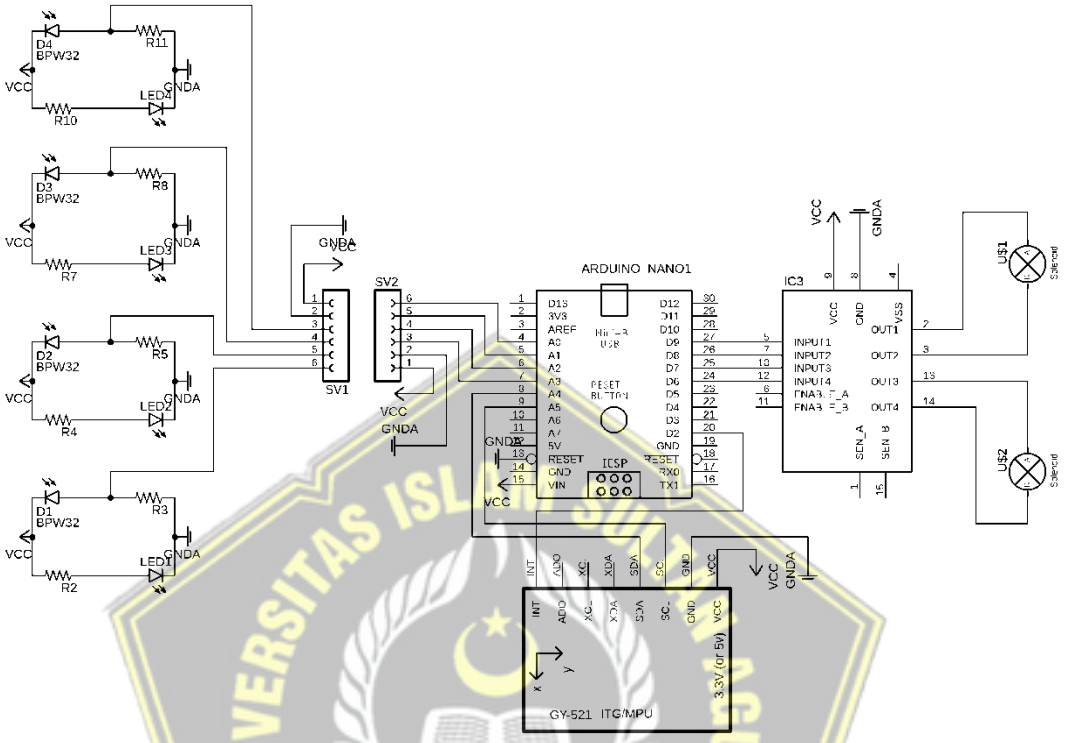
Lampiran

Skema Rangkaian



Desain Robot

Program Robot

```
#include <PID_v1.h>


//#include <PID_v1.h>
#include <LMotorController.h>
#include "I2Cdev.h"


#include "MPU6050_6Axis_MotionApps20.h"


#if I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE
    #include "Wire.h"
#endif
#define LOG_INPUT 0
#define MANUAL_TUNING 0
#define LOG_PID_CONSTANTS 0 //MANUAL_TUNING must be 1
#define MOVE_BACK_FORTH 0
#define MIN_ABS_SPEED 30
//MPU
MPU6050 mpu;
// MPU control/status vars
bool dmpReady = false;  // set true if DMP init was
successful
uint8_t mpuIntStatus;   // holds actual interrupt
status byte from MPU
uint8_t devStatus;      // return status after each
device operation (0 = success, !0 = error)
uint16_t packetSize;    // expected DMP packet size
(default is 42 bytes)
uint16_t fifoCount;     // count of all bytes currently
in FIFO
```

```
uint8_t fifoBuffer[64]; // FIFO storage buffer


// orientation/motion vars
Quaternion q;           // [w, x, y, z]
quaternion container
VectorFloat gravity;    // [x, y, z]            gravity
vector
float ypr[3];           // [yaw, pitch, roll]
yaw/pitch/roll container and gravity vector
//PID
#if MANUAL_TUNING
  double kp , ki, kd;
  double prevKp, prevKi, prevKd;
#endif
double originalSetpoint = 176.29;//-1.10; //176.29;
double setpoint = originalSetpoint;
double movingAngleOffset = 0.3;
double input, output;
int moveState=0; //0 = balance; 1 = back; 2 = forth


#if MANUAL_TUNING
  PID pid(&input, &output, &setpoint, 0, 0, 0, DIRECT);
#else
  PID pid(&input, &output, &setpoint, 40, 270, 1.9,
DIRECT);
#endif


//photodioda
int IR1=10;      //Right sensor
int IR2=11;    //left Sensor
```

```
int a;
int b;
//MOTOR CONTROLLER
int ENA = 3;
int IN1 = 8;
int IN2 = 4;
int IN3 = 7;
int IN4 = 5;
int ENB = 6;
LMotorController motorController(ENA, IN1, IN2, ENB,
IN3, IN4, 0.6, 1);
//timers
long time1Hz = 0;
long time5Hz = 0;
volatile bool mpuInterrupt = false;     // indicates
whether MPU interrupt pin has gone high
void dmpDataReady()
{
    mpuInterrupt = true;
}
void setup()
{
  pinMode(IR1, INPUT);
  pinMode(IR2, INPUT);
    // join I2C bus (I2Cdev library doesn't do this
automatically)
    #if I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE
        Wire.begin();
```

```
        TWBR = 24; // 400kHz I2C clock (200kHz if CPU
is 8MHz)
    #elif I2CDEV_IMPLEMENTATION ==
I2CDEV_BUILTIN_FASTWIRE
        Fastwire::setup(400, true);
    #endif

    // initialize serial communication
    // (115200 chosen because it is required for Teapot
Demo output, but it's
    // really up to you depending on your project)
    Serial.begin(115200);
    while (!Serial); // wait for Leonardo enumeration,
others continue immediately

    // initialize device
    Serial.println(F("Initializing I2C devices..."));
    mpu.initialize();

    // verify connection
    Serial.println(F("Testing device connections..."));
    Serial.println(mpu.testConnection() ? F("MPU6050
connection successful") : F("MPU6050 connection
failed"));

    // load and configure the DMP
    Serial.println(F("Initializing DMP..."));
    devStatus = mpu.dmpInitialize();

    // supply your own gyro offsets here, scaled for
min sensitivity
```

```
    mpu.setXGyroOffset(220);
    mpu.setYGyroOffset(76);
    mpu.setZGyroOffset(-85);
    mpu.setZAccelOffset(1788); // 1688 factory default
for my test chip

    // make sure it worked (returns 0 if so)
    if (devStatus == 0)
    {
        // turn on the DMP, now that it's ready
        Serial.println(F("Enabling DMP..."));
        mpu.setDMPEnabled(true);

        // enable Arduino interrupt detection
