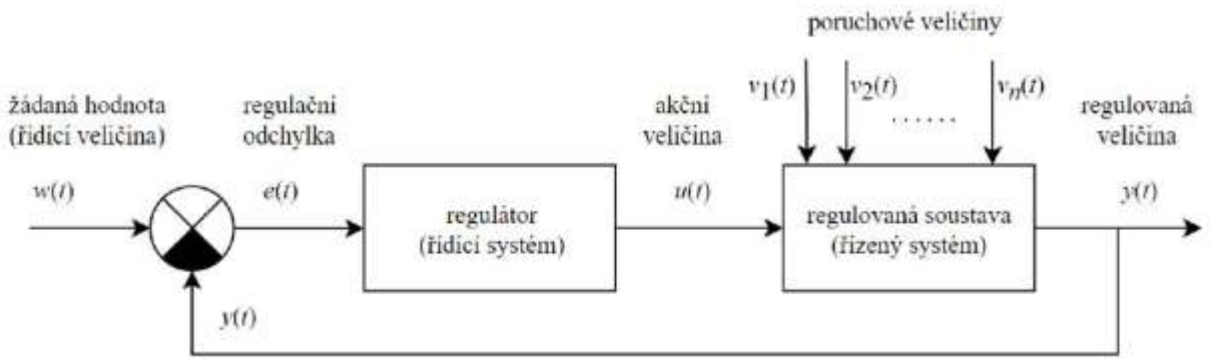
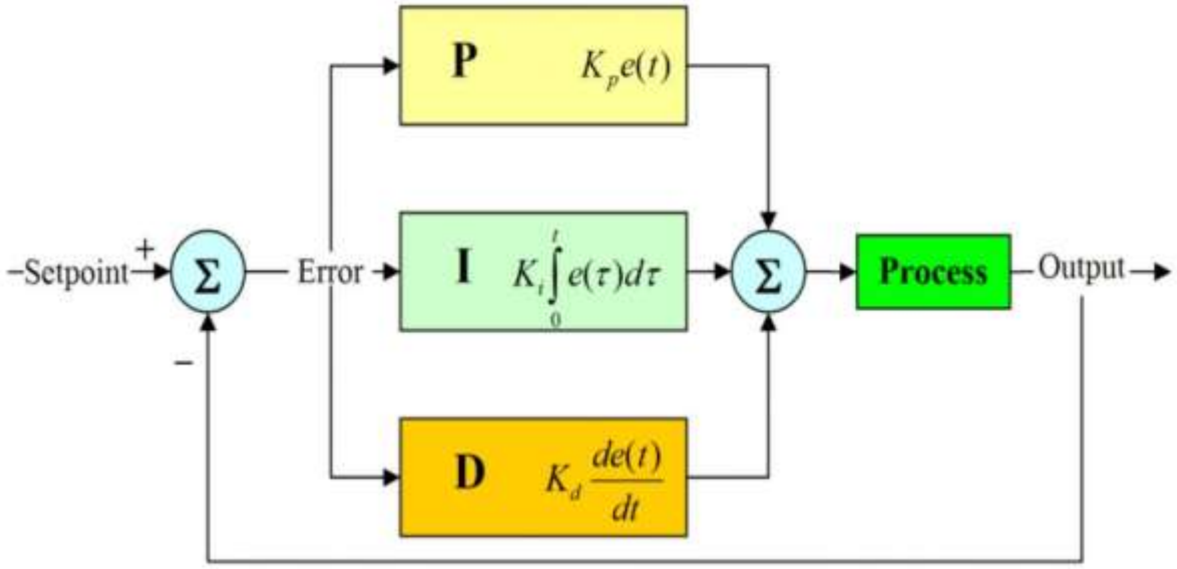
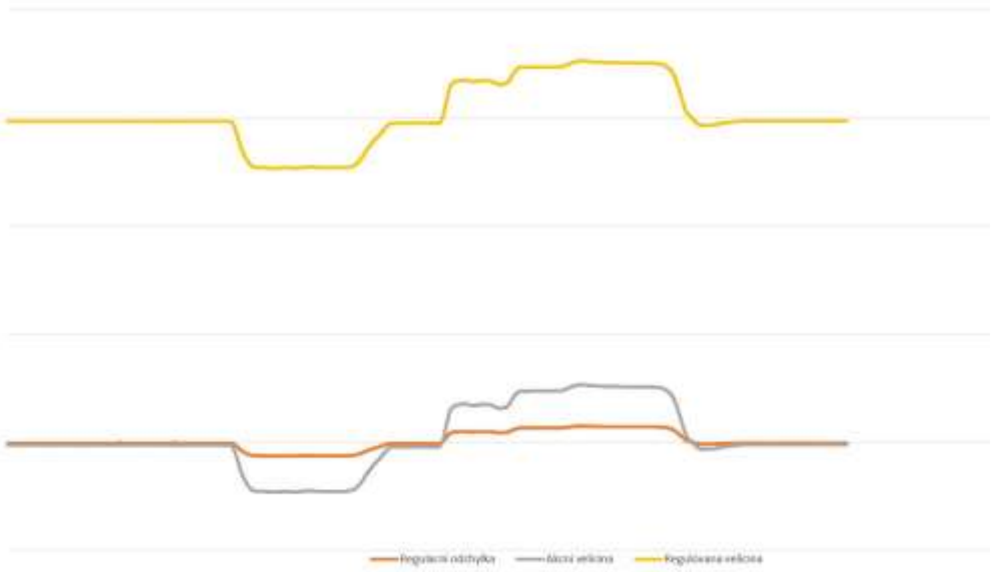
PID Regulation







```
// ------------- SIMPLE TEST SERVO LEVEL REGULATION (PID+MCU)
// ------------- DuPe 2/2021
// ------------- ARDUINO NANO + MPU 6050 (MCU GYRO/ACC/DMP)
// ------------- I2C MCU SCL - pin A5
// ------------- I2C MCU SDA - pin A4
// ------------- INTERRUPT INT0 - pin D2
// ------------- Servo PWM - pin D5
#include "Wire.h"
#include "I2Cdev.h"
#include "MPU6050_6Axis_MotionApps20.h"
#include "Servo.h"
Servo MyServo;
// 0x68 or 0x69 (AD0)
// MPU6050 mpu(0x69)
MPU6050 mpu;
bool dmpReady = false;
uint8_t mpuIntStatus;
uint8_t devStatus;
uint16_t packetSize;
float real_angle = 0;
float Time, previousTime;
int pwm;
int pwmnul=1500; // pwm pro nulovou pozici serva

// -------------- SET PID -------------
  const float kp=3.6;//3.55
  const float ki=3;//0.003
  const float kd=0.3; //2.05
  const float setpoint = 0; // nuloveho uhlu se snazime dosahnout
// ----- obsluha preruseni
volatile bool mpuInterrupt = false;
void dmpDataReady() {
  mpuInterrupt = true;
}

void setup() {
  MyServo.attach(5);
  Wire.begin();
  Serial.begin(57600);
  // Serial.println(F("MCU Initialization.."));
  mpu.initialize();
  // Serial.println(F("Test IMU Connection."));
  // Serial.println(mpu.testConnection() ? F("IMU is connected OK.") : F("IMU connection failed."));
  // Serial.println(F("Initialization Digital Motion Processor (DMP)."));
  devStatus = mpu.dmpInitialize();
  if (devStatus == 0) {
      // Serial.println(F("Enabling DMP."));
      mpu.setDMPEnabled(true);
      //  ---- nastavení preruseni
      attachInterrupt(0, dmpDataReady, RISING);
