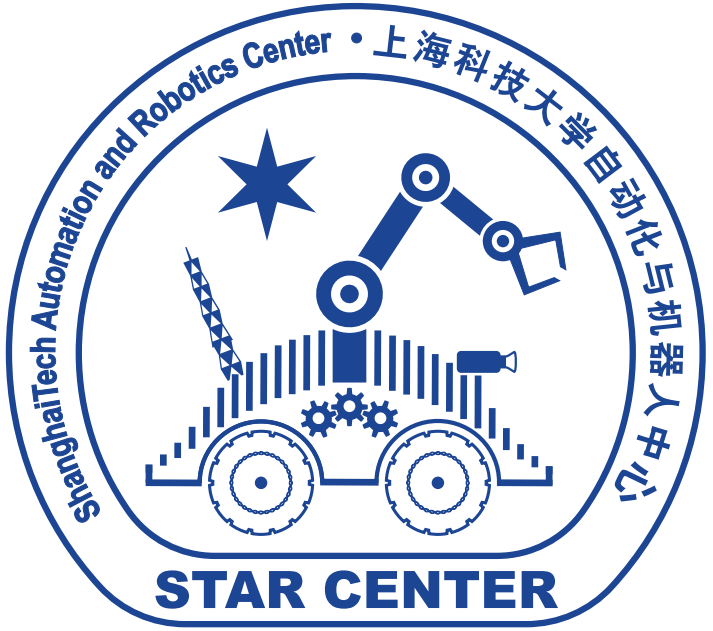
上海科技大学

ShanghaiTech University

CS283: **Robotics Fall 2020**

Sören Schwertfeger & Qingwen Xu

ShanghaiTech University



Teaching Team

Sören Schwertfeger

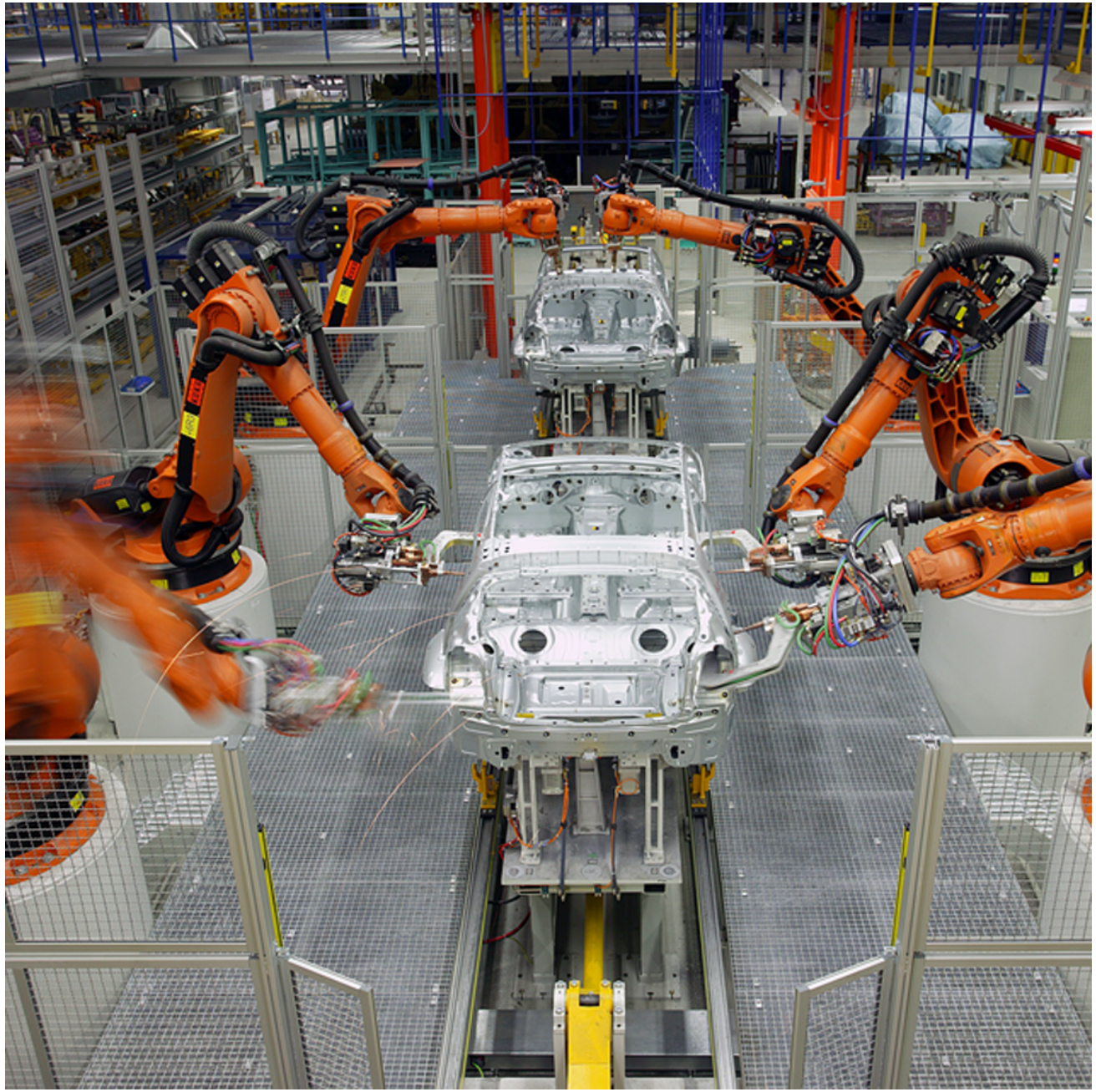
Qingwen Xu

Outline

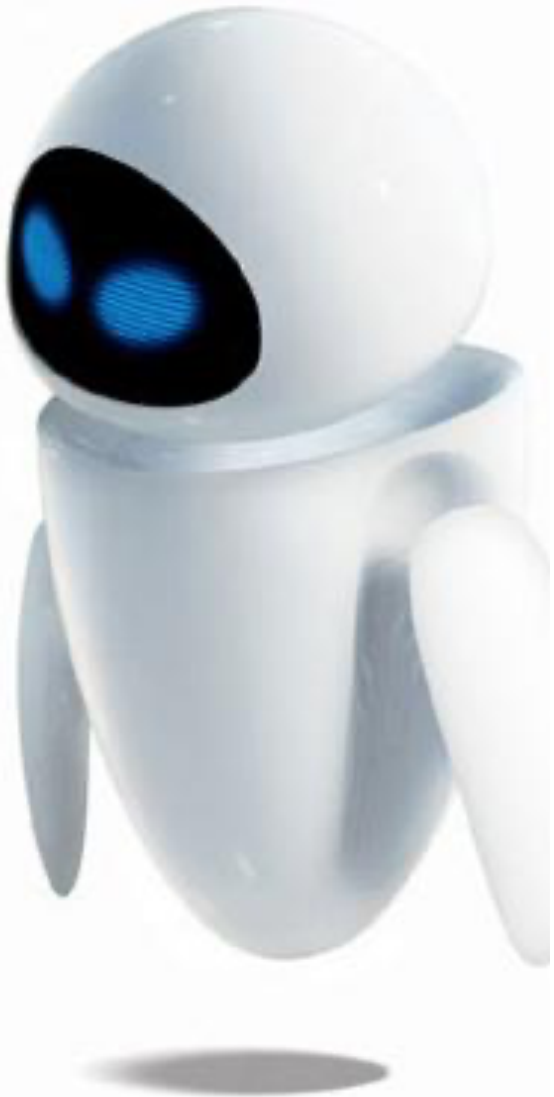
- What is a Robot?
- Why Mobile Robotics?
- Why Autonomous Mobile Robotics?
- Course Overview
- Brief History
- Software

