UNIVERSIDADE FEDERAL DE UBERLÂNDIA - UFU

FACULDADE DE ENGENHARIA ELÉTRICA – FEELT

SINAIS E SISTEMAS EM ENGENHARIA BIOMÉDICA

**Trabalho Final: Pêndulo Invertido**



**Alunos:**

1. Ítalo Gustavo Sampaio Fernandes - 11511EBI004
2. Paulo Camargos Silva - 11611EBI023

**Prof. Dr.** Sérgio Ricardo de Jesus Oliveira

Uberlândia, **18** de **dezembro** de 2017

**1 – Introdução**

Um sistema de pêndulo invertido tem diversas aplicações. O sistema proposto é relativamente simples, porém requer uma engenhosidade na montagem e pode ser tornar complexo. A utilização dos controladores PID nestes sistemas é relativamente comum. Este projeto foi montado utilizando os controladores PID com Kd próximo de 0 devido a complexidade de configuração dos parâmetros Kp, Ki e Kd e de forma que pudéssemos avaliar a influência do controlador D na resposta do sistema.

O sistema de pendulo invertido tem o princípio de manter a haste em um ângulo definido pelo SetPoint, mesmo quando uma força externa é aplicada ao sistema, deslocando a haste para um ângulo diferente. Para isto, foram utilizados motor e potenciômetros para aplicação sinal e detecção do ângulo respectivamente.

O sistema conta ainda com a utilização de de módulo bluetooth para comunicação com o computador e smartphone sem fio. Um aplicativo simples criado para smartphone permite a visualização da posição angular da haste, das constantes Kp e Ki e sua configuração.

**2 – Materiais e Métodos**

**2.1 - Materiais**

Os materiais utilizados neste projeto foram:;

* 1 Arduíno Due;
* 1 LED RGB e 3 resistores de 10 kΩ;;
* 1 potenciômetro de 10 kΩ;
* 1 módulo bluetooth HC 05;
* 1 driver para motor L298N;
* 1 motor DC (Pololu Micro GearMotor Low Power - 6V - torque 0.3 Kg.cm - caixa de redução de 30:1 e consumo de 0.36A);
* 1 roda para motor Pololu;
* 2 pilhas de 3.7 V recarregáveis de lítio, montadas em série;
* 1 chave HH - On/Off;
* 6 Jumpers macho-fêmea para conexão da ponte H;
* 2 Jumpers macho-fêmea para conexão do bluetooth;
* 1 palito de picolé montado como seta.

**2.2 - Descrição geral de funcionamento**

O projeto tem como principal objetivo a criação de um sistema de pêndulo invertido, onde uma posição desejada deverá ser fornecida e a haste do pêndulo deverá se manter naquela posição. Distúrbios externos podem ser aplicados de forma que alterem a posição da haste. Com a utilização dos controladores PID, a haste deverá retornar a posição desejada.

Para a confecção do projeto, foi utilizado um Arduino Due para o sistema de controle e um software foi escrito de forma que permitisse o ajuste das constantes através da porta serial. Um motor DC com caixa de redução foi utilizado para rotacionar a haste. Este motor foi conectado ao módulo LN298 contendo uma ponte H. Este módulo foi conectado ao Arduino.

De forma que o sistema pudesse ser utilizado sem estar conectado ao computador e a alguma fonte de alimentação fixa, foi adicionado ao sistema um módulo bluetooth para compartilhamento e alteração dos parâmetros do sistema e um conjunto de 2 pilhas em série foi utilizado para alimentação de todo o sistema.

Através de um aplicativo simples no smartphone, é possível verificar e alterar os valores de Kp, Ki e Kd, bem como o estado do sistema (ligado/desligado). Estes procedimentos também podem ser efetuados pelo computador através da porta serial. Foi adicionado também ao sistema, um LED RGB de aviso indicando o erro do sistema. A cor do LED varia com a intensidade do erro. Erro grande, o LED possui cor próxima ao vermelho, enquanto erro pequeno, a cor fica próxima ao verde.

**2.3 - Diagrama de blocos do sistema.**

****

Figura 1: Diagrama de blocos do sistema.

