Reinforcement Learning using OpenAI_ROS

Gabriel Paludo Licks

Mobile Intelligent Robotics 2018/2

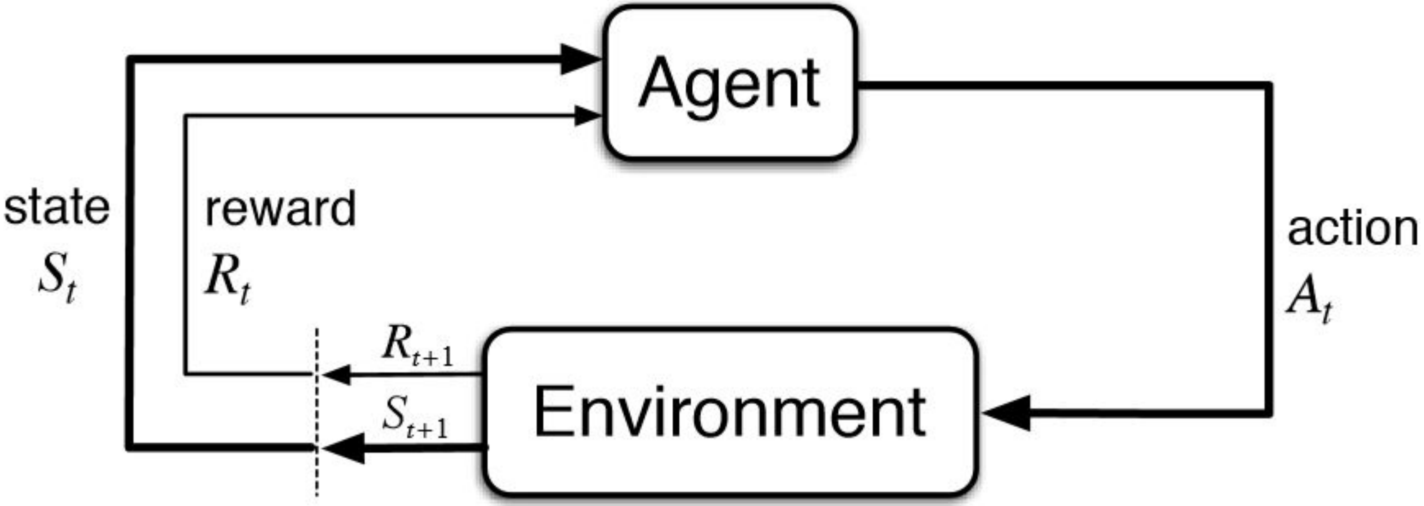
Outline

1. What's reinforcement learning?
2. OpenAI Gym
3. OpenAI_ROS package
    a. Tasks and environments
    b. Q-Learning
4. Experiments
5. Limitations
6. Final considerations

What’s reinforcement learning?

- Area of Machine Learning (among other learning paradigms)
- Learning paradigm that relates the most to how humans learn
    - We learn from actions (trial and error, positive or negative rewards)
- Approach to model agents in order to act optimally in unknown environments
- Environments can be formulated as Markov Decision Processes (MDPs)
    - States, actions, rewards, transitions
- What can I use it for?
    - Computer games
    - Robotics, industrial applications …

Agent-environment interaction in a MDP



Source: Sutton, R. S., Barto, A. G., & Bach, F. (1998). Reinforcement learning: An introduction. MIT press.

OpenAI Gym

- Gym is a toolkit for developing and comparing reinforcement learning algorithms
- It has a collection of test problems — environments — to use to work out reinforcement learning algorithms
- Provides the environment while users focus on the algorithm
- It supports teaching agents everything from walking to playing games like Pong or Pinball
- Has recently gained popularity in the machine learning community

Source: gym.openai.com

OpenAI Gym

Acrobot-v1
Swing up a two-link robot.

CartPole-v1
Balance a pole on a cart.

MountainCar-v0
Drive up a big hill.

