adalah mikrokontroler yang dapat digunakan untuk pengolahan sinyal dengan frekuensi tinggi seperti sinyal suara agar sinyal yang masuk melalui ADC sesuai dengan sinyal yang terbaca pada lapangan, perhitungan pada mikrokontroler ini juga memungkinkan untuk perhitungan untuk prosesi sinyal atau *Digital Signal Processing* (DSP), oleh karena itu mikrokontroler ini akan mengeluarkan hasil yang lebih akurat dalam prosesi sinyal digital daripada mikrokontroler dengan prosesor lain seperti arduino atau ATMega.

Dalam perancangan *software* pada proyek ini agar mikrokontroler dapat memproses sinyal suara dengan baik digunakan beberapa mode pada sistem ADC dan alokasi memori pada *background* agar kinerja

prosesor jauh lebih efisien dan frekuensi sampling yang dapat diaplikasikan pada program bisa lebih besar lagi.

3.4.1 ADC *Dual Mode – Simultaneous*

Pada STM32 yang memiliki ADC lebih dari 2 ini terdapat mode dimana kedua ADC dapat bekerja secara bersamaan, proses ini melakukan 2 konversi ADC secara bersamaan sesuai dengan sinkornisasi ADC 1 dan 2. Masing-masing ADC mengonversi susunan channel atau satu channel. Proses konversi dapat dimulai dengan *trigger* eksternal atau melalui *software*. Pada mode ini, hasil konversi ADC1 dan ADC2 akan disimpan pada register data ADC1 (32-bit format). Gambar 3.5 menunjukkan bagaimana ADC1 dan ADC2 mengonversi 2 bagian secara bersamaan. ADC1 mengonversi channel 15 sampai 0 sedangkan ADC2 mengonversi channel 0 sampai 15.



Gambar 3.5 Proses konversi ADC1 dan ADC2

```
/* ADC1 init function */
static void MX_ADC1_Init(void)
{

  ADC_MultiModeTypeDef multimode;
  ADC_ChannelConfTypeDef sConfig;

    /**Configure the global features of the ADC (Clock, Resolution,
Data Alignment and number of conversion)
    */
  hadc1.Instance = ADC1;
  hadc1.Init.ClockPrescaler = ADC_CLOCK_SYNC_PCLK_DIV4;
  hadc1.Init.Resolution = ADC_RESOLUTION_12B;
```

```
  hadc1.Init.ScanConvMode = DISABLE;
  hadc1.Init.ContinuousConvMode = DISABLE;
  hadc1.Init.DiscontinuousConvMode = DISABLE;
  hadc1.Init.ExternalTrigConvEdge = ADC_EXTERNALTRIGCONVEDGE_RISING;
  hadc1.Init.ExternalTrigConv = ADC_EXTERNALTRIGCONV_T2_TRGO;
  hadc1.Init.DataAlign = ADC_DATAALIGN_RIGHT;
  hadc1.Init.NbrOfConversion = 1;
  hadc1.Init.DMAContinuousRequests = ENABLE;
  hadc1.Init.EOCSelection = ADC_EOC_SINGLE_CONV;
  if (HAL_ADC_Init(&hadc1) != HAL_OK)
  {
    Error_Handler();
  }

    /**Configure the ADC multi-mode
    */
  multimode.Mode = ADC_DUALMODE_REGSIMULT;
  multimode.DMAAccessMode = ADC_DMAACCESSMODE_2;
  multimode.TwoSamplingDelay = ADC_TWOSAMPLINGDELAY_5CYCLES;
  if (HAL_ADCEx_MultiModeConfigChannel(&hadc1, &multimode) != HAL_OK)
  {
    Error_Handler();
  }

    /**Configure for the selected ADC regular channel its
corresponding rank in the sequencer and its sample time.
    */
  sConfig.Channel = ADC_CHANNEL_1;
  sConfig.Rank = 1;
  sConfig.SamplingTime = ADC_SAMPLETIME_3CYCLES;
  if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
  {
    Error_Handler();
  }

}
```

Inisialisasi ADC dan *dual mode*, pada rancangan ini resolusi ADC yang digunakan yaitu 12 bit, dengan mode *continuous conversion* non aktif, *trigger* yang digunakan adalah trigger eksternal yaitu *timer* 2 dengan DMA *continuous request* aktif. Mode yang digunakan adalah ADC *multi mode*, channel yang digunakan yaitu channel 1 dan 2. Port yang digunakan yaitu port A1 dan port A2.

3.4.2 *Direct Memory Access – Double Buffer*

Direct Memory Accessi (DMA) adalah proses yang memungkinkan proses transfer data berjalan pada *background* atau bekerja tanpa membebani kinerja prosesor. Pada proses ini, prosesor utama dapat melakukan pekerjaan lain dan hanya dapat di-*interrupt* ketika

blok data sudah siap untuk diproses. Data dengan jumlah besar dapat diproses tanpa menyebabkan efek yang besar pada performa sistem.

Mode *double buffer* disini adalah operasi dimana pengisian data pada memori dilakukan menggunakan 2 buah *memory pointer*, jadi pengisian data awalnya dilakukan pada memori 1, setelah penuh data akan mengisi memori 2, dengan data pada memori 1 siap untuk diproses. Setelah data memori 2 penuh, data pada memori 2 akan diproses dan pengisian data berpindah kembali pada memori 1.

```
uint8_t GROGALAN_CORR_Init_F32(GROGALAN_CORR_F32_t* CORR, uint16_t
CORR_Size0, uint16_t CORR_Size1) {
	/* Set to zero */
	CORR->CORR_Size0 = CORR_Size0;
	CORR->CORR_Size1 = CORR_Size1;

	/* Check if fft size valid */
	if (CORR->CORR_Size0 == 0) {
		/* There is not valid input, return */
		return 1;
	}
	/* Return OK */
	return 0;
void GROGALAN_CORR_SetBuffers_F32(GROGALAN_CORR_F32_t* CORR,
float32_t* InputBuffer0, float32_t* InputBuffer1, float32_t*
OutputBuffer) {

	/* Set pointers */
	CORR->Input0 = InputBuffer0;
	CORR->Input1 = InputBuffer1;
	CORR->Output = OutputBuffer;
}
```

Program di atas adalah program inisialisasi *buffer* sebagai input pada *cross-correaltion*, variabel input buffer yang digunakan adalah InputBuffer 0, 1, dan 2.

```
/**
  * Enable DMA controller clock
  */
static void MX_DMA_Init(void)
{
  /* DMA controller clock enable */
  __HAL_RCC_DMA1_CLK_ENABLE();
  __HAL_RCC_DMA2_CLK_ENABLE();

  /* DMA interrupt init */
  /* DMA1_Stream5_IRQn interrupt configuration */
  HAL_NVIC_SetPriority(DMA1_Stream5_IRQn, 0, 0);
  HAL_NVIC_EnableIRQ(DMA1_Stream5_IRQn);
  /* DMA2_Stream0_IRQn interrupt configuration */
```

```
  HAL_NVIC_SetPriority(DMA2_Stream0_IRQn, 0, 0);
  HAL_NVIC_EnableIRQ(DMA2_Stream0_IRQn);
  /* DMA2_Stream2_IRQn interrupt configuration */
  HAL_NVIC_SetPriority(DMA2_Stream2_IRQn, 0, 0);
  HAL_NVIC_EnableIRQ(DMA2_Stream2_IRQn);

}
```

Program diatas adalah inisialisasi DMA, program inisialisasi ini mengubah prioritas awal dari DMA menjadi 0 lalu mengaktifkan *interrupt*.

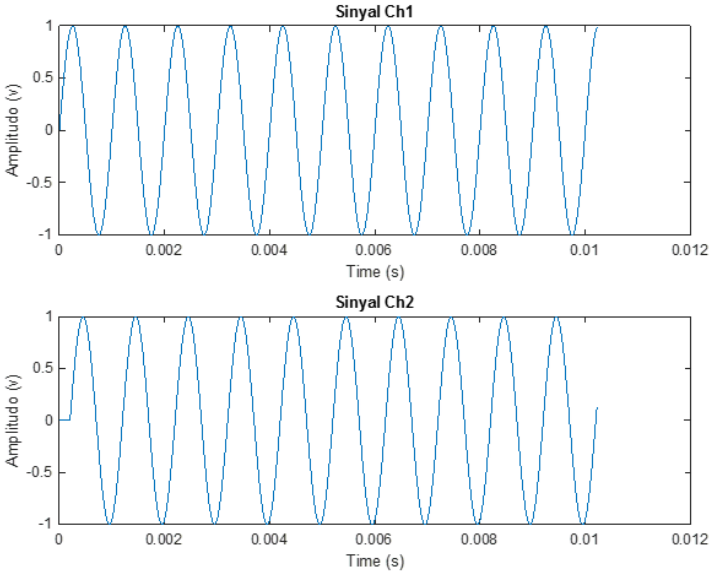
```
//multi buffer
        if(Grogalan_HAL_ADC_Start_DMA_MultiBuffer_IT(&hadc1,
        (uint32_t)ADC1_DR_ADDRESS, (uint32_t)&dmabuffer[0][0]
        ,(uint32_t)&dmabuffer[1][0] ,CORR_SAMPLE_SIZE) != HAL_OK){
                TM_SSD1306_GotoXY(0, 0);
                TM_SSD1306_Puts("Error Multi Buff", &TM_Font_7x10,
                SSD1306_COLOR_WHITE);
                TM_SSD1306_UpdateScreen();
                while(1);
        }

        if(Grogalan_HAL_ADC_Start_DMA_MultiBuffer_IT(&hadc2,
        (uint32_t)ADC2_DR_ADDRESS, (uint32_t)&dmabuffer1[0][0]
        ,(uint32_t)&dmabuffer1[1][0] ,CORR_SAMPLE_SIZE) != HAL_OK){
                TM_SSD1306_GotoXY(0, 0);
                TM_SSD1306_Puts("Error Multi Buff ADC2",
                &TM_Font_7x10, SSD1306_COLOR_WHITE);
                TM_SSD1306_UpdateScreen();
                while(1);
        }
```