O diagrama de blocos exibido acima monsta como a posição angular de saída (PV) é subtraída da posição angular de entrada (SP). Esta diferença é o erro e então sinal atuante é então calculado utilizando os valores das constantes de cada controlador, Kp, Ki e Kd. Para conversão da posição angular do sistema na unidade de medida utilizada pelo arduino (tensão elétrica), utilizamos um potenciômetro acoplado ao eixo do motor. A tensão no potenciômetro é proporcional ao ângulo do seu eixo. O sinal atuante é então enviado do Arduino para um driver (LN298). O sinal do sinal atuante indicará o sentido da rotação na qual o motor deverá girar.

**3 – Resultados e discussões**

Inicialmente, o sistema deveria incluir dois motores para a sustentação de um robô em duas rodas. O sistema incluiria o controlador PID e um sensor inercial MPU6050. O sistema foi montado e durantes os teste, algumas dificuldades como calibração dos valores das constantes e velocidade de resposta do sensor inercial. Pudemos avaliar na prática a dificuldade de calibração dos 3 parâmetros do PID e a real necessidade de um sistema rápido, pois a queda do robô pode ocorrer em um intervalo de tempo relativamente pequeno.

Após os testes do primeiro sistema, foi decidido utilizar somente 1 motor para controlar uma haste. Este sistema foi montado e foi necessário a substituição do módulo do sensor inercial por um potenciometro para leitura da posição angular. Pudemos observar que o potenciômetro proporcionou um controle mais fácil e mais preciso que o sensor.

Durante a fase de testes, ocorreram problema com o encaixe do eixo do motor com o potenciômetro de leitura do ângulo, pois o encaixe do potenciômetro é diferente do eixo da caixa de redução do motor. Foram testadas formas diferentes de encaixe. Ao final, foi decidido usar uma pequena roda com o encaixe da caixa de redução e colar junto ao eixo do potenciômetro. Não houve problemas com a transmissão de torque do motor para a carga por ela ser relativamente pequena.

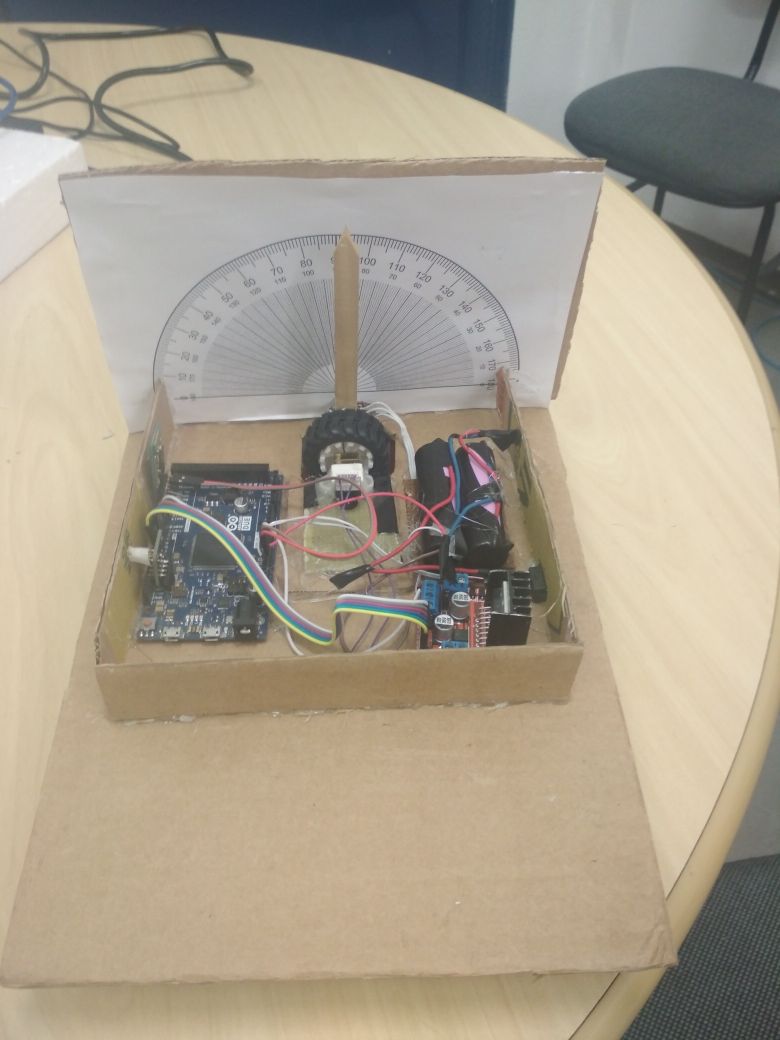
****

Figura 2: Sistema de pêndulo invertido montado.

