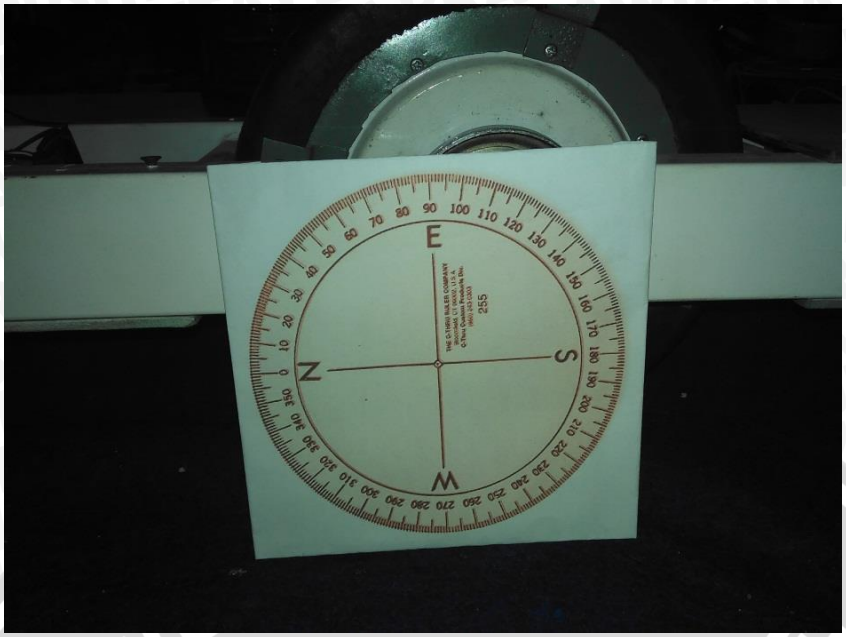
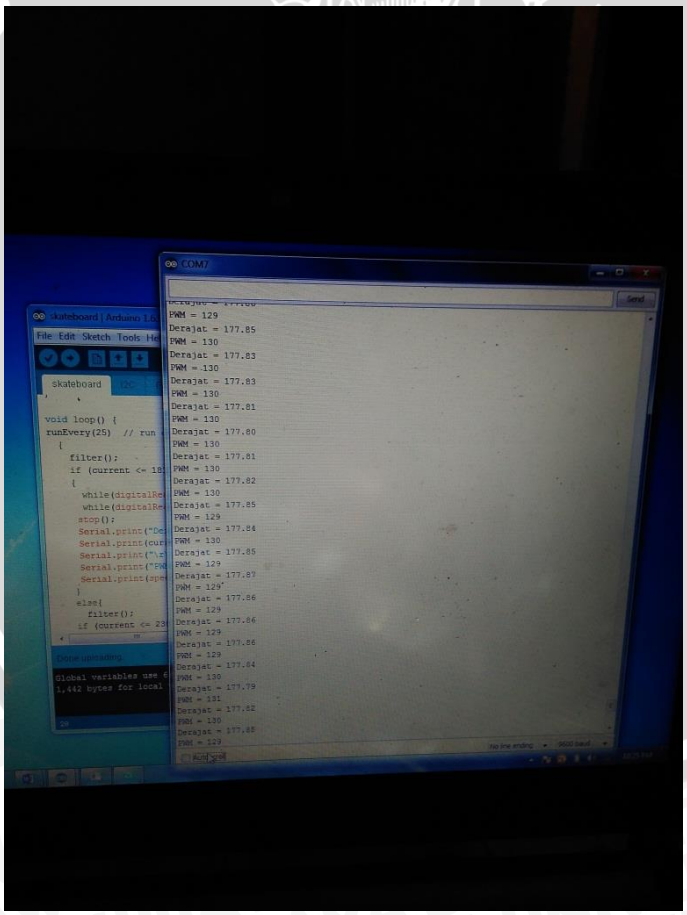
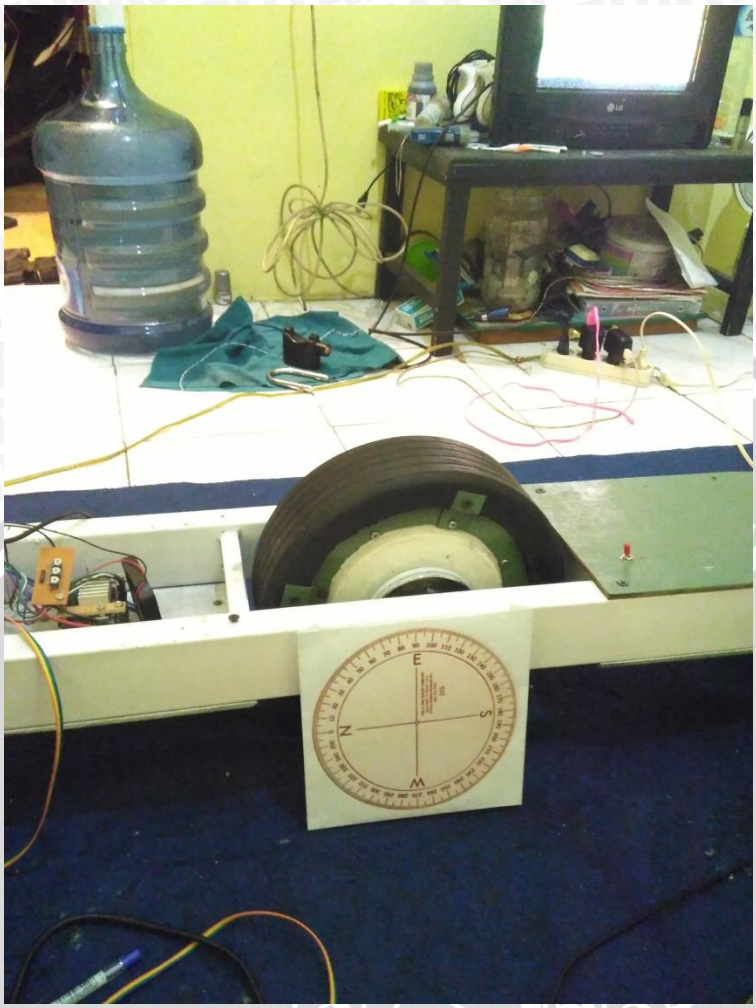
LAMPIRAN 1
Foto Alat



