ROS: ROBOTIC OPERATING SYSTEM (Middleware)

Gabriela Ishikura, Henrique Barros, Fernando Zolubas Preto

Aula de hoje

- Motivação para utilização do ROS
- Definição prática de ROS
  - Exemplo: ROS
- Comunicação Assíncrona
  - Topics
  - Publisher/Subscriber
  - Código Exemplo
- Comunicação Síncrona
  - Services
  - Código Exemplo
- Definição Formal de ROS
- Exemplo TurtleSim
  - Como usar a documentação para construir o exemplo
  - Como rodar um controle de posição em python
- Resumo
- Como utilizar o ROS
- Exemplos:
  - c++
  - python
  - matlab

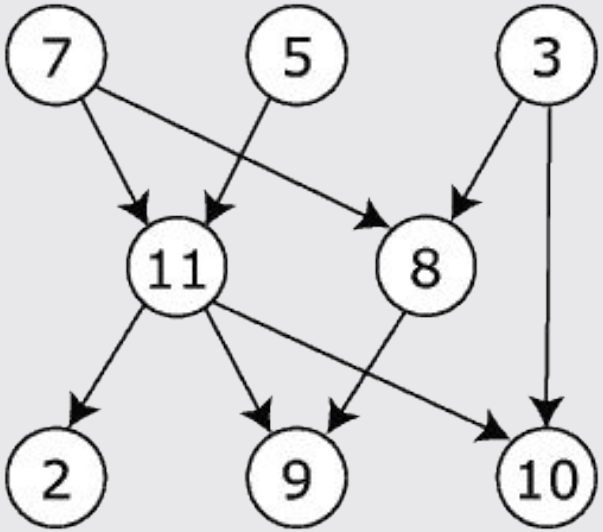
Motivação para uso do ROS

- Comunicação padronizada entre Hardware e Software
- Poder trocar de hardware sem alterar o controle/missão

Definição prática de ROS

Definição prática de ROS

- Ros como grafo de comunicação
  - Cada **node** representa:
    - Componente de hardware traduzido em software (**driver**)
    - Algoritmo (**script**)
  - Drivers e scripts trocam informações através do grafo do ROS

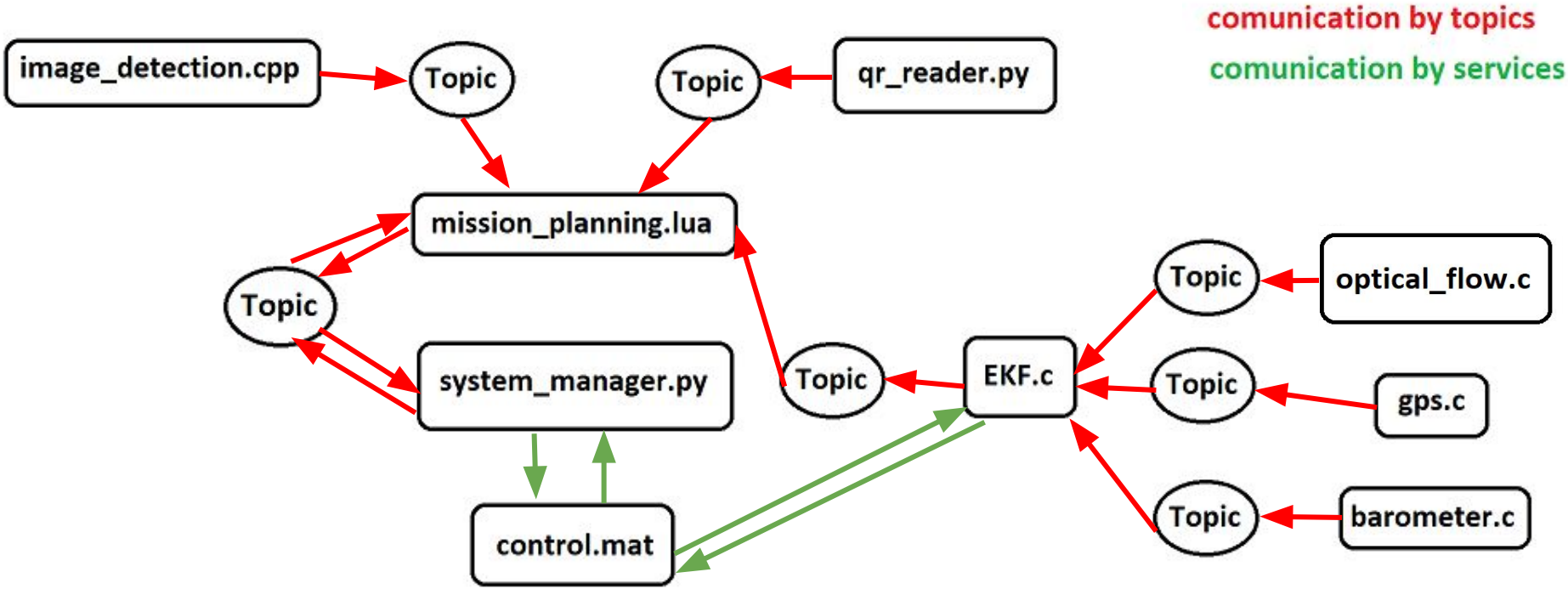


Grafo do ROS*

Obs*: o node 1 é o “roscore”.