Selanjutya program *start adc dual mode* dengan konfigurasi DMA, serta penampilan hasil input pada LCD

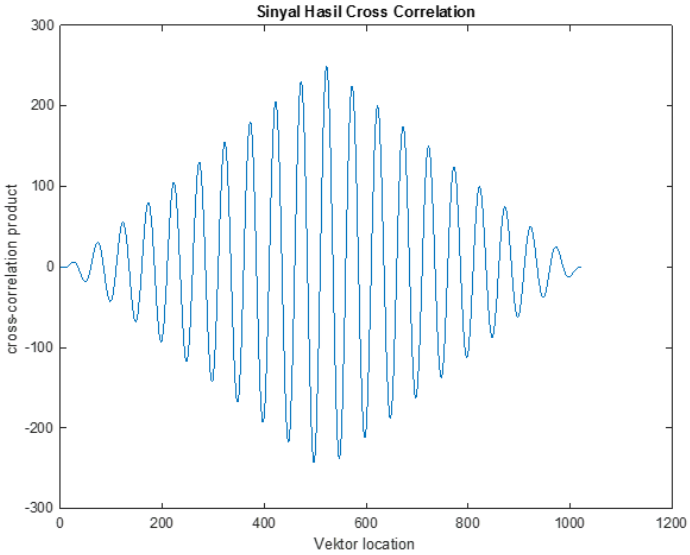
3.4.3 *Cross-correlation*

Program *cross-correlation* pada proyek kali ini digunakan untuk mencari jeda waktu sampai suara dari sumber menuju kedua mikrofon, proses ini dilakukan setelah pembacaan mikrofon dengan ADC lalu data konversi dari ADC masuk ke memori melalui DMA. Setelah itu data mulai diproses oleh prosesor utama untuk masuk ke perhitungan *cross-correlation*.



Gambar 3.6 Contoh input data sinusoidal pada MATLAB

Dikarenakan sumber suara yang digunakan adalah suara dengan bentuk sinyal sinusoidal. Dilakukan simulasi pada MATLAB, yaitu program *cross-correlation* dengan input 2 buah sinyal sinusoidal dengan perbedaan 0,2 ms seperti dilihat pada gambar 3.?.



Gambar 3.7 Hasil *cross-correlation* kedua sinyal pada simulasi MATLAB

Sesuai dengan perhitungan pada rumus *cross-correlation*, hasil grafiknya menjadi seperti gambar 3.? dengan nilai maksimum dari grafik sebagai nilai perbedaan antara kedua sinyal input.

Pada pemrograman proses untuk mencari nilai jeda dari kedua sinyal ini melalui 2 tahap yaitu proses korelasi untuk mendapatkan fungsi dari *cross-correlation*, lalu proses pencarian nilai maksimum dari fungsi -*cross-correlation* untuk mendapatkan nilai jeda sebagai variabel untuk mencari sudut lokasi sumber suara.

```
  /* -------------------------
   * Initializations of stage1
   * -------------------------*/

  /* sum = x[0] * y[srcBlen - 1]
   * sum = x[0] * y[srcBlen-2] + x[1] * y[srcBlen - 1]
   * ....
   * sum = x[0] * y[0] + x[1] * y[1] +...+ x[srcBLen - 1] * y[srcBLen
- 1]
   */
```

```
 /* In this stage the MAC operations are increased by 1 for every
iteration.
 The count variable holds the number of MAC operations performed */
/* -------------------------
   * Initializations of stage2
   * ------------------------*/

  /* sum = x[0] * y[0] + x[1] * y[1] +...+ x[srcBLen-1] * y[srcBLen-
1]
   * sum = x[1] * y[0] + x[2] * y[1] +...+ x[srcBLen] * y[srcBLen-1]
   * ....
   * sum = x[srcALen-srcBLen-2] * y[0] + x[srcALen-srcBLen-1] * y[1]
+...+ x[srcALen-1] * y[srcBLen-1]
   */
/* -------------------------
   * Initializations of stage3
   * ------------------------*/

  /* sum += x[srcALen-srcBLen+1] * y[0] + x[srcALen-srcBLen+2] * y[1]
+...+ x[srcALen-1] * y[srcBLen-1]
   * sum += x[srcALen-srcBLen+2] * y[0] + x[srcALen-srcBLen+3] * y[1]
+...+ x[srcALen-1] * y[srcBLen-1]
   * ....
   * sum +=  x[srcALen-2] * y[0] + x[srcALen-1] * y[1]
   * sum +=  x[srcALen-1] * y[0]
   */
```

Gambaran Program Proses Korelasi, program ini diambil dari *library* STM32F4, program ini memiliki 3 tahap perhitungan dengan rumus mengikuti perhitungan *cross-correlation*, hasil dari program ini adalah *array* data dengan jumlah data dua kali dari jumlah data input. Sebagai contoh jika data digital yang terbaca dari ADC sebanyak 512 data, hasil *array* data yang dihasilkan dari program ini sebanyak 1024 data. Selanjutnya dicari nilai maksimum dari *array* data tersebut untuk mendapatkan jeda perbedaan dari kedua sinyal input menggunakan program pencari nilai maksimum dengan gambaran program di bawah.

```
while(blkCnt > 0u)
  {
    /* Initialize maxVal to the next consecutive values one by one */
    maxVal1 = *pSrc++;

    maxVal2 = *pSrc++;

    /* compare for the maximum value */
    if(out < maxVal1)
    {
      /* Update the maximum value and its index */
      out = maxVal1;
      outIndex = count + 1u;
    }
```

```
    maxVal1 = *pSrc++;

    /* compare for the maximum value */
    if(out < maxVal2)
    {
      /* Update the maximum value and its index */
      out = maxVal2;
      outIndex = count + 2u;
    }

    maxVal2 = *pSrc++;

    /* compare for the maximum value */
    if(out < maxVal1)
    {
      /* Update the maximum value and its index */
      out = maxVal1;
      outIndex = count + 3u;
    }

    /* compare for the maximum value */
    if(out < maxVal2)
    {
      /* Update the maximum value and its index */
      out = maxVal2;
      outIndex = count + 4u;
    }

    count += 4u;

    /* Decrement the loop counter */
    blkCnt--;
  }

  /* if (blockSize - 1u) is not multiple of 4 */
  blkCnt = (blockSize - 1u) % 4u;
```

Program Pencarian nilai maksimum, fungsi ini akan menghasilkan nilai maksimum dari *array* data input yang pada kasus ini adalah data dari hasil *cross-correlation*, hasil dari fungsi ini adalah jeda atau beda fasa dari kedua sinyal input yang selanjutnya digunakan pada program pencarian sudut atau program ITD.

3.4.4 Rancangan ITD

Setelah jeda waktu dari *cross-correlation* didapat, proses perhitungan sudut arah sumber suara menggunakan metode ITD dilakukan, perhitungan sudut dilakukan menggunakan rumus trigonometri yang variabelnya sudah didapat dari jarak mikrofon, frekuensi sampling, dan hasil perhitungan *cross-correlation.*

```
GROGALAN_CORR_Process_F32(&CORR);
  /* lihat eq no. 5 */
  TimeDelay = ((float32_t)GROGALAN_CORR_GetMaxIndex(&CORR) -
  (float32_t)ZERO_DEGREES_INDEX_X_CORR_OUTPUT) * 0.00002;
  Angle_Degrees = asinf((TimeDelay*V_SOUNDS)/JARAK_ANTAR_MIC);
```

Program rumus pencarian sudut, rumus yang digunakan adalah rumus ITD yang prinsipnya sama dengan trigonometri, pada rumus ini yang digunakan adalah arc sinus perkalian jeda waktu dengan kecepatan suara dibagi jarak antar mikrofon.

Halaman Ini Sengaja Dikosongkan

BAB IV
PENGUJIAN DAN ANALISA SISTEM

Bab ini menjelaskan tentang hasil pengujian dan analisa dari perancangan sistem yang telah dilakukan dan dijelaskan pada bab sebelumnya. Pengujian dari sistem pendeteksi arah suara ini dilakukan menjadi dua tahap, yaitu pengujian *hardware* dan *software*. Pengujian *hardware* disini berupa pengujian mikrofon. Sedangkan pengujian *software* disini berupa pengujian rumus *Interaural Time Difference* (ITD).

