Controle de um meio-drone

Apresentação do projeto

Rubens Heitor Vitor

Aplicação

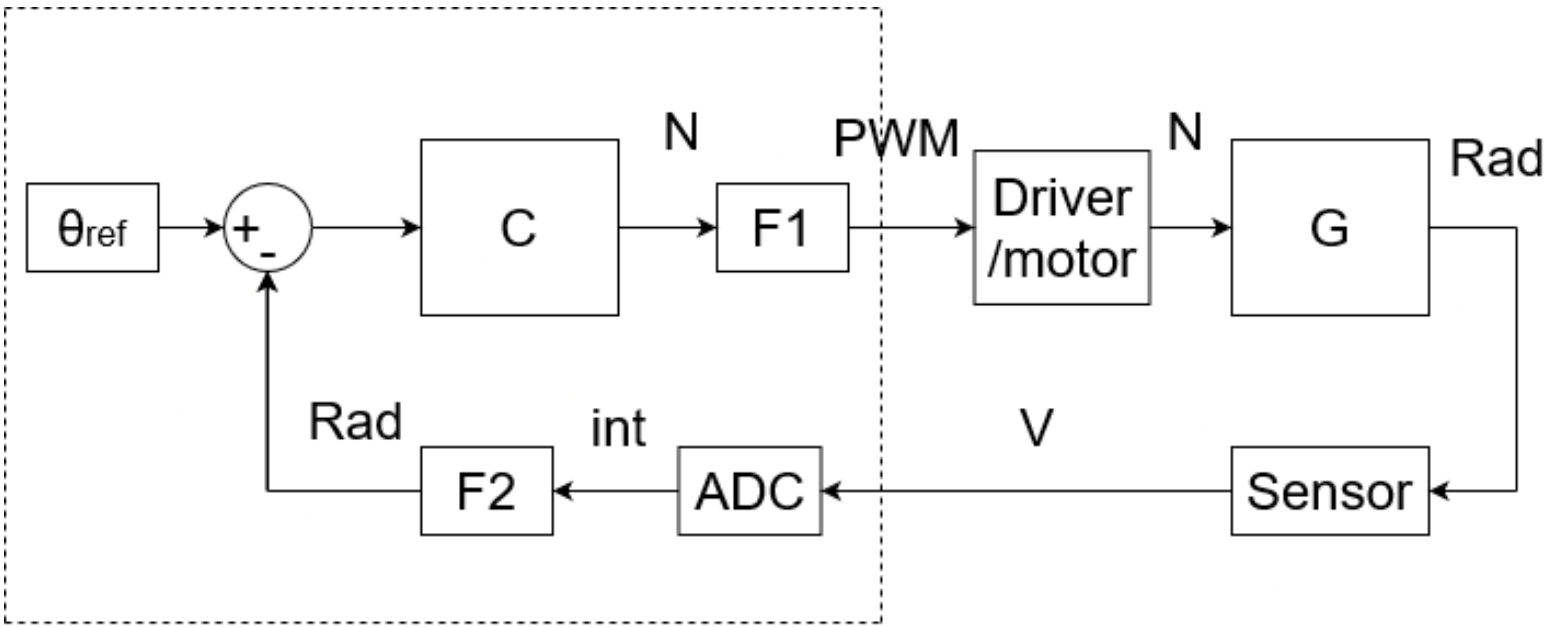
- Drone cinematográfico que realiza a gravação de vídeos e tira fotografias

O que foi feito

- Diagrama de blocos/Overview
- Modelar a planta [θ/F]
- Projetar o controlador [F/θ]
- Caracterização do sensor [V/θ]
  - Em conjunto com o ADC [int/θ]
- Caracterização dos atuadores [F/PWM]
  - Converter a saída do controlar de Newton para um sinal PWM
- Planejador de trajetória
- Melhorias/Discrepâncias

Diagrama de blocos

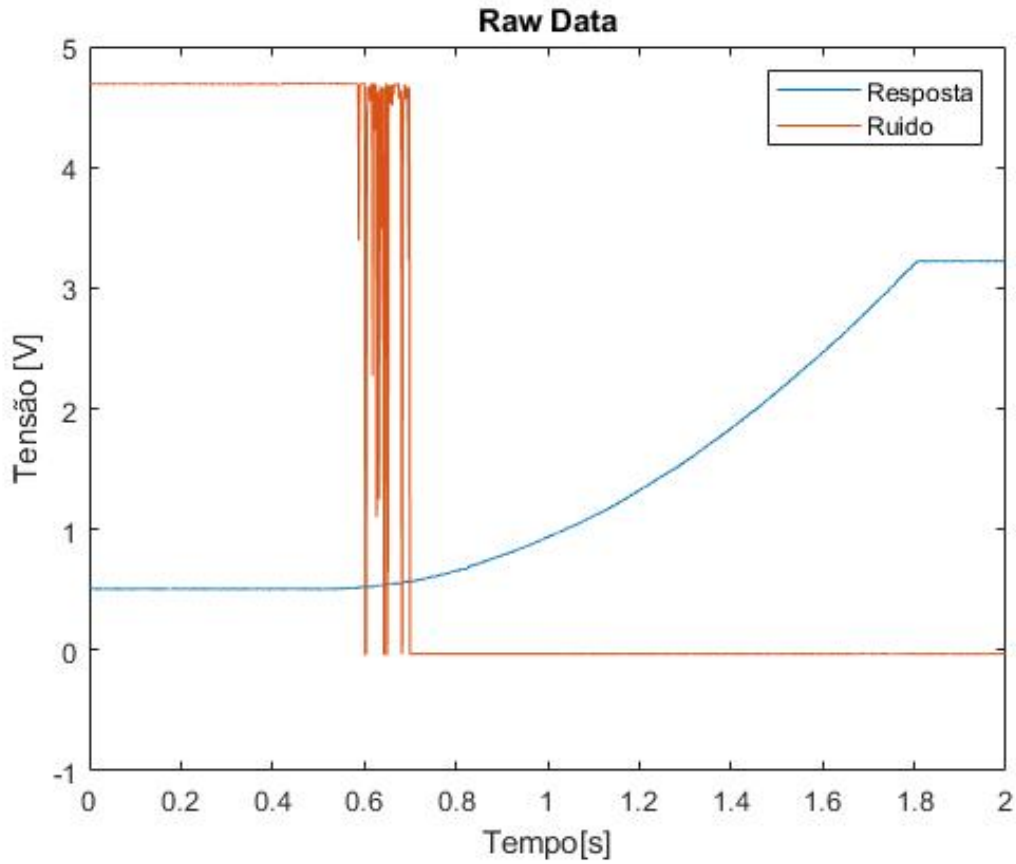
Figure: Diagrama de blocos



Fonte: Autoria própria.