Exemplo: ROS

- Exemplo: Drone

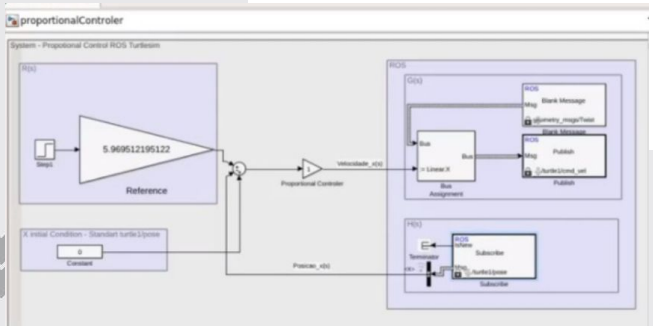


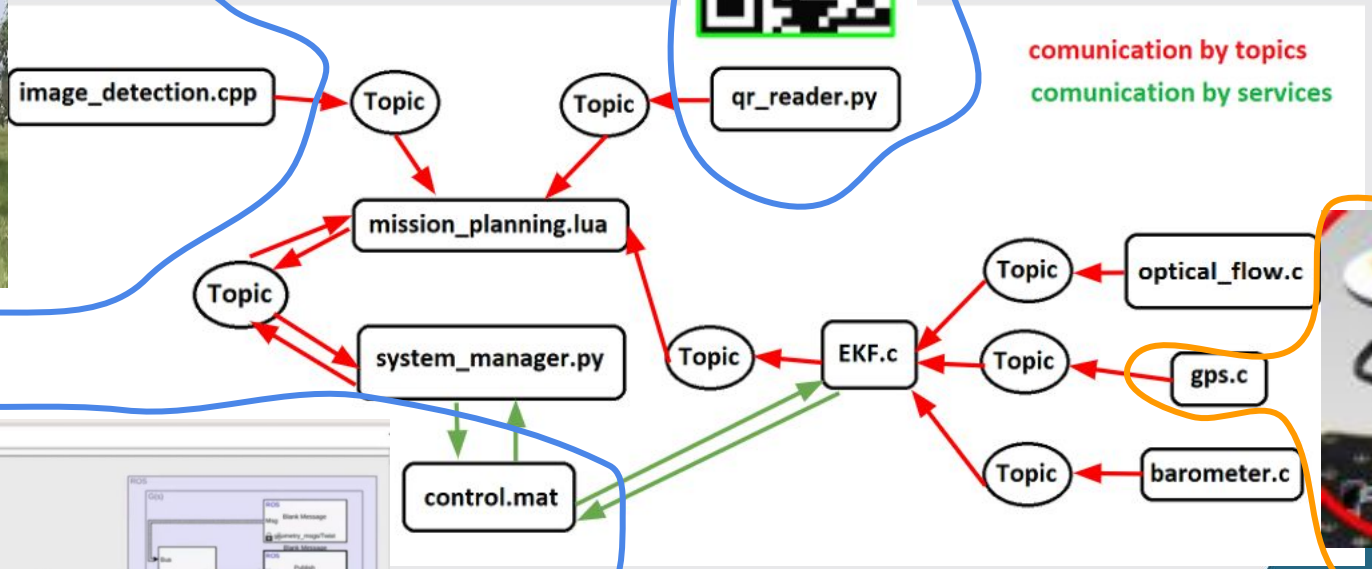
Legenda do grafo do drone

- image_detection (script) = detect objects on camera image
- qr_reader (script) = leitor de código QR
- mission_planning (script) = algorítimo que planjea a missão dado as informações fornecidas
- system_manager (script) = algorítimo que executa a missão planejadas
- control.mat (script) = controla o drone
- EKF (script) = Filtro de Kalman para do estimação vetor de estados de posição
- optical_flow (driver) = recebe as informações do sensor e publica no tópico ROS
- gps (driver) = recebe as informações do sensor e publica no tópico ROS
- barometer (driver) = recebe as informações do sensor e publica no tópico ROS

Exemplo: ROS

- Exemplo: Drone



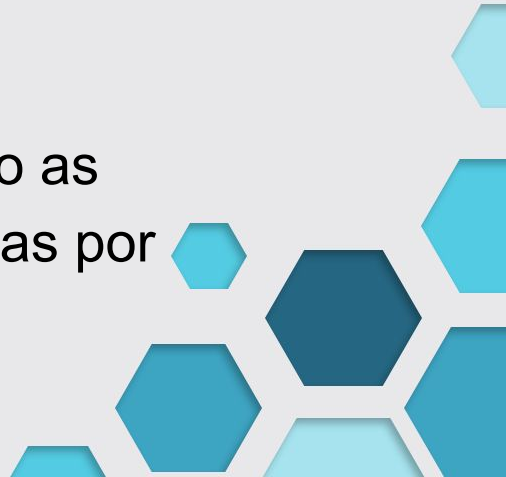


Esquemas de Comunicação do ROS

Comunicação Assíncrona

Assíncrona: Publishers/Subscribers

- Subscriber
  - Node inscrito em um tópico para receber informações assíncronamente
- Publisher
  - Node inscrito em um tópico para enviar informações assíncronamente
- Topic
  - Um tópico de interesse (assunto) onde estão as mensagens lidas por subscribers e publicadas por publishers



Assíncrona: Publishers/Subscribers

Quando o gps (publisher) tem uma nova leitura, a informação de posição é encapsulada em uma **rosmessage** que é enviada ao **topic gps_info**. Quando o dado chega no tópico, o ekf (subscriber) inscrito no **topic gps_info** recebe uma **rosmessage** contendo a informação de posição do GPS.

No exemplo a seguir, o assunto da conversa é a informação de GPS

Nodes

Programas executáveis do ROS

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

import rospy
from mavbase.MAV import MAV
import numpy as np

def go():

    rospy.init_node("mav_test")
    mav = MAV("1")

    takeoff_alt = 10
    goal_x = 5
    goal_y = 5
    altitude = 2
```

Comandos úteis

```
>>> rosnode list
>>> rosnode info [nome do node]
>>> rosrun [nome do package] [nome do node]
```

Exemplo Node

```
aluno@aluno-VirtualBox:~$ rosnode list
/gotogoal
/rosout
/turtlesim
aluno@aluno-VirtualBox:~$ rosnode info /gotogoal
--------------------------------------------------------------------------------
Node [/gotogoal]
Publications:
 * /rosout [rosgraph_msgs/Log]
 * /turtle1/cmd_vel [geometry_msgs/Twist]

Subscriptions:
 * /turtle1/pose [turtlesim/Pose]

Services:
 * /gotogoal/get_loggers
 * /gotogoal/set_logger_level


contacting node http://aluno-VirtualBox:44325/ ...
Pid: 30691
Connections:
 * topic: /turtle1/cmd_vel
    * to: /turtlesim
    * direction: outbound (42171 - 127.0.0.1:40052) [9]
    * transport: TCPROS
 * topic: /rosout
    * to: /rosout
    * direction: outbound (42171 - 127.0.0.1:40054) [14]
    * transport: TCPROS
 * topic: /turtle1/pose
    * to: /turtlesim (http://aluno-VirtualBox:40259/)
    * direction: inbound
    * transport: TCPROS
```

Topics

- Publisher : node que publica mensagem em um tópico
- Subscriber : node que se inscreve nesse tópico e recebe as mensagens publicadas nele

```
>>> rostopic list
>>> rostopic info /nome_topico
>>> rostopic echo /nome_topico
>>> rostopic pub /nome_topic [tipo de mensagem] [args]
>>> rostopic find [tipo de mensagem]
```

Exemplo Topics

```
aluno@aluno-VirtualBox:~$ rostopic list
/rosout
/rosout_agg
/turtle1/cmd_vel
/turtle1/color_sensor
/turtle1/pose
aluno@aluno-VirtualBox:~$ rostopic info /turtle1/pose
Type: turtlesim/Pose

