Frameworks e Simuladores de Robótica

Rafael.Prudencio@gmail.com

LAROCS
LABORATORY OF ROBOTICS AND COGNITIVE SYSTEMS

UNICAMP

Conteúdo

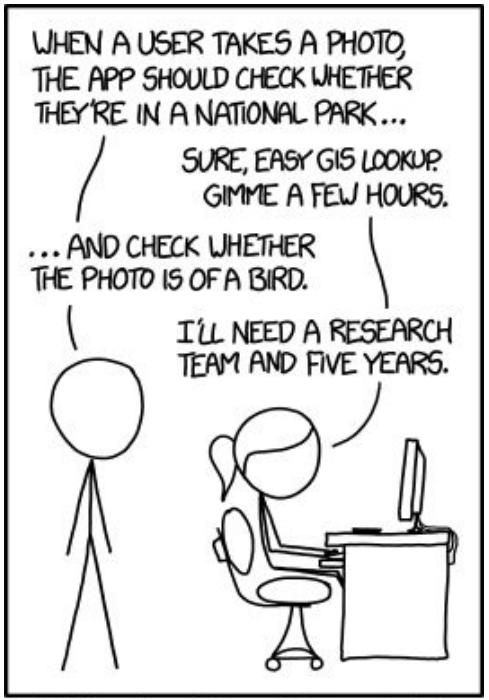
- Frameworks
  - Nomenclatura
  - ROS
  - Benchmarks
  - Robotino View
- Simuladores
  - Benchmark
  - V-REP
  - Robotino SIM

Nomenclatura

- **Middleware:** contempla os drivers e a infraestrutura básica para a comunicação entre nós. Invisível para o desenvolvedor.
- **Framework:** coleção de ferramentas, bibliotecas e convenções visando simplificar o desenvolvimento de software para tarefas complexas de robótica.
- **Arquitetura:** descrição abstrata de como os módulos devem comunicar-se e interagir uns com os outros.

Frameworks

- A construção de sistemas robóticos complexos requer um nível de especialização muito alto.
- A união de um grande número de módulos é necessária para o funcionamento de um sistema robótico.
- Frameworks de robótica visam evitar a reinvenção da roda, disponibilizando uma base sólida de ferramentas para um desenvolvedor construir sobre.



LAROCS
LABORATORY OF ROBOTICS AND COGNITIVE SYSTEMS

ROS (Robot Operating System)

- Framework de robótica desenvolvido desde 2007 com suporte para UNIX.
- Meta-sistema operacional: abstração de hardware, controle de baixo-nível para dispositivos, comunicação entre processos através de mensagens e gerenciamento de pacotes.

ROS

- Visa atrair desenvolvedores para contribuir e compartilhar no desenvolvimento.
- Adotado pela comunidade científica e comercial
- Algumas plataformas comerciais que usam ROS são:

Care-O-Bot 3

iRobot Care

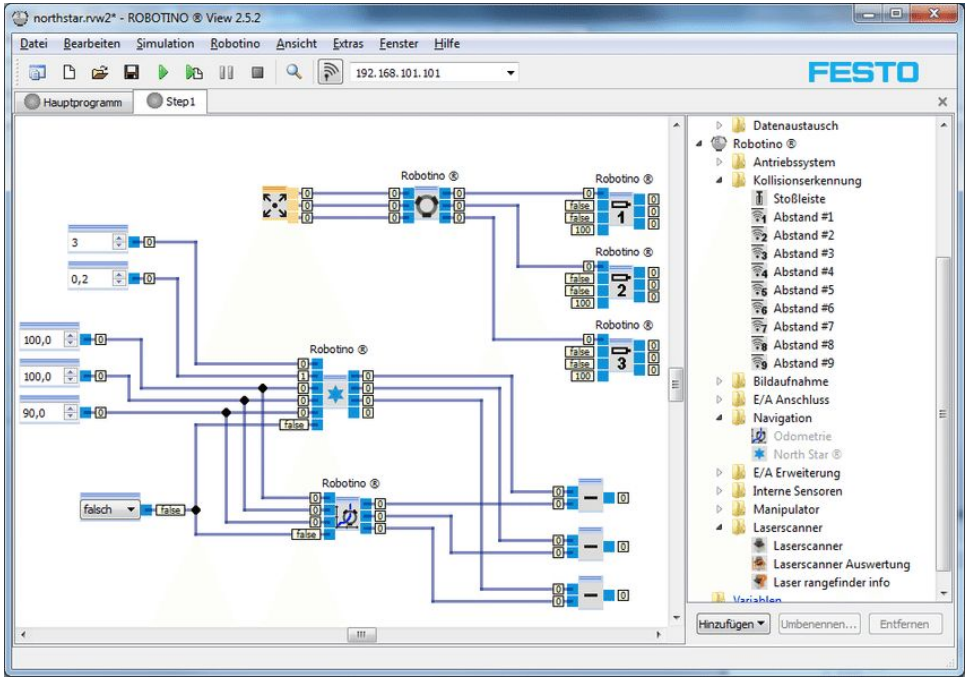
Nao

RFWs	OS	Programming language	Open source	Distributed architecture	HW interfaces and drivers	Robotic algorithms	Simulation	Cntrol / Realtime oriented
ROS	Unix	C++, Python, Lisp	✓	✓	✓	✓	~	✗
HOP	Unix, Windows	Scheme, Javascript	✓	✓	~	✗	✗	✗
Player/Stage/Gazebo	Linux, Solaris, BSD	C++, Tcl, Java, Python	✓	~	✓	✓	✓	✗
MSRS (MRDS)	Windows	C#	✗	✓	~	✗	✓	✗
ARIA	Linux, Win	C++, Python, Java	✓	✗	✓	✓	✗	✗
Aseba	Linux	Aseba	✓	✓	✓	✗	~	✓
Carmen	Linux	C++	✓	✓	✓	✓	✓	✗
CLARAty	Unix	C++	✓	✓	✓	✓	✗	✗
CoolBOT	Linux, Win	C++	✓	✓	~	✗	✗	✗
ERSP	Linux, Win	?	✗	✓	✓	✓	✗	✗
iRobot Aware	?	?	✗	?	✓	?	✗	?
Marie	Linux	C++	✓	✓	✓	✗	✗	✗
MCA2	Linux, Win32, OS/X	C, C++	✓	✓	✓	✗	✗	✓
Miro	Linux	C++	✓	✓	✓	✗	✗	✗
MissionLab	Linux, Fedora	C++	✓	✓	✓	✓	✓	✗
MOOS	Windows, Linux, OS/X	C++	✓	~	✓	✓	✗	✗
OpenRAVE	Linux, Win	C++, Python	✓	✗	✗	✓	✓	✗
OpenRDK	Linux, OS/X	C++	✓	✓	✓	✗	✗	✗
OPRoS	Linux, Win	C++	✓	✓	✓	✓	✓	✗
Orca	Linux, Win, QNX Neutrino	C++	✓	✓	✓	~	✗	✗
Orocos	Linux, OS/X	C++	✓	✓	✓	✓	✗	✓
RoboFrame	Linux, BSD, Win	C++	?	✓	✓	✗	✗	✗
RT middleware	Linux, Win, CORBA platform	C++, Java, Python, Erlang	✓	✓	✓	✗	✗	✗
Pyro	Linux, Win, OS/X	Python	✓	✗	✓	✓	✓	✗
ROCI	Win	C#	✓	✓	✗	✗	✗	✗
RSCA	?	?	✗	✗	✓	✗	✗	✓
ROCK	Linux	C++	✓	?	✓	✓	✗	✓
SmartSoft	Linux	C++	✓	✓	✗	✗	✗	✗
TeamBots	Linux, Win	Java	✓	✗	✓	✓	✓	✗
Urbi (language)	Linux, OS/X, Win	C++ like	✓	✗	✓	✗	✗	✗
Webots	Win, Linux, OS/X	C, C++, Java, Python, Matlab, Urbi	✗	✗	✓	✗	✓	✗
YARP	Win, Linux, OS/X	C++	✓	✓	✓	✗	✓	✗

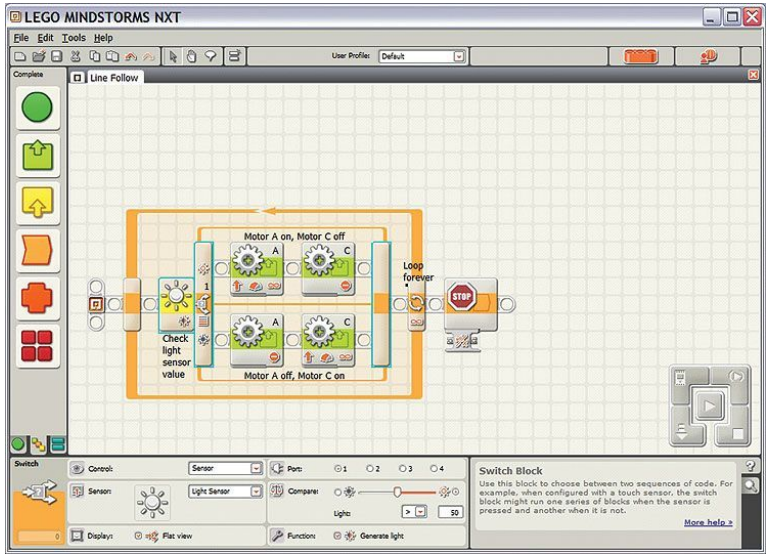
Fonte: https://arxiv.org/abs/1711.06842

Robotino View

- Ambiente interativo de programação gráfica.
- Programação em blocos:
  - Processamento de imagens: detector de linhas, busca por intervalos de cor e detecção de marcadores
  - Navegação: navegador de posição e distância, bem como evitar obstáculos
  - Troca de dados: UDP, TCP/IP cliente/servidor e OPC