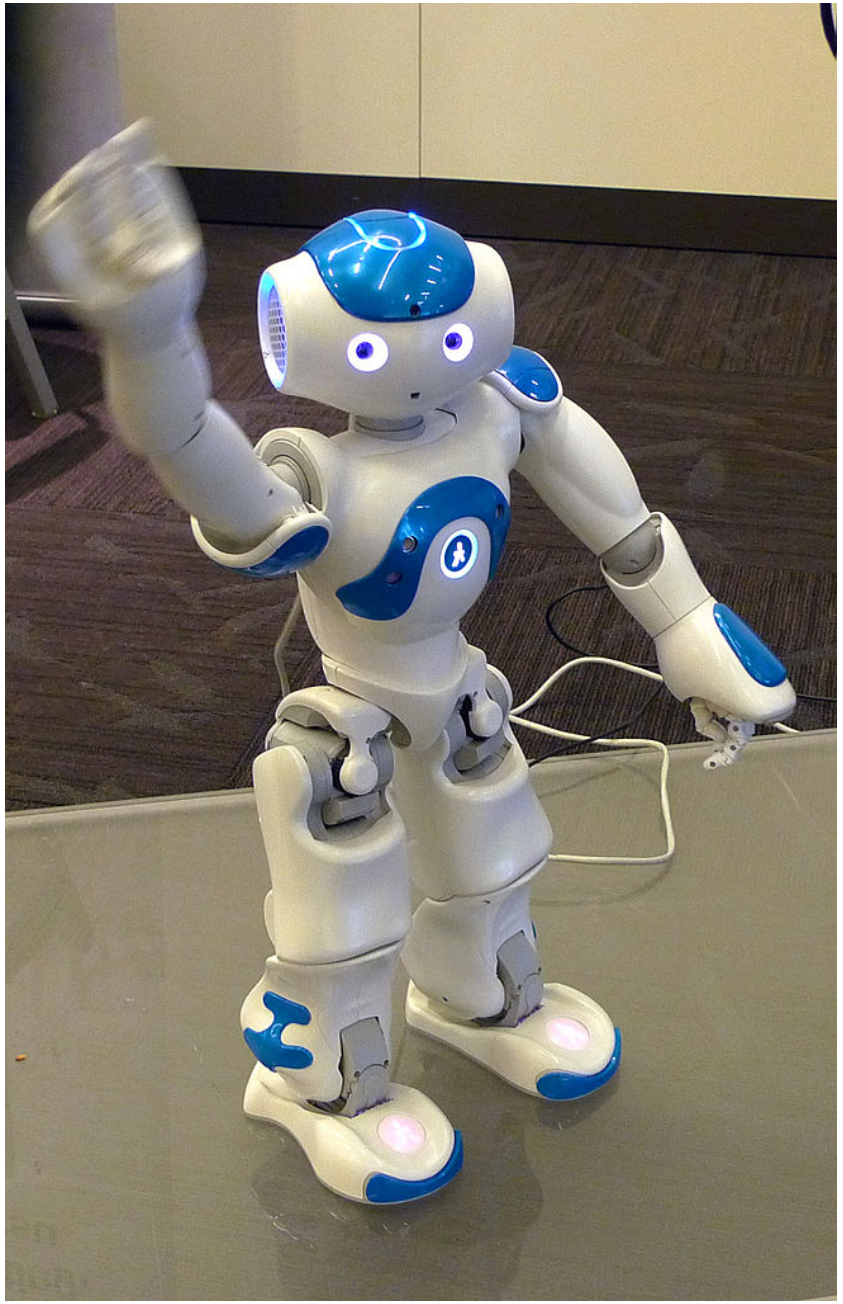
What is a Robot?

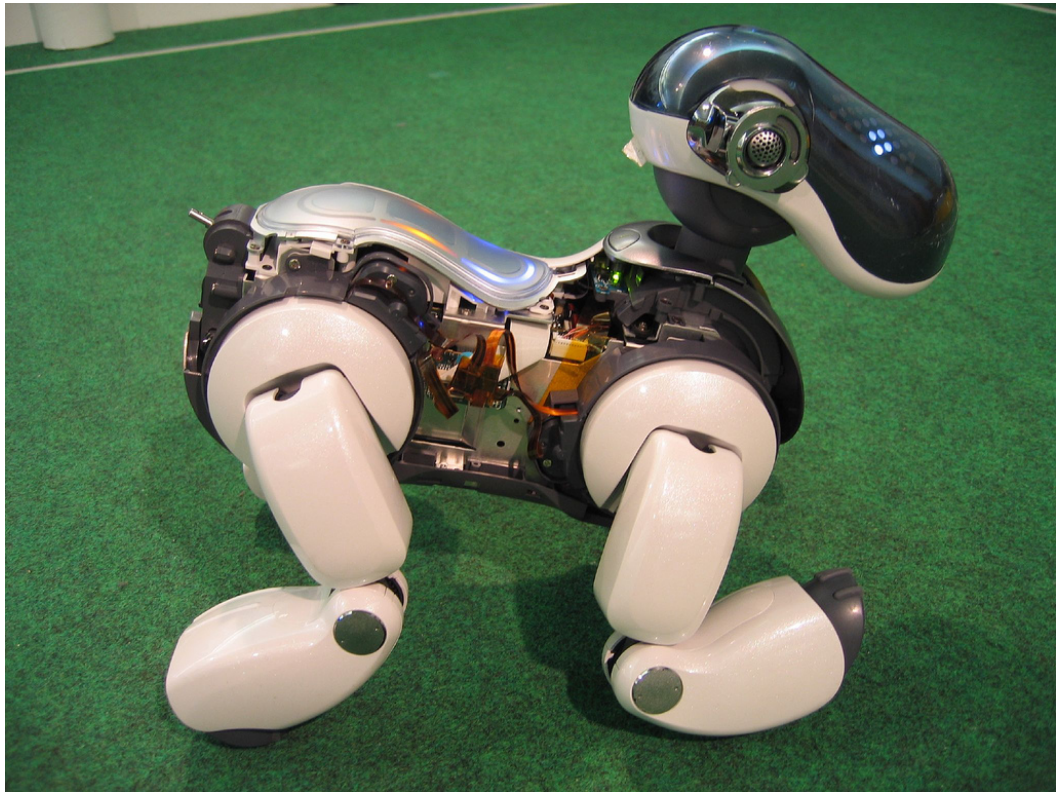
Pictures on the following slides all from http://commons.wikimedia.org