Modelagem da planta I

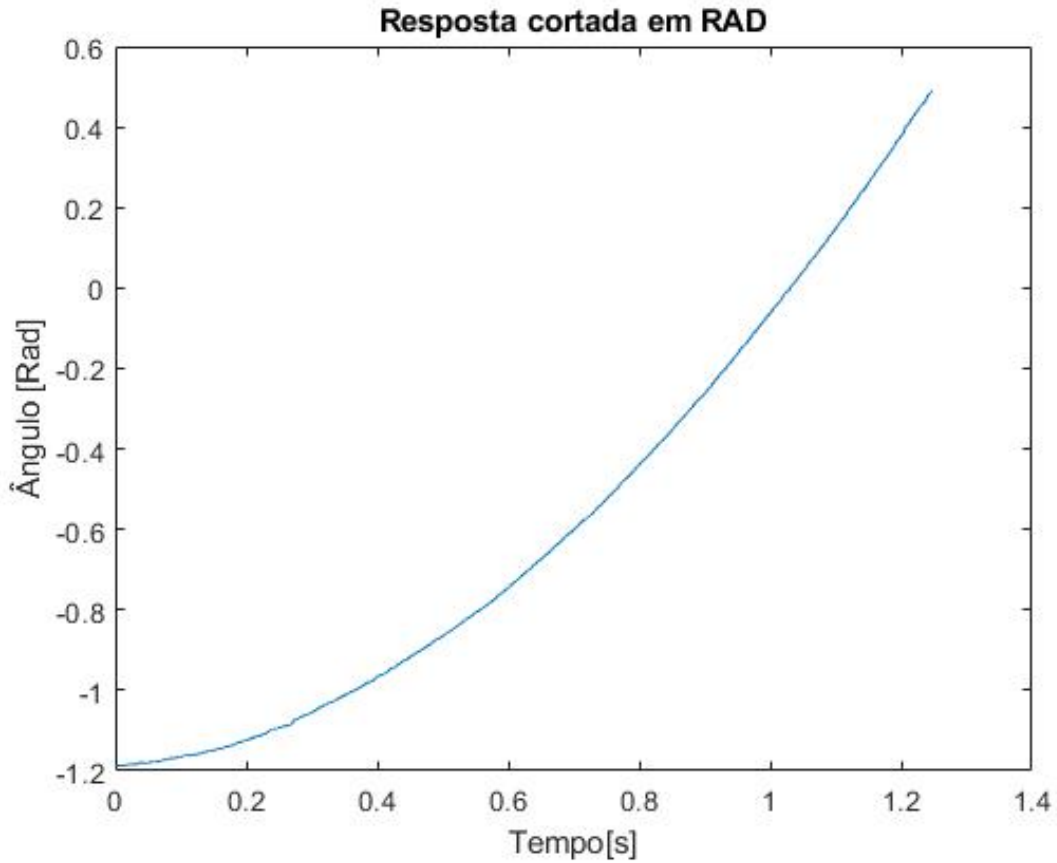
Figure: Resposta a entrada degrau



Fonte: Dados fornecidos "scope_0.csv"

Modelagem da planta II

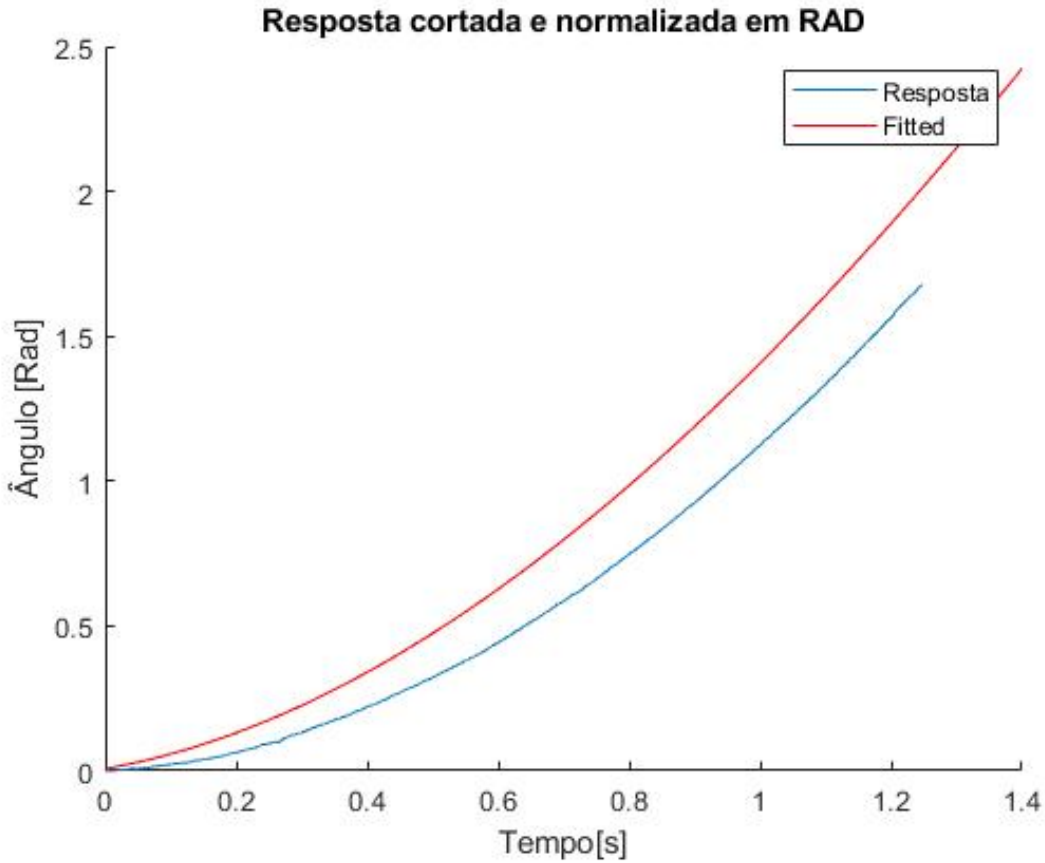
Figure: Janelamento da resposta



Fonte: Autoria própria.

Modelagem da planta III

Figure: Fitting da curva



Fonte: Autoria própria.

Modelagem da planta IV

Figure: Eq. do fitting

```
General model:
val(t) = a*t - b + c*exp(-d*t)
Coefficients (with 95% confidence bounds):
  a =       5.429  (5.203, 5.655)
  b =       10.83  (9.738, 11.93)
  c =       10.84  (9.746, 11.94)
  d =      0.4646  (0.4379, 0.4914)
```

Fonte: Autoria própria.

