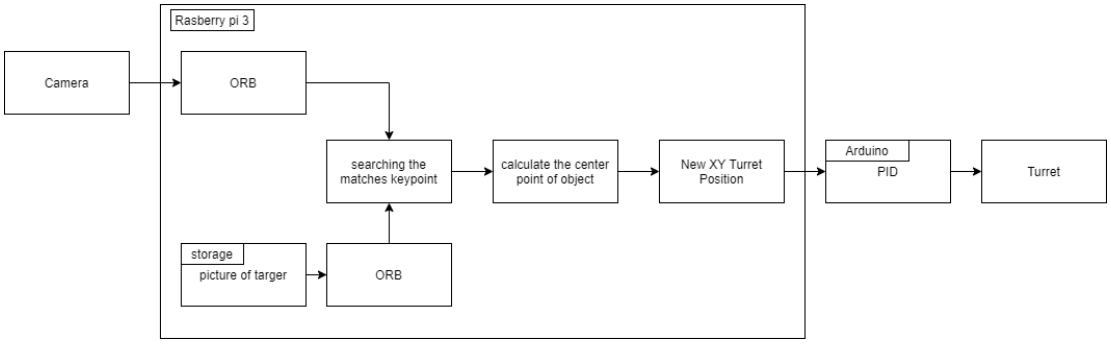
BAB III

METODE DAN PERANCANGAN SISTEM

3.1 Diagram Blok Sistem

Sistem ini dirancang untuk mencari mengubah posisi meriam pada prototipe tank agar mengikuti target yang sudah ditentukan. Sistem ini terdiri dari 2 *controller* yaitu, Raspberry Pi 3 dan Arduino, aktuator berupa 3 servo yang mengatur meriam dan sebagai sensor, terdapat kamera yang digunakan untuk melihat area di depan prototipe tank secara langsung dan *gyroscope* sebagai *feedback* yang digunakan untuk sistem kendali PID.

Pada Gambar 3.1, *frame* yang didapatkan oleh kamera dan Gambar dari objek yang menjadi target akan diproses dengan ORB lalu *keypoints* yang didapatkan dari Gambar objek akan dicocokkan ke *keypoints* yang didapatkan dari *frame* kamera. Jika benda yang menjadi target berada didalam *frame* tersebut, maka Raspberry Pi akan mencari titik tengah benda dan nilai titik tengah benda akan menjadi nilai X dan Y baru untuk Arduino yang akan menggerakan servo – servo meriam. Raspberry Pi diperlukan karena metode ORB terdapat dalam *library* di openCV dan untuk menggunakan openCV diperlukan sebuah sistem operasi dan arduino UNO atau MEGA tidak memiliki sistem operasi didalamnya, salah satu komputer mini yang dapat digunakan adalah Raspberry Pi.

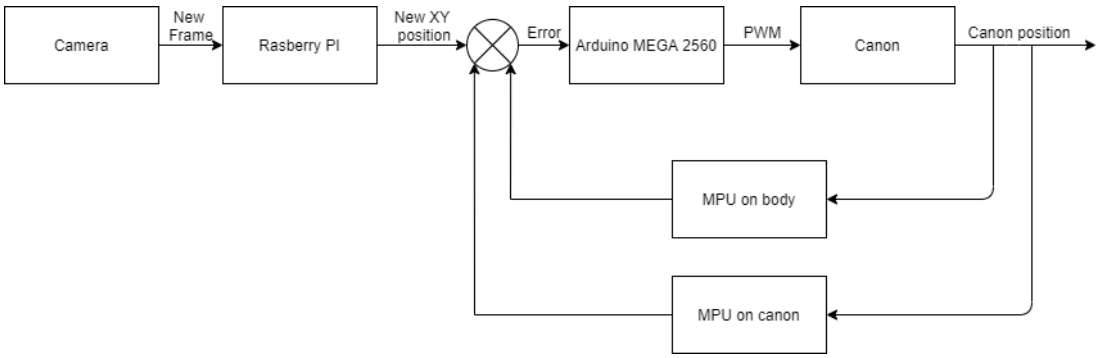


Gambar 3.1 *Architecture* Block Diagram

Sistem ini bekerja dengan cara *frame* yang didapat oleh kamera akan dikirimkan ke Raspberry Pi secara terus – menerus. Di dalam Raspberry Pi, *Frame* yang diterima akan diproses melalui metode ORB dan menghasilkan suatu nilai, nilai tersebut adalah *keypoints* kamera. Begitu juga dengan Gambar objek yang menjadi target, akan diproses melalui metode ORB dan menghasilkan *keypoints* objek. Lalu *keypoints* objek akan dicari didalam *keypoints* kamera, jika ditemukan kecocokkan, itu berarti objek ditemukan didalam objek kamera dan dicari titik tengah dari objek tersebut. Titik tengah tersebut dikirimkan ke Arduino Mega sebagai nilai X dan Y baru.

Nilai yang diterima oleh arduino akan dikonversi terlebih dahulu, dimana pada Arduino Mega, nilai X yang diterima akan mempengaruhi servo Z dan nilai Y akan mempengaruhi servo X. Ini terjadi dikarenakan perbedaan *point of view* pada sistem Raspberry Pi dengan Arduino Mega dimana servo Z pada arduino akan mengatur pergerakan secara horizontal, dan servo X akan mengatur pergerakan secara vertikal. Nilai X dan Y tersebut akan menjadi titik acuan dari posisi meriam. Setelah itu, posisi tersebut itu akan dikonversi menjadi nilai PWM (pulse width modulation) untuk menggerakkan servo sebagai aktuatornya. Sensor *gyroscope* berfungsi untuk melakukan pengecekan apakah ada perubahan sudut atau posisi

pada tank. Bacaan dari sensor *gyroscope* yang berupa derajat per detik nantinya akan dikonversikan menjadi posisi sudut, dan setelah itu akan dibandingkan dengan posisi sudut yang sudah menjadi acuan untuk dilihat nilai perbedaannya. Apabila didapatkan nilai yang berbeda antara bacaan sensor gyroscope dan posisi sudut acuan, maka *controller* Arduino Mega akan melakukan kalkulasi sekaligus mengoreksi nilai dari PWM untuk menggerakkan servo ke sudut yang sudah ditentukan sebelumnya. Diagram blok sistem ini dapat dilihat pada Gambar 3.2.

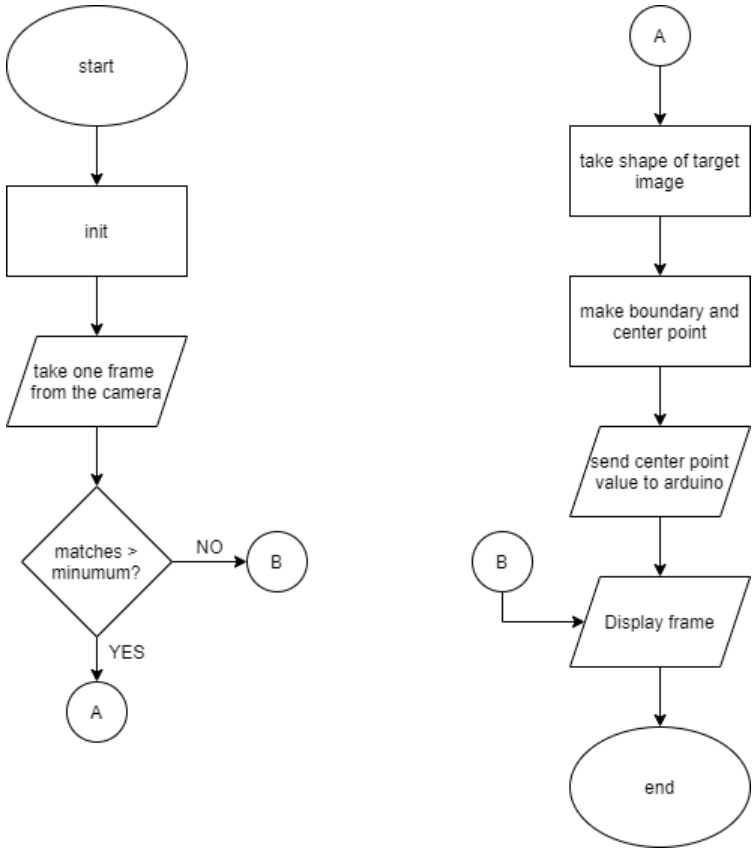


Gambar 3.2 Diagram blok sistem

3.2 Flowchart Sistem

Terdapat dua sistem pada sistem ini, sistem yang berada di Raspberry Pi dan sistem yang berada di Arduino Mega. Pada Raspberry Pi, sistem pertama – tama akan melakukan proses inisialisasi, dimana sistem akan mengambil semua *library* yang dibutuhkan, mengatur resolusi tangkapan kamera, menyalakan kamera, menyiapkan modul ORB dengan mengatur maksimal *keypoints* yang bisa diambil, *scaling*-nya, dan mencari set *keypoints* dari Gambar objek yang menjadi target, dan jumlah minimum kecocokkan pada kedua set *keypoints*. Selanjutnya, sistem akan mengambil satu *frame* dari kamera dan mencari set *keypoints* yang terdapat pada *frame* tersebut. Setelah itu, *keypoints* dari Gambar objek akan dicocokkan ke *keypoints* dari *frame*, dan dihitung jumlah kecocokkan pada kedua set *keypoints*.