Gambar 4.1 Susunan alat pendeteksi sudut

Gambar 4.1 disini menunjukkan *smartphone* sebagai sumber suara, 2 buah mikrofon kondenser dengan modul MAX 4466 sebagai *preamplifier*, box hitam yang berisi STM32F4 sebagai mikrokontroler, LCD sebagai penampil *output* serta 2 buah corong yang terbuat dari botol plastik sebagai penangkap suara.

4.1 Pengujian *Hardware*

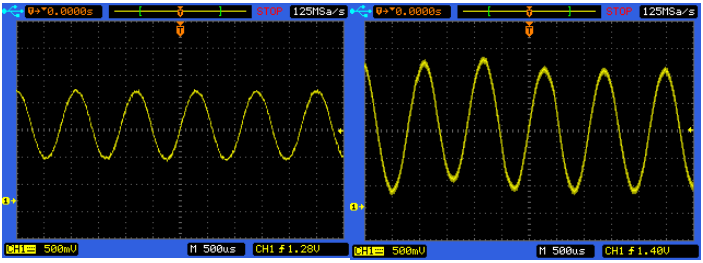
Pada pengujian *hardware* proyek ini yang diuji adalah mikrofon dan ADC pada mikrokontroler STM32F4, pengujian mikrofon disini

mencakup pembacaan 3 bentuk sinyal yang berbeda yaitu sinyal dengan bentuk sinusoidal, sinyal kotak, lalu sinyal segitiga.

Selanjutnya pengujian ADC yaitu pembacaan sinyal suara sinusoidal lalu data digital yang dihasilkan dijadikan grafik data, setelah itu kedua input yang sudah diproses dengan *cross-correlation* juga akan dijadikan grafik data.

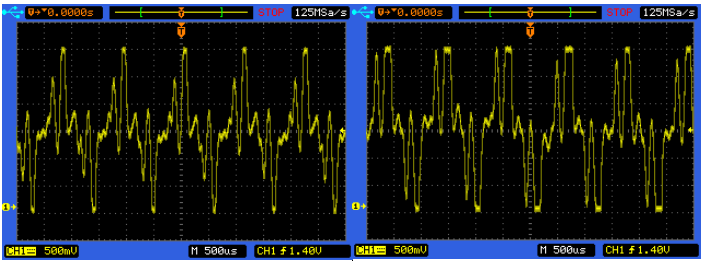
4.1.1 Pengujian mikrofon

Dalam proyek ini *hardware* yang digunakan adalah mikrofon kondensor sebagai sensor suara, mikrokontroler STM32F4 sebagai prosesor utama dan LCD sebagai *output*. Mikrofon yang digunakan adalah mikrofon kondensor dengan preamplifier modul MAX 4466 sehingga output langsung berupa sinyal suara yang dapat diolah pada mikrokontroler, gambar 4.2 disini menunjukkan hasil percobaan mikrofon 1 dan 2 dalam membaca sinyal sinusoidal. Suara sinyal sinusoidal dihasilkan menggunakan *frequency generator* dari aplikasi *smartphone*



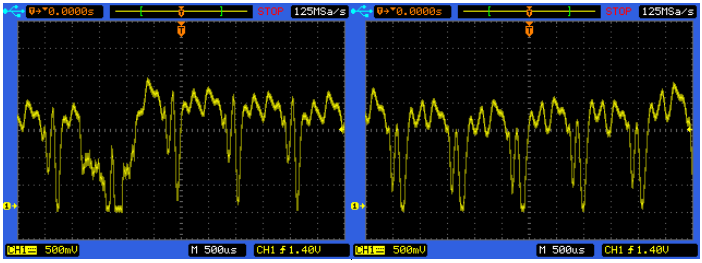
Gambar 4.2 Hasil pengujian kedua mikrofon membaca sinyal sinusoidal

Untuk percobaan selanjutnya dilakukan dengan menggunakan suara dengan bentuk sinyal kotak dan segitiga untuk melihat kemampuan mikrofon dalam menangkap suara dengan bentuk sinyal yang merupakan kombinasi dari beberapa sinyal, hasilnya dapat dilihat pada gambar 4.3 dan gambar 4.4



Gambar 4.3 Hasil pengujian kedua mikrofon membaca suara sinyal kotak

Pada pembacaan sinyal kotak tidak terbaca sinyal kotak melainkan sinyal dengan bentuk tertentu secara beraturan, hal ini dikarenakan kualitas *speaker* dari *smartphone* dan akurasi *output* sinyal yang dihasilkan dari aplikasi yang kurang sesuai. Hal yang sama terjadi pada pengujian berikutnya yaitu pembacaan sinyal segitiga seperti yang terlihat pada gambar 4.4

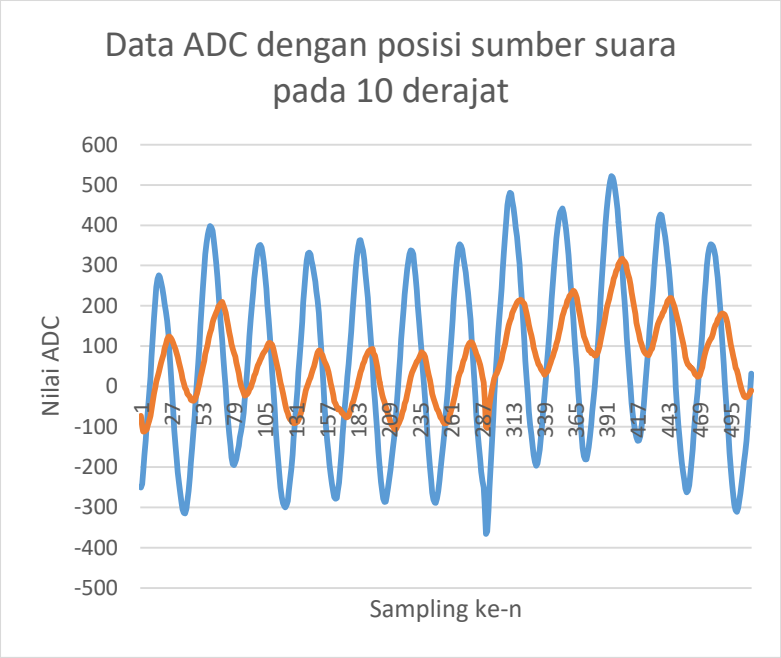


Gambar 4.4 Hasil pengujian kedua mikrofon membaca sinyal segitiga

4.1.2 Pengujian ADC STM32F4

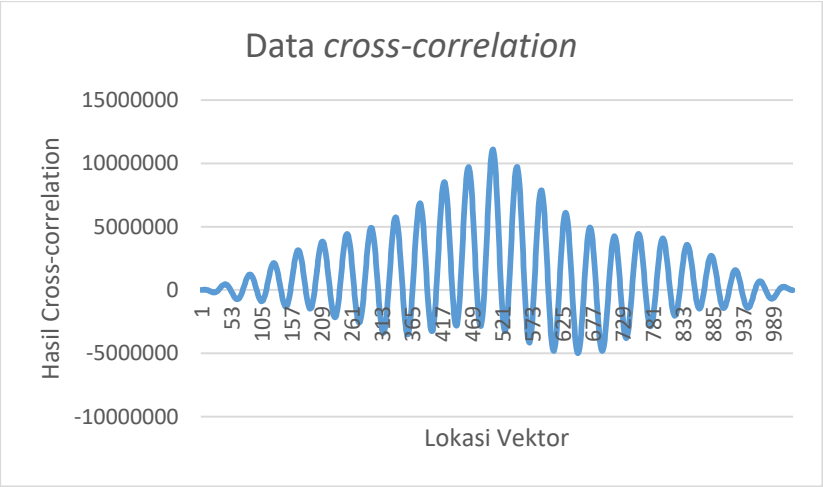
Pengujian selanjutnya yang dilakukan adalah pengujian ADC dalam menangkap sinyal dari mikrofon, data analog dari mikrofon akan dibaca oleh ADC menghasilkan nilai digital, lalu nilai per-*sampling* akan ditampilkan sebagai grafik data, setelah itu input akan diproses dalam fungsi *cross-correlation* menghasilkan grafik data baru berupa hasil perhitungan fungsi tersebut. Data hasil ADC yang dicatat dalam

pengujian ini sebanyak 512 data sedangkan data dari hasi *cross-correlation* sebanyak 1024 data.



