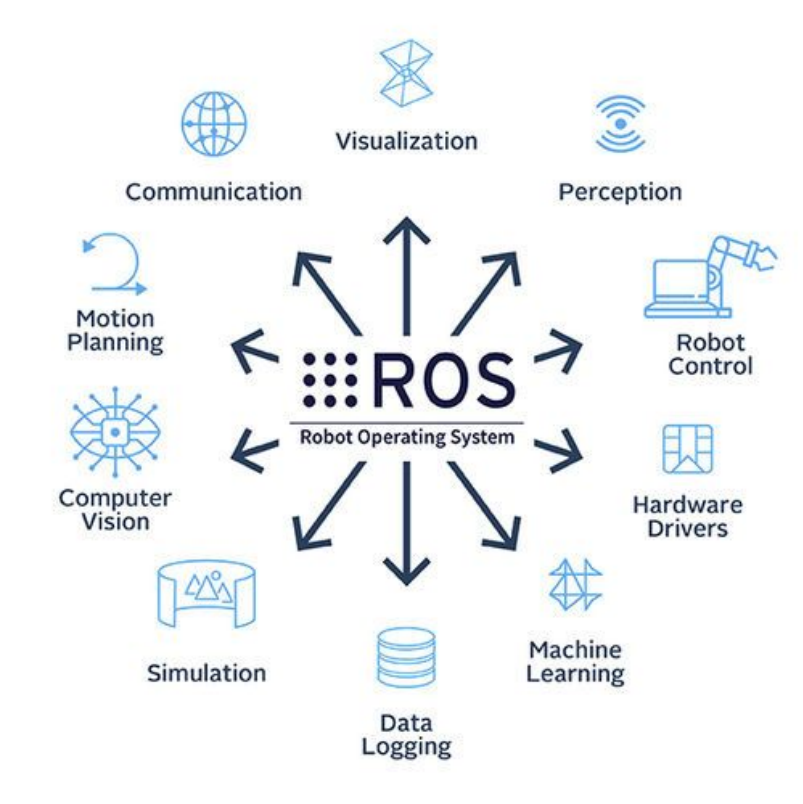
ROS2: ROBOTIC OPERATING SYSTEM (Middleware)



ROS: ROBOTIC OPERATING SYSTEM (Middleware)

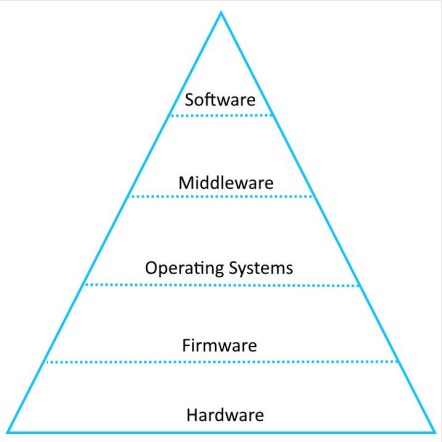
ROS is:

Robot Operating System (**ROS or ros**) is an open-source robotics middleware suite. Although ROS is not an operating system but a collection of software frameworks for robot software development, it provides services designed for a heterogeneous computer cluster such as hardware abstraction, low-level device control

fonte: wikipedia

ROS: ROBOTIC OPERATING SYSTEM (Middleware)

Software: “sequência de instruções a serem seguidas e/ou executadas, na manipulação, redirecionamento ou modificação de um dado (informação) ou acontecimento”.

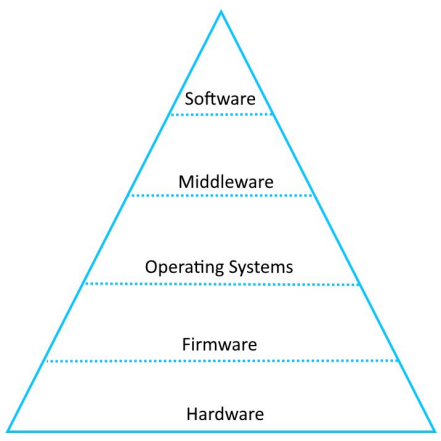


fonte: wikipedia

ROS: ROBOTIC OPERATING SYSTEM (Middleware)

Middleware: Middleware é o software de computador que fornece serviços para softwares aplicativos além daqueles disponíveis pelo sistema operacional. Pode ser descrito como "cola de software".