      mpuIntStatus = mpu.getIntStatus();
      // Serial.println(F("DMP is ready - wait to INTERRUPT."));
      dmpReady = true;
      packetSize = mpu.dmpGetFIFOPacketSize();
      // Serial.println (packetSize);
  }
  else {
      // Error Occured:
```

```
      // 1 : Failed Connection to DMP
      // 2 : Failed Initialization DMP
      Serial.print(F("DMP Initialization failed (code "));
      Serial.print(devStatus);
      Serial.println(F(")"));
  }
 delay (1000);
}

void loop() {
 if (!dmpReady) return;
 // ----- pri interrupt jsou data z IMU ready
 // ----- data z IMU poslu jako vstup do PID
 Time = millis();
 if (mpuInterrupt) {
  real_angle = DataRead(packetSize);
 }
 float pid = runPID(real_angle);
 // ----- dle vystupu regulatoru upravim pwm signal do serva
 pwm = pwmnul + pid;
 // ----- kontrola rozmezi hodnot a odeslani pwm do serva
 pwm = mez(pwm, 1000, 2000);
 Serial.print(" ");
 Serial.println(pwm);
 MyServo.writeMicroseconds(pwm);
 previousTime = Time;
}

float DataRead(uint16_t bufferSize){
// ----------  IMU ----------
  uint16_t fifoCount;
  uint8_t fifoBuffer[64];
  uint8_t mpuStatus;
  float rotace[3];       // yaw/pitch/roll
  float angle;
  Quaternion q;          // [w, x, y, z] kvaternion
  VectorFloat gravity;   // [x, y, z] vektor setrvačnosti
  VectorInt16 aa;        // [x, y, z] accel sensor measurements
  VectorInt16 aaReal;    // [x, y, z] gravity-free accel sensor
  mpuStatus = mpu.getIntStatus();
  fifoCount = mpu.getFIFOCount();
  if ((mpuStatus & 0x10) || fifoCount >= 1024) {
    mpu.resetFIFO();
    Serial.println(F("Buffer Overflow !"));
    // nutné častěji vyčítat data
  }
  else if (mpuStatus & 0x02) {
