AULA 3

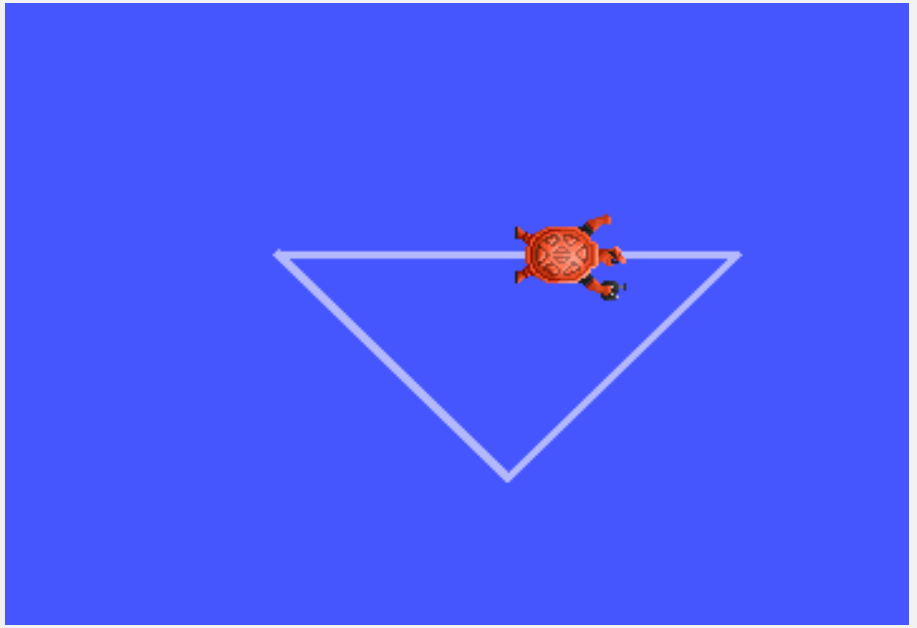
Mensagens, Parâmetros e Launch

NA AULA DE HOJE...

- Conceitos básicos de ROS (continuação)
  - Parâmetros
    - Conceitos, uso em nós
    - Arquivos de parâmetros
  - Mensagens
    - Como são estruturadas?
    - Alguns tipos mais usados (std_msgs, geometry_msgs, nav_msgs)
    - Como construir as próprias mensagens
  - Arquivos de Launch

REVISÃO DESAFIO DA ÚLTIMA AULA

- Fazer um nó que publique comandos para o turtle_sim andar em um trajetória triangular
- Solução: Disponível em https://github.com/nycolasRamires/ros2_ufc/
  - Máquina de estados:
    - FORWARD
    - STOP_FORWARD
    - TURN
    - STOP_TURN
  - Contador de voltas para incrementar a velocidade



PARÂMETROS EM NÓS

- Configuração qualquer que podemos definir na hora de chamar um nó
  - Por intermédio de instrumentar o nó para receber isso

```
// no construtor
this->declare_parameter<std::string>("robot_name", "rover");
this->declare_parameter<int>("speed", 5);
```

- Usando o parâmetro

```
ros2 run pkg_name node_exe_name --ros-args -p robot_name:=rob -p speed:=10
```

PARÂMETROS EM NÓS

- Acessando o valor dentro do código:

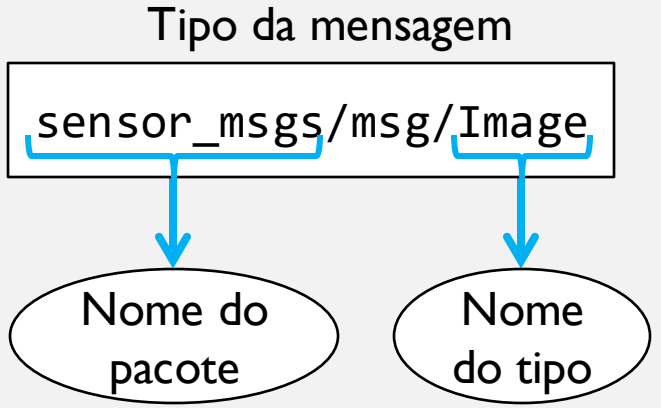
```
robot_name_ = this->get_parameter("robot_name").as_string();
speed_ = this->get_parameter("speed").as_int();
```