Programação em Blocos



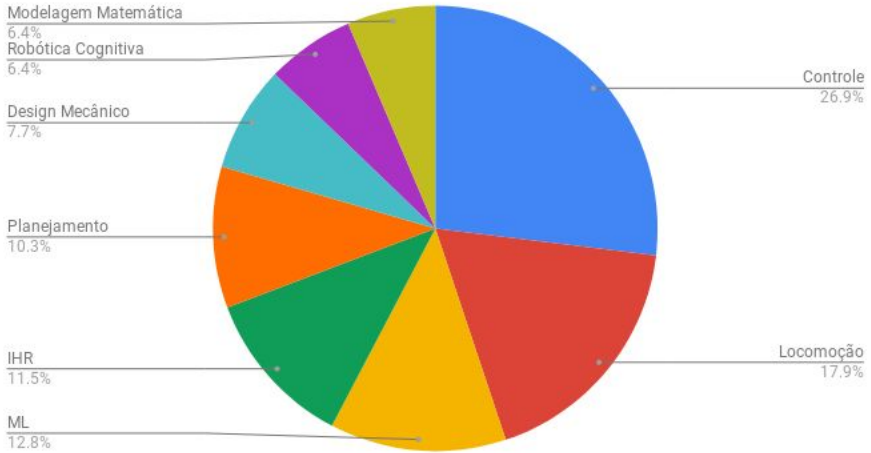
LEGO Mindstorms NXT

Simuladores de Robótica

- Permitem a criação de uma aplicação para um robô físico sem depender de hardware, poupando custo e tempo.
- Modelagem 3D de um robô e seu ambiente.
- Emula movimento e colisões realísticas através de engines físicas.

Benchmark

- Pesquisa com 119 participantes que trabalham com robótica em universidades (70%) ou com P&D em institutos públicos (16%) e privados (14%).
- Aŕeas de atuação:



Fonte: https://arxiv.org/pdf/1402.7050.pdf

Critérios

Rank	Feature	Overall Evaluation	Rating	Median rating
1	**Stability of simulation**	Very important	4.50 ± 0.58	5
2	**Speed**	Important	4.05 ± 0.75	4
3	**Precision of simulation**	Important	4.02 ± 0.71	4
4	**Accuracy of contact resolution**	Important	3.91 ± 0.92	4
5	**Same interface between real & simulated system**	Important	3.67 ± 1.26	4
6	Computational load (CPU)	Neutral	3.53 ± 0.85	3
7	Computational load (memory)	Neutral	3.22 ± 0.90	3
8	Visual rendering	Neutral	3.02 ± 1.02	3

Features mais importantes de um simulador avaliadas de 1 (irrelevantes) a 5 (muito importante, crucial).

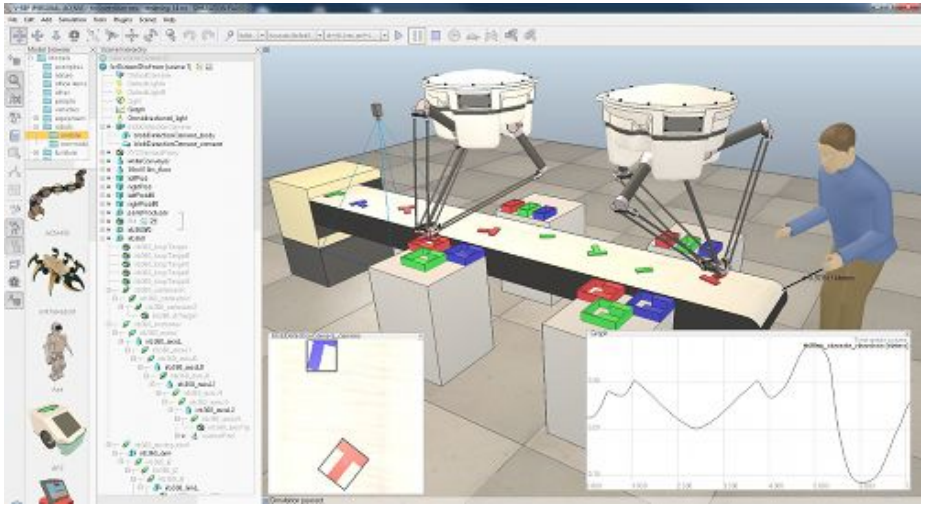
Satisfação dos Usuários

Tool	Documentation	Support	Installation	Tutorials	Advanced use	Active project & community	API	Global
Gazebo	3.47 ± 0.99	4.00 ± 1.07	3.93 ± 1.03	3.53 ± 1.12	3.80 ± 0.86	4.73 ± 0.45	3.67 ± 0.82	3.88 ± 0.91
ARGoS	3.40 ± 0.70	3.90 ± 0.99	4.70 ± 0.48	4.20 ± 0.63	4.60 ± 0.70	4.10 ± 0.74	4.30 ± 0.67	4.17 ± 0.70
ODE	3.80 ± 0.63	3.40 ± 1.07	4.10 ± 1.28	3.20 ± 1.13	3.90 ± 1.37	3.30 ± 1.25	3.40 ± 1.26	3.59 ± 1.15
Bullets	3.37 ± 1.06	3.62 ± 0.91	4.75 ± 0.46	4.00 ± 0.76	3.75 ± 0.71	4.37 ± 0.74	3.87 ± 0.83	3.96 ± 0.78
V-Rep	4.28 ± 0.76	4.43 ± 0.79	4.71 ± 0.76	4.14 ± 0.90	4.28 ± 0.76	4.43 ± 0.53	4.14 ± 1.07	4.25 ± 0.80
Webots	3.86 ± 1.07	3.57 ± 1.13	4.43 ± 0.79	3.43 ± 1.51	4.42 ± 0.78	4.14 ± 0.69	4.57 ± 0.53	4.20 ± 0.96
OpenRave	3.50 ± 0.55	4.67 ± 0.52	4.17 ± 0.75	3.50 ± 1.22	4.33 ± 0.82	4.33 ± 0.52	4.33 ± 0.52	4.12 ± 0.70
Robotran	3.60 ± 0.55	3.80 ± 0.45	3.80 ± 0.45	3.20 ± 0.84	4.20 ± 0.84	3.20 ± 0.84	3.80 ± 0.45	3.66 ± 0.63
Vortex	3.33 ± 1.15	3.67 ± 1.53	5.00 ± 0.00	2.67 ± 0.58	3.67 ± 0.58	2.67 ± 1.15	3.33 ± 0.58	3.48 ± 0.80
OpenSIM	4.33 ± 0.58	4.67 ± 0.58	3.67 ± 0.58	3.00 ± 1.00	4.00 ± 0.00	4.67 ± 0.58	3.67 ± 0.58	4.00 ± 0.55
MuJoCo	2.33 ± 1.15	1.67 ± 0.58	4.33 ± 1.15	3.33 ± 1.15	4.67 ± 0.57	4.00 ± 0.00	5.00 ± 0.00	3.62 ± 0.66
XDE	1.40 ± 0.55	2.80 ± 1.09	3.60 ± 0.55	2.80 ± 1.09	3.40 ± 1.10	2.80 ± 0.84	3.00 ± 1.00	2.83 ± 1.07

Nível de satisfação dos usuários para para cada ferramenta por categoria de 1 (muito insatisfeito) a 5 (muito satisfeito).

