Robot Programming with Lisp

1. Introduction, Setup

Gayane Kazhoyan

Institute for Artificial Intelligence
University of Bremen

18th October, 2018

General Info

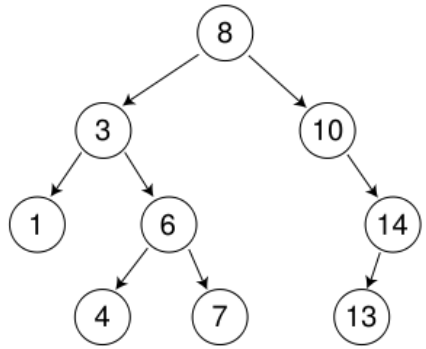
- Lecturer: Gaya (PhD student at IAI)
- Tutor: Arthur (HiWi at IAI)
- Correspondence: gaya@cs.uni-bremen.de, artnie91@cs.uni-bremen.de
- Dates: Thursdays, 14:15 - 15:45, 16:15 - 17:45
- Language: English and German
- Credits: 6 ECTS (4 SWS)
- Course type: practical course
- Course number: 03-BE-710.98d
- Location: TAB Building, Room 0.36 EG

Plan

Course content

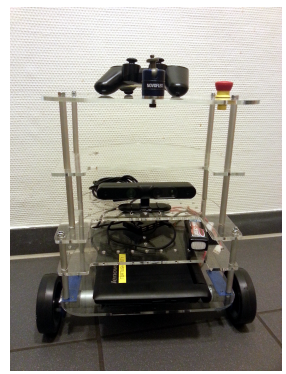
Common Lisp

Artificial Intelligence



Robot Operating System (ROS)

Robot platform



Common Lisp

- Full-featured industry-standard programming language

Common Lisp

- Full-featured industry-standard programming language
- Means for functional programming
- Means for imperative programming
- Means for OOP

Common Lisp

- Full-featured industry-standard programming language
- Means for functional programming
- Means for imperative programming
- Means for OOP
- Fast prototyping through read-eval-print loop and dynamic typing

Common Lisp

- Full-featured industry-standard programming language
- Means for functional programming
- Means for imperative programming
- Means for OOP
- Fast prototyping through read-eval-print loop and dynamic typing
- Compiles into machine code

Common Lisp

- Full-featured industry-standard programming language
- Means for functional programming
- Means for imperative programming
- Means for OOP
- Fast prototyping through read-eval-print loop and dynamic typing
- Compiles into machine code
- Best choice for symbolic processing (AI, theorem proving, etc.)

Common Lisp

- Full-featured industry-standard programming language
- Means for functional programming
- Means for imperative programming
- Means for OOP
- Fast prototyping through read-eval-print loop and dynamic typing
- Compiles into machine code
- Best choice for symbolic processing (AI, theorem proving, etc.)
- Good choice for writing domain-specific programming languages (e.g., robot programming languages)

Common Lisp

- Full-featured industry-standard programming language
- Means for functional programming
- Means for imperative programming
- Means for OOP
- Fast prototyping through read-eval-print loop and dynamic typing
- Compiles into machine code
- Best choice for symbolic processing (AI, theorem proving, etc.)
- Good choice for writing domain-specific programming languages (e.g., robot programming languages)

Applications using / written in dialects of Lisp:
Emacs, AutoCAD, Grammarly, Mirai (Gollum animation), Google ITA (airplane ticket price planner AI), DART (DARPA logistics AI), Maxima (computer algebra system), AI and robotics frameworks, ...

ROS

- Middleware for communication of the components of a robotic system

ROS

- Middleware for communication of the components of a robotic system
- "Meta-Operating System" for programming robotics software (configuring, starting / stopping, logging etc. software components)

ROS

- Middleware for communication of the components of a robotic system
- "Meta-Operating System" for programming robotics software (configuring, starting / stopping, logging etc. software components)
- Powerful build system (based on CMake), with a strong focus on integration and documentation

ROS

- Middleware for communication of the components of a robotic system
- "Meta-Operating System" for programming robotics software (configuring, starting / stopping, logging etc. software components)
- Powerful build system (based on CMake), with a strong focus on integration and documentation
- Language-independent architecture (C++, Python, Lisp, Java, JavaScript, ...)