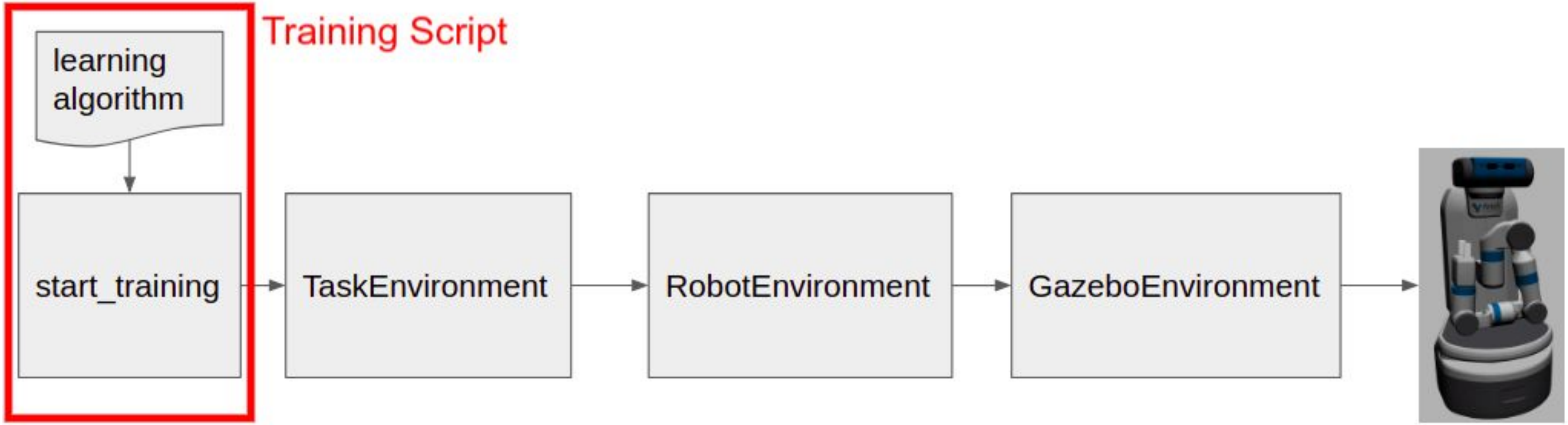
Source: gym.openai.com

OpenAI_ROS package

- Unfortunately, even if the Gym allows to train robots, does not provide environments to train ROS based robots using Gazebo simulations…
- The OpenAI_ROS package aims to provide the environments for ROS roboticists to have a common ground when training robots
- In general terms, the structure of the package can be divided in 2 big parts:
    - **Training Environments**: the ones in charge of providing to the learning algorithm all the data needed in order to make the robot learn.
    - **Training Script**: defines and sets up the learning algorithm that is going to be used in order to train the robot. This is where the user work takes place.

Source: wiki.ros.org/openai_ros

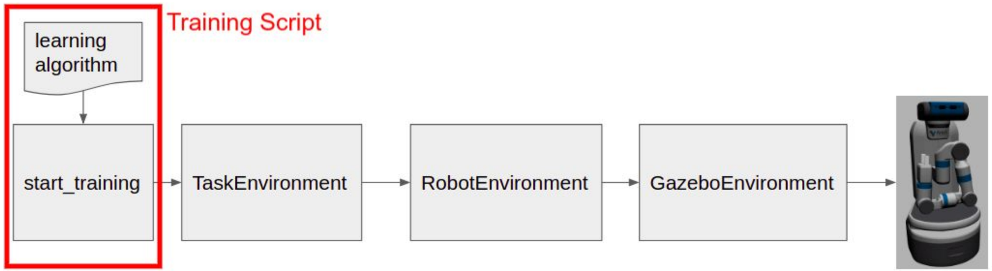
OpenAI_ROS pipeline



Source: wiki.ros.org/openai_ros

OpenAI_ROS pipeline

- It contains the **GazeboEnvironment** class that connects the OpenAI fundamental structure/methods to work with Gazebo
- It provides a group of already made **RobotEnvironments** for the most popular ROS-based robots
- It provides a group of already made **TaskEnvironments** to use together with the RobotEnvironments and GazeboEnvironment in order to train the robots



Source: wiki.ros.org/openai_ros

Gazebo class

- The **GazeboEnvironment** takes care of the resets of the simulator after each step, it also takes care of all the steps that need to be done on the simulator when doing a training step (typical in the RL loop)
- This class is the one that implements the functions required by the OpenAI Gym infrastructure:
  - step function
  - reset function
  - close function
- However, it calls children classes functions (on the **RobotEnvironment** and **TaskEnvironment** classes) to get the observations and apply the actions