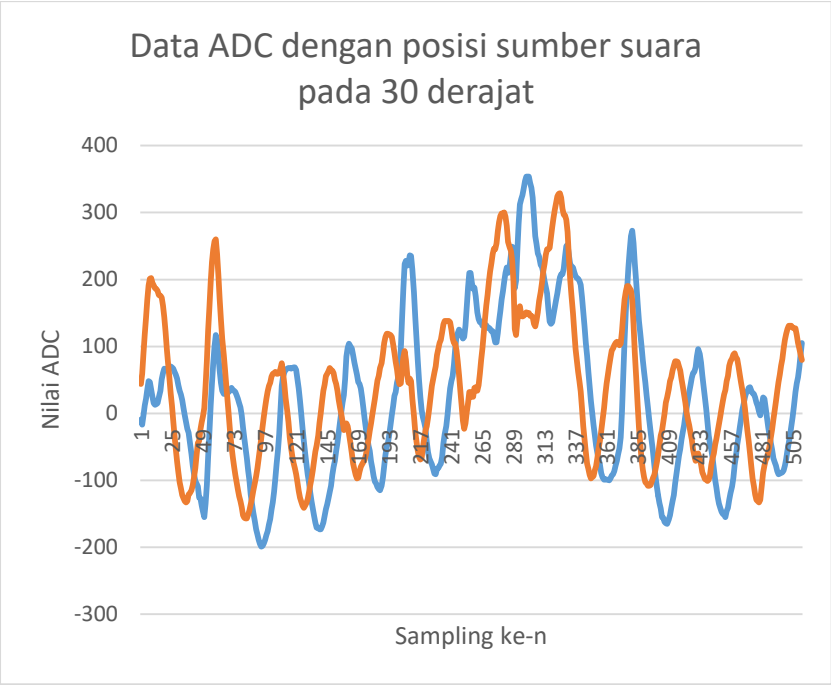
Gambar 4.5 Data ADC dengan posisi sumber suara pada 10 derajat

Dari pengujian di atas dapat dilihat kedua sinyal dengan fasa yang hampir sama dikarenakan pembacaan sinyal dengan posisi sumber suara pada 10 derajat atau hampir tepat di antara kedua mikrofon sebagai sumber suara. Besar kedua sinyal disini berbeda dikarenakan kualitas pembacaan mikrofon yang berbeda juga.



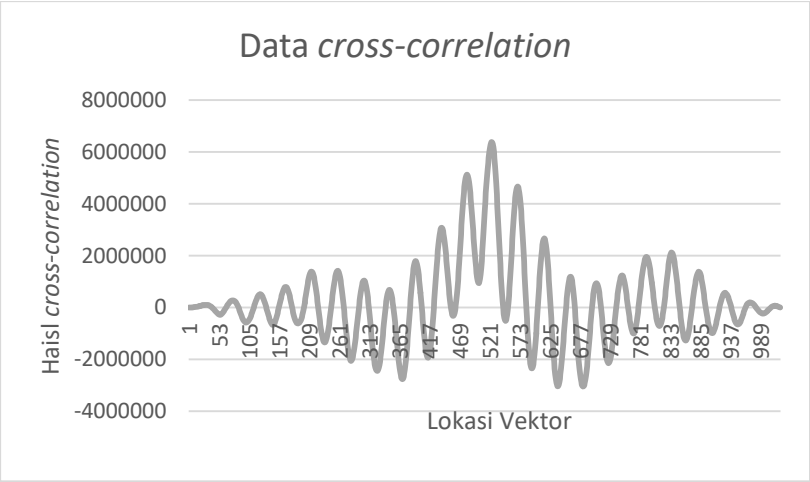
Gambar 4.6 Hasil *cross-correlation* kedua sinyal dengan posisi sumber suara pada 10 derajat

Selanjutnya gambar 4.6 menunjukkan hasil perhitungan *cross-correlation* kedua sinyal input, sesuai dengan beda fasa dari kedua sinyal, nilai terbesar dari hasil perhitungan *cross-correlation* berada pada posisi tengah atau hampir tidak ada perbedaan fasa pada kedua sinyal tersebut.



Gambar 4.7 Data ADC dengan posisi sumber suara pada 30 derajat

Gambar 4.7 menunjukkan pengujian pembacaan pada ADC dengan posisi sumber suara pada 30 derajat, pada pengujian ini dapat terlihat adanya perbedaan fasa diantara kedua sinyal. Bentuk sinyal juga disini mulai kacau karena jarak sumber suara mulai menjauh dari salah satu mikrofon.



Gambar 4.8 Hasil *cross-correlation* kedua sinyal dengan posisi sumber suara pada 30 derajat

Berbeda dengan percobaan sebelumnya, pada hasil perhitungan ini terdapat pergeseran pada nilai maksimum pada hasil *cross-correlation*, hal ini disebabkan bergeraknya sumber suara mendekati salah satu mikrofon.

4.2 Pengujian *Software*

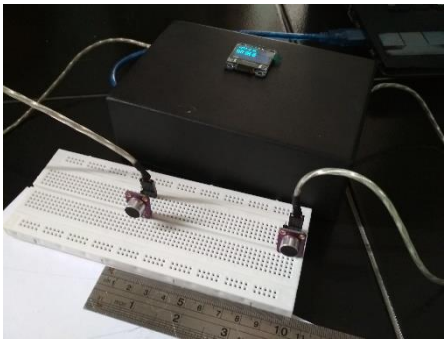
Dalam perancangan *software* sistem pendeteksi arah suara ini dilakukan pengujian untuk membuktikan kebenaran metode *cross corelation* sebagai metode untuk mencari perbedaan waktu sampai suara dari sumber menuju kedua mikrofon serta membuktikan rumus ITD untuk menghitung arah sudut suara.

Pada pengujian kali ini variabel yang digunakan adalah jarak antar mikrofon, sudut sumber suara terhadap mikrofon, penggunaan parabola pada mikrofon, serta jenis suara yang digunakan. Suara yang digunakan pada pengujian kali ini rata-rata sebesar 73 dB.

4.2.1 Pengujian sudut dengan variabel jarak sumber suara

Jarak antar mikrofon 10 cm

Pada pengujian kali ini menggunakan input berupa suara yang dihasilkan dari smartphone yang menghasilkan suara dengan frekuensi konstan sebesar 900 Hz, serta hasil perhitungan akan ditampilkan pada LCD dengan susunan hardware seperti yang ditampilkan pada gambar 4.5.



Gambar 4.9 Percobaan sudut dengan jarak antar mikrofon 10 cm

Jarak sumber suara 10 cm

Tabel 4.1 Pengujian sudut dengan jarak antar mikrofon 10 cm dan jarak sumber suara 10 cm

Sudut tujuan (^{o})	Sudut yang terukur (^{o})	Error Sudut (%)
-90	-52,94	41,62
-70	-39,7	43,29
-50	-28,73	42,54
-30	-23,65	21,17
-10	-13,95	39,5
0	-4,61	-
10	0	100
30	9,25	69,17
50	28,73	42,54
70	45,79	34,59
90	52,54	41,62

Tabel 4.1 Menunjukkan bahwa dengan jarak antara kedua mikrofon sebesar 10 cm dan jarak sumber suara dengan mikrofon sebesar 10 cm, terdapat nilai *error* yang cukup besar pada sudut diatas 30^o, sedangkan pada sudut 10^o menghasilkan nilai sudut 0^o seakan-akan posisi berada tepat didepan kedua mikrofon, hal ini dikarenakan kualitas mikrofon yang berbeda sehingga menyebabkan sinyal yang terbaca memiliki sedikit perbedaan meskipun suara yang dibaca sama. Akan tetapi besar sudut yang dihasilkan tetap membesar dan mengecil mengikuti gerakan sumber suara.

Jarak sumber suara 20 cm

Tabel 4.2 Pengujian sudut dengan jarak antar mikrofon 10 cm dan jarak sumber suara 20 cm

Sudut tujuan (º)	Sudut yang terukur (º)	Error Sudut (%)
-90	-60,37	32,92
-70	-45,79	34,59
-50	-28,73	42,54
-30	-18,73	37,57
-10	-13,95	39,5
0	-4,61	-
10	0	100
30	13,95	53,5
50	28,73	42,54
70	45,79	34,59
90	52,54	41,62

Dapat dilihat dari tabel 4.2 bahwa hasil sudut yang dihasilkan tidak jauh berbeda dengan percobaan sebelumnya, percobaan ini juga menunjukkan bahwa mikrofon yang digunakan masih dapat membaca sinyal suara dengan jarak lebih dari 20 cm.

Jarak sumber suara 30 cm

Tabel 4.3 Pengujian sudut dengan jarak antar mikrofon 10 cm dan jarak sumber suara 30 cm

Sudut tujuan (º)	Sudut yang terukur (º)	Error Sudut (%)
-90	Tidak beraturan	100
-70	-23,65	66,21
-50	-18,73	62,54
-30	-9,25	69,17
-10	-4,61	53,9
0	0	-
10	4,61	53,9
30	4,61	84,63
50	9,25	81,5
70	13,95	80,07
90	Tidak beraturan	100

Tabel 4.3 sudah mulai menunjukkan banyak kesalahan pada perhitungan sudut, bahkan terdapat hasil yang tidak beraturan pada pengujian sudut 90º, pada layar LCD hasil sudut yang terbaca bernilai positif dan negatif secara bergantian sehingga hasil tidak akurat.

Hal ini terjadi akibat pembacaan sinyal yang dilakukan oleh sensor suara atau mikrofon sudah mulai berkurang akurasinya, suara yang dihasilkan oleh sumber suara terlalu jauh sehingga sinyal yang terbaca oleh sensor suara tidak sesuai dengan sinyal pada sumber suara. Hasil sudut yang dihasilkan akan semakin stabil apabila sinyal yang terbaca oleh sensor suara semakin mendekati sinyal pada sumber suara.

Jarak antar mikrofon 20 cm

Pengujian selanjutnya menggunakan input sama yaitu suara dengan frekuensi tetap 900 Hz dengan perbedaan pada jarak antar mikrofon yaitu pada 20 cm, perbedaan jarak mikrofon ini akan berakibat pada resolusi pembacaan, perubahan variabel jarak antar mikrofon pada rumus azimutnya, serta jarak sumber suara dengan masing-masing mikrofon.