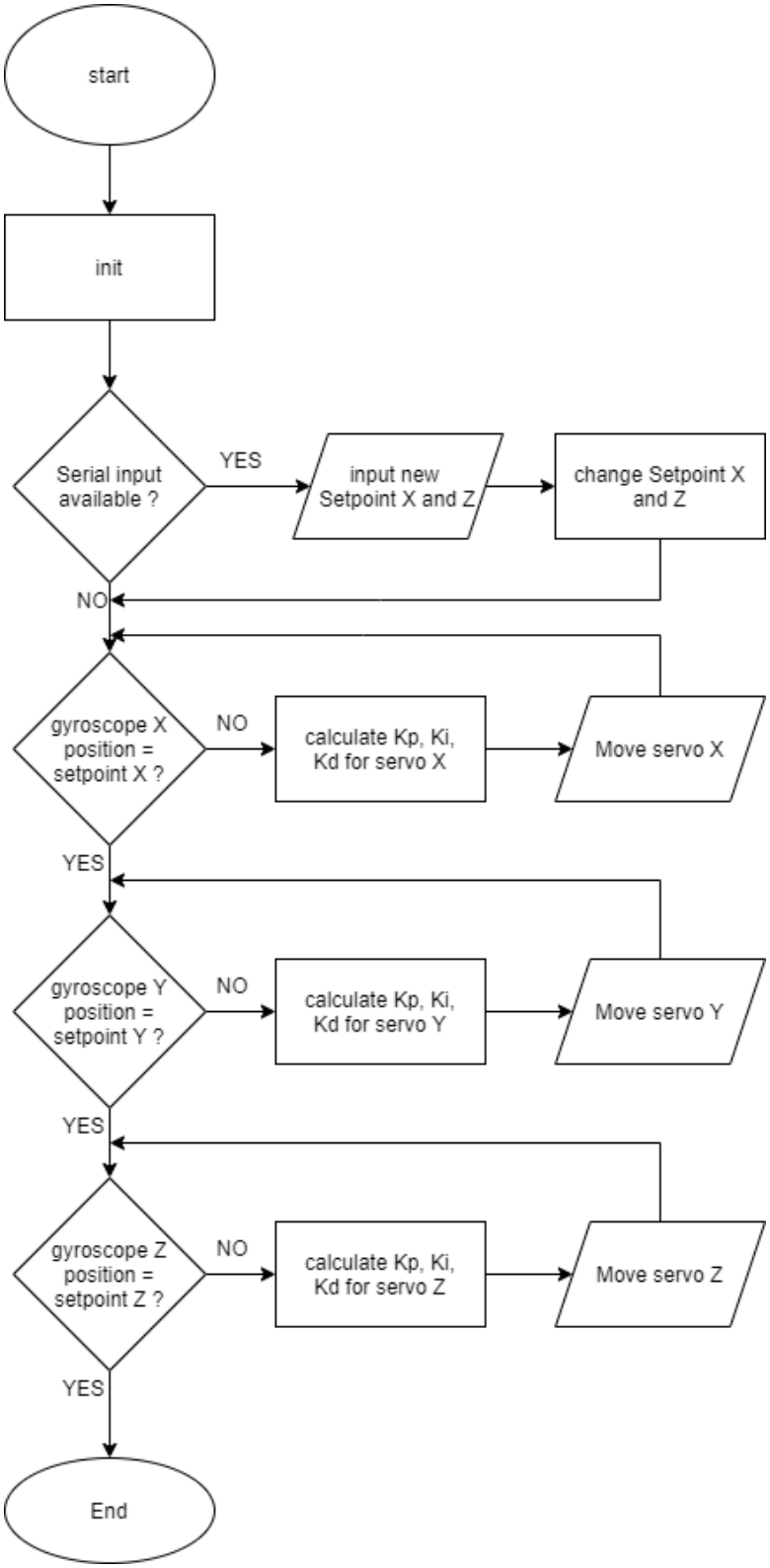
Apabila jumlah *keypoints* yang cocok kurang dari minimum maka sistem akan menampilkan *frame* ke *display*, dan program akan mengulang dari awal dan mengambil *frame* baru dari kamera. Apabila jumlah *keypoints* yang cocok telah melebihi jumlah minimum yang ditentukan, maka program akan mengambil bentuk dari Gambar target untuk membuat batasan dan titik tengah pada *frame* kamera yang akan menjadi *display* untuk ditampilkan pada monitor. Setelah itu, nilai titik tengah yang didapatkan akan dikirimkan ke Arduino Mega. Flowchart pada sistem Raspberry Pi dapat dilihat pada Gambar 3.3.



Gambar 3.3 Flowchart sistem Raspberry Pi

Pada Arduino Mega, sistem akan melakukan proses inisalisasi, dimana Arduino Mega akan mengambil *library* yang dibutuhkan, mengatur semua pin yang digunakan, memberi nilai awal untuk setpoint servo X, Y, dan Z, dan menyiapkan

parameter untuk sistem kendali PID. Selanjutnya mencoba komunikasi dengan semua perangkat yang terhubung, seperti servo dan *gyroscope*. Lalu, Arduino Mega akan mengecek apakah ada input dari komunikasi serial. Jika ada input dari komunikasi serial, maka nilai X dan nilai Y yang terkirim akan mengganti nilai *setpoint* X dan Z dimana nilai X akan mengganti *setpoint* Z dan nilai Y akan mengganti *setpoint* X. Ini terjadi dikarenakan perbedaan *point of view* pada sistem Raspberry Pi dengan Arduino Mega dimana servo Z pada arduino akan mengatur pergerakan secara horizontal, dan servo X akan mengatur pergerakan secara vertikal. Jika tidak ada input, maka akan tetap memakai *setpoint* yang sudah ada. Selanjutnya, *gyroscope* yang bekerja sebagai sensor posisi akan mengecek apakah posisi dari servo X sudah sesuai dengan nilai *setpoint* atau belum. Apabila terjadi perbedaan nilai, maka kontroler akan melakukan proses kalkulasi nilai kp, ki dan kd untuk nantinya nilai tersebut digunakan sebagai nilai PID pada servo. Setelah itu, sistem akan melakukan pengecekan untuk posisi sumbu Y dan sumbu Z. Apabila didapatkan adanya perbedaan dengan nilai acuan, maka kontroler akan melakukan proses yang sama dengan yang telah disebutkan diatas. Proses tersebut akan dilakukan secara berulang-ulang agar posisi dari meriam yang melalui sumbu X, sumbu Y dan sumbu Z akan selalu stabil pada nilai acuan yang telah ditentukan sebelumnya. Flowchart pada sistem Arduino Mega dapat dilihat pada Gambar 3.4.



Gambar 3.4 Flowchart sistem Arduino Mega

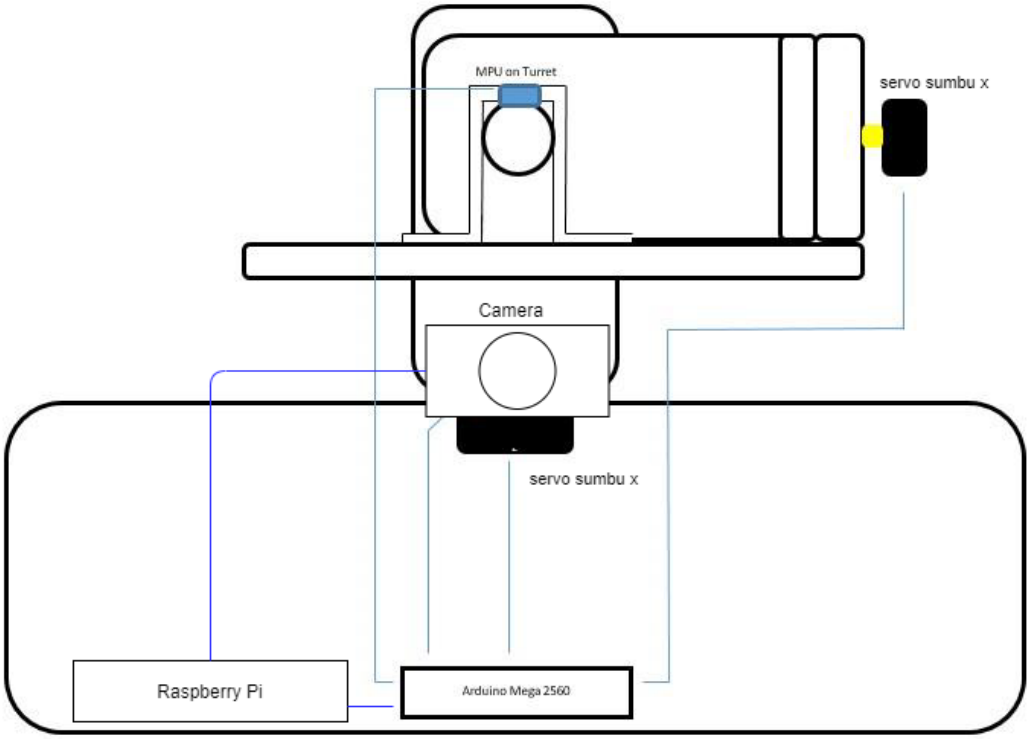
3.3 Perancangan *Hardware*

Pada perancangan *hardware* terdiri dari perancangan fisik sistem, penyesuaikan tangkapan kamera dengan pergerakan servo, dan perancangan sistem kendali PID pada Arduino Mega.