O middleware facilita aos desenvolvedores de software implementarem comunicação e entrada/saída, de forma que eles possam focar no propósito específico de sua aplicação.



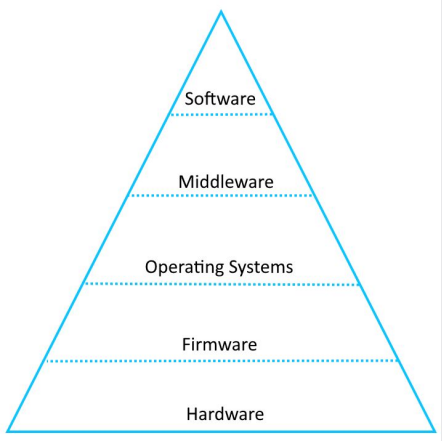
fonte: wikipedia

ROS: ROBOTIC OPERATING SYSTEM (Middleware)

Operating System:

Um sistema operacional (SO) é um software de sistema que gerencia o hardware do computador, recursos de software e fornece serviços comuns para programas de computador.

Os “Time-sharing operating systems” programam tarefas para o uso eficiente do sistema e também podem incluir software de contabilidade para alocação de custos de tempo de processador, armazenamento em massa, impressão e outros recursos.

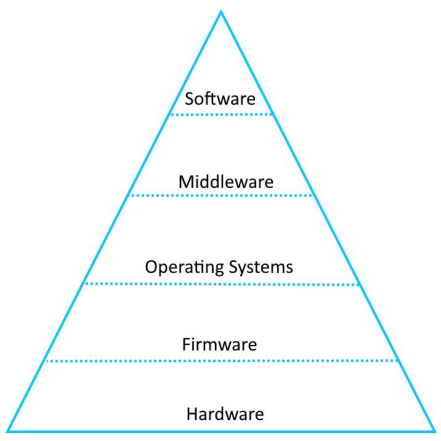


fonte: wikipedia

ROS: ROBOTIC OPERATING SYSTEM (Middleware)

Firmware:

Em eletrônica e computação, firmware é uma classe específica de software de computador que fornece controle de baixo nível para o hardware específico do dispositivo. O firmware pode fornecer um ambiente operacional padronizado para o software mais complexo do dispositivo (permitindo maior independência de hardware) ou, para dispositivos menos complexos, atuar como o sistema operacional completo do dispositivo, executando todas as funções de controle, monitoramento e manipulação de dados. Exemplos típicos de dispositivos que contêm firmware são sistemas embarcados.



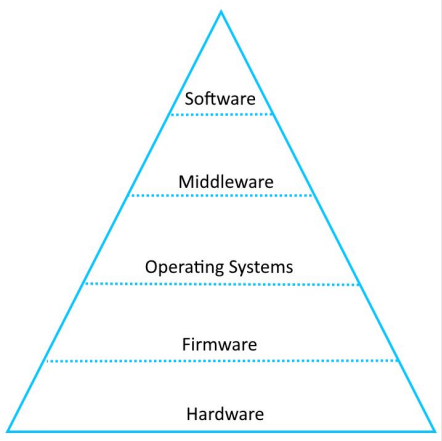
fonte: wikipedia

ROS: ROBOTIC OPERATING SYSTEM (Middleware)

Hardware:

Computing and electronics

- Computer hardware: physical parts of a computer
- Digital electronics: electronics that operate on digital signals
- Electronic component: device in an electronic system used to affect electrons, usually industrial products
- Electronic hardware: interconnected electronic components which perform analog or logic operations
- Networking hardware: devices that enable use of a computer network
- Hardware acceleration: the speedup of computing tasks by performing them in customized hardware rather than software