Jarak sumber suara 10 cm

Tabel 4.4 Pengujian sudut dengan jarak antar mikrofon 20 cm dan jarak sumber suara 10 cm

Sudut tujuan (°)	Sudut yang terukur (°)	Error Sudut (%)
-90	-60,37	32,92
-70	-45,79	34,59
-50	-31,35	37,3
-30	-23,65	21,17
-10	-4,61	53,9
0	0	-
10	4,61	53,9
30	23,65	21,17
50	31,36	37,28
70	36,79	47,44
90	52,54	41,62

Pada percobaan kali ini dilakukan perubahan pada jarak mikrofon yang seharusnya meningkatkan resolusi dan akurasi pembacaan sudut, akan tetapi hasil yang didapat tidak jauh berbeda dengan percobaan dengan jarak antar mikrofon 10 cm, dapat dilihat juga bahwa nilai error membesar saat sudut yang diujikan juga membesar, hal ini diakibatkan jarak sumber suara yang semakin menjauh dari salah satu mikrofon saat sudut yang diujikan membesar. Contohnya saat pengujian pada sudut 90°, sumber suara berada pada lokasi dekat sekali dengan mikrofon satu tetapi sangat jauh dengan mikrofon lainnnya, sehingga pembacaan sinyal pada kedua mikrofon tidak sama.

Jarak sumber suara 20 cm

Tabel 4.5 Pengujian sudut dengan jarak antar mikrofon 20 cm dan jarak sumber suara 20 cm

Sudut tujuan (º)	Sudut yang terukur (º)	Error Sudut (%)
-90	Tidak beraturan	100
-70	-42,68	39,03
-50	-34,06	31,88
-30	-21,17	29,43
-10	-16,33	63,3
0	-9,25	-
10	0	100
30	13,95	53,5
50	39,7	20,6
70	Tidak beraturan	100
90	Tidak beraturan	100

Pada percobaan ini didapat hasil yang cukup mirip dengan percobaan dengan jarak sumber suara sejauh 30 cm karena jaraknya yang tidak berbeda jauh dengan percobaan tersebut, terdapat beberapa sudut dengan hasil tidak beraturan dikarenakan jarak sumber suara yang terlalu jauh dengan salah satu mikrofon.

Jarak antar mikrofon 30 cm dengan jarak sumber suara 10 cm

Percobaan kali ini menggunakan jarak antar mikrofon 30 cm dengan sumber suara hanya pada 10 cm, karena apabila diuji dengan jarak sumber suara lebih jauh lagi pembacaan sinyal tidak akan akurat dan hasil sudut yang dihasilkan tidak beraturan.

Tabel 4.6 Pengujian sudut dengan jarak antar mikrofon 30 cm dan jarak sumber suara 10 cm

Sudut tujuan (º)	Sudut yang terukur (º)	Error Sudut (%)
-90	Tidak beraturan	100
-70	Tidak beraturan	100
-50	Tidak beraturan	100
-30	-17,13	42,9
-10	-20,36	100
0	-13,95	-
10	-4,61	53,9
30	7,7	74,33
50	Tidak beraturan	100
70	Tidak beraturan	100
90	Tidak beraturan	100

Percobaan dengan jarak antar mikrofon 30 cm ini memiliki hasil dengan akurasi sangat rendah, sudut yang dihasilkan juga sering berpindah-pindah, gerakan sedikit pada sumber suara berpengaruh besar pada hasil sudut yang dihitung. Dan pada pengujian 50º keatas, suara sudah tidak terbaca pada salah satu mikrofon dengan jarak jauh sehingga hasil perhitungan jadi tidak beraturan.

Jarak antar mikrofon 10 cm suara frekuensi 900 Hz menggunakan corong botol plastik

Percobaan selanjutnya dilakukan dengan variasi jarak dan suara yang sama akan tetapi digunakan corong yang dibuat menggunakan botol air mineral sebagai penangkap suara seperti yang diperlihatkan pada gambar 4.6, dengan harapan dapat memerkuat sinyal suara yang ditangkap oleh mikrofon.

Gambar 4.10 Percobaan sudut dengan corong botol plastik

Jarak sumber suara 10 cm

Tabel 4.7 Pengujian sudut dengan jarak antar mikrofon 10 cm dan jarak sumber suara 10 cm

Sudut tujuan (º)	Sudut yang terukur (º)	Error Sudut (%)
-90	-90,85	0,94
-70	-70,33	0,47
-50	-45,79	8,42
-30	-23,65	21,17
-10	-9,25	7,5
0	0	-
10	-13,95	39,5
30	28,73	4,23
50	52,54	5,08
70	70,33	0,47
90	90,85	0,94

Pada pengujian kali ini digunakan parabola sebagai pembantu mikrofon dalan menangkap suara yang datang dari sumber suara, hasilnya sudut yang dihitung memiliki hasil stabil dengan akurasi cukup tinggi, akan tetapi dikarenakan jarak sumber suara yang sangat dekat, menyebabkan resolusi pembacaan juga menjadi cukup kecil, jadi pergeseran sudut yang sangat kecil tidak akan terbaca.

Jarak sumber suara 20 cm

Tabel 4.8 Pengujian sudut dengan jarak antar mikrofon 10 cm dan jarak sumber suara 20 cm

Sudut tujuan (^{o})	Sudut yang terukur (^{o})	Error Sudut (%)
-90	-90,85	0,94
-70	-70,33	0,47
-50	-52,54	5,08
-30	-34,06	13,53
-10	-9,25	7,5
0	0	-
10	9,25	7,5
30	28,73	4,23
50	39,76	20,48
70	60,37	13,76
90	90,85	0,94

Pengujian berikutnya menggunakan sumber suara dengan jarak 20 cm dengan mikrofon, sudut yang dihasilkan masih akurat dan stabil, mikrofon masih dapat membaca suara pada jarak 20 cm dengan parabola dengan stabil dan akurat, terjadi beberapa kesalahan pada sudut 50^o dan 70^o sehingga hasil sudut menjadi sedikit tidak akurat

Jarak sumber suara 30 cm

Tabel 4.9 Pengujian sudut dengan jarak antar mikrofon 10 cm dan jarak sumber suara 30 cm

Sudut tujuan (°)	Sudut yang terukur (°)	Error Sudut (%)
-90	-90,85	0,94
-70	-70,33	0,47
-50	-52,54	5,08
-30	-28,73	4,23
-10	0	100
0	9,25	-
10	23,65	100
30	39,7	32,33
50	52,54	5,08
70	70,33	0,47
90	90,85	0,94

Pengujian berikutnya dilakukan dengan jarak sumber suara yang cukup jauh yaitu 30 cm, akan tetapi hasil sudut yang dihitung masih tetap akurat pada sudut-sudut yang besar seperti 70° dan 90°, pada posisi tengah atau tepat didepan kedua mikrofon terjadi *error* hasil perhitungan sehingga *error* pada sudut -10° sampai sudut 30° memiliki nilai yang cukup besar.

Jarak antar mikrofon 20 cm

Sama seperti sebelumnya, percobaan kali ini mengubah variabel jarak antara mikrofon menjadi 20 cm, perubahan ini akan berpengaruh pada resolusi pembacaan suara dan juga jarak sumber suara dengan masing-masing mikrofon.

Jarak sumber suara 10 cm

Tabel 4.10 Pengujian sudut dengan jarak antar mikrofon 20 cm dan jarak sumber suara 10 cm

Sudut tujuan (^{o})	Sudut yang terukur (^{o})	Error Sudut (%)
-90	-77,22	14,2
-70	-70,33	0,47
-50	-52,54	5,08
-30	-36,83	22,77
-10	-11,59	15,9
0	0	-
10	11,59	15,9
30	36,83	22,77
50	52,54	5,08
70	77,22	10,31
90	90,85	0,94

Dengan jarak antar mikrofon diubah menjadi 20 cm, sudut yang dihasilkan tidak berbeda jauh dengan jarak antar mikrofon 10 cm, akurasi masih cukup tinggi kecuali pada saat pengukuran -90^o, sedikit pergeseran sumber suara pada pengukuran sudut tersebut akan mengubah hasil perhitungan menjadi 77,22^o, jadi resolusi pembacaan masih belum meningkat secara signifikan.

Jarak sumber suara 20 cm

Tabel 4.11 Pengujian sudut dengan jarak antar mikrofon 20 cm dan jarak sumber suara 20 cm

Sudut tujuan (°)	Sudut yang terukur (°)	Error Sudut (%)
-90	Tidak beraturan	100
-70	-77,22	10,31
-50	-45,79	8,42
-30	-21,17	29,43
-10	-4,61	53,9
0	0	-
10	11,59	1,59
30	36,83	6,83
50	52,54	2,54
70	64,96	5,04
90	Tidak beraturan	100

Pada pengujian kali ini, nilai *error* semakin membesar terutama pada saat pengukuran 90° dan -90° dikarenakan jarak salah satu mikrofon dengan sumber suara yang terlalu jauh sehingga tidak dapat membaca sinyal dengan baik, selain pada sudut itu sudut yang dihasilkan masi memiliki akurasi yang cukup tinggi.