ASIMO
HONDA



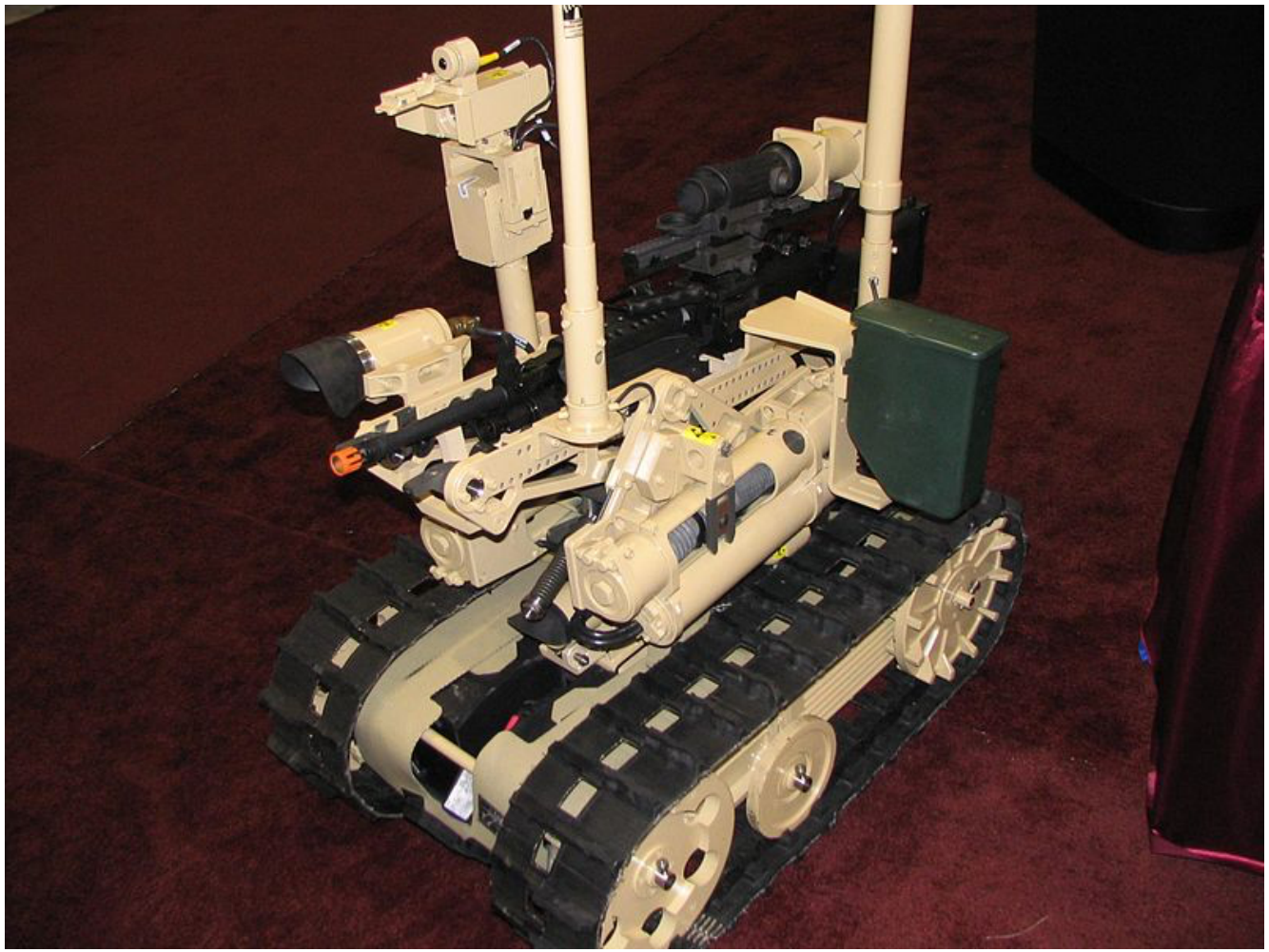




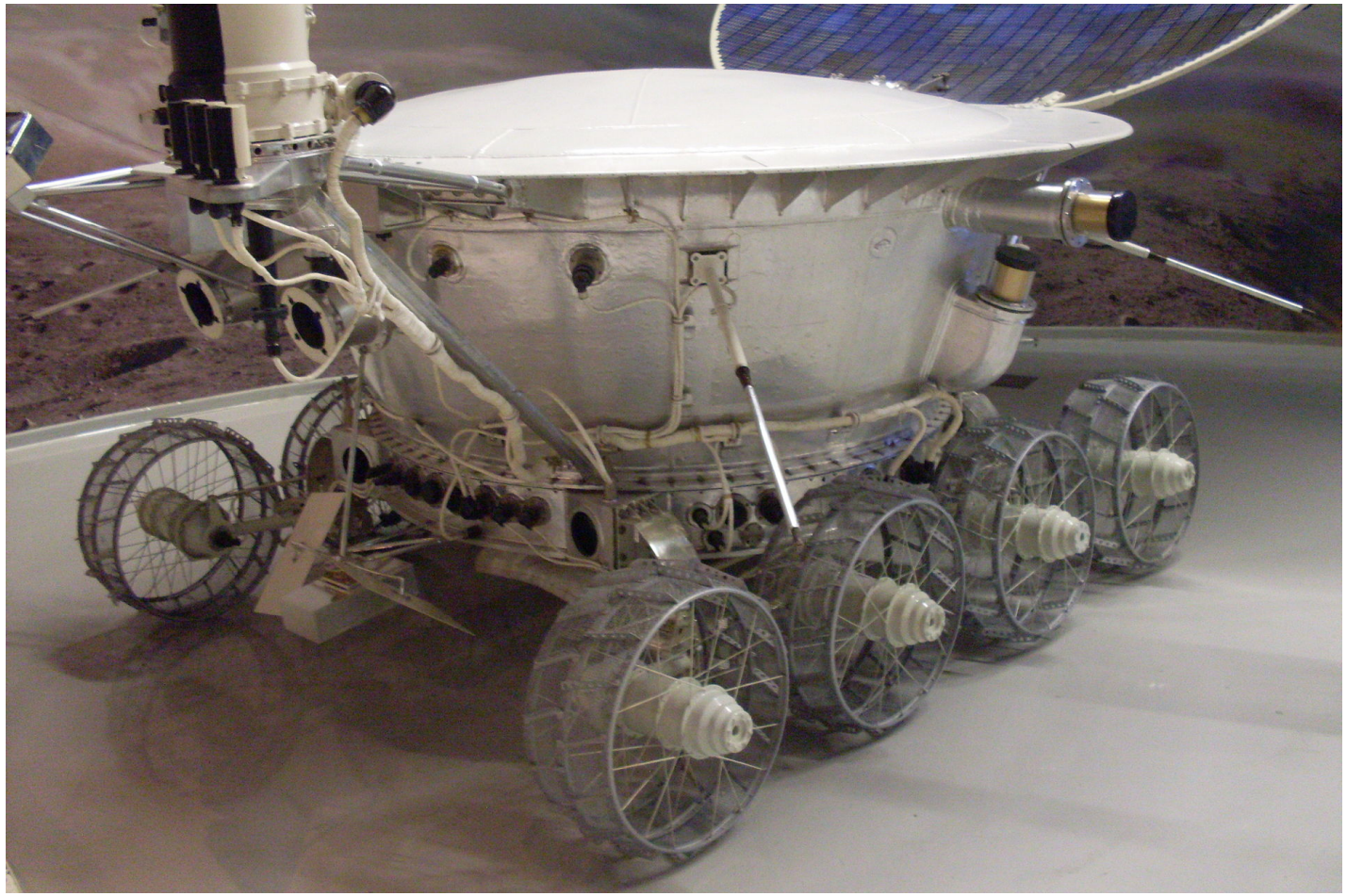

KEEP CLEAR

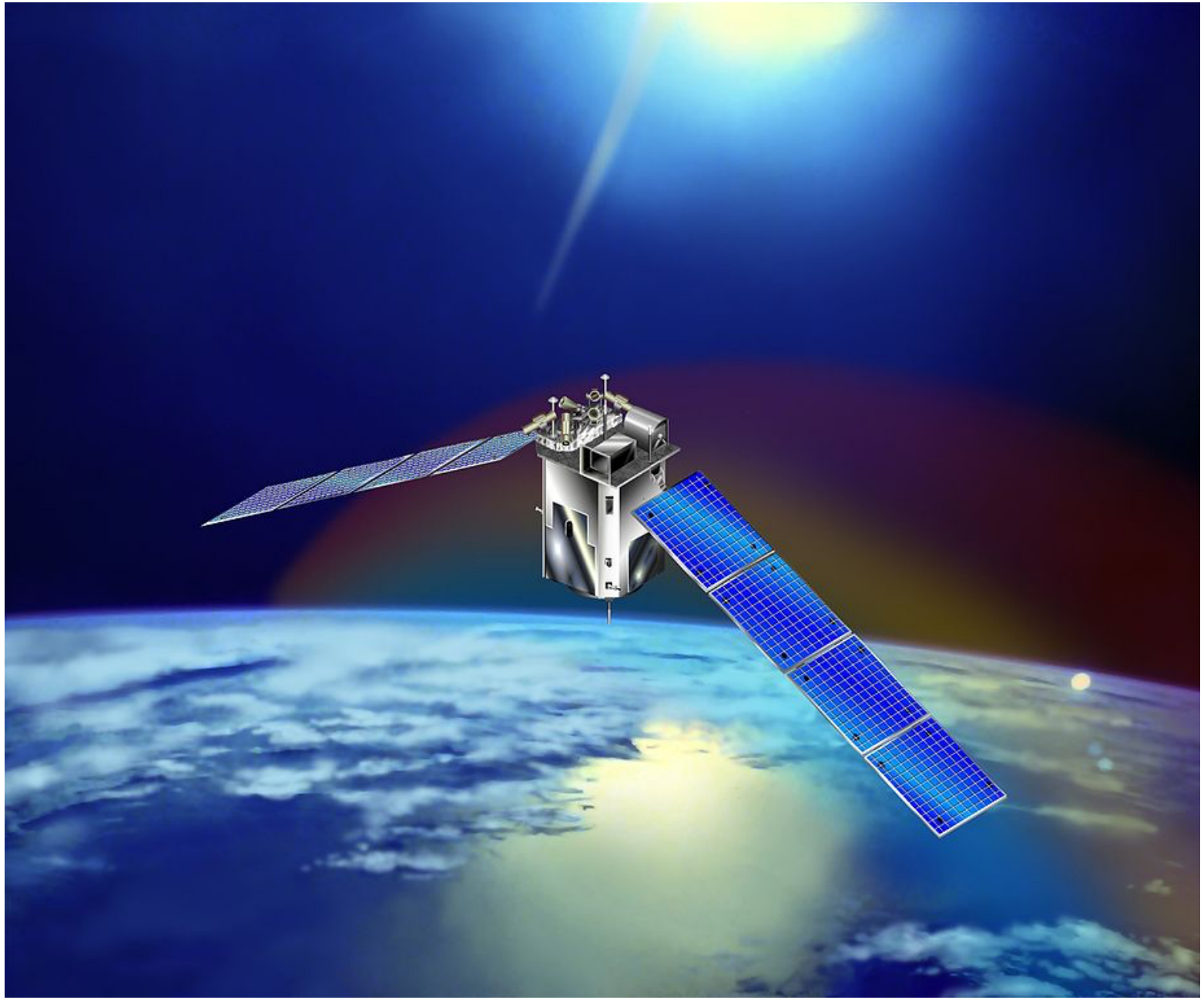

iRobot
Roomba
Robotic FloorVac
S
M
L
Select Room Size to Start / Stop
Always empty both sides of particle bin after each use