Modelagem da planta V

- A partir dos coeficientes encontrados, sendo $A = 0.1455\ [N]$, e sabendo que a resposta é dada por:

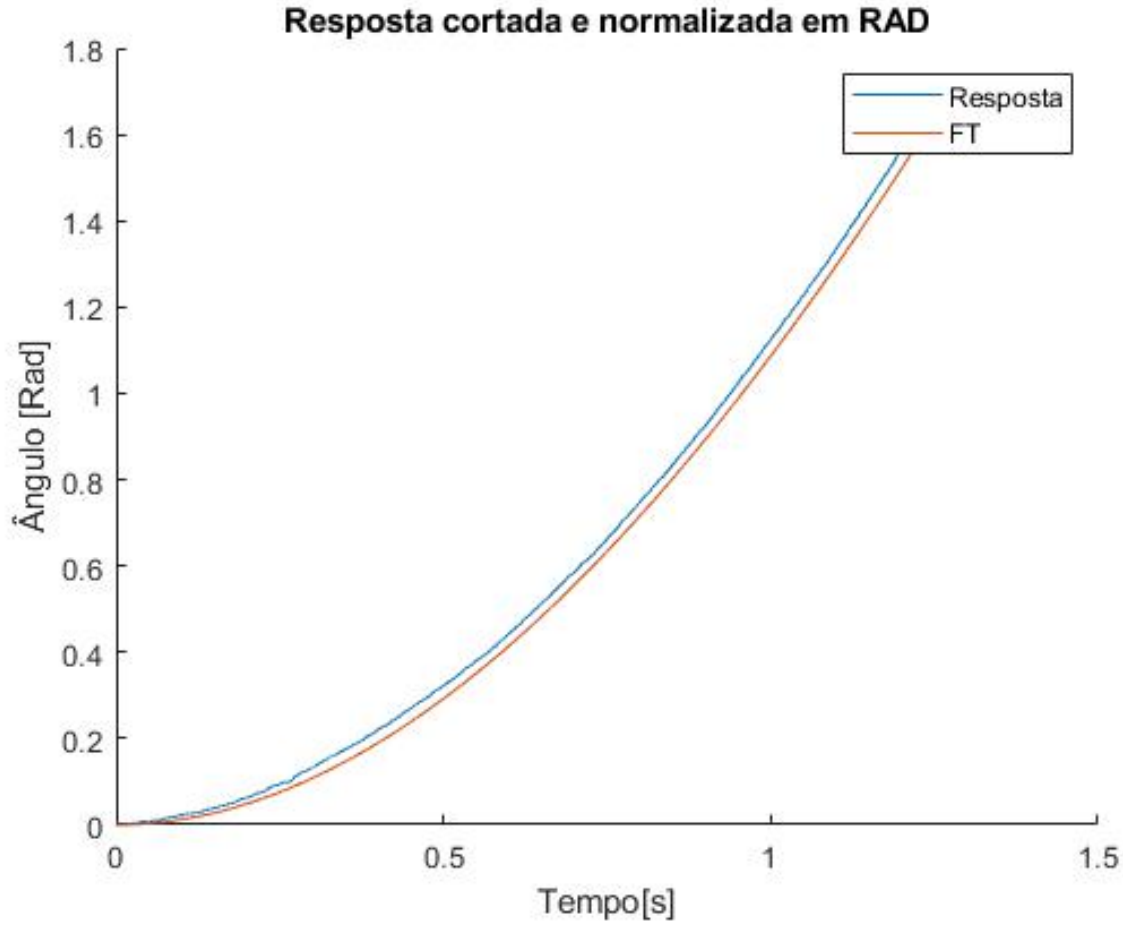
$$H(s) = \frac{K}{Js^2 + Bs} * \frac{A}{s}$$

- Podemos então estimar a planta como:

$$G(s) = \frac{17,34}{s^2 + 0,4646s} \quad \left[\frac{\theta}{F}\right]$$

Modelagem da planta VI

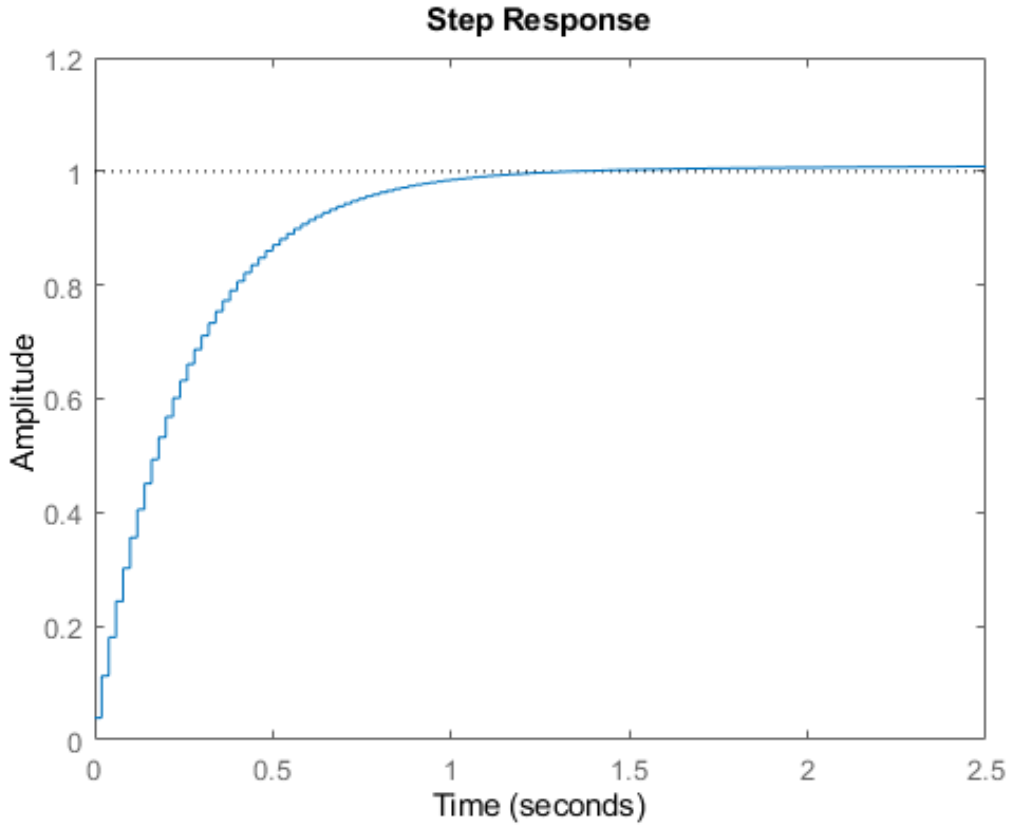
Figure: Resposta degrau, experimental e $G(s)$



Fonte: Autoria própria.

Projeto do controlador I

Figure: Resposta degrau, com controlador $C(z)$



Fonte: Autoria própria.