- Podemos salvar um conjunto de parâmetros para alguns nós em um arquivo de configuração:
  - Formato .yaml
  - Permite ter várias configurações salvas para os nossos nós

```
turtle_triangle:
  ros__parameters:
    pub_period_ms: 5
    angles_deg: [90.0, 143.1, 126.9]
    side_lengths: [4.0, 5.0, 3.0]
    speedup: 0.2
    initial_velocity: 1.0
    turtle_name: "tortuguita"
    use_sim_time: false
```

EXPLORANDO MENSAGENS

- Mensagens são as estruturas de dados bem definidas usadas para trocar informação entre nós
  - Como um contrato. Se um tópico contém mensagens do tipo `sensor_msgs/msg/Image`, qualquer nó que assine esse tópico sabe o que esperar.
- Definidas em `.msg` a partir de tipos básicos ou outras mensagens
- Pode-se criar mensagens personalizadas
  - Agnóstico a linguagem de programação



EXPLORANDO MENSAGENS

Pacote	Funcionalidade
std_msgs	Tipos primitivos, `Header`, sinal vazio (`Empty`), cor RGBA.
geometry_msgs	Operações espaciais: Pontos, vetores, poses, transformações, inércia, aceleração.
sensor_msgs	Sensores: imagens, lasers, IMU, nuvens de pontos, temperatura etc.
nav_msgs	Navegação: posição estimada, trajetórias, mapas, caminhos.
diagnostic_msgs	Informação de estado e diagnóstico de componentes e nós.

https://index.ros.org/?search_packages=true

https://github.com/ros2/common_interfaces

STD_MSGS

- Mensagens padrão do ROS, tipos primitivos

Int
Bool
Byte
Float(32,64)

Multiarray (para vários tipos, e.g. Float64MultiArray)
Char
String
Header

- Podemos ver todas as interfaces do pacote com:

```
ros2 interface package std_msgs
```

```
std_msgs/msg/ByteMultiArray
std_msgs/msg/String
std_msgs/msg/UInt16
std_msgs/msg/Int8
std_msgs/msg/Int64
std_msgs/msg/Bool
std_msgs/msg/Float32MultiArray
std_msgs/msg/Int8MultiArray
std_msgs/msg/UInt64
std_msgs/msg/Byte
std_msgs/msg/Empty
std_msgs/msg/UInt16MultiArray
std_msgs/msg/UInt64MultiArray
std_msgs/msg/Int32MultiArray
std_msgs/msg/ColorRGBA
```

```
std_msgs/msg/Float64MultiArray
std_msgs/msg/Int16
std_msgs/msg/UInt8MultiArray
std_msgs/msg/UInt32MultiArray
std_msgs/msg/Header
std_msgs/msg/Int64MultiArray
std_msgs/msg/UInt8
std_msgs/msg/Int32
std_msgs/msg/Float32
std_msgs/msg/Float64
std_msgs/msg/MultiArrayDimension
std_msgs/msg/MultiArrayLayout
std_msgs/msg/Char
std_msgs/msg/UInt32
std_msgs/msg/Int16MultiArray
```

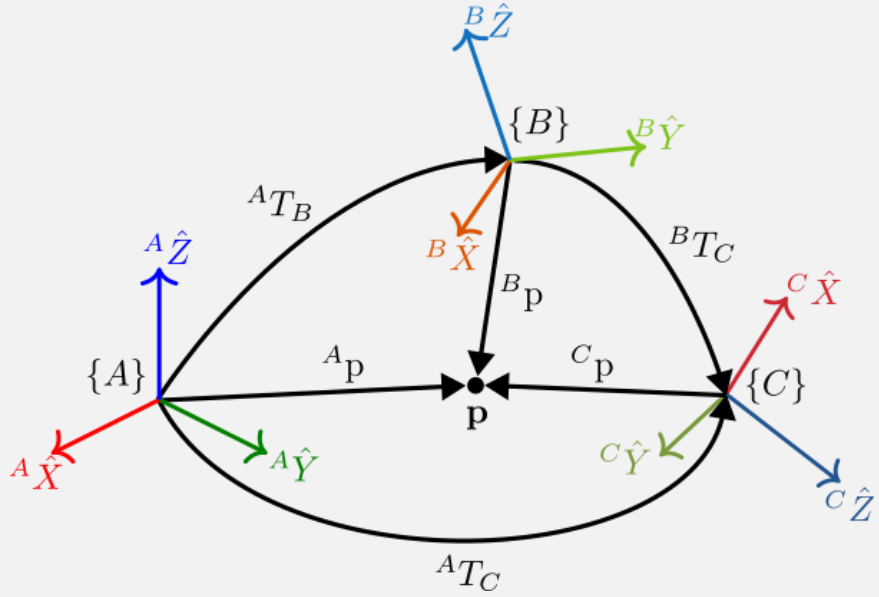
Resultado do comando. Total de 30 tipos básicos.