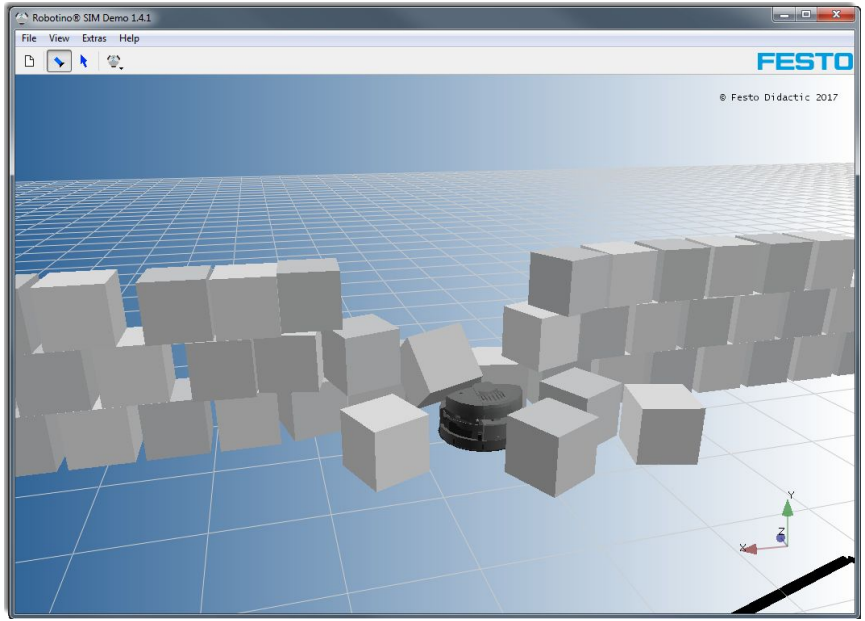
V-REP

- Multiplataforma: Windows, Linux e MacOS
- Suporte para 4 engines físicas: Bullet Physics, ODE, Newton e Vortex Dynamics.
- Inúmeras abordagens de programação: scripts embutidos, API remota, plugins e **nós do ROS**.



Robotino SIM

- Suporte apenas para Windows.
- Suporte para uma única engine física, NVIDIA PhysX.
- Abordagens de programação: Robotino View, APIs para C++, Java e **ROS.**



Utilização do Catkin Tools

Guilherme.Abreu@gmail.com

O que é um Catkin Workspace?

Catkin Workspace é um espaço de trabalho padrão do ROS onde você pode modificar, montar e instalar os seus pacotes. Ele é um diretório que contém essencialmente 3 pastas:

- src/
- build/
- devel/

Source Space (src/)

- Código fonte

Build Space (build/)

- Montagem dos pacotes
- Cache de informação
- Arquivos intermediários

Development Space (devel/)

- Testes e desenvolvimento de pacotes
- Dispensa invocação da instalação de pacotes

Criando seu Catkin Workspace

Para criar o seu espaço de trabalho catkin execute os comandos a seguir no terminal apenas trocando ‘catkin_ws’ para o nome que você preferir para o seu espaço de trabalho:

```
mkdir –p ~/catkin_ws/src  #Cria espaço de trabalho na pasta do usuário

cd ~/catkin_ws

catkin init      #Cria link que aponta para toplevel.cmake do catkin
```

Pacotes Catkin

Para que um pacote seja considerado catkin ele precisa cumprir alguns requisitos, são eles:

- Conter os arquivos: package.xml e CMakeLists.txt
- Não conter múltiplos pacotes no mesmo diretório

Criando um Pacote Catkin

Para criar um pacote vá até o seu source space e execute o seguinte comando apenas trocando ‘nome_do_pacote’ pelo nome do pacote que você deseja criar:

```
catkin_create_pkg nome_do_pacote dependencia1 dependencia2 ...
```

O comando ‘catkin_create_pkg’ pede que você dê o nome do pacote e opcionalmente uma lista de dependências, como as dadas acima.

Montando seus pacotes

Depois de ter criado o seu espaço de trabalho e o pacote catkin, você pode montá-lo com o seguinte comando a partir da pasta raiz:

```
catkin build
```

O comando catkin build é uma ferramenta de conveniência para se trabalhar com espaços de trabalho catkin.

Sobrepondo o Espaço de Trabalho

Após montar e instalar um pacote, para que ele possa ser utilizado é necessário sobrepor o seu espaço de trabalho. Para isso, vá até a pasta raiz do seu espaço de trabalho e execute o comando abaixo:

```
source devel/setup.bash
```

Hands-on: Criação do Pacote “hello_world”

Guilherme.Abreu@gmail.com

Criando um pacote

Crie o pacote de nome ´hello_world´ com a seguinte lista de dependências:
std_msgs rospy roscpp

Introdução aos Fundamentos do ROS

Gabriel.Previato@gmail.com

LAROCS
LABORATORY OF ROBOTICS AND COGNITIVE SYSTEMS

UNICAMP

Conteúdo

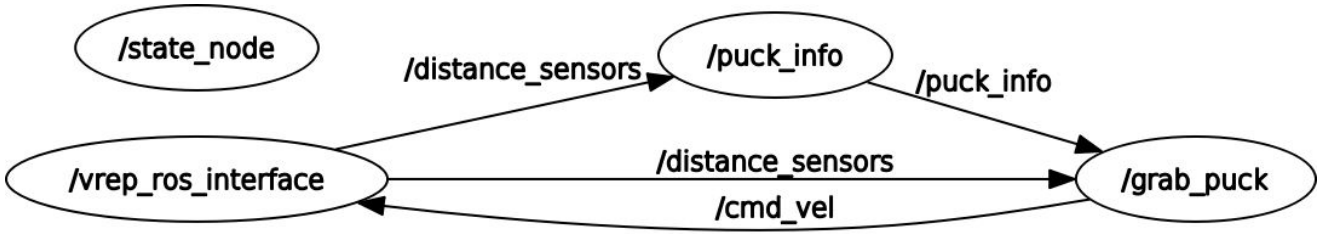
- Nó
  - Rosnode
  - Rosrun
- Tópico
  - Mensagem
  - Rostopic
- Comunicação
  - Exemplo
  - Hands-on

Nó

- Um processo que executa algum tipo de computação.

Nó

- Um processo que executa algum tipo de computação.
- Um executável dentro de um pacote.



Nó - rosnode

- ROS disponibiliza diversas ferramentas que pode ser utilizadas para obtermos informações sobre os nós.

```
previato ~ rosnode list
/grab_puck
/puck_info
/rosout
/state_node
/vrep_ros_interface
```

Nó - rosnode

```
previato ~ rosnode info /grab_puck
--------------------------------------------------------------------------------
Node [/grab_puck]
Publications:
 * /cmd_vel [geometry_msgs/Twist]
 * /grab_puck/feedback [grab_puck/GrabPuckActionFeedback]
 * /grab_puck/result [grab_puck/GrabPuckActionResult]
 * /grab_puck/status [actionlib_msgs/GoalStatusArray]
 * /rosout [rosgraph_msgs/Log]

Subscriptions:
 * /distance_sensors [sensor_msgs/PointCloud]
 * /grab_puck/cancel [unknown type]
 * /grab_puck/goal [unknown type]
 * /puck_info [puck_info/PuckInfoMsg]

Services:
 * /grab_puck/get_loggers
 * /grab_puck/set_logger_level
```

Nó - rosrun

- Rosrun é o comando responsável para executar os Nós.
- rosrun <nome_do_pacote> <nome_do_nó_executavel>

```
~/LaRoCS/catkin_ws$ rosrun hello_world ouvinte_node
```

Tópico

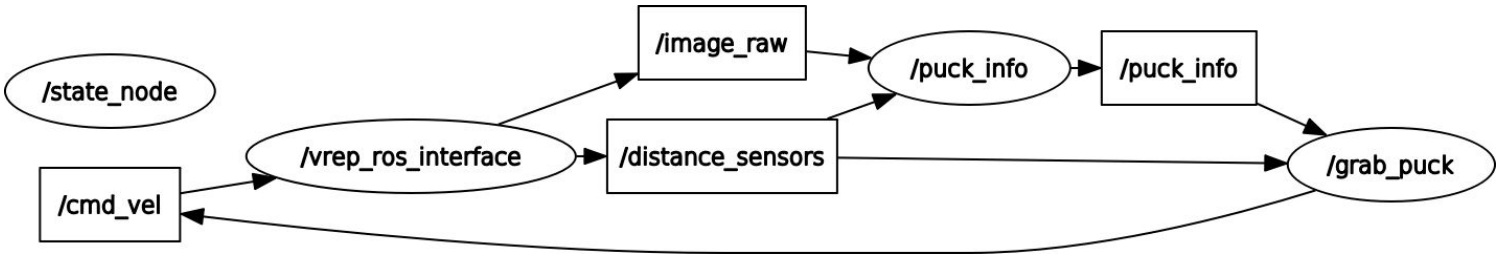
- Canais de comunicação em que os nós trocam mensagens.

Tópico

- Canais de comunicação em que os nós trocam mensagens.
- Tópicos possuem tipagem forte de acordo com a mensagem que passa pelo tópico.

Tópico

- Canais de comunicação em que os nós trocam mensagens.
- Tópicos possuem tipagem forte de acordo com a mensagem que passa pelo tópico.



Tópico - Mensagem

- No ROS, mensagens são a forma como os dados são transmitidos.