Utilizando o código no arduino, foi possível medir o erro do sistema, configurar valores para Kp e Ki, aplicar o sinal e corrigir a posição angular da haste. Foi inserido também meio ciclo trigonométrico para verificação da posição angular como referência da haste.

Ao ligar o sistema, o sistema exige a inicialização também através do software e consequente configuração dos valores de Kp, Ki Kd. Com estes parâmetros zerados no início, a haste mantém a posição inicial em um ângulo qualquer. Ao configurar o primeiro valor de Kp, podemos observar que a haste tende a se mover para a posição desejada (em 90º), porém ao se aproximar (erro aproximadamente 10º), a velocidade se reduz inversamente proporcional ao erro. Isto é esperado, pois no controlador proporcional o sinal atuante é diretamente proporcional ao erro, e com sua redução, o sinal atuante também reduz até que não seja suficiente para mover a haste e ela se fixa em uma posição, diferente da posição desejada. Esta característica é o erro de regime permanente igual a zero.

No instante que o valor de Ki é configurado, observamos a haste mover-se até o setpoint, porém lentamente. Isto também é esperado, pois o controlador integral tem uma ação lenta, porém um bom compromisso com o erro de regime igual a 0. Desta forma, a atuação dos dois controladores nos permite uma velocidade de resposta relativamente boa para a aplicação (apenas visualização) e uma boa precisão da posição angular.

Foi testado também o funcionamento do sistema apenas com o controlador Ki. Foi observado que o sinal atuante é aplicado a todo momento e o sistema não se estabilizou. Isto é devido a própria característica do controlador I, que vai utilizando o erro acumulado para cálculo do sinal atuante. Sem a presença do controlador P, este erro acumulado não é reduzido e o sistema não atinge a estabilidade. Abaixo são apresentados gráfico plotados com o serial plotter do arduino para os valores de sinal PI e posição angular. A legenda das cores na ordem são:

* Azul escuro: setpoint;
* Vermelho ângulo medido;
* Verde: erro;
* Laranja: resposta P;
* Roxo: resposta I;
* Verde escuro: resposta D;
* Azul claro: resposta PID.