Gambar busur terhadap *Plant skateboard one wheel*



Gambar data sensor MPU6050



Gambar *Plant* tampak samping

Gambar *Plant* tampak atas

UNIVERSITAS BRAWIJAYA

LAMPIRAN 2
Listing Program

Buzzer

```
void bunyi_1(){
 digitalWrite(buzer,HIGH);
 delay(100);
 digitalWrite(buzer,LOW);
 delay(100);
 digitalWrite(buzer,HIGH);
 delay(100);
 digitalWrite(buzer,LOW);
 delay(100);
 digitalWrite(buzer,HIGH);
 delay(100);
 digitalWrite(buzer,LOW);
 delay(100);
 digitalWrite(buzer,HIGH);
 delay(100);
 digitalWrite(buzer,LOW);
 delay(100);
 digitalWrite(buzer,HIGH);
 delay(500);
 digitalWrite(buzer,LOW);
 delay(100);
 digitalWrite(buzer,HIGH);
 delay(100);
 digitalWrite(buzer,LOW);
 delay(100);
 digitalWrite(buzer,HIGH);
 delay(100);
 digitalWrite(buzer,LOW);
 delay(100);
 digitalWrite(buzer,HIGH);
 delay(100);
 digitalWrite(buzer,LOW);
 delay(100);
```

```
 }
void bunyi_2(){
 digitalWrite(buzer,HIGH);
 delay(500);
 digitalWrite(buzer,LOW);
 delay(100);
 digitalWrite(buzer,HIGH);
 delay(500);
 digitalWrite(buzer,LOW);
 delay(100);
 digitalWrite(buzer,HIGH);
 delay(500);
 digitalWrite(buzer,LOW);
 delay(100);

 }
```

I2C

```
const uint8_t IMUAddress = 0x68; // AD0 is logic low on the PCB
const uint16_t I2C_TIMEOUT = 1000; // Used to check for errors in I2C communication

uint8_t i2cWrite(uint8_t registerAddress, uint8_t data, bool sendStop) {
 return i2cWrite(registerAddress, &data, 1, sendStop); // Returns 0 on success
}

uint8_t i2cWrite(uint8_t registerAddress, uint8_t *data, uint8_t length, bool sendStop) {
 Wire.beginTransmission(IMUAddress);
 Wire.write(registerAddress);
 Wire.write(data, length);
 uint8_t rcode = Wire.endTransmission(sendStop); // Returns 0 on success
 if (rcode) {
  Serial.print(F("i2cWrite failed: "));
  Serial.println(rcode);
 }
```

```
 return rcode; // See: http://arduino.cc/en/Reference/WireEndTransmission
}

uint8_t i2cRead(uint8_t registerAddress, uint8_t *data, uint8_t nbytes) {
 uint32_t timeOutTimer;
 Wire.beginTransmission(IMUAddress);
 Wire.write(registerAddress);
 uint8_t rcode = Wire.endTransmission(false); // Don't release the bus
 if (rcode) {
  Serial.print(F("i2cRead failed: "));
  Serial.println(rcode);
  return rcode; // See: http://arduino.cc/en/Reference/WireEndTransmission
 }
 Wire.requestFrom(IMUAddress, nbytes, (uint8_t)true); // Send a repeated start and then release the bus after reading
 for (uint8_t i = 0; i < nbytes; i++) {
  if (Wire.available())
   data[i] = Wire.read();
  else {
   timeOutTimer = micros();
   while (((micros() - timeOutTimer) < I2C_TIMEOUT) && !Wire.available());
   if (Wire.available())
    data[i] = Wire.read();
   else {
    Serial.println(F("i2cRead timeout"));
    return 5; // This error value is not already taken by endTransmission
   }
  }
 }
 return 0; // Success
}
```