ROS

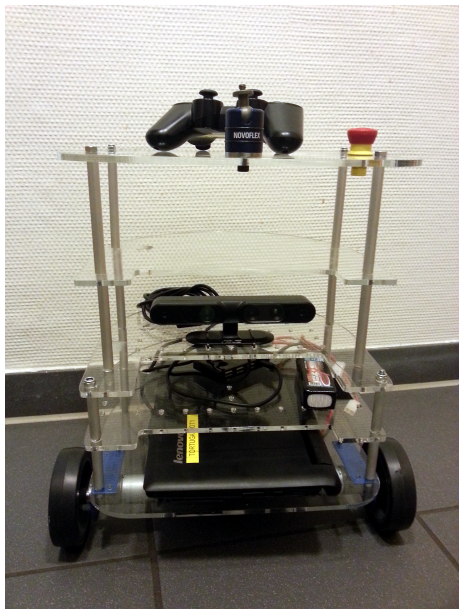
- Middleware for communication of the components of a robotic system
- "Meta-Operating System" for programming robotics software (configuring, starting / stopping, logging etc. software components)
- Powerful build system (based on CMake), with a strong focus on integration and documentation
- Language-independent architecture (C++, Python, Lisp, Java, JavaScript, ...)
- According to ROS 2018 Community Metrics Report,
  - More than 1.6 million unique pageviews of `wiki.ros.org` a month
  - More than 16 million downloads of `.deb` packages a month

ROS

- Middleware for communication of the components of a robotic system
- "Meta-Operating System" for programming robotics software (configuring, starting / stopping, logging etc. software components)
- Powerful build system (based on CMake), with a strong focus on integration and documentation
- Language-independent architecture (C++, Python, Lisp, Java, JavaScript, ...)
- According to ROS 2018 Community Metrics Report,
  - More than 1.6 million unique pageviews of `wiki.ros.org` a month
  - More than 16 million downloads of `.deb` packages a month
- *De facto* standard in modern robotics

TortugaBot

- 2 controllable wheels
- 2D laser scanner
- Optional 2.5D vision sensor
- Asus Eee PC with bluetooth
- Optional basket in the top part
- PlayStation joystick



Why Lisp for robots?

- ROS supports a number of languages: C++, Python, Lisp and Java

Why Lisp for robots?

- ROS supports a number of languages: C++, Python, Lisp and Java
- Lisp is good for rapid prototyping

Why Lisp for robots?

- ROS supports a number of languages: C++, Python, Lisp and Java
- Lisp is good for rapid prototyping
- It is more suitable for symbolic reasoning and AI

Why Lisp for robots?

- ROS supports a number of languages: C++, Python, Lisp and Java
- Lisp is good for rapid prototyping
- It is more suitable for symbolic reasoning and AI
- There are existing robot programming languages in Lisp that automate decision making

Rough schedule

- Introduction, Setup

- Lisp basics
- Functional programming
- OOP

- ROS, ROS Lisp API (*roslisp*)
- *roslisp*, 2D world of *turtlesim*
- coordinate frames, *tf*

- TortugaBot, navigation
- Collision avoidance
- Project scenario
- Project implementation
- The big day: competition

Software requirements

Bringing a *personal laptop* is encouraged.

OS:	Ubuntu 16.04 (18.04 or 14.04 with manual setting up)
IDE:	Emacs 24+
Version control:	Git
Packaging system:	ROS
Lisp software:	SBCL compiler, ASDF build system, Emacs plugin for Common Lisp

Bottom line

You will learn / improve your skills in the following:

- Linux
- Git
- Emacs
- Functional programming
- Common Lisp, of course
- ROS (for future roboticists)

...and get to play with a real little robot!

Plan

Grading

- Course final grade: 100 points = 50 homework + 50 group project.

Grading

- Course final grade: 100 points = 50 homework + 50 group project.
- To participate in the project you need at least 25 points from the homeworks, otherwise it's a fail.

Grading

- Course final grade: 100 points = 50 homework + 50 group project.
- To participate in the project you need at least 25 points from the homeworks, otherwise it's a fail.
- Final grade: 50 of 100 points - 4.0, 100 of 100 points - 1.0.

Homework assignments

- Homework assignments will mostly consist of filling in the missing gaps in already existing code.

Homework assignments

- Homework assignments will mostly consist of filling in the missing gaps in already existing code.
- That code will be hosted on GitLab.

Homework assignments

- Homework assignments will mostly consist of filling in the missing gaps in already existing code.
- That code will be hosted on GitLab.
- The code you write should be uploaded to GitLab (`https://gitlab.informatik.uni-bremen.de/`).

Homework assignments

- Homework assignments will mostly consist of filling in the missing gaps in already existing code.
- That code will be hosted on GitLab.
- The code you write should be uploaded to GitLab (`https://gitlab.informatik.uni-bremen.de/`).
- Homework is due in one week.

Homework assignments

- Homework assignments will mostly consist of filling in the missing gaps in already existing code.
- That code will be hosted on GitLab.
- The code you write should be uploaded to GitLab (`https://gitlab.informatik.uni-bremen.de/`).
- Homework is due in one week.
- Solutions are discussed in the tutorial.

Homework assignments

- Homework assignments will mostly consist of filling in the missing gaps in already existing code.
- That code will be hosted on GitLab.
- The code you write should be uploaded to GitLab (`https://gitlab.informatik.uni-bremen.de/`).
- Homework is due in one week.
- Solutions are discussed in the tutorial.
- Can get 60 of 50 points in homework (can skip one homework).