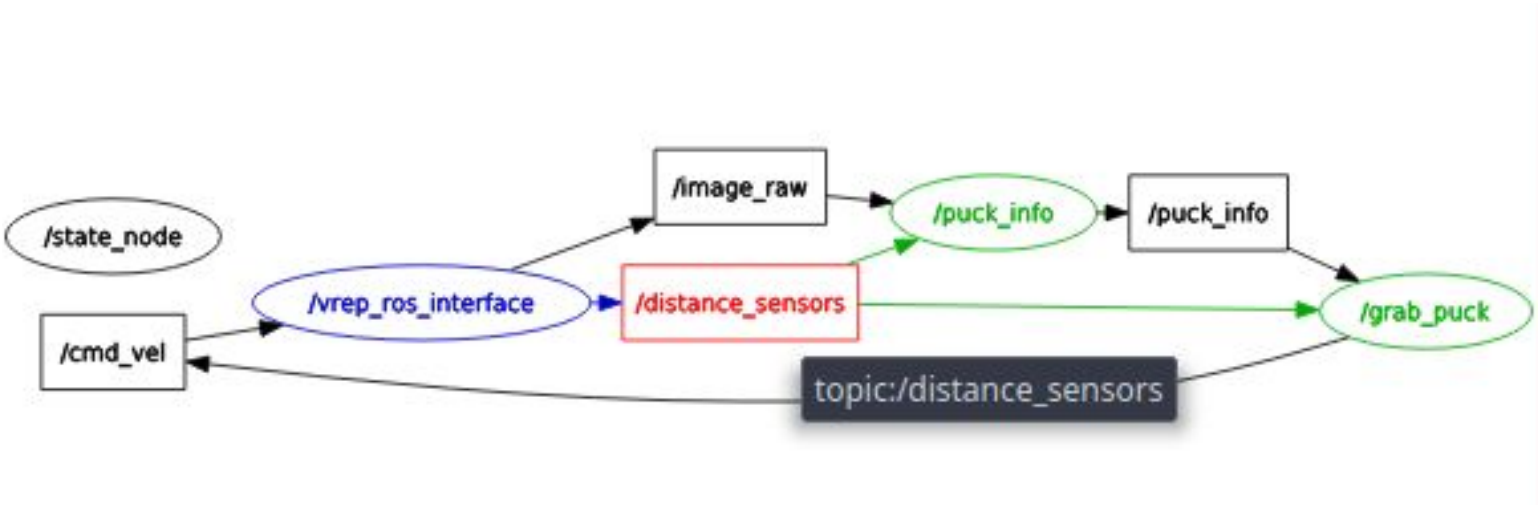
Tópico - Mensagem

- No ROS, mensagens são a forma como os dados são transmitidos.
- O tipo do tópico é definido pelo tipo da mensagem.

Tópico - Mensagem

- No ROS, mensagens são a forma como os dados são transmitidos.
- O tipo do tópico é definido pelo tipo da mensagem.
- Nós podem subscrever de um tópico (ler mensagens desse tópico) ou podem publicar em um tópico (escrever mensagens neste tópico)

Tópico - Mensagem



Tópico - rostopic

```
previato ~ rostopic info /distance_sensors
Type: sensor_msgs/PointCloud

Publishers:
 * /vrep_ros_interface (http://previato-kubuntu-desk:36099/)

Subscribers:
 * /grab_puck (http://previato-kubuntu-desk:44939/)
 * /puck_info (http://previato-kubuntu-desk:40591/)
```

Tópico - rostopic

```
previato ~ rostopic list
/bumper
/cmd_vel
/distance_sensors
/floor_image_raw
/grab_puck/cancel
/grab_puck/feedback
/grab_puck/goal
/grab_puck/result
/grab_puck/status
/image_raw
/inductive
/puck_info
/rosout
/rosout_agg
/state
/statistics
```

Hands-on: Hello World!

Gabriel.Previato@gmail.com

LAROCS
LABORATORY OF ROBOTICS AND COGNITIVE SYSTEMS

UNICAMP

Primeiros passos

- Utilizaremos o pacote “hello_world” que já construímos.

Primeiros passos

- Utilizaremos o pacote “hello_world” que já construímos.
- Criar um arquivo no diretório “hello_world/src” chamado “falante.cpp” e outro chamado “ouvinte.cpp”

LaRoCS@LARC17

Samuel.Chenatti@gmail.com

LAROCS
LABORATORY OF ROBOTICS AND COGNITIVE SYSTEMS

UNICAMP

Objetivo

- Demonstrar que o ROS é uma plataforma viável para competição
  - Utilizaremos nossa solução como use-case
  - Mostraremos a aplicação do conteúdo que aprenderam hoje
  - Vamos apresentar alguns pacotes que são distribuídos junto ao ROS
  - Daremos alguns insights sobre possíveis problemas que podem aparecer durante o desenvolvimento

Introdução

- Equipe formada por alunos de diferentes cursos
  - Ciência da Computação
  - Engenharia da Computação
  - Engenharia Elétrica
  - Mestrado/Doutorado em Ciência da Computação
  - Engenharia de Controle e Automação
- Cada aluno se especializa em resolver um problema específico

Introdução

- Nosso objetivo é desenvolver tecnologia e promover conhecimento
  - Trazendo para as competições técnicas recém publicadas
  - Desenvolvendo e publicando novas técnicas
  - Organizando workshops e cursos
  - Disponibilizando nossa base de código

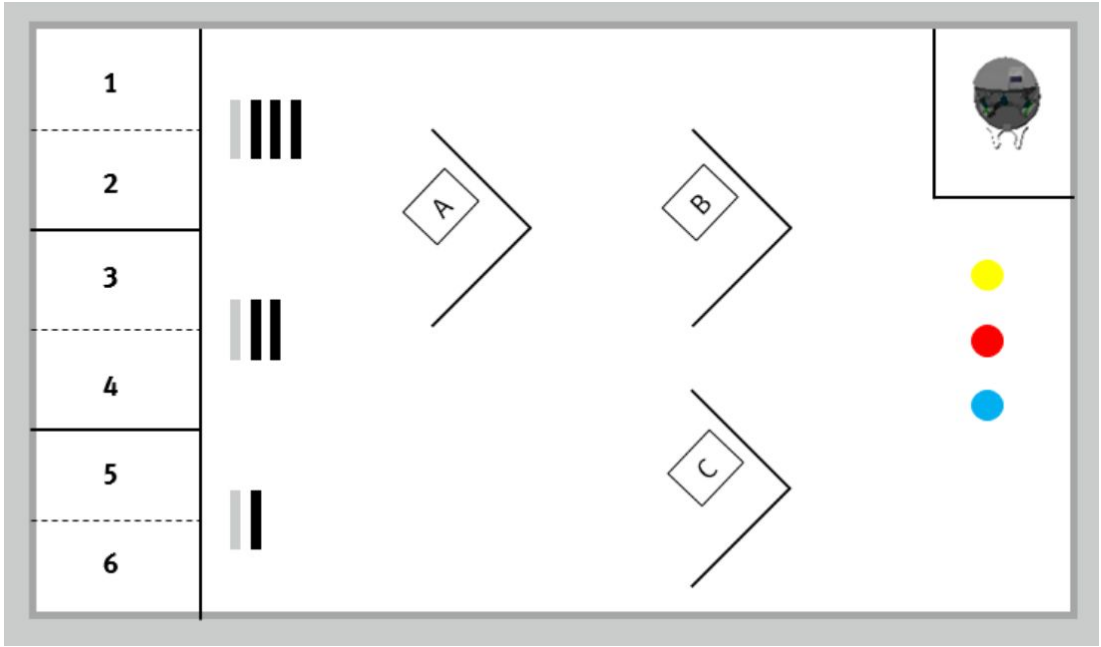
RoboCup Logistics League 2017

Plataforma
Robotino2
Festo

RoboCup Logistics 2017

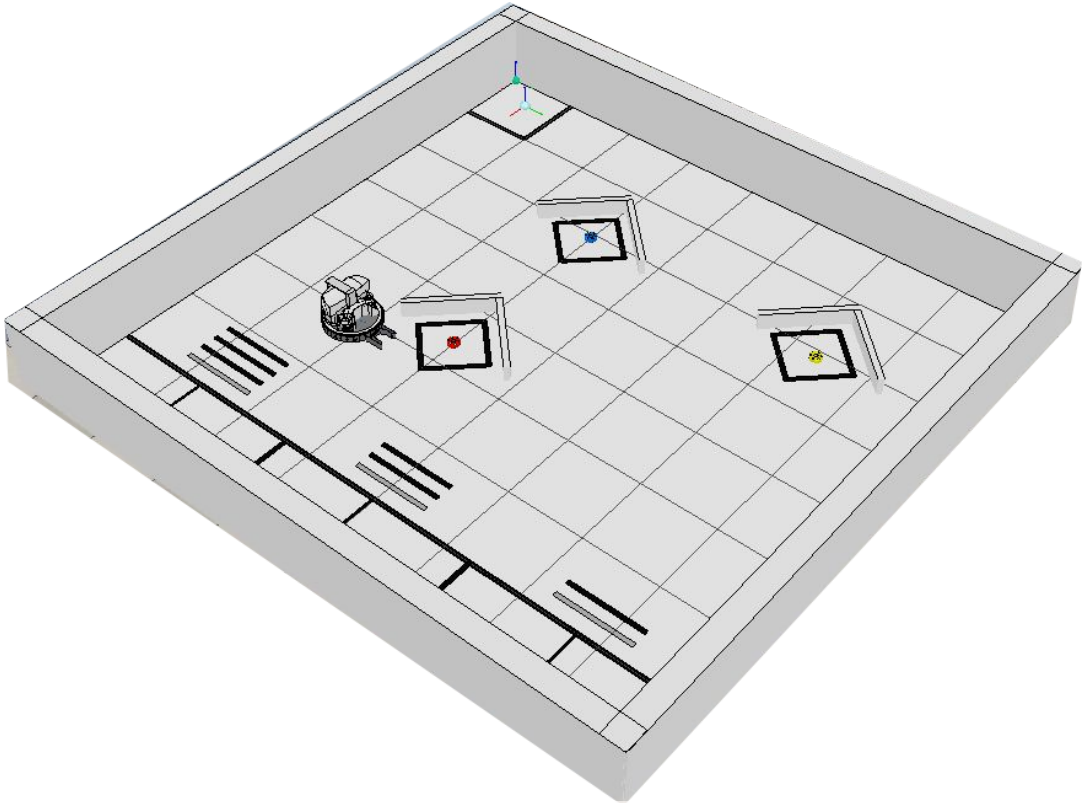
- Estrutura da competição
  - Cada equipe possui um Robotino
  - O Robotino deve ser capaz de transportar pucks em uma arena de 4x4 metros
  - Os pucks devem ser retirados de uma máquina e entregues em um centro de distribuição
  - Os pucks são coloridos, e a ordem em que devem ser entregues é definida pelos juízes
  - A disposição das máquinas e centros de distribuição é conhecida pelo robô
  - **O robô deve realizar a tarefa de forma autônoma**