Motor

```
void Motors(){
 if (speed > 0)
 {
  //forward
  //speed = map(speed,0,-255,0,255);
  analogWrite25k(rem, speed);
  analogWrite25k(dir, 0);
 }
 else
 {
  // backward
  speed = map(speed,0,-255,0,255);
  analogWrite25k(rem, 0);
  analogWrite25k(dir, speed);
 }
}

void stop(){
 analogWrite25k(rem, 0);
 analogWrite25k(dir, 0);
}
```

MPU6050

```
void imu_data(){
 #if ARDUINO >= 157
 Wire.setClock(400000UL); // Set I2C frequency to 400kHz
#else
 TWBR = ((F_CPU / 400000UL) - 16) / 2; // Set I2C frequency to 400kHz
#endif

 i2cData[0] = 7; // Set the sample rate to 1000Hz - 8kHz/(7+1) = 1000Hz
```

```
  i2cData[1] = 0x00; // Disable FSYNC and set 260 Hz Acc filtering, 256 Hz Gyro filtering, 8 KHz sampling
  i2cData[2] = 0x00; // Set Gyro Full Scale Range to ±250deg/s
  i2cData[3] = 0x00; // Set Accelerometer Full Scale Range to ±2g
  while (i2cWrite(0x19, i2cData, 4, false)); // Write to all four registers at once
  while (i2cWrite(0x6B, 0x01, true)); // PLL with X axis gyroscope reference and disable sleep mode

  while (i2cRead(0x75, i2cData, 1));
  if (i2cData[0] != 0x68) {
   Serial.print(F("Error reading sensor"));
   while (1);
  }

  delay(100); // Wait for sensor to stabilize

  /* Set kalman and gyro starting angle */
  while (i2cRead(0x3B, i2cData, 6));
  accX = (i2cData[0] << 8) | i2cData[1];
  accY = (i2cData[2] << 8) | i2cData[3];
  accZ = (i2cData[4] << 8) | i2cData[5];

  // It is then converted from radians to degrees
#ifdef RESTRICT_PITCH // Eq. 25 and 26
  double roll  = atan2(accY, accZ) * RAD_TO_DEG;
  double pitch = atan(-accX / sqrt(accY * accY + accZ * accZ)) * RAD_TO_DEG;
#else // Eq. 28 and 29
  double roll  = atan(accY / sqrt(accX * accX + accZ * accZ)) * RAD_TO_DEG;
```

```
  double pitch = atan2(-accX, accZ) * RAD_TO_DEG;
#endif

  kalmanX.setAngle(roll); // Set starting angle
  kalmanY.setAngle(pitch);
  gyroXangle = roll;
  gyroYangle = pitch;
  compAngleX = roll;
  compAngleY = pitch;

  timer = micros();
}

void sensor(){
   /* Update all the values */
  while (i2cRead(0x3B, i2cData, 14));
  accX = ((i2cData[0] << 8) | i2cData[1]);
  accY = ((i2cData[2] << 8) | i2cData[3]);
  accZ = ((i2cData[4] << 8) | i2cData[5]);
  tempRaw = (i2cData[6] << 8) | i2cData[7];
  gyroX = (i2cData[8] << 8) | i2cData[9];
  gyroY = (i2cData[10] << 8) | i2cData[11];
  gyroZ = (i2cData[12] << 8) | i2cData[13];

  double dt = (double)(micros() - timer) / 1000000; // Calculate delta time
  timer = micros();

  // It is then converted from radians to degrees
```

```
#ifdef RESTRICT_PITCH // Eq. 25 and 26
  double roll  = atan2(accY, accZ) * RAD_TO_DEG;
  double pitch = atan(-accX / sqrt(accY * accY + accZ * accZ)) * RAD_TO_DEG;
#else // Eq. 28 and 29
  double roll  = atan(accY / sqrt(accX * accX + accZ * accZ)) * RAD_TO_DEG;
  double pitch = atan2(-accX, accZ) * RAD_TO_DEG;
#endif

  double gyroXrate = gyroX / 131.0; // Convert to deg/s
