



Aula 4

ECA419 - Robótica

Aulas Passadas

- Catkin Workspace
- Pacote (Package)
 - package.xml
 - CMakeLists.txt
 - /src
 - arquivos.cpp
 - /include
 - arquivos.h
- Nó (Node)
 - Publisher (emissor)
 - Subscriber (receptor)



Mensagens (*Messages*)

- Nós se comunicam uns com os outros através da publicação de mensagens em tópicos.
- A mensagem é uma estrutura de dados simples formada por tipos primitivos (inteiro, float, booleano, array, etc).
- ROS possui várias mensagens com nomes e tipos padrões:
 - geometry_msgs/Twist
 - linear.x, linear.y, linear.z, angular.x,...
 - geometry_msgs/Pose2D
 - x, y, theta



Mensagens (*Messages*)

File: `sensor_msgs/LaserScan.msg`

Raw Message Definition

```
# Single scan from a planar laser range-finder
#
# If you have another ranging device with different behavior (e.g. a sonar
# array), please find or create a different message, since applications
# will make fairly laser-specific assumptions about this data

Header header          # timestamp in the header is the acquisition time of
                        # the first ray in the scan.
                        #
                        # in frame frame_id, angles are measured around
                        # the positive Z axis (counterclockwise, if Z is up)
                        # with zero angle being forward along the x axis





float32 angle_min      # start angle of the scan [rad]
float32 angle_max      # end angle of the scan [rad]
float32 angle_increment # angular distance between measurements [rad]

float32 time_increment  # time between measurements [seconds] - if your scanner
                        # is moving, this will be used in interpolating position
                        # of 3d points
float32 scan_time       # time between scans [seconds]

float32 range_min      # minimum range value [m]
float32 range_max      # maximum range value [m]

float32[] ranges        # range data [m] (Note: values < range_min or > range_max should be discarded)
float32[] intensities   # intensity data [device-specific units]. If your
                        # device does not provide intensities, please leave
                        # the array empty.
```

Mensagens (*Messages*)

Primitive Type	Serialization	C++	Python
bool (1)	unsigned 8-bit int	uint8_t (2)	bool
int8	signed 8-bit int	int8_t	int
uint8	unsigned 8-bit int	uint8_t	int (3)
int16	signed 16-bit int	int16_t	int
uint16	unsigned 16-bit int	uint16_t	int
int32	signed 32-bit int	int32_t	int
uint32	unsigned 32-bit int	uint32_t	int
int64	signed 64-bit int	int64_t	long
uint64	unsigned 64-bit int	uint64_t	long
float32	32-bit IEEE float	float	float
float64	64-bit IEEE float	double	float
string	ascii string (4)	std::string	str
time	secs/nsecs unsigned 32-bit ints	 ros::Time	 rospy.Time
duration	secs/nsecs signed 32-bit ints	 ros::Duration	 rospy.Duration



Tópicos (*Topics*)

- Tópicos são barramentos por onde as mensagens trafegam.
- As trocas de informações são **unilaterais**, assim, um nó publisher e outro subscriber são produtor e consumidor de informações, respectivamente.
- Um nó publicador **desconhece** quem subscreve o tópico e vice-versa.
- É um sistema **passivo**, assim, para receber uma resposta de um determinado pedido, não é possível utilizar o tópico. Para isso existem os **serviços**.



Tópicos (*Topics*)

O ROS trabalha com vários tópicos com nomes padrões:

- /cmd_vel → geometry_msgs/Twist
- /scan → sensor_msgs/LaserScan
- /image_raw → sensor_msgs/Image
- /odom → nav_msgs/Odometry
- /sonar → sensor_msgs/PointCloud