Projeto do controlador II

- Baseado na planta discretizada
- Resposta superamortecida
- $Kp = 0.113,\ \ Ti = 16.9,\ \ Td = 2.16$

Implementação do controlador I

$$u[k] = kp * \left[1 + \frac{1}{Ti}\left(I[k-1] + Ts\frac{e[k]+e[k-1]}{2}\right) + Td\left(\frac{e[k]-e[k-1]}{Ts}\right)\right] \quad (1)$$

- Integral tipo trapezoidal
- Derivada tipo Euler para trás

Implementação do controlador II

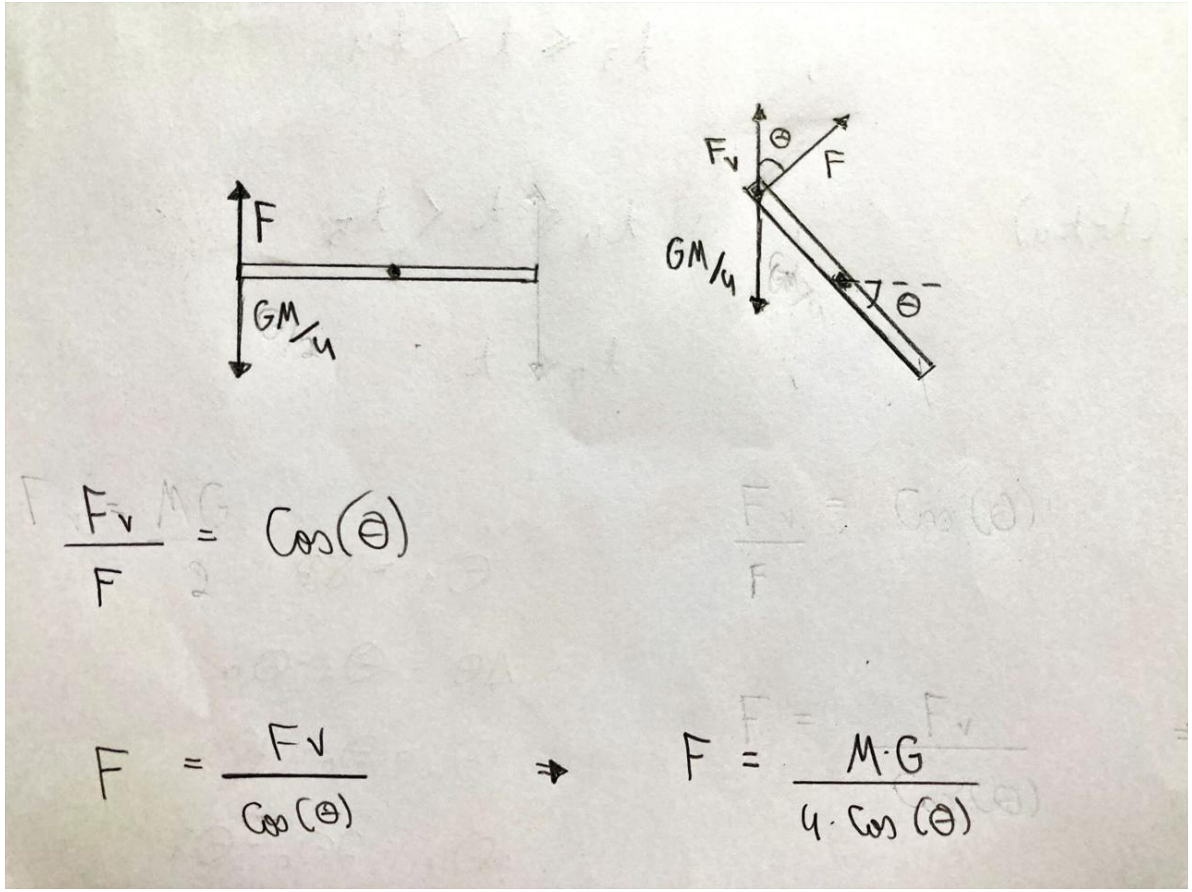
Figure: Implementação de $C(z)$

```
321         u = Kp * (angle_rad_error + 1/Ti * integral + Td * derivate);
322
323         pwm_r = f_to_pwm_rigth(M*G/(4*cosf(angle_rad)) + u/2);
324         pwm_l = f_to_pwm_left(M*G/(4*cosf(angle_rad)) - u/2);
325
326         PWM_setCombinedValue(&htim3, clip((int16_t) (10*pwm_l), 50, 950)
327                 , clip((int16_t) (10*pwm_r), 50, 950));
```

Fonte: Autoria própria.

Implementação do controlador III

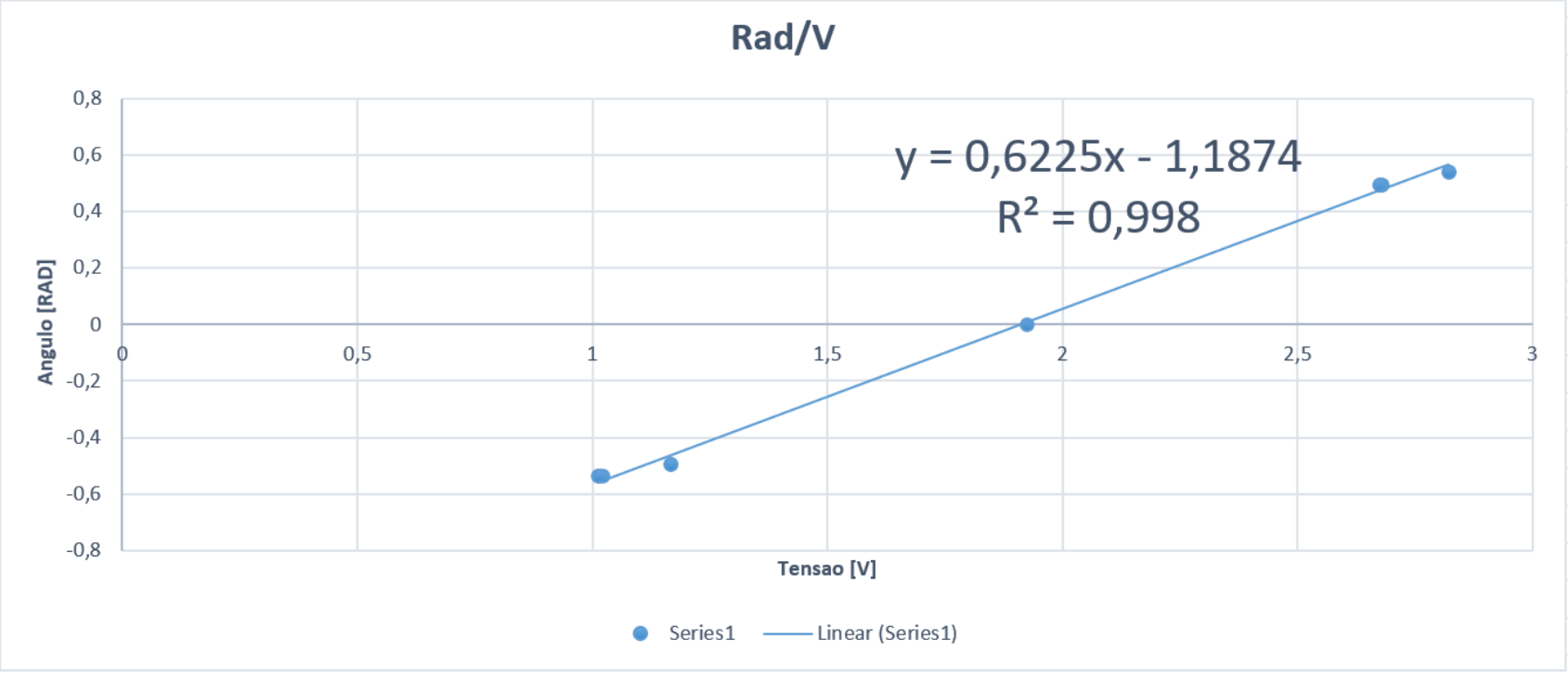
Figure: Forças sobre o drone



Fonte: Autoria própria.