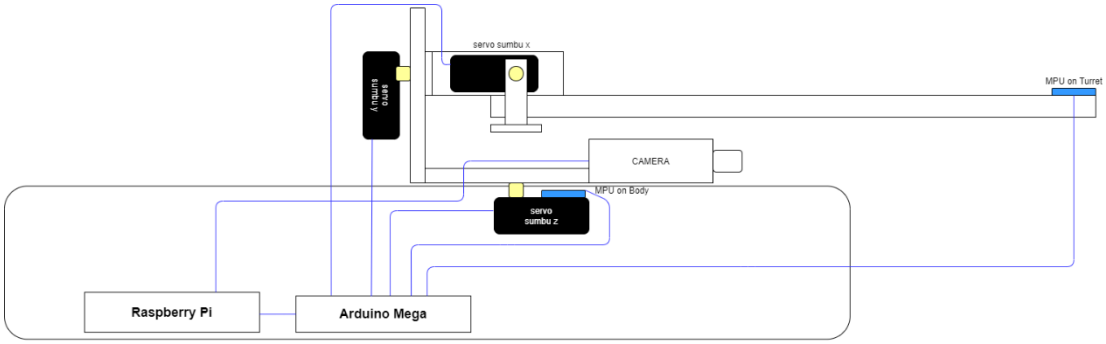
3.3.1 Perancangan fisik sistem

Terdapat beberapa komponen fisik yang dimiliki oleh sistem, diantaranya adalah satu Raspberry Pi, satu kamera yang dipasang pada bagian depan tank, satu Arduino Mega, dua sensor *gyroscope* yang terpasang pada badan tank dan ujung meriam, dan tiga servo yang menggerakan tiga sumbu X, Y, dan Z. Raspberry Pi 3 model B bertindak sebagai komputer kecil yang berfungsi memroses Gambar yang diterima dari kamera dan mencari target dalam Gambar tersebut. Kamera Logitech C920 yang diletakkan pada bagian depan tank digunakan sebagai sensor yang mencari objek target. Kamera Logitech C920 dihubungkan dengan Raspberry Pi melalui kabel USB. Arduino Mega berfungsi sebagai *microcontroller* yang mengatur sistem keseimbangan pada posisi meriam. *Gyroscope* yang terpasang pada badan tank berfungsi sebagai sensor *feedback* dengan mengirimkan posisi badan tank kepada Arduino Mega yang terhubung dengan pin I2C. *gyroscope* yang terpasang pada ujung meriam berfungsi sebagai sensor *feedback* dengan mengirimkan posisi ujung meriam kepada Arduino Mega yang terhubung dengan pin I2C. Servo sumbu X berfungsi sebagai penggerak meriam pada posisi koordinat sumbu X. Servo sumbu X terhubung dengan pin 8 pada Arduino Mega. Servo sumbu Y berfungsi sebagai penggerak meriam pada posisi koordinat sumbu

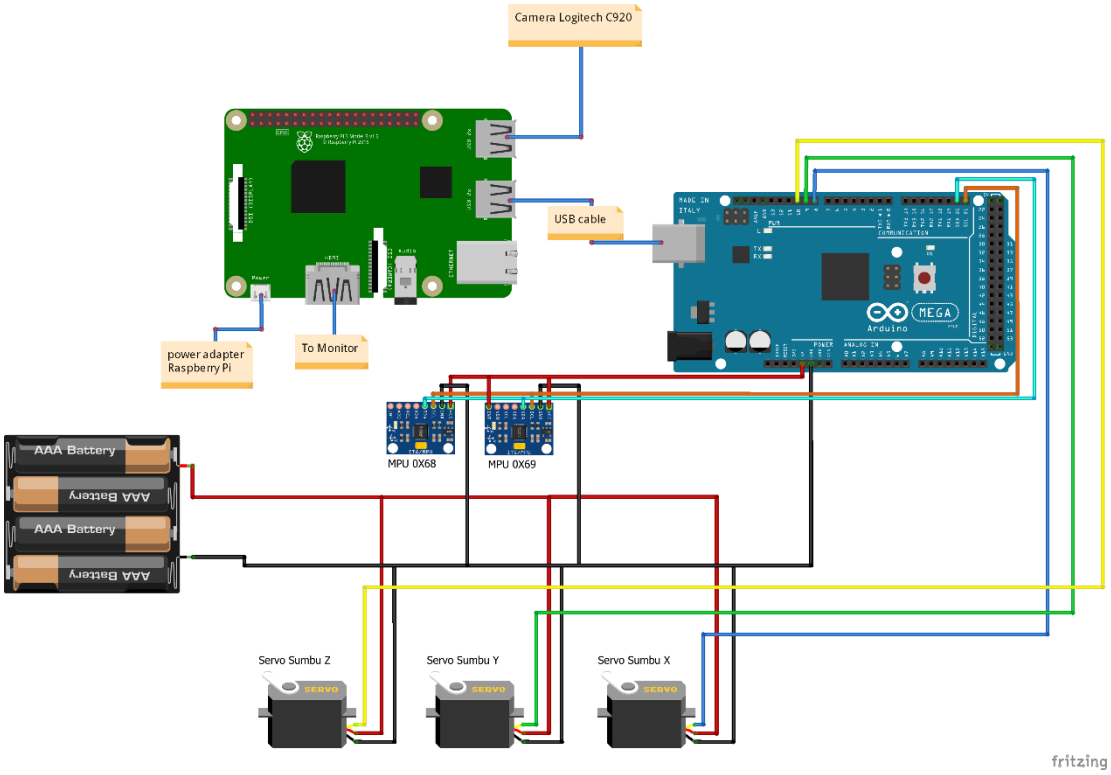
Y. Servo sumbu Y terhubung dengan pin 9 pada Arduino Mega. Servo sumbu Z berfungsi sebagai penggerak meriam pada posisi koordinat sumbu Z. Servo sumbu Z terhubung dengan pin 10 pada Arduino Mega.



Gambar 3.5 Mekanisme kendali pada prototipe tank (tampak depan)



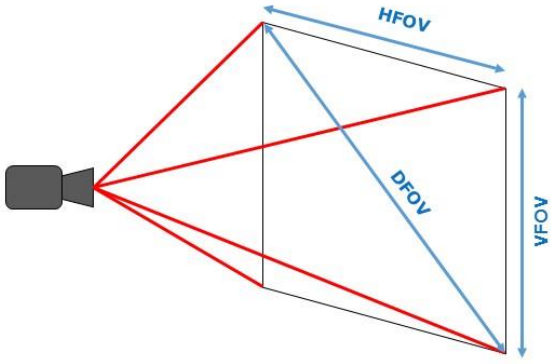
Gambar 3.6 Mekanisme kendali pada prototipe tank (tampak samping)



Gambar 3.7 Diagram sirkuit prototipe tank

3.3.2 Penyesuaian tangkapan kamera dengan pergerakan servo

Pada kamera Logitech C920 memiliki *aspect ratio* 16:9 dan *diagonal field of view* (DFOV) sebesar 78 derajat. Karena servo bergerak searah dengan *horizontal field of view* (HFOV) dan *vertical field of view* (VFOV) maka DFOV yang dimiliki oleh kamera harus diubah ke dalam HFOV dan VFOV terlebih dahulu sebelum dapat digunakan.



Gambar 3.8 perbedaan DFOV, HFOV, dan VFOV

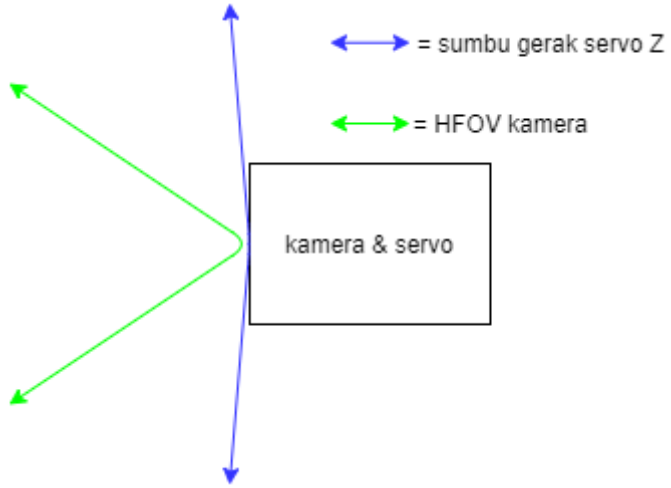
Perhitungan yang digunakan untuk menghitung HFOV dan VFOV dari DFOV adalah:

$$HFOV = 2 \times \arctan\left(\tan\left(\frac{DFOV}{2}\right) \times \cos\left(arctan\left(\frac{9}{16}\right)\right)\right) \quad (16)$$

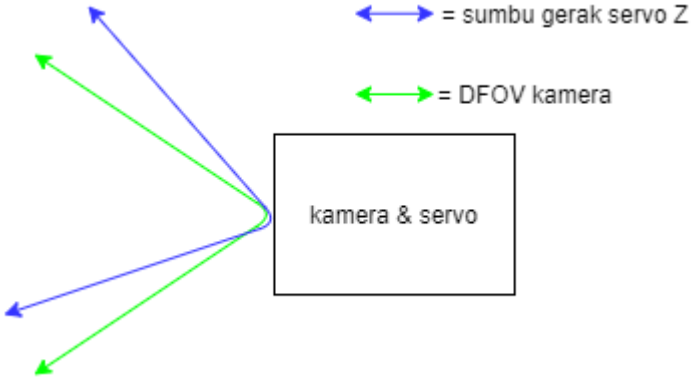
$$dan$$

$$VFOV = 2 \times \arctan\left(tan\left(\frac{DFOV}{2}\right) \times sin\left(arctan\left(\frac{9}{16}\right)\right)\right) \quad (17)$$

Dengan menggunakan persamaan (16) dan (17) maka HFOV dan VFOV yang dimiliki oleh kamera Logitech C920 adalah 70,43 dan 43,3 derajat. Untuk penyesuaian kamera dengan pergerakkan servo, pada servo sumbu Z menggunakan HFOV karena servo bergerak pada sumbu horizontal kamera. Pada servo sumbu X menggunakan VFOV karena servo bergerak pada sumbu vertikal kamera. Diasumsikan kamera dengan meriam berada pada satu titik sumbu yang sama, pada sumbu horizontal, karena pergerakan servo sumbu Z dapat bergerak sebesar 170 derajat (5 sampai 175 derajat) dan HFOV kamera sebesar 70,43 derajat (54,785 sampai 125,215 derajat), maka untuk menyesuaikan pergerakan servo dengan kamera, servo hanya dapat bergerak dari 54,785 sampai 125,215 derajat. Pada sumbu vertikal, terdapat perbedaan sudut pandangan pada sumbu vertikal, dimana pergerakan servo sumbu X hanya dapat bergerak sebesar 45 derajat (75 sampai 120 derajat) dan VFOV kamera sebesar 43,3 derajat (68,35 sampai 111,65 derajat) maka ketika kamera melihat target di bawah 75 derajat, servo sumbu X hanya dapat bergerak sampai 75 derajat, dan untuk pergerakan diatas 90 derajat, servo hanya dapat bergerak sampai 111,65 derajat.