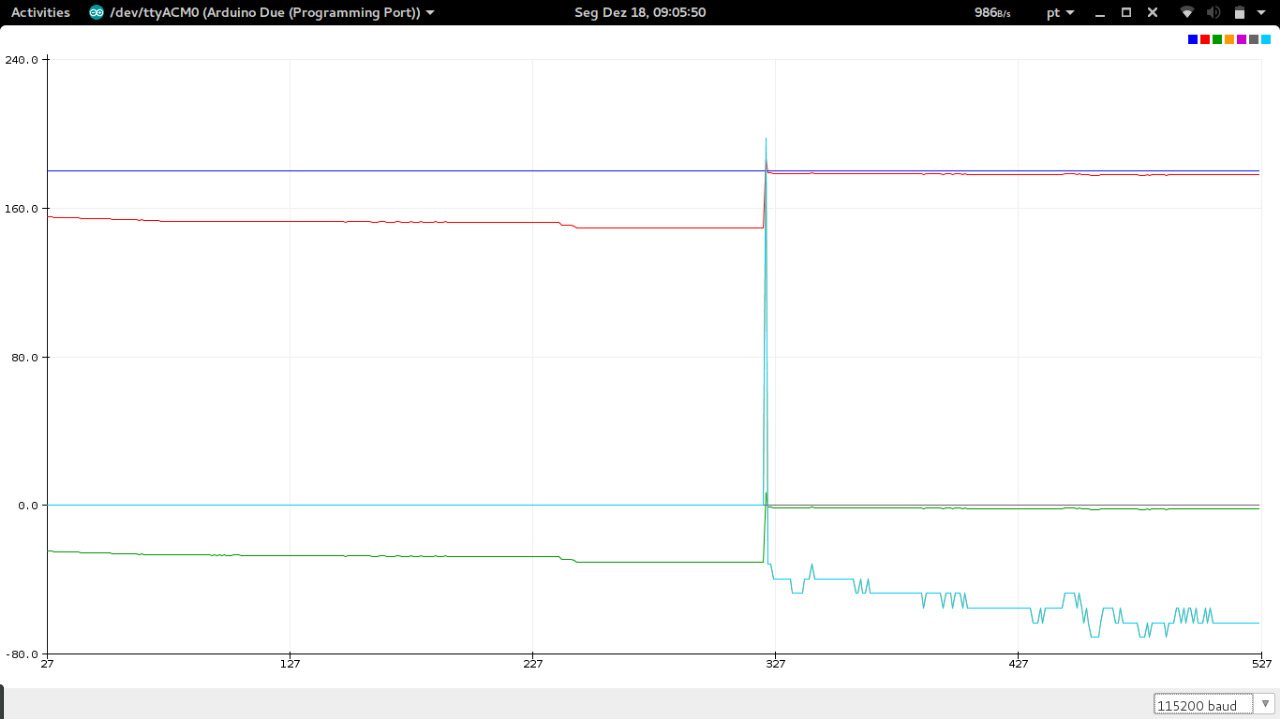


Figura 3: Kp = 30; Ki = 0. Resposta rápida, porém é possível notar a presença do erro de regime permanente.

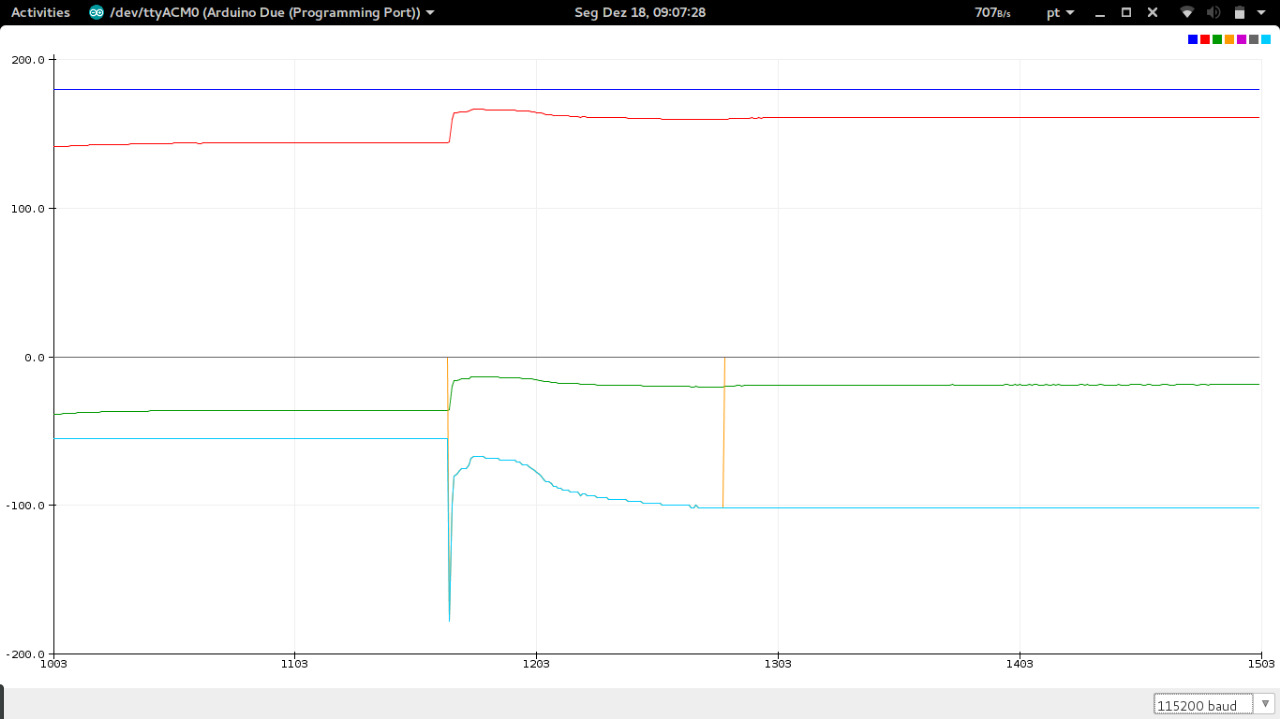


Figura 4: Kp = 5; Ki = 0. Resposta mais lenta que a anterior e com erro de regime permanente maior que o anterior.



Figura 5: Kp = 5; Ki = 2. resposta relativamente lenta, porém erro de regime permanente próximo de 0.