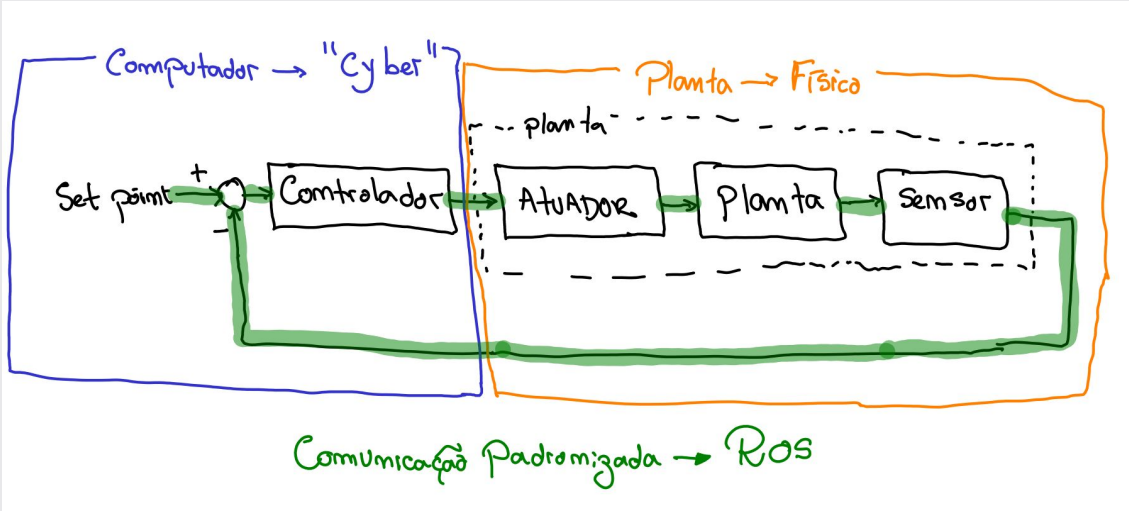


fonte: wikipedia

ROS: Uma ferramenta para sistemas ciberfísicos

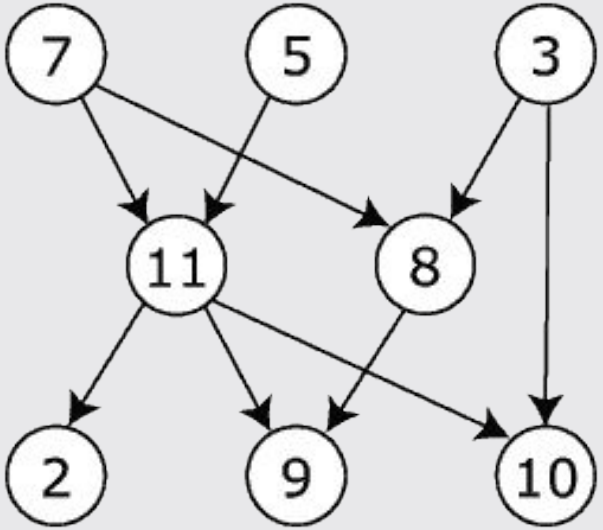
Sistema Ciberfísico: A cyber-physical system (CPS) is an integration of computation with physical processes whose behavior is defined by both cyber and physical parts of the system.

Prof Anis Koubaa:
“Cheguei a conclusão de que era impossível desenvolver um robô sem uma comunicação padronizada”.
-Colaborador do ROS



