O Navigator do IntelliBrain

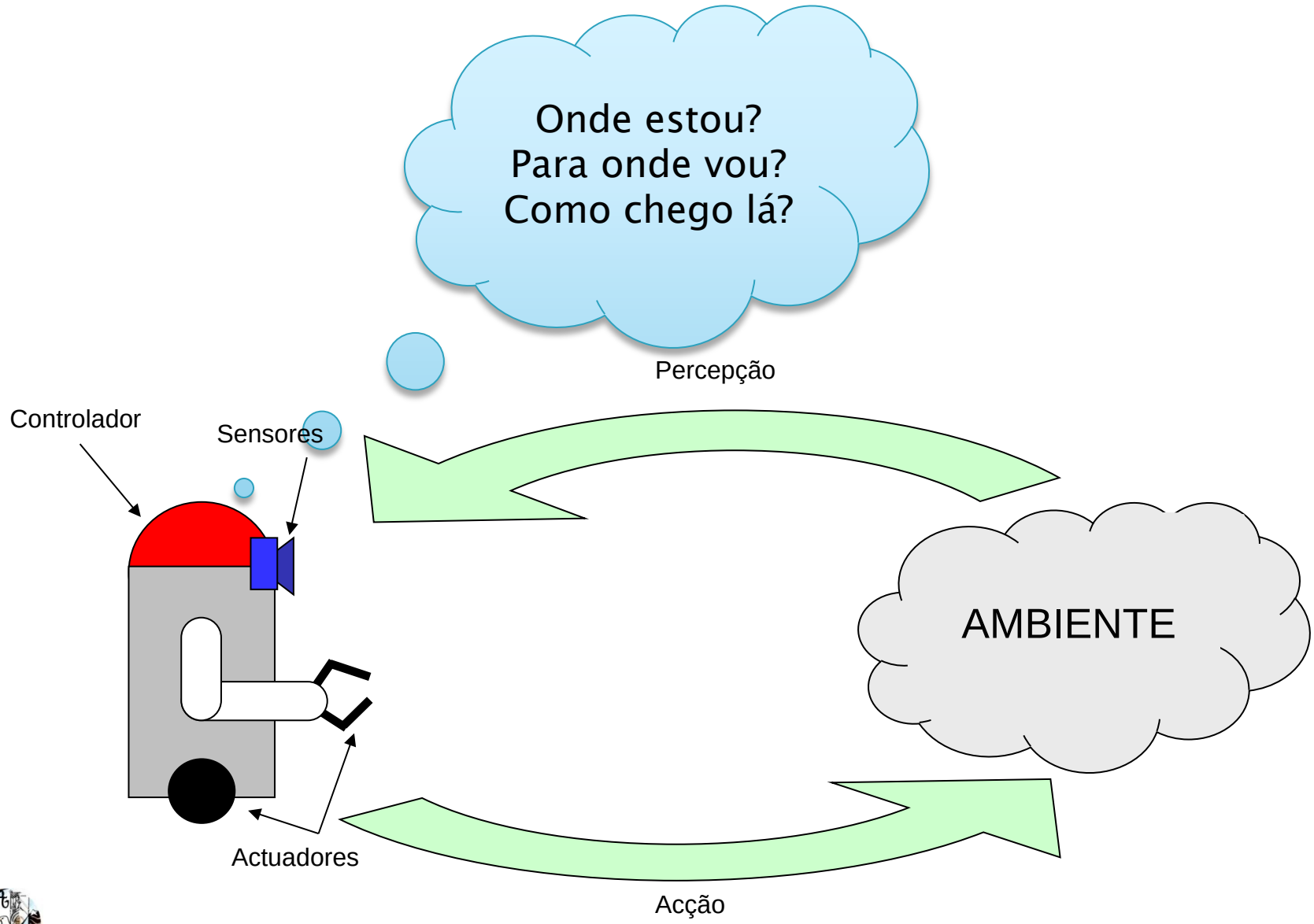
Robótica

Prof. Carlos Carreto

2011/2012

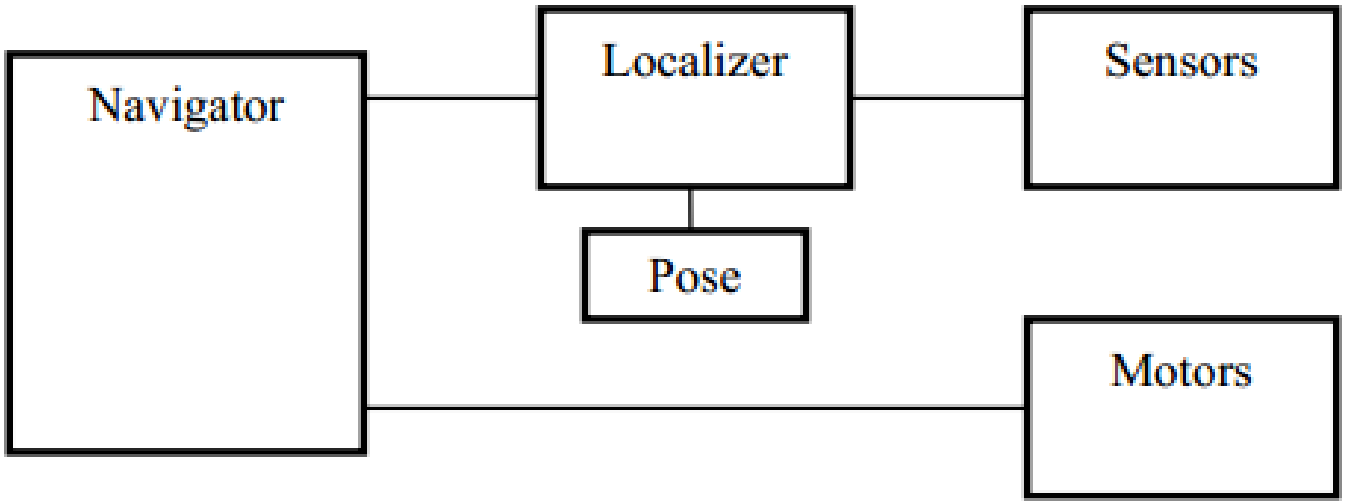
Engenharia Informática
Instituto Politécnico da Guarda