Resumindo

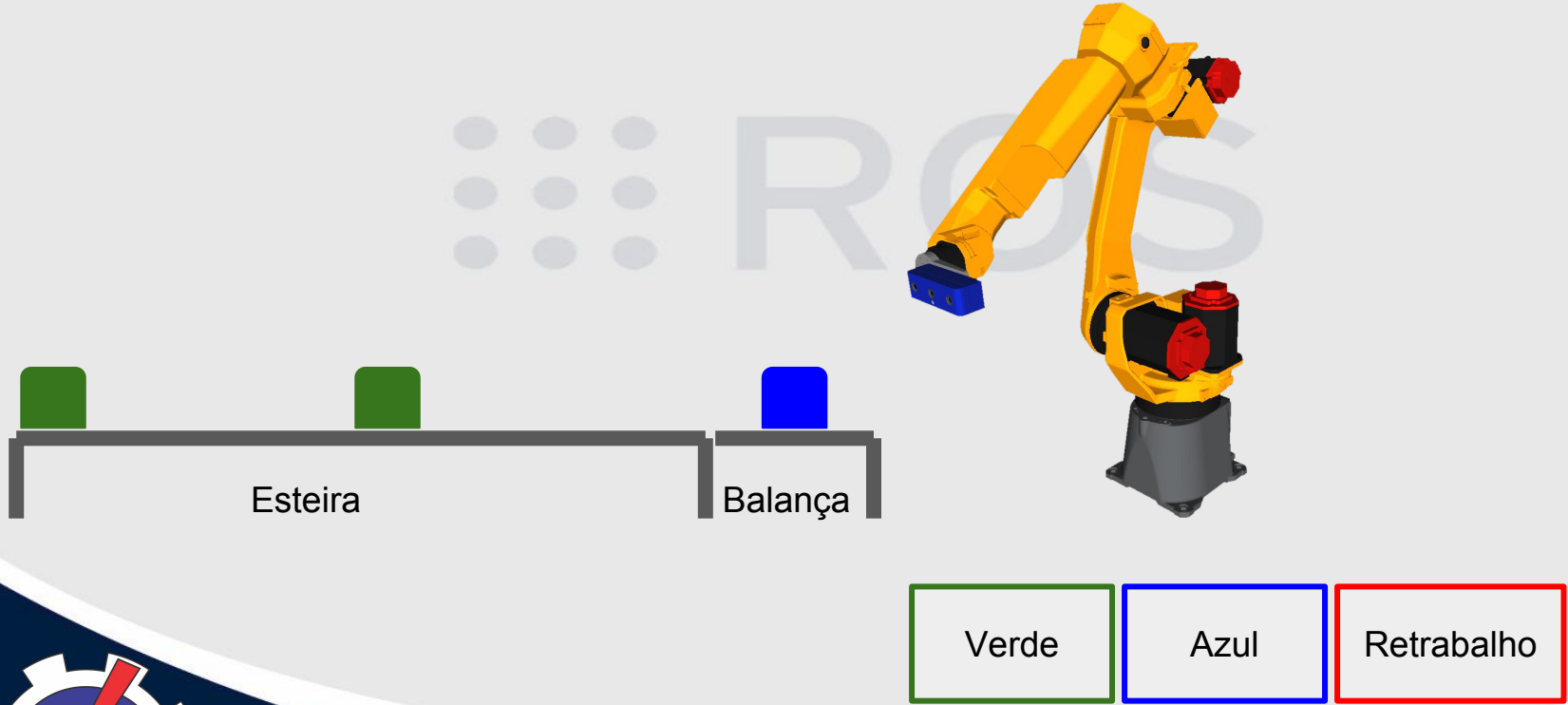


Modelando um Problema

- Suponha que em uma montadora automobilística exista um manipulador com uma câmera acoplada em sua estrutura.
- Sua função é separar e organizar as peças verdes das azuis.
- Todas as peças são pesadas e as que estiverem fora dos padrões da empresa são descartadas utilizando o manipulador.
- A câmera, além de identificar as cores, fornece informações (x,y,z) do centro de massa do objeto.
- Faça uma modelagem do problema utilizando nós, tópicos e mensagens.



