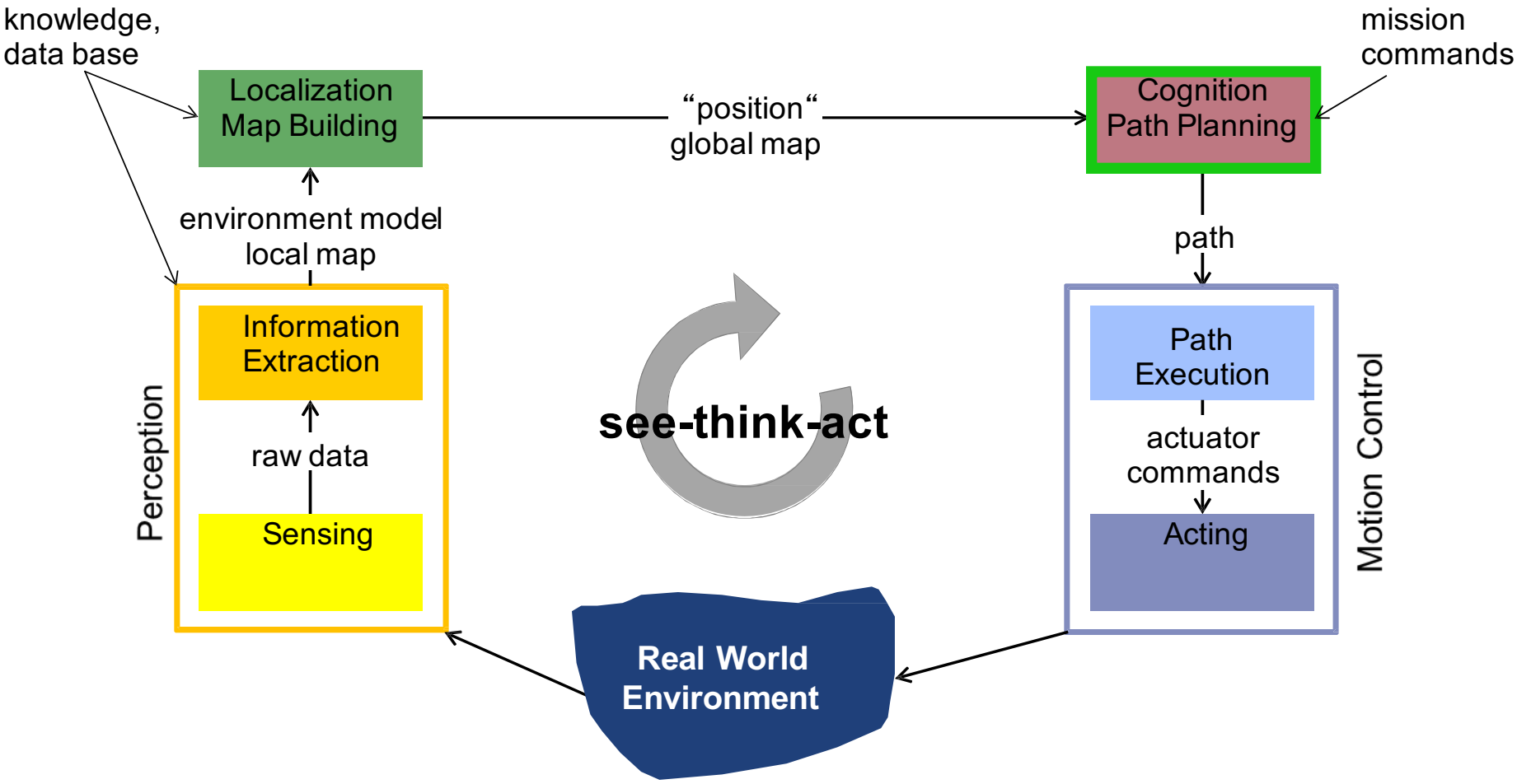
Universidade Tecnológica Federal do Paraná (UTFPR)
Disciplina: CPGEI/PPGCA - Robótica Móvel

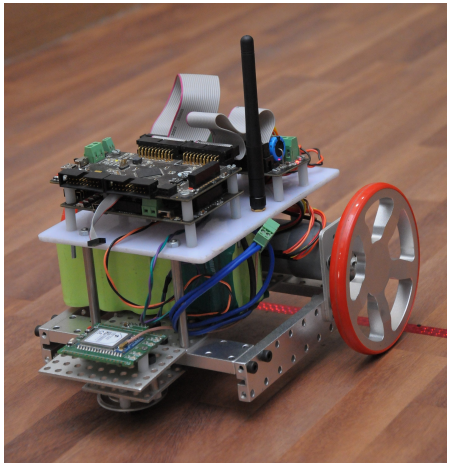
Movimentação, Localização, Odometria e Cinemática

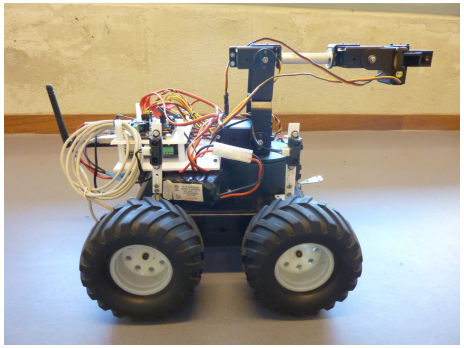
Prof. André Schneider de Oliveira
Prof. João Alberto Fabro

Ciclo "*ver-pensar-agir*"