Gambar 3.9 Horizontal *point of view*



Gambar 3.10 Vertikal *point of view*

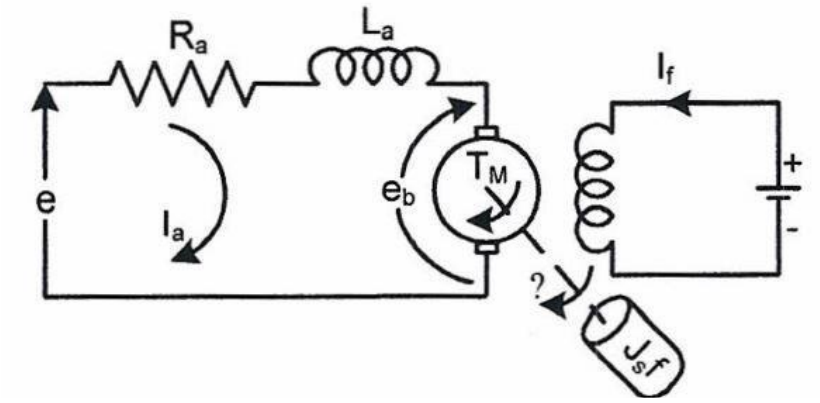
3.3.3 Perancangan sistem kendali PID pada Arduino Mega

Untuk perancangan sistem kendali PID, akan terbagi menjadi beberapa bagian, yaitu:

1. Blok diagram servo
2. Fungsi alih servo
3. Pemodelan sistem kendali PID

3.3.3.1 Blok diagram servo

Pada dasarnya, motor servo memiliki konstruksi dasar seperti motor dc. Dapat dilihat pada Gambar 3.11.



Gambar 3.11 Rangkaian listrik servo

Torsi motor (Tm) akan memiliki nilai yang sebanding dengan hasil perkalian antara fluksi celah udara dengan arus jangkar (Ia), sehingga akan membentuk persamaan :

$$Tm = K_I \times i_a \times K_f \times i_f \qquad (18)$$

Dimana K_I merupakan suatu konstanta.

Karena arus medan (i_f) selalu terjaga konstan, maka persamaan (20) dapat ditulis juga seperti:

$$Tm = K_{tm} \times i_a \qquad (19)$$

Dimana K_{tm} adalah konstanta dari torsi motor.

Lalu, emf lawan (e_b) pada motor akan selalu bernilai sebanding dengan kecepatannya. Sehingga persamannya adalah:

$$e_b = K_b \times \frac{d\theta}{dt} \qquad (20)$$

Dengan K_b sebagai konstanta emf lawan.

Lalu, didapatkan persamaan diferensial dan persamaan torsinya seperti pada persamaan (21) dan (22):

$$L_a \frac{di_a}{dt} + R_a \times i_a + e_b = e \qquad (21)$$

$$dan$$

$$J\frac{d\theta^2}{dt^2} + f_o\frac{d\theta}{dt} = Tm = K_{tm} \times i_a \qquad (22)$$

Dengan menggunakan transformasi Laplace pada persamaan (20), (21), dan (22) dengan mengganggap syarat awal = 0, maka didapatkan persamaan:

$$E_b(s) = K_b S\theta(s) \qquad (23)$$

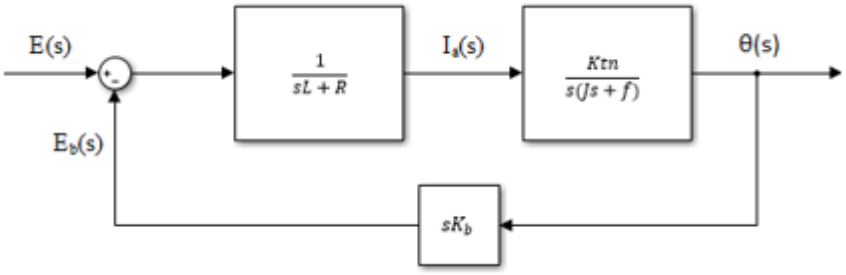
$$(L_a s + R_a)I_a(s) = E(s) - E_b(s) \qquad (24)$$

$$(Js^2 + f_o s)\theta(s) = T_m(s) = K_{tm} I_a(s) \qquad (25)$$

Maka didapatkan fungsi alih sistem servo dengan menggabungkan persamaan (23), (24), dan (25):

$$G(s) = \frac{\theta(s)}{E(s)} = \frac{K_{tm}}{s[(R_a + sL_a)(Js + f_o) + K_T K_b]} \qquad (26)$$

Dari persamaan (26) dapat dibentuk diagram blok seperti:



Gambar 3.12 Diagram blok servo

3.3.3.2 Fungsi alih servo

Untuk menentukan fungsi alih dari servo, persamaan (26) ditulis menjadi:

$$\frac{\theta(s)}{E(s)} = \frac{K_{tm}}{s[La\ Js^2 + (La\ f + Ra\ J)s + Ra\ f + K_{tm}\ Kb]} \qquad (27)$$

Dengan memasukkan nilai pada parameter – parameter tersebut, maka didapatkan nilai berupa:

Ra = *Armature Resistor* = 2,5 **Ω**

L_a =*Armature Inductance* = 0,062 H

I_a = *Armarture Current*

V_a = *Applied Armature Voltage*

τ = Torsi Motor

J = Momen Inersia Motor = 0,00004 $\frac{Kg}{m^2}$

f = Koefisien Gesekan Motor dan Beban = 0,001 $\frac{N}{rad/s}$

K_{tm} = 0,026 $\frac{Nm}{A}$

K_b = 0,02 $\frac{v}{rad/s}$

Maka didapatkan persamaan seperti berikut:

$$\frac{\theta(s)}{E(s)} = \frac{0.026}{0.00000248\,s^3+0.000162\,s^2+0.00302\,s} \qquad (28)$$

Pada persamaan (28) merupakan persamaan servo tanpa adanya nilai dari rasio gir. Pada penelitian ini, pada masing – masing servo memiliki beban yang berbeda. Pada servo X memiliki massa 55 gram dengan panjang 30 cm dari prototipe meriam. Pada servo Y memiliki massa 52 gram dengan jarak 4 cm dari servo X. Dan pada servo Z memiliki massa 52 gram dengan jarak 4 cm dari servo X, 52 gram dengan jarak 6 cm dari servo Y, dan 55 gram dengan jarak 30 cm dari prototipe meriam. Maka perlu dihitung kembali menggunakan persamaan:

$$\tau = F \times d \qquad (29)$$

Dengan F adalah gaya yang diberikan dan d adalah jarak dari gaya yang diberikan dalam satuan meter. Dengan menggunakan persamaan (29), dihasilkan besar torsi seperti berikut:

$$\tau_{servo\ x} = 0.055 \times 9.8 \times \frac{0.3}{2} = 0.08085\ Nm \quad (30)$$

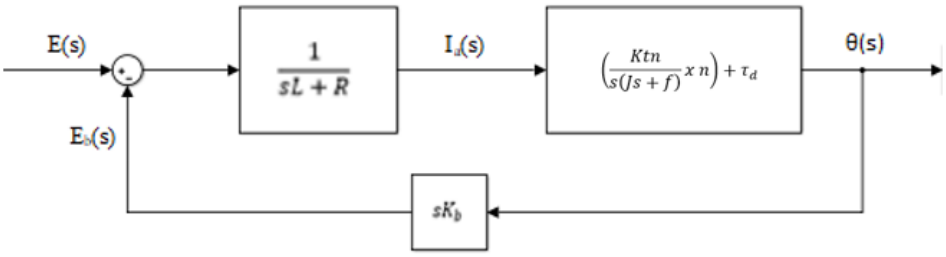
$$\tau_{servo\ y} = 0.052 \times 9.8 \times 0.04 = 0.020384\ Nm \quad (31)$$

$$\tau_{servo\ z} = 0.020384 + 0.08085 + 0.055 \times 9.8 \times 0.06 = 0.131806\ Nm \quad (32)$$

Dengan memperhatikan nilai dari rasio gir dan torsi dari setiap servo, maka persamaan torsi pada servo menjadi:

$$T_{total} = (T_m \times n) + T_{disturbance} \quad (33)$$

Dimana n merupakan rasio gir. Maka diagram blok akan berubah menjadi:



Gambar 3.13 Diagram blok servo dengan rasio gir dan *torque disturbance*

Dengan menggunakan diagram blok pada Gambar 3.13 akan didapatkan fungsi alih untuk servo X, Y, dan Z:

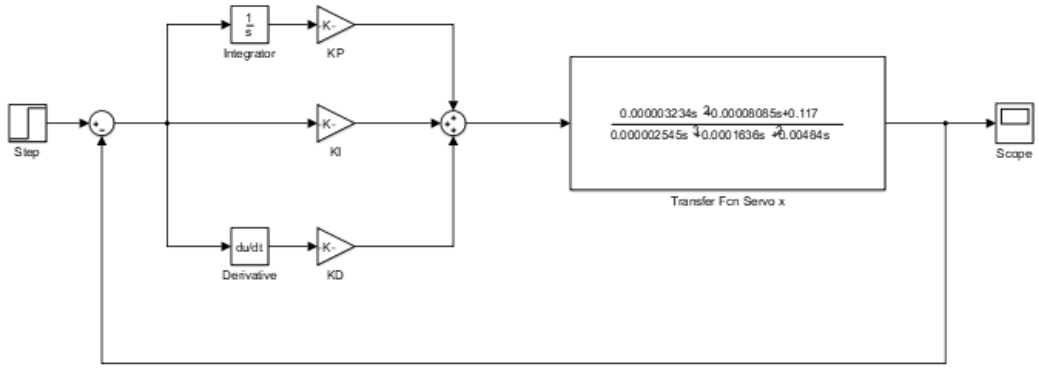
$$TF_{servo\ x} = \frac{0.000003234\ s^2 + 0.00008085\ s + 0.117}{0.000002545\ s^3 + 0.0001636\ s^2 + 0.00484\ s} \quad (34)$$

$$TF_{servo\ y} = \frac{0.0000008154\ s^2 + 0.00002038\ s + 0.117}{0.000002486\ s^3 + 0.0001624\ s^2 + 0.00484\ s} \quad (35)$$

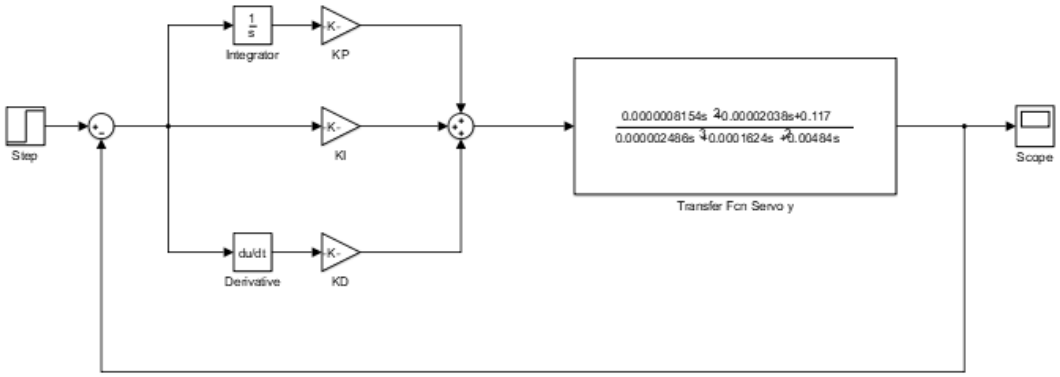
$$TF_{servo\ z} = \frac{0.000005272\ s^2 + 0.0001318\ s + 0.117}{0.000002585\ s^3 + 0.0001646\ s^2 + 0.00484\ s} \quad (36)$$

3.3.3.3 Pemodelan sistem kendali PID

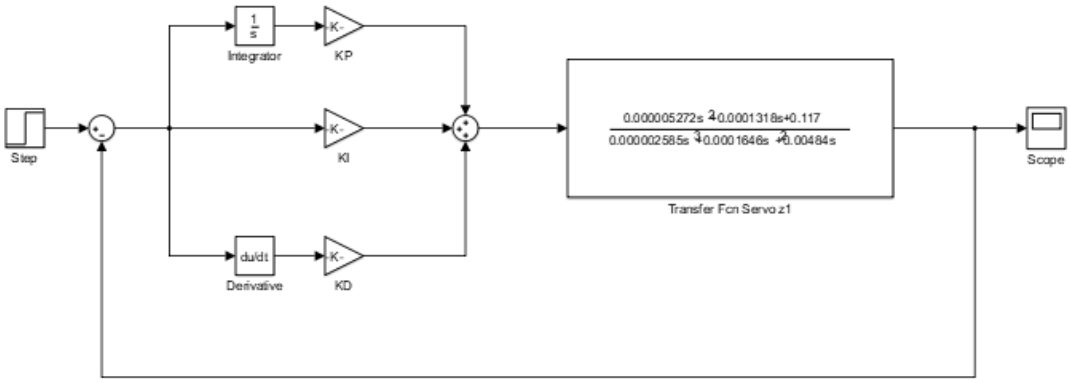
Dengan memanfaatkan program Simulink pada aplikasi matlab, maka akan didapatkan blok diagram untuk sistem prototipe tank seperti pada Gambar 3.14, 3.15, dan 3.16.



Gambar 3.14 Diagram blok servo X pada Simulink

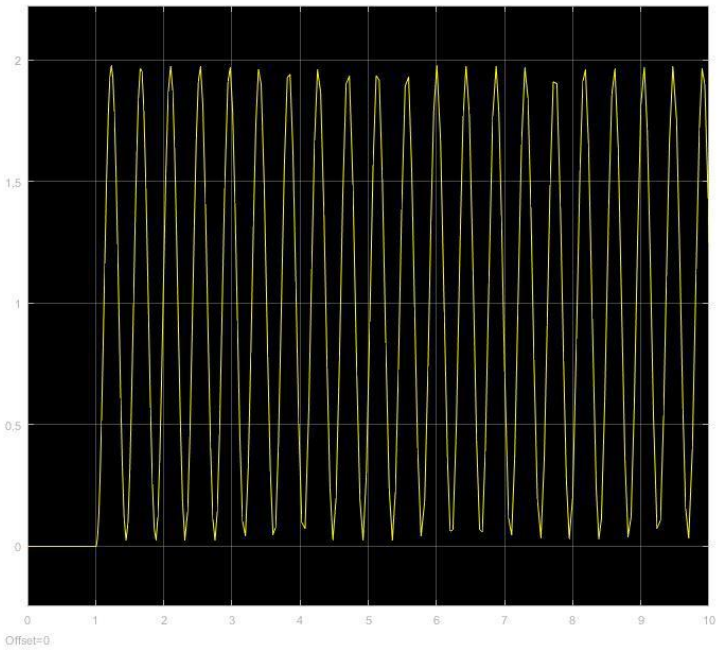


Gambar 3.15 Diagram blok servo Y pada Simulink



Gambar 3.16 Diagram blok servo Z pada Simulink

Dengan menggunakan Metode kedua Ziegle Nichols, nilai Ku (*ultimate gain*) dapat dicari dengan membuat nilai grafik *output* menjadi osislasi. Pertama, nilai Ki dan Kd diatur menjadi 0 terlebih dahulu, kemudian nilai Kp dinaikkan sampai *output* membentuk osilasi.Grafik osilasi sistem dapat dilihat pada Gambar 3.17.



Gambar 3.17 Grafik osilasi

Untuk mendapatkan grafik osilasi, pada servo X dibutuhkan nilai Kp sebesar 2,9538, pada servo Y dibutuhkan nilai Kp sebesar

2,77233 dan pada servo Z dibutuhkan nilai Kp sebesar 3,148799. Nilai Kp yang didapatkan kemudian digunakan sebagai nilai Ku. Untuk mencari nilai Tu (*ultimate periode*) dapat dicari dengan menghitung lebar jarak antara satu gelombang dengan gelombang yang lain. Pada servo X nilai Tu didapatkan sebesar 0,137, pada servo Y nilai Tu didapatkan sebesar 0,137, dan pada servo Z nilai Tu didapatkan sebesar 0,135. Dengan menggunakan Tabel Ziegler Nichols tipe 2 (ditunjukan pada Tabel 3.1), maka didapatkan nilai Kp, Ki, dan Kd untuk masing – masing servo yang ditunjukan pada Tabel 3.2 sampai 3.4.

Tabel 3.1 Zigler Nichols Tipe 2

Jenis pengendali	Kp	Ti	Td	Ki	Kd
P	0.5Ku	-	-	-	-
PI	0.45Ku	Tu/1,2	-	0,52Ku/Tu	-
PD	0.8Ku	-	Tu/8	-	KuTu/10
Classic PID	0.6Ku	Tu/2	Tu/8	1,2Ku/Tu	3KuTu/40
Pessen Integral Rule	7Ku/10	2Tu/5	3Tu/20	1,75Ku/Tu	21KuTu/200
Some Overshoot	Ku/3	Tu/2	Tu/3	0,667Ku/Tu	KuTu/9

No Overshoot	Ku/5	Tu/2	Tu/3	(2/5)Ku/Tu	KuTu/15

Tabel 3.2 Tabel Ziegler Nichols tipe 2 Servo X

KU	TU	Type	KP	KI	KD
2.953800	0.137000	Classical PID	1.772280	25.872701	0.030350
		P	1.476900	-	-
		PI	1.329210	11.642715	-
		PD	2.363040	-	0.040467
		Some Overshoot	0.984600	14.373723	0.044963
		No Overshoot	0.590760	8.624234	0.026978

Tabel 3.3 Tabel Ziegler Nichols tipe 2 Servo Y

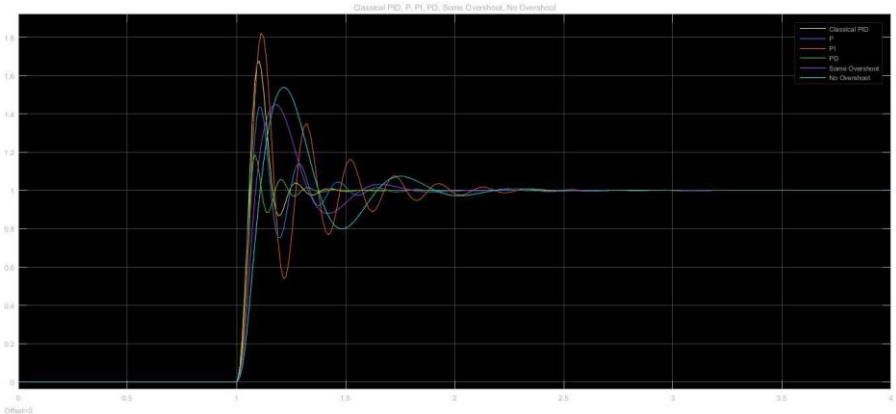
KU	TU	Type	KP	KI	KD
2.772330	0.137000	Classical PID	1.663398	24.283182	0.028486
		P	1.386165	-	-
		PI	1.247549	10.927432	-
		PD	2.217864	-	0.037981
		Some Overshoot	0.924110	13.490657	0.042201