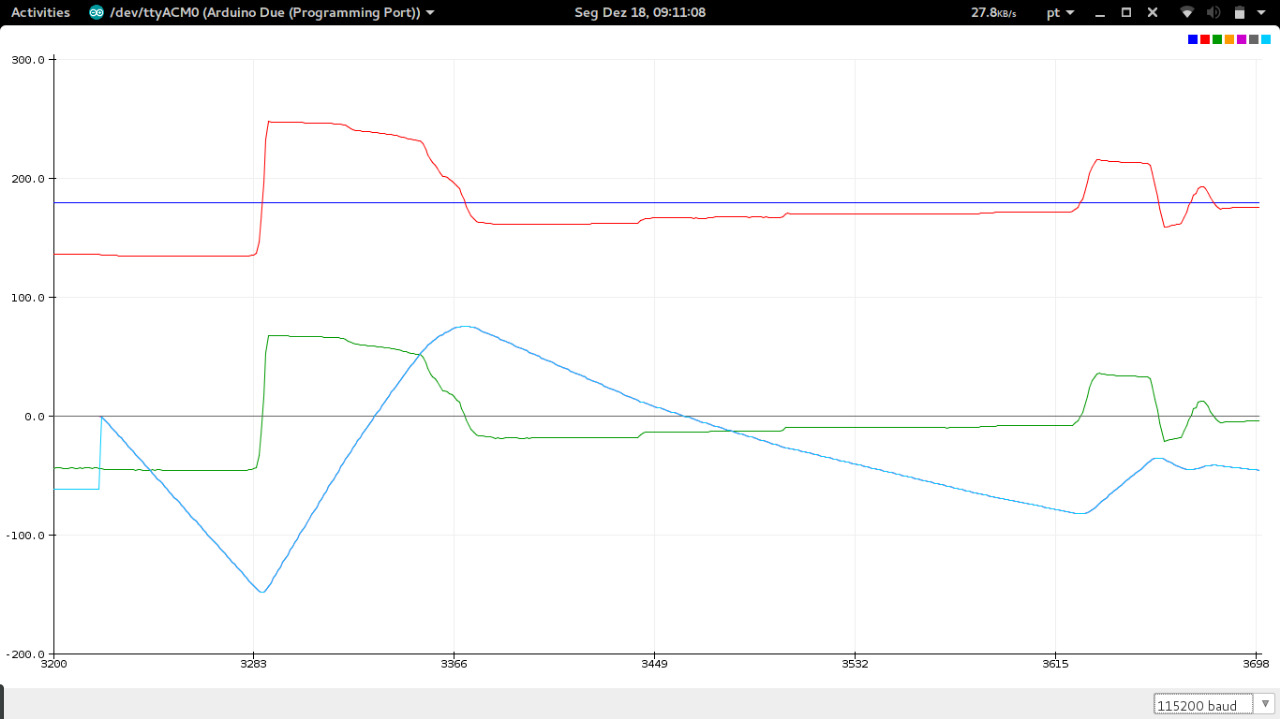


Figura 6: Kp = 0; Ki = 1. O sistema oscilou e não atingiu a estabilidade.

Foi adicionado um valor em Kd igual a 0.01 para observar se a estabilização do sistema seria mais precisa e mais rápida que os controladores PI usados anteriormente, porém, não foi possível observar diferença significativa na resposta. A figura abaixo exibe as informações deste sistema:



Figura 7: Kp =5; Ki = 2; Kd = 0.01. Inserção do controlador derivativo com este valor de Kd não afetou a resposta do sistema de forma significativa. Resposta igual a do PI mas o derivativo faz com que o sistema responda suporte melhor distúrbios externos (picos dos gráficos mostram oscilação manual do pêndulo, pode-se notar q ele nao se alterou significativamente).

**4 – Conclusão**

É possível concluir que a utilização do sistema de controle PID é uma importante ferramenta com diversas aplicações tecnológicas. Através dele foi possível implementar um sistema de robusto, capaz de se adequar a diversas aplicações, não somente ao hardware montado. Tal sistema pode ser aplicado ao controle de equilíbrio de robôs, controle de posição em próteses eletrônicas e, recentemente, foi aplicado para o controle de um robô ciclista capaz de andar de bicicleta mantendo seu equilíbrio como um humano.