        Serial.println(F("Enabling interrupt detection
(Arduino external interrupt 0)..."));
        attachInterrupt(0, dmpDataReady, RISING);
        mpuIntStatus = mpu.getIntStatus();

        // set our DMP Ready flag so the main loop()
function knows it's okay to use it
        Serial.println(F("DMP ready! Waiting for first
interrupt..."));
        dmpReady = true;

        // get expected DMP packet size for later
comparison
        packetSize = mpu.dmpGetFIFOPacketSize();

        //setup PID
```

```
        pid.SetMode(AUTOMATIC);
        pid.SetSampleTime(10);
        pid.SetOutputLimits(-255, 255);
    }
    else
    {
        // ERROR!
        // 1 = initial memory load failed
        // 2 = DMP configuration updates failed
        // (if it's going to break, usually the code
will be 1)
        Serial.print(F("DMP Initialization failed (code
"));
        Serial.print(devStatus);
        Serial.println(F(")"));
    }
 // pinMode(IR1,INPUT);
 // pinMode(IR2,INPUT);
}
void loop()
{
    // if programming failed, don't try to do anything
    if (!dmpReady) return;

    // wait for MPU interrupt or extra packet(s)
available
    while (!mpuInterrupt && fifoCount < packetSize)
    {
```

```
        //no mpu data - performing PID calculations and
output to motors
         Serial.println(setpoint);
         pid.Compute();
         Serial.print (" y : "); Serial.println(input);
//Serial.print(" =>"); Serial.println(output);
         motorController.move(output, MIN_ABS_SPEED);


        if(input>150 && input<200)
        {
          a = digitalRead(IR1);
          b = digitalRead(IR2);

          if(output>0)
          {
            if(a == HIGH && b == HIGH)
            {
              /*digitalWrite(IN1,LOW);
                digitalWrite(IN2,LOW);
                digitalWrite(IN3,LOW);
                digitalWrite(IN4,LOW);
                analogWrite (ENA, 0);
                analogWrite (ENB, 0);*/
                pid.Compute();
                motorController.move(output,
MIN_ABS_SPEED);
            }
            else if(a == LOW && b == LOW)
            {
```

```
    /*digitalWrite(IN1,HIGH);
digitalWrite(IN2,LOW);
digitalWrite(IN3,LOW);
digitalWrite(IN4,HIGH);
analogWrite (ENA, 200);
analogWrite (ENB, 200);*/
 pid.Compute();
      motorController.move(output,
MIN_ABS_SPEED);
  }
  else if(a == LOW && b == HIGH)
  {
    digitalWrite(IN1,HIGH);
digitalWrite(IN2,LOW);
digitalWrite(IN3,LOW);
digitalWrite(IN4,HIGH);
analogWrite (ENA, 200);
analogWrite (ENB, 100);
  }
  else if(a == HIGH && b == LOW)
  {
    digitalWrite(IN1,LOW);
digitalWrite(IN2,HIGH);
digitalWrite(IN3,HIGH);
digitalWrite(IN4,LOW);
analogWrite (ENA, 100);
analogWrite (ENB, 200);
  }
}
```

```
          else if(output == -255)
          {
             pid.Compute();
                 motorController.move(output,
MIN_ABS_SPEED);
          }
        }
        else
        {
           pid.Compute();
                 motorController.move(output,
MIN_ABS_SPEED);
        }

       /* unsigned long currentMillis = millis();

        if (currentMillis - time1Hz >= 1000)
        {
            loopAt1Hz();
            time1Hz = currentMillis;
        }

        if (currentMillis - time5Hz >= 5000)
        {
            loopAt5Hz();
            time5Hz = currentMillis;
        }*/
    }
```

```
    // reset interrupt flag and get INT_STATUS byte
    mpuInterrupt = false;
    mpuIntStatus = mpu.getIntStatus();

    // get current FIFO count
    fifoCount = mpu.getFIFOCount();

    // check for overflow (this should never happen
unless our code is too inefficient)