GEOMETRY_MSGS

- Mensagens de pose, orientação, e relacionadas ao movimento dos robôs. Definem, para um robô, o "onde" e "como"

Inertia	Point
Accel	Vector
Twist	Pose
Wrench	Quaternion
	Transform

- Frequentemente são usadas junto com o subsistema tf2
- Vamos explorar alguns desses termos na próxima aula...



Três sistemas de coordenadas, um ponto, e as relações entre eles.

https://docs.ros.org/en/noetic/api/geometry_msgs/html/index-msg.html

SENSOR_MSGS

- Mensagens relacionada a sensores do robô (“*duh*”)

JointState	LaserScan
PointCloud2	Joy
Imu	Image
Temperature	BatteryState
Range	

Exemplo de um LIDAR (SICK LMS291), tipo de sensor comum em robótica móvel

- Vários tipos de sensores mais comuns já estão definidos: https://docs.ros2.org/foxy/api/sensor_msgs/index-msg.html
  - Eventualmente, alguma fabricante pode ter suas próprias mensagens para seus sensores, e.g. velodyne_msgs (mensagens para LIDARs da velodyne)

Câmera de profundidade RealSense D455

CRIANDO MENSAGENS

- Assim como nós, as mensagens devem ser criadas dentro de pacotes

```
cd ~/ros2_ws/src
ros2 pkg create --build-type ament_cmake  my_interfaces
mkdir -p my_interfaces/msg
```

  - Note que `-build-type` deve ser `ament_cmake`
    - `ament_python` não tem suporte para gerar interfaces!
- Definir a estrutura da mensagem em um arquivo `.msg`
- Adicionar dependências e descrição em `CMakeLists.txt` e `package.xml`

CRIANDO MENSAGENS

- Assim como nós, as mensagens devem ser criadas dentro de pacotes

```
my_interface/
├── include/
├── src/
├── msg/
│     MinhaMsg.msg
├── CMakeLists.txt
└── package.xml
```

MinhaMsg.msg:

```
string robot_name
int32[] sensor_values          # tam. dinamico
builtin_interfaces/Time stamp # msg composta
float64 battery_voltage
```

EXPLORANDO UM POUCO MAIS MENSAGENS

- Mensagens são definidas por arquivos .msg
  - Usado para gerar o código fonte pelo pacote `rosidl_default_generators`
- Conteúdo do arquivo dita o que a mensagem carrega
- Podemos ter mensagens que carregam estruturas de dados definidas por outras mensagens. Exemplo:

`sensor_msgs/msg/Temperature`, que contém:

- `std_msgs/msg/Header`, que contém:
  - `builtin_interfaces/msg/Time stamp:`
    - `int32 sec`
    - `uint32 nanosec`
  - `string frame_id`
- `double temperature`
- `double variance`

File: `sensor_msgs/msg/Temperature.msg`

Raw Message Definition

```
# Single temperature reading.

#std_msgs/Header header # timestamp is the time the temperature was measured

# # frame_id is the location of the temperature reading

#float64 temperature # Measurement of the Temperature in Degrees Celsius.

#float64 variance # 0 is interpreted as variance unknown.
```

Compact Message Definition

```
std_msgs/msg/Header header
double temperature
double variance
```

A estrutura da mensagem pode ser vista por CLI (`ros2 interface show <message_type>`), mas algumas também estão disponíveis em páginas web.