Caracterização do sensor I

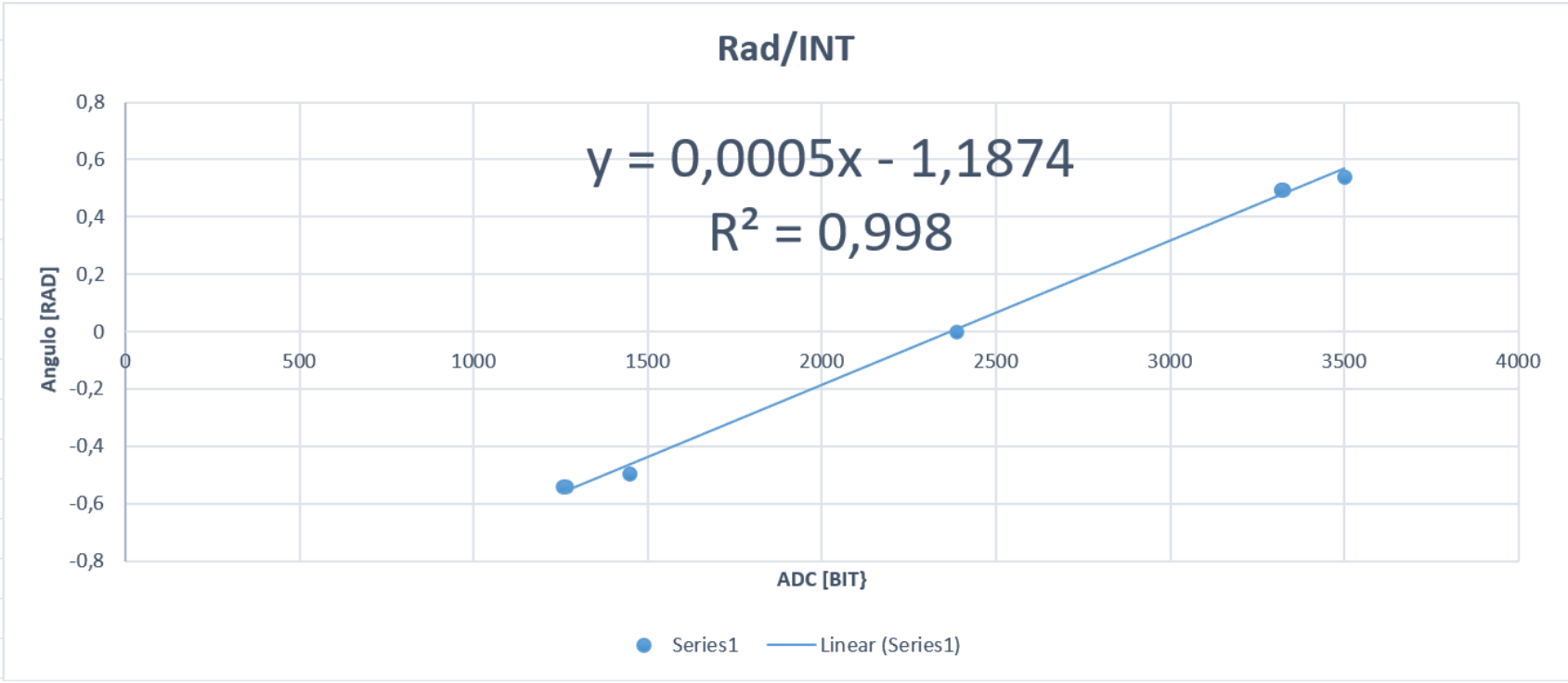
Figure: Resposta do sensor [V/RAD]



Fonte: Dados fornecidos "angulo.xlsx"

Caracterização do sensor II

Figure: Resposta do sensor [int/RAD]



Fonte: Dados fornecidos "angulo.xlsx"

Caracterização do sensor III

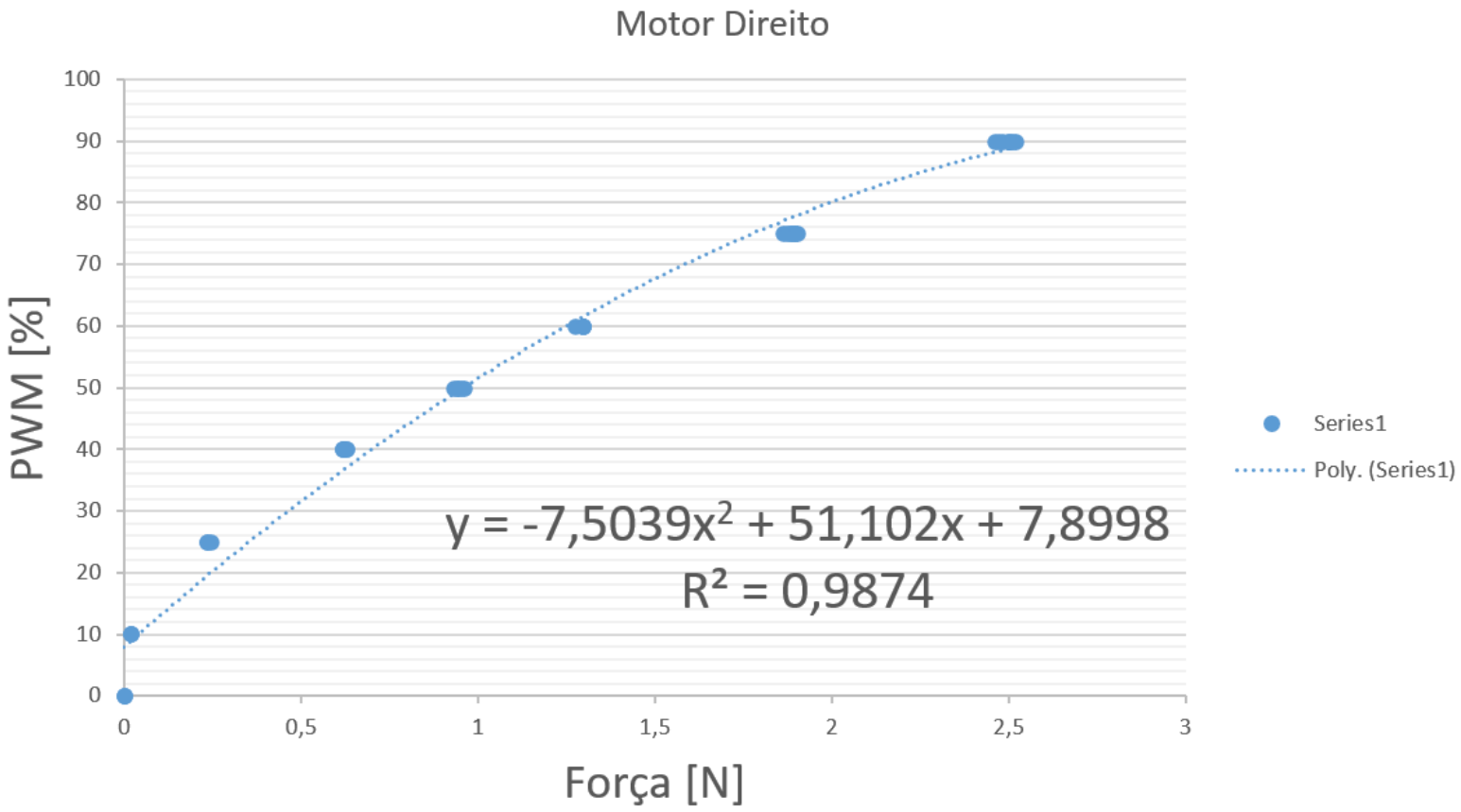
Figure: Função de conversão, valor do ADC para RAD

```
142 float ADC_to_rad(uint32_t ADC_value){
143     return 0.0005*ADC_value - 1.1874;
144 }
```