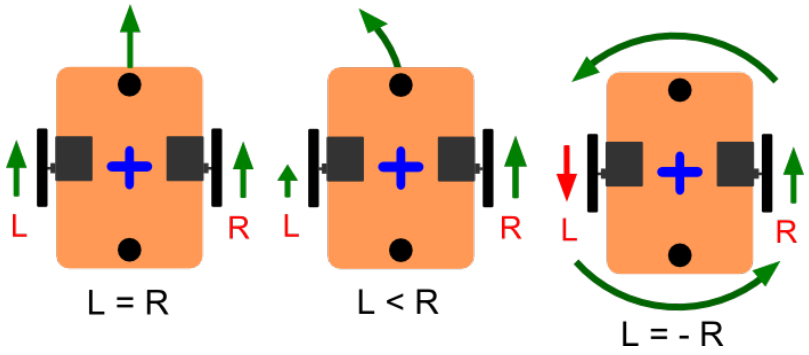
Robô diferencial

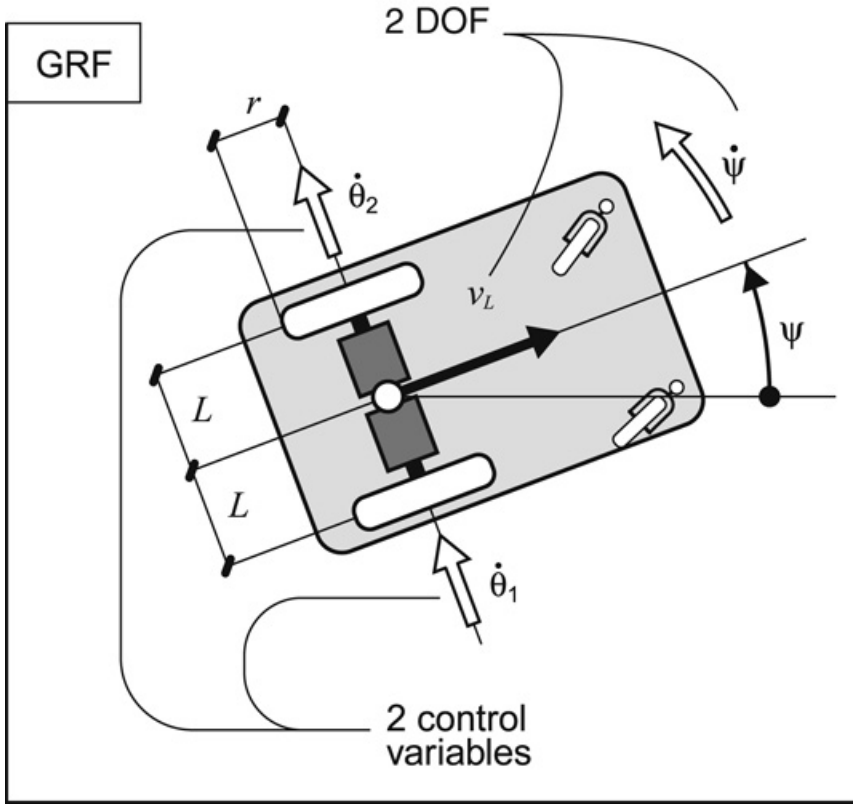




Robô diferencial

- É composto por um conjunto de rodas fixas e seu movimento é realizado pela diferença das velocidades dos atuadores
- **Como modelar isso?**





Cinemática

- É o estudo da descrição dos movimentos desprezando suas causas (forças/dinâmica).
- Na robótica móvel a cinemática pode ser empregada para realizar a análise de velocidade nos sistemas.
- Existem duas formas de análise cinemática:
  - **direta**: expressa como o movimento dos atuadores age sobre o robô
  - **inversa**: expressa como o movimento do robô age sobre os seus atuadores

Cinemática Inversa

A question that is of interest for a robot is: if we know the robot's instantaneous twist $\dot{\xi}$, what are the wheel speeds? Wheel velocity $\dot{\varphi}$, the derivative of the wheel angle φ, can be calculated by dividing the instantaneous forward velocity v of the wheel by the **wheel radius** r:

$$\dot{\varphi} = \frac{\mathrm{v}}{\mathrm{r}}$$

Hence, we need to figure out at what velocities the wheels will move, in response to a twist $\dot{\xi}$. For a differential drive robot, the answer is easy, if we reason about two cases separately:

- Just forward motion with velocity v_x: the wheel velocities are then

$$\dot{\varphi}_R = \frac{\mathrm{v_x}}{\mathrm{r}}$$

$$\dot{\varphi}_L = \frac{\mathrm{v_x}}{\mathrm{r}}$$

- Just angular velocity ω: the wheel velocities then depend on the **length L of wheel axis**:

$$\dot{\varphi}_R = \frac{\omega \mathrm{L}}{2\mathrm{r}}$$

$$\dot{\varphi}_L = -\frac{\omega L}{2r}$$

Cinemática Inversa

If we have both, we can simply add them:

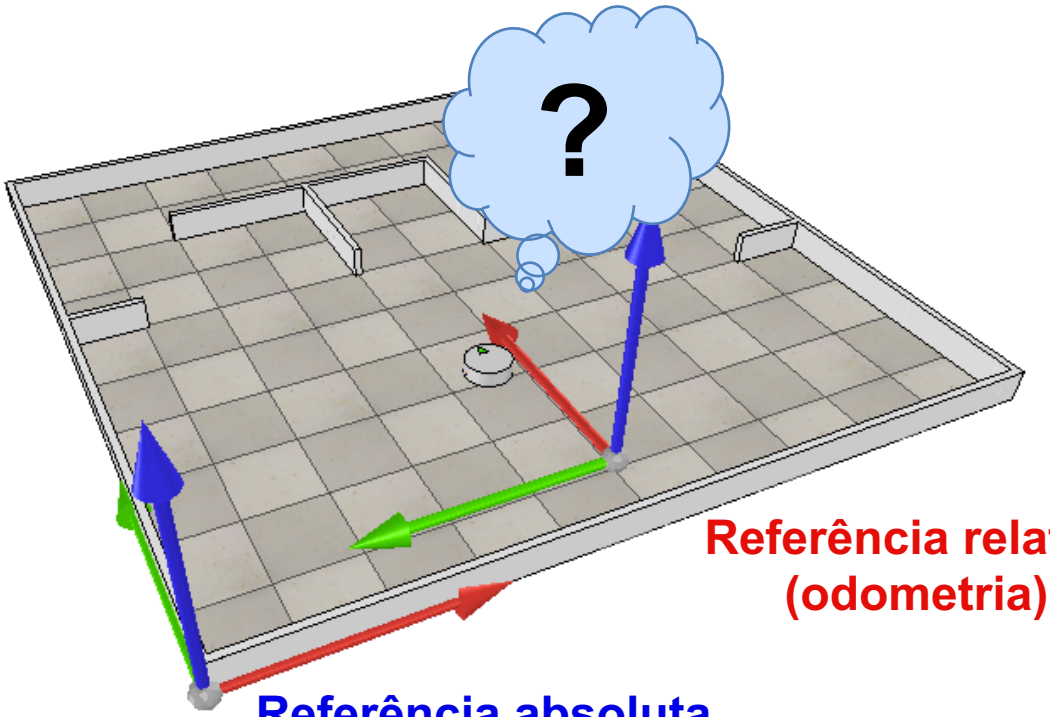
$$\dot{\varphi}_R = \frac{\omega \mathrm{L}}{2\mathrm{r}} + \frac{\mathrm{v_x}}{\mathrm{r}} \tag{1}$$

$$\dot{\varphi}_L = -\frac{\omega L}{2r} + \frac{\mathrm{v_x}}{\mathrm{r}} \tag{2}$$

This is called **inverse kinematics**, because it calculates the wheels speeds given a velocity, rather than the other way around.