Modelando um Problema



Modelando um Problema

balanca

controlador_
manipulador

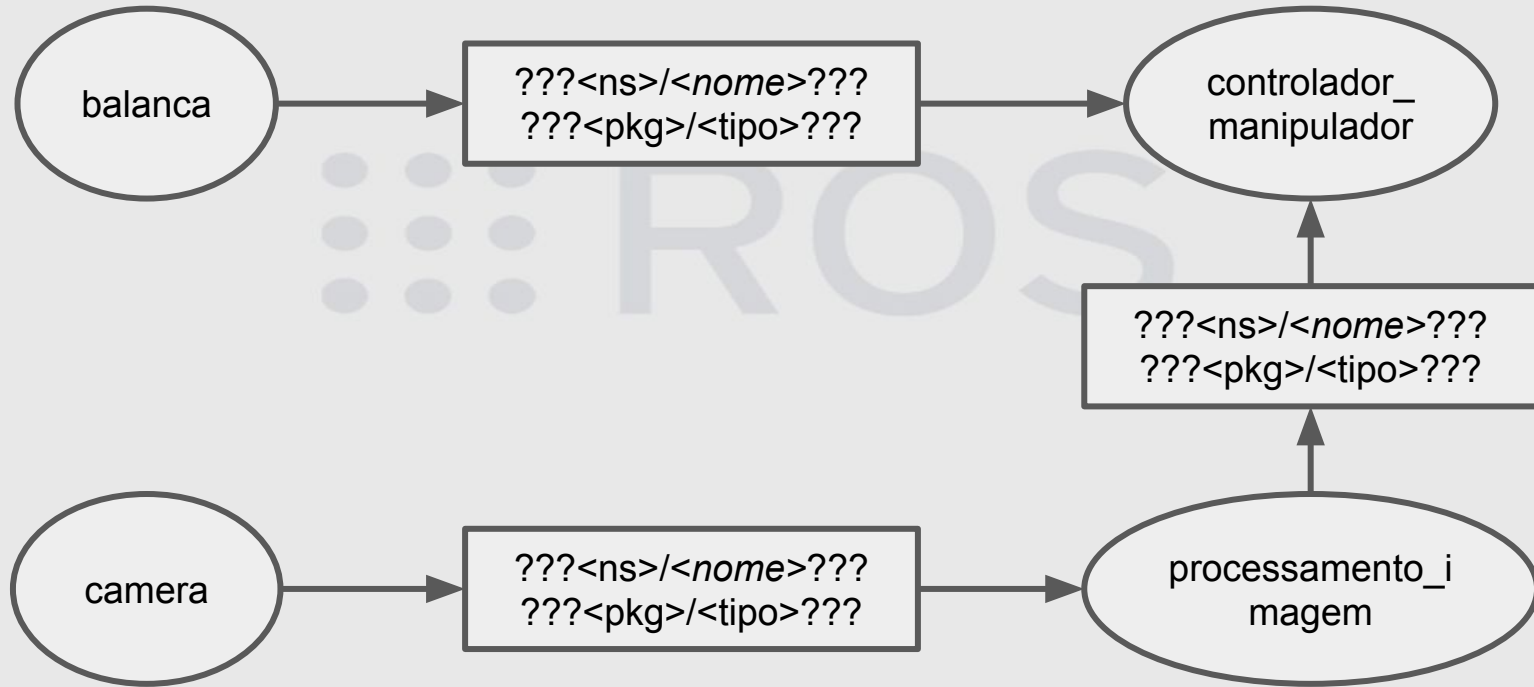
ROS

camera

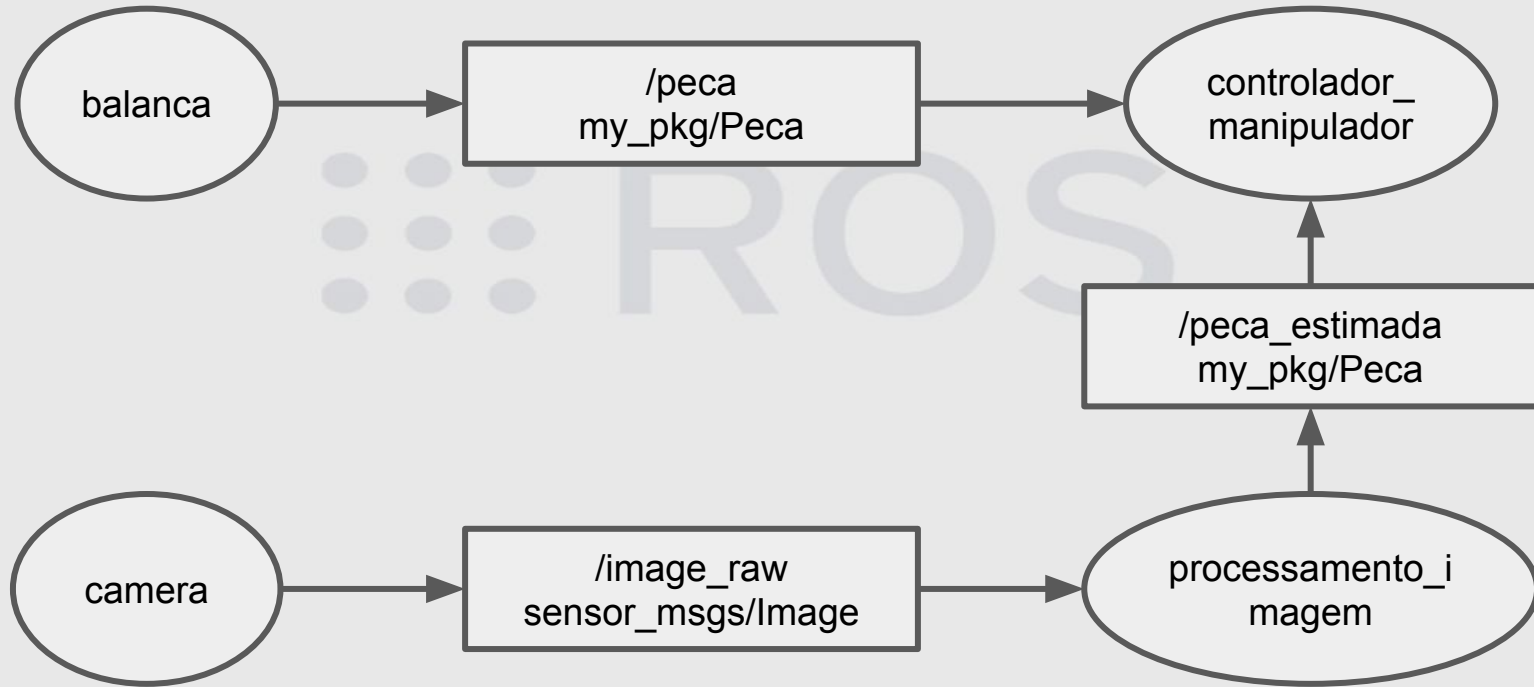
processamento_
imagem



Modelando um Problema



Modelando um Problema



Modelando um Problema

O tipo de mensagem `my_pkg/Peca` é definida pelo arquivo `Peca.msg` que está localizado no pacote `my_pkg`:

<code>uint8 cor</code>	<code># (-1)Desconhecida; (0)Verde; (1)Azul</code>
<code>float32 massa</code>	<code># massa mensurada</code>
<code>geometry_msgs/Point</code>	<code># coordenada geom. estimada</code>



Serviços (Services)

Em qualquer sistema grande baseado em ROS, existem casos em que algum nó gostaria de enviar um **pedido** para um outro nó desempenhar uma determinada **tarefa** e ainda receber uma **resposta** com respeito a sua realização. Para isso utilizamos **serviços** do ROS.Exemplo:

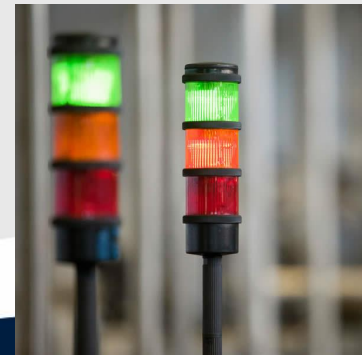
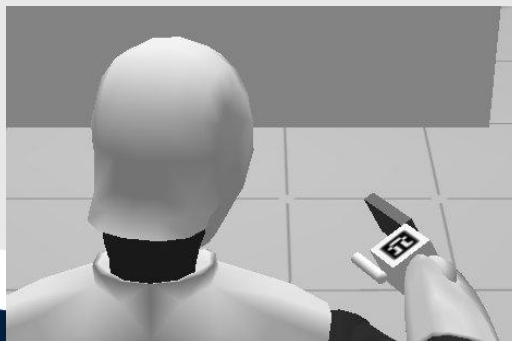
tarefa: (1) identificar distância do objeto de interesse; (2) sinalizar.

pedido: (1) imagem, tipo do objeto e método; (2) cores a ascender.

resposta: (1) distância estimada; (2) *empty*.



<http://wiki.ros.org/Services>



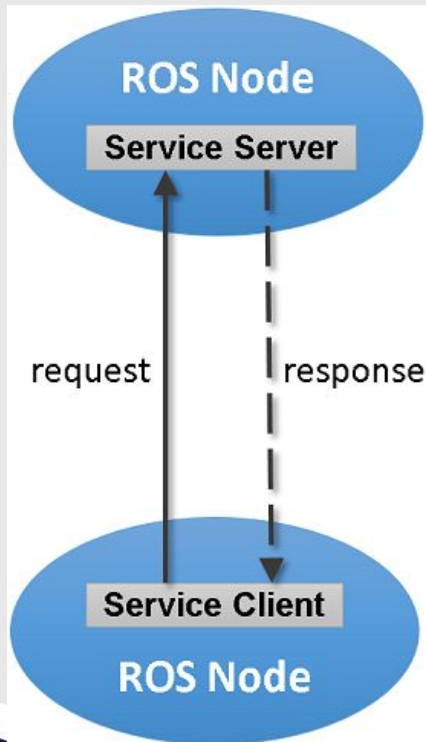
Serviços (*Services*)

É definido por um **par de mensagens**: uma para definir o **pedido** (*request*) e para definir a **resposta** (*response*).