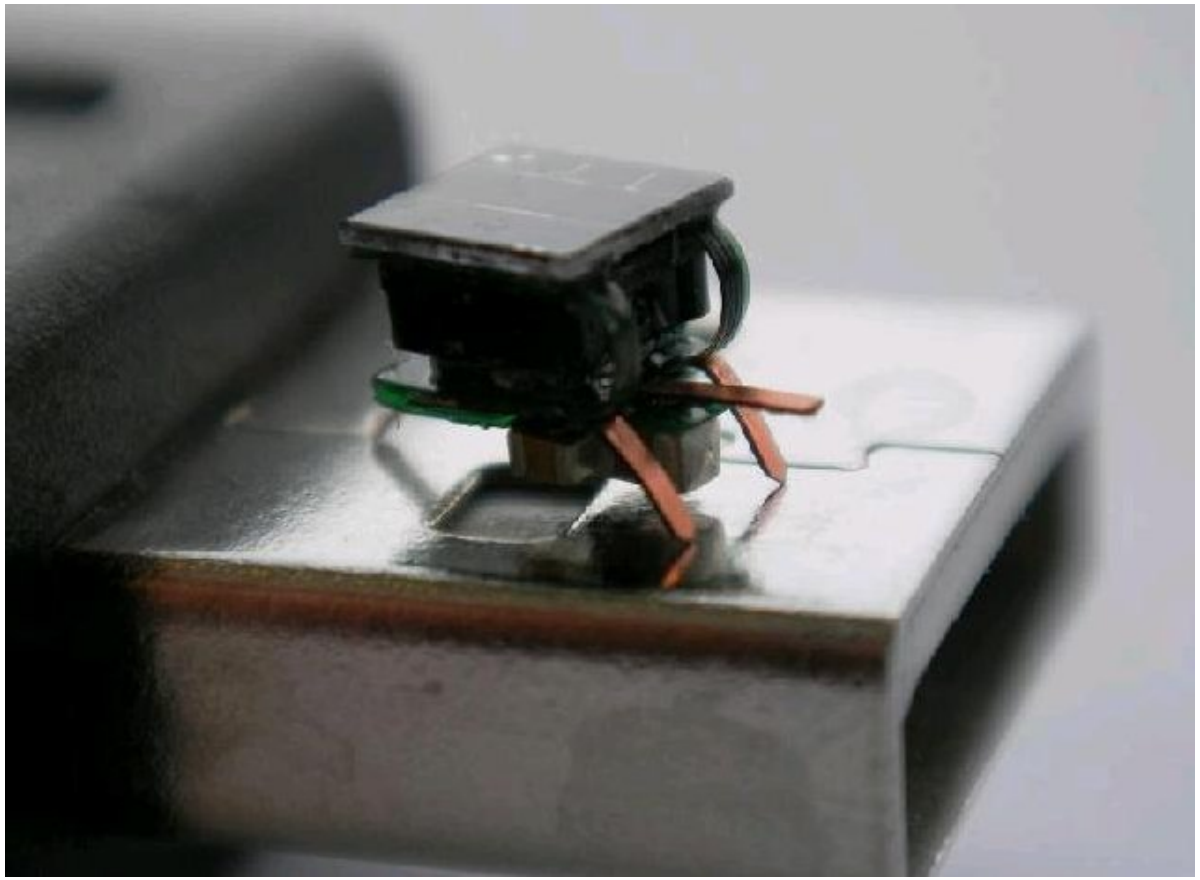

Robotics Technology
Development and Commercialization
Robots that Work in the Real World
Unmanned Vehicle
Systems Engineering
Engineering the Future of Unmanned Vehicles
Autonomous
Navigation Systems
NREC