Jarak antar mikrofon 30 cm dengan jarak sumber suara 10 cm

Percobaan terakhir dilakukan dengan jarak antar mikrofon sebesar 30 cm dengan jarak antara sumber suara dan mikrofon 10 cm.

Tabel 4.12 Pengujian sudut dengan jarak antar mikrofon 30 cm dan jarak sumber suara 10 cm

Sudut tujuan (º)	Sudut yang terukur (º)	Error Sudut (%)
-90	Tidak beraturan	100
-70	Tidak beraturan	100
-50	Tidak beraturan	100
-30	-32,45	8,17
-10	-9,25	7,5
0	3,27	-
10	10,81	8,1
30	30,48	1,6
50	Tidak beraturan	100
70	Tidak beraturan	100
90	Tidak beraturan	100

Dari hasil pengujian diatas terdapat hasil yang tidak beraturan pada pengukuran sudut yang besar karena jarak sumber suara terlalu jauh dengan salah satu mikrofon, akan tetapi pada posisi dekat dengan sudut 0º dapat dilihat hasil sudut dengan akurasi yang cukup tinggi dan nilai *error* minimal, terlihat peningkatan resolusi pada pengukuran sudut tengah dengan peningkatan jarak antar mikrofon.

4.2.2 Jarak antar mikrofon 10 cm dengan Intensitas suara 60 dB

Percobaan kali ini dilakukan dengan menurunkan intensitas suara dari sumber untuk mengetahui intensitas minimal yang dapat terbaca oleh mikrofon.

Tabel 4.13 Pengujian sudut dengan jarak antar mikrofon 10 cm dan intensitas suara 60 dB

Sudut tujuan (^{o})	Sudut yang terukur (^{o})	Error Sudut (%)
-90	-45,79	49,12
-70	-34,06	51,34
-50	-28,73	42,54
-30	-18,73	37,57
-10	-9,25	7,5
0	0	-
10	9,25	7,5
30	23,65	21,17
50	34,06	31,88
70	52,54	24,94
90	70,33	21,86

Dari hasil pengujian dengan menurunkan intensitas suara, hasilnya mirip dengan pengujian dengan jarak sumber suara yang diperbesar, *error* sudut bernilai tinggi saat pengukuran sudut yang besar, dan seperti percobaan sebelumnya nilai *error* bernilai rendah saat pengukuran sudut yang kecil atau posisi diantara kedua mikrofon.

Intensitas suara 50 dB

Tabel 4.14 Pengujian sudut dengan jarak antar mikrofon 10 cm dan intensitas suara 50 dB

Sudut tujuan (º)	Sudut yang terukur (º)	Error Sudut (%)
-90	Tidak beraturan	100
-70	-45,79	34,59
-50	-34,06	31,88
-30	-23,65	21,17
-10	-9,25	7,5
0	0	-
10	9,25	7,5
30	18,73	37,57
50	28,73	42,54
70	45,79	34,59
90	70,33	21,86

Pada pengujian dengan intensitas suara kecil ini didapatkan hasil yang tidak jauh berbeda dengan percobaan sebelumnya, akan tetapi terdapat hasil perhitungan yang tidak stabil pada sudut -90º, hal ini dikarenakan kualitas kedua mikrofon yang berbeda.

4.3 Analisa

Berikut analisa yang diperoleh dari perbandingan hasil pengujian dengan perancangan pada bab sebelumnya antara lain:

1. Dalam penelitian ini dilakukan operasi pemrosesan sinyal sehingga membutuhkan kinerja prosesor dengan kecepatan tinggi, maka dari itu digunakan mikrokontroler jenis ARM agar kecepatan pemrosesan data pada prosesor dapat mengatasi input data yang diterima oleh sensor.

2. Tipe mikrofon yang digunakan akan berpengaruh pada hasil sinyal suara yang dihasilkan saat dibaca di komputer, kecepatan rambat suara pada udara yang sangat cepat membuat perbedaan waktu sampai suara dari sumber menuju mikrofon menjadi sangat kecil sehingga dibutuhkan jarak yang cukup untuk dapat membedakan waktu sampainya.

3. Selain tipe mikrofon, kuat suara yang dibaca oleh mikrofon juga berpengaruh pada akurasi hasil perhitungan sudut, pada proyek ini faktor *noise* diabaikan karena kuat suaranya yang kecil dan tidak berpengaruh pada hasil perhitungan sudut, jika mikrofon yang digunakan memiliki sensitifitas tinggi, faktor *noise* akan mengubah hasil perhitungan akhirnya.

4. Dilakukan percobaan menggunakan 2 buah sumber suara pada lokasi yang berbeda, hasil sudut yang didapat adalah sudut dengan posisi diantara kedua sumber suara. Sebagai contoh jika 2 buah sumber suara berada tepat didepan masing-masing mikrofon, hasil sudut yang dihasilkan bernilai 0^o karena sinyal yang diterima pada mikrofon tidak memiliki perbedaan fasa.

BAB 5
KESIMPULAN DAN SARAN

5.1 Kesimpulan

Pada Penelitian telah dirancang dan dibuat suatu alat pendeteksi arah sumber suara berdasarkan metode *Interaural Time Differnece* menggunakan mikrokontroler STM32F4. Berdasarkan data hasil uji implementasi sistem pendeteksi arah ini diperoleh beberapa kesimpulan antara lain sebagai berikut:

1. Dengan menggunakan 2 buah mikrofon sebagai sensor suara, jarak antar mikrofon yang dapat membaca sinyal dengan nilai *error* di bawah 10 derajat yaitu sebesar 10 cm.
2. Secara rata-rata keseluruhan hasil pengujian diperoleh error antara sudut pengujian dengan hasil perhitungan sebesar 38,64% dengan error terkecil pada 7,44% dan terbesar pada 87,11%.
3. Mikrofon kondenser yang digunakan pada proyek ini dapat membaca sinyal suara secara optimal pada intensitas suara 70 dB keatas.
4. Resolusi yang didapat dari pengujian dengan jarak antar mikrofon 10 cm dan intensitas suara 75 dB, yaitu bernilai sekitar 10^{o}.
5. Resolusi pembacaan sudut dengan 2 buah mikrofon dapat ditingkatkan dengan cara menjauhkan jarak antar mikrofon dengan kondisi mikrofon dapat membaca sinyal suara dengan baik.

5.2 Saran

Terkait dengan kendala dan kekurangan dalam penyusunan tugas akhir ini, ada beberapa hal yang dapat penulis sarankan untuk pengembangan selanjutnya. Antara lain sebagai berikut:

1. Penambahan jumlah mikrofon yang digunakan, sehingga sudut yang dapat dideteksi dapat melebihi 180 derajat.
2. Pemakaian tipe mikrofon yang lebih sensitif sehingga pembacaan suara bisa lebih jauh lagi.

Demikian saran yang dapat penulis sampaikan. Semoga dapat bermanfaat untuk ke depannya.

Halaman ini sengaja dikosongkan

DAFTAR PUSTAKA

[1] M Agung Nursyeha, Muhammad, "*Pengenalan Suara Burung menggunakan Mel Frequency Cepstrum Coefficient dan Jaringan Syaraf Tiruan pada Sistem Pengusir Hama Burung*", Institut Teknologi Sepuluh Nopember, Surabaya, 2016.

[2] Rusli, Meifal, John Malta, Irsyad, "*Prediksi Arah Sumber Suara untuk Perawatan Prediktif*", Prosiding Seminar Nasional Tahunan Teknik Mesin (SNTTM), Palembang, 2010.

[3] Kim, Yong-Eun, Dong-Hyun Su, Jin-Gyung Chung, Xinming Huang, Chul Dong Lee, "*Efficient Sound Source Localization Method Using Region Selection*", IEEE 2009

[4] Firdaus, Nor Ain, "*Alat Pelacak Arah Suara pada Sistem Pengusir Hama Burung Menggunakan ARM STM32F4*", Institut Teknologi Sepuluh Nopember, Surabaya, 2015.