LARC Logistics 2017



Layout da primeira tarefa
Manual da competição/Festo

LARC Logistics 2017

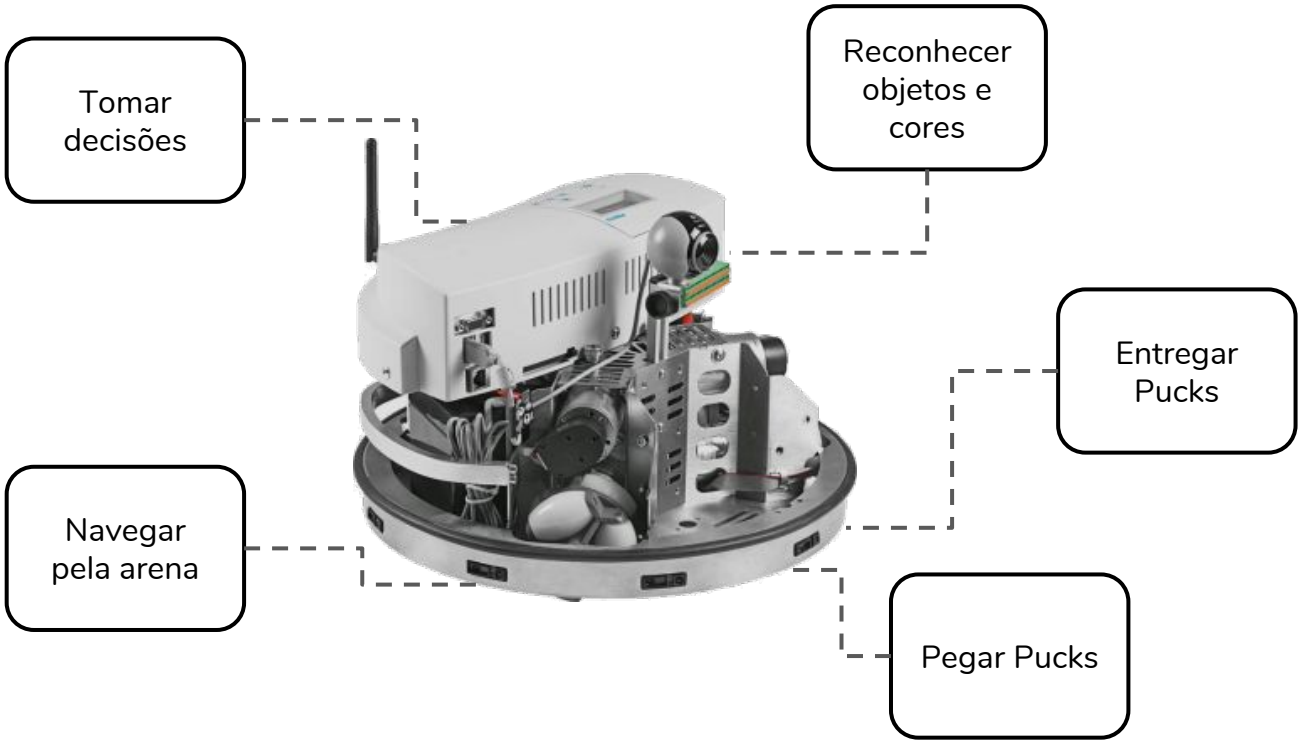


Simulação da primeira tarefa
LaRoCS

Nossa estratégia

- Quebrar a tarefa em problemas menores
  - A equipe é muito grande, então podemos alocar pessoas para tarefas específicas
  - A estrutura do ROS permite que o código seja dividido em pacote; temos cada problema quebrado em um pacote (modularização do código)
  - Trabalhar em simulação elimina a necessidade de ter o robô presente
  - Podemos trabalhar de maneira iterativa
  - A comunicação constante entre os membros da equipe facilita a integração do código

Divisão dos problemas



Navegação

- Stack de navegação do ROS
  - O ROS possuí uma pilha de navegação própria, desenvolvida para funcionar com qualquer robô (na teoria)
  - A stack é dividida em Global Planner e Local Planner
  - Ambos fazem uso de um mapa de custo usado na exploração de ambientes (não é o nosso caso) e planejamento de rotas
  - O Global Planner traça uma rota global (usando Dijkstra ou A*)
  - O Local Planner controla o robô para garantir que a rota seja seguida

Navegação

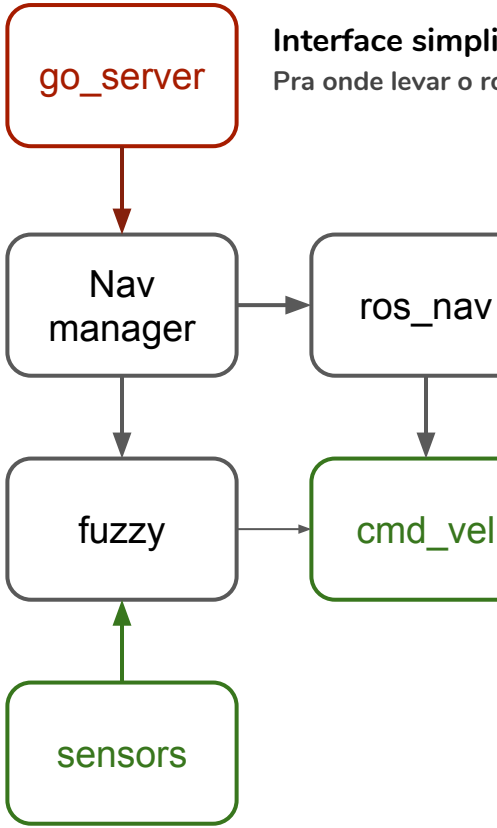
Navegação

- Solução pronta?
  - O nav-stack padrão do ROS por sí só não atende as necessidades da competição
    - Nós ainda precisamos implementar comportamentos reativos (desviar de obstáculos)
    - Precisamos facilitar a comunicação entre os demais nós e a navegação
  - Em resumo: precisamos de mais controle do que o pacote naturalmente oferece

Navegação

- Navigation Manager
  - Construído para oferecer uma interface abstrata de navegação aos demais nós
  - Basicamente, chamamos um nó pedindo que o robô vá até um destino especificado por um identificador único
    - 0 leva o robô à base
    - 1 - 6 leva o robô a uma máquina
    - 7 a 13 leva o robô a um centro de distribuição

Navegação



Interface simplificada
Pra onde levar o robô?

Gerenciador de Navegação
Quem deve mandar velocidades para o robô agora?