O Problema da Navegação

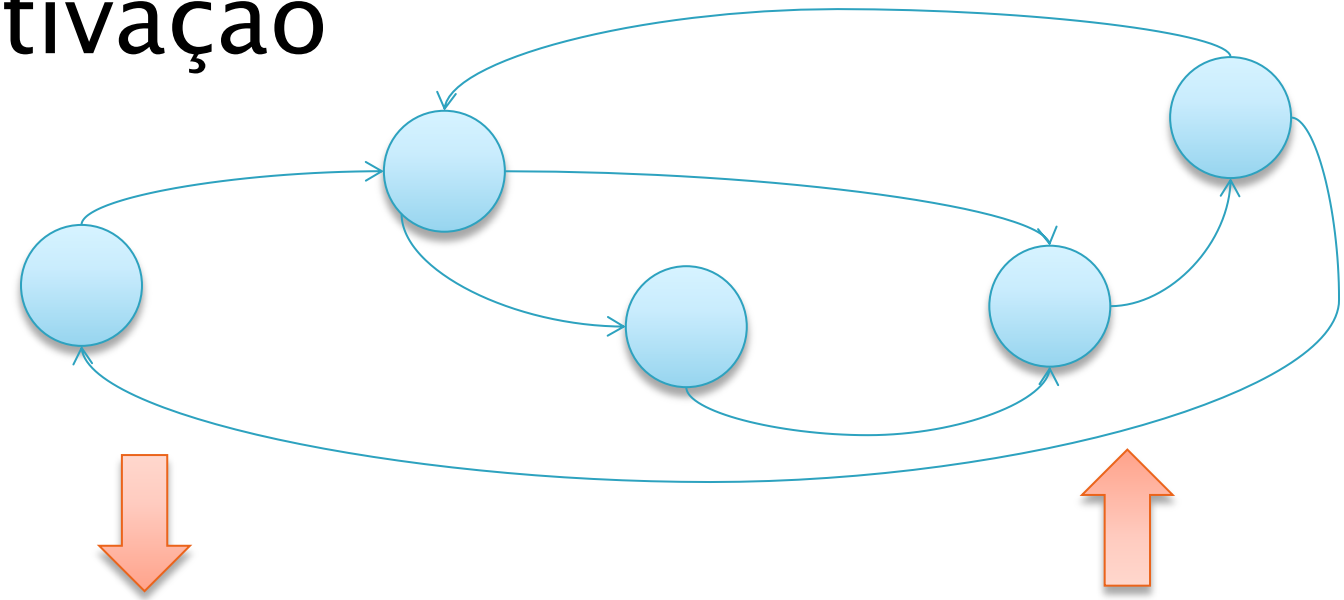


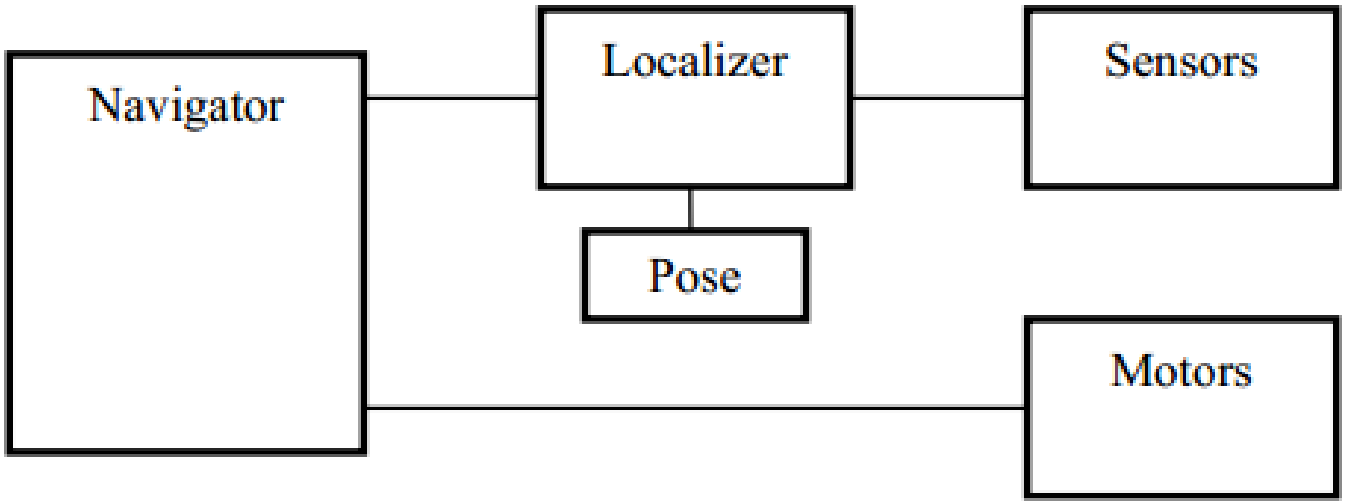
Motivação

- Implementação de um “Piloto Automático“
- Comandos:
  - 1. moveTo – move to a specified location,
  - 2. turnTo – turn to face a particular direction,
  - 3. go – move continuously in one direction,
  - 4. stop.