Homework assignments

- Homework assignments will mostly consist of filling in the missing gaps in already existing code.
- That code will be hosted on GitLab.
- The code you write should be uploaded to GitLab (`https://gitlab.informatik.uni-bremen.de/`).
- Homework is due in one week.
- Solutions are discussed in the tutorial.
- Can get 60 of 50 points in homework (can skip one homework).
- Bonus points for very good homework solutions.

Links

- Emacs cheat sheet:

  `https://www.gnu.org/software/emacs/refcards/pdf/refcard.pdf`

- Git reference book:

  `http://git-scm.com/book/de`

- Lisp books:

  `http://landoflisp.com/`, `http://www.paulgraham.com/onlisp.html`, `http://www.gigamonkeys.com/book/`

Info summary

Next class:

- Date: 25.10
- Time: 14:15 (14:00 - 14:15 for questions)
- Place: same room (TAB 0.36)

Assignment:

- Due: 24.10, Wednesday, 23:59
- Points: 3 points
- For questions: write an email to Arthur or Gaya

Plan

Assignment goals

Set up your working environment

Set up your Git repositories

Get comfortable with Emacs

Task 1: Install Ubuntu 16.04

- Find out your processor architecture (32 vs. 64 bit).
  Hint: unless your computer is very old, it's most likely 64 bit.

Task 1: Install Ubuntu 16.04

- Find out your processor architecture (32 vs. 64 bit).
  Hint: unless your computer is very old, it's most likely 64 bit.
- Download Ubuntu 16.04 installation `.iso`
  (`ubuntu-16.04.5-desktop-amd64.iso`)
  `http://releases.ubuntu.com/16.04/`

Task 1: Install Ubuntu 16.04

- Find out your processor architecture (32 vs. 64 bit).
  Hint: unless your computer is very old, it's most likely 64 bit.
- Download Ubuntu 16.04 installation `.iso`
  (`ubuntu-16.04.5-desktop-amd64.iso`)
  `http://releases.ubuntu.com/16.04/`
- Create a boot USB with the `.iso` (or burn a DVD).
  Hint: for a bootable USB, in Windows use the Universal USB installer:
  `http://www.pendrivelinux.com/universal-usb-installer-easy-as-1-2-3/`;
  and in Linux you could, e.g., use the Startup Disk Creator or `unetbootin`.

Task 1: Install Ubuntu 16.04

- Find out your processor architecture (32 vs. 64 bit).
  Hint: unless your computer is very old, it's most likely 64 bit.
- Download Ubuntu 16.04 installation `.iso`
  (`ubuntu-16.04.5-desktop-amd64.iso`)
  `http://releases.ubuntu.com/16.04/`
- Create a boot USB with the `.iso` (or burn a DVD).
  Hint: for a bootable USB, in Windows use the Universal USB installer:
  `http://www.pendrivelinux.com/universal-usb-installer-easy-as-1-2-3/`;
  and in Linux you could, e.g., use the Startup Disk Creator or `unetbootin`.
- Install Ubuntu 16.04 (aka Xenial).
  Dual boot installation with default settings is a one click thing.

Task 1: Install Ubuntu: FAQ

- *How do I boot from USB / CD?*
  You should enter either “Boot Menu” or “BIOS Menu” during reboot
  `https://www.desertcrystal.com/bootkeys`

Task 1: Install Ubuntu: FAQ

- *How do I boot from USB / CD?*
  You should enter either "Boot Menu" or "BIOS Menu" during reboot
  `https://www.desertcrystal.com/bootkeys`
- *Windows 8+ doesn't let me into "BIOS Menu"!*
  You should restart into the "Boot Options Menu" of your Windows:
  hold down "Shift" while pressing "Restart".
  `http://www.makeuseof.com/tag/how-to-access-the-bios-on-a-windows-8-computer/`

Task 1: Install Ubuntu: FAQ

- *How do I boot from USB / CD?*
  You should enter either "Boot Menu" or "BIOS Menu" during reboot
  `https://www.desertcrystal.com/bootkeys`
- *Windows 8+ doesn't let me into "BIOS Menu"!*
  You should restart into the "Boot Options Menu" of your Windows:
  hold down "Shift" while pressing "Restart".
  `http://www.makeuseof.com/tag/how-to-access-the-bios-on-a-windows-8-computer/`
- *My BIOS supports UEFI, Ubuntu won't install!*
  It should work but if you can't get it to run turn off the UEFI mode:
  restart into the "Boot Options Menu" of your Windows,
  choose "Troubleshoot", then "UEFI Firmware Settings"
  `http://www.makeuseof.com/tag/how-to-access-the-bios-on-a-windows-8-computer/`