		No Overshoot	0.554466	8.094394	0.025321

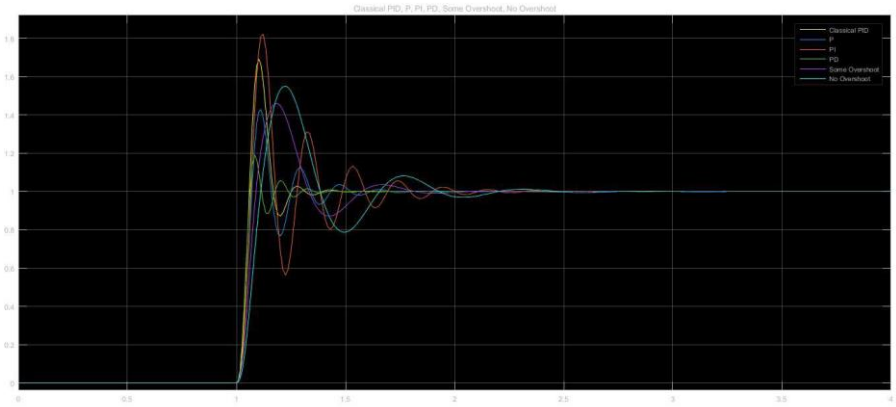
Tabel 3.4 Tabel Ziegler Nichols tipe 2 Servo Z

KU	TU	Type	KP	KI	KD
3.148799	0.135000	Classical PID	1.889279	27.989324	0.031882
		P	1.574400	-	-
		PI	1.416960	12.595196	-
		PD	2.519039	-	0.042509
		Some Overshoot	1.049600	15.549625	0.047232
		No Overshoot	0.629760	9.329775	0.028339

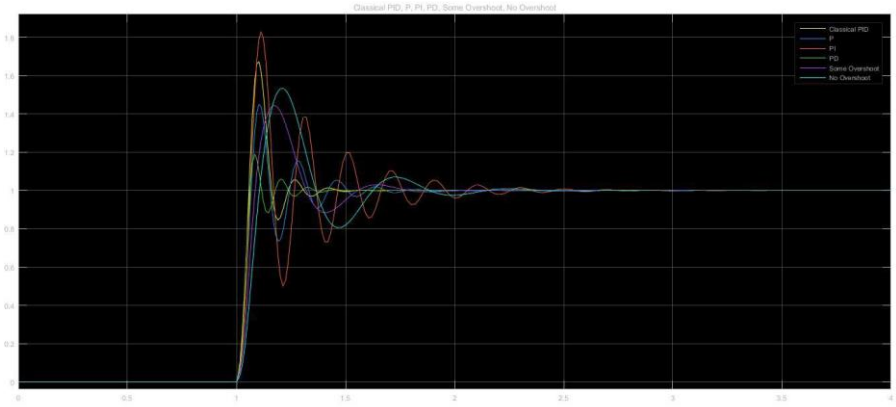
Setelah semua nilai didapatkan pada Tabel 3.2 sampai Tabel 3.4, maka didapatkan hasil perbandingan bentuk grafik PID yang ditunjukan pada Gambar 3.18 sampai Gambar 3.20.



Gambar 3.18 Grafik PID servo X



Gambar 3.19 Grafik PID servo Y



Gambar 3.20 Grafik PID servo Z

3.4 Perancangan *Sofware*

Perancangan *software* dari sistem ini terbagi menjadi dua, sistem yang berada pada Raspberry Pi dan Arduino Mega. Pada Raspberry Pi, pemrograman sistem menggunakan bahasa python dan Visual Studio Code. Pada Arduino Mega, pemrograman sistem menggunakan bahasa C dan Arduino IDE.

3.4.1 Pemrograman Sistem Raspberry Pi

Pada Raspberry Pi, sistem dimulai dengan Raspberry Pi mengambil *library* numpy, imutils.video, cv2, time, dan serial. Numpy digunakan karena memiliki modul *reshape* yang digunakan untuk mengambil bentuk dari Gambar objek yang akan dipasang pada *frame* kamera sebagai display.

Dari imutils.video, mengambil *library* WebcamVideoStream yang berfungsi sebagai pengambilan *frame* pada kamera dengan metode *multi-threading* dimana artinya adalah kemampuan sebuah program untuk melakukan lebih dari satu pekerjaan sekaligus. Keuntungan dari *multi-threading* adalah sifat respons yang interaktif dan real-time. *multi-threading* disini digunakan agar kamera dapat tetap mengambil *frame* tanpa terhenti dari proses pengolahan *frame* sebelumnya. *Library* cv2 digunakan karena metode ORB berada di *library* ini. *Library* time digunakan untuk memberikan *delay* pada program. *Library* serial digunakan untuk komunikasi serial antara Raspberry Pi dengan Arduino Mega. Tahap ini dapat dilihat pada Gambar 3.21.

```
 9   import numpy as np
10   from imutils.video import WebcamVideoStream
11   import cv2
12   import time
13   import serial
```

Gambar 3.21 tahap pengambilan *library*

Selanjutnya, program akan mengatur tinggi dan lebar yang akan ditangkap oleh kamera, setelah itu kamera dinyalakan selama dua detik yang bertujuan pemanasan kamera. Selanjutnya, program membuat modul ORB dengan maksimal pengambilan 1000 *keypoints* dengan *scaling* 1,2. Tahap ini ditunjukkan pada Gambar 3.22.

```
#set the resolution
width =  1280 #640 1280
height =  720 #480 720

#set the camera and warming up
cap = WebcamVideoStream(src=0,resolution=(width,height)).start()
time.sleep(2.0)

# Initiate ORB detector with max 1000 features and scaling 1.2
orb = cv2.ORB_create(1000, 1.2)
```

Gambar 3.22 tahap pengaturan kamera

Tahap berikutnya, program mengambil Gambar objek yang menjadi target sesuai dengan *directory* menyesuaikan pada Gambar tersebut, lalu Gambar itu diubah ke warna hitam putih karena metode ORB menggunakan intensitas cahaya untuk menentukan *keypoints* dari Gambar tersebut. Lalu dari Gambar tersebut dicari *keypoints* dan dibuat *descriptor*-nya. Setelah itu, program mengambil bentuk dari Gambar tersebut untuk dipasang di *frame*. Setelahnya, program memanggil modul BFMatcher atau *brute force matcher* untuk melihat kecocokkan pada Gambar objek dengan *frame* kamera. Tahap ini dapat dilihat pada Gambar 3.23.

```
#input the reference image
input_image = cv2.imread('raspberry pi.jpg')
input_image = cv2.cvtColor(input_image, cv2.COLOR_BGR2GRAY)
#search the keypoint of input image
kp1, des1 = orb.detectAndCompute(input_image,None)
#search the height and Width of the input image
#to make the boundary on display
h,w = input_image.shape
pts = np.float32([ [0,0],[0,h-1],[w-1,h-1],[w-1,0] ]).reshape(-1,1,2)

# create BFMatcher object
bf  = cv2.BFMatcher(cv2.NORM_HAMMING, crossCheck=False)
```

Gambar 3.23 tahap pemrosesan Gambar objek

Berikutnya masuk ke tahap pengaturan komunikasi serial dengan Arduino Mega dan parameter yang digunakan. Pada tahap pengaturan

komunikasi serial, harus ditentukan terlebih dahulu alamat *peripheral* yang dituju, pengaturan *baud rate*, dan *timeout*, pada program ini alamat *peripheral* yang dipakai menyesuaikan dengan alamat Arduino yang terpasang, *baud rate* 9600 bps, dan *timeout* sebesar 0,05 detik. *Baud rate* disini harus sama dengan *baud rate* dari Arduino Mega. Setelahnya, kita harus membersihkan semua data yang tidak lengkap atau berada pada jalur komunikasi serial tersebut, ini dilakukan agar tidak ada data yang cacat atau tertinggal pada pemakaian sebelumnya, lalu diberikan *delay* selama tiga detik yang berfungsi untuk menjalankan program *setup* pada Arduino Mega. Selanjutnya masuk ketahap penyiapan parameter, terdapat dua parameter yang dipakai, pertama menentukan jumlah minimal kecocokkan antara Gambar dengan *frame* yang diambil. Kedua mengatur pergerakan servo X dan Y minimum serta maksimal yang sesuai dengan tangkapan kamera. Perhitungan penyesuaian gerak servo dengan tangkapan kamera telah dibahas pada perancangan *hardware*. Tahap ini dapat dilihat pada Gambar 3.24.

```
ser = serial.Serial('COM3', 9600, timeout=0.05)

# Get rid of garbage/incomplete data

ser.flush()
time.sleep(3)

#variable
#the minimum matching keypoint to detect the object
MIN_MATCH_COUNT = 12

#degree motor
min_X_degree = 55   #min camera X angle 54.79
max_X_degree = 125  #max camera X angle 125.21
min_Y_degree = 70   #min camera Y angle 68.35
max_Y_degree = 110  #max camera Y angle 111.65
```