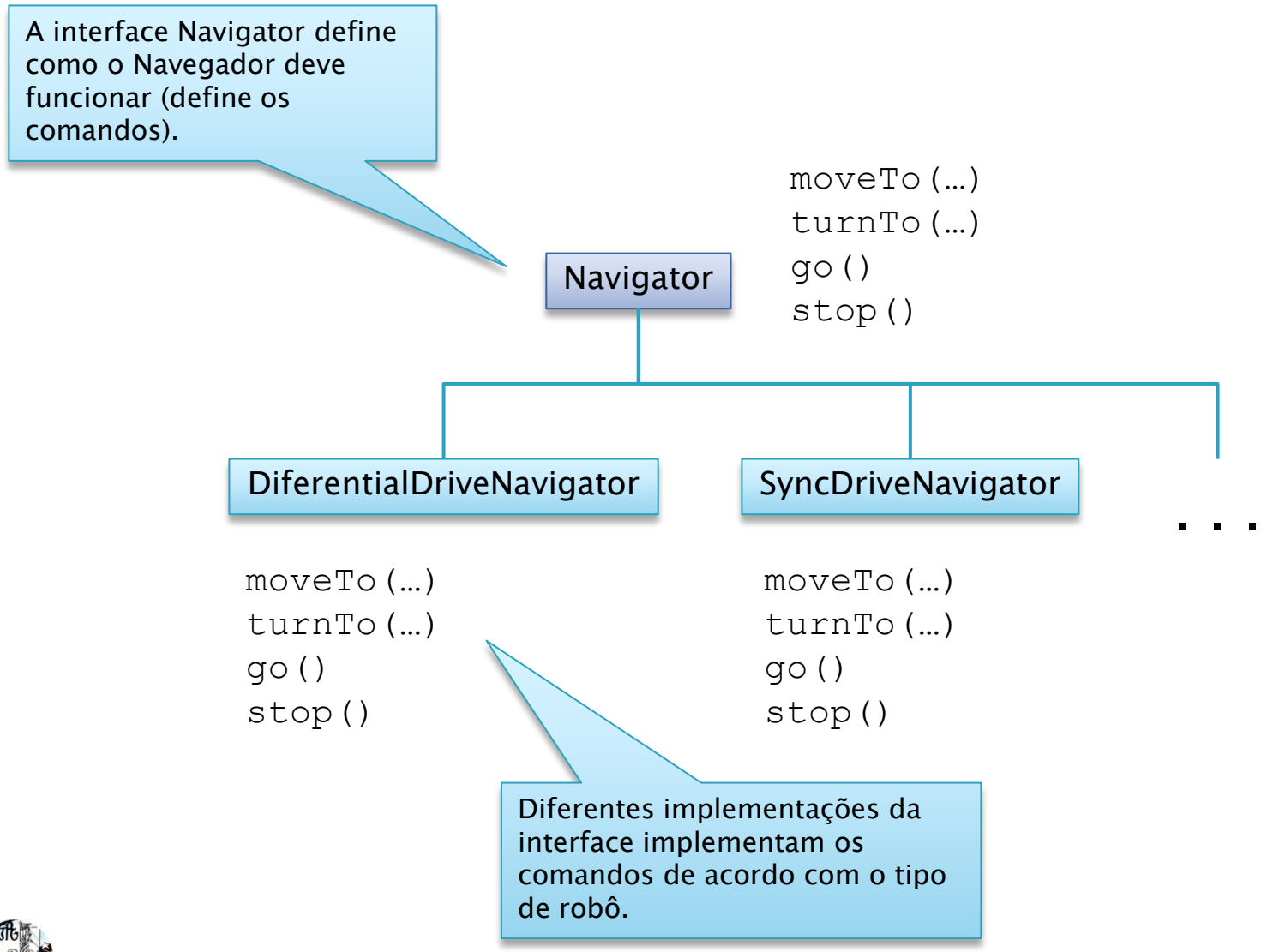


Motivação

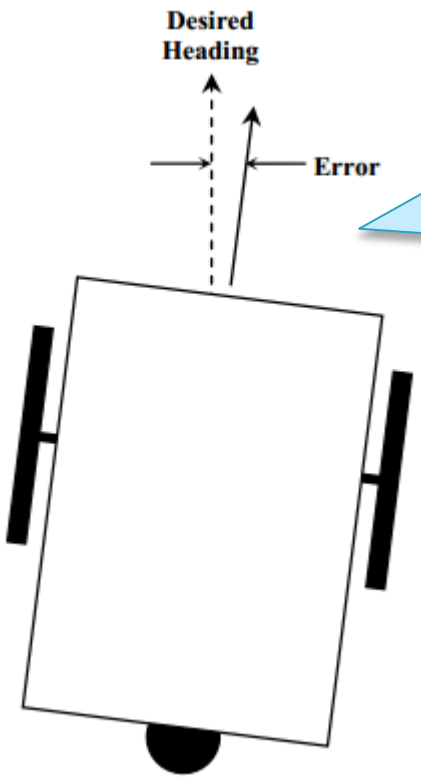




Navigator


A interface Navigator define como o Navegador deve funcionar (define os comandos).
moveTo(…)
turnTo(…)
go()
stop()
Navigator
DiferentialDriveNavigator
SyncDriveNavigator
. . .
moveTo(…)
turnTo(…)
go()
stop()
moveTo(…)
turnTo(…)
go()
stop()
Diferentes implementações da interface implementam os comandos de acordo com o tipo de robô.

Implementação do DiferentialDriveNavigator



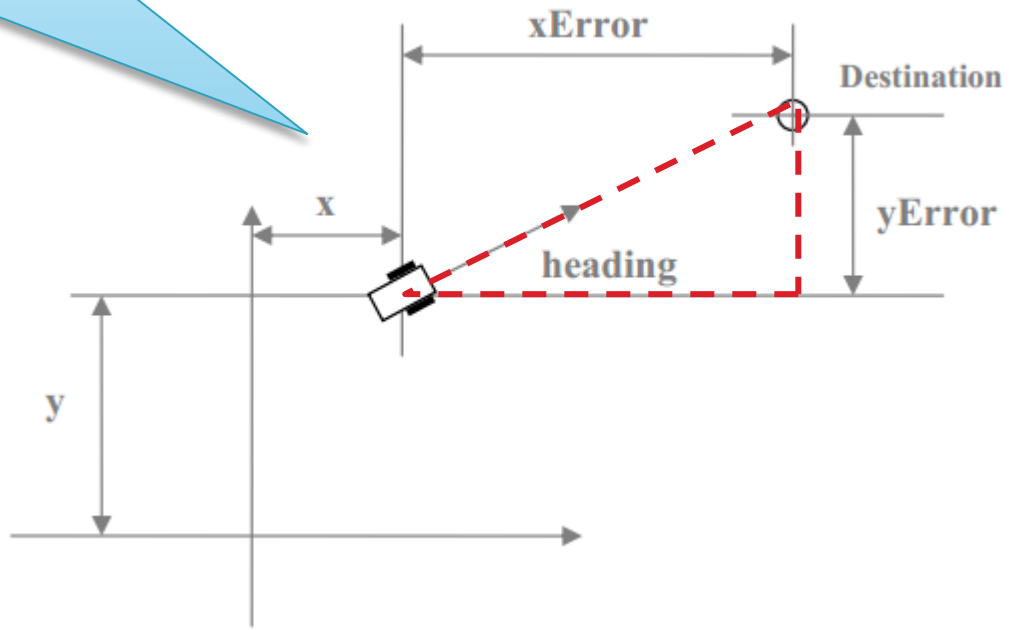
O comando moveTo(x, y) usa o comando turnTo(a) para fazer o robô rodar sobre si próprio de modo a ficar virado na direção de (x, y) e depois usa o comando go() para fazer o robô andar em frente até chagar ao destino.