Task 1: Install Ubuntu: FAQ

- *How do I boot from USB / CD?*
  You should enter either "Boot Menu" or "BIOS Menu" during reboot
  `https://www.desertcrystal.com/bootkeys`
- *Windows 8+ doesn't let me into "BIOS Menu"!*
  You should restart into the "Boot Options Menu" of your Windows:
  hold down "Shift" while pressing "Restart".
  `http://www.makeuseof.com/tag/how-to-access-the-bios-on-a-windows-8-computer/`
- *My BIOS supports UEFI, Ubuntu won't install!*
  It should work but if you can't get it to run turn off the UEFI mode:
  restart into the "Boot Options Menu" of your Windows,
  choose "Troubleshoot", then "UEFI Firmware Settings"
  `http://www.makeuseof.com/tag/how-to-access-the-bios-on-a-windows-8-computer/`
- *It still doesn't work!*
  Write an email to Arthur or Gaya

Task 2: Install ROS

Consult the official installation instructions for troubleshooting:
`http://wiki.ros.org/kinetic/Installation/Ubuntu`

In short, it boils down to executing the following in the terminal (*hint*: to open a fresh terminal press `<Ctrl>+<Alt>+t`):

- Add ROS repositories to your sources list:

```
sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu xenial main" > /etc/apt/sources.list.d/ros-latest.list'
```

Task 2: Install ROS

Consult the official installation instructions for troubleshooting:
`http://wiki.ros.org/kinetic/Installation/Ubuntu`

In short, it boils down to executing the following in the terminal (*hint*: to open a fresh terminal press `<Ctrl>+<Alt>+t`):

- Add ROS repositories to your sources list:

```
sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu xenial main" > /etc/apt/sources.list.d/ros-latest.list'
```

- Add their key to your trusted public keys:

```
sudo apt-key adv --keyserver hkp://ha.pool.sks-keyservers.net:80 --recv-key 421C365BD9FF1F717815A3895523BAEEB01FA116
```

Task 2: Install ROS

Consult the official installation instructions for troubleshooting:
`http://wiki.ros.org/kinetic/Installation/Ubuntu`

In short, it boils down to executing the following in the terminal (*hint*: to open a fresh terminal press `<Ctrl>+<Alt>+t`):

- Add ROS repositories to your sources list:

```
sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu xenial main" > /etc/apt/sources.list.d/ros-latest.list'
```

- Add their key to your trusted public keys:

```
sudo apt-key adv --keyserver hkp://ha.pool.sks-keyservers.net:80 --recv-key 421C365BD9FF1F717815A3895523BAEEB01FA116
```

- Update your Debian package index:

```
sudo apt-get update
```

Task 2: Install ROS

Consult the official installation instructions for troubleshooting:
`http://wiki.ros.org/kinetic/Installation/Ubuntu`

In short, it boils down to executing the following in the terminal (*hint*: to open a fresh terminal press <Ctrl>+<Alt>+t):

- Add ROS repositories to your sources list:
  ```
  sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu xenial main" > /etc/apt/sources.list.d/ros-latest.list'
  ```
- Add their key to your trusted public keys:
  ```
  sudo apt-key adv --keyserver hkp://ha.pool.sks-keyservers.net:80 --recv-key 421C365BD9FF1F717815A3895523BAEEB01FA116
  ```
- Update your Debian package index:
  ```
  sudo apt-get update
  ```
- The version of ROS distributed with Ubuntu 16.04 is **ROS Kinetic**. Install the **desktop** package. Say <No> if asked about `hddtemp`.
  ```
  sudo apt-get install ros-kinetic-desktop
  ```

Task 2: Install ROS

Consult the official installation instructions for troubleshooting:
`http://wiki.ros.org/kinetic/Installation/Ubuntu`

In short, it boils down to executing the following in the terminal (*hint*: to open a fresh terminal press <Ctrl>+<Alt>+t):

- Add ROS repositories to your sources list:
  ```
  sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu xenial main" > /etc/apt/sources.list.d/ros-latest.list'
  ```
- Add their key to your trusted public keys:
  ```
  sudo apt-key adv --keyserver hkp://ha.pool.sks-keyservers.net:80 --recv-key 421C365BD9FF1F717815A3895523BAEEB01FA116
  ```
- Update your Debian package index:
  ```
  sudo apt-get update
  ```
- The version of ROS distributed with Ubuntu 16.04 is **ROS Kinetic**. Install the **desktop** package. Say <No> if asked about `hddtemp`.
  ```
  sudo apt-get install ros-kinetic-desktop
  ```
- Install the workspace management tools:
  ```
  sudo apt-get install python-rosinstall python-wstool
  ```

Task 3: Setup ROS

Consult the official installation instructions for troubleshooting:
`http://wiki.ros.org/kinetic/Installation/Ubuntu`

In short, it boils down to executing the following in the terminal:

- Setup rosdep:

```
sudo rosdep init && rosdep update
```

Task 3: Setup ROS

Consult the official installation instructions for troubleshooting:
`http://wiki.ros.org/kinetic/Installation/Ubuntu`