Source: wiki.ros.org/openai_ros

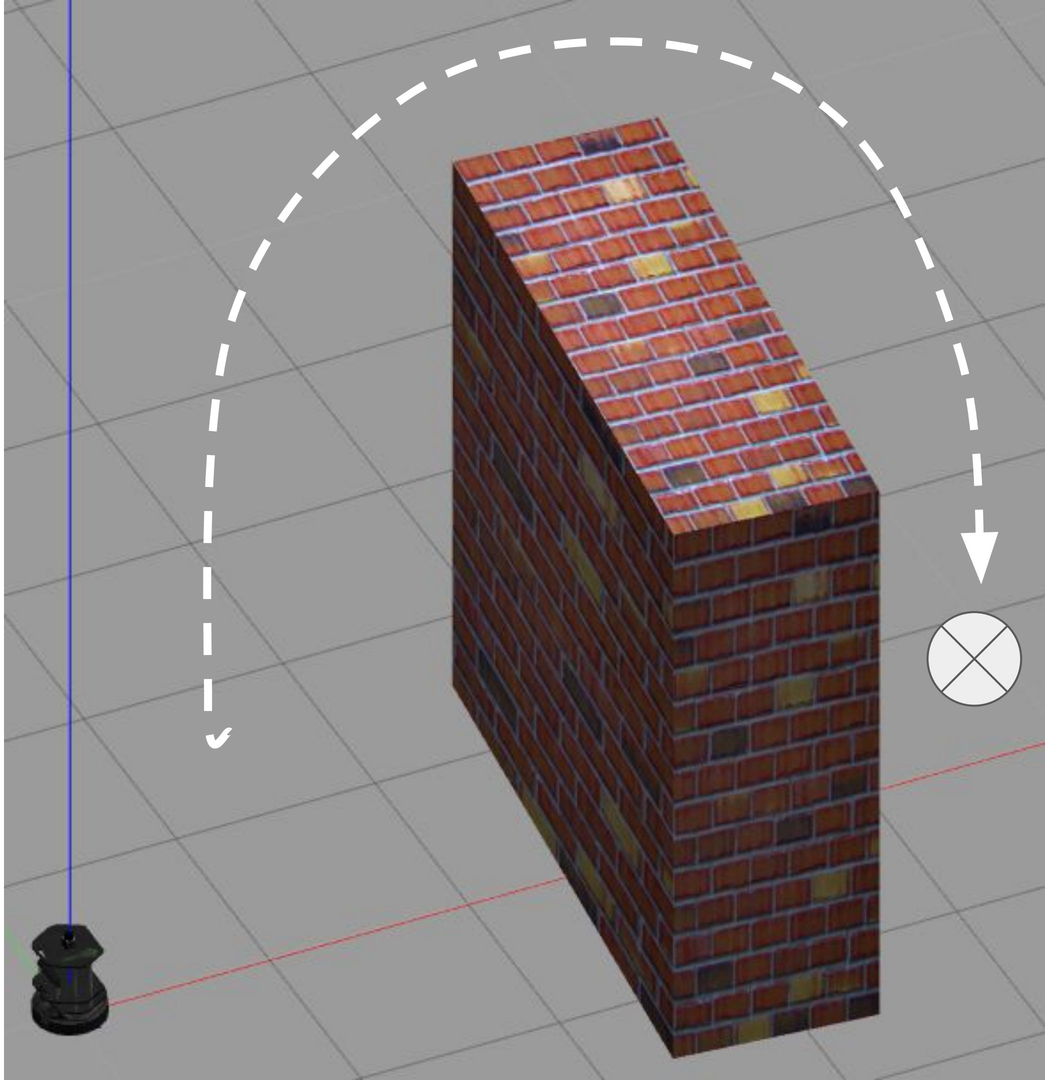
Robot and Task classes

- The **RobotEnvironment** contains all the functions associated to the specific robot that will be trained with all ROS functionalities that the robot will need in order to be controlled
  - Checks that every ROS stuff required is up and running on the robot (topics, services...)
- The **TaskEnvironment** class provides all the context for the task we want the robot to learn. This includes:
  - The available actions
  - The reward function
  - The state observation