É usado controlo proporcional para manter o robô na direção correta. Isto é, enquanto avança em relação ao destino, o robô calcula o erro entre a direcção desejada e a direcção actual medida pelo Localizador. Quando o erro é diferente de zero, o robô faz pequenas correcções proporcionais ao erro (rodando sobre si próprio), para corrigir a direcção.

Implementação do DiferentialDriveNavigator

O robô sabe se chegou ao destino quando a distância entre a sua localização atual e a localização de destino for inferior a um dado limiar.

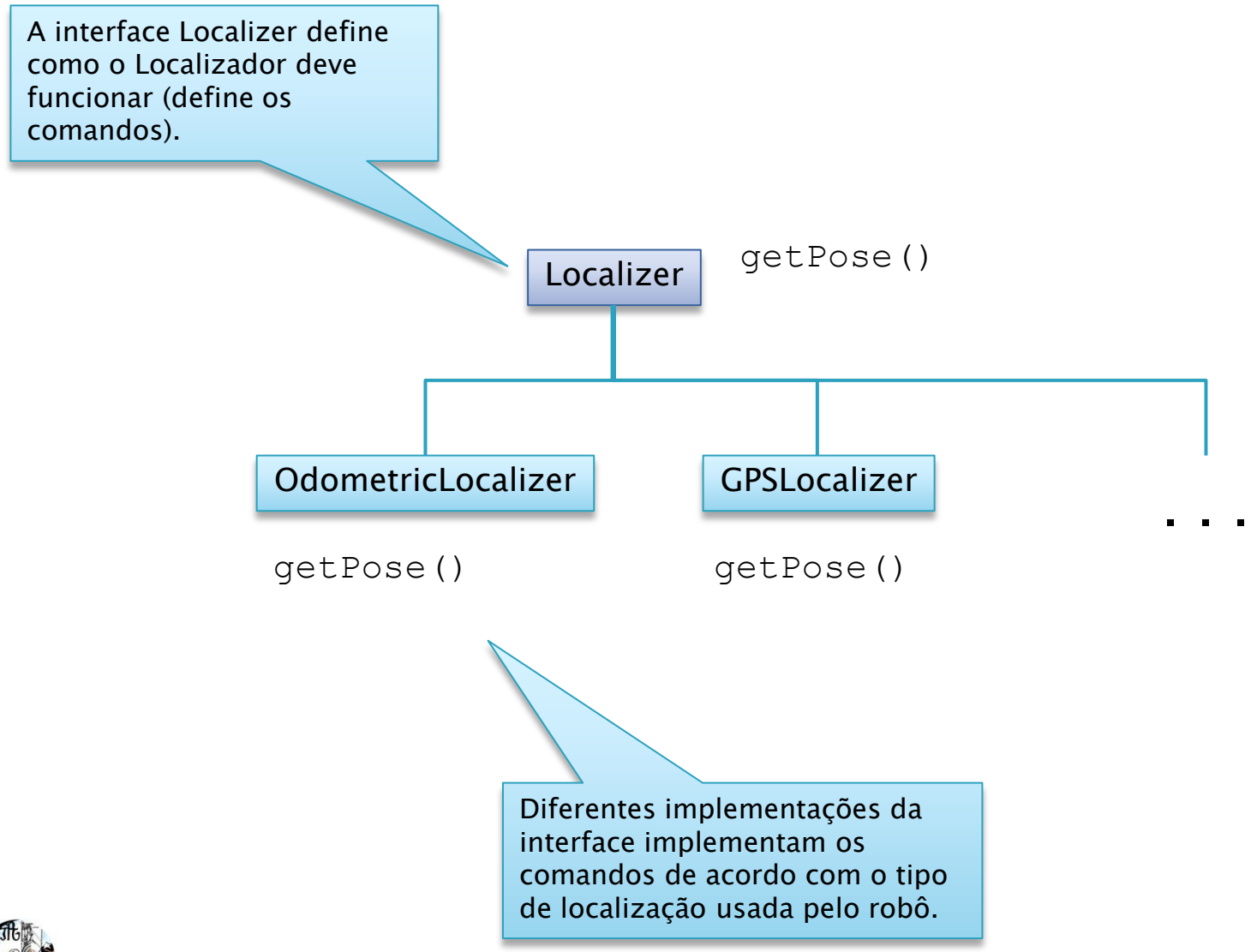
Para evitar cálculos com raízes quadradas, a distância é calculada somando os dois catetos.