Nav-stack do ROS
Planejamento de rotas

Fuzzy node
Responsável pelo comportamento reativo do robô

Controle de Velocidade
Recebe em taxa constante comandos de velocidade

Sensores
Publica em taxa constante a informação dos sensores

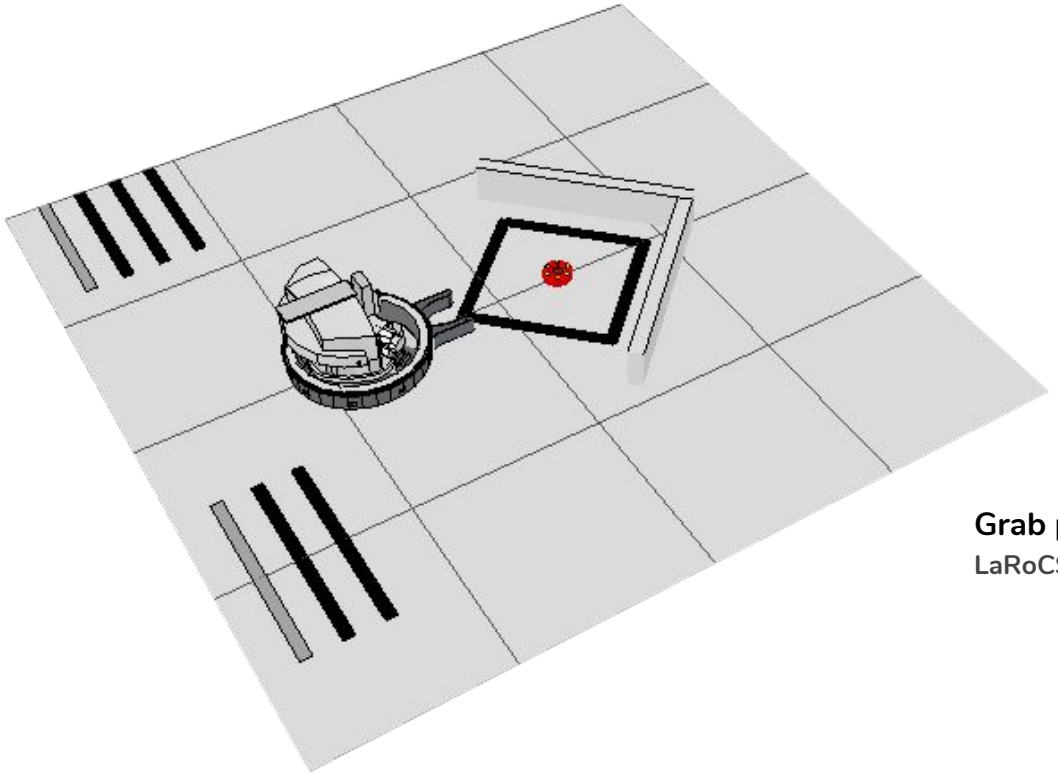
Nós vindos do robô
Nós de ação
Nós de atuação interna às ações

LAROCS
LABORATORY OF ROBOTICS AND COGNITIVE SYSTEMS

Reconhecimento de imagens

- Parte da tarefa requer que o robô seja capaz de identificar e pegar pucks
  - Para pegar um puck, o robô precisa fazer uso dos sensores
    - **Sensor de imagem:** onde está o puck de cor X?
    - **Sensor de distância:** eu já estou com o Puck?
- Verificar se o robô possui o puck é relativamente fácil
- Se alinhar com o puck e identificar sua cor é um pouco mais complicado…
  - É necessário utilizar uma biblioteca externa (no nosso caso, OpenCV)
  - A iluminação perfeita é essencial

Reconhecimento de imagens



Grab puck
LaRoCS

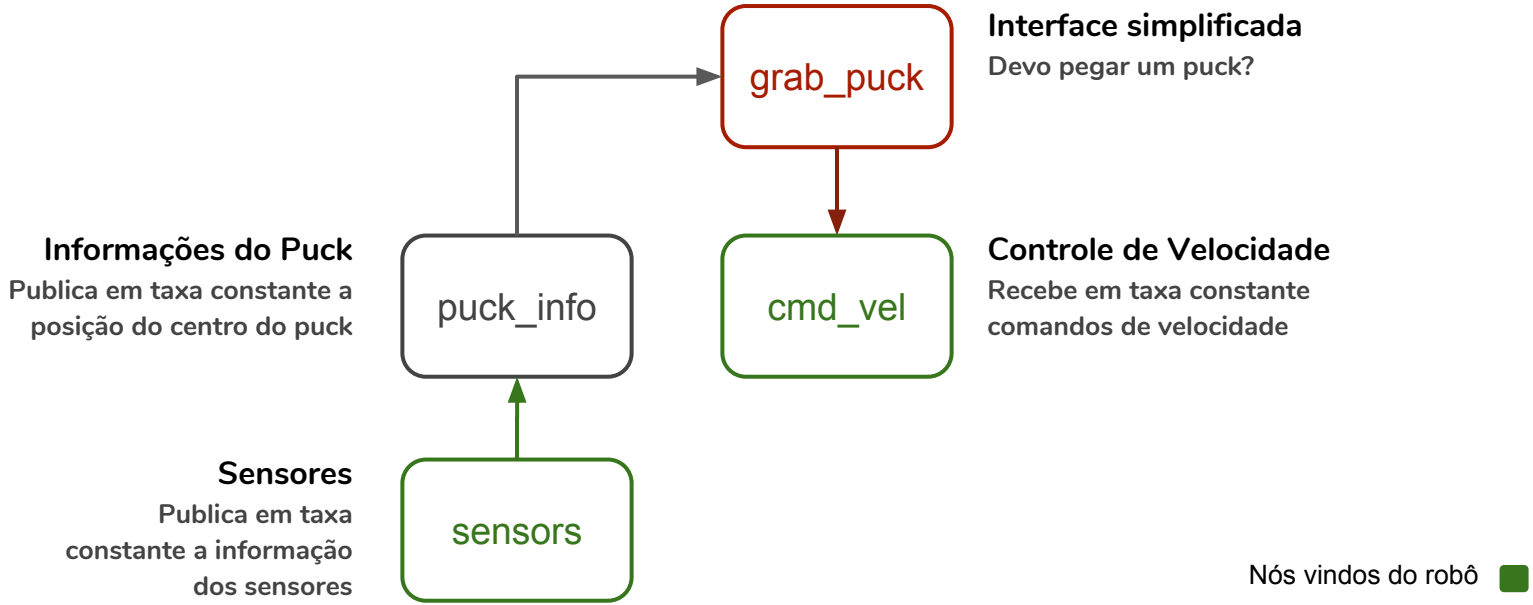
Pegando um Puck

- Agora podemos mover o robô na direção do puck
  - Usamos as informações adquiridas a partir dos sensores

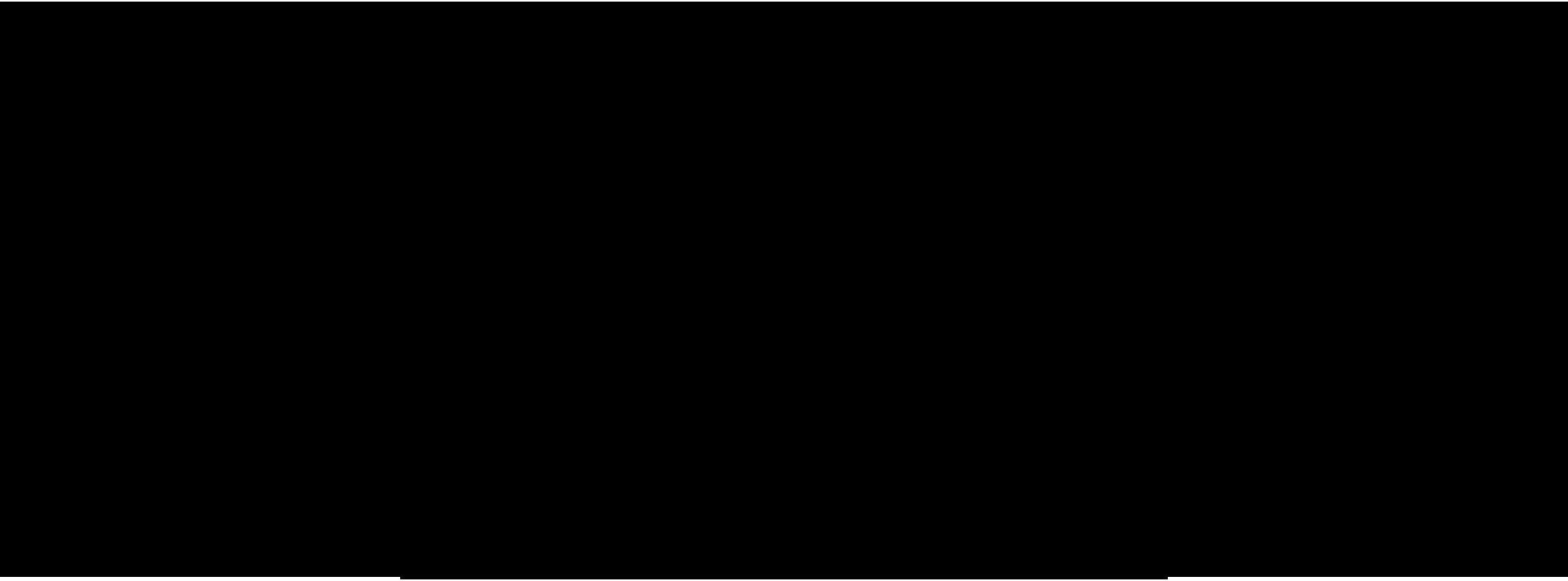
$$0.05\left(\tan^{-1}(0.025\,(0.9\times 260 - x) - 2.5) + \frac{\pi}{2}\right)$$

$$0.15\tan^{-1}(0.01\,(160 - x))$$

Pegando um Puck



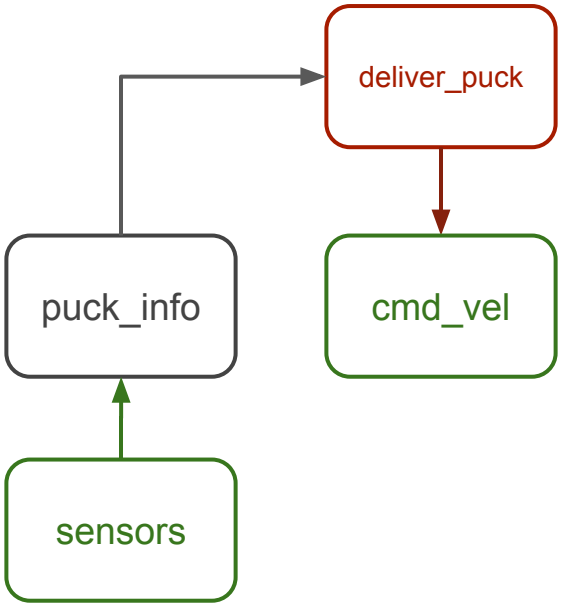
Pegando um Puck



Deixando um Puck

- Deixar um puck é um pouco mais difícil do que pegá-lo
  - O robô precisa de uma referência para saber que está de frente para o CD
  - As linhas de demarcação dos C.D.s são perfeitas pra isso
  - O DC é pequeno; pequenos erros de odometria podem levar o robô a derrubar uma parede

Pegando um Puck



Interface simplificada
Devo entregar um puck?