Source: wiki.ros.org/openai_ros

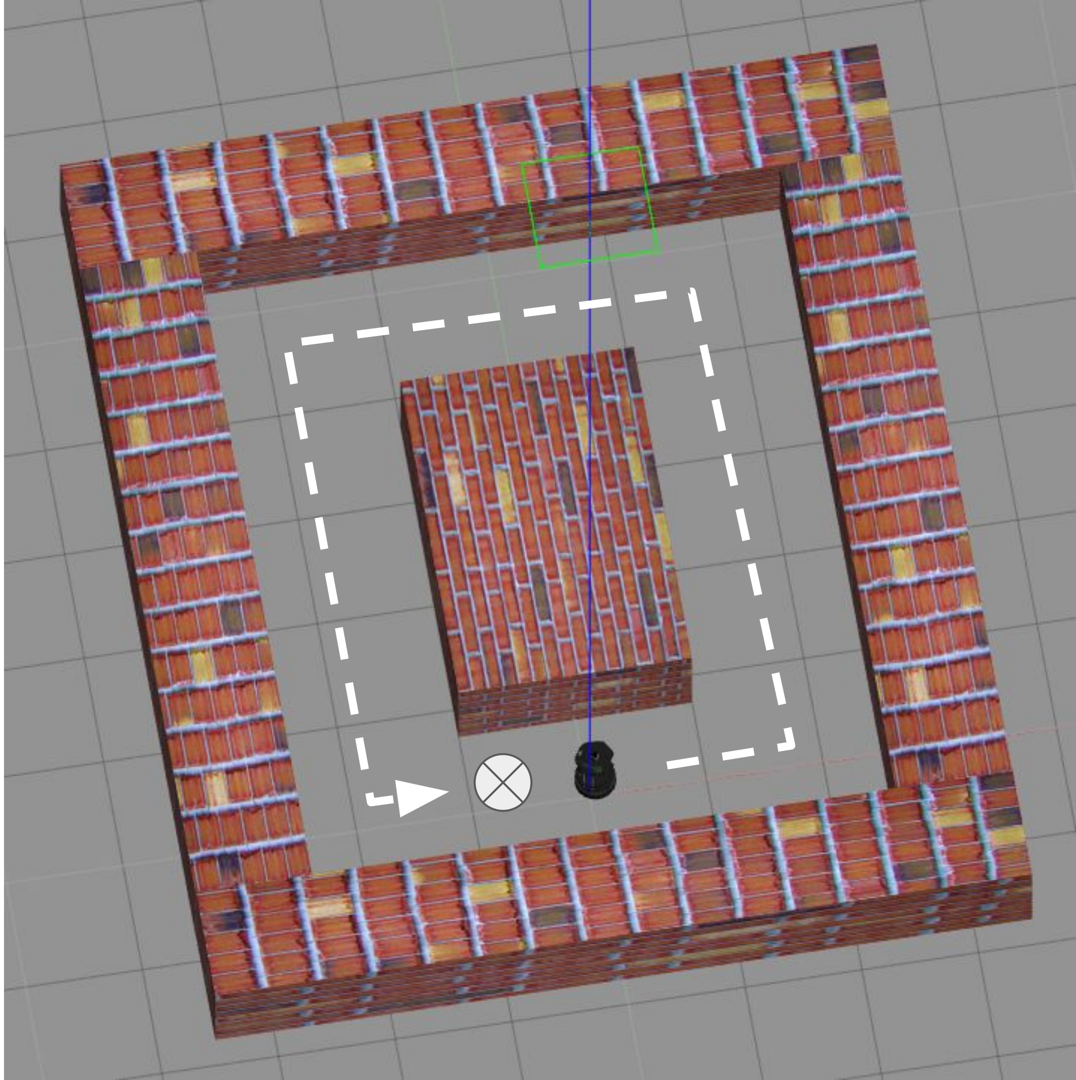
Wall environment

- **Robot**: Turtlebot 2
- **Task**: Learn to move avoiding the wall and reaching a center point on the backside
- Uses laser sensor to verify if robot is too close to the wall
- Uses XYZ location to determine current position