In short, it boils down to executing the following in the terminal:

- Setup rosdep:

  ```
  sudo rosdep init && rosdep update
  ```

- Initialize the ROS environment for this particular terminal:

  ```
  source /opt/ros/kinetic/setup.bash
  ```

Task 3: Setup ROS

Consult the official installation instructions for troubleshooting:
`http://wiki.ros.org/kinetic/Installation/Ubuntu`

In short, it boils down to executing the following in the terminal:

- Setup rosdep:
  ```
  sudo rosdep init && rosdep update
  ```
- Initialize the ROS environment for this particular terminal:
  ```
  source /opt/ros/kinetic/setup.bash
  ```
- Create a directory where the code you'll write will be stored (the name `ros_ws` and the location ~ can be changed):
  ```
  mkdir -p ~/ros_ws/src
  ```

Task 3: Setup ROS

Consult the official installation instructions for troubleshooting:
`http://wiki.ros.org/kinetic/Installation/Ubuntu`

In short, it boils down to executing the following in the terminal:

- Setup rosdep:
  ```
  sudo rosdep init && rosdep update
  ```
- Initialize the ROS environment for this particular terminal:
  ```
  source /opt/ros/kinetic/setup.bash
  ```
- Create a directory where the code you'll write will be stored (the name `ros_ws` and the location ~ can be changed):
  ```
  mkdir -p ~/ros_ws/src
  ```
- Initialize the workspace:
  ```
  cd ~/ros_ws && catkin_make
  ```

Task 3: Setup ROS

Consult the official installation instructions for troubleshooting:
`http://wiki.ros.org/kinetic/Installation/Ubuntu`

In short, it boils down to executing the following in the terminal:

- Setup rosdep:
  ```
  sudo rosdep init && rosdep update
  ```
- Initialize the ROS environment for this particular terminal:
  ```
  source /opt/ros/kinetic/setup.bash
  ```
- Create a directory where the code you'll write will be stored (the name `ros_ws` and the location ~ can be changed):
  ```
  mkdir -p ~/ros_ws/src
  ```
- Initialize the workspace:
  ```
  cd ~/ros_ws && catkin_make
  ```
- Update your bash startup script and make sure it worked:
  ```
  echo -e "\n# ROS\nsource $HOME/ros_ws/devel/setup.bash\n" >> ~/.bashrc && tail ~/.bashrc && source ~/.bashrc
  ```

Task 4: Git and GitLab

- Log into university GitLab with your LDAP / TZI account:

  `https://gitlab.informatik.uni-bremen.de/`

Task 4: Git and GitLab

- Log into university GitLab with your LDAP / TZI account:
  `https://gitlab.informatik.uni-bremen.de/`
- Click on "+ New Project", call the project `lisp_course_exercises` and make sure it is private.

Task 4: Git and GitLab

- Log into university GitLab with your LDAP / TZI account:
  `https://gitlab.informatik.uni-bremen.de/`
- Click on "+ New Project", call the project `lisp_course_exercises` and make sure it is private.
- Once created, in "Members" tab add "Arthur Niedzwiecki" and "G. Kazhoyan" as collaborators. "Project Access" should be master.

Task 4: Git and GitLab

- Log into university GitLab with your LDAP / TZI account:

  `https://gitlab.informatik.uni-bremen.de/`
- Click on "+ New Project", call the project `lisp_course_exercises` and make sure it is private.
- Once created, in "Members" tab add "Arthur Niedzwiecki" and "G. Kazhoyan" as collaborators. "Project Access" should be master.
- Install Git:

  ```
  sudo apt-get install git
  ```

Task 4: Git and GitLab

- Log into university GitLab with your LDAP / TZI account:

  `https://gitlab.informatik.uni-bremen.de/`
- Click on "+ New Project", call the project `lisp_course_exercises` and make sure it is private.
- Once created, in "Members" tab add "Arthur Niedzwiecki" and "G. Kazhoyan" as collaborators. "Project Access" should be master.
- Install Git:

  ```
  sudo apt-get install git
  ```
- Download the course material into your ROS workspace:

  ```
  roscd && cd ../src
  git clone https://gitlab.informatik.uni-bremen.de/lisp-course/lisp_course_exercises.git && ll
  ```

Task 4: Git and GitLab

- Log into university GitLab with your LDAP / TZI account:

  `https://gitlab.informatik.uni-bremen.de/`
- Click on "+ New Project", call the project `lisp_course_exercises` and make sure it is private.
- Once created, in "Members" tab add "Arthur Niedzwiecki" and "G. Kazhoyan" as collaborators. "Project Access" should be master.
- Install Git:

  ```
  sudo apt-get install git
  ```
- Download the course material into your ROS workspace:

  ```
  roscd && cd ../src
  git clone https://gitlab.informatik.uni-bremen.de/lisp-course/lisp_course_exercises.git && ll
  ```
- Define a remote target with the address of your new GitLab repo:

  ```
  cd lisp_course_exercises
  git remote add my-repo https://gitlab.informatik.uni-bremen.de/YOUR_GITLAB_USERNAME/lisp_course_exercises.git
  ```