  double gyroYrate = gyroY / 131.0; // Convert to deg/s

#ifdef RESTRICT_PITCH
  // This fixes the transition problem when the accelerometer angle jumps between -180 and 180 degrees
  if ((roll < -90 && kalAngleX > 90) || (roll > 90 && kalAngleX < -90)) {
    kalmanX.setAngle(roll);
    compAngleX = roll;
    kalAngleX = roll;
    gyroXangle = roll;
  } else
    kalAngleX = kalmanX.getAngle(roll, gyroXrate, dt); // Calculate the angle using a Kalman filter

  if (abs(kalAngleX) > 90)
    gyroYrate = -gyroYrate; // Invert rate, so it fits the restriced accelerometer reading
  kalAngleY = kalmanY.getAngle(pitch, gyroYrate, dt);
#else
  // This fixes the transition problem when the accelerometer angle jumps between -180 and 180 degrees
```

```
  if ((pitch < -90 && kalAngleY > 90) || (pitch > 90 && kalAngleY < -90)) {
    kalmanY.setAngle(pitch);
    compAngleY = pitch;
    kalAngleY = pitch;
    gyroYangle = pitch;
  } else
    kalAngleY = kalmanY.getAngle(pitch, gyroYrate, dt); // Calculate the angle using a
Kalman filter

  if (abs(kalAngleY) > 90)
    gyroXrate = -gyroXrate; // Invert rate, so it fits the restriced accelerometer reading
  kalAngleX = kalmanX.getAngle(roll, gyroXrate, dt); // Calculate the angle using a
Kalman filter
#endif

  gyroXangle += gyroXrate * dt; // Calculate gyro angle without any filter
  gyroYangle += gyroYrate * dt;
  //gyroXangle += kalmanX.getRate() * dt; // Calculate gyro angle using the unbiased rate
  //gyroYangle += kalmanY.getRate() * dt;

  compAngleX = 0.93 * (compAngleX + gyroXrate * dt) + 0.07 * roll; // Calculate the
angle using a Complimentary filter
  compAngleY = 0.93 * (compAngleY + gyroYrate * dt) + 0.07 * pitch;

  // Reset the gyro angle when it has drifted too much
  if (gyroXangle < -180 || gyroXangle > 180)
    gyroXangle = kalAngleX;
  if (gyroYangle < -180 || gyroYangle > 180)
    gyroYangle = kalAngleY;
```

```
  /* Print Data */
#if 0 // Set to 1 to activate
  Serial.print(accX); Serial.print("\t");
  Serial.print(accY); Serial.print("\t");
  Serial.print(accZ); Serial.print("\t");

  Serial.print(gyroX); Serial.print("\t");
  Serial.print(gyroY); Serial.print("\t");
  Serial.print(gyroZ); Serial.print("\t");

  Serial.print("\t");
#endif

  Serial.print(roll); Serial.print("\t");
  Serial.print(gyroXangle); Serial.print("\t");
  Serial.print(compAngleX); Serial.print("\t");
  Serial.print(kalAngleX); Serial.print("\t");

  Serial.print("\t");

  Serial.print(pitch); Serial.print("\t");
  Serial.print(gyroYangle); Serial.print("\t");
  Serial.print(compAngleY); Serial.print("\t");
  Serial.print(kalAngleY); Serial.print("\t");

#if 0 // Set to 1 to print the temperature
  Serial.print("\t");
```

```
  double temperature = (double)tempRaw / 340.0 + 36.53;
  Serial.print(temperature); Serial.print("\t");
#endif

  Serial.print("\r\n");
  delay(2);
}
```