EXPLORANDO MENSAGENS

- Para expor as nossas mensagens (e eventualmente) para que outros pacotes as possam utilizar, precisamos:
  - Ter um arquivo .msg com a definição delas
  - Chamar o `rosidl_default_generators` no nosso CMakeLists.txt
  - Incluir o `rosidl_default_generators` como dependência de build e runtime no package.xml, e declarar que o nosso pacote faz parte do grupo de dependência `rosidl_interface_packages`

```
# CMakeLists.txt

find_package(geometry_msgs REQUIRED)
find_package(rosidl_default_generators REQUIRED)

rosidl_generate_interfaces(${PROJECT_NAME}
    "msg/MinhaPrimeiraMensagem.msg"
    "msg/MinhaOutraMensagem.msg"
    "srv/UmServicoAquiTambem.srv"
    DEPENDENCIES geometry_msgs
    # Adicione os pacotes que os seus arquivos
    # *.msg/*.srv usam como dependencias!
)
```

Comandos a serem inclusos no `CMakeLists.txt`

```
<!-- package.xml -->

<!-- eventuais dependencias das mensagens -->
<depend>geometry_msgs</depend>

<!-- declara dependencia do buildtool no rosidl, chama runtime -->
<buildtool_depend>rosidl_default_generators</buildtool_depend>
<exec_depend>rosidl_default_runtime</exec_depend>
<member_of_group>rosidl_interface_packages</member_of_group>
```

Declarações a serem inclusas no `package.xml`

ARQUIVOS DE LAUNCH

- Basicamente um script para realizar ações em série
  - Descreve quais nós devem ser lançados, seus parâmetros, remaps e configurações
- Pode ser escrito em Python, XML ou YAML
  - Linguagem oficial para launch no ROS2 é Python
    - Pode usar todas as ferramentas que a linguagem tem
  - Mas XML é tão usado quanto, já que é possível chamar launch.py dentro de launch.xml

```
meu_pacote/
├── package.xml
├── CMakeLists.txt
├── src/
├── launch/
│   └── my_launch.launch.py
```

CRIANDO ARQUIVOS DE LAUNCH

Assim como todo o resto arquivos de launch devem ser criados dentro de pacotes

```
cd ~/ros2_ws/src
ros2 pkg create --build-type ament_cmake pkg_name
mkdir -p pkg_name/launch
```

Adicionar ao CMakeList.txt

```
install(DIRECTORY
  launch
  DESTINATION share/${PROJECT_NAME}/
)
```

CRIANDO ARQUIVOS DE LAUNCH

- Exemplo de um launch em xml

```xml
<!-- Exemplo.launch.xml -->
<launch>
  <arg name="publish_freq" default="10"/>

  <node pkg="<pkg_name1>" exec="<exe_name1>" name="<new_node_name>"/>

  <node pkg="<pkg_name2>" exec="<exe_name2>" name="<another_node>">
    <param name="publish_frequency" value="$(var publish_freq)"/>
  </node>

  <include file="$(find-pkg-share my_pkg)/launch/some_launch_file.xml">
    <arg name="some_argument" value="dummy_value"/>
  </include>
</launch>
```

Definição completa do esquema de XML usado no ROS 2 está disponível online: https://design.ros2.org/articles/roslaunch_xml.html

CRIANDO ARQUIVOS DE LAUNCH

- Crie o arquivo de launch dentro de launch/
  - Convenção <nome>.launch.py ou <nome>.launch.xml
- Para todo pacote que terá algum conteúdo lançado é preciso listar no package.xml

```
<exec_depend>pkg_name</exec_depend>
```

- Usa o launch com

```
ros2 launch <pkg_name> <launch_file_name>
```

CRIANDO ARQUIVOS DE LAUNCH

- Mudando o nome do nó:

```
<!-- Lança 3 nós -->
<launch>
  <node pkg="pkg_name1" exec="exec_name1" name="new_node_name"/>
  <node pkg="pkg_name2" exec="exec_name2"/>
  <node pkg="pkg_name3" exec="exec_name3"/>
</launch>
```

- Mudando o tópico

```
<launch>
  <node pkg="pkg_name1" exec="exec_name1" name="new_node_name">
    <remap from="old_name" to="new_name">
  </node>
</launch>
```

CRIANDO ARQUIVOS DE LAUNCH

- Alterando parâmetros do nó no lançamento

```
<launch>
  <node pkg="<pkg_name1>" exec="<exec_name1>" name="<new_node_name>">
    <param name="<param_name>" value="<param_value>"/>
  </node>
</launch>
```

- É possível olhar e alterar em run time também:
  - `ros2 param get node_name parameter_name`
  - `ros2 param set node_name parameter_name new_value`