Task 4: Git and GitLab

- Log into university GitLab with your LDAP / TZI account:

  `https://gitlab.informatik.uni-bremen.de/`
- Click on "+ New Project", call the project `lisp_course_exercises` and make sure it is private.
- Once created, in "Members" tab add "Arthur Niedzwiecki" and "G. Kazhoyan" as collaborators. "Project Access" should be master.
- Install Git:

  ```
  sudo apt-get install git
  ```
- Download the course material into your ROS workspace:

  ```
  roscd && cd ../src
  git clone https://gitlab.informatik.uni-bremen.de/lisp-course/lisp_course_exercises.git && ll
  ```
- Define a remote target with the address of your new GitLab repo:

  ```
  cd lisp_course_exercises
  git remote add my-repo https://gitlab.informatik.uni-bremen.de/YOUR_GITLAB_USERNAME/lisp_course_exercises.git
  ```
- Upload the files to your new GitLab repo:

  ```
  git push -u my-repo master
  ```

Task 4 (alternative): Git and GitHub

- Create an account on GitHub and get a student discount:

  `https://education.github.com/`

Task 4 (alternative): Git and GitHub

- Create an account on GitHub and get a student discount:
  `https://education.github.com/`
- Click on "Start a project", call the project `lisp_course_exercises`. Once you get student discount, make the project private.

Task 4 (alternative): Git and GitHub

- Create an account on GitHub and get a student discount:
  `https://education.github.com/`
- Click on "Start a project", call the project `lisp_course_exercises`. Once you get student discount, make the project private.
- In project "Settings" → "Collaborators" add "Arthur Niedzwiecki" and "Gayane Kazhoyan" as collaborators.

Task 4 (alternative): Git and GitHub

- Create an account on GitHub and get a student discount:

  `https://education.github.com/`
- Click on "Start a project", call the project `lisp_course_exercises`. Once you get student discount, make the project private.
- In project "Settings" → "Collaborators" add "Arthur Niedzwiecki" and "Gayane Kazhoyan" as collaborators.
- Install Git:

  ```
  sudo apt-get install git
  ```

Task 4 (alternative): Git and GitHub

- Create an account on GitHub and get a student discount:

  `https://education.github.com/`
- Click on "Start a project", call the project `lisp_course_exercises`. Once you get student discount, make the project private.
- In project "Settings" → "Collaborators" add "Arthur Niedzwiecki" and "Gayane Kazhoyan" as collaborators.
- Install Git:

  ```
  sudo apt-get install git
  ```
- Download the course material into your ROS workspace:

  ```
  roscd && cd ../src
  ```

  ```
  git clone https://github.com/lisp-course/lisp_course_exercises.git && ll
  ```

Task 4 (alternative): Git and GitHub

- Create an account on GitHub and get a student discount:

  `https://education.github.com/`
- Click on "Start a project", call the project `lisp_course_exercises`. Once you get student discount, make the project private.
- In project "Settings" → "Collaborators" add "Arthur Niedzwiecki" and "Gayane Kazhoyan" as collaborators.
- Install Git:

  ```
  sudo apt-get install git
  ```
- Download the course material into your ROS workspace:

  ```
  roscd && cd ../src
  git clone https://github.com/lisp-course/lisp_course_exercises.git && ll
  ```
- Define a remote target with the address of your new GitHub repo:

  ```
  cd lisp_course_exercises
  git remote add my-repo https://github.com/YOUR_GITHUB_USERNAME/lisp_course_exercises.git
  ```

Task 4 (alternative): Git and GitHub

- Create an account on GitHub and get a student discount:

  ```
  https://education.github.com/
  ```

- Click on "Start a project", call the project `lisp_course_exercises`. Once you get student discount, make the project private.
- In project "Settings" → "Collaborators" add "Arthur Niedzwiecki" and "Gayane Kazhoyan" as collaborators.
- Install Git:

  ```
  sudo apt-get install git
  ```

- Download the course material into your ROS workspace:

  ```
  roscd && cd ../src
  ```

  ```
  git clone https://github.com/lisp-course/lisp_course_exercises.git && ll
  ```

- Define a remote target with the address of your new GitHub repo:

  ```
  cd lisp_course_exercises
  ```

  ```
  git remote add my-repo https://github.com/YOUR_GITHUB_USERNAME/lisp_course_exercises.git
  ```

- Upload the files to your new GitHub repo:

  ```
  git push -u my-repo master
  ```