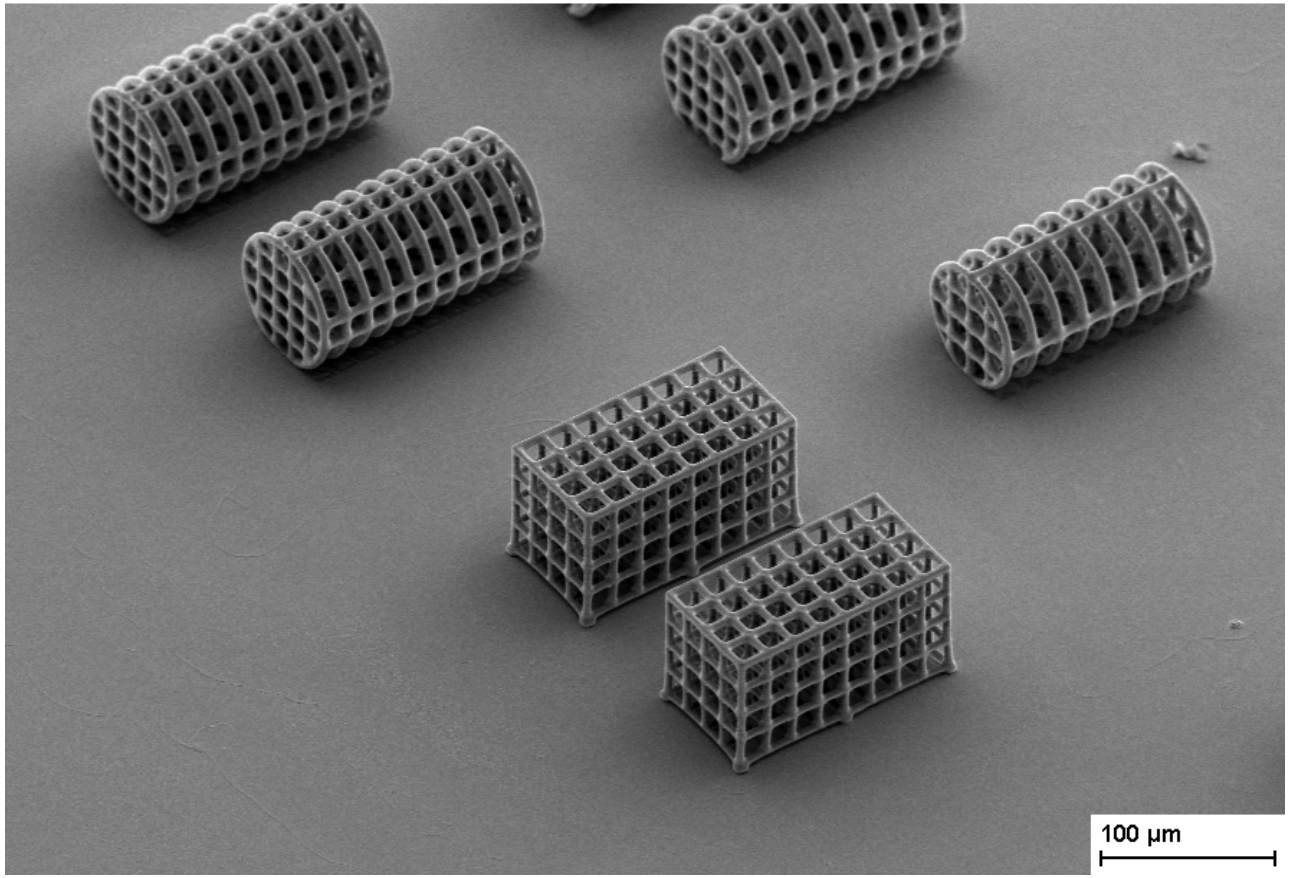



U.S. AIR FORCE
GLOBAL HAWK
HIGH ALTITUDE LONG ENDURANCE
UNMANNED AERIAL VEHICLE SYSTEM
Full Scale Mock-up
NORTHROP GRUMMAN



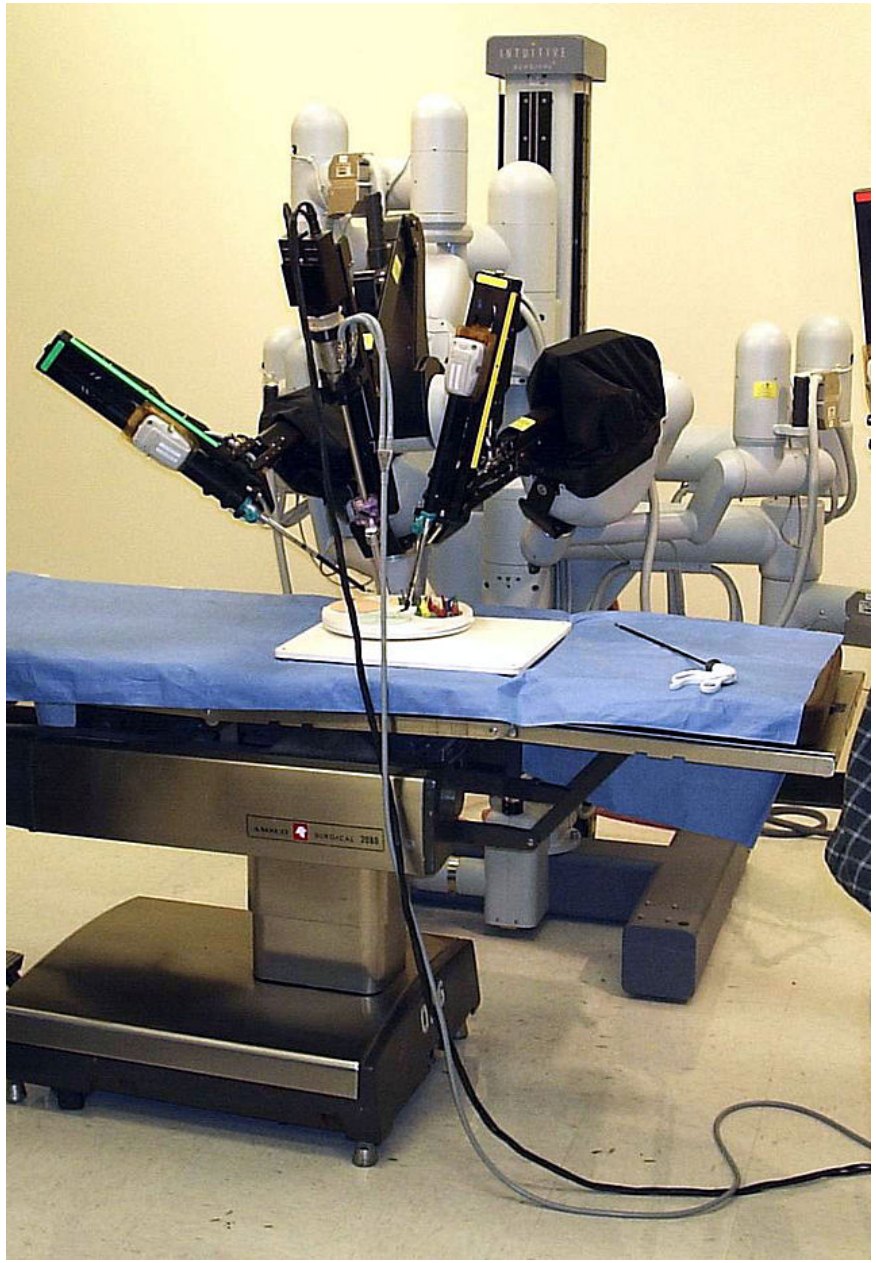





100 µm


EXPLOSIVE


JASON
WOODS HOLE
OCEANOGRAPHIC INSTITUTION



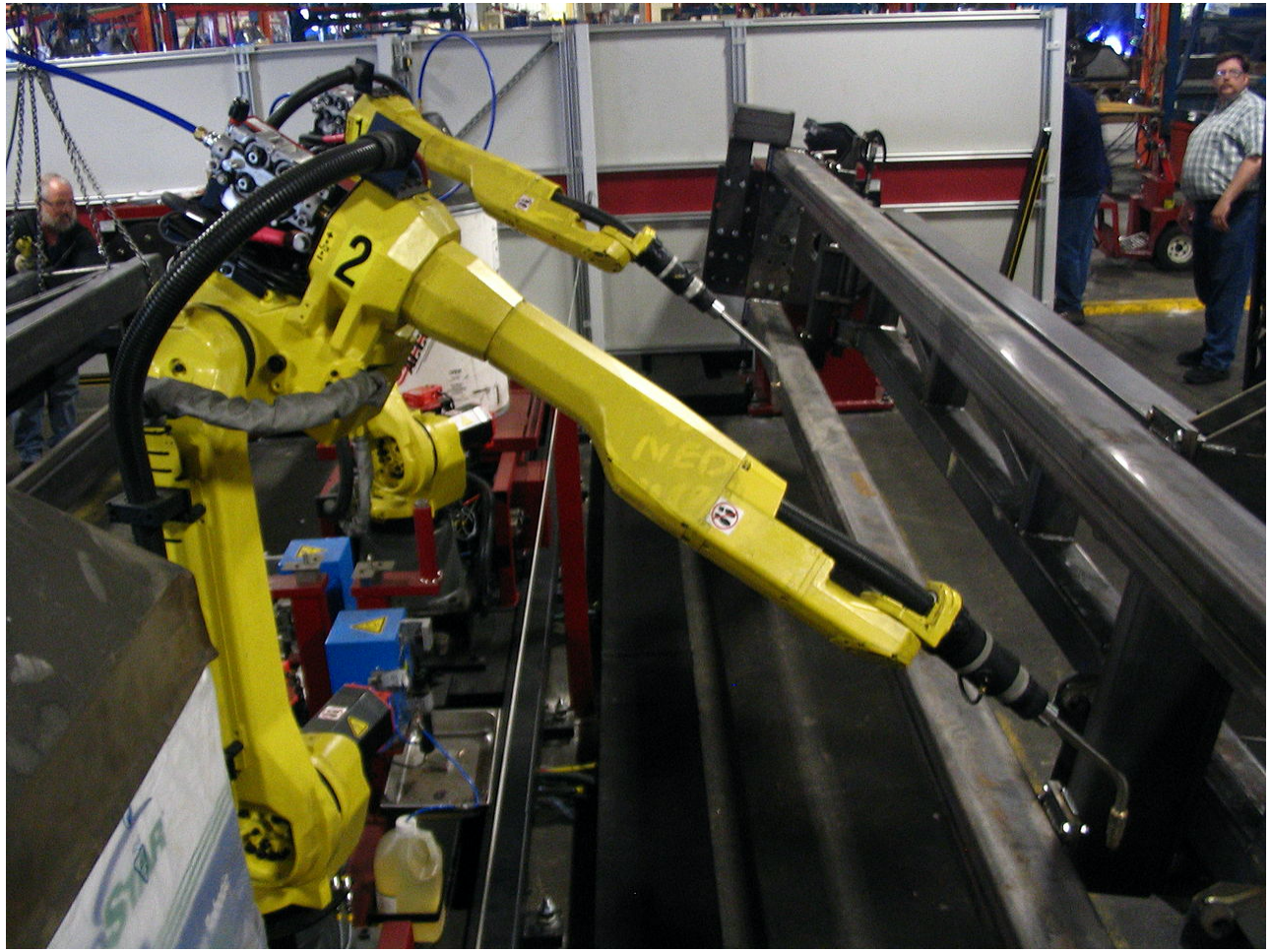

oogle
intel
Powered by Diesel
MDV
MOHR DAVIDOW VENTURES
RedBull
NP
ibeo
CATERPILLAR
TORC


Daied
3
Daied
Daied
Daied

Definition: A Robot is ...