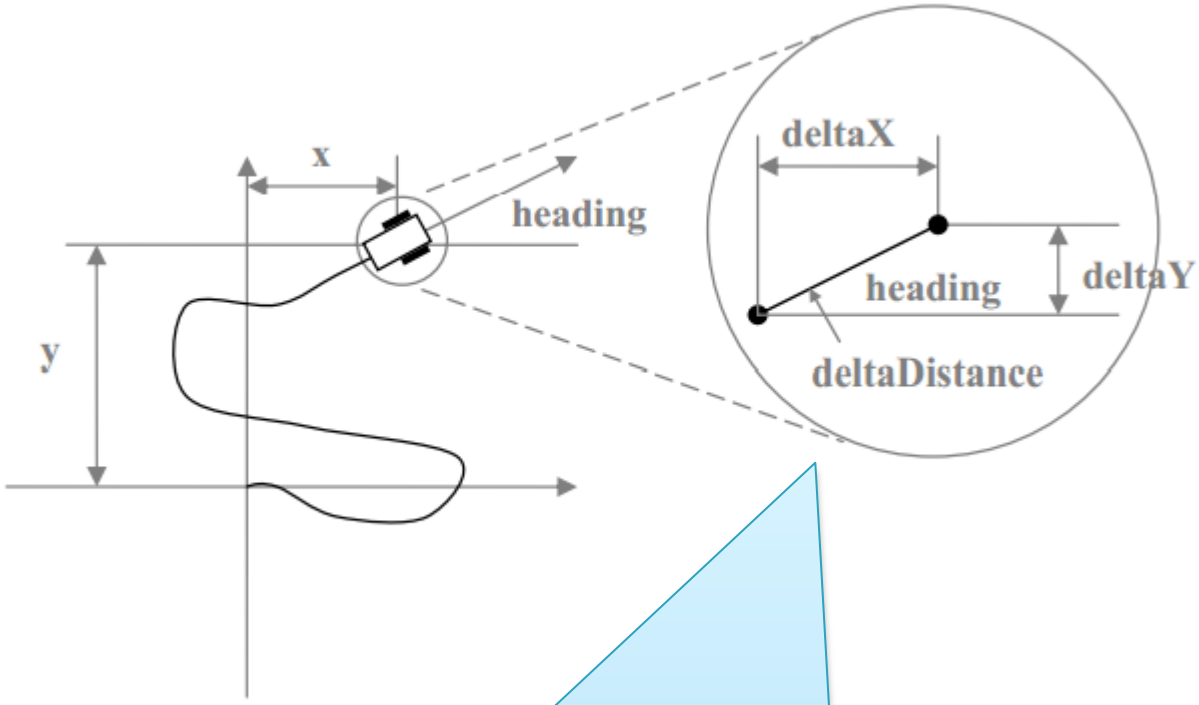
Implementação do DiferentialDriveNavigator

```
import com.ridgesoft.robotics.Navigator;
import com.ridgesoft.robotics.DifferentialDriveNavigator;
. . .
Navigator navigator = new DifferentialDriveNavigator(
  leftMotor, rightMotor, //Objetos da classe Motor
  localizer, //Objeto da classe Localizer
  8, 6,    //Velocidades base para andar e frente e rodar
  25.0f,   //Ganho para o controlo proporcional (go)
  0.5f,    //Limiar para detetar se já chegou ao destino(moveTo)
  0.08f,   //Limiares para detetar se já rodou o pretendido (turnTo)
  Thread.MAX_PRIORITY - 2, //Prioridade do Thread
  50);     //período do Thread
. . .
// Drive to the four corners of the square.
navigator.moveTo(SIDE_LENGTH, 0.0f);
navigator.moveTo(SIDE_LENGTH, -SIDE_LENGTH);
navigator.moveTo(0.0f, -SIDE_LENGTH);
navigator.moveTo(0.0f, 0.0f);
navigator.turnTo(0.0f);
```

Localizer


A interface Localizer define como o Localizador deve funcionar (define os comandos).
Localizer
getPose()
OdometricLocalizer
GPSLocalizer
. . .
getPose()
getPose()
Diferentes implementações da interface implementam os comandos de acordo com o tipo de localização usada pelo robô.

Implementação do OdometricLocalizer



Em cada instante de tempo (muito pequeno), consideramos que o robô roda sobre si próprio um ângulo **heading** e percorre uma distância **deltaDistance**.

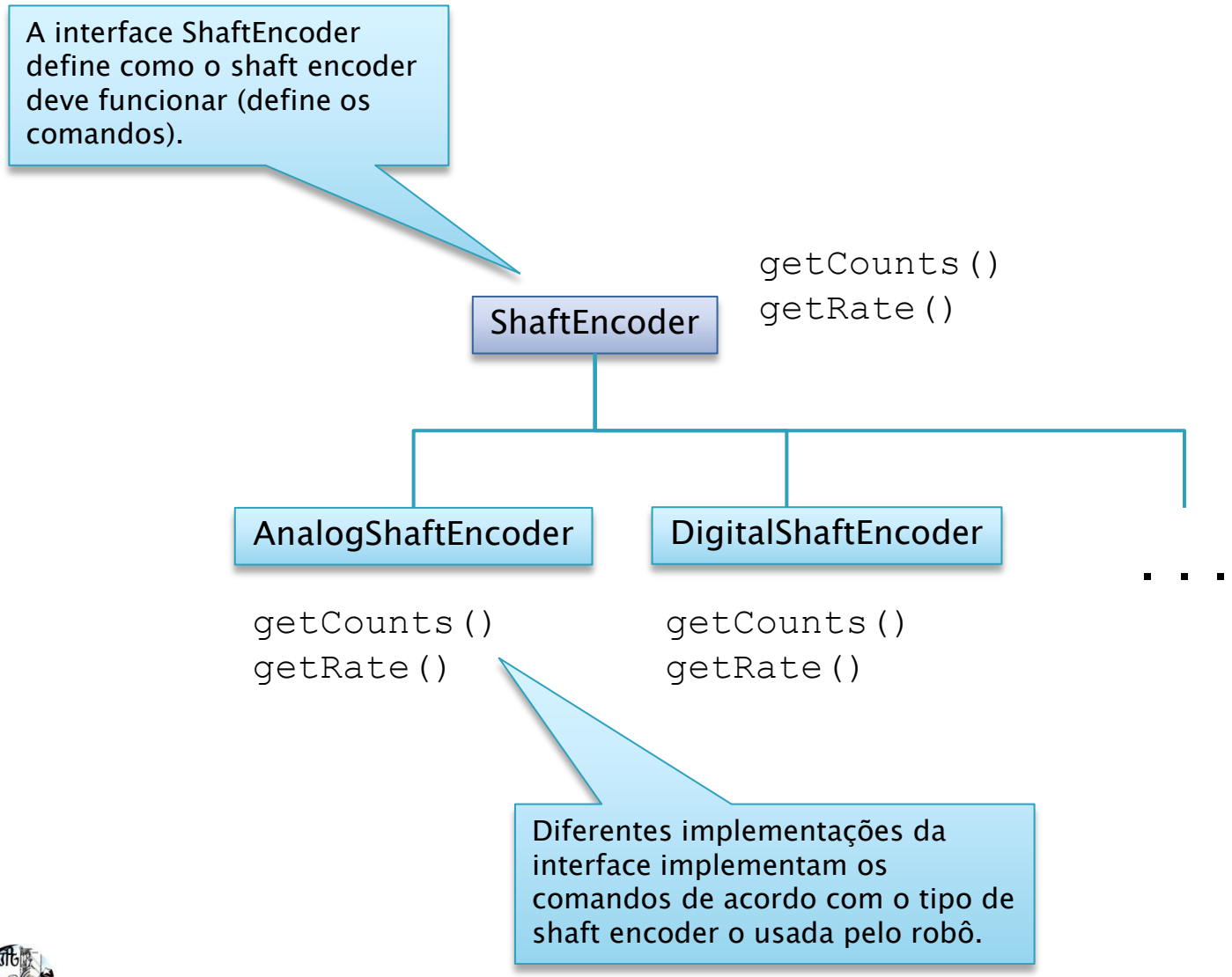
Aplicando o Teorema de Pitágoras, podemos determinar o **deltaX** e o **deltaY** que somados aos valores anteriores de **x** e **y** dão uma estimativa da nova pose do robô.