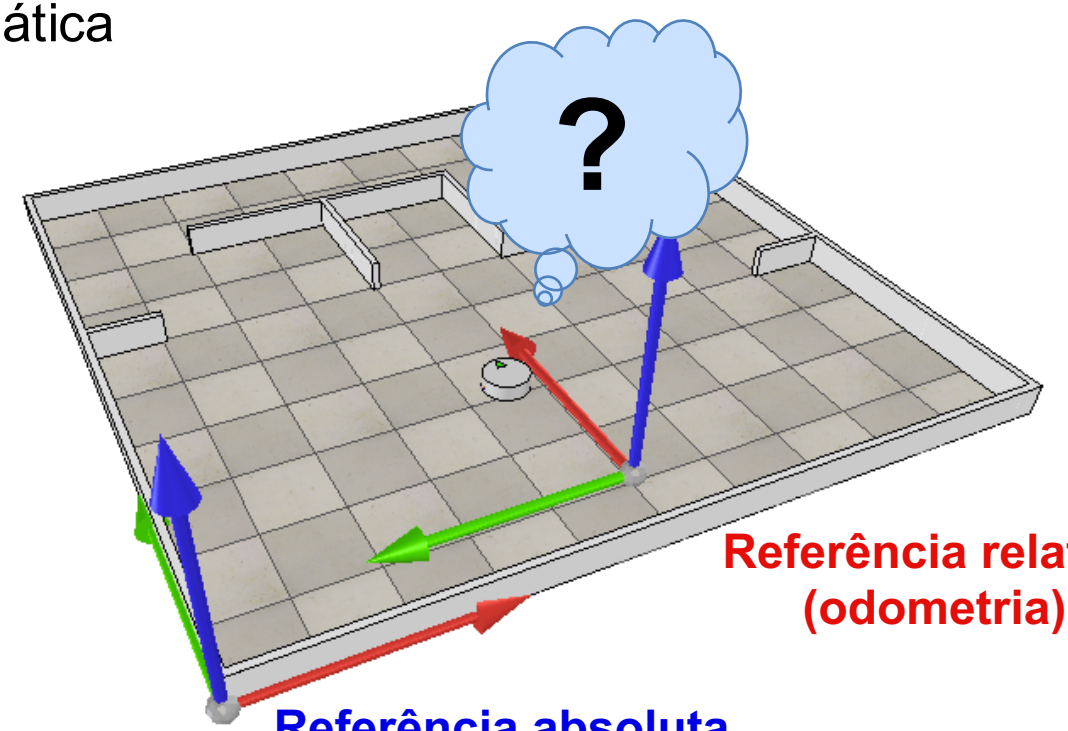
Localização e Odometria

- **Localização** – pose (posição e orientação) absoluta, em relação ao ambiente
  - Calculada com sensores do ambiente (gps, landmarks)



Localização e Odometria

- **Odometria** – pose relativa, em relação à sua pose inicial
  - Calculada com sensores do robô (encoder's, giroscópios, acelerômetros)
  - Integração da cinemática



Odometria no ROS

Localização simplificada no ROS

Type: turtlesim/Pose

float32 x
float32 y
float32 theta

float32 linear_velocity
float32 angular_velocity

Type: nav_msgs/Odometry

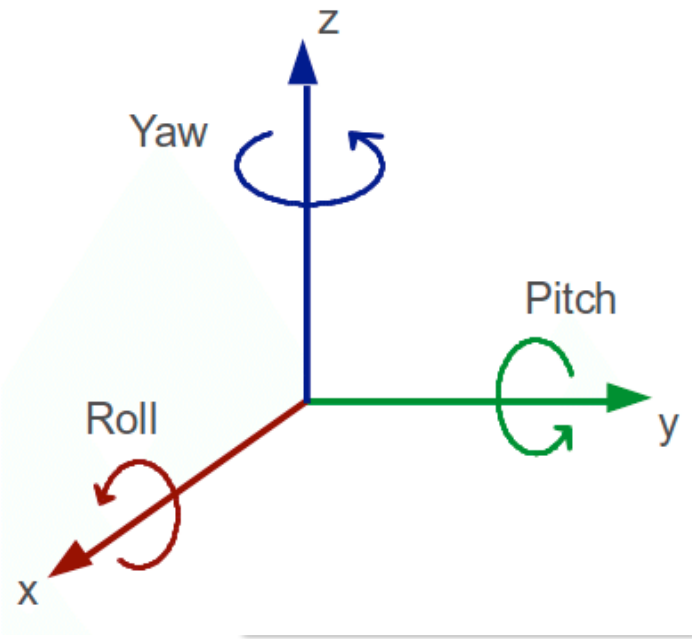
std_msgs/Header header
  uint32 seq
  time stamp
  string frame_id
string child_frame_id
geometry_msgs/PoseWithCovariance pose
  geometry_msgs/Pose pose
    geometry_msgs/Point position
      float64 x
      float64 y
      float64 z
    geometry_msgs/Quaternion orientation
      float64 x
      float64 y
      float64 z
      float64 w
  float64[36] covariance
geometry_msgs/TwistWithCovariance twist
  geometry_msgs/Twist twist
    geometry_msgs/Vector3 linear
      float64 x
      float64 y
      float64 z
    geometry_msgs/Vector3 angular
      float64 x
      float64 y
      float64 z
  float64[36] covariance

Representação da odometria

$$q = w + x\mathbf{i} + y\mathbf{j} + z\mathbf{k}$$

- ***w, x, y, z*** são números reais **(NÃO ANGULOS!)**
- ***w*** é a parcela escalar
- ***i,j,k*** é parcela imaginária

* É necessário decodificar para extrair os ângulos de orientação Roll-Pitch-**Yaw**




```
#include <tf/transform_datatypes.h>
// ...
tf::Quaternion q(quat.x, quat.y, quat.z, quat.w);
tf::Matrix3x3 m(q);
double roll, pitch, yaw;
m.getRPY(roll, pitch, yaw);
std::cout << "Roll: " << roll << ", Pitch: " << pitch << ", Yaw: " << yaw << std::endl;
```

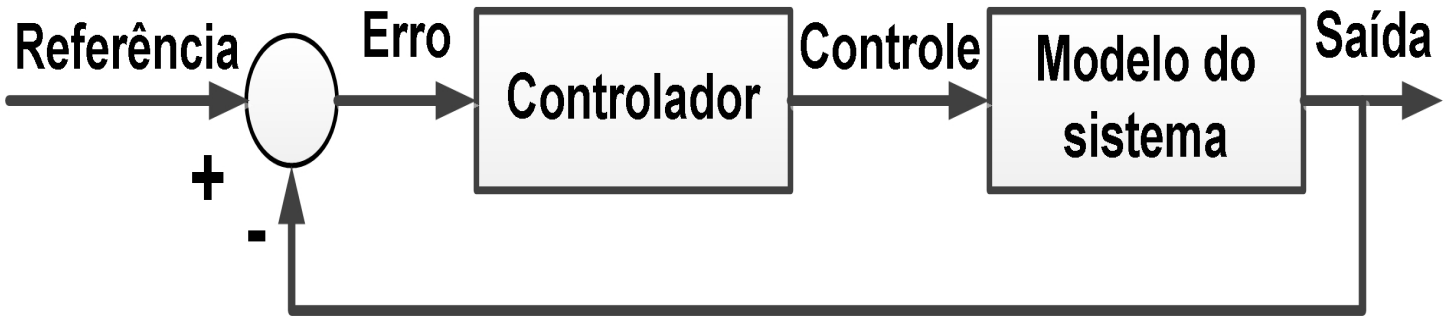
Movimentação

- Conceitos básicos de controle
- Estratégias P, PD, PI e PID
- Realimentação baseada na odometria/localização
- Implementação de controladores com ROS

Controle

- As estratégias de controle automático são algoritmos para o controle de uma determinada grandeza em um processo

- O controlador é o elemento do sistema que aplica o sinal de erro para corrigir a atuação do sistema