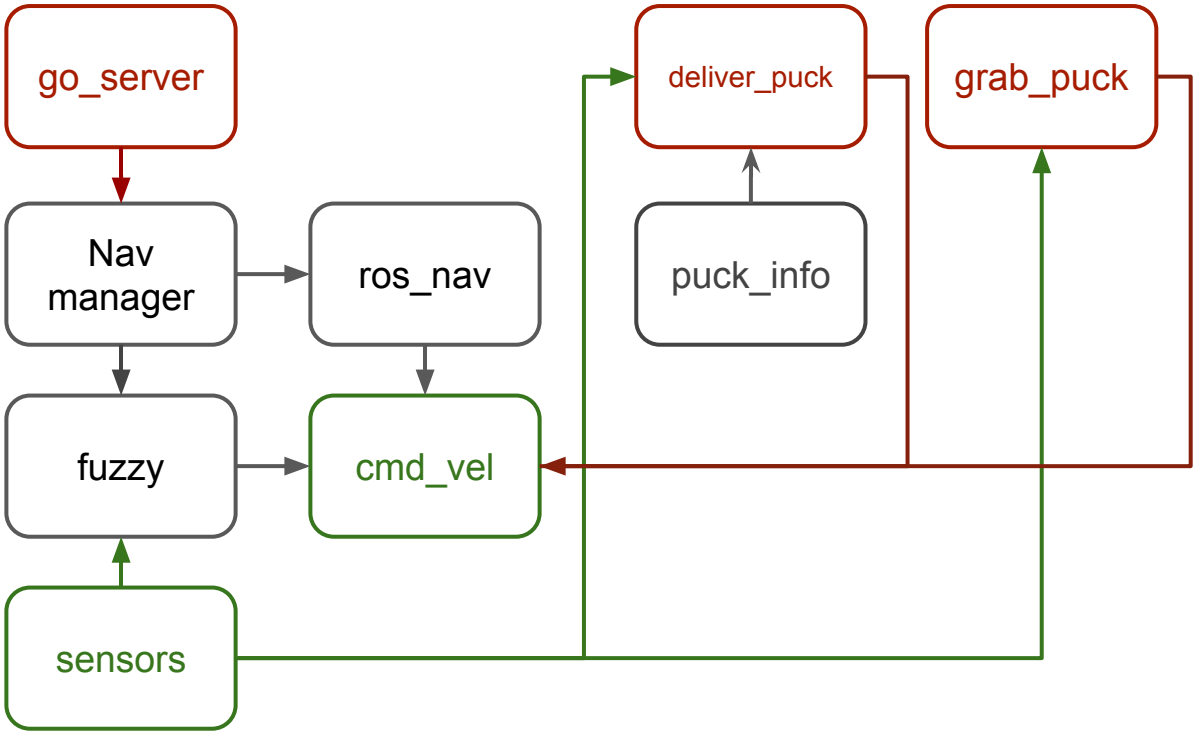
Informações do Puck
Publica em taxa constante a posição do centro do puck

Controle de Velocidade
Recebe em taxa constante comandos de velocidade

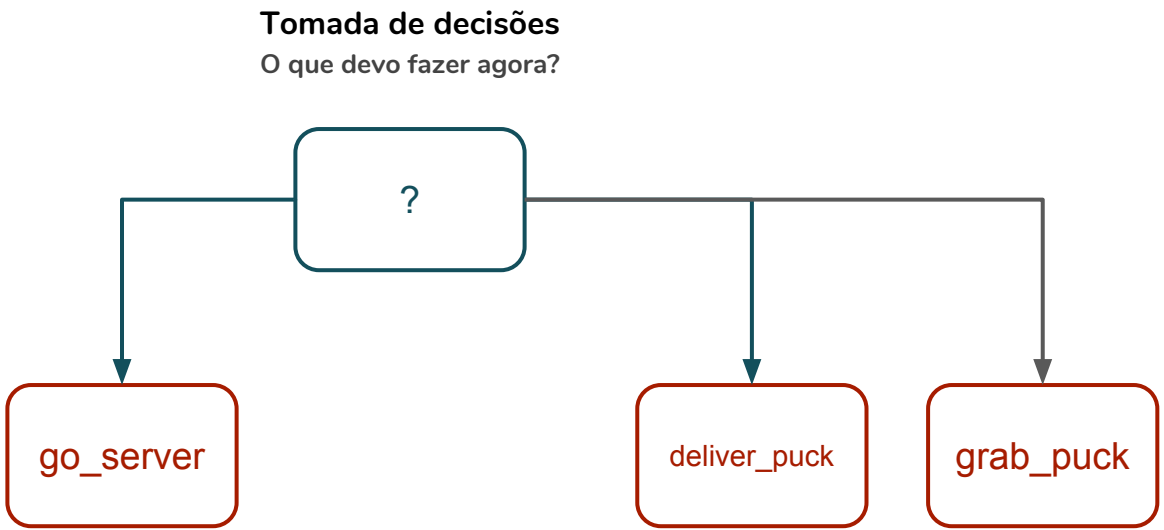
Sensores
Publica em taxa constante a informação dos sensores

Nós vindos do robô
Nós de ação
Nós de atuação interna às ações

Nossa arquitetura até agora



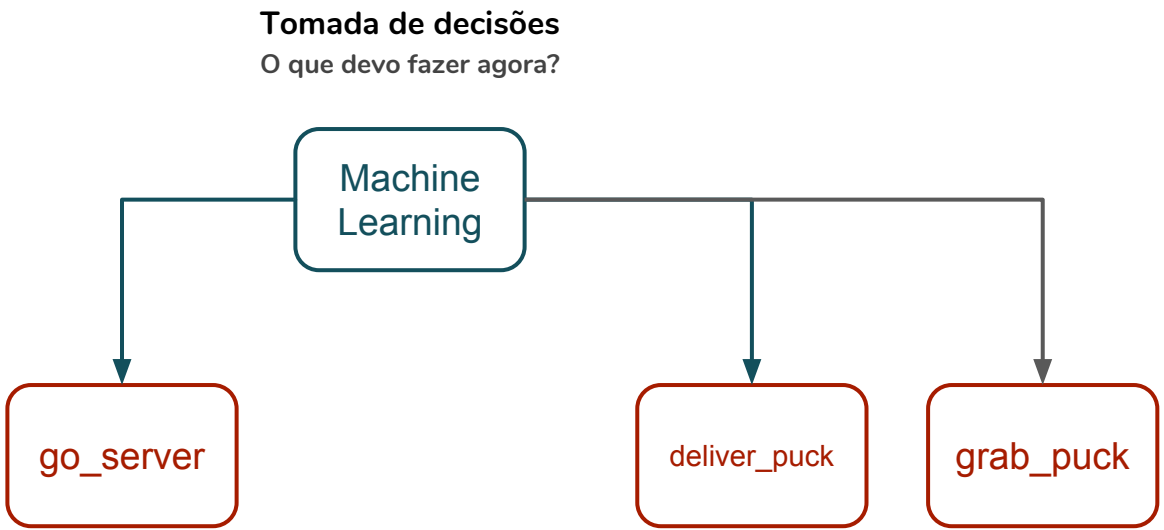
Nossa arquitetura até agora



Tomada de decisões

- Precisamos de um nó que coordene as ações
  - O nó só pode chamar uma ação por vez
  - O nó deve assumir que as ações sempre concluídas com sucesso
  - O nó deve conter um modelo interno da competição para saber o que fazer em seguida
  - **O nó deve maximizar a pontuação da competição**
- Máquina de estados?