Um nó provedor (ou **servidor**) oferece um serviço sob um nome (assim como nós e tópicos) e um **cliente** usa o serviço ao enviar uma mensagem com seu pedido esperando pela resposta.



Resumindo



Service Name: `/example_service`
Service Type: `roscpp_tutorials/TwoInts`

Request Type: `roscpp_tutorials/TwoIntsRequest`
Response Type: `roscpp_tutorials/TwoIntsResponse`



Tipo de Serviço

Assim como tópicos, serviço são tipados. Isto é, as mensagens pedido e resposta compõem um arquivo .srv que formaliza a utilização do serviço.

Podemos definir um serviço que simplesmente adiciona dois inteiros da seguinte forma:

AddTwoIntegers.srv

```
int64 a      # comentários pertinentes sobre a
int64 b      # comentários pertinentes sobre b
---
int64 result      # result = a + b
```

Mensagem Pedido

Mensagem Resposta



Arquivo .srv

A partir deste arquivo, classes são geradas automaticamente em C++ (e outras linguagens de programação) durante o processo de compilação do pacote que ele pertence.

Por este fato, sempre que um novo arquivo .srv for criado, o arquivo **CMakeLists.txt** do seu pacote deve ser alterado de modo dizer ao compilador sobre sua existência.



Serviços Padronizados

Tipos de serviço já existentes na comunidade ROS:

nav_msgs/GetMap

http://docs.ros.org/api/nav_msgs/html/srv/GetMap.html

nav_msgs/GetPlan

http://docs.ros.org/api/nav_msgs/html/srv/GetPlan.html

nav_msgs/SetMap

http://docs.ros.org/api/nav_msgs/html/srv/SetMap.html

sensor_msgs/SetCameraInfo

http://docs.ros.org/api/sensor_msgs/html/srv/SetCameraInfo.html

std_srvs/Empty

http://docs.ros.org/api/std_srvs/html/srv/Empty.html

std_srvs/SetBool

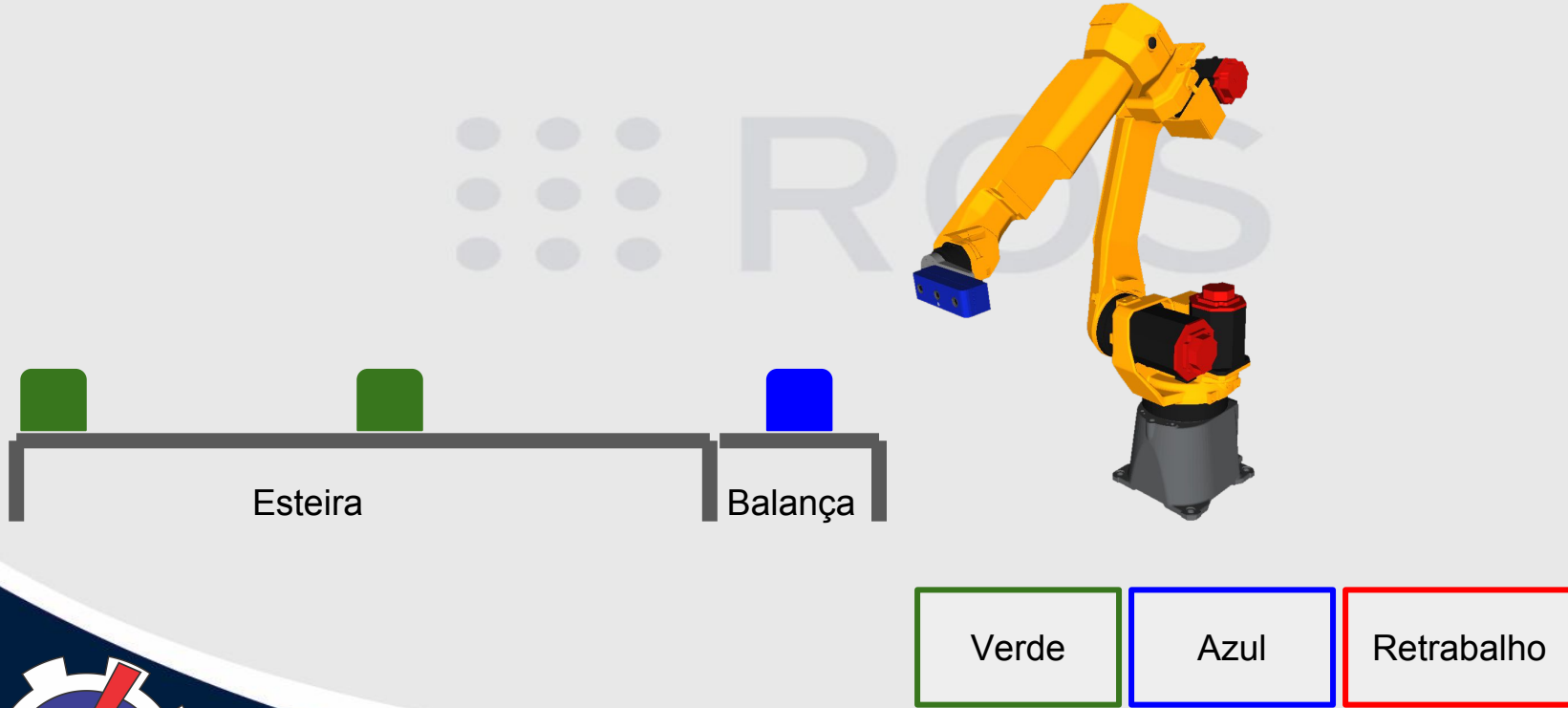
http://docs.ros.org/api/std_srvs/html/srv/SetBool.html