Task 5: Install the IDE

- Install the editor itself (Emacs), the Common Lisp compiler (SBCL), the linker (ASDF) and the Emacs Common Lisp plugin (Slime):

```
sudo apt-get install ros-kinetic-roslisp-repl
```

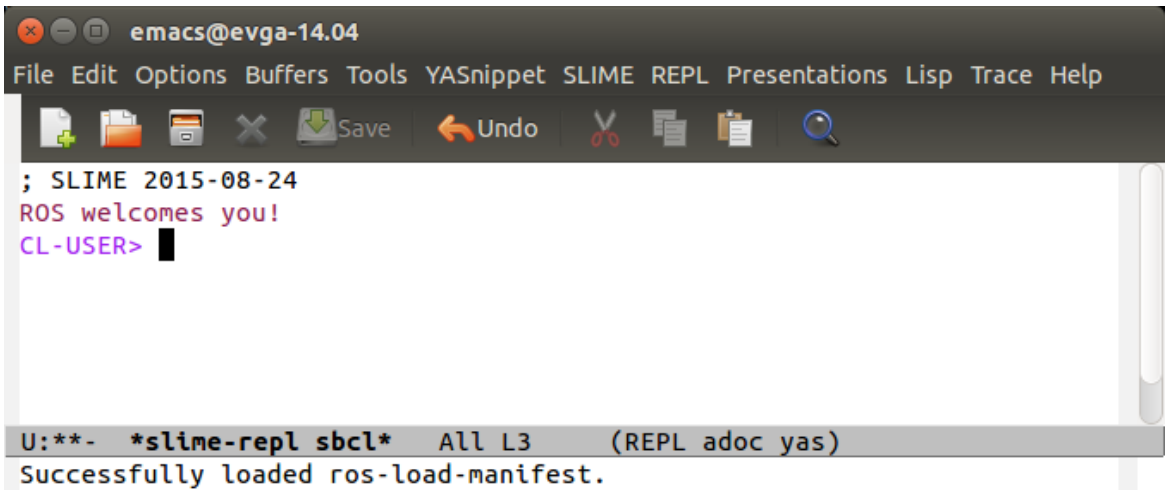
Task 5: Install the IDE

- Install the editor itself (Emacs), the Common Lisp compiler (SBCL), the linker (ASDF) and the Emacs Common Lisp plugin (Slime):

```
sudo apt-get install ros-kinetic-roslisp-repl
```

- Start the editor (after compilation is finished you'll see the Lisp shell):

```
roslisp_repl &
```



Task 6: Get familiar with Emacs

The following notation is used in Emacs for keyboard shortcuts:

- `C` for `<Ctrl>`
- `M` for `<Alt>`
- `SPC` for `<Space>`
- `RET` for `<Enter>`
- `-` for when two keys are pressed together (e.g. `C-x` for `<Ctrl>+x`)

The basic shortcuts you will need are listed below:

- `C-x C-f` opens a file
- `C-x 3` or `C-x 2` opens a new tab, `C-x 0` closes it, `C-x 1` maximizes
- `C-x o` switches between tabs
- `C-x b` switches buffers, `C-x C-b` lists all open buffers, `C-x k` kills
- `C-g` cancels a command half-way, `C-x C-c yes` exits Emacs

Task 6: Get familiar with Emacs

The following notation is used in Emacs for keyboard shortcuts:

- `C` for `<Ctrl>`
- `M` for `<Alt>`
- `SPC` for `<Space>`
- `RET` for `<Enter>`
- `-` for when two keys are pressed together (e.g. `C-x` for `<Ctrl>+x`)

The basic shortcuts you will need are listed below:

- `C-x C-f` opens a file
- `C-x 3` or `C-x 2` opens a new tab, `C-x 0` closes it, `C-x 1` maximizes
- `C-x o` switches between tabs
- `C-x b` switches buffers, `C-x C-b` lists all open buffers, `C-x k` kills
- `C-g` cancels a command half-way, `C-x C-c yes` exits Emacs

Open the file with your first assignment and follow the instructions:

```
ROS_WORKSPACE/src/lisp_course_exercises/assignment_1/src/orc-battle.lisp
```

Task 7: Get familiar with Git

- Once done editing `orc-battle.lisp`, check what's new in your local repo (the one on your hard drive):

```
cd ROS_WORKSPACE/src/lisp_course_exercises && git status
```

Task 7: Get familiar with Git

- Once done editing `orc-battle.lisp`, check what's new in your local repo (the one on your hard drive):

```
cd ROS_WORKSPACE/src/lisp_course_exercises && git status
```

- To see which exactly lines changed ask for the `diff` (*q* to exit):

```
git diff
```

Task 7: Get familiar with Git

- Once done editing `orc-battle.lisp`, check what's new in your local repo (the one on your hard drive):

```
cd ROS_WORKSPACE/src/lisp_course_exercises && git status
```

- To see which exactly lines changed ask for the `diff` (*q* to exit):

```
git diff
```