Fonte: Autoria própria

Caracterização dos atuadores I

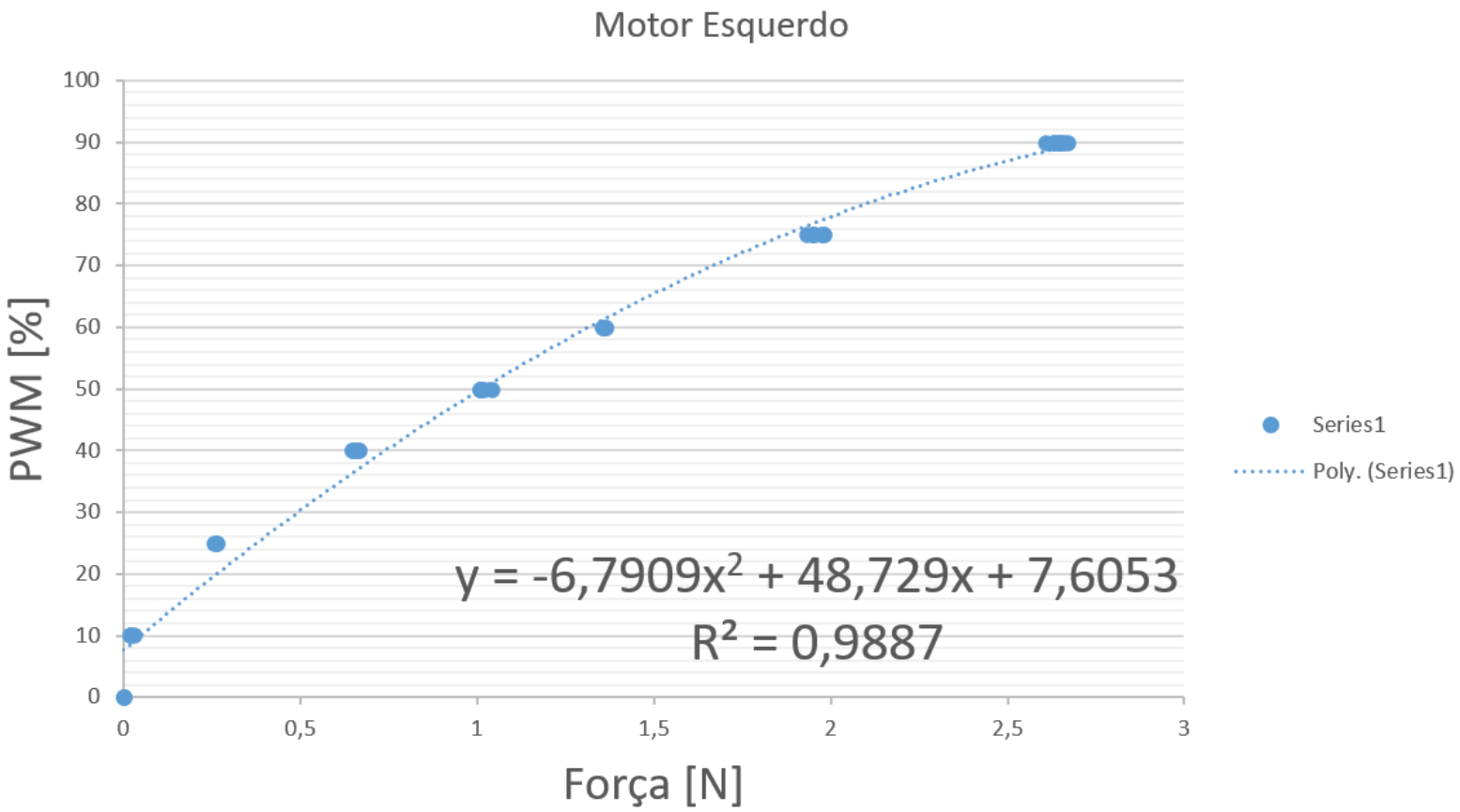
Figure: Caracterização do motor direito [F/PWM]



Fonte: Dados fornecidos "forca.xlsx"

Caracterização dos atuadores II

Figure: Caracterização do motor esquerdo [F/PWM]



Fonte: Dados fornecidos "forca.xlsx"