std_srvs/Trigger

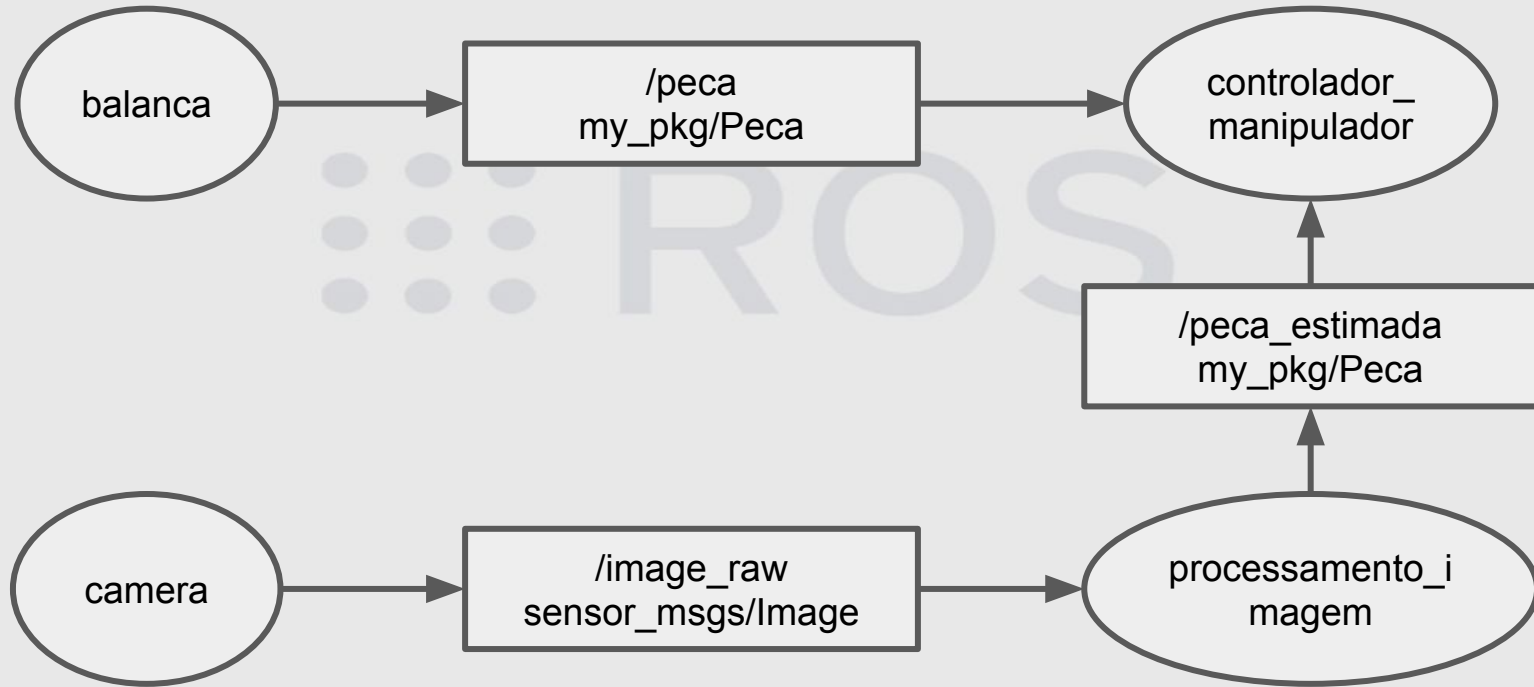
http://docs.ros.org/kinetic/api/std_srvs/html/srv/Trigger.html