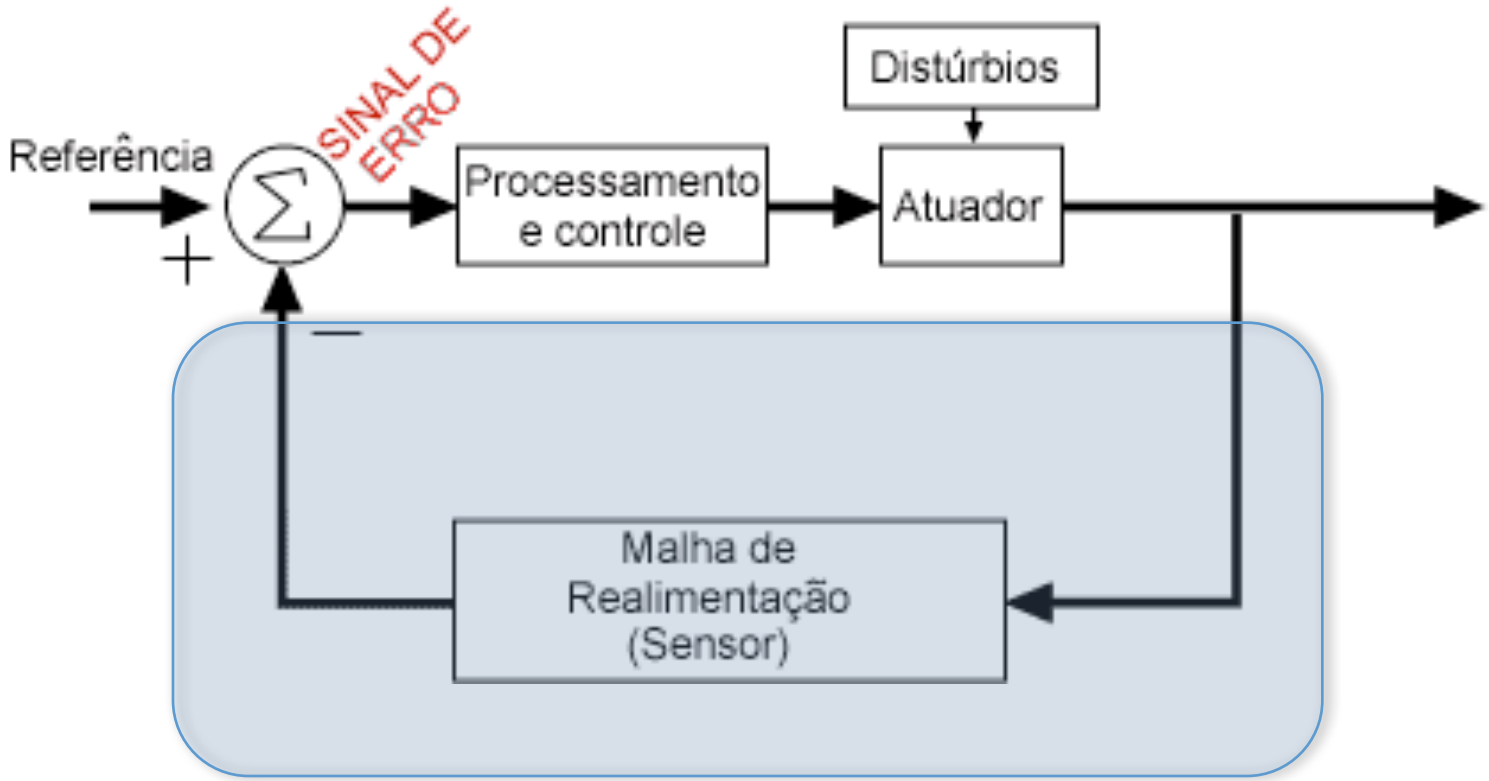
Definição

- Sistema de controle deve alcançar e estabilizar sua resposta no valor desejado

- A precisão está relacionada com o erro do sistema (diferença entre o desejado e obtido) após estabilizar sua resposta

Estrutura básica de controle

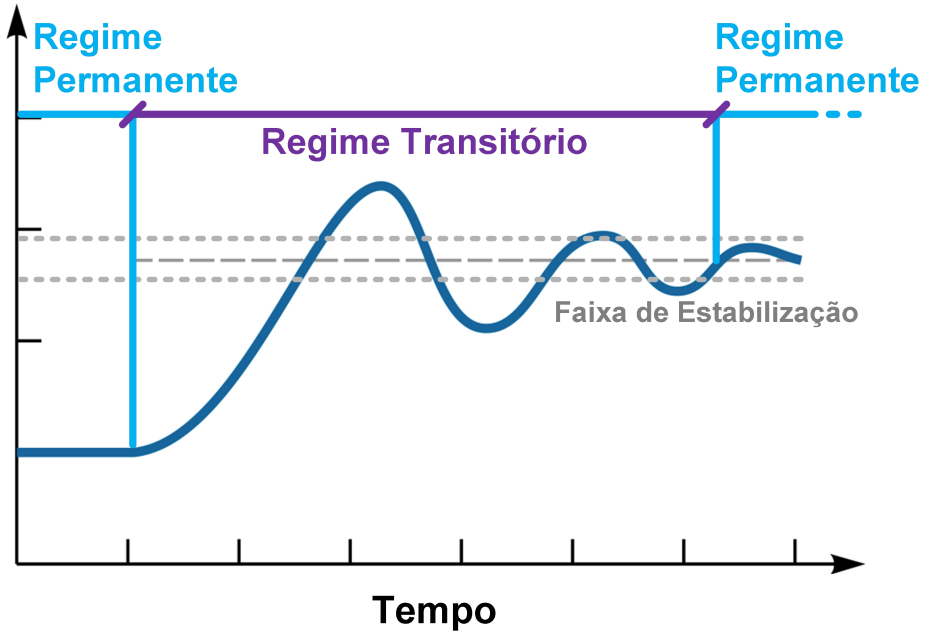
- Malha Fechada – com realimentação



Estágios da resposta

- O período entre o estímulo de entrada e a estabilização da resposta do sistema é denominado de resposta em **regime transitório**

- O período após a transição da resposta do sistema, onde, o sistema estabiliza em um determinado valor é denominado de resposta em **regime permanente**



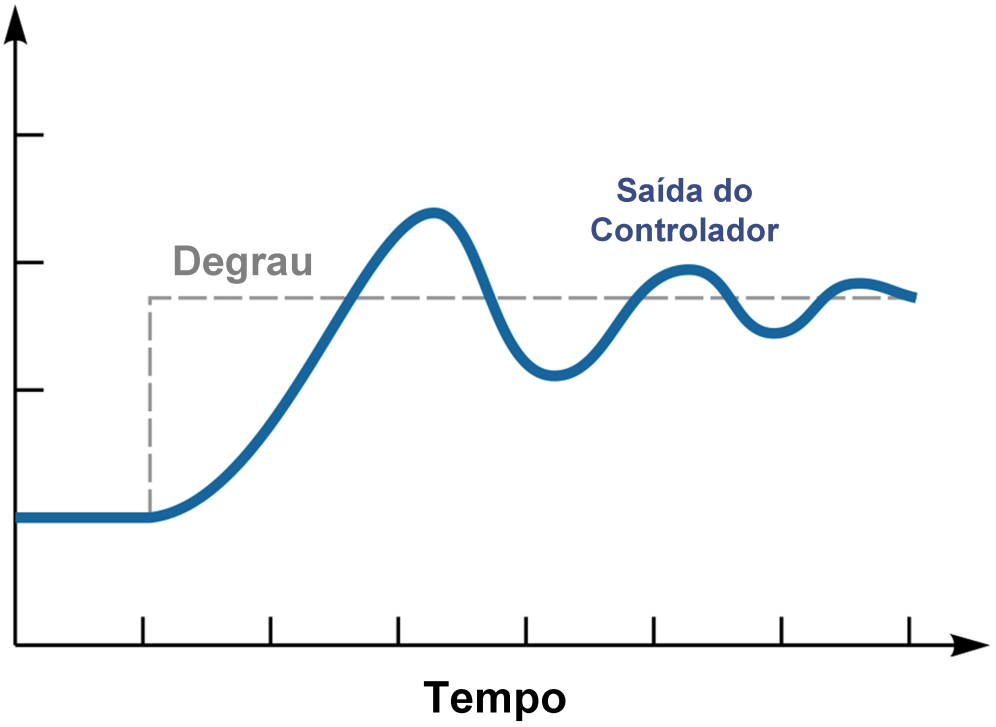
Estímulo degrau

- As estratégias de controle modificam suas ações corretivas de acordo com os estímulos de entrada