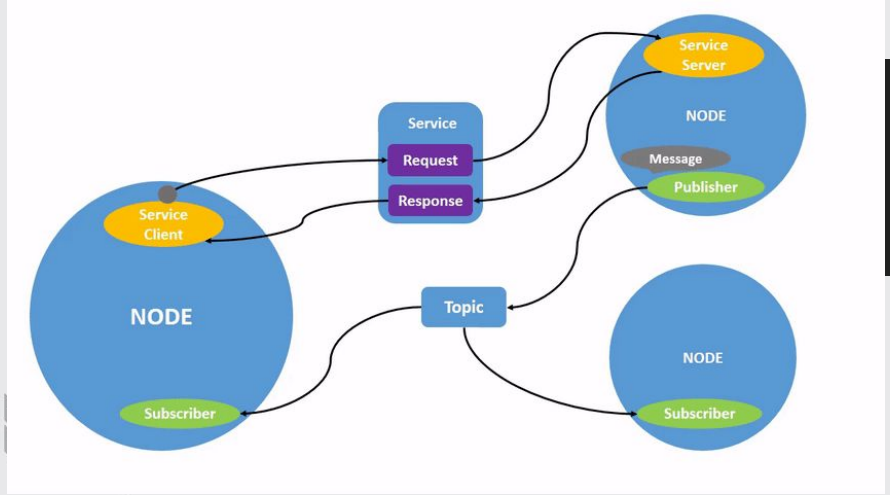
ROS2 - Robot Operating System

- O funcionamento do ROS é representado por um grafo, que organiza a interação entre os programas através de topics, services e actions
- Grafos são estruturas muito utilizadas na computação e na matemática, e consistem em um conjunto de vértices ou nós, ligados por arestas direcionadas.
- No caso do ROS, o grafo representa a comunicação: os nós são os programas (nodes), e as arestas representam vias de comunicação (topics, services e actions) entre eles



Nodes

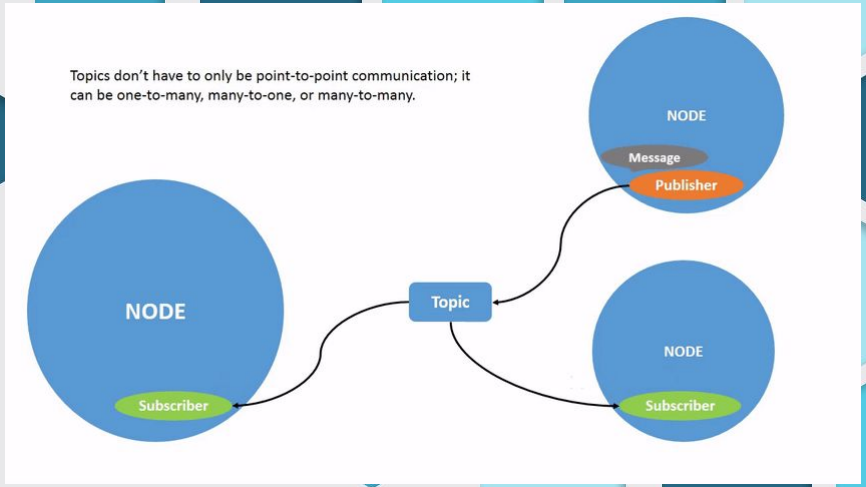
- Nodes têm uma função modular, de propósito único (um node controla os motores das hélices, outro controla a câmera do drone).
- Nodes não transportam informações, e sim as recebem ou mandam informações através de topics, services e actions.
- De forma simplificada, temos abaixo um programa sendo executado.




```
>>> ros2 node list  # lista nodes rodando
>>> ros2 node info [nome do node]  # lista de subs, pubs, etc. que interagem com o node
>>> ros2 run [nome do package] [nome do node]  # roda node
```

Topics

- Os topics são os canais de comunicação entre nodes. É por eles que passam as mensagens, levando informações de um a outro.
- Tópicos não funcionam numa rota única, ele pode ir de um node para vários.
- Publisher : publica mensagem em um tópico
- Subscriber : se inscreve nesse tópico e recebe as mensagens publicadas nele




```
>>> ros2 topic list  # lista tópicos abertos
>>> ros2 topic info /nome_topico  # mostra tipo de mensagem trafegando no tópico
>>> ros2 topic echo /nome_topico  # mostra o conteúdo da mensagem que está trafegando
>>> ros2 topic pub /nome_topic [tipo de mensagem] [args]  # publica [args] no tópico
>>> ros2 topic find [tipo de mensagem]  # devolve tópicos que trafegam [tipo de mensagem]
```

```
>>> ros2  interface show [tipo de mensagem] # mostra o formato da mensagem
```