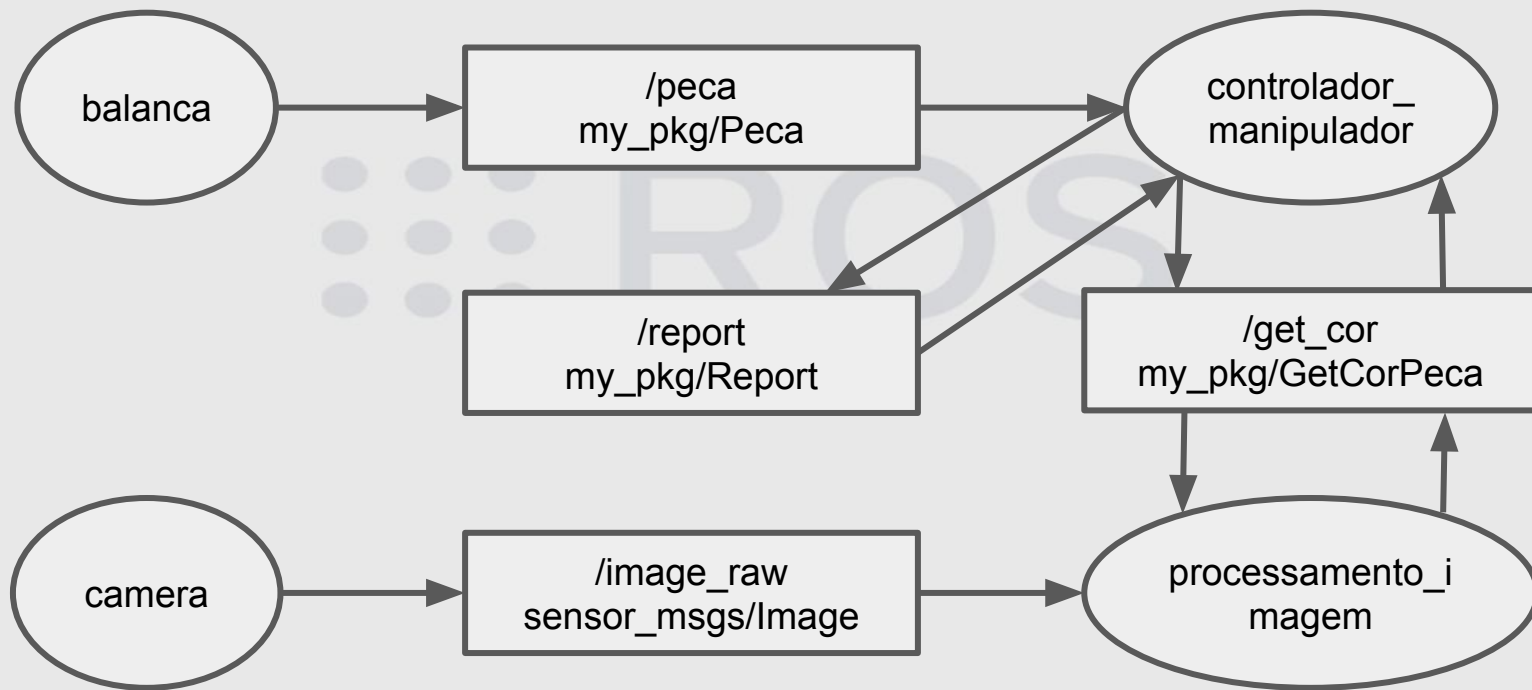
Voltando ao Problema anterior



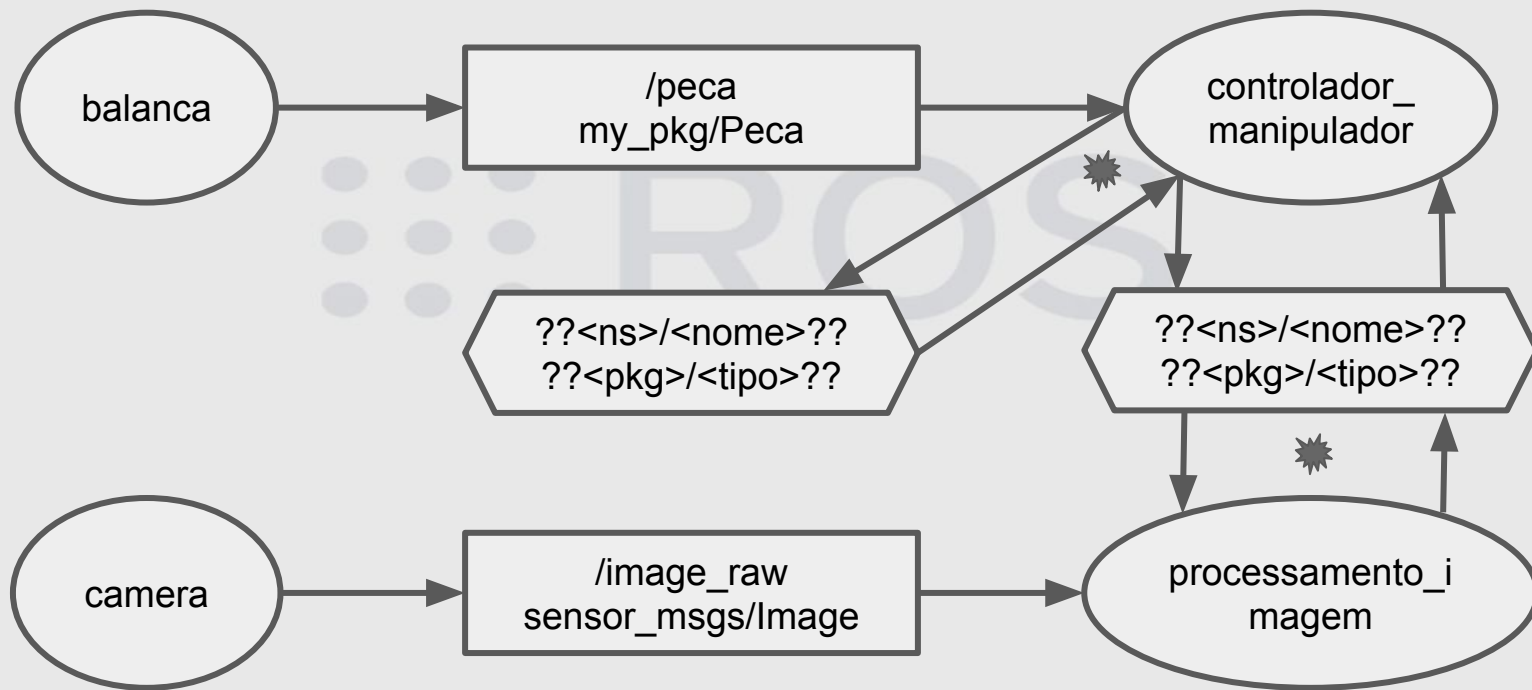
Modelo do Problema



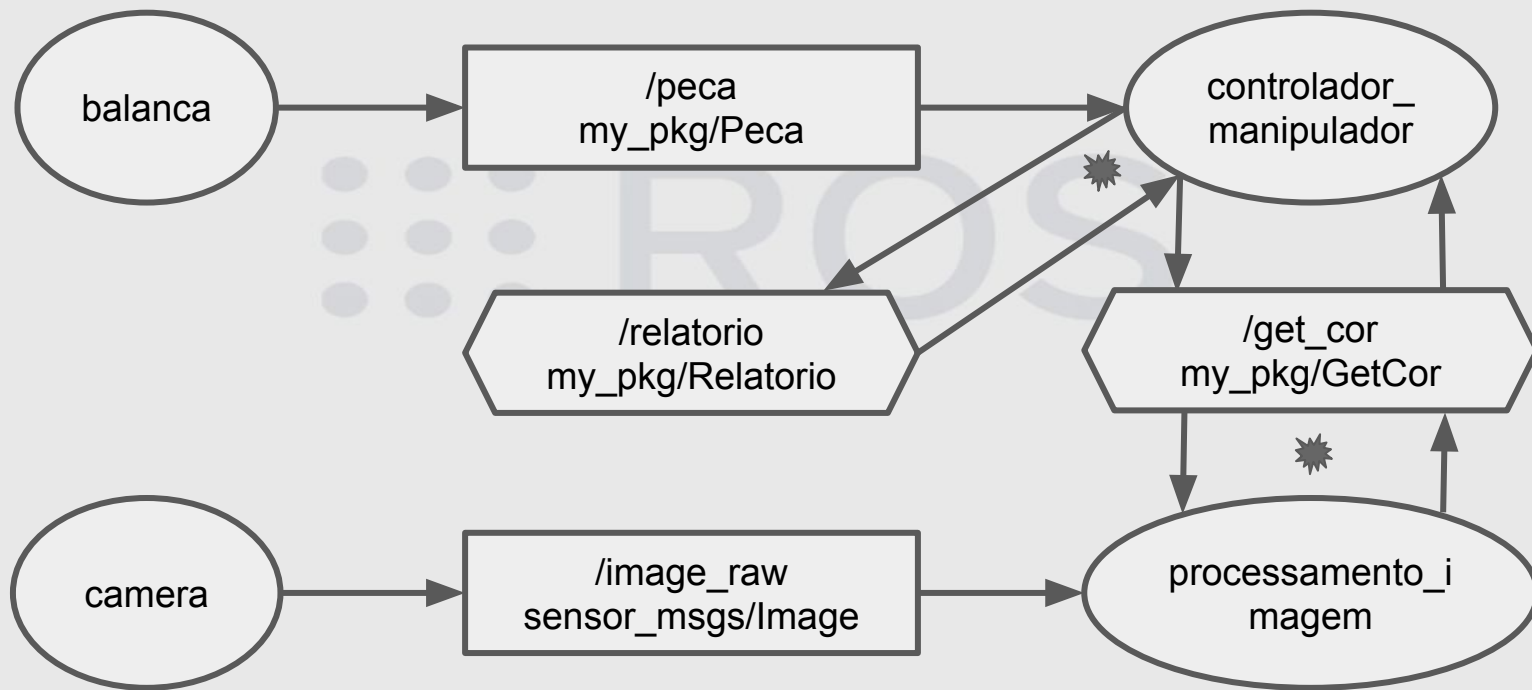
Remodelando o Problema Anterior



Remodelando o Problema Anterior



Remodelando o Problema Anterior



Remodelando o Problema Anterior

O tipo de serviço my_pkg/GetCor é definido pelo arquivo GetCor.srv que se localiza no pacote denominado my_pkg:

```
my_pkg/Peca peca      # apenas a massa mensurada
---
bool sucesso          # verdadeiro se peça foi detectada
string mensagem       # relata o que aconteceu
my_pkg/Peca peca      # com todas as informações
```



Remodelando o Problema Anterior

O tipo de serviço my_pkg/Relatorio é definido por pelo arquivo Relatorio.srv que se localiza no pacote denominado my_pkg:

uint64[] cor	# (cor[0]) num. verdes; (cor[1]) num. azuis;
uint64 retrabalho	# cont. retrabalho



Tópicos *versus* Serviços

Tópicos:

Canal que transporta mensagens de um único tipo em um sentido apenas (origem: Publicador, destino: Assinante).