Os valores de heading e deltaDistance são determinados através de Odometria.

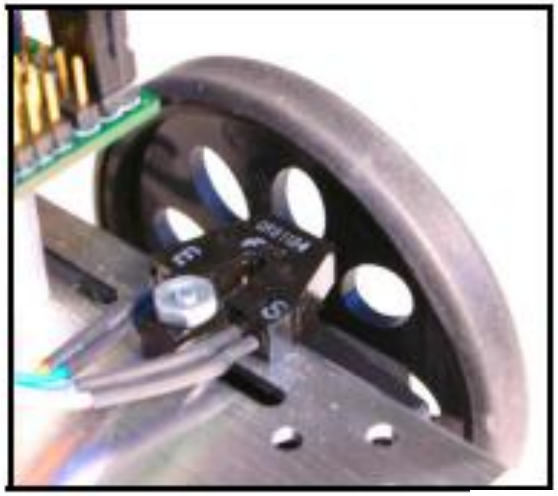
Implementação do OdometricLocalizer

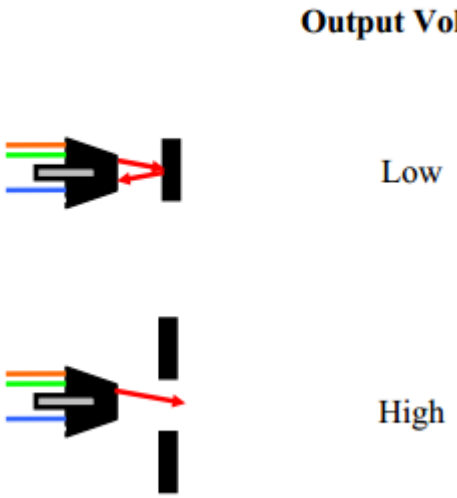
```
import com.ridgesoft.robotics.Localizer;
import com.ridgesoft.robotics.OdometricLocalizer;
. . .
Localizer localizer = new OdometricLocalizer(
   leftEncoder, rightEncoder, //Objetos ShaftEncoder
   2.65f,   //Diâmetro das rodas (polegadas)
   4.55f,   //Distância entre rodas (polegadas)
   16,      //Número de pulsos por volta
   Thread.MAX_PRIORITY - 1, //Prioridade do Thread
   30);     //Período do Thread
...
Navigator navigator = new DifferentialDriveNavigator(
  leftMotor, rightMotor, //Objetos da classe Motor
  localizer, //Objeto da classe Localizer
. . .
```