A machine
capable of performing complex tasks
in the physical world,
that is using sensors to perceive the environment
and acts tele-operated or autonomous.

Industry vs Mobile Robots



- Industrial Robots rule:
  - 2015: 254,000 industrial robots sold
  - Over 1.6 million industrial robots installed – rising by about 9% per year
  - China biggest robot market regarding annual sales - also fasted growing market worldwide
- Industrial Robots stay at one place!
- Almost all other robots move => **Mobile Robotics**

Why **Autonomous** Mobile Robotics?

- Tele-operating robots: boring and inefficient
- Autonomous robots: Robots that act by their own reasoning
  - Human operator might be present: Gives high level tasks
- Why autonomy?
  - Autonomous behaviors might be **better** than remote control by humans
  - Remote control might be **boring** or **stressful** and **tiresome**
  - Human operators might be a **scarce** resource or **expensive**
  - Multi robot approaches: One operator for many robots
- Semi-autonomy:
  - Autonomous behaviors that help the operator, for example:
  - Way-point navigation, autonomous stair climbing, assisted manipulation
  - Gradual development from tele-operation to full autonomy possible

- Autonomous mobile robots move around in the environment. Therefore **ALL** of them:
  - They need to know **where** they **are**.
  - They need to know **where** their **goal** is.
  - They need to know **how** to get there.

- Autonomous mobile robots move around in the environment. Therefore **ALL** of them:
  - They need to know **where** they **are.**
  - They need to know **where** their **goal** is.
  - They need to know **how** to get there.

- Where am I?
  - Global Positioning System: outdoor, error measured in meters
  - Guiding system: (painted lines, inductive guides), markers, iBeacon
  - Model of the environment:
    - Map, Localize yourself in this model
    - Mapping: Build the map while driving

- Autonomous mobile robots move around in the environment. Therefore **ALL** of them:
  - They need to know **where** they **are**.
  - They need to know **where** their **goal** is.
  - They need to know **how** to get there.

- Where is my goal?
- Two part problem:
  - What is the goal?
    - Expressed using the world model (map)
    - Using object recognition
    - No specific goal (random)
  - Where is that goal?
    - Coordinates in the map
    - Localization step at the end of the object recognition process
    - User input

- Autonomous mobile robots move around in the environment. Therefore **ALL** of them:
  - They need to know **where** they **are**.
  - They need to know **where** their **goal** is.
  - They need to know **how** to get there.
- Different levels:
  - Control:
    - How much power to the motors to move in that direction, reach desired speed
  - Navigation:
    - Avoid obstacles
    - Classify the terrain in front of you
    - Follow a path
  - Planning:
    - Long distance path planning
    - What is the way, optimize for certain parameters

Most important capability

(for autonomous mobile robots)

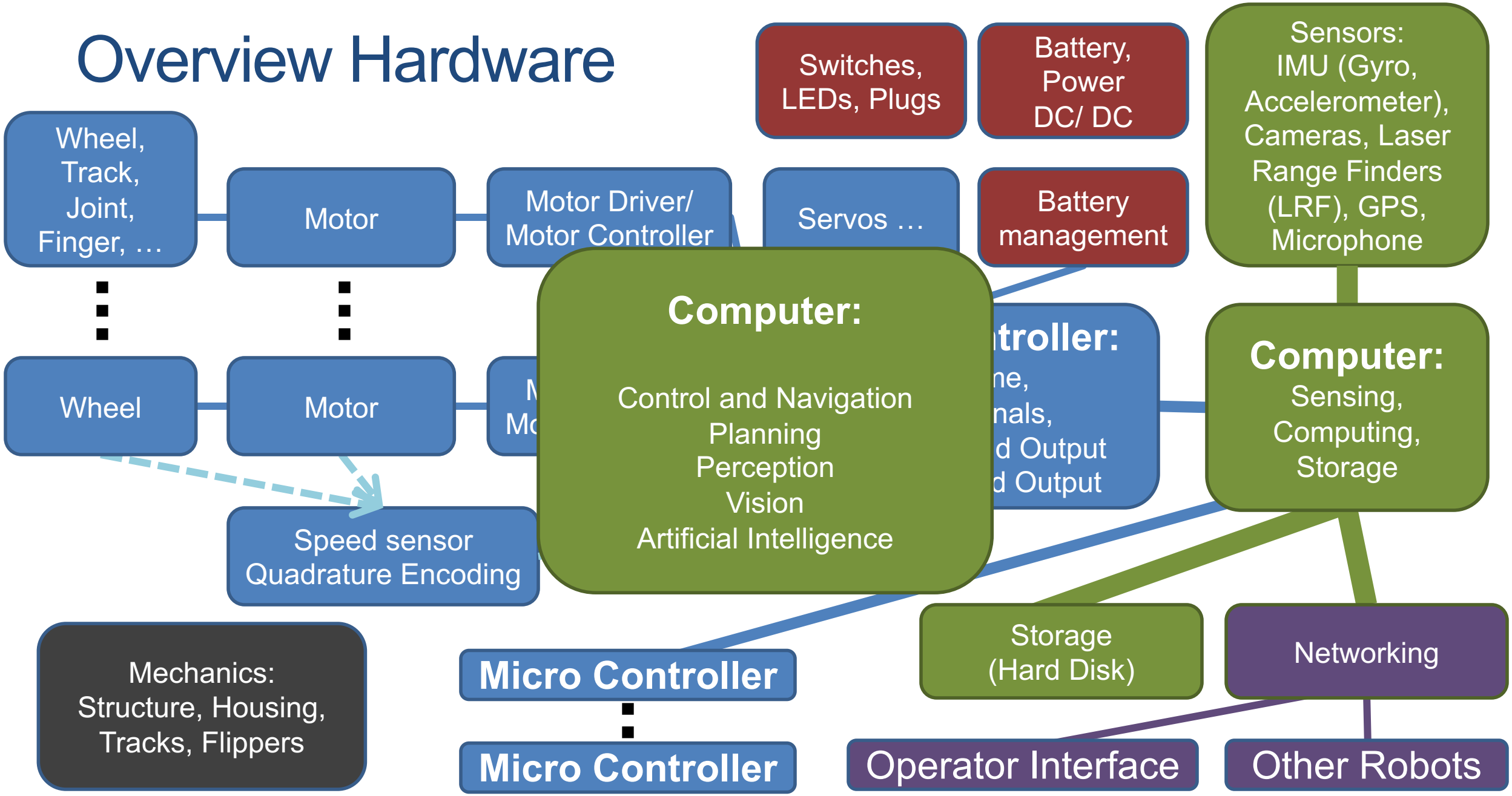
How to get from place A to place B?

(safely and efficiently)

How to get from A to B?

What are the components of a ROBOT?