**5 – Referências**

1. **Embarcados.** Controlador Proporcional Eletrônico. Disponível em: <<https://www.embarcados.com.br/controlador-proporcional-eletronico/>>;
2. **Arduino**. Documentação. Disponível em: <<https://www.arduino.cc/reference/>>;
3. **National Instruments.** PID Control. Disponível em: <<http://www.ni.com/white-paper/3782/pt/>>
4. **Sebastian O.H. Madgwick**. 2010. An efficient orientation filter for inertial and
5. inertial/magnetic sensor arrays.

**Anexo I - CÓDIGO**

Disponível em: <https://github.com/italogfernandes/SEB/tree/master/projeto_final_potenciometro_e_motor/projeto_final_potenciometro>

/\* UNIVERSIDADE FEDERAL DE UBERLANDIA  
 Biomedical Engineering  
 Autors: Ítalo G S Fernandes  
 Paulo Camargos Silva  
 contact: italogsfernandes@gmail.com  
 URLs: https://github.com/italogfernandes/SEB  
 Este codigo faz parte da disciplina de sinais e sistemas  
 para engenhara biomedica e visa controlar o angulo de  
 Os sinais de entrada e saida sao enviados para a interface Serial  
 podendo ser visualizado pelo serial plotter.  
 Esquema de montagem:  
 Arduino - Dispositivo  
 A0 - Potenciometro para configurar o setPoint de 0 a 360 graus  
  
 A1 - Potenciometro para configurar o Kp de 0 a 3  
 A2 - Potenciometro para configurar o Ki de 0 a 3  
 A3 - Potenciometro para configurar o Kd de 0 a 3  
  
 VCC - Potenciometros entrada  
 GND - Potenciometros GND  
  
 6 - Pino Enable ou de Velocidade da PonteH  
 5 - Pino In1 da PonteH  
 4 - Pino In2 da PonteH  
  
 A4 - SCL do MPU66050  
 A5 - SDA do MPU6050  
  
 Obs: O sentido do motor  
\*/  
#include<led\_rgb.h>  
#include<cores\_rgb.h>  
led\_rgb status\_led(11, 12, 13); //R\_pin,G\_pin,B\_pin  
  
#define ANALOG\_ANGULO A0  
  
#define PINO\_MOTOR\_A 2  
#define PINO\_IN1 3  
#define PINO\_IN2 4  
#define PINO\_IN3 5  
#define PINO\_IN4 6  
#define PINO\_MOTOR\_B 7  
  
float set\_point; //Angulo desejado,  
float angulo\_lido; //Angulo lido do sensor de 45º a 315ª no potenciometro  
float erro\_angulo; //Erro = setpoint - valor\_atual  
float last\_erro;  
float deltaErro; //Erro(t) - Erro(t-1)  
  
float Kp, Ki, Kd;  
float res\_pid; //Saida o Controlador PID  
float res\_proporcional; //Saida o Controlador proporcional  
float res\_integral; //Saida o Controlador integral  
float res\_derivativo; //Saida o Controlador derivativo  
float deltaT;  
unsigned long actualTime;  
unsigned long lastTime;  
  
unsigned long last\_print\_time;  
  
bool executando;  
  
int counter\_bt;  
  
void setup() {  
 Kp = 0;  
 Ki = 0;  
 Kd = 0;  
 set\_point = 180;  
 executando = false;  
  
 Serial.begin(115200);//Inicia a Serial  
 Serial1.begin(9600);//Inicia a Serial  
 pinMode(PINO\_MOTOR\_A, OUTPUT);  
 pinMode(PINO\_IN1, OUTPUT);  
 pinMode(PINO\_IN2, OUTPUT);  
 pinMode(PINO\_MOTOR\_B, OUTPUT);  
 pinMode(PINO\_IN3, OUTPUT);  
 pinMode(PINO\_IN4, OUTPUT);  
 status\_led.init();  
 setDirection(true);  
}  
  
  
String cmdserial;  
void loop() {  
 //----------------------------------------------------  
 //Ajusta Valor das constanste Kp,Ki e Kd se necessario  
  
 recebe\_comando\_serial(); //Atualiza variaveis  
 recebe\_comando\_bluetooth(); //Atualiza variaveis  
  
 //----------------------------------------------------  
 //Realiza leitura do angulo e conversao  
 angulo\_lido = analogRead(ANALOG\_ANGULO) \* 270.0f / 1024.0f + 45.0f;; //Angulo lido de 45 ate 315  
  