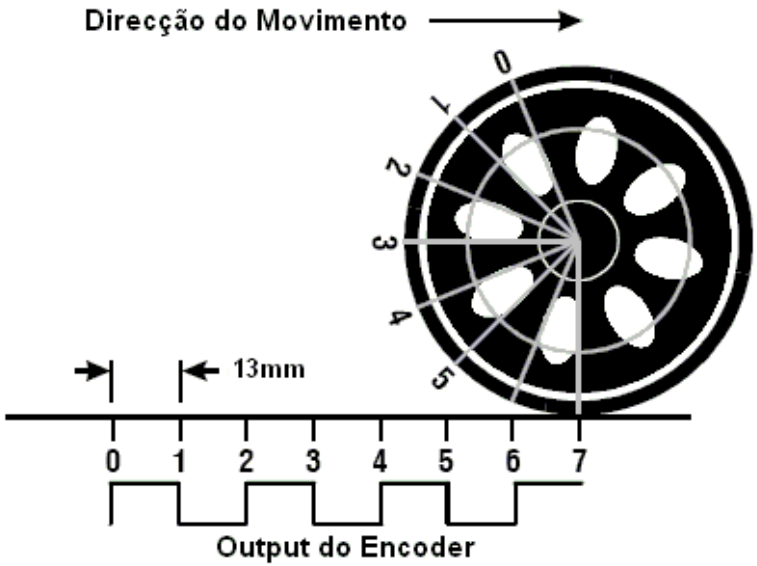
ShaftEncoder



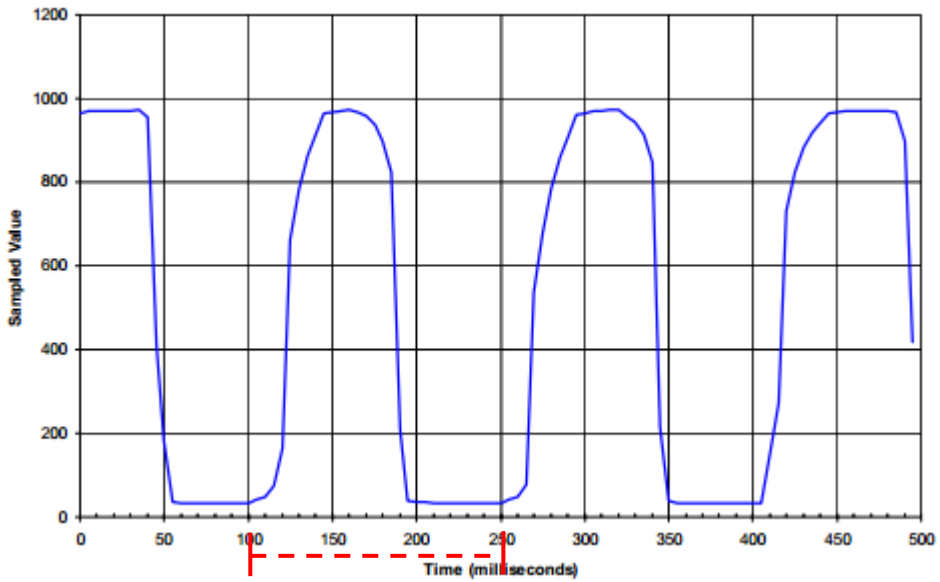
Implementação do AnalogShaftEncoder





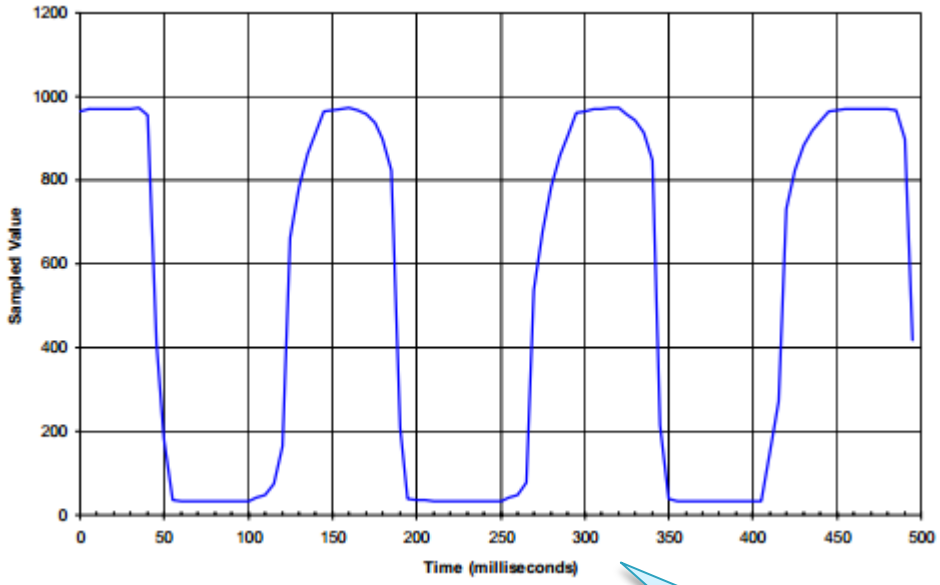


Implementação do AnalogShaftEncoder



O tempo entre arestas ascendentes é aproximadamente 150 milissegundos que corresponde à passagem de um "buraco" e de um "não buraco" (1/8 da roda).
Multiplicando por 8, são necessários 1.2 segundos para a roda completar uma volta completa. Considerando que o motor gira à velocidade máxima e a roda gira livremente, a velocidade máxima do robô é proximamente 50 rpm.

Implementação do AnalogShaftEncoder



O shaft encoder é implementado de modo a detetar a aresta ascendente e a aresta descendente. Isto permite contar 16 pulsos por volta.

Uma vez que demora 150 milissegundos para que a roda gire 1/8 de volta à velocidade máxima, o programa deve ler o sensor a cada 75 milissegundos (no máximo), para garantir que não perde nenhuma aresta.

Implementação do AnalogShaftEncoder

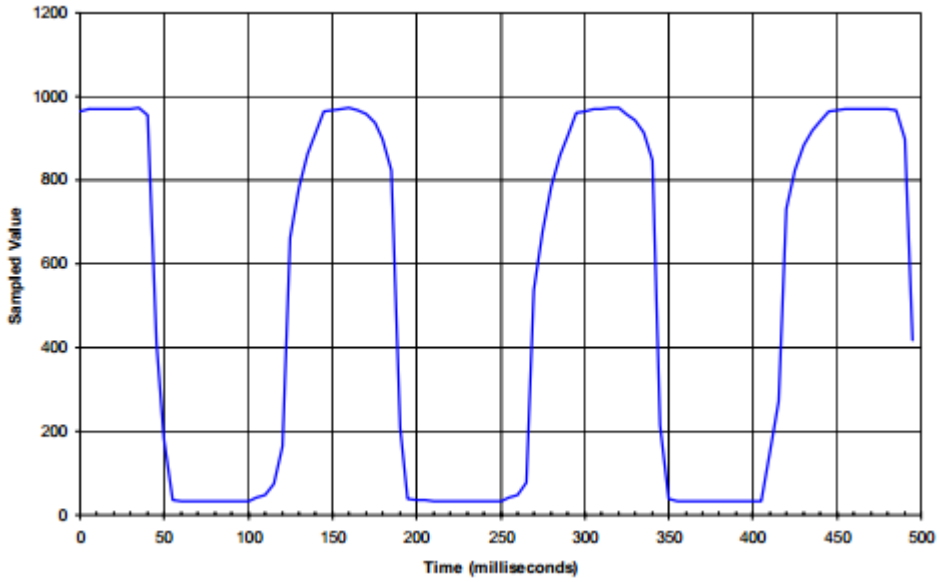
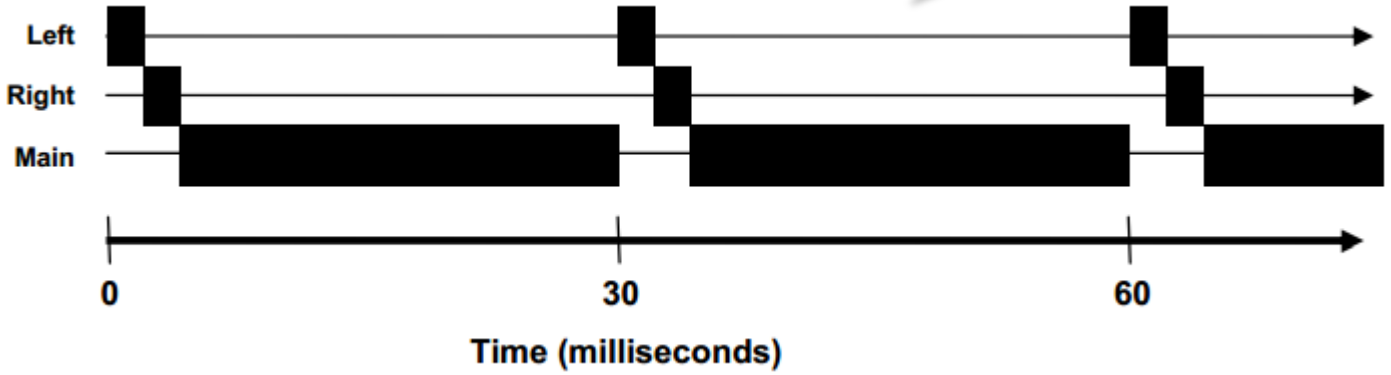