- Na análise da resposta de sistema de controle tradicionalmente é realizado um estímulo de degrau na estrada do controlador para analisar a saída

Estímulo degrau

- O degrau é uma transição instantânea do sinal de referencia do regime permanente inicial (geralmente nulo) para o valor desejado



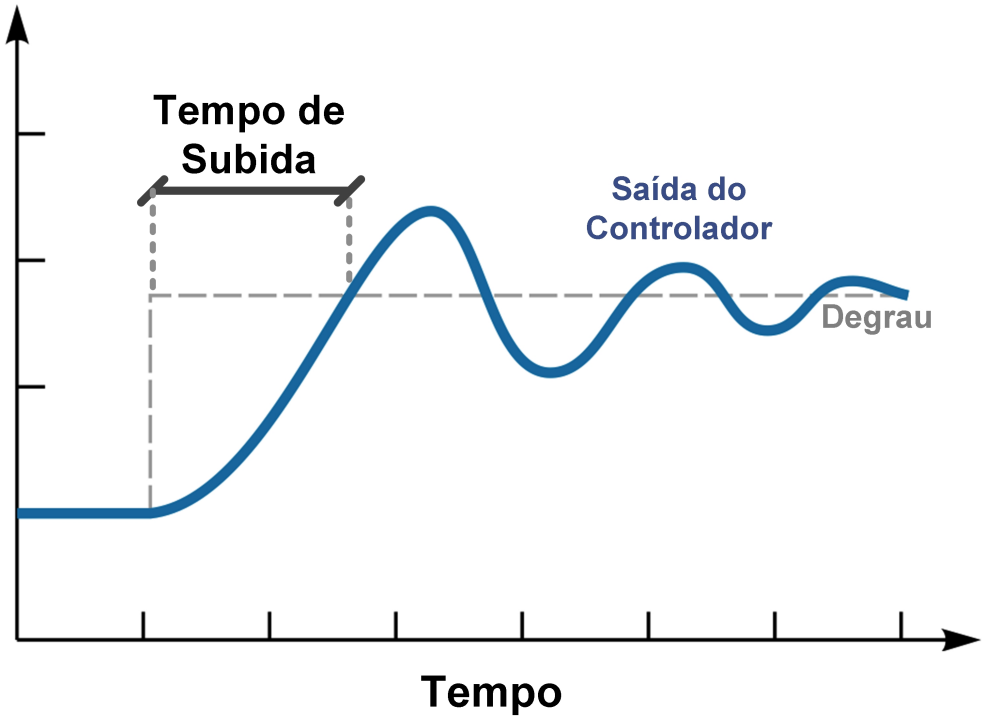
Tempo de subida

- Tempo de subida (*rise time*) é o tempo gasto para que a resposta do sistema a um degrau vá de seu valor (normalmente zero) até o valor final

- Tempo gasto para a resposta aumentar de algum percentual indicado (por exemplo 20%) para outra porcentagem especificada (por exemplo 70%)

Tempo de subida

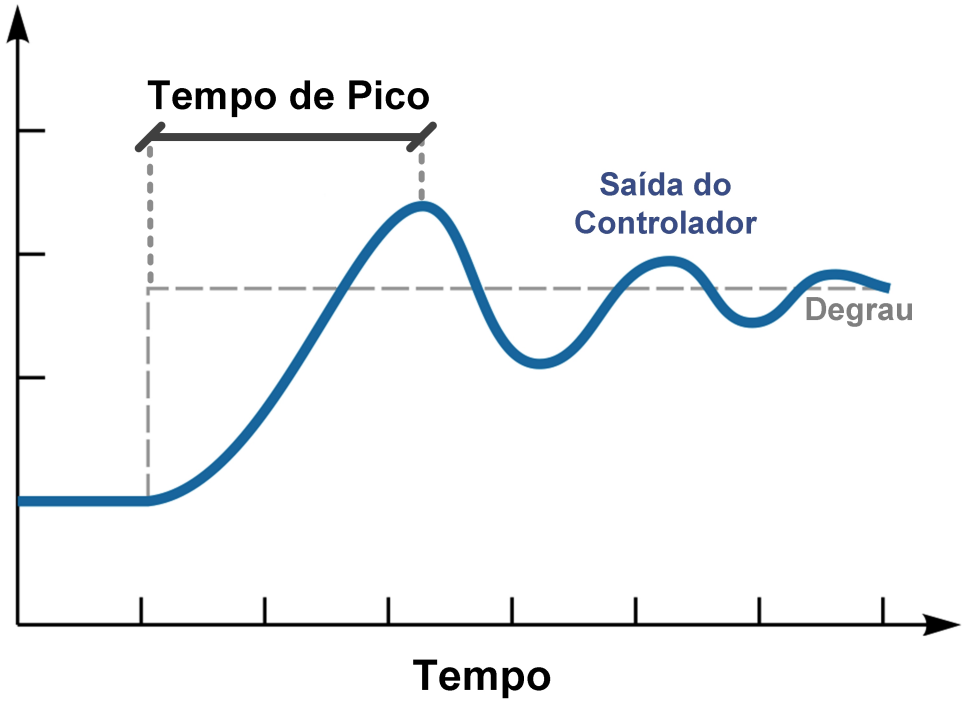
- Quanto menor for o valor do tempo de subida menor será o atraso do sistema para responder ao estímulo do degrau