Publishers:
 * /turtlesim (http://aluno-VirtualBox:40259/)

Subscribers:
 * /gotogoal (http://aluno-VirtualBox:44325/)


aluno@aluno-VirtualBox:~$ rostopic echo /turtle1/pose
x: 7.74840116501
y: 6.10192584991
theta: -0.635185301304
linear_velocity: 0.0
angular_velocity: 0.0
---
x: 7.74840116501
y: 6.10192584991
theta: -0.635185301304
linear_velocity: 0.0
angular_velocity: 0.0
---
```

Messages

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

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

Exemplo Messages

```
aluno@aluno-VirtualBox:~$ rosmsg list
trajectory_msgs/MultiDOFJointTrajectory
trajectory_msgs/MultiDOFJointTrajectoryPoint
turtle_actionlib/ShapeAction
turtle_actionlib/ShapeActionFeedback
turtle_actionlib/ShapeActionGoal
turtle_actionlib/ShapeActionResult
turtle_actionlib/ShapeFeedback
turtle_actionlib/ShapeGoal
turtle_actionlib/ShapeResult
turtle_actionlib/Velocity
turtlesim/Color
turtlesim/Pose
uuid_msgs/UniqueID
visualization_msgs/ImageMarker
visualization_msgs/InteractiveMarker
visualization_msgs/InteractiveMarkerControl
aluno@aluno-VirtualBox:~$ rosmsg show turtlesim/Pose
float32 x
float32 y
float32 theta
float32 linear_velocity
float32 angular_velocity
```

Comunicação Síncrona

Services

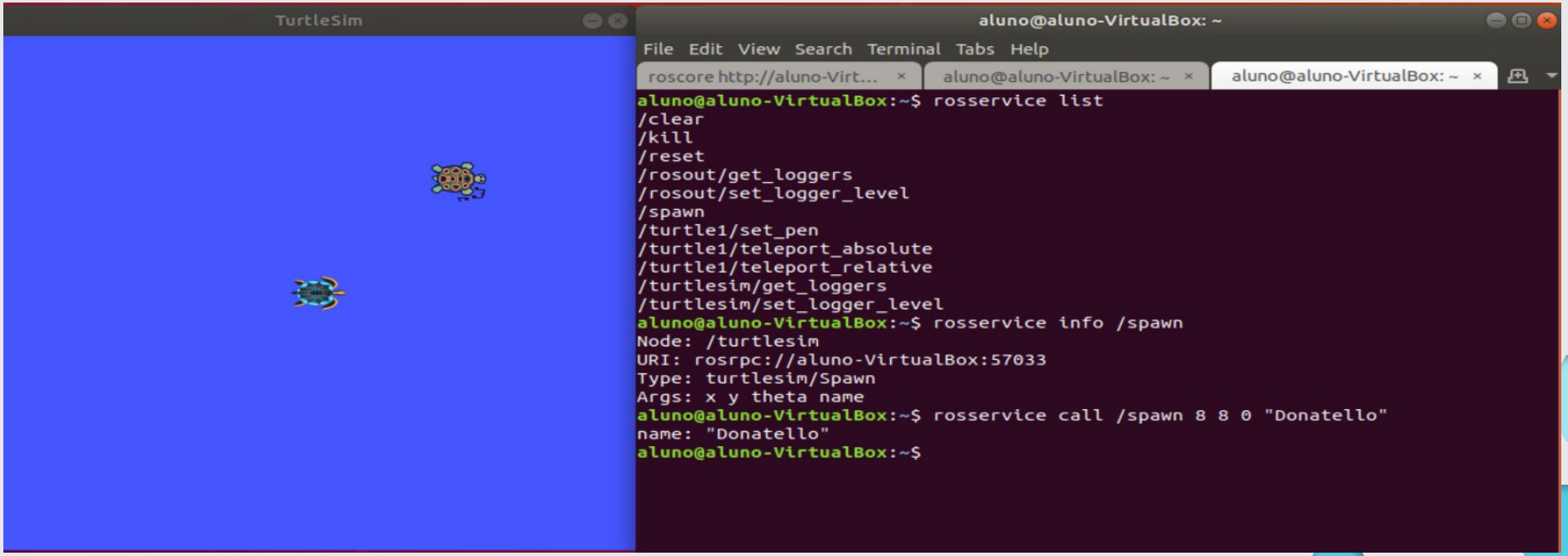
São as chamadas diretas de um node para o outro

- Request : mensagem que o node “cliente” realiza para o servidor
- Response : é a mensagem que ele retorna

```
>>> rosservice list
>>> rosservice info /nome_service
>>> rosservice call /nome_service [args]
```