- The red files are the new untracked ones, the green ones are already in the Git index. To add new files to the index use

```
git add .
```

Task 7: Get familiar with Git

- Once done editing `orc-battle.lisp`, check what's new in your local repo (the one on your hard drive):

```
cd ROS_WORKSPACE/src/lisp_course_exercises && git status
```

- To see which exactly lines changed ask for the `diff` (*q* to exit):

```
git diff
```

- The red files are the new untracked ones, the green ones are already in the Git index. To add new files to the index use

```
git add .
```

- If you deleted some files, to remove them from the index use

```
git add -u
```

Task 7: Get familiar with Git

- Once done editing `orc-battle.lisp`, check what's new in your local repo (the one on your hard drive):

```
cd ROS_WORKSPACE/src/lisp_course_exercises && git status
```

- To see which exactly lines changed ask for the `diff` (*q* to exit):

```
git diff
```

- The red files are the new untracked ones, the green ones are already in the Git index. To add new files to the index use

```
git add .
```

- If you deleted some files, to remove them from the index use

```
git add -u
```

- Once you're sure the changes are final, commit locally:

```
git commit -m "A meaningful commit message."
```

Task 7: Get familiar with Git

- Once done editing `orc-battle.lisp`, check what's new in your local repo (the one on your hard drive):

```
cd ROS_WORKSPACE/src/lisp_course_exercises && git status
```

- To see which exactly lines changed ask for the `diff` (*q* to exit):

```
git diff
```

- The red files are the new untracked ones, the green ones are already in the Git index. To add new files to the index use

```
git add .
```

- If you deleted some files, to remove them from the index use

```
git add -u
```

- Once you're sure the changes are final, commit locally:

```
git commit -m "A meaningful commit message."
```

- Finally, to upload your local commits to the GitLab server, push the changes upstream:

```
git push # or git push my-repo master
```

If you decided to go for Ubuntu 14.04

- Download the latest version of the Lisp compiler:

  `https://sourceforge.net/projects/sbcl/files/sbcl/1.3.1/`

If you decided to go for Ubuntu 14.04

- Download the latest version of the Lisp compiler:
  `https://sourceforge.net/projects/sbcl/files/sbcl/1.3.1/`
- You will most likely need the x86-64 version (NOT arm64):
  `https://sourceforge.net/projects/sbcl/files/sbcl/1.3.1/sbcl-1.3.1-x86-64-linux-binary.tar.bz2/download`

If you decided to go for Ubuntu 14.04

- Download the latest version of the Lisp compiler:
  `https://sourceforge.net/projects/sbcl/files/sbcl/1.3.1/`
- You will most likely need the x86-64 version (NOT arm64):
  `https://sourceforge.net/projects/sbcl/files/sbcl/1.3.1/sbcl-1.3.1-x86-64-linux-binary.tar.bz2/download`
- Extract the archive you just downloaded.
  In Nautilus, the file browser of Ubuntu, it will be "right click → Extract Here".

If you decided to go for Ubuntu 14.04

- Download the latest version of the Lisp compiler:
  `https://sourceforge.net/projects/sbcl/files/sbcl/1.3.1/`
- You will most likely need the x86-64 version (NOT arm64):
  `https://sourceforge.net/projects/sbcl/files/sbcl/1.3.1/sbcl-1.3.1-x86-64-linux-binary.tar.bz2/download`
- Extract the archive you just downloaded.
  In Nautilus, the file browser of Ubuntu, it will be "right click → Extract Here".
- In terminal go to the place you just extracted files to, e.g.:
  `cd /Downloads/sbcl-1.3.1-x86-64-linux`

If you decided to go for Ubuntu 14.04

- Download the latest version of the Lisp compiler:
  `https://sourceforge.net/projects/sbcl/files/sbcl/1.3.1/`
- You will most likely need the x86-64 version (NOT arm64):
  `https://sourceforge.net/projects/sbcl/files/sbcl/1.3.1/sbcl-1.3.1-x86-64-linux-binary.tar.bz2/download`
- Extract the archive you just downloaded.
  In Nautilus, the file browser of Ubuntu, it will be "right click → Extract Here".
- In terminal go to the place you just extracted files to, e.g.:
  `cd /Downloads/sbcl-1.3.1-x86-64-linux`
- Install the compiler:
  `sh install.sh`

If you decided to go for Ubuntu 18.04

- The name of the ROS distribution will be not “kinetic” but “melodic”. Follow the official instructions on ROS webpage for installing ROS melodic.

If you decided to go for Ubuntu 18.04

- The name of the ROS distribution will be not “kinetic” but “melodic”. Follow the official instructions on ROS webpage for installing ROS melodic.
- The current version of “roslisp-repl” might not work for you. So you will either have to wait a week or two for the update that fixes the problems, or install the REPL from source. Write an email to Gaya if you decide to do that.

Q & A

Thanks for your attention!