Gambar 3.24 tahap pengaturan komunikasi serial dan parameter

Dengan begitu proses inisialisasi program telah selesai, selanjutnya masuk ke dalam program utama (*main program*). Pada program utama, pertama – tama mengambil satu *frame* pada kamera, lalu *frame* tersebut diproses pada modul LIVE_CAM_ORB yang telah dibuat. Setelah *frame* tersebut diproses, *frame* akan ditampilkan pada *display* yang terhubung. Program ini akan terus menampilkan *frame* sampai *user* menekan tombol enter. Setelah tombol enter ditekan, maka program akan berhenti. Isi program utama dapat dilihat pada Gambar 3.25.

```
while(True):
    frame = cap.read()
    fps = FPS().start()

    frame = LIVE_CAM_ORB(frame)

    cv2.imshow("the actual frame", frame)
    #for debugging
    #receive serial print from arduino for checking value that rasp send

    #receive_string = ser.readline().decode('utf-8', 'replace').rstrip()
    #print(receive_string)

    #press enter to exit
    if cv2.waitKey(1) == 13:
        break
    #time.sleep(0.05)


cap.stop()
cv2.destroyAllWindows()
```

Gambar 3.25 program utama

Isi dari modul LIVE_CAM_ORB adalah, dengan mengambil *frame* yang telah ditangkap oleh kamera, kemudian *frame* tersebut diubah ke Gambar hitam putih lalu dicari *keypoints* dan dibuat *descriptor*-nya sama seperti yang dilakukan oleh Gambar objek. Kemudian kedua *descriptor* Gambar tersebut (*frame* kamera dan Gambar objek) dicocokkan menggunakan modul *brute force* yang telah disiapkan. Lalu dengan menggunakan tes rasio, semua hasil kecocokkan tersebut dipilah agar hanya kecocokkan yang bagus yang tersimpan. Selanjutnya, jika jumlah kecocokkan yang bagus melebihi jumlah minimum kecocokkan yang telah ditetapkan maka akan masuk ke tahap selanjutnya, jika jumlah kecocokkan kurang dari jumlah minimum yang telah ditetapkan, maka program akan langsung keluar dari modul dan mengambil *frame* kembali. Tahap selanjutnya adalah mengambil bentuk dari Gambar objek dan dipasangkan

ke Gambar *frame* untuk membuat batasan atau *boundary* mengelilingi objek yang menjadi target pada Gambar *frame*. Disini terdapat parameter agar ketika isi M (dapat dilihat pada Gambar 3.10 baris 83) kosong tidak menyebabkan program menjadi *error*. Setelah itu, program akan menghitung titik tengah objek pada *frame*. Perhitungan mencari titik tengah menggunakan rumus *centroid* yang ditunjukkan pada tinjauan pustaka persamaan (14) menggunakan titik – titik yang didapatkan saat membuat batasan pada tahap sebelumnya. Selanjutnya, nilai titik tengah tersebut dikirim ke modul motor_degree untuk dikonversi ke dalam nilai yang dapat diterima oleh servo karena nilai titik tengah masih dalam area (0,0) sampai (640,480) dimana servo motor yang dipakai memiliki area (5,75) sampai (175,120). Tahap ini ditampilkan pada Gambar 3.26.

```
65  def LIVE_CAM_ORB(live_cam):
66      img = live_cam
67
68      #change the camera image and input image to black and white
69      img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
70
71      #search the keypoint of live cam
72      kp2, des2 = orb.detectAndCompute(img,None)
73
74      # Match descriptors.
75      matches = bf.knnMatch(des1,des2,k=2)
76
77      # Apply ratio test
78      good = []
79      for m,n in matches:
80          if m.distance < 0.75*n.distance:
81              good.append(m)
82
83      #take the shape of reference image to make boundary to the display
84      if len(good)>MIN_MATCH_COUNT:
85          src_pts = np.float32([ kp1[m.queryIdx].pt for m in good ]).reshape(-1,1,2)
86          dst_pts = np.float32([ kp2[m.trainIdx].pt for m in good ]).reshape(-1,1,2)
87          M, mask = cv2.findHomography(src_pts, dst_pts, cv2.RANSAC,5.0)
88          #(for debugging)
89          #matchesMask = mask.ravel().tolist()
90          #print( "M = {}".format(M))
91
92          if M is not None:
93              dst = cv2.perspectiveTransform(pts,M)
94              (tl, tr, br, bl) = dst
95              midpoint = (tl+bl+br+tr)/4
96              midpoint = (tl[0]+bl[0]+br[0]+tr[0])/4
97              #midpoint = np.rint(midpoint) //round the integer
98              midpoint = midpoint.astype(int)
99              live_cam = cv2.polylines(live_cam,[np.int32(dst)],True,255,3, cv2.LINE_AA)
100             live_cam = cv2.circle(live_cam,(midpoint[0],midpoint[1]),5,255,-1)
101
102             #for img3 (debugging with black and white)
103             #img2 = cv2.polylines(img2,[np.int32(dst)],True,255,3, cv2.LINE_AA)
104
105             #searching degree for motor value
106             motor_degree(midpoint)
107
108     return live_cam
```

Gambar 3.26 modul LIVE_CAM_ORB

Pada modul motor_degree, program mengkonversi nilai titik tengah yang diterima dengan menggunakan perhitungan sederhana. Perhitungan untuk nilai servo X seperti berikut:

$$servo\ x = \left(\frac{titik\ tengah\ X}{lebar\ maksimum\ frame} \times (maksimum\ servo\ X - minimum\ servo\ X) + minimum\ servo\ X\right) \quad (37)$$

Jangkauan hasil yang didapat dari perhitungan servo X adalah -90 sampai 90. Pada perhitungan servo Y sedikit berbeda dengan servo X dikarenakan nilai titik tengah yang didapat untuk nilai Y terbalik dimana semakin tinggi target, maka nilai Y akan semakin kecil begitu juga sebaliknya. Maka perhitungan servo Y seperti berikut:

$$servo\ y = \left(maksimum\ servo\ Y - \left(\frac{titik\ tengah\ y}{tinggi\ maksimum\ frame} \times \right. \right.$$

$$\left. \left. (maksimum\ servo\ Y - minimum\ servo\ Y) \right) \right) \qquad (38)$$

Setelah kedua nilai X dan Y dikonversi, nilai tersebut dikirimkan ke Arduino Mega menggunakan komunikasi serial jika telah melebihi parameter dimana nilai X harus berada di antara nilai minimum servo X dan nilai maksimum servo X begitu juga dengan nilai Y harus berada di antara nilai minimum servo Y dan nilai maksimum servo Y. Selanjutnya, nilai tersebut akan dikirimkan melalui komunikasi serial menggunakan modul send_to_arduino. Pada modul send_to_arduino dengan *delay* 0,2 detik yang berfungsi agar satu set data yang dikirim dapat dibaca terlebih dahulu oleh Arduino Mega dan tidak *overflow*, sebelum nilai tersebut dikirimkan, nilai tersebut diubah menjadi tipe data *string*. Tahap ini dapat dilihat pada Gambar 3.27.

```
def motor_degree(midpoint):
    #get X degree
    X_degree = int((midpoint[0]/width*(max_X_degree-min_X_degree))+min_X_degree)

    #get Y degree
    #because you got a flipped value
    #(the higher you get, the smaller the midpoint value)
    #the formula is different from getting X
    Y_degree = int(max_Y_degree-(midpoint[1]/height*(max_Y_degree-min_Y_degree)))

    #because the servo only can move until 75 degree
    if (Y_degree < 75):
        Y_degree = 75

    #for debugging
    #print( "X,Y = ({},{})".format(midpoint[0],midpoint[1]))
    #print( "X,Y = ({},{})".format(X_degree,Y_degree))

    #only send data if the value between the parameter
    if(X_degree<max_X_degree and X_degree>min_X_degree and Y_degree<max_Y_degree and Y_degree>min_Y_degree):
        #sending the value to arduino
        send_to_arduino(X_degree,Y_degree)

def send_to_arduino(x,y):
    angle_value_list = [str(x),str(y)]
    send_string = ','.join(angle_value_list)
    send_string += "\n"

    #send_string ="["
    #send_string += ','.join(angle_value_list)
    #send_string += "]\n"

    #debugging
    #to check the value in send_string
    print( "send string = {}".format(send_string))

    # Send the string. Make sure you encode it before you send it to the Arduino.

    #ser.write(send_string.encode('utf-8'))
    #time.sleep(0.2)
```

Gambar 3.27 modul motor_degree dan send_to_arduino

Selanjutnya program akan mengulangi proses ini dari program utama sampai *user* menekan tombol *enter*.

3.4.2 Perancangan Sistem Arduino Mega

Pada Arduino Mega terdapat sistem kendali PID yang mengatur *output* servo dengan menggunakan *feedback* dari *gyroscope* yang digunakan sebagai penyeimbang antara badan tank dan meriam tank. Tahap pertama sama seperti perancangan pada sistem Raspberry Pi, yaitu pengambilan *library* yang dibutuhkan. *Library* yang dibutuhkan adalah Servo.h dan Wire.h, dimana *library* Servo.h digunakan untuk mengendalikan servo dan *library* Wire.h digunakan untuk Arduino Mega berkomunikasi dengan

sensor *gyroscope* MPU-6500 menggunakan protokol I2C. Selanjutnya memasukkan alamat untuk kedua *gyroscope*. Alamat yang dipakai adalah 0x68 untuk *gyroscope* yang berada di badan tank dan 0x69 untuk *gyroscope* yang berada pada ujung meriam. Lalu menyiapkan parameter yang akan digunakan pada *gyroscope*, PID, *setpoint* PID, dan nilai yang diterima dari Raspberry Pi. Untuk nilai Kp, Ki, dan Kd yang akan digunakan pada servo X, Y, dan Z akan dibahas pada perancangan *hardware*. Tahap ini ditunjukkan pada Gambar 3.28 dan Gambar 3.29.