    while (fifoCount < bufferSize) fifoCount = mpu.getFIFOCount();
    mpu.getFIFOBytes(fifoBuffer, bufferSize);
    fifoCount -= bufferSize;
    mpu.dmpGetQuaternion(&q, fifoBuffer);
    mpu.dmpGetAccel(&aa, fifoBuffer);
    mpu.dmpGetGravity(&gravity, &q);
    mpu.dmpGetLinearAccel(&aaReal, &aa, &gravity);
    mpu.dmpGetYawPitchRoll(rotace, &q, &gravity);
    angle = rotace[2] * 180/M_PI;
    /*
    Serial.print("Rotace \t X ");
    Serial.print(rotace[2] * 180/M_PI);
    Serial.print("st \t Y ");
    Serial.print(rotace[1] * 180/M_PI);
    Serial.print("st \t Z ");
    Serial.print(rotace[0] * 180/M_PI);
    Serial.println("st");

    Serial.print("Acc \t X ");
    Serial.print(aaReal.x);
    Serial.print("\t");
    Serial.print("Acc \t Y ");
    Serial.print(aaReal.y);
    Serial.print("\t");
    Serial.print("Acc \t Z ");
    Serial.println(aaReal.z);
    */
    return angle;
   }
}

float runPID(float input) {
  float previousError, error, dT;
  float pid_i, pid_d, output;
  error = setpoint - input;
  previousError = error;
  dT = (Time - previousTime) / 1000;
  pid_i += (ki*((error-previousError))*dT);  // integral jako soucet plochy
  // float pid_i += ki*error;  // integral jako soucet odchylek
  // float pid_d = kd*((error - previousError)/dT); //derivative
  output = kp*error + pid_i + kd*((error - previousError)/dT);
  // Serial.print(" ");
  // Serial.print(Time - previousTime);
  // Serial.print(" ");
  Serial.print(input);
  Serial.print(" ");
  Serial.print(error);
  Serial.print(" ");
  Serial.print(output);
  return output;
}

int mez(int x, int xmin, int xmax){
 if(x < xmin) { x=xmin; }
 if(x > xmax) { x=xmax;}
 return x;
}
```