```
############ Services #################
self.arm = rospy.ServiceProxy(mavros_arm, CommandBool)
self.set_mode_srv = rospy.ServiceProxy(mavros_set_mode, SetMode)
```

Exemplo Services



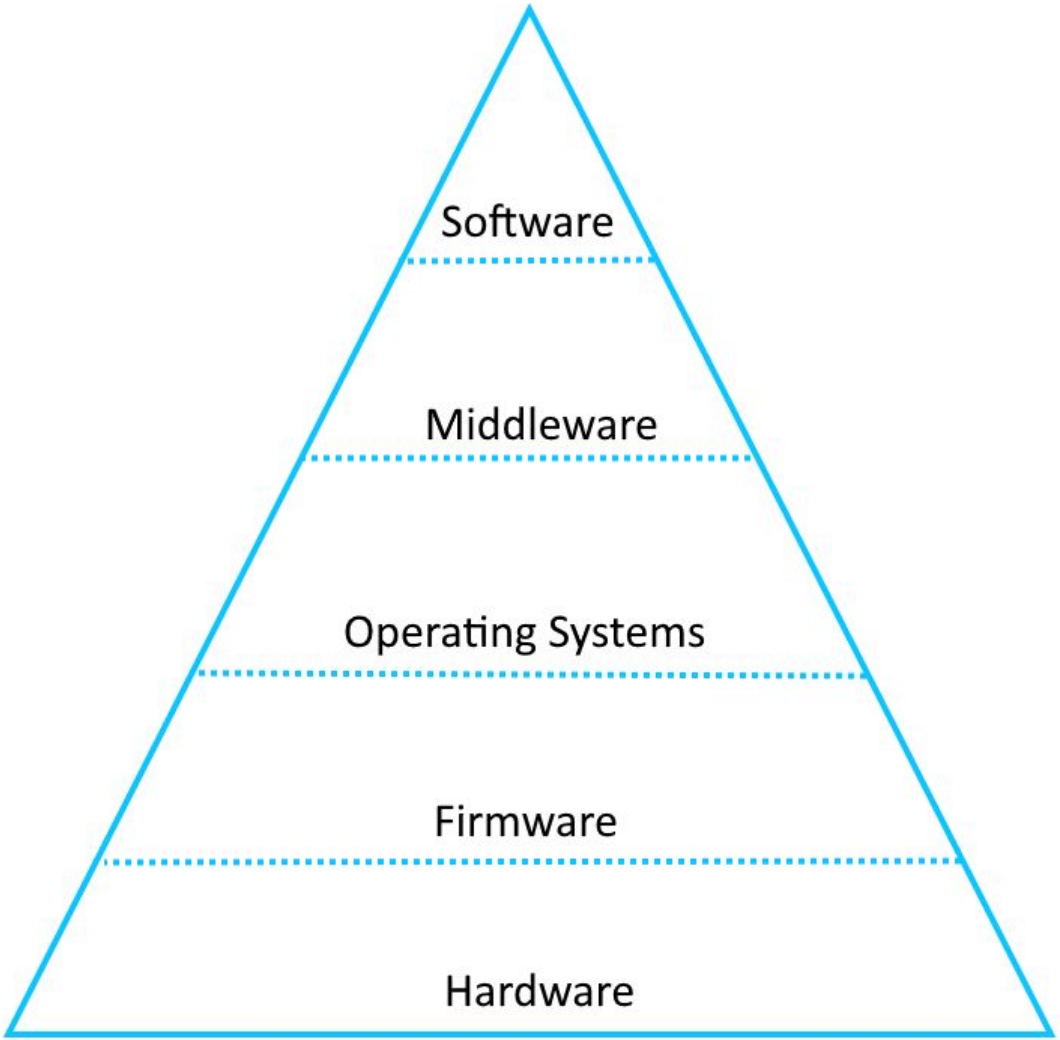
Definição formal de ROS

ROS: ROBOTIC OPERATING SYSTEM (Middleware)

ROS is:

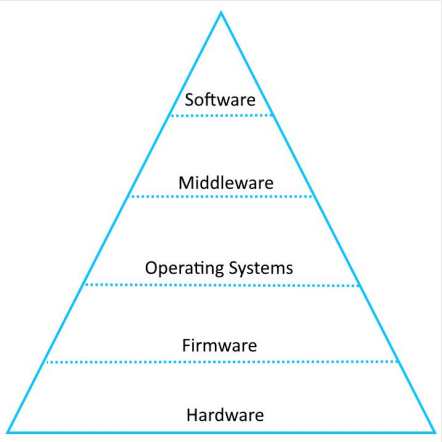
Robot Operating System (**ROS or ros**) is an open-source robotics middleware suite. Although ROS is not an operating system but a collection of software frameworks for robot software development, it provides services designed for a heterogeneous computer cluster such as hardware abstraction, low-level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

fonte: wikipedia


Software
Middleware
Operating Systems
Firmware
Hardware

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”.

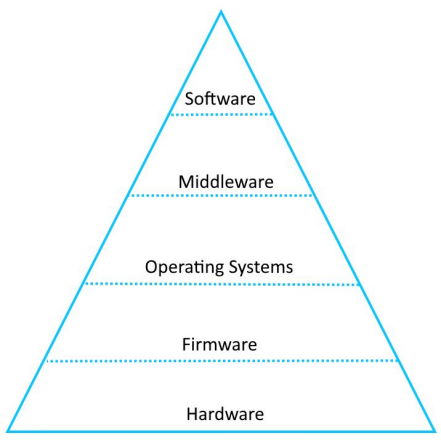


fonte: wikipedia

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.



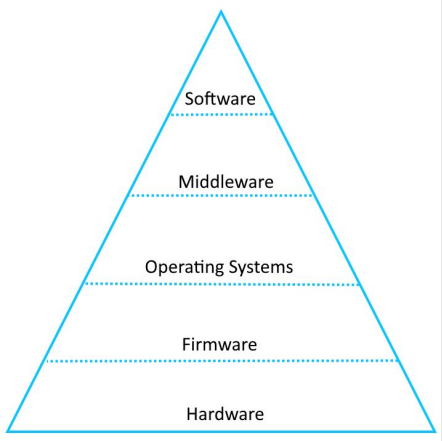
fonte: wikipedia

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.

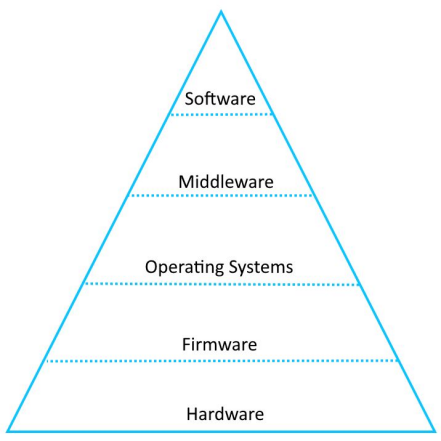


fonte: wikipedia

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.



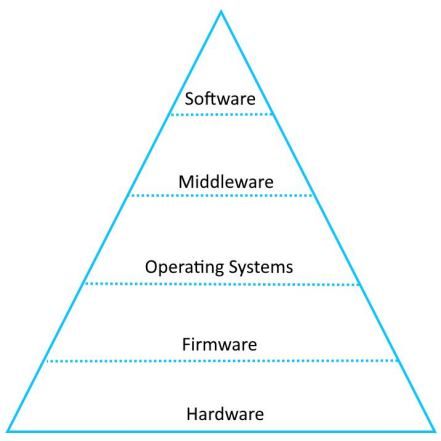
fonte: wikipedia

ROS: ROBOTIC OPERATING SYSTEM (Middleware)