Caracterização dos atuadores III

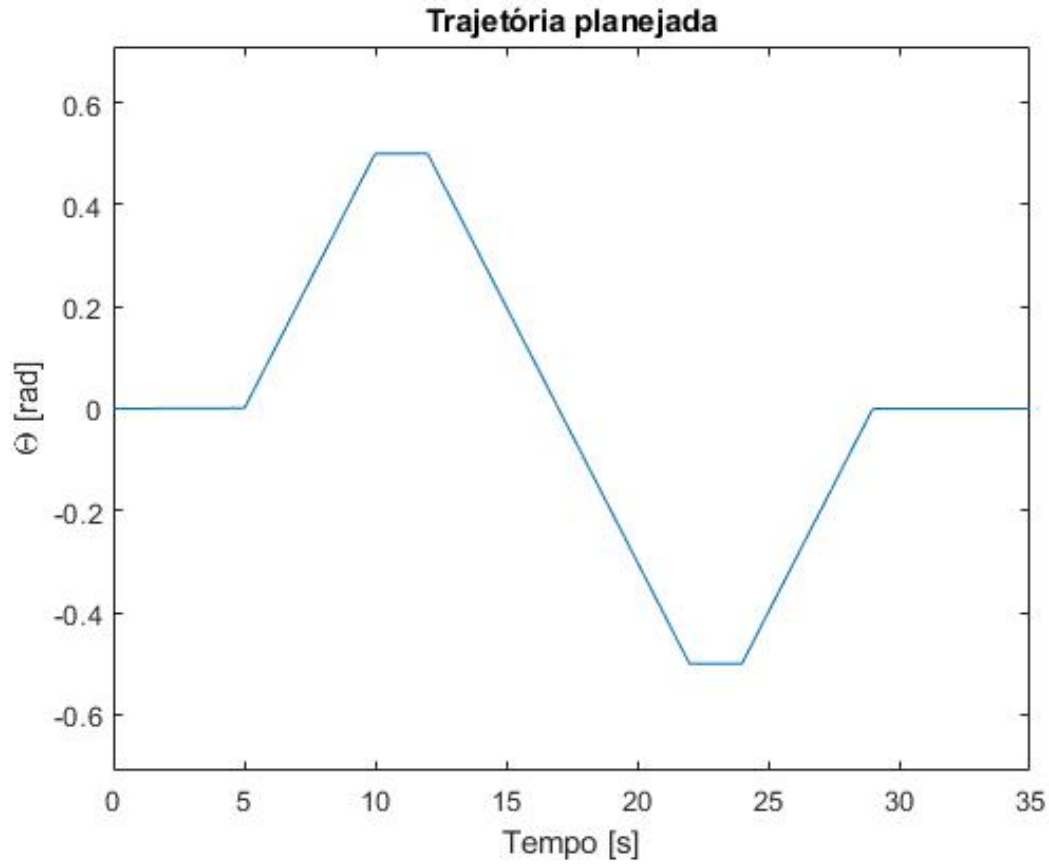
Figure: Funções de conversão, Força para PWM

```
153 float f_to_pwm_rigth(float f){
154     return -7.5039*pow(f,2) + 51.102*f + 7.8998;
155 }
156
157 float f_to_pwm_left(float f){
158     return -6.7909*pow(f,2) + 48.729*f + 7.6053;
159 }
```

Fonte: Autoria própria

Planejador de trajetória I

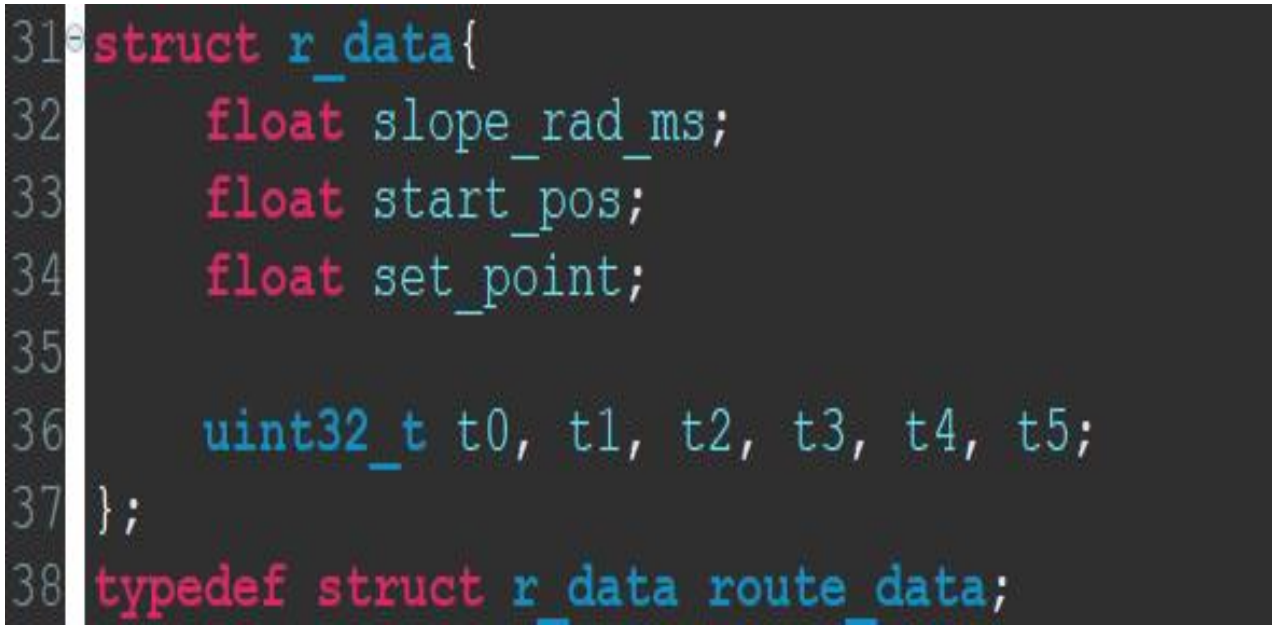
Figure: Trajetória planejada



Fonte: Autoria própria

Planejador de trajetória II

Figure: Estrutara "route_data"



```
31 struct r_data{
32     float slope_rad_ms;
33     float start_pos;
34     float set_point;
35
36     uint32_t t0, t1, t2, t3, t4, t5;
37 };
38 typedef struct r_data route_data;
```

Fonte: Autoria própria

Planejador de trajetória III

Figure: Função inicializadora da estrutura

```
161 void set_route_struct(route_data *d, float start_pos,
162         float set_point, uint32_t time_constant_rad_ms){
163 
164     float slope_rad_ms;
165 
166     if(set_point-start_pos >= 0.){
167         slope_rad_ms = MAX_SLOPE_RAD_MS;
168     }
169     else{
170         slope_rad_ms = -MAX_SLOPE_RAD_MS;
171     }
172 
173     d->slope_rad_ms = slope_rad_ms;
174     d->start_pos = start_pos;
175     d->set_point = set_point;
176 
177     d->t0 = HAL_GetTick();
178     d->t1 = (uint32_t) ((set_point-start_pos)/slope_rad_ms);
179     d->t2 = d->t1 + time_constant_rad_ms;
180     d->t3 = d->t2 + 2*d->t1;
181     d->t4 = d->t3 + time_constant_rad_ms;
182     d->t5 = d->t4 + d->t1;
183     return;
184 }
```