    if ((mpuIntStatus & 0x10) || fifoCount == 1024)
    {
        // reset so we can continue cleanly
        mpu.resetFIFO();
        Serial.println(F("FIFO overflow!"));

    // otherwise, check for DMP data ready interrupt
(this should happen frequently)
    }
    else if (mpuIntStatus & 0x02)
    {
        // wait for correct available data length,
should be a VERY short wait
        while (fifoCount < packetSize) fifoCount =
mpu.getFIFOCount();

        // read a packet from FIFO
        mpu.getFIFOBytes(fifoBuffer, packetSize);

        // track FIFO count here in case there is > 1
packet available
```

```
        // (this lets us immediately read more without
waiting for an interrupt)
        fifoCount -= packetSize;

        mpu.dmpGetQuaternion(&q, fifoBuffer);
        mpu.dmpGetGravity(&gravity, &q);
        mpu.dmpGetYawPitchRoll(ypr, &q, &gravity);
        #if LOG_INPUT
            Serial.print("ypr\t");
            Serial.print(ypr[0] * 180/M_PI);
            Serial.print("\t");
            Serial.print(ypr[1] * 180/M_PI);
            Serial.print("\t");
            Serial.println(ypr[2] * 180/M_PI);
        #endif
        input = ypr[1] * 180/M_PI + 180;
    }
    //Serial.print("y:");
Serial.println(input);if(Serial.available()){setpoint =
input;}
}

/*void sensor()
{
 if(digitalRead(IR1)==HIGH && digitalRead(IR2)==HIGH)
        {
          digitalWrite(IN1,LOW);
          digitalWrite(IN2,LOW);
          digitalWrite(IN3,LOW);
```

```
          digitalWrite(IN4,LOW);
          analogWrite (ENA, 0);
          analogWrite (ENB, 0);
        }
        if(digitalRead(IR1)==LOW &&
digitalRead(IR2)==LOW)
        {
          digitalWrite(IN1,HIGH);
          digitalWrite(IN2,LOW);
          digitalWrite(IN3,LOW);
          digitalWrite(IN4,HIGH);
          analogWrite (ENA, 200);
          analogWrite (ENB, 200);
        }
        if(digitalRead(IR1)==LOW &&
digitalRead(IR2)==HIGH)
        {
          digitalWrite(IN1,HIGH);
          digitalWrite(IN2,LOW);
          digitalWrite(IN3,LOW);
          digitalWrite(IN4,HIGH);
          analogWrite (ENA, 200);
          analogWrite (ENB, 100);
        }
        if(digitalRead(IR1)==HIGH &&
digitalRead(IR2)==LOW)
        {
          digitalWrite(IN1,LOW);
          digitalWrite(IN2,HIGH);
```

```
            digitalWrite(IN3,HIGH);
            digitalWrite(IN4,LOW);
            analogWrite (ENA, 100);
            analogWrite (ENB, 200);
          }
}*/
/*void loopAt1Hz()
{
#if MANUAL_TUNING
    setPIDTuningValues();
#endif
}


void loopAt5Hz()
{
    #if MOVE_BACK_FORTH
        moveBackForth();
    #endif
}


//move back and forth


void moveBackForth()
{
    moveState++;
```

```
    if (moveState > 2) moveState = 0;

    if (moveState == 0)
      setpoint = originalSetpoint;
    else if (moveState == 1)
      setpoint = originalSetpoint - movingAngleOffset;
    else
      setpoint = originalSetpoint + movingAngleOffset;
}


//PID Tuning (3 potentiometers)

#if MANUAL_TUNING
void setPIDTuningValues()
{
    readPIDTuningValues();

    if (kp != prevKp || ki != prevKi || kd != prevKd)
    {
#if LOG_PID_CONSTANTS
        Serial.print(kp);Serial.print(",
");Serial.print(ki);Serial.print(",
");Serial.println(kd);
#endif

        pid.SetTunings(kp, ki, kd);
        prevKp = kp; prevKi = ki; prevKd = kd;
    }
```

```
}


void readPIDTuningValues()
{
    int potKp = analogRead(A0);
    int potKi = analogRead(A1);
    int potKd = analogRead(A2);

    kp = map(potKp, 0, 1023, 0, 25000) / 100.0; //0 -
250
    ki = map(potKi, 0, 1023, 0, 100000) / 100.0; //0 -
1000
    kd = map(potKd, 0, 1023, 0, 500) / 100.0; //0 - 5
}
#endif*/
```