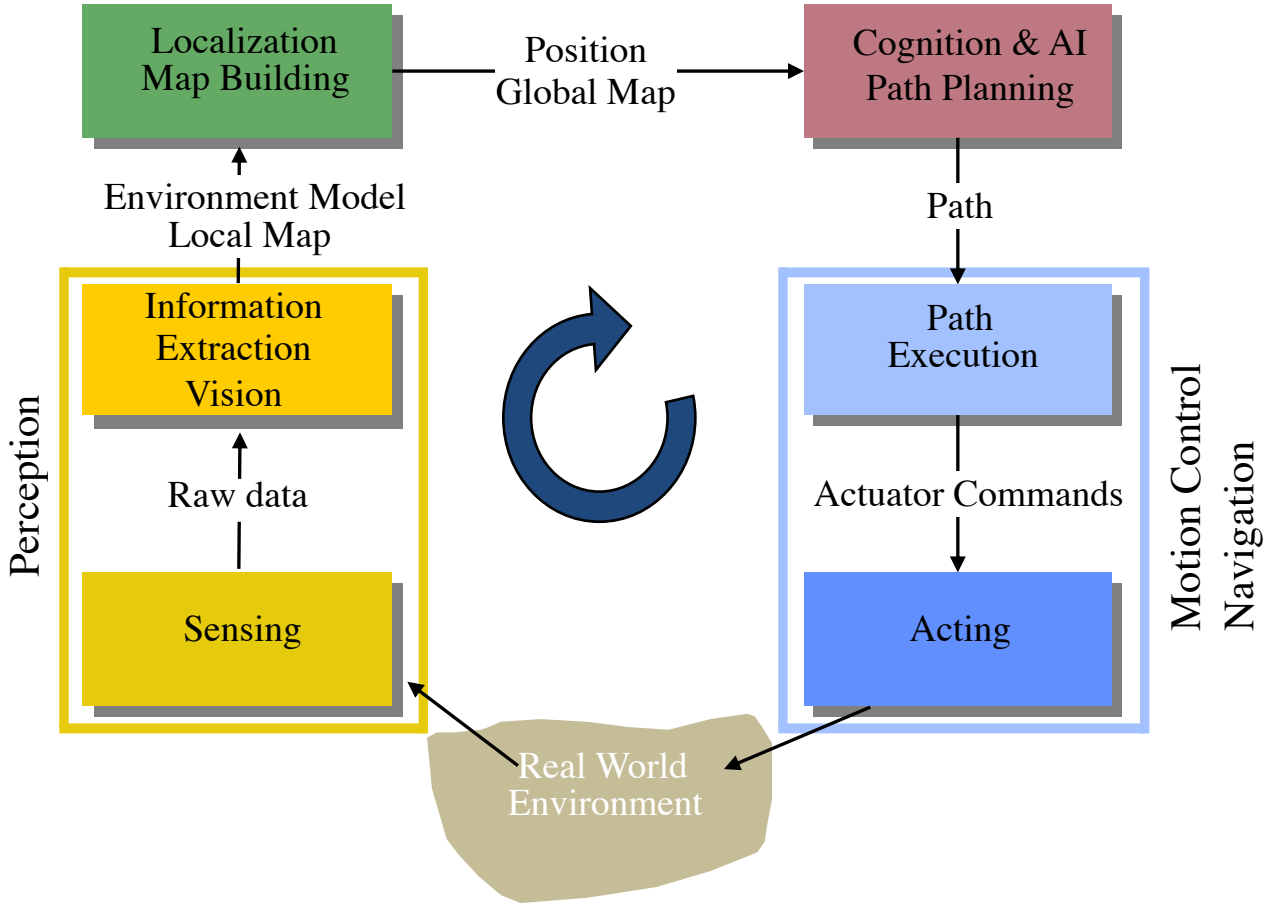
Overview Hardware



How to get from A to B?

How to program an intelligent ROBOT to go from A to B?

General Control Scheme for Mobile Robot Systems



With material from Roland Siegwart and Davide Scaramuzza, ETH Zurich

ADMINISTRIVIA

Teaching Plan

- Lectures
- Homework
  - Including one real Hardware Group Project
- Presentation about robotics paper (related to your project)
- Midterm and Final Exams
- Project…

Mandatory Reading

- Only 2 credit points of lectures => 16 lectures => need to compress the lectures.
- Certain topics that you can just read and learn will be covered only very briefly in the lecture.
- You will be given exact paragraphs to read till next week.
- In the next week there might be a short in-lecture Quiz about the reading material, possibly covering all topics of the course taught so far.
- More complex topics will be covered in more detail in the lecture.

Project

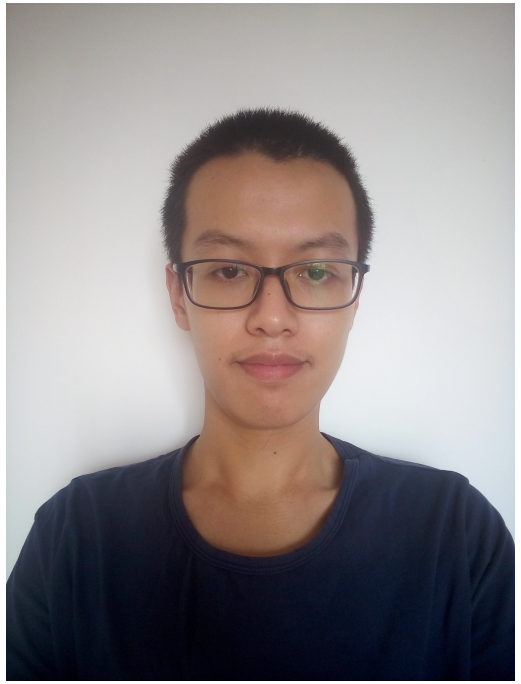
- 2 credit points!
- Work in groups, min 2 students, max 3 students!
- Next lecture: Topics will be proposed…
  - You can also do your own topic, but only after approval of Prof. Schwertfeger
    - Prepare a short, written proposal till next Tuesday!
- Topic selection: Next Thursday!
  - One member writes an email for the whole group to Cui Jiadi: cuijd(at)shanghaitech.edu.cn ; Put the other group members on CC
  - Subject: [Robotics] Group Selection
- One graduate student from my group will co-supervise your project
- Weekly project meetings!
- Oral ”exams” to evaluate the contributions of each member
- No work on project => bad grade of fail

Grading

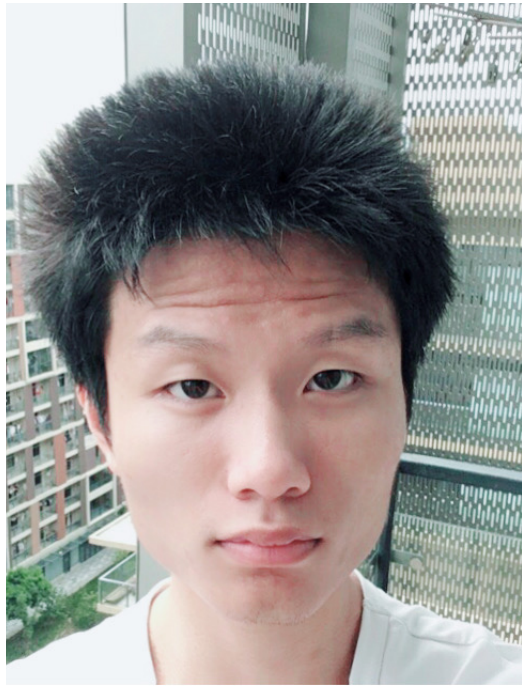
- Grading scheme is not 100% fixed
- Approximately:
  - Lecture: 50%
    - Quizzes during lecture (reading assignments): 4%
    - Homework: 18%
    - Midterm: 8%
    - Final: 20%
  - Project: 50%
    - Paper Presentation: 5%
    - Project Proposal: 5%
    - Intermediate Report: 5%
    - Weekly project meetings: 10%
    - Final Report: 10%
    - Final Demo: 10%
    - Final Webpage: 5%

Getting Help

- Piazza:
  - For discussions and announcements
  - https://piazza.com/shanghaitech.edu.cn/fall2020/cs283
  - Ask questions regarding your reading assignments and homework
  - You are not allowed to give the solutions – just guidance
- Ask questions during the lecture!
- Upon request we can organize a tutorial session
- Only if everything else fails: write e-mails
- Office Hours Prof. Schwertfeger: Tuesday afternoon
- Office Hours TAs: look in piazza



Xiting Zhao 赵希亭



Jiadi Cui 崔佳迪

Policy on Plagiarism

- The homework are individual tasks!
- You may discuss the ideas and algorithms of homework with others but:
  - At no time should you read the source code or possess the source files of any other person, including people outside this course.
  - We will **detect plagiarism** using automated tools and will **prosecute** all violations to the fullest extent of the university regulations, including failing this course, academic probation, and expulsion from the university.