 //----------------------------------------------------  
 //Calculo do erro, deltaErro e DeltaT  
 erro\_angulo = angulo\_lido - set\_point;  
 deltaErro = erro\_angulo - last\_erro;  
  
 actualTime = micros();  
 deltaT = (float) (actualTime - lastTime ) / 1000000.0;  
  
 last\_erro = erro\_angulo;  
 lastTime = actualTime;  
 //----------------------------------------------------  
 //Executa o controle PID  
 if (executando) {  
 controle\_pid(); //Executa o controle em si  
 } else {  
 digitalWrite(PINO\_MOTOR\_B, 0);  
 }  
  
 //----------------------------------------------------  
 //Mostra status  
 if (millis() - last\_print\_time >= 100) {  
 last\_print\_time = millis();  
 enviar\_status();//Mostra o status do sistema  
 }  
 atualizar\_led(); //E muda a cor de um led rgb  
}  
  
void recebe\_comando\_serial() {  
 if (Serial.available()) {  
 cmdserial = Serial.readStringUntil('\n');  
 if (cmdserial.startsWith("?")) {  
 mostrar\_constantes\_serial();  
 res\_proporcional = 0; res\_integral = 0; res\_derivativo = 0;  
 } else if (cmdserial.startsWith("kp")) {  
 Kp = cmdserial.substring(2).toFloat();  
 mostrar\_constantes\_serial();  
 res\_proporcional = 0; res\_integral = 0; res\_derivativo = 0;  
 } else if (cmdserial.startsWith("ki")) {  
 Ki = cmdserial.substring(2).toFloat();  
 mostrar\_constantes\_serial();  
 res\_proporcional = 0; res\_integral = 0; res\_derivativo = 0;  
 } else if (cmdserial.startsWith("kd")) {  
 Kd = cmdserial.substring(2).toFloat();  
 mostrar\_constantes\_serial();  
 res\_proporcional = 0; res\_integral = 0; res\_derivativo = 0;  
 } else if (cmdserial.startsWith("vai")) {  
 executando = true;  
 Serial.println("\*\*\*\*\*\*\*\*\*EXECUCAO INICIADA\*\*\*\*\*\*\*\*");  
 res\_proporcional = 0; res\_integral = 0; res\_derivativo = 0;  
 } else if (cmdserial.startsWith("para")) {  
 executando = false;  
 Serial.println("\*\*\*\*\*\*\*\*\*EXECUCAO INTERROMPIDA\*\*\*\*\*\*\*\*");  
 res\_proporcional = 0; res\_integral = 0; res\_derivativo = 0;  
 } else if (cmdserial.startsWith("set")) {  
 //----------------------------------------------------  
 //Realiza leitura do setPoint  
 set\_point = cmdserial.substring(3).toFloat();  
 Serial.print("Set Point setado para: ");  
 Serial.println(set\_point);  
 res\_proporcional = 0; res\_integral = 0; res\_derivativo = 0;  
 }  
 }  
}  
  
void recebe\_comando\_bluetooth() {  
 if (Serial1.available()) {  
 cmdserial = Serial1.readStringUntil('\n');  
 if (cmdserial.startsWith("?")) {  
 mostrar\_constantes\_bt();  
 res\_proporcional = 0; res\_integral = 0; res\_derivativo = 0;  
 } else if (cmdserial.startsWith("kp")) {  
 Kp = cmdserial.substring(2).toFloat();  
 mostrar\_constantes\_bt();  
 res\_proporcional = 0; res\_integral = 0; res\_derivativo = 0;  
 } else if (cmdserial.startsWith("ki")) {  
 Ki = cmdserial.substring(2).toFloat();  
 mostrar\_constantes\_bt();  
 res\_proporcional = 0; res\_integral = 0; res\_derivativo = 0;  
 } else if (cmdserial.startsWith("kd")) {  
 Kd = cmdserial.substring(2).toFloat();  
 mostrar\_constantes\_bt();  
 res\_proporcional = 0; res\_integral = 0; res\_derivativo = 0;  
 } else if (cmdserial.startsWith("vai")) {  
 executando = true;  
 Serial1.println("\*\*\*\*\*\*\*\*\*EXECUCAO INICIADA\*\*\*\*\*\*\*\*");  
 res\_proporcional = 0; res\_integral = 0; res\_derivativo = 0;  
 } else if (cmdserial.startsWith("para")) {  
 executando = false;  
 Serial1.println("\*\*\*\*\*\*\*\*\*EXECUCAO INTERROMPIDA\*\*\*\*\*\*\*\*");  
 res\_proporcional = 0; res\_integral = 0; res\_derivativo = 0;  
 } else if (cmdserial.startsWith("set")) {  
 //----------------------------------------------------  
 //Realiza leitura do setPoint  
 set\_point = cmdserial.substring(3).toFloat();  
 Serial1.print("Set Point setado para: ");  
 Serial1.println(set\_point);  
 res\_proporcional = 0; res\_integral = 0; res\_derivativo = 0;  
 }  
 }  
}  
  