arduino_code_PID

```
#include <Servo.h>
Servo servo_x;
Servo servo_y;
Servo servo_z;
int j = 0;
float correct_x;
float correct_y;
float correct_z;

// how much serial data we expect before a newline/enter
const unsigned int MAX_INPUT = 10;

//====================================================================

#include <Wire.h>
const int MPU = 0x68; // MPU1 I2C address
float AccX, AccY, AccZ;
float GyroX, GyroY, GyroZ;
float accAngleX, accAngleY, gyroAngleX, gyroAngleY, gyroAngleZ;
float roll, pitch, yaw;
float AccErrorX, AccErrorY, GyroErrorX, GyroErrorY, GyroErrorZ;
float elapsedTime, currentTime, previousTime;
int c = 0;

//---------------------------------------------------------------
const int MPU2 = 0x69; // MPU2 I2C address (AD0 == HIGH)
float AccX2, AccY2, AccZ2;
float GyroX2, GyroY2, GyroZ2;
float accAngleX2, accAngleY2, gyroAngleX2, gyroAngleY2, gyroAngleZ2;
float roll2, pitch2, yaw2;
float AccErrorX2, AccErrorY2, GyroErrorX2, GyroErrorY2, GyroErrorZ2;
float elapsedTime2, currentTime2, previousTime2;
int c2 = 0;
```

Gambar 3.28 tahap pengambilan *library* dan pengaturan *gyroscope*

```
//------------------------------------------------------------

float px, ix, dx, pidx;
float errorx, prev_errorx = 0;

float py, iy, dy, pidy;
float errory, prev_errory = 0;

float pz, iz, dz, pidz;
float errorz, prev_errorz = 0;

float spX = 90;

float spY = 90;

float spZ = 90;

int raspiX = 0;
int raspiY = 0;

float kpx = 1.063040;
float kix = 0;
float kdx = 0.010467;

float kpy = 1.017864;
float kiy = 0;
float kdy = 0.017981;

float kpz = 1.019039;
float kiz = 0;
float kdz = 0.012509;
```

Gambar 3.29 parameter PID dan nilai dari Raspberry Pi

Selanjutnya masuk ke dalam modul setup. Pertama, mengatur besaran *baud rate* yang akan dipakai, karena pada Raspberry Pi menggunakan 115200, maka pada Arduino Mega juga menggunakan 115200. Setelah itu, memasang nomor *pinout* yang sesuai dengan servo dan mengatur pada titik 90 derajat. Sebelum Arduino Mega dapat berkomunikasi dengan *gyroscope*, *gyroscope* harus di-*reset* terlebih dahulu. Berdasarkan *datasheet* MPU-6500, untuk me-*reset*-nya dengan memberi nilai 0x01 ke register 6B. Lalu, mengatur sensitifitas pada *accelerometer* dan *gyroscope* pada MPU6500. Tahap ini dapat dilihat pada Gambar 3.30.

```
void setup() {
  Serial.begin(115200);

  servo_x.attach(8);                  // Set pin 8 untuk servo x
  servo_y.attach(9);                  // Set pin 9 untuk servo y
  servo_z.attach(10);                 // Set pin 10 untuk servo z

  servo_x.write(90);                  // Set posisi servo x ke 90 derajat
  servo_y.write(90);                  // Set posisi servo y ke 90 derajat
  servo_z.write(90);                  // Set posisi servo z ke 90 derajat
  delay(1000);                        // Memberikan delay agar servo sampai ke posisi set point

  Wire.begin();                      // Inisialisasi komunikasi
  Wire.beginTransmission(MPU);       // Memulai komunikasi dengan MPU1
  Wire.write(0x6B);                  // Talk to the register 6B
  Wire.write(0x01);                  // Make reset - memasukan nilai 0 ke register 6B
  Wire.endTransmission(true);        // Mengakhiri transmisi

  // Mengkonfigurasi sensitivitas Accelerometer - Full Scale Range (default +/- 2g)
  Wire.beginTransmission(MPU);
  Wire.write(0x1C);                  // Komunikasi dengan register ACCEL_CONFIG (1C hex)
  Wire.write(0x10);                  // Men-Set nilai register bits menjadi 00010000 (+/- 8g full scale range)
  Wire.endTransmission(true);

  // Mengkonfigurasi sensitivitas Gyro - Full Scale Range (default +/- 250deg/s)
  Wire.beginTransmission(MPU);
  Wire.write(0x1B);                   // Komunikasi dengan register GYRO_CONFIG (1B hex)
  Wire.write(0x10);                   // Men-Set nilai register bits menjadi 00010000 (1000deg/s full scale)
  Wire.endTransmission(true);
  delay(20);
```

Gambar 3.30 modul setup Arduino Mega

Sekarang masuk ke modul *loop*, program akan membaca *input* dari komunikasi serial, jika terdapat *input* pada komunikasi serial, nilai *setpoint* servo X dan servo Z akan terganti. Jika tidak ada *input* maka program akan langsung membaca data dari *accelerometer* dan *gyroscope*. Data yang didapat dari *accelerometer* dan *gyroscope* dikalkulasi sehingga mendapatkan nilai *roll*, *pitch*, dan *yaw*. Kemudian nilai *roll*, *pitch*, dan *yaw* di-*mapping* ke dalam variabel integer servo0Value1 untuk mendapatkan nilai X posisi dari badan tank dan posisi dari ujung meriam. Untuk mendapatkan nilai Y dan Z, menggunakan variabel servo1Value1 dan servo2Value1. Nilai tersebut digunakan untuk mencari *error* pada PID dengan parameter Kp, Ki, dan Kd yang sudah ditetapkan. Setelah mendapatkan semua nilai PID pada servo X, Y, dan Z, nilai PID tersebut

dikirimkan ke servo yang sesuai. Tahap ini dapat dilihat pada Gambar 3.31sampai Gambar 3.34.

```
void loop() {
  // === Read serial input if available ===//
  if (Serial.available() > 0) {
    processIncomingByte (Serial.read ());
    spX = raspiY;
    spZ = raspiX;
    }
  delay(50);
  // === Read acceleromter data === //
  Wire.beginTransmission(MPU);
  Wire.write(0x3B); // Start with register 0x3B (ACCEL_XOUT_H)
  Wire.endTransmission(false);
  Wire.requestFrom(MPU, 6, true); // Read 6 registers total, tiap nilai sumbu disimpan tiap 2 registers

  // Untuk nilai akselero +-2g, nilainya harus dibagi dengan 16384, sesuai dengan datasheet
  AccX = (Wire.read() << 8 | Wire.read()) / 16384.0; // X-axis value
  AccY = (Wire.read() << 8 | Wire.read()) / 16384.0; // Y-axis value
  AccZ = (Wire.read() << 8 | Wire.read()) / 16384.0; // Z-axis value

  // Menghitung data Roll dan Pitch dari accelerometer
  accAngleX = (atan(AccY / sqrt(pow(AccX, 2) + pow(AccZ, 2))) * 180 / PI) ; // Masih dalam bentuk radian, dikali 180/pi untuk menjadi derajat
  accAngleY = (atan(-1 * AccX / sqrt(pow(AccY, 2) + pow(AccZ, 2))) * 180 / PI); // Masih dalam bentuk radian, dikali 180/pi untuk menjadi deraja

  // === Read gyroscope data === //
  previousTime = currentTime;        // Previous time merupakan waktu yg disimpan sebelum waktu aktual dibaca
  currentTime = millis();            // Current time merupakan waktu aktua;
  elapsedTime = (currentTime - previousTime) / 1000; // Dibagi 1000 agar nilainya menjadi seconds

  Wire.beginTransmission(MPU);
  Wire.write(0x43); // Gyro data first register address 0x43 (GYRO_XOUT_H)
  Wire.endTransmission(false);
  Wire.requestFrom(MPU, 6, true); // Read 6 registers total, tiap nilai sumbu disimpan tiap 2 registers
  GyroX = (Wire.read() << 8 | Wire.read()) / 131.0; // Untuk nilai gyro 250deg/s, nilainya harus dibagi 131.0, sesuai dengan datasheet
  GyroY = (Wire.read() << 8 | Wire.read()) / 131.0;
  GyroZ = (Wire.read() << 8 | Wire.read()) / 131.0;

  // Nilai Output dikoreksi dengan dijumlahkan dengan nilai yg didapat dari void calculate_IMU_error()
  GyroX = GyroX - 3.65;          //0.74;
  GyroY = GyroY + 0.05;          //0.66;
  GyroZ = GyroZ - 0.16;          //0.03;
```

Gambar 3.31 modul loop, pengambilan data dari komunikasi serial dan *gyroscope*, dan perhitungan *gyroscope*

```
  // Karena hasilnya masih berupa degrees per seconds (deg/s), Jadi harus dikalikan dengan satuan waktu (seconds) untuk mendapatkan nilai sudut dalam degree
  gyroAngleX = gyroAngleX + GyroX * elapsedTime; // deg/s * s = deg
  gyroAngleY = gyroAngleY + GyroY * elapsedTime;
  yaw =  yaw + GyroZ * elapsedTime;
  // Mengkombinasikan nilai acceleromter dan gyro untuk mendapatkan posisi sudut yg akurat
  roll = 0.96 * gyroAngleX + 0.04 * accAngleX;
  pitch = 0.96 * gyroAngleY + 0.04 * accAngleY;
```

Gambar 3.32 perhitungan *roll*, *pitch*, dan *yaw*

```
426  //---------------------------------------------------------------
427  //                              PID
428  //---------------------------------------------------------------
429
430  errorx = spX - servo0Value1;
431  px = kpx * errorx;
432  ix = kix * (prev_errorx + errorx);
433  dx = kdx * (errorx - prev_errorx);
434  pidx = px + ix + dx;
435  prev_errorx = errorx;
436  int pid_x = spX - pidx;
437  if (pid_x < 70) {
438    pid_x = 70;
439  }
440  else if (pid_x > 130) {
441    pid_x = 130;
442  }
```

Gambar 3.33 perhitungan PID untuk servo X

```
484   servo_x.write(pid_x);
485   servo_y.write(pid_y);
486   servo_z.write(pid_z);
```

Gambar 3.34 pengiriman nilai PID ke servo