- Homework, project submissions, etc. will be submitted through git – using gitlab. We will create accounts for you on:
  - https://star-center.shanghaitech.edu.cn/gitlab
- We will also use gradescope (exams, HW1, …). We will add you.

Mobile Robotics

- Topic Robots and how to program them:
  - Applications of robotics, software design, locomotion, hardware, sensing, localization, motion planning, autonomy for mobile robots, manipulation

- **Literature:**
- Mobile Robotics Mathematics, Models, and Methods
  - Alonzo Kelly
  - ISBN 978-1-107-03115-9
- Introduction to Autonomous Mobile Robots
  - Roland Siegwart, Illah R. Nourbakhsh, Davide Scaramuzza
  - ISBN: 978-0-262-01535-6

Material

- Webpage
  - https://robotics.shanghaitech.edu.cn/teaching/robotics2020
  - Slides will be available on the webpage
- Piazza
  - https://piazza.com/shanghaitech.edu.cn/fall2020/cs283

- Where to find us:
  Office: SIST 1D 201.A
  Lab: SIST 1D 203
- E-Mail:
  - soerensch@ShanghaiTech.edu.cn

Prerequisite: Robot Operating System 1

- Program in C++ (or python) and ROS 1 (wiki.ros.org)
- Prerequisite for that: Operating System Ubuntu Linux (www.ubuntu.com)
  - **Best option:** Dual boot on your own Laptop/ Computer – needs min. 40 GB from HD
  - Very sub-optimal option: Run Ubuntu in a virtual machine (suggestion: VirtualBox) – needs 40 GB and a modern Laptop (at least 4GB RAM – more is better)
- Preferred version: Ubuntu 18.04 (long term support)
  - ROS Melodic (current version)
- Other tools: git, LaTeX, …

Schedule

- May change – take a look at webpage for most recent version!

	Topic	HW	Project
Week 1	Intro	HW1: ROS	
Week 2	Kinematics/ Sensors	HW2: Pose	
Week 3	Perception	HW3: Mobile Manipulation Sim	
Week 4	Localization		
~~Week 5~~			
Week 6	ICP / Project	HW4: Particle Filter	Proposal Due
Week 7	Presentations		
Week 8	SLAM / Project	HW5: Kalman	
Week 9	Navigation / Project		
Week 10	Planning / Project	Group HW due: Real Robot Manip.	
Week 11	**Midterm**/ Project		Mid-Report Due
Week 12	Autonomy / Project		
Week 13	Summary / Project		
Week 14	Project		
Week 15	Project		
Week 16	Project		
Week 17-18	**Final**		Report Web Demo

Mobile Manipulation HW

上海科技大学
ShanghaiTech University

Mobile M
IEEE Robotics

Jiawei
Andre Rosen

Fetch picking up

上海科技大学
ShanghaiTech University

Mobile Manipulation Tutorial
IEEE Robotics and Automation Magazine

Jiawei Hou, Yizheng Zhang,
Andre Rosendo and Sören Schwertfeger

Fetch picking up bottle in real environment

Music by audionautix.com

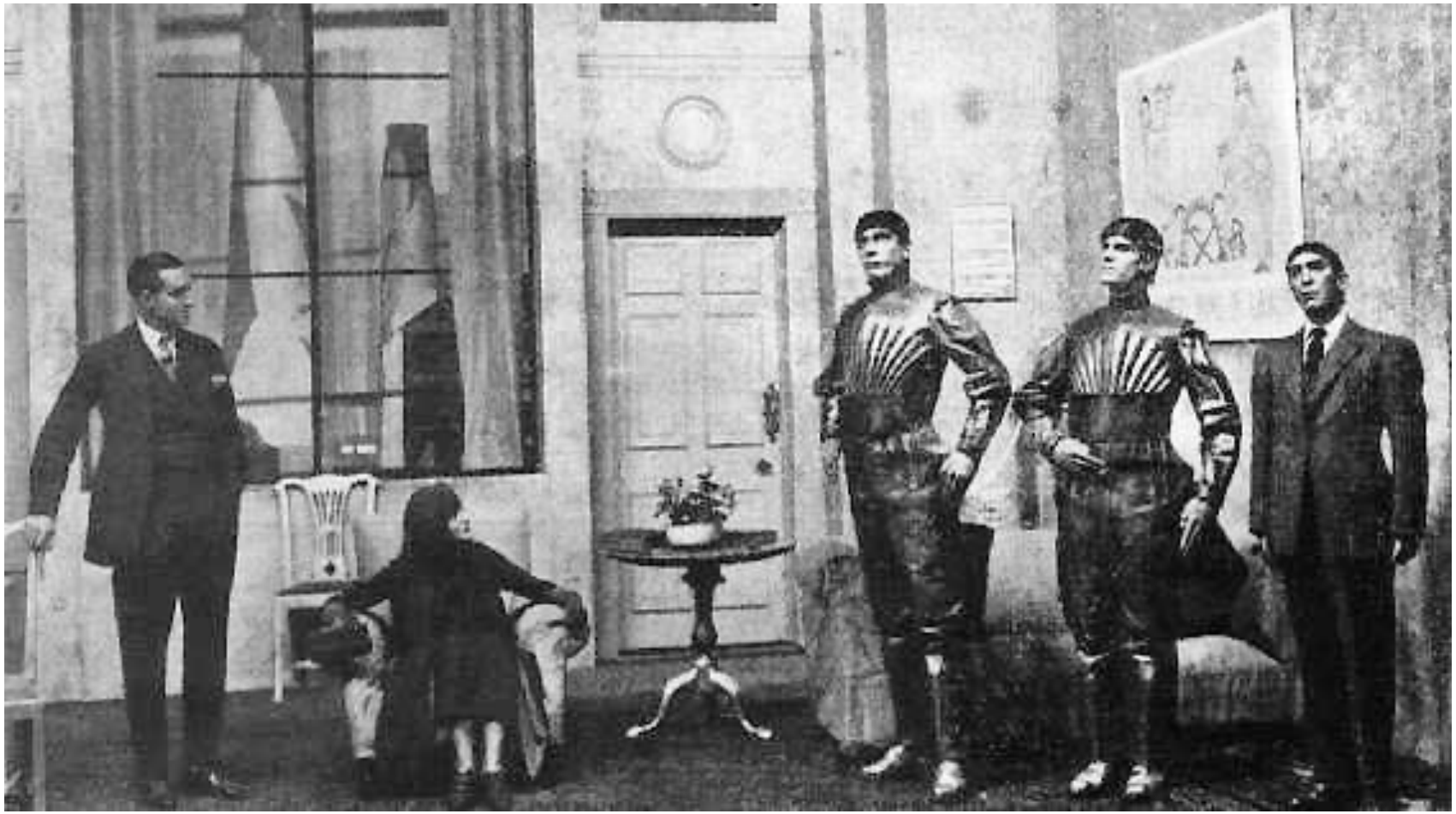
For this week

- Join the lecture on piazza
- Organize access to the two text books
- Do HW 1 (due Thursday, Sep 17, 22:00)
  - For the dual-boot installation of Ubuntu:
    - Backup your all your data
    - Free enough space (40 GB)
    - Download Ubuntu

BRIEF HISTORY

Brief History

Robota “forced labor”: Czech, Karel Čapek R.U.R. 'Rossum's Universal Robots' (1920).



Isaac Asimov - Three Laws of Robotics (1942)

1. A robot may not injure a human being or, through inaction, allow a human being to come to harm.