Fonte: Autoria própria

Planejador de trajetória IV

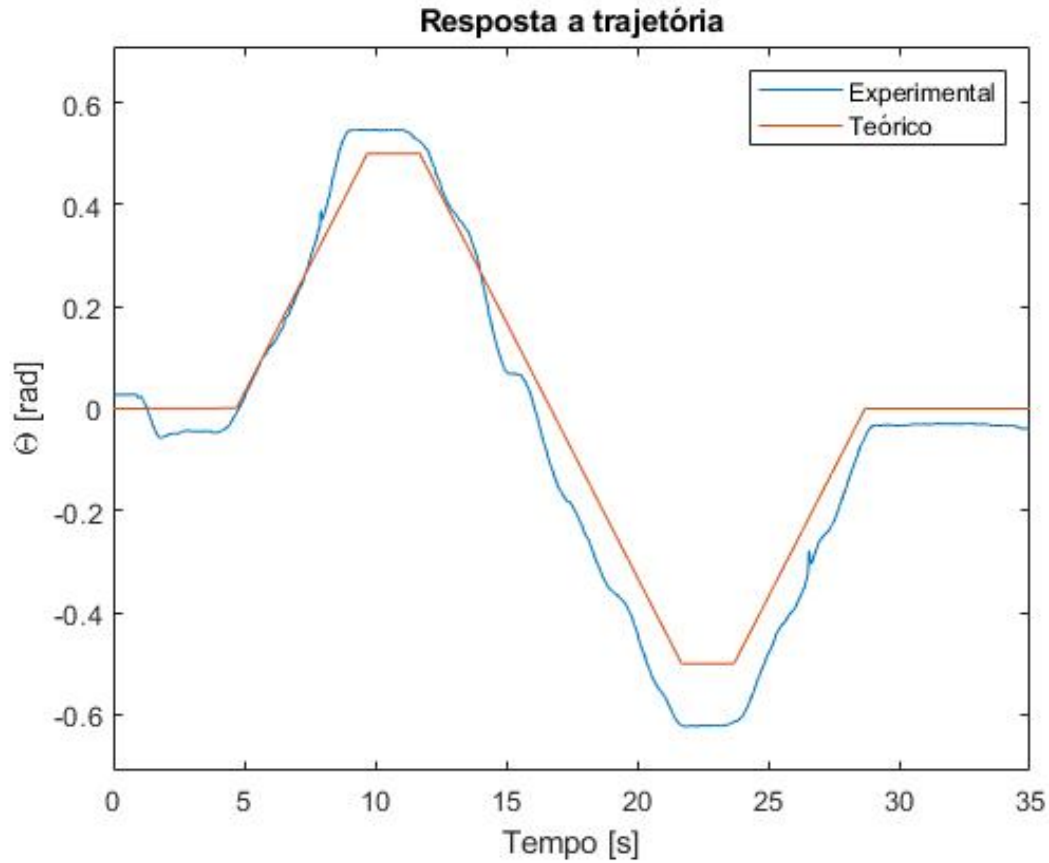
Figure: Função que implementa a rota

```
186 float route_planner(route_data *d){
187     uint32_t t = HAL_GetTick() - d->t0;
188     float relative_set_point = 0;
189 
190     if(t >= 0 && t < d->t1){
191         relative_set_point = d->start_pos + d->slope_rad_ms*t;
192     }
193     else if(t >= d->t1 && t <d->t2){
194         relative_set_point = d->set_point;
195     }
196     else if(t >= d->t2 && t < d->t3){
197         relative_set_point = d->set_point - d->slope_rad_ms*(t - d->t2);
198     }
199     else if (t >= d->t3 && t < d->t4){
200         relative_set_point = 2*d->start_pos - d->set_point;
201     }
202     else if (t >= d->t4 && t < d->t5){
203         relative_set_point = 2*d->start_pos - d->set_point
204                 + d->slope_rad_ms*(t - d->t4);
205     }
206     else{
207         relative_set_point = d->start_pos;
208     }
209 
210     return relative_set_point;
211 }
```

Fonte: Autoria própria

Planejador de trajetória V

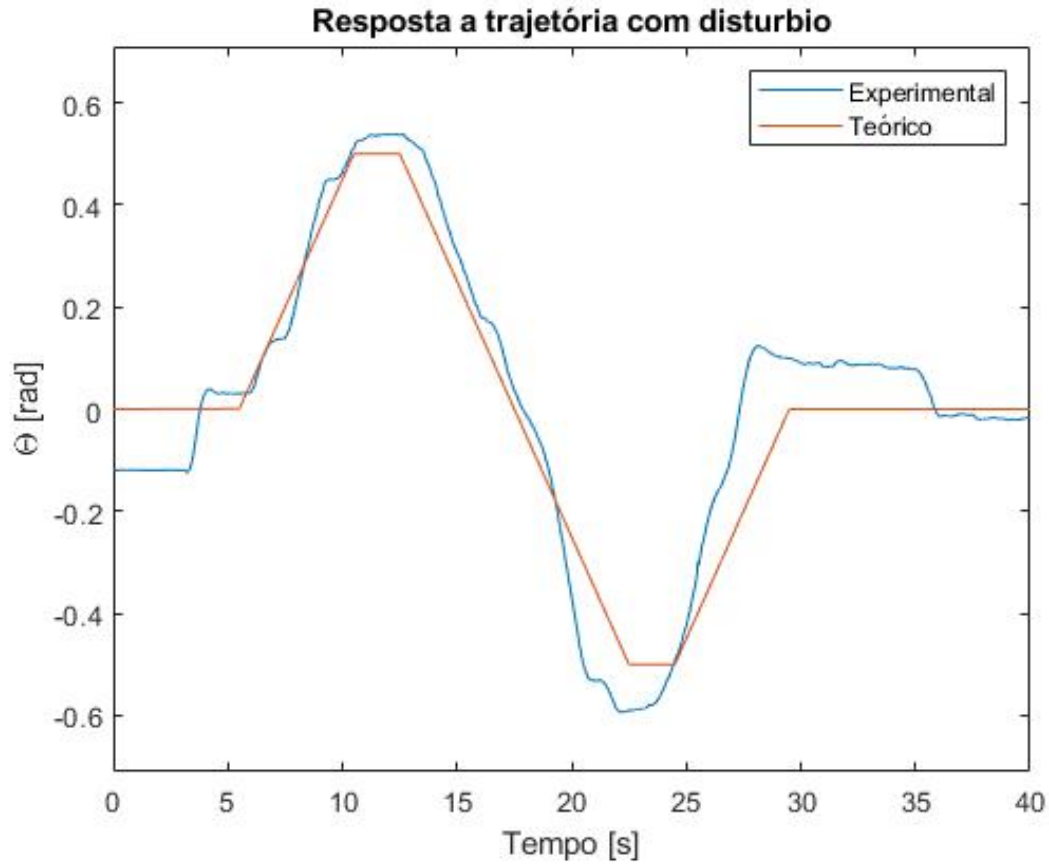
Figure: Comparação entre trajetória experimental e a planejada



Fonte: Autoria própria

Planejador de trajetória VI

Figure: Comparação entre trajetória experimental com distúrbio e a planejada



Fonte: Autoria própria

Melhorias/Discrepâncias

- Offset em regime permanente pode ser causado pelo sensor
- Kd foi muito reduzido por causa de vibrações no motor, o que pode ter comprometido a resposta a distúrbios
- Foi utilizado $\omega_{MAX} = 0.1$ [rad/s], seria necessários testes com câmeras para saber se é lento o suficiente