void mostrar\_constantes\_serial() {  
 Serial.print("Kp setado para: ") +  
 Serial.println(Kp);  
 Serial.print("Ki setado para: ") +  
 Serial.println(Ki);  
 Serial.print("Kd setado para: ") +  
 Serial.println(Kd);  
}  
void mostrar\_constantes\_bt() {  
 Serial1.print("Kp setado para: ") +  
 Serial1.println(Kp);  
 Serial1.print("Ki setado para: ") +  
 Serial1.println(Ki);  
 Serial1.print("Kd setado para: ") +  
 Serial1.println(Kd);  
}  
  
void atualizar\_led() {  
 if (abs(erro\_angulo) > 30 ) {  
 status\_led.acender(0xFF << 16 | 0x00 << 8);  
 } else {  
 uint16\_t erro\_porcento = (abs(erro\_angulo) \* 255.0f / 30.0f);  
 uint8\_t vermelho = erro\_porcento > 255 ? 255 : erro\_porcento;  
 vermelho = erro\_porcento < 0 ? 0 : erro\_porcento;  
 uint8\_t verde = 255 - vermelho;  
 status\_led.acender(vermelho << 16 | verde << 8);  
 }  
}  
  
void enviar\_status() {  
 Serial.println(  
 String(set\_point, 2) + "\t" +  
 String(angulo\_lido, 2) + "\t" +  
 String(erro\_angulo, 2) + "\t" +  
 String(res\_proporcional, 2) + "\t" +  
 String(res\_integral, 2) + "\t" +  
 String(res\_derivativo, 2) + "\t" +  
 String(res\_pid, 2)  
 );  
 ++counter\_bt %= 5;  
 if (counter\_bt == 0) {  
 Serial1.println(  
 String(set\_point, 2) + "\t" +  
 String(angulo\_lido, 2) + "\t" +  
 String(erro\_angulo, 2) + "\t" +  
 String(res\_proporcional, 2) + "\t" +  
 String(res\_integral, 2) + "\t" +  
 String(res\_derivativo, 2) + "\t" +  
 String(res\_pid, 2)  
 );  
 }  
}  
  
void setDirection(bool is\_clockwise) {  
 digitalWrite(PINO\_IN1, is\_clockwise);  
 digitalWrite(PINO\_IN2, !is\_clockwise);  
 digitalWrite(PINO\_IN3, is\_clockwise);  
 digitalWrite(PINO\_IN4, !is\_clockwise);  
}  
  
void controle\_pid() {  
  
 //----------------------------------------------------  
 //Calcula as respostas P, I e D  
 res\_proporcional = Kp \* erro\_angulo ; //Multiplicaçao  
 res\_integral = Ki \* erro\_angulo \* deltaT + res\_integral; //Integração  
 res\_derivativo = Kd \* deltaErro / deltaT; //Derivação  
  
 res\_pid = res\_proporcional + res\_integral + res\_derivativo;  
 res\_pid = res\_pid < -255 ? -255 : res\_pid; //Limite minimo de -255  
 res\_pid = res\_pid > 255 ? 255 : res\_pid; //Limite maximo de 255  
  
 //----------------------------------------------------  
 //Joga o sinal na saida do sistema  
 setDirection(res\_pid < 0);  
 analogWrite(PINO\_MOTOR\_B, (uint8\_t) abs(res\_pid));  
  
 //----------------------------------------------------  
 //Protecao do sistema..  
 if (angulo\_lido > 245 && res\_pid < 0) {  
 analogWrite(PINO\_MOTOR\_B, 0);  
 }  
   
 if (angulo\_lido < 130 && res\_pid > 0) {  
 analogWrite(PINO\_MOTOR\_B, 0);  
 }  
}