Serviços:

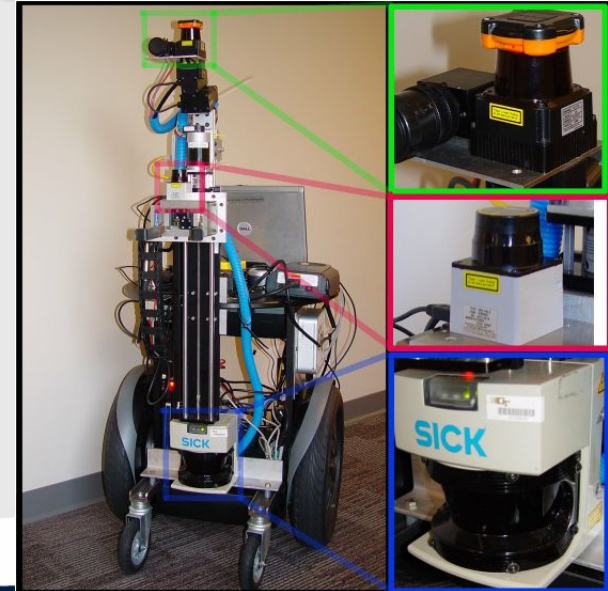
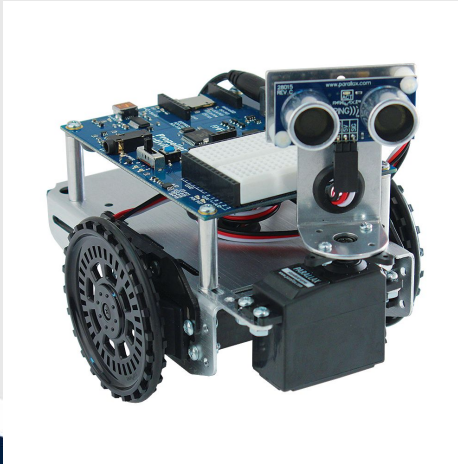
Canal que transporta dois tipos de mensagens, cada uma em um sentido, as quais são denominadas: Pedido (origem: Cliente, destino: Servidor) e Resposta (origem: Servidor, destino: Servidor).



Tópicos

Deve ser utilizado para **transportar fluxo de dados contínuos**. Exemplo:

- dados de sensores (imagem, infra-red, odometria, presença, óptico);
- estado de agentes (posição do robô, posição das juntas do robô);
- comandos para atuadores (velocidades, servo-mecanismos).



Tópicos

Origem e destino do dado não é de interesse.

Conexão muitos-para-muitos (n:n ou **many-to-many**).

A callback de cada assinante recebe o dado (quando disponíveis).

O publicador decide quando enviar o dado.



Serviços

Deve ser utilizada por chamadas de procedimentos remotos que são realizados rapidamente, por exemplo, (1) deseja-se identificar o estado de um nó ou (2) rápido cálculo de cinemática inversa de um determinado manipulador.