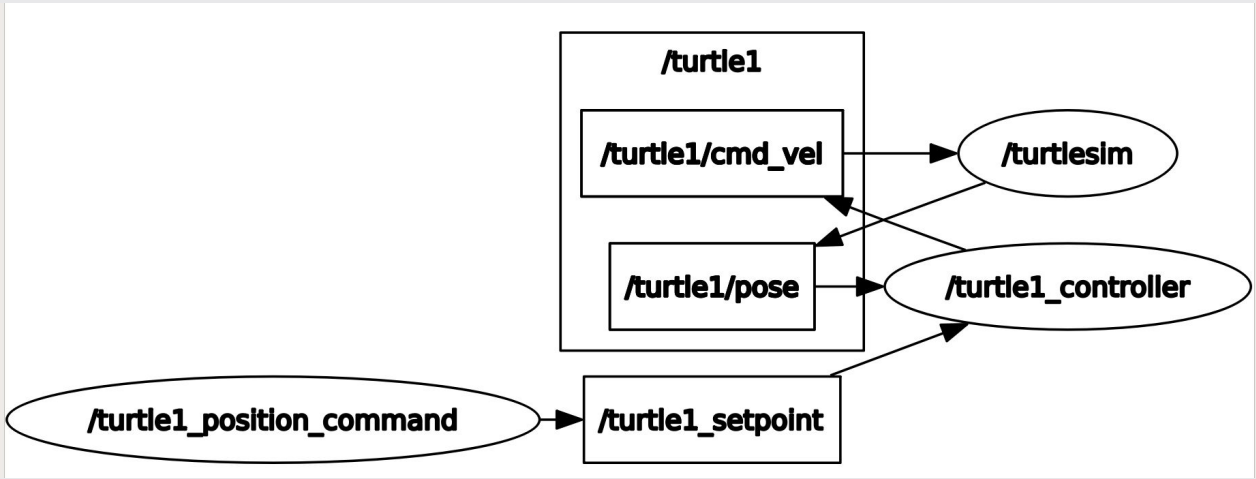
Messages

São os diferentes tipos de informações que podem ser passadas pelos tópicos

```
>>> rosmsg list
>>> rosmsg show [tipo de mensagem]
```

Retomada de Conceitos

1) ROS Core
2) Nodes
3) Topics
    a) Publishers
    b) Subscribers
4) Messages



Como utilizar o ROS

1) Raciocínio
   a) Documentação do Robô na Internet: ROS Wiki e GitHub
   b) Inspeção local: explorar nodes, topics, etc. pelo terminal
   c) Avaliar as formas de interação (ações e dados disponíveis)
   d) Planejar forma de implementação
      i) Ex: Controle de Posição ou de Velocidade?

2) Implementação
   a) Estrutura dos packages
      i) CMakeLists.txt
      ii) package.xml
      iii) Pastas: src, scripts, launch, include, etc.
   b) Linguagens: C++, Python, Matlab

Python

Estratégia Cartesiana:

Avançar em x e em y até que o erro seja nulo (sem rodar a tartaruga)

A trajetória esperada é uma reta diagonal

VANTAGEM:
Estratégias cartesianas conseguem utilizar a máxima precisão numérica imposta pelo sistema de informações (rostopic/ rosmsg). Desse modo o robô atinge uma posição final mais próxima da desejada.

DESVANTAGEM:
O robô “não olha para frente” ao longo da trajetória. Isso pode ser contornado se o sistema sensor do robô for acoplado a um motor que forneça uma rotação individual do sensor em relação ao robô.