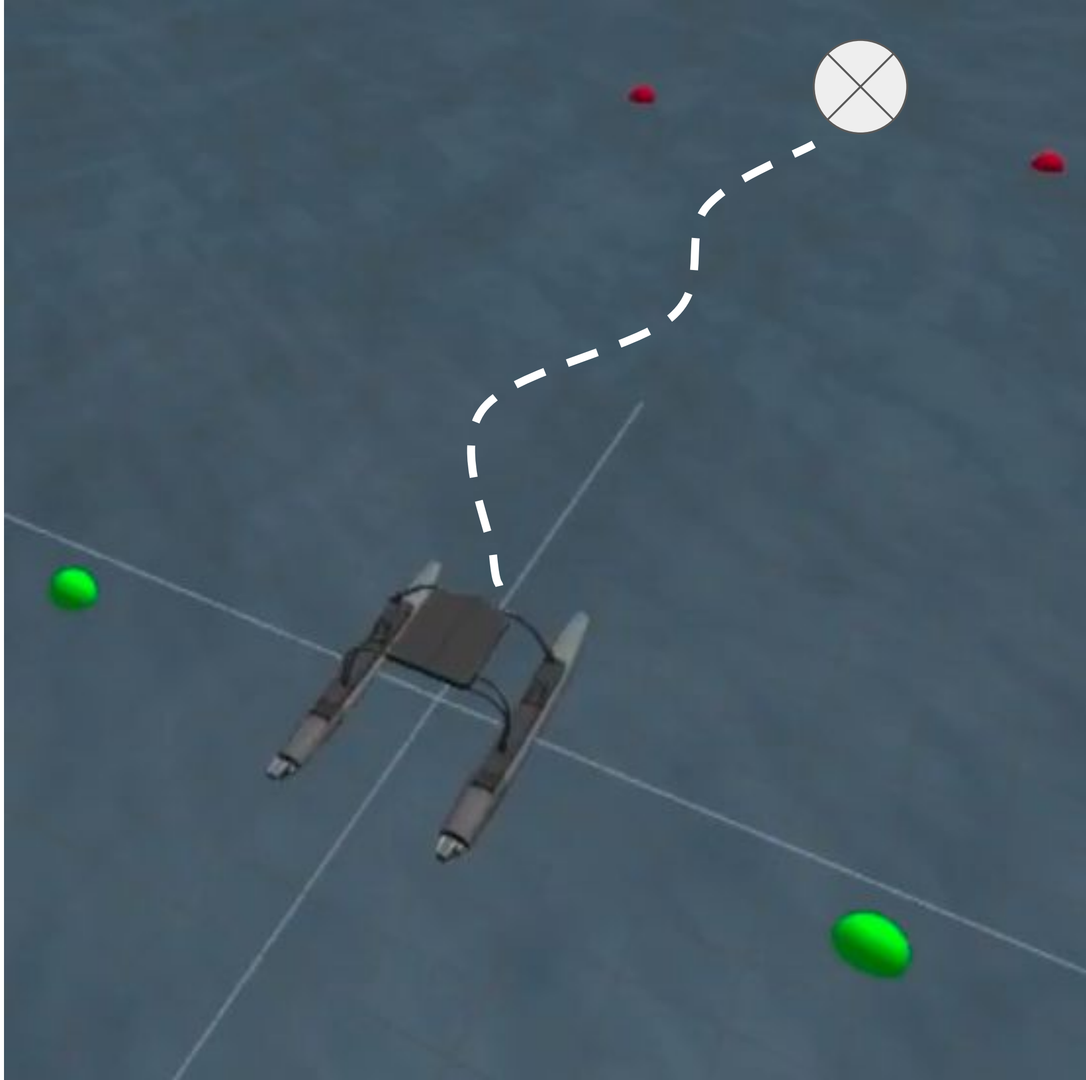
Maze environment

- **Robot**: Turtlebot 2
- **Task**: Learn to move through the maze without hitting the walls until it gets back to the starting position
- Uses laser sensor to verify if robot is too close to the wall
- Uses XYZ location to determine current position



Boat environment

- **Robot**: WAM-V boat from RobotX Challenge
- **Task**: Learn to get to the other side of the corridor determined by the buoys
- Uses XYZ location to determine current position and to check whether robot is out of the area demarcated by the buoys