Halaman ini sengaja dikosongkan

LAMPIRAN

```
#include "main.h"
#include "stm32f4xx_hal.h"

/* USER CODE BEGIN Includes */

#include "grogalan_stm32_dma_multi_buffer.h"
#include <stdio.h>
#include "defines.h"
#include "tm_stm32_ssd1306.h"
#include "arm_math.h"
#include "tm_stm32_fft.h"
#include "grogalan_itd.h"

char Buffer_LCD[25];
uint8_t CurMem = 0;
uint16_t My_ADC1 = 0;
uint16_t My_ADC2 = 0;
uint8_t My_ADC1_buffer_flag = 0;
uint32_t counter_XFER_M1CPLT = 0;
uint32_t counter_ConvCplt = 0;
uint16_t ADC1_Buffer0_Processed = 0;
uint16_t ADC1_Buffer1_Processed = 0;
uint32_t FFT_Freq = 0;
uint32_t FFT_Freq1 = 0;
uint32_t FFT_MaxValue = 0;
float32_t Output_FFT_Buff[FFT_SIZE];

uint8_t Buffer0_ADC1_Flag = 0;
uint8_t Buffer0_ADC2_Flag = 0;

uint8_t Buffer1_ADC1_Flag = 0;
uint8_t Buffer1_ADC2_Flag = 0;

/* Global variables CORRELATION*/
GROGALAN_CORR_F32_t CORR;
#define CORR_SAMPLE_SIZE      512
#define CORR_OUTPUT_SIZE      ((2*CORR_SAMPLE_SIZE)-1)
float32_t Input0[CORR_SAMPLE_SIZE];
float32_t Input1[CORR_SAMPLE_SIZE];
float32_t Output_Corr[CORR_OUTPUT_SIZE];

float32_t sudut_suara = 0.0;
float32_t time_delay = 0.0;
uint32_t previousMillis = 0;


// IIni parameter Vsound dan jarak yg bisa d ganti2
```

```
#define DELTA   (1/FS)
#define ZERO_DEGREES_INDEX_X_CORR_OUTPUT      511
#define V_SOUNDS                        384.0          //
kecepatan suara 384 m/s di ruangan dg temperatur 24 derajat
celcius
#define JARAK_ANTAR_MIC                 0.3     // dalam satuan
meter

/* Variable used to get converted value */
__IO uint16_t uhADC1ConvertedValue = 0;
__IO uint16_t dmabuffer[2][CORR_SAMPLE_SIZE];//DMA Double
Buffer
__IO uint16_t dmabuffer1[2][CORR_SAMPLE_SIZE];//DMA Double
Buffer

int main(void)
{

  /* USER CODE BEGIN 1 */
        //uint32_t i = 0;
        uint32_t freq = 20;

  HAL_Init();
  SystemClock_Config();
  MX_GPIO_Init();
  MX_DMA_Init();
  MX_ADC1_Init();
  MX_TIM2_Init();
  MX_USART1_UART_Init();
  MX_TIM1_Init();
  MX_I2C1_Init();
  MX_ADC2_Init();
  MX_DAC_Init();
  MX_TIM6_Init();
        TM_DELAY_Init();

        /* Init SSD1306 LCD 128 x 64 px */
        if (TM_SSD1306_Init()) {
                /* SSD1306 is connected */
                HAL_GPIO_WritePin(GPIOD, GPIO_PIN_12,
GPIO_PIN_SET);
        } else {
                /* SSD1306 is not connected */
                Error_Handler();
        }

        GROGALAN_CORR_Init_F32(&CORR, CORR_SAMPLE_SIZE,
CORR_SAMPLE_SIZE);
        GROGALAN_CORR_SetBuffers_F32(&CORR, Input0, Input1,
Output_Corr);
```

```
        TM_SSD1306_UpdateScreen();
        if(HAL_TIM_Base_Start_IT(&htim6) != HAL_OK){
                Error_Handler();
        }
        HAL_DAC_Start_DMA(&hdac, DAC_CHANNEL_1,
(uint32_t*)Signal_Sine, SINE_RES, DAC_ALIGN_12B_R);
        if(HAL_TIM_PWM_Start(&htim1, TIM_CHANNEL_1) !=
HAL_OK){
                Error_Handler();
        }

        __HAL_TIM_SET_COMPARE(&htim1, TIM_CHANNEL_1, 1800);

        //multi buffer
        if(Grogalan_HAL_ADC_Start_DMA_MultiBuffer_IT(&hadc1,
(uint32_t)ADC1_DR_ADDRESS, (uint32_t)&dmabuffer[0][0]
,(uint32_t)&dmabuffer[1][0] ,CORR_SAMPLE_SIZE) != HAL_OK){
                TM_SSD1306_GotoXY(0, 0);
                TM_SSD1306_Puts("Error Multi Buff",
&TM_Font_7x10, SSD1306_COLOR_WHITE);
                TM_SSD1306_UpdateScreen();
                while(1);
        }

        if(Grogalan_HAL_ADC_Start_DMA_MultiBuffer_IT(&hadc2,
(uint32_t)ADC2_DR_ADDRESS, (uint32_t)&dmabuffer1[0][0]
,(uint32_t)&dmabuffer1[1][0] ,CORR_SAMPLE_SIZE) != HAL_OK){
                TM_SSD1306_GotoXY(0, 0);
                TM_SSD1306_Puts("Error Multi Buff ADC2",
&TM_Font_7x10, SSD1306_COLOR_WHITE);
                TM_SSD1306_UpdateScreen();
                while(1);
        }

        //start timer2
        if(HAL_TIM_Base_Start_IT(&htim2) != HAL_OK){
                Error_Handler();
        }

        /* Go to location X = 30, Y = 4 */
        TM_SSD1306_GotoXY(0, 0);
        TM_SSD1306_Puts("HAL", &TM_Font_7x10,
SSD1306_COLOR_WHITE);
        TM_SSD1306_UpdateScreen();

        previousMillis = HAL_GetTick();
  while (1)
  {
          currentMillis = HAL_GetTick();
```

```
          if (currentMillis - previousMillis >= 50) {
                previousMillis = currentMillis;

                if(!isnan(sudut_suara)){
                        TM_SSD1306_GotoXY(0, 0);
                        sprintf(Buffer_LCD, "sudut:%.2f    ",
sudut_suara);
                        TM_SSD1306_Puts(Buffer_LCD,
&TM_Font_7x10, SSD1306_COLOR_WHITE);

                        TM_SSD1306_GotoXY(0, 20);
                        sprintf(Buffer_LCD, "a[0]:%.2f    ",
(float32_t)My_ADC1);
                        TM_SSD1306_Puts(Buffer_LCD,
&TM_Font_7x10, SSD1306_COLOR_WHITE);

                        TM_SSD1306_GotoXY(0, 30);
                        sprintf(Buffer_LCD, "b[1]:%.2f    ",
(float32_t)My_ADC2);
                        TM_SSD1306_Puts(Buffer_LCD,
&TM_Font_7x10, SSD1306_COLOR_WHITE);

                          /* Update screen */
                          TM_SSD1306_UpdateScreen();
                }
          }

  }
}

/* ADC1 init function */
static void MX_ADC1_Init(void)
{

  ADC_MultiModeTypeDef multimode;
  ADC_ChannelConfTypeDef sConfig;

    /**Configure the global features of the ADC (Clock,
Resolution, Data Alignment and number of conversion)
    */
  hadc1.Instance = ADC1;
  hadc1.Init.ClockPrescaler = ADC_CLOCK_SYNC_PCLK_DIV4;
  hadc1.Init.Resolution = ADC_RESOLUTION_12B;
  hadc1.Init.ScanConvMode = DISABLE;
  hadc1.Init.ContinuousConvMode = DISABLE;
  hadc1.Init.DiscontinuousConvMode = DISABLE;
  hadc1.Init.ExternalTrigConvEdge =
ADC_EXTERNALTRIGCONVEDGE_RISING;
  hadc1.Init.ExternalTrigConv =
ADC_EXTERNALTRIGCONV_T2_TRGO;
```

```
  hadc1.Init.DataAlign = ADC_DATAALIGN_RIGHT;
  hadc1.Init.NbrOfConversion = 1;
  hadc1.Init.DMAContinuousRequests = ENABLE;
  hadc1.Init.EOCSelection = ADC_EOC_SINGLE_CONV;
  if (HAL_ADC_Init(&hadc1) != HAL_OK)
  {
    Error_Handler();
  }

    /**Configure the ADC multi-mode
    */
  multimode.Mode = ADC_DUALMODE_REGSIMULT;
  multimode.DMAAccessMode = ADC_DMAACCESSMODE_2;
  multimode.TwoSamplingDelay = ADC_TWOSAMPLINGDELAY_5CYCLES;
  if (HAL_ADCEx_MultiModeConfigChannel(&hadc1, &multimode)
!= HAL_OK)
  {
    Error_Handler();
  }
  sConfig.Channel = ADC_CHANNEL_1;
  sConfig.Rank = 1;
  sConfig.SamplingTime = ADC_SAMPLETIME_3CYCLES;
  if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
  {
    Error_Handler();
  }
}
/* ADC2 init function */
static void MX_ADC2_Init(void)
{
  ADC_ChannelConfTypeDef sConfig;
  hadc2.Instance = ADC2;
  hadc2.Init.ClockPrescaler = ADC_CLOCK_SYNC_PCLK_DIV4;
  hadc2.Init.Resolution = ADC_RESOLUTION_12B;
  hadc2.Init.ScanConvMode = DISABLE;
  hadc2.Init.ContinuousConvMode = DISABLE;
  hadc2.Init.DiscontinuousConvMode = DISABLE;
  hadc2.Init.DataAlign = ADC_DATAALIGN_RIGHT;
  hadc2.Init.NbrOfConversion = 1;
  hadc2.Init.DMAContinuousRequests = ENABLE;
  hadc2.Init.EOCSelection = ADC_EOC_SINGLE_CONV;
  if (HAL_ADC_Init(&hadc2) != HAL_OK)
  {
    Error_Handler();
  }
  sConfig.Channel = ADC_CHANNEL_2;
  sConfig.Rank = 1;
  sConfig.SamplingTime = ADC_SAMPLETIME_3CYCLES;
  if (HAL_ADC_ConfigChannel(&hadc2, &sConfig) != HAL_OK)
  {
```

```
    Error_Handler();