Eles não devem ser usados em processos demorados, especialmente, aqueles em que é exigido tomadas preditivas para evitar situações indesejadas.

Usado para chamada de blocos simples.



Um problema que eles (sozinhos) não resolvem

Em alguns casos, se a execução do serviço é muito longa, o usuário pode querer ter a possibilidade de **cancelar o pedido durante sua execução** e/ou ter **feedback periódico da sua progressão**.

O pacote **actionlib** fornece ferramentas para criar servidores que executam objetivos de longa duração e que podem ser antecipados. Além disso, esse pacote também fornece uma interface cliente de modo a enviar pedidos para servidores.

Este curso não apresentará mais detalhes sobre este pacote.



Exercício

Pede-se para criar um modelo baseado em ROS para atender as exigências de um problema de resgate, conforme descrito a seguir.

Antes que uma equipe de apoio (policiais, bombeiros, enfermeiros, etc.) entre em um prédio que sofre uma explosão, deseja-se que um grupo de robôs identifique o estado da estrutura predial, se existem sobreviventes, etc.



Exercício

A equipe de apoio, fica aguardando a equipe de robôs realizar seu trabalho em uma distância segura (conforme mostra a figura ao lado).



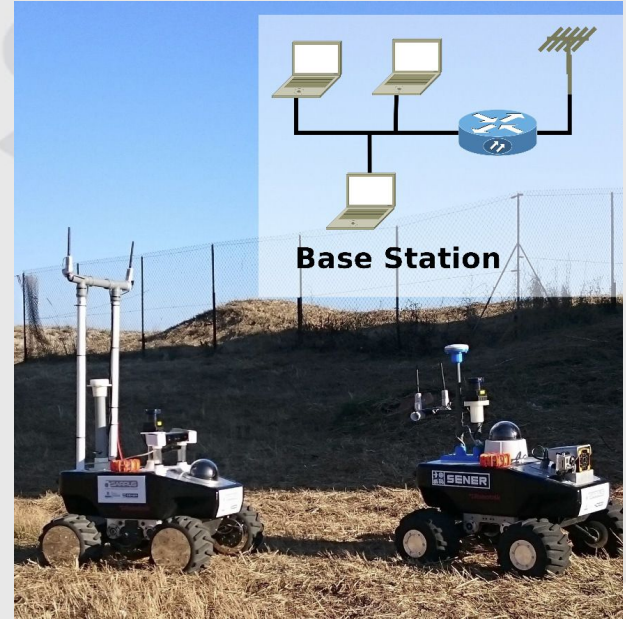
Saindo da estação de controle, junto da equipe de apoio, os robôs têm que chegar até uma localização próxima às instalações do prédio de forma autônoma.



Exercício

Os robôs seguem até as aproximações do prédio em formação de uma linha indiana. De modo que o robô na frente da fila (mestre) é responsável por desviar de obstáculos e otimizar a trajetória para chegar até o destino de forma segura e rápida.

Os outros robôs seguem os movimentos do robô da frente, mantendo sempre uma distância segura.

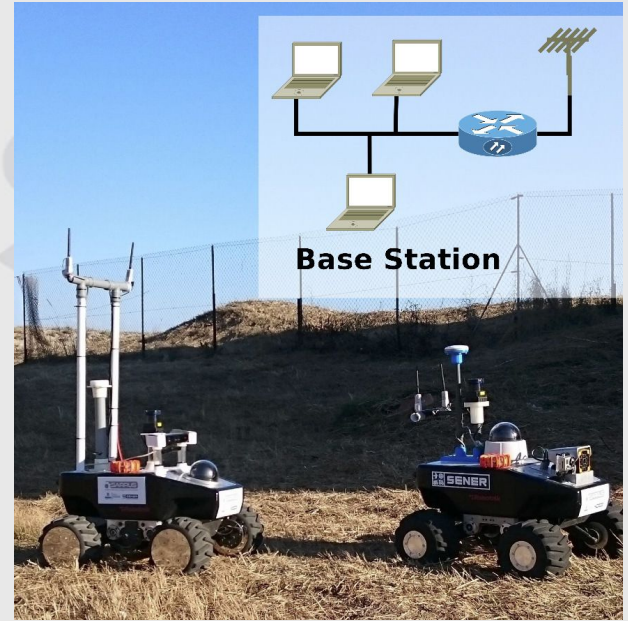


(1) Exercício

Para isso, pede-se modelar pacotes, nós, tópicos e serviços necessários para que os robôs tenham este tipo de comportamento.

Não é necessário se preocupar com a implementação dos nós, nem tão pouco com o sistema como um todo.

Preocupe-se com nomes e tipos das entidades e recursos do sistema ROS.



Exercício

Para a segunda parte do problema pede-se:

Uma vez que os robô estão localizados no interior do prédio, eles deverão desfazer da formação linear e adotar uma formação de exploração;



(2) Exercício

Cada robô possui uma câmera e um sensor laserscan, cujas funções são, respectivamente:

- Procurar por sobreviventes (a câmera identifica o rosto de pessoas) e informar sua localização;
- Fazer o mapeamento do local e analisar a estrutura do prédio.



(3) Exercício

Com os dois comportamentos anteriores: (1) formação de fila indiana e (2) análise de estrutura e procura por sobreviventes, deseja-se modelar um controlador dos dois comportamentos.

Este controlador será responsável pela inibição e ativação de comportamento de cada robô, designando o papel de cada robô apropriadamente.



Exercício

Trabalho individual (mas pode trocar idéias com os colegas ao lado a respeito do exercício).

Entregar hoje (**30/09/2016**) até às **23:55** para **eca419.unifei@gmail.com**.

Enviar **foto ou desenho do esboço do modelo** (em papel, editor de imagem, ou diagramador), utilizando a mesma notação de **ellipse** (nó), **retângulo** (tópico), **hexágono** (serviço) e **estrela** (servidor).