Q-Learning

- Learns an action-utility representation using the notation Q(s, a)
- Incrementally estimates Q-values for actions based on reward signals
- Values learned by the agent are stored and represented in a tabular form
- Q-learning is an exploration-intensive method
  - We use an epsilon-greedy exploration function

$$Q(s,a) \leftarrow Q(s,a) + \alpha(R(s) + \gamma \max_{a'} Q(s',a') - Q(s,a))$$

	S_1	...	S_n
A_1	10	...	5
...	...	...	...
A_n	7	...	0

Q-Learning

function Q-LEARNING-AGENT(*percept*) **returns** an action
  inputs: *percept*, a percept indicating the current state s' and reward signal r'
  persistent: Q, a table of action values indexed by state and action, initially zero
    N_{sa}, a table of frequencies for state–action pairs, initially zero
    s, a, r, the previous state, action, and reward, initially null

  if TERMINAL?(s) **then** $Q[s, None] \leftarrow r'$
  if s is not null **then**
    increment $N_{sa}[s, a]$
    $Q[s, a] \leftarrow Q[s, a] + \alpha(N_{sa}[s, a])(r + \gamma \max_{a'} Q[s', a'] - Q[s, a])$
  $s, a, r \leftarrow s', \text{argmax}_{a'} f(Q[s', a'], N_{sa}[s', a']), r'$
  return a

Experiments

- We trained these robots in their respective available environments and observed their behavior throughout the episodes
- We trained the Turtlebot in both environments provided by the package, however no decent accumulated reward per episode was achieved
- For the WAM-V boat, we started to get decent accumulated rewards per episode after training over approximately 24 hours, which resulted in 6000 episodes of training.
- These iterations do not seem to bring the algorithm close to convergence
    - Different tweaks with different values of learning rate, discount factor and epsilon decay
- The robots have learned suboptimal policies, normally with unstable behavior

Experiments

- There are two reasons that strongly relate to this result:
  - The number of iterations and/or the environment modelling
- The number of iterations is fundamental for learning optimal policies by exploring all states and actions in an environment
  - If an agent does not iterate enough, the policies tend to be always suboptimal due to the lack of environment knowledge
- In our case, it is likely that the environment modelling provided by the package results in a large state space
  - Every XYZ point location in the map represents a new state for the robot agent
- For this reason, there might not be an estimated number of iterations needed in order to learn optimal policies and learn consistent state-action pair values

Limitations

- The main limitation encountered during the experiments is training duration
  - We found that the simulation could not be accelerated significantly for speeding up the training
- Gazebo simulation tends to be unstable and inaccurate if accelerated
  - When testing with a Turtlebot, increasing the real time factor of the simulation caused the robot to crash into the walls
  - The reason for this to happen is most likely to the lack of processing power for physics and not interpreting the laser signals to stop the robot
- This package still only supports working with Gazebo

Limitations

- Another reason that may cause slow convergence is the large state space along with non-deterministic transitions due to the simulator's physics

- The state space of the environment is represented by each XYZ point
  - It considerably raises the number of combinations for state-action pairs that have to be learned by the agent through iterations

- The transition from one state to another is also non-deterministic
  - Which means that choosing an action in a state does not guarantee arriving in the expected state (e.g. the wind, in the WAM-V environment)

Limitations

- In general, installation is not a very clear process
- Instead of installing the OpenAI_ROS package and its dependencies in the user's local machine, the authors tend to recommend their web platform called ROS Development Studio
- They provide the infrastructure and a ready-to-use ROS environment with all dependencies met
  - Only available for free at a low CPU power
  - For a 30 hours limited session
- Thus, the use of this platform is out of the scope of this report.

Final considerations

- Training in reinforcement learning is often a time consuming task
    - Is directly related to the size of the state space
- It would be necessary for the training process to be held in a more powerful environment than a local machine in order to converge faster
- The OpenAI_ROS is yet a tentative solution for integrating OpenAI with ROS simulated robots that has recently started to be developed
    - Nonetheless, an important tool for the use of RL in the context of robotics
- Definitely, there are improvements necessary for the environments available and also explicit the constraints that may arise throughout its use

Thank you!

Questions?

Gabriel Paludo Licks
gabriel.paludo@acad.pucrs.br