Hardware:

Computação e eletrônica

- Hardware de computador: partes físicas de um computador;
- Eletrônica digital: eletrônica que opera em sinais digitais;
- Componente eletrônico: dispositivo em um sistema eletrônico usado para alterar dinâmicas de tensão ou corrente elétricas;
- Hardware eletrônico: componentes eletrônicos interconectados que executam operações analógicas e/ou digitais;
- Hardware de rede: dispositivos que permitem o uso de uma rede de computadores;
- Aceleração de hardware: a aceleração de tarefas de computação executando-as em hardware personalizado em vez de software;

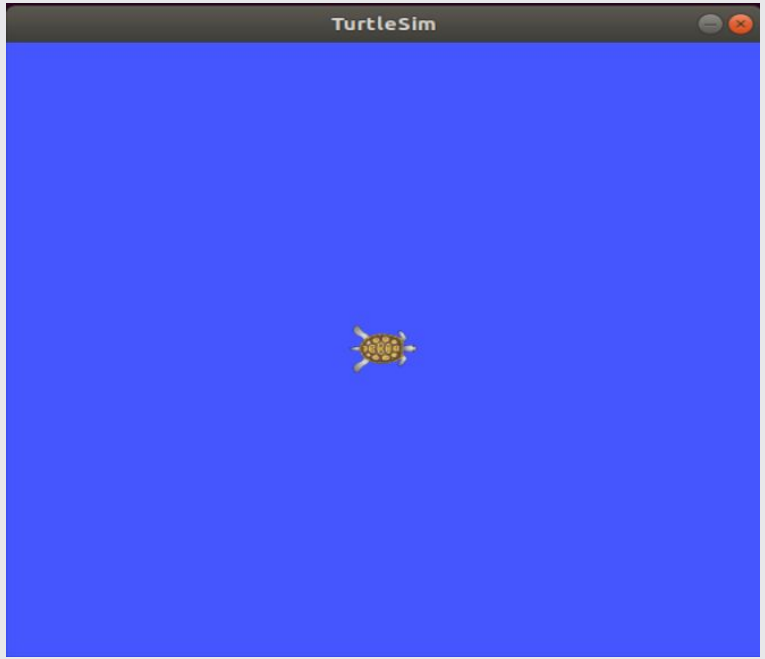


fonte: wikipedia

Exemplo TurtleSim

Exemplo TurtleSim

Vamos construir um script **go_pose** para mandar a tartaruga do turtlesim para uma posição desejada usando **comunicação assíncrona.**



Exemplo TurtleSim

Configurar do zero:

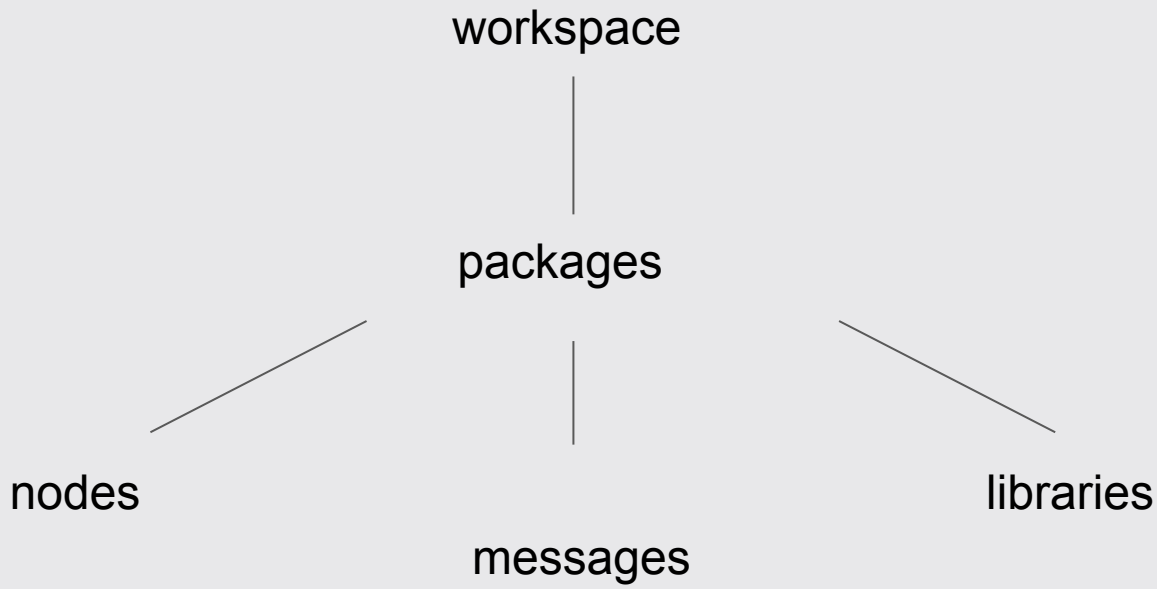
1) Crie pasta **turtle_ws** para ser o seu workspace
2) Crie uma pasta **src** dentro de **turtle_ws**
3) Use o comando **catkin build -j2** no diretório **turtle_ws** para definir o workspace
4) Adicione um script.py na pasta **src** e escreva nele o seu código
5) Repita o 3 passo
6) Execute o programa (veremos como fazer isso a seguir)

Exemplo TurtleSim

Criando package