Nossa arquitetura até agora



Tomada de decisões

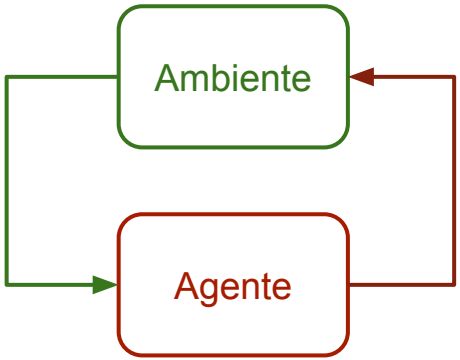
- A Indústria 4.0 requer soluções inteligentes
  - Algoritmos de decisão “hard-coded” funcionam bem dentro dos trilhos
    - Eles estão limitados apenas às situações que o programador conhece
    - Não são capazes de lidar com situações adversas
    - Conforme a tarefa cresce, a complexidade de um algoritmo do tipo também cresce e aumentam as chances de falhas (além de se tornar lento)
    - Nem sempre podem considerar todas as variáveis ligadas à produção
- Aprendizado de máquina é uma tecnologia viável

Tomada de decisões

- Deep Reinforcement Learning
  - Ramo de Machine Learning/IA que lida com problemas de tomada de decisão
  - Nestes problemas, geralmente há dependência temporal entre as ações
  - Utiliza Processos de Decisão de Markov como "*framework*"
  - De maneira simples, temos um *agente* interagindo com um *ambiente: S* é o *espaço de estados*, *A (já definimos isso antes!)* é o *espaço de ações*.
  - Um episódio dura um tempo *T* de iterações
  - Temos também uma política estocástica $\pi(s_t)$, tal que $a_t \sim \pi(s_t)$

Tomada de decisões

Estado s_{t+1}
Recompensa r_{t+1}



Ação a_t

$<s_0, a_0, r_1, s_1, a_1, r_2, \ldots>$

Tomada de decisões

- Deep Reinforcement Learning
  - O objetivo do agente é maximizar a recompensa total do episódio que, no nosso caso, é a pontuação da competição
  - Aumentamos a pontuação da competição ao passo que diminuímos o número de ações/tempo necessário para cumprir a tarefa

$$V_\pi(s) = \mathbb{E}[\sum_{i=1}^{T} \gamma^{i-1} r_i | S_t = s]$$

Tomada de decisões

- Deep Reinforcement Learning
  - Finalmente, podemos reduzir o problema a descobrir a função $\pi^*(s_t)$ que maximize a recompensa total (retorno) da competição
  - Mas com o que essa função se parece?

Tomada de decisões

- Deep Reinforcement Learning
  - Finalmente, podemos reduzir o problema a descobrir a função $\pi^*(s_t)$ que maximize a recompensa total (retorno) da competição
  - Mas com o que essa função se parece?
- De fato, não importa. Podemos usar uma Rede Neural que, dentre outras propriedades, **é capaz de aproximar qualquer função** [Universal Approximation Theorem]
  - Queremos então descobrir $\pi_\theta^*(s_t)$, onde θ é o vetor de parâmetros de uma rede neural

Tomada de decisões

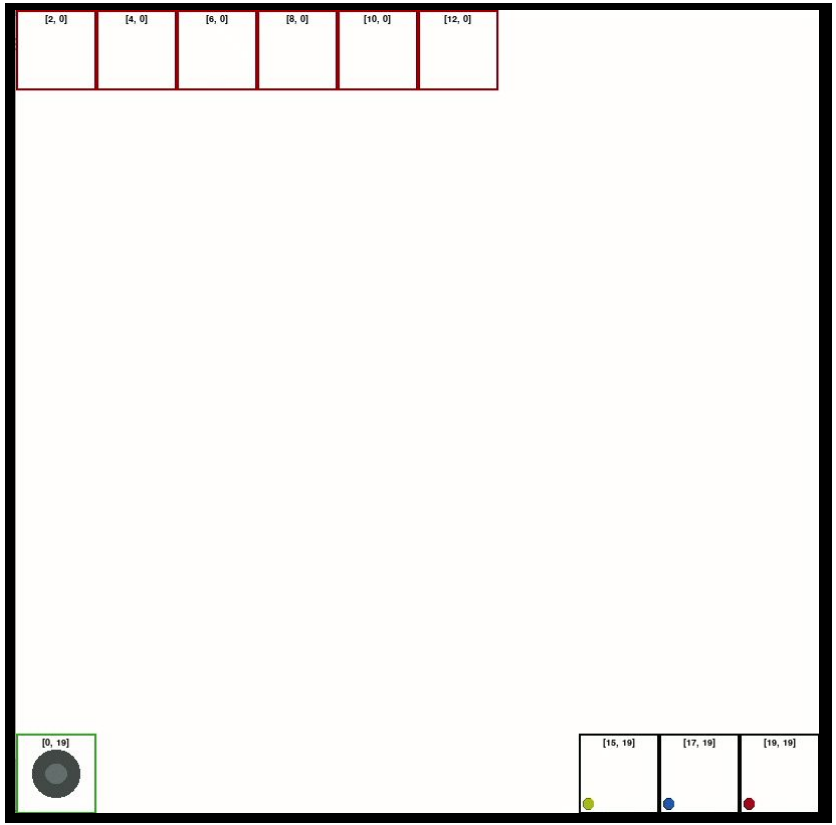
- Teorema da Politica do Gradiente

$$\frac{1}{T} \nabla_\theta \sum_{t=1}^{T} v_t log(\pi_\theta(a_t|s_t)) \qquad \text{(Policy Gradient Theorem)}$$

Tomada de decisões

- Do que esses algoritmos são capazes?
  - Quando bem treinadas, redes neurais são capazes de generalizar
  - Em tarefas como a da competição, comportamentos complexos podem emergir
  - Um destes comportamentos foi a “reposição reativa” de pucks, tarefa para a qual o robô não foi treinado

Tomada de decisões

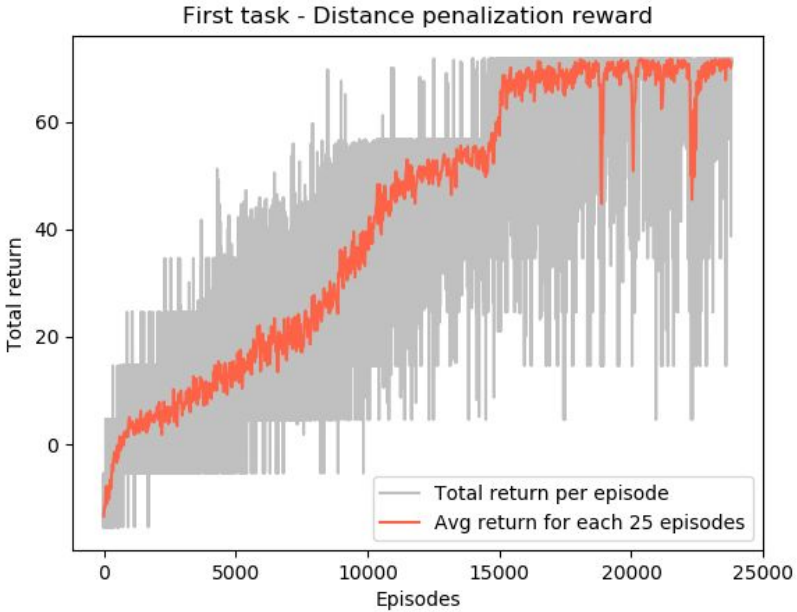


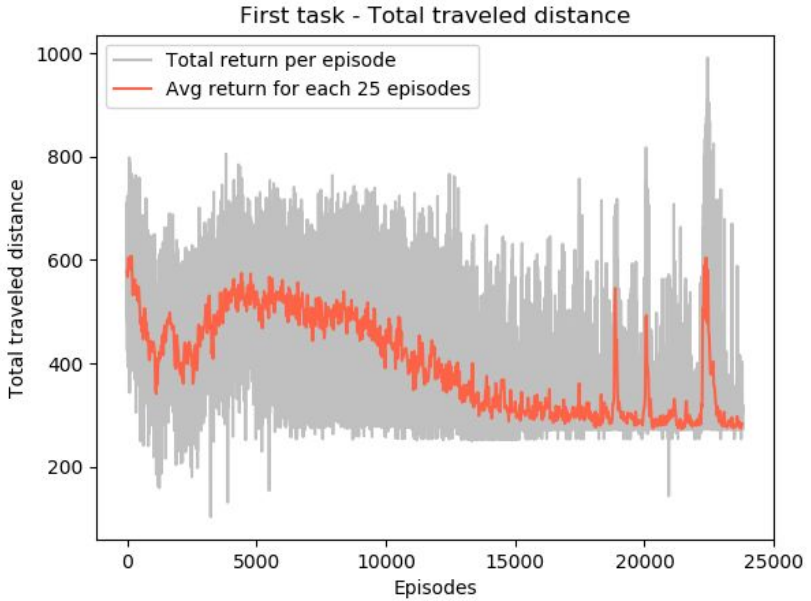
Puck Recover Behavior

O robô aprende a repor pucks

LAROCS
LABORATORY OF ROBOTICS AND COGNITIVE SYSTEMS

Tomada de decisões





O que vem agora?

- Estamos mais maduros em termo de base de código
  - O que nos dá tempo zzzzzzzz

Obrigado pelo seu tempo ;D

Perguntas?