Tempo de pico

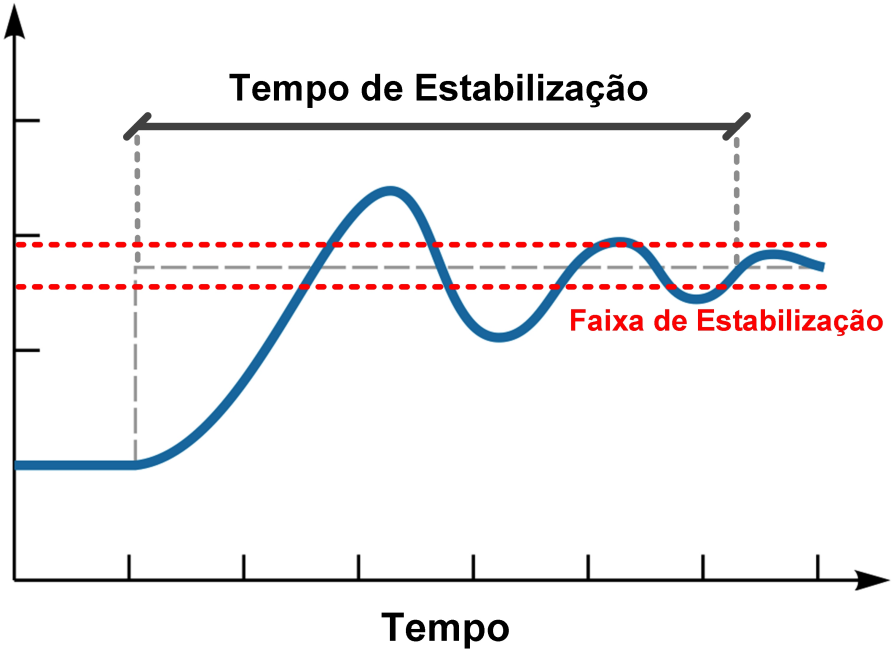
- O tempo de pico (*peak time*) é o tempo gasto para a resposta ir do valor nulo até o primeiro valor de pico.

- Tempo necessário para a resposta do sistema sair do repouso e ir para o primeiro pico.



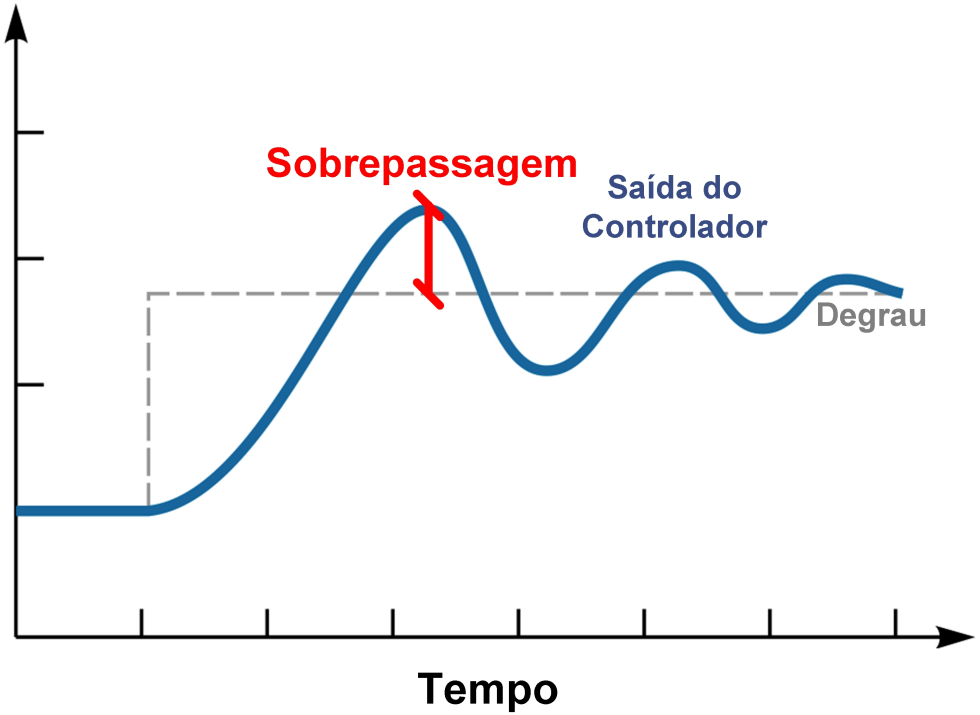
Tempo de estabilização

- Intervalo para que as oscilações na resposta do sistema desapareçam
- Tempo necessário para a resposta diminuir e permanecer dentro de um percentual desejado, como por exemplo, 1% do valor final



Sobre-passagem

- Sobre-passagem (sobre-sinal ou *overshoot*) é a quantidade máxima em que a resposta ultrapassa o valor de referencia
- Ou simplesmente, é quanto a resposta do sistema ultrapassa o valor desejado



Estabilidade

- A principal forma de determinar o desempenho de um sistema de controle é a estabilidade do sistema.

- A estabilidade de um sistema de controle está associado com a capacidade desse sistema gerar uma resposta única invariante no tempo.

- É possível definir o conceito de estabilidade como:

“Um sistema de controle é estável, se e somente se, um sinal de entrada de amplitude limitada promove uma resposta com sinal saída também de amplitude limitada”.

Instabilidade

- O termo instabilidade, está relacionado com geração de uma saída que tende a infinito, ou seja, variante no tempo, e pode ser definido como:

“Um sistema de controle é instável, se a resposta do sistema é variante no tempo e, naturalmente, tende ao infinito.“

- A estabilidade de um sistema de controle não é uma medida única, ou seja, um sistema não é unicamente estável ou instável

Controle Proporcional

- O controle proporcional utiliza a saída do controlador para realizar uma realimentação proporcional em sua entrada

- O sinal de erro é utilizado para corrigir a saída do controlador onde um ganho proporcional é empregado para "agilizar" a correção do sistema, assim:

$$Saída = K_p . erro$$

- Onde *Kp* é o ganho proporcional e a saída do controlador depende apenas da amplitude (tamanho) do erro

Controle Proporcional

- O controlador é apenas um amplificador com um ganho constante
- Um grande erro em algum instante de tempo pode levar a um alto valor na saída do controlador
- Pode ocorrer a sobrepassagem (overshoot), por isso o ganho proporcional deve ser ajustado com cautela

Controle Proporcional-Derivativo

- As dificuldades de ajuste do controlador proporcional em relação ao atraso e a sobrepassagem podem ser reduzidos com a inclusão de uma correção derivativa

- O controle derivativo assim que exista um sinal de erro (erro diferente de zero) a saída do controlador pode possuir um grande grau de correção porque a ação do controlador é relativa a variação do erro no tempo e não ao seu valor absoluto

$$Saída = K_d . \frac{\Delta erro}{\Delta t}$$

Controle Proporcional-Integral

- Ação proporcional juntamente com a derivativa acarreta comumente na ocorrência de erros quando existe uma baixa variação de erro

- A ação corretiva integral que regula a saída do controlador de forma regular

- A parcela integral delimita que a saída do controlador em qualquer instante de tempo é proporcional ao acúmulo dos efeitos do erro em instantes anteriores

Controle Proporcional-Integral-Derivativo (PID)

- O controlador proporcional, integral e derivativo (*PID*) é uma associação clássica de três controladores, combinando as vantagens desses

- A ação integral está diretamente ligada à precisão do sistema, sendo responsável pelo erro nulo em regime permanente

- O efeito desestabilizador do termo integrativo é contrabalançado pela ação derivativa que tende a aumentar a estabilidade relativa do sistema, ao mesmo tempo em que torna a resposta do sistema mais rápida, devido ao seu efeito antecipatório

Controle Proporcional-Integral-Derivativo (PID)

- A parcela proporcional (*P*) do controle realiza uma multiplicação do sinal de correção (realimentação) por um ganho proporcional.
- Isso influência o controlador a gerar um sinal de resposta em função da magnitude do erro.
- Dessa forma, se o sinal de erro for amplo, o termo proporcional causará uma correção de mesma dimensão.
- O efeito da parcela proporcional reduz o erro à aproximadamente zero.