```
aluno@aluno-VirtualBox:~/ros_workspaces/turtle_ws/src$ catkin_create_pkg turtle_control rospy roscpp
Created file turtle_control/CMakeLists.txt
Created file turtle_control/package.xml
Created folder turtle_control/include/turtle_control
Created folder turtle_control/src
Successfully created files in /home/aluno/ros_workspaces/turtle_ws/src/turtle_control. Please adjust the values in package.xml.
aluno@aluno-VirtualBox:~/ros_workspaces/turtle_ws/src$ ls
turtle_control
aluno@aluno-VirtualBox:~/ros_workspaces/turtle_ws/src$ cd turtle_control/
aluno@aluno-VirtualBox:~/ros_workspaces/turtle_ws/src/turtle_control$ ls
CMakeLists.txt  include  package.xml  src
aluno@aluno-VirtualBox:~/ros_workspaces/turtle_ws/src/turtle_control$ ls
CMakeLists.txt  include  package.xml  src
aluno@aluno-VirtualBox:~/ros_workspaces/turtle_ws/src/turtle_control$ mkdir scripts
aluno@aluno-VirtualBox:~/ros_workspaces/turtle_ws/src/turtle_control$ ls
CMakeLists.txt  include  package.xml  scripts  src
aluno@aluno-VirtualBox:~/ros_workspaces/turtle_ws/src/turtle_control$ ls
CMakeLists.txt  include  package.xml  scripts  src
aluno@aluno-VirtualBox:~/ros_workspaces/turtle_ws/src/turtle_control$ cd scripts/
aluno@aluno-VirtualBox:~/ros_workspaces/turtle_ws/src/turtle_control/scripts$ ls
aluno@aluno-VirtualBox:~/ros_workspaces/turtle_ws/src/turtle_control/scripts$ ls
turtle_go_topic.py
aluno@aluno-VirtualBox:~/ros_workspaces/turtle_ws/src/turtle_control/scripts$ chmod +x turtle_go_topic.py
aluno@aluno-VirtualBox:~/ros_workspaces/turtle_ws/src/turtle_control/scripts$ ls
turtle_go_topic.py
```

ROS WORKSPACE



Workspace

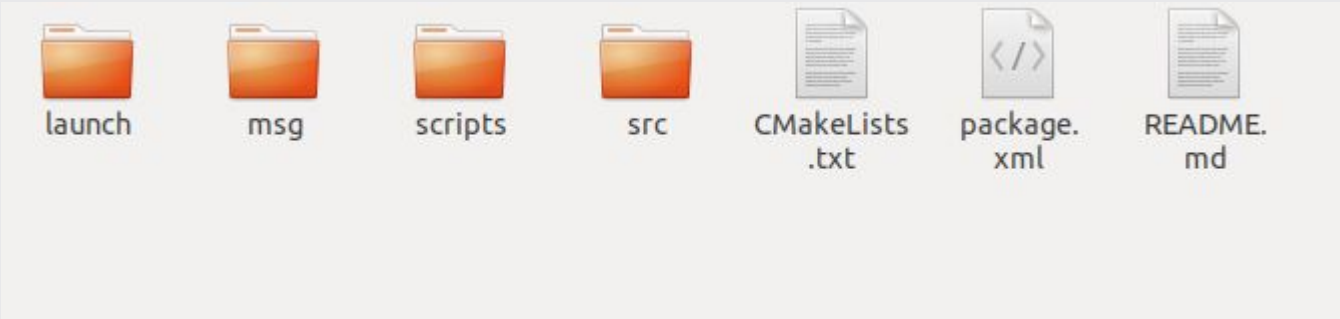
Catkin workspace é o local onde todos os diferentes packages do ROS serão guardados, modificados e compilados

- **src** : onde ficam todos os códigos compiláveis desenvolvidos
- **build** : guarda as informações e arquivos intermediários de compilação do cmake
- **devel**: resultados da compilação
- **install**: programas podem ser instalados utilizando o make install

Packages

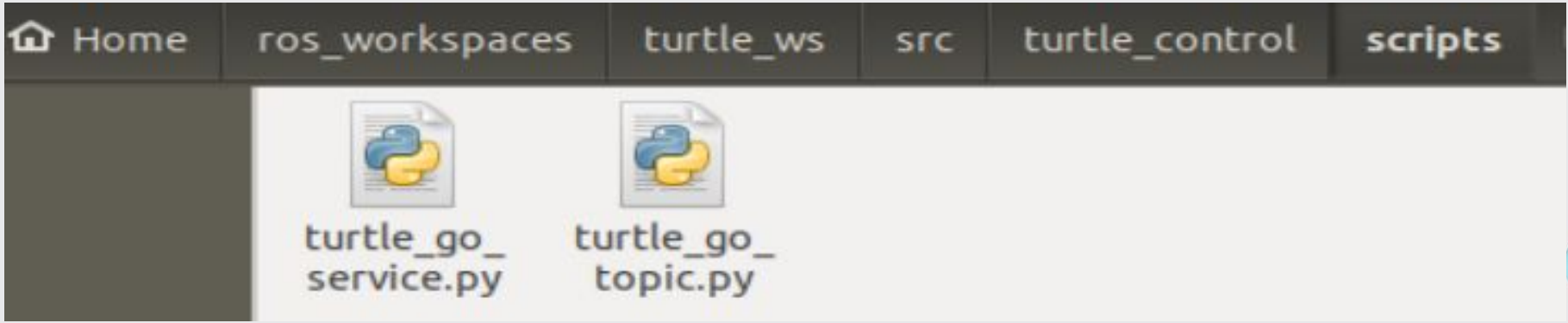
Método de organizar e separar códigos com funções diferentes

No geral, cada package é independente dos outros, porém é possível importar funções de um package para outro.



Exemplo de Package

Exemplo do packge turtle_control



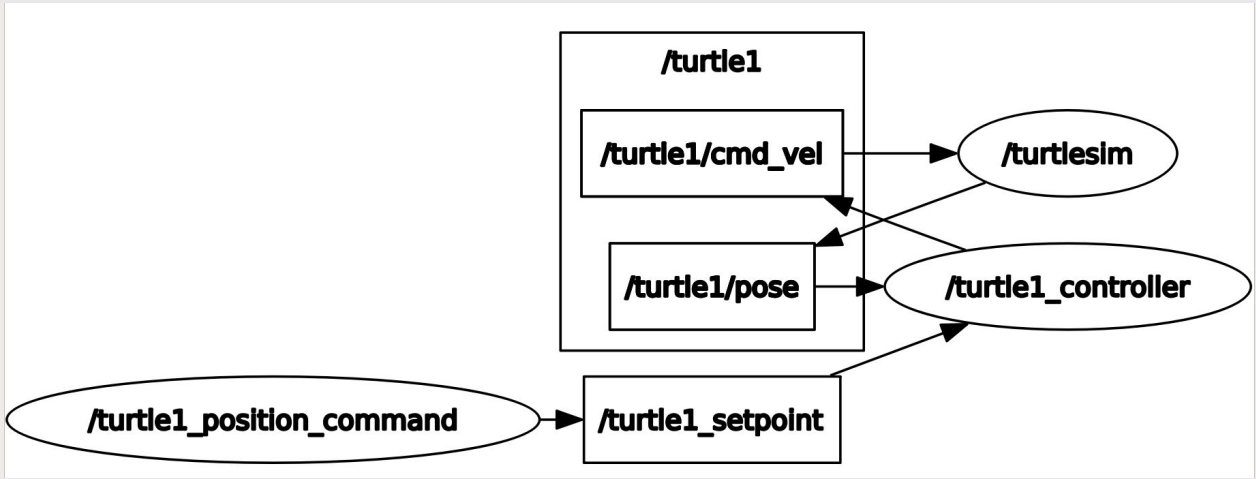
Recapitulando

- Sistema Embarcado
- O que é um Sistema Ciber-físico?
- Exemplo: Drone
  - Por que um drone é complexo?
  - Estados
  - Como funciona a movimentação?
  - Como se faz a estimação de estados?
  - Quais os componentes de Hardware de um drone?
  - Quais elementos de software vamos usar?
  - Quem sabe outra aplicação de drones?

Resumo do ROS

Resumo

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



Como Utilizar o ROS

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

Exemplos:

C++

Estratégia Polar com ajuste angular dinâmico:

Acertar a posição angular do vetor posição entre a posição atual e a posição objetivo enquanto se desloca para frente.

A trajetória esperada é curva

VANTAGEM:
O robô segue a trajetória “olhando para frente”. Isso é desejável caso o robô tenha uma câmera ou algum sensor frontal de parada / desvio de obstáculos.

DESVANTAGEM:
O fato da estratégia envolver o cálculo de ângulos implica que a precisão máxima possível de ser alcançada é menor.

C++