Python

```python
#! /usr/bin/env python

#Importing Libraries
import rospy
from geometry_msgs.msg import Twist
from turtlesim.msg import Pose
import math

class Turtle:
    def __init__(self):
        #Starting rosnode
        rospy.init_node("goToGoal_proportionalCartesianControl")

        #Instantiating objects
        self.pose_acctual = Pose()
        self.vel = Twist()
        self.goal_pose =  Pose()

        #Define sampling frequency
        self.rate = rospy.Rate(10) #10Hz
```

Python

```python
        #= Publihsers Instantiation =
        #============================

        #Velocity Publisher
        self.vel_pub = rospy.Publisher("/turtle1/cmd_vel", Twist,queue_size=10)

        #= Subscribers Instantiation =
        #=============================

        #Pose Subscriber
        self.pose_sub = rospy.Subscriber("/turtle1/pose", Pose, self.pose_callback)

    def pose_callback(self,data_from_new_msg):
        #This function is automaticaly called by ROS. It's read a new message and save it's content
        #at variable data_from_new_msg. From this variable we will code a callback function that
        #extracts only the position information. It's will not take info about velocity.

        self.pose_acctual.x = data_from_new_msg.x # x postion
        self.pose_acctual.y = data_from_new_msg.y # y position
        self.pose_acctual.theta = data_from_new_msg.theta #angular position

    def error_x(self, pose, goal_pose):
        #This function calculates the error in x direction between pose_acctual and goal_pose
        return  self.goal_pose.x - self.pose_acctual.x

    def error_y(self, pose, goal_pose):
        #This function calculates the error in x direction between pose_acctual and goal_pose
        return  self.goal_pose.y - self.pose_acctual.y
```

Python

```
52    set_position(self):
53    #This method will take a turtle from (x,y) position and drive the turtle to a (x_goal,y_goal) postiion
54
55    #Using the high level of precision of Turtlesim_node acording to rostopic echo /turtle1/pose we define
56    #a "ros_zero" variable as an aproximation of absolute zero with a tolerance.
57    self.ros_zero = 0.000000001
58
59    #Menu of goal selection
60    self.goal_pose.x = float(input("Set turtle's x goal: "))
61    self.goal_pose.y = float(input("Set turtle's y goal: "))
62
63    #Proportional gain controller
64    self.k = 1 #[(m/s)/m]
65
66    while abs(self.error_x(self.pose_acctual, self.goal_pose)) > self.ros_zero or abs(self.error_y(self.pose_acctual, self.goal_pose)) > self.ros_zero and not rospy.is_shutdown():
67
68        #= X direction control =
69        #======================
70        if abs(self.error_x(self.pose_acctual, self.goal_pose)) > self.ros_zero:
71            # Set velocity by proportional control 10[(m/s)/m]*distnace()[m] = vel [m/s]
72            self.vel.linear.x = self.k*self.error_x(self.pose_acctual, self.goal_pose)
73
74            self.vel_pub.publish(self.vel)
75            self.rate.sleep()
76        else:
77            #Stop when x goal was achived
78            self.vel.linear.x = 0
79            self.vel_pub.publish(self.vel)
80
81        #= Y direction control =
82        #======================
83        if abs(self.error_y(self.pose_acctual, self.goal_pose)) > self.ros_zero:
84            # Set velocity by proportional control 10[(m/s)/m]*distnace()[m] = vel [m/s]
85            self.vel.linear.y = self.k*self.error_y(self.pose_acctual, self.goal_pose)
86
87            self.vel_pub.publish(self.vel)
88            self.rate.sleep()
89        else:
90            #Stop when y goal was achived
91            self.vel.linear.y = 0
92            self.vel_pub.publish(self.vel)
```