PID

```
void Pid(){
  error = 180 - current;
  pTerm = Kp * error;
  integrated_error += error;
  iTerm = Ki * constrain(integrated_error, -GUARD_GAIN, GUARD_GAIN);
  dTerm = Kd * (error - last_error);
  last_error = error;
  speed = constrain(K*(pTerm + iTerm + dTerm), -255, 255);
  }

void filter(){
  double dt = (double)(micros() - timer) / 1000000; // Calculate delta time
  while(i2cRead(0x3B,i2cData,14));
  accX = ((i2cData[0] << 8) | i2cData[1]);
  accY = ((i2cData[2] << 8) | i2cData[3]);
  accZ = ((i2cData[4] << 8) | i2cData[5]);
  tempRaw = ((i2cData[6] << 8) | i2cData[7]);
  gyroX = ((i2cData[8] << 8) | i2cData[9]);
```

```
gyroY = ((i2cData[10] << 8) | i2cData[11]);
gyroZ = ((i2cData[12] << 8) | i2cData[13]);
accXangle = (atan2(accY,accZ)+PI)*RAD_TO_DEG;
double gyroXrate = (double)gyroX/131.0;
current = kalmanX.getAngle(accXangle, gyroXrate, dt);
timer = micros();
}
```

Skateboard

```
#include <Wire.h>
#include <Kalman.h>
#define RESTRICT_PITCH // Comment out to restrict roll to ±90deg instead - please read: http://www.freescale.com/files/sensors/doc/app_note/AN3461.pdf

Kalman kalmanX; // Create the Kalman instances
Kalman kalmanY;

int push1=13;
int push2=12;
int dir = 9;
int rem = 10;
int buzer =4;
/* IMU Data */
int16_t accX, accY, accZ;
int16_t gyroX, gyroY, gyroZ;
int16_t tempRaw;

float gyroXangle, gyroYangle, accXangle; // Angle calculate using the gyro only
```

```
float compAngleX, compAngleY; // Calculated angle using a complementary filter
float kalAngleX, kalAngleY; // Calculated angle using a Kalman filter

uint32_t timer;
uint8_t i2cData[14]; // Buffer for I2C data
float current;
// PID
const float Kp = 6.98;
const float Ki = 4.61;
const float Kd = 1.15;
float pTerm, iTerm, dTerm, integrated_error, last_error, error;
const float K = 1.9*1.12;
#define   GUARD_GAIN   10.0
int speed;

#define runEvery(t) for (static typeof(t) _lasttime;(typeof(t))((typeof(t))millis() - _lasttime)
> (t);_lasttime += (t))

// TODO: Make calibration routine
void analogWrite25k(int pin, int value)
{
   switch (pin) {
      case 9:
         OCR1A = value;
         break;
      case 10:
         OCR1B = value;
         break;
```

```
    default:
      // no other pin will work
      break;
  }
}
void setup() {
 Serial.begin(9600);
 // Configure Timer 1 for PWM @ 25 kHz.
  //TCCR1B = TCCR1B & B11111000 | B00000001; // Set PWM frequency for D9 &
D10 :
  TCCR1A = 0;         // undo the configuration done by...
  TCCR1B = 0;          // ...the Arduino core library
  TCNT1  = 0;          // reset timer
  TCCR1A = _BV(COM1A1)  // non-inverted PWM on ch. A
       | _BV(COM1B1)  // same on ch; B
       | _BV(WGM11);  // mode 10: ph. correct PWM, TOP = ICR1
  TCCR1B = _BV(WGM13)   // ditto
       | _BV(CS10);   // prescaler = 1
  ICR1   = 320;        // TOP = 320

 pinMode(push1,INPUT_PULLUP);
 pinMode(push2,INPUT_PULLUP);
 pinMode(dir,OUTPUT);
 pinMode(rem,OUTPUT);
 pinMode(buzer,OUTPUT);
  bunyi_1();
 Wire.begin();
 imu_data();
```

```
}

void loop() {
runEvery(25)  // run code @ 40 Hz
 {
  filter();
  if (current <= 181.6 && current >= 178.98)
  {
   while(digitalRead(push1)==HIGH){speed=200;Motors();}
   while(digitalRead(push2)==HIGH){speed=-200;Motors();}
  stop();
  Serial.print("Derajat = ");
  Serial.print(current);
  Serial.print("\r\n");
  Serial.print("PWM = ");
  Serial.print(speed);Serial.print("\r\n");
  }
 else{
   filter();
  if (current <= 230 && current >= 130)
  {
  Pid();
  Motors();
  Serial.print("Derajat = ");
  Serial.print(current);
  Serial.print("\r\n");
```

```
  Serial.print("PWM = ");
  Serial.print(speed);Serial.print("\r\n");
  }
 else
 {
  stop();
  Serial.print("Derajat = ");
  Serial.print(current);
  Serial.print("\r\n");
  Serial.print("PWM = ");
  Serial.print(speed);Serial.print("\r\n");
  }
 }
}
}
```

LAMPIRAN 3
Datasheet