```cpp
#include "turtlesim_control/position_control.h"

TurtleController::TurtleController(ros::NodeHandle* nh, std::string turtle_name, int freq) {
    this->nh = nh; //ptr
    this->turtle_name = turtle_name;
    this->freq = freq;

    // Checks parameters
    if (!nh->hasParam("/" + turtle_name + "_controller/eps_distance")) {
        nh->setParam("/" + turtle_name + "_controller/eps_distance", 0.1);
    } if (!nh->hasParam("/" + turtle_name + "_controller/eps_theta")) {
        nh->setParam("/" + turtle_name + "_controller/eps_theta", 0.1);
    } if (!nh->hasParam("/" + turtle_name + "_controller/Kp_distance")) {
        nh->setParam("/" + turtle_name + "_controller/Kp_distance", 1.0);
    } if (!nh->hasParam("/" + turtle_name + "_controller/Kp_theta")) {
        nh->setParam("/" + turtle_name + "_controller/Kp_theta", 1.0);
    }

    // Creates publishers
    velocity_pub = nh->advertise<geometry_msgs::Twist>("/" + turtle_name + "/cmd_vel", 1000);

    // Creates subscribers
    pose_sub = nh->subscribe(turtle_name + "/pose", 1000, &TurtleController::poseCallback, this);
    setpoint_sub = nh->subscribe<turtlesim::Pose>(turtle_name + "_setpoint", 1000, &TurtleController::setPointCallback, this);

    ros::spinOnce();
}
```

C++

```
29  void TurtleController::run() {
30      ROS_INFO("%s controller started!", turtle_name.c_str());
31
32      ros::Rate rate(freq);
33      while (ros::ok()) {
34          // Update parameters
35          nh->param<float>("/" + turtle_name + "_controller/eps_distance", epsilon_distance, 0.001);
36          nh->param<float>("/" + turtle_name + "_controller/eps_theta", epsilon_theta, 0.001);
37          nh->param<double>("/" + turtle_name + "_controller/Kp_distance", Kp_distance, 1.0);
38          nh->param<double>("/" + turtle_name + "_controller/Kp_theta", Kp_theta, 1.0);
39
40          // Control command
41          updateVel();
42
43          ros::spinOnce();
44          rate.sleep();
45      }
46
47      ROS_INFO("%s controller stopped!", turtle_name.c_str());
48  }
```

C++

```
50  void TurtleController::updateVel() {
51      float distance = sqrt(pow(goal_pose.x - pose.x, 2) + pow(goal_pose.y - pose.y, 2));
52      float goal_theta = calculateAngle();
53      float theta = pose.theta;
54
55      double v = Kp_distance*distance;
56      double omega = Kp_theta*abs(theta - goal_theta);
57
58      if (go) {
59          // If goal position has been reached, stop
60          if (distance < epsilon_distance) {
61              ROS_INFO("Position reached!");
62              vel.linear.x = 0.0;
63              vel.angular.z = 0.0;
64              go = false;
```

C++

```
        } else {
            // If the angle is correct, stop rotating and move forwards
            if (abs(theta - goal_theta) < epsilon_theta) {
                vel.linear.x = v;
                vel.angular.z = 0.0;
            }
            // If angle is not yet correct, rotate
            else {
                vel.linear.x = 0.0;

                if (theta < goal_theta) {
                    if (abs(theta - goal_theta) < M_PI) {
                        vel.angular.z = omega;
                    } else {
                        vel.angular.z = -omega;
                    }
                } else {
                    if (abs(theta - goal_theta) < M_PI) {
                        vel.angular.z = -omega;
                    } else {
                        vel.angular.z = omega;
                    }
                }
            }
        }

        velocity_pub.publish(vel);
    }
}
```

C++

```
95  float TurtleController::calculateAngle() {
96      return atan2(goal_pose.y - pose.y, goal_pose.x - pose.x);
97  }
98
99  void TurtleController::poseCallback(const turtlesim::Pose::ConstPtr& msg) {
100     pose = *msg;
101 }
102
103 void TurtleController::setPointCallback(const turtlesim::Pose::ConstPtr& msg) {
104     goal_pose = *msg;
105     go = true;
106     ROS_INFO("Setpoint received at (%f, %f)", goal_pose.x, goal_pose.y);
107 }
108
109 int main(int argc, char **argv) {
110
111     std::string turtle_name = "turtle1";
112     int freq = 100;
113
114     ROS_INFO("Running turtlesim position control");
115
116     ros::init(argc, argv, turtle_name + "_controller");
117     ros::NodeHandle nh;
118
119     TurtleController *c = new TurtleController(&nh, turtle_name, freq);
120
121     c->run();
122
123     return 0;
124 }
```

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)
```