  }

}

/* I2C1 init function */
static void MX_I2C1_Init(void)
{

  hi2c1.Instance = I2C1;
  hi2c1.Init.ClockSpeed = 100000;
  hi2c1.Init.DutyCycle = I2C_DUTYCYCLE_2;
  hi2c1.Init.OwnAddress1 = 0;
  hi2c1.Init.AddressingMode = I2C_ADDRESSINGMODE_7BIT;
  hi2c1.Init.DualAddressMode = I2C_DUALADDRESS_DISABLE;
  hi2c1.Init.OwnAddress2 = 0;
  hi2c1.Init.GeneralCallMode = I2C_GENERALCALL_DISABLE;
  hi2c1.Init.NoStretchMode = I2C_NOSTRETCH_DISABLE;
  if (HAL_I2C_Init(&hi2c1) != HAL_OK)
  {
    Error_Handler();
  }

}

/* TIM2 init function */
static void MX_TIM2_Init(void)
{

  TIM_ClockConfigTypeDef sClockSourceConfig;
  TIM_MasterConfigTypeDef sMasterConfig;

  htim2.Instance = TIM2;
  htim2.Init.Prescaler = 83;
  htim2.Init.CounterMode = TIM_COUNTERMODE_UP;
  htim2.Init.Period = 20 - 1; //100khz
  htim2.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
  if (HAL_TIM_Base_Init(&htim2) != HAL_OK)
  {
    Error_Handler();
  }

  sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
  if (HAL_TIM_ConfigClockSource(&htim2, &sClockSourceConfig)
!= HAL_OK)
  {
    Error_Handler();
  }

  sMasterConfig.MasterOutputTrigger = TIM_TRGO_UPDATE;
```

```
  sMasterConfig.MasterSlaveMode =
TIM_MASTERSLAVEMODE_DISABLE;
  if (HAL_TIMEx_MasterConfigSynchronization(&htim2,
&sMasterConfig) != HAL_OK)
  {
    Error_Handler();
  }

}

/* USART1 init function */
static void MX_USART1_UART_Init(void)
{

  huart1.Instance = USART1;
  huart1.Init.BaudRate = 115200;
  huart1.Init.WordLength = UART_WORDLENGTH_8B;
  huart1.Init.StopBits = UART_STOPBITS_1;
  huart1.Init.Parity = UART_PARITY_NONE;
  huart1.Init.Mode = UART_MODE_TX_RX;
  huart1.Init.HwFlowCtl = UART_HWCONTROL_NONE;
  huart1.Init.OverSampling = UART_OVERSAMPLING_16;
  if (HAL_UART_Init(&huart1) != HAL_OK)
  {
    Error_Handler();
  }

}

/**
  * Enable DMA controller clock
  */
static void MX_DMA_Init(void)
{
  /* DMA controller clock enable */
  __HAL_RCC_DMA1_CLK_ENABLE();
  __HAL_RCC_DMA2_CLK_ENABLE();

  /* DMA interrupt init */
  /* DMA1_Stream5_IRQn interrupt configuration */
  HAL_NVIC_SetPriority(DMA1_Stream5_IRQn, 0, 0);
  HAL_NVIC_EnableIRQ(DMA1_Stream5_IRQn);
  /* DMA2_Stream0_IRQn interrupt configuration */
  HAL_NVIC_SetPriority(DMA2_Stream0_IRQn, 0, 0);
  HAL_NVIC_EnableIRQ(DMA2_Stream0_IRQn);
  /* DMA2_Stream2_IRQn interrupt configuration */
  HAL_NVIC_SetPriority(DMA2_Stream2_IRQn, 0, 0);
  HAL_NVIC_EnableIRQ(DMA2_Stream2_IRQn);

}
```

```
/** Configure pins as
        * Analog
        * Input
        * Output
        * EVENT_OUT
        * EXTI
*/
static void MX_GPIO_Init(void)
{

  GPIO_InitTypeDef GPIO_InitStruct;

  /* GPIO Ports Clock Enable */
  __HAL_RCC_GPIOH_CLK_ENABLE();
  __HAL_RCC_GPIOA_CLK_ENABLE();
  __HAL_RCC_GPIOB_CLK_ENABLE();
  __HAL_RCC_GPIOE_CLK_ENABLE();
  __HAL_RCC_GPIOD_CLK_ENABLE();

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(GPIOD, LD4_Pin|LD3_Pin|LD5_Pin|LD6_Pin,
GPIO_PIN_RESET);

  /*Configure GPIO pin : B1_Pin */
  GPIO_InitStruct.Pin = B1_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_IT_RISING;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  HAL_GPIO_Init(B1_GPIO_Port, &GPIO_InitStruct);

  /*Configure GPIO pin : BOOT1_Pin */
  GPIO_InitStruct.Pin = BOOT1_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  HAL_GPIO_Init(BOOT1_GPIO_Port, &GPIO_InitStruct);

  /*Configure GPIO pins : LD4_Pin LD3_Pin LD5_Pin LD6_Pin */
  GPIO_InitStruct.Pin = LD4_Pin|LD3_Pin|LD5_Pin|LD6_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(GPIOD, &GPIO_InitStruct);

}

void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef* hadc)
{
        float32_t TimeDelay;
        float32_t Angle_Degrees;
        uint32_t idx = 0;
```

```
        float32_t signal_val;

        if(hadc == &hadc1){
                Buffer0_ADC1_Flag = 1;
                do{
                        signal_val =
(float32_t)dmabuffer[0][idx] - (float32_t)2048.0;
                        idx++;
                }while (!GROGALAN_CORR_AddToBuffer0(&CORR,
signal_val));
                My_ADC1 = dmabuffer[0][2] -
(float32_t)2048.0;

        }
        else{
                Buffer0_ADC2_Flag = 1;
                do{
                        signal_val =
(float32_t)dmabuffer1[0][idx] - (float32_t)2048.0;
                        idx++;
                }while (!GROGALAN_CORR_AddToBuffer1(&CORR,
signal_val));
                My_ADC2 = dmabuffer1[0][2] -
(float32_t)2048.0;
        }

        if(Buffer0_ADC1_Flag && Buffer0_ADC2_Flag){
                HAL_GPIO_TogglePin(GPIOD, GPIO_PIN_13);

                GROGALAN_CORR_Process_F32(&CORR);
                /* lihat eq no. 5 */
                TimeDelay =
((float32_t)GROGALAN_CORR_GetMaxIndex(&CORR) -
(float32_t)ZERO_DEGREES_INDEX_X_CORR_OUTPUT) * 0.00002;
                Angle_Degrees =
asinf((TimeDelay*V_SOUNDS)/JARAK_ANTAR_MIC);

                time_delay = TimeDelay;
                //if(isnan(Angle_Degrees)){
                        sudut_suara =
map_to_float(Angle_Degrees, -1.5, 1.5, -90, 90);
                //}

                Buffer0_ADC1_Flag = 0;
                Buffer0_ADC2_Flag = 0;
        }
}
```

```
void HAL_ADC_ConvHalfCpltCallback(ADC_HandleTypeDef* hadc){


}

void HAL_ADC_DMA_XFER_M1CPLT_Callback(ADC_HandleTypeDef*
hadc){
        float32_t TimeDelay;
        float32_t Angle_Degrees;
        uint32_t idx = 0;
        float32_t signal_val;

        if(hadc == &hadc1){
                Buffer1_ADC1_Flag = 1;
                do{
                        signal_val =
(float32_t)dmabuffer[1][idx] - (float32_t)2048.0;
                        idx++;
                }while (!GROGALAN_CORR_AddToBuffer0(&CORR,
signal_val));
                My_ADC1 = dmabuffer[1][2] -
(float32_t)2048.0;

        }
        else{
                Buffer1_ADC2_Flag = 1;
                do{
                        signal_val =
(float32_t)dmabuffer1[1][idx] - (float32_t)2048.0;
                        idx++;
                }while (!GROGALAN_CORR_AddToBuffer1(&CORR,
signal_val));
                My_ADC2 = dmabuffer1[1][2] -
(float32_t)2048.0;
        }

        if(Buffer1_ADC1_Flag && Buffer1_ADC2_Flag){
                HAL_GPIO_TogglePin(GPIOD, GPIO_PIN_14);
                Buffer1_ADC1_Flag = 0;
                Buffer1_ADC2_Flag = 0;
        }
}

/* USER CODE END 4 */
void Error_Handler(void)
{
          while(1)
          {
                /* Toggle leds */
                HAL_GPIO_TogglePin(GPIOD, GPIO_PIN_15);
```

```
                /* Delay a little */
                Delayms(50);
            }
  /* USER CODE END Error_Handler */
}

#endif
```

Halaman Ini Sengaja Dikosongkan

DAFTAR RIWAYAT HIDUP

Mohamad Asfari. Lahir di Bandung, 19 Januari 1992. Putra dari Hoetomo Fadli dan Hilyah. Menamatkan pendidikan Sekolah Dasar di SD Priangan pada tahun 2003. Di tahun yang sama meneruskan pendidikan di SMPN 07 Bandung hingga 2007. Setelah itu masuk di SMAN 16 Bandung. Kemudian diterima di jurusan D3 Teknik Mekatronika, Politeknik Manufaktur Negeri Bandung, lulus pada tahun 2013 dan mengikuti program lintas jalur ke S1 Teknik Elektro di Institut Teknologi Sepuluh Nopember, mengambil program studi Elektronika.

E-mail :moh.asfari@yahoo.co.id

Halaman Ini Sengaja Dikosongkan