Diagrama temporal dos períodos dos threads dos shaft encoders.

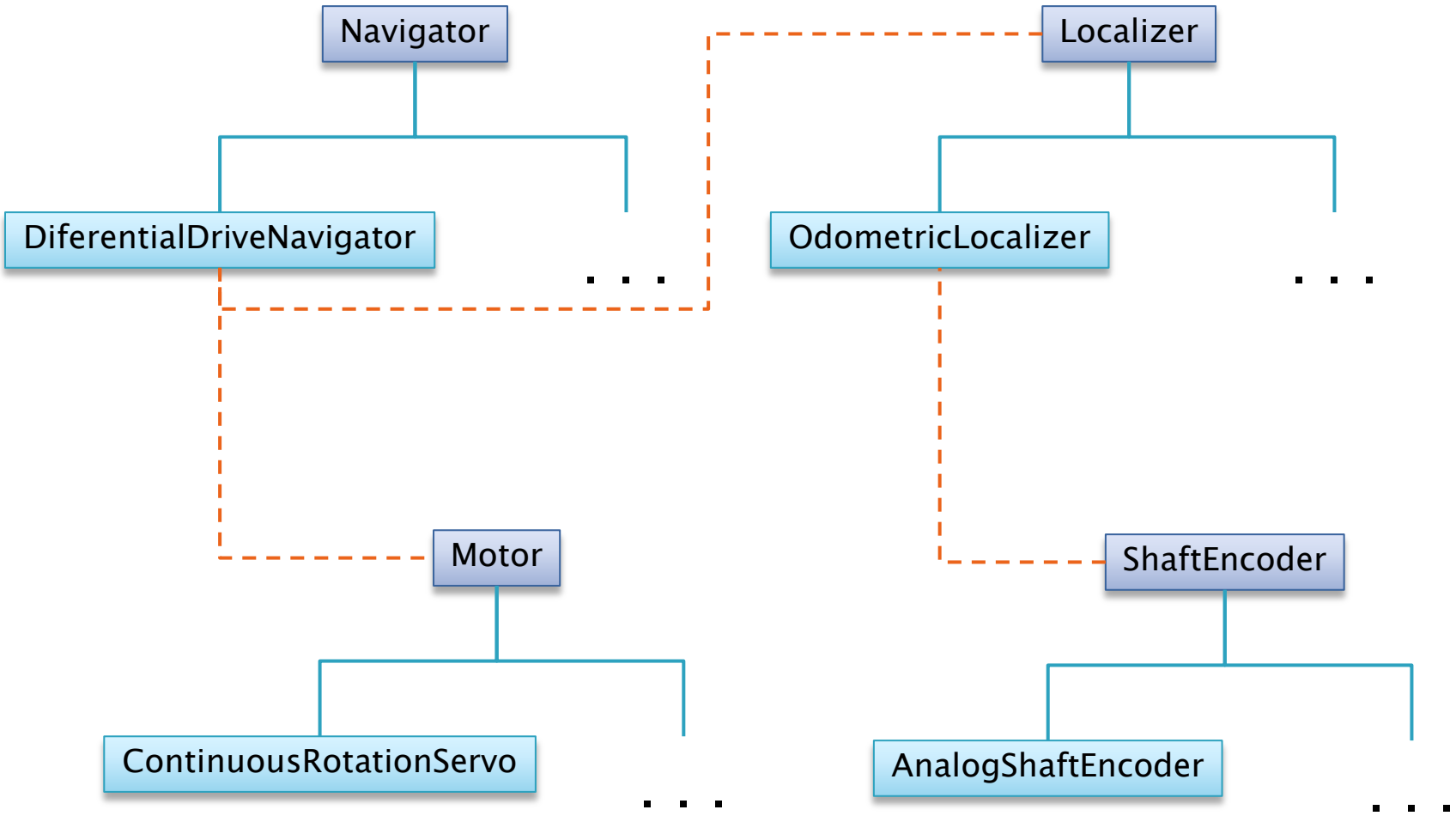


Implementação do AnalogShaftEncoder

```
import com.ridgesoft.robotics.ShaftEncoder;
import com.ridgesoft.robotics.AnalogShaftEncoder;
. . .
ShaftEncoder leftEncoder = new AnalogShaftEncoder(
   leftWheelSensor,   //Objeto AnalogInput
   250,               //Limiar aresta descendente
   750,               //Limiar aresta ascendente
   30,                //Período do Thread
   Thread.MAX_PRIORITY); //Prioridade do Thread

ShaftEncoder rightEncoder = new AnalogShaftEncoder(
    rightWheelSensor, 250, 750, 30, Thread.MAX_PRIORITY);
. . .
Localizer localizer = new OdometricLocalizer(
 leftEncoder, rightEncoder, //Objetos ShaftEncoder
. . .
```

Classes do Sistema de Navegação



Teste