Controle Proporcional-Integral-Derivativo (PID)

- Em vários sistemas, o erro aproxima-se de zero, mas não converge para esse valor, o que resulta no erro em regime permanente
- O termo integral (*I*) é introduzido para corrigir pequenos erros em regime permanente.
- Um pequeno erro nessas condições pode acarretar em um grande erro acumulado no tempo.
- A saída do controlador é gerada pela multiplicação desse erro acumulado pelo fator de ganho integral.

Controle Proporcional-Integral-Derivativo (PID)

- O termo diferencial (*D*) aumenta a velocidade de resposta do controlador e opera sobre a variação do sinal de erro
- A multiplicação dessa variação pelo fator de ganho derivativo gera a saída do controlador.
- Nem todos os controladores possuem o termo derivativo, é menos comum ainda a utilização da parcela integral.
- Isso ocorre porque certos efeitos tornam-se desprezíveis devido ao comportamento do sistema.

Implementação do PID

$$\text{Output} = K_p e(t) + K_I \int e(t)\,dt + K_D \frac{d}{dt} e(t)$$

$$\text{Where : } e = \text{Setpoint - Input}$$

```
1  /*working variables*/
2  unsigned long lastTime;
3  double Input, Output, Setpoint;
4  double errSum, lastErr;
5  double kp, ki, kd;
6  void Compute()
7  {
8     /*How long since we last calculated*/
9     unsigned long now = millis();
10    double timeChange = (double)(now - lastTime);
11
12    /*Compute all the working error variables*/
13    double error = Setpoint - Input;
14    errSum += (error * timeChange);
15    double dErr = (error - lastErr) / timeChange;
16
17    /*Compute PID Output*/
18    Output = kp * error + ki * errSum + kd * dErr;
19
20    /*Remember some variables for next time*/
21    lastErr = error;
22    lastTime = now;
23 }
```

ganhos

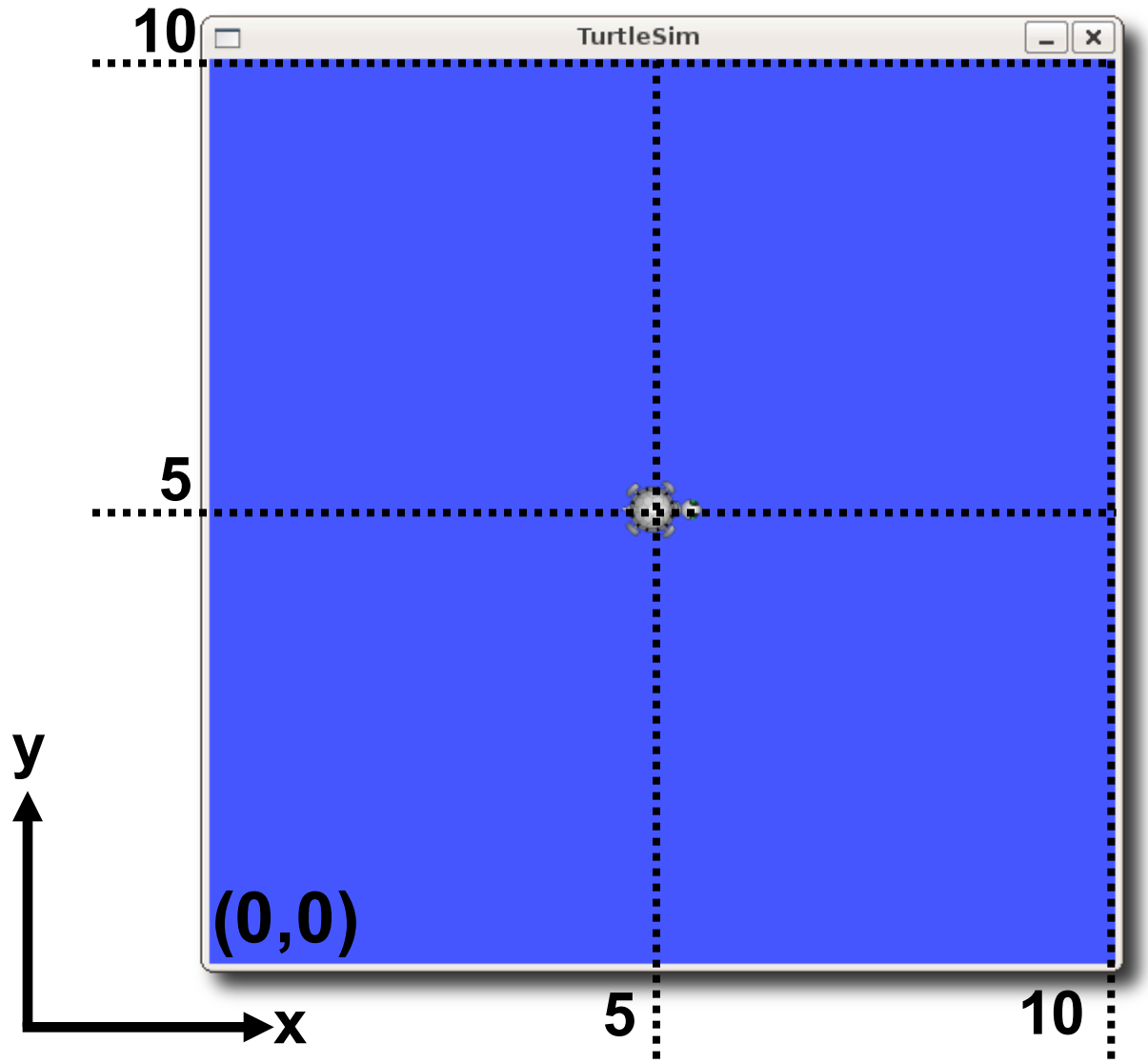
variação de tempo

Erro proporcional

Erro integral

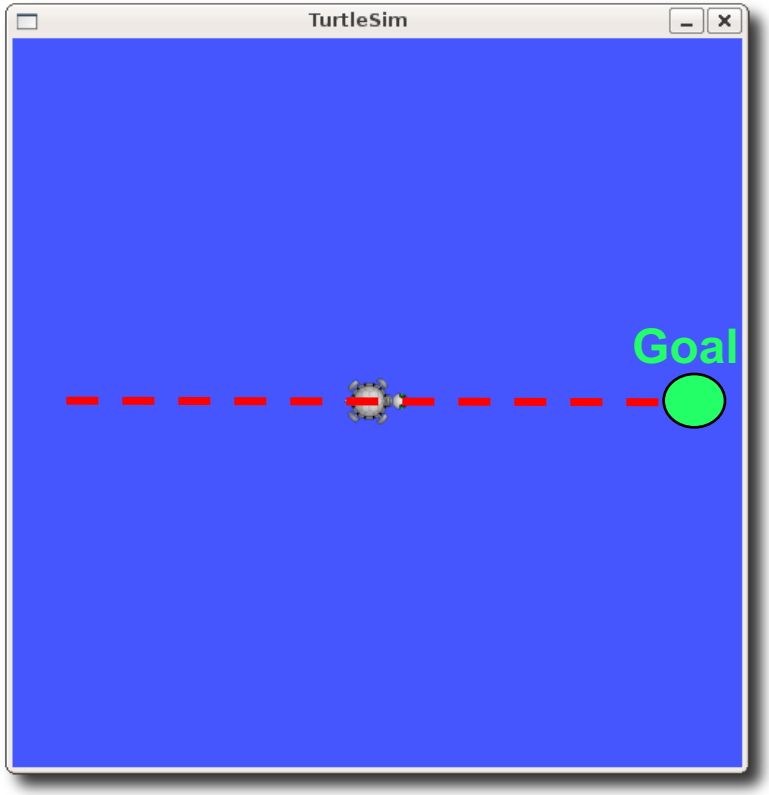
Erro derivativo

Ambiente *TurtleSim*



Controle de posição em X

- **Objetivo:** realizar o controle do movimento do robô turtle ao longo do eixo X para atingir uma determinada posição (Goal)



Controle de posição em X

```
#include "ros/ros.h"
#include "std_msgs/String.h"
#include "geometry_msgs/Twist.h"
#include "turtlesim/Pose.h"
#include <iostream>
using namespace std;

turtlesim::Pose feedback;

void subCallback(const turtlesim::Pose::ConstPtr& msg) {
        feedback.x = msg->x;
        feedback.y = msg->y;
        feedback.theta = msg->theta;
        feedback.linear_velocity = msg->linear_velocity;
        feedback.angular_velocity = msg->angular_velocity;
}
```

Controle de posição em X

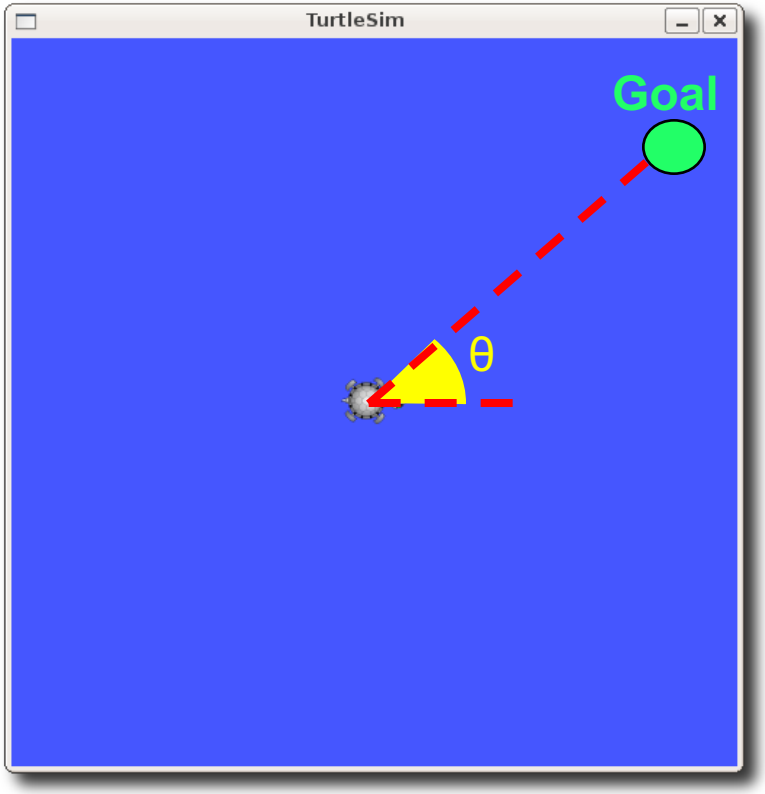
```
int main(int argc, char **argv) {
        ros::init(argc, argv, "turtle_controle_X");
        ros::NodeHandle n;
        ros::Publisher pub = n.advertise<geometry_msgs::Twist>("turtle1/cmd_vel", 1000);
        ros::Subscriber sub = n.subscribe("turtle1/pose", 1000, subCallback);
        ros::Rate loop_rate(10);
        system("rosservice call reset");

        if (ros::ok()) {
                geometry_msgs::Twist msg;
                float desejado, erro=99;
                float tolerance = 0.01;
                float Kpos = 10; cout << "Digite a posicao\nX>>";
                cin >> desejado;
```

Controle de posição em X

```
            while (abs(erro) > tolerance) {
                        erro = desejado-feedback.x;
                        msg.linear.x = Kpos*erro/(60/10);
                        ROS_INFO("X>>%f,Erro>>%f",feedback.x,erro);
                        pub.publish(msg);
                        ros::spinOnce();
                        loop_rate.sleep();
            }
            ROS_WARN("...Posicao alcancada...");
        }
return 0;
}
```

Controle de posição em X e Y

- **Objetivo:** realizar o controle do movimento do robô turtle ao longo dos eixos X e Y para atingir uma determinada posição (Goal)



Controle de posição em X e Y

```
#include "ros/ros.h"
#include "std_msgs/String.h"
#include "geometry_msgs/Twist.h"
#include "turtlesim/Pose.h"
#include <iostream>
#include <math.h>
using namespace std;

turtlesim::Pose feedback;

void subCallback(const turtlesim::Pose::ConstPtr& msg) {
        feedback.x = msg->x;
        feedback.y = msg->y;
        feedback.theta = msg->theta;
}
```

Controle de posição em X e Y

```
//Controle de posicao
            while (dist > tolerance_pos){
                        dist = sqrt(pow(posdesejada[0]-feedback.x,2)+pow(posdesejada[1]-feedback.y,2));
                         msg.linear.x = abs(Kpos*(dist) /(60/10));
                         ROS_INFO("Dist>>%f",dist);
                        pub.publish(msg);
                        ros::spinOnce();
                        loop_rate.sleep();
            }
            ROS_WARN("...Posicao alcancada...");
      }
return 0;
}
```

Controle de posição em X e Y

```
// Controle da orientacao
        while (abs(erroorie) > tolerance_orie) {
                erroorie = angulo-feedback.theta;
                msg.angular.z = Korie*erroorie/(60/10);
                ROS_INFO("theta>>%f,Erro>>%f",feedback.theta,erroorie);
                pub.publish(msg);
                ros::spinOnce();
                loop_rate.sleep();
        }
        msg.angular.z = 0;
        pub.publish(msg);
        ros::spinOnce();
        ROS_WARN("...Orientacao alcancada...");
```

Controle de posição em X e Y

```
int main(int argc, char **argv) {
        ros::init(argc, argv, "turtle_controle_posicao");
        ros::NodeHandle n;
        ros::Publisher pub = n.advertise<geometry_msgs::Twist>("turtle1/cmd_vel", 1000);
        ros::Subscriber sub = n.subscribe("turtle1/pose", 1000, subCallback);
        ros::Rate loop_rate(10);

        if (ros::ok()) {
                geometry_msgs::Twist msg;
                float posdesejada[2], oridesejada, dist=99, erroorie=99;
                float tolerance_orie = 0.005, tolerance_pos = 0.05;
                float Kpos = 10, Korie = 15;
                float angulo;

                cout << "Digite a posicao\nX>>";
                cin >> posdesejada[0];
                cout << "Y>>"; cin >> posdesejada[1];
                ros::spinOnce();
                angulo = atan2(posdesejada[1]-feedback.y,posdesejada[0]-feedback.x);
                ROS_WARN("angulo>>%f\n",angulo);
```