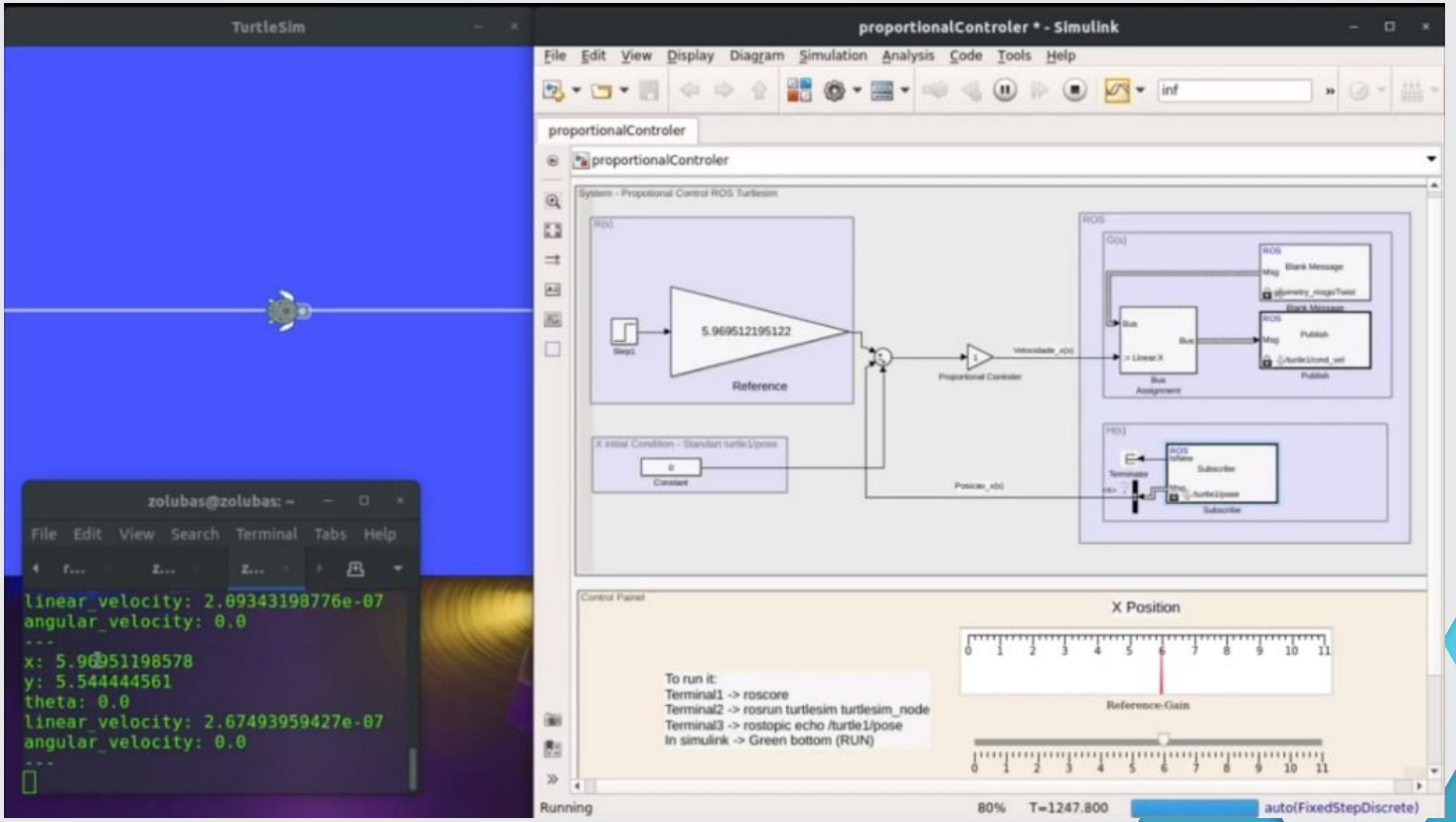
Python

```
99
100  Main ============================
101
102  me__ == '__main__':
103
104  stantiating a turtle
105  tle = Turtle()
106  king for a goal
107  tle.set_position()
```

Exemplo ROS + MATLAB

Exemplo de implementação de controle proporcional utilizando o matlab integrado ao ROS.

Aplica-se um controle unidimensional em x.



Exemplo ROS + MATLAB

Exemplo de implementação de controle proporcional utilizando o matlab integrado ao ROS.

Aplica-se um controle unidimensional em x.