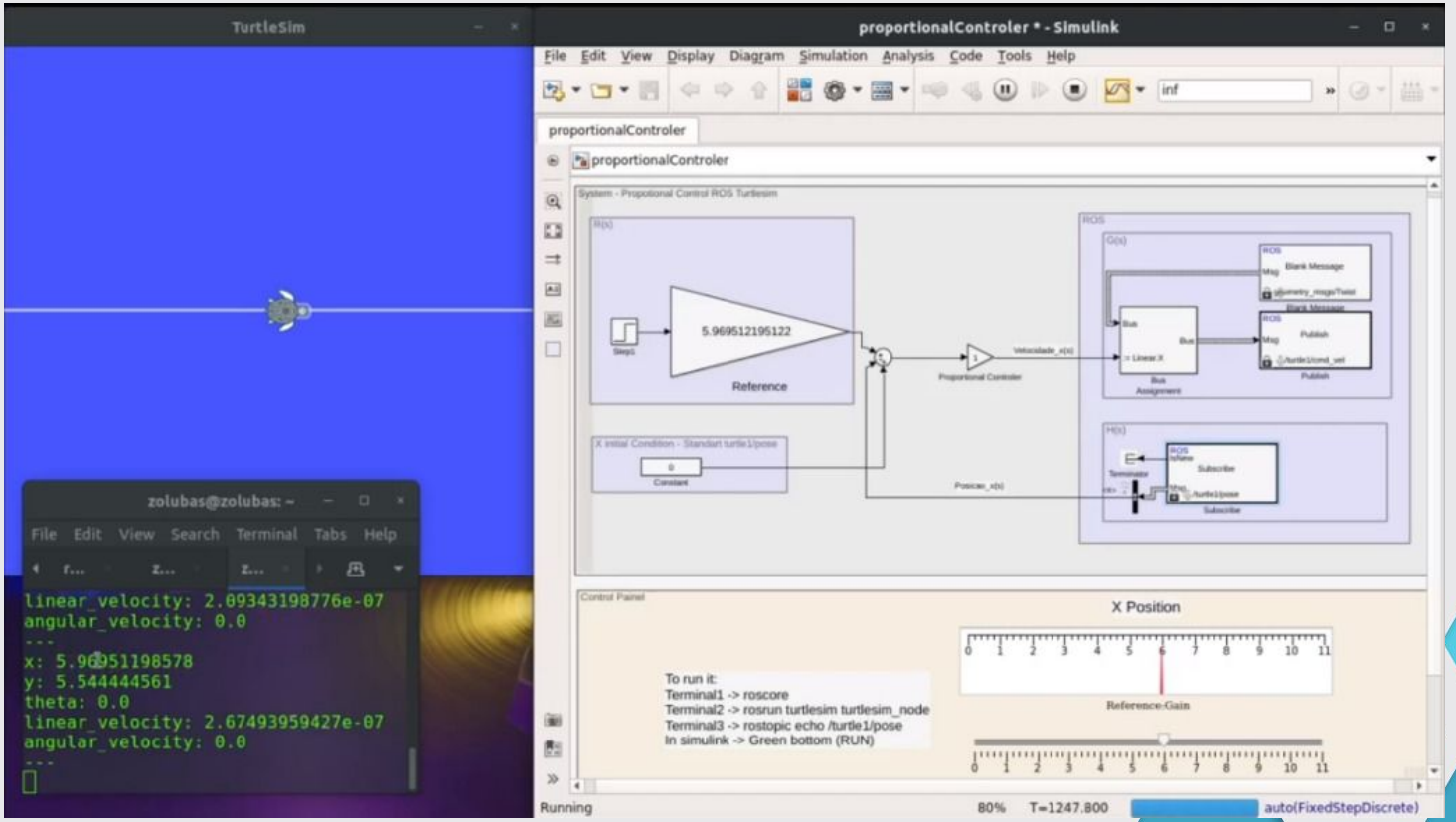
Python

```
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.

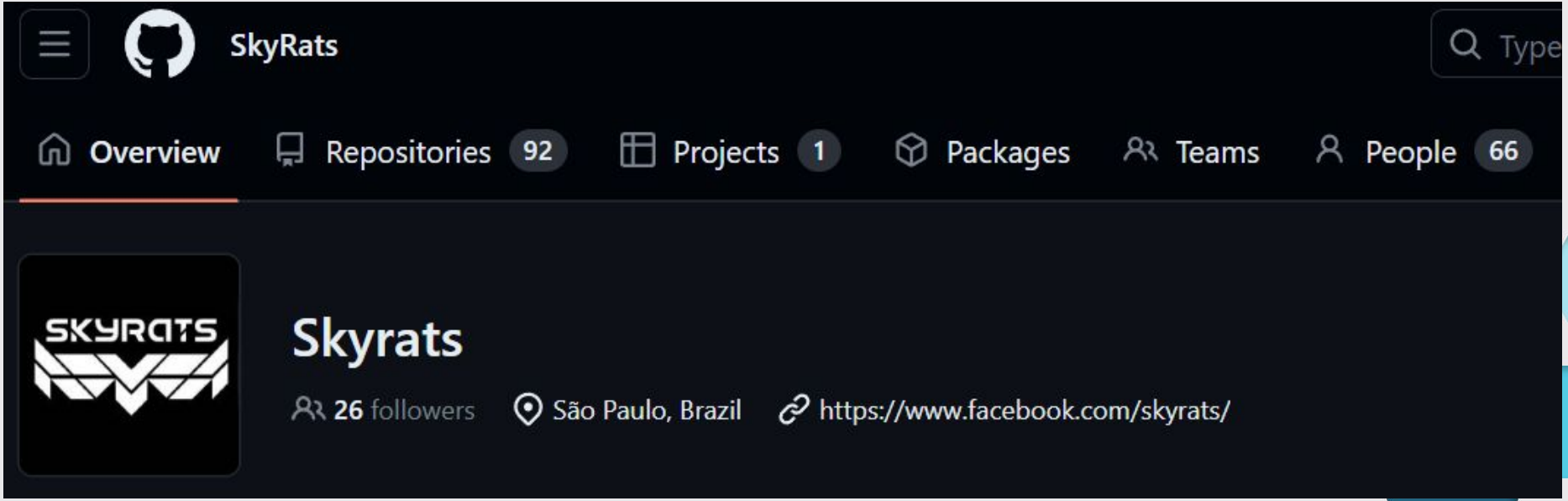
Aplica-se um controle unidimensional em x.



Fim

Extra: Conteúdo Skyrats no github

https://github.com/SkyRats/psi3442/tree/master/Curso2023/3a_Aula



Extra: Launch

Serve para rodar diversos nodes ao mesmo tempo

- compila o workspace
- dá source no setup.bash

```
>>> roslaunch package_name file.launch
```

src > dummie_recognition > launch > control.launch

```
<launch>
    <rosparam command="load" file="$(find mavbase)/config/mavros_params.yaml"/>
    <param name="/vel_topic" value="/mavros/setpoint_velocity/cmd_vel"/>
    <node pkg="dummie_recognition" name="detection_control" type="control.py" output="screen"/>
    <node pkg="dummie_recognition" name="dummie_detection" type="dummie_detection_node" output="screen"/>

</launch>
```

Extra: Parameter server

- Parameter Server : usado para definir nomes, opções e configurações ao executar um programa

src > mavbase > config > ! mavros_params.yaml

```
1   mavros_local_position_pub: '/mavros/setpoint_position/local'
2   mavros_velocity_pub : '/mavros/setpoint_velocity/cmd_vel'
3   mavros_local_atual :  '/mavros/local_position/pose'
4   mavros_state_sub :    '/mavros/state'
5   mavros_arm :          '/mavros/cmd/arming'
6   mavros_set_mode :     '/mavros/set_mode'
7   mavros_battery_sub :  '/mavros/battery'
8   extended_state_sub:   '/mavros/extended_state'
9   mavros_global_position_sub: '/mavros/global_position/global'
10  mavros_pose_target_sub:  '/mavros/setpoint_raw/local'
11  mavros_set_global_pub: '/mavros/setpoint_position/global'
```

Extra: Outras funções úteis

- Rosbag : grava e reproduz mensagens ROS
- rviz : vizualização 3D pro ROS

```
>>> rosbag record -a
>>> rosbag play [nome da rosbag].bag

>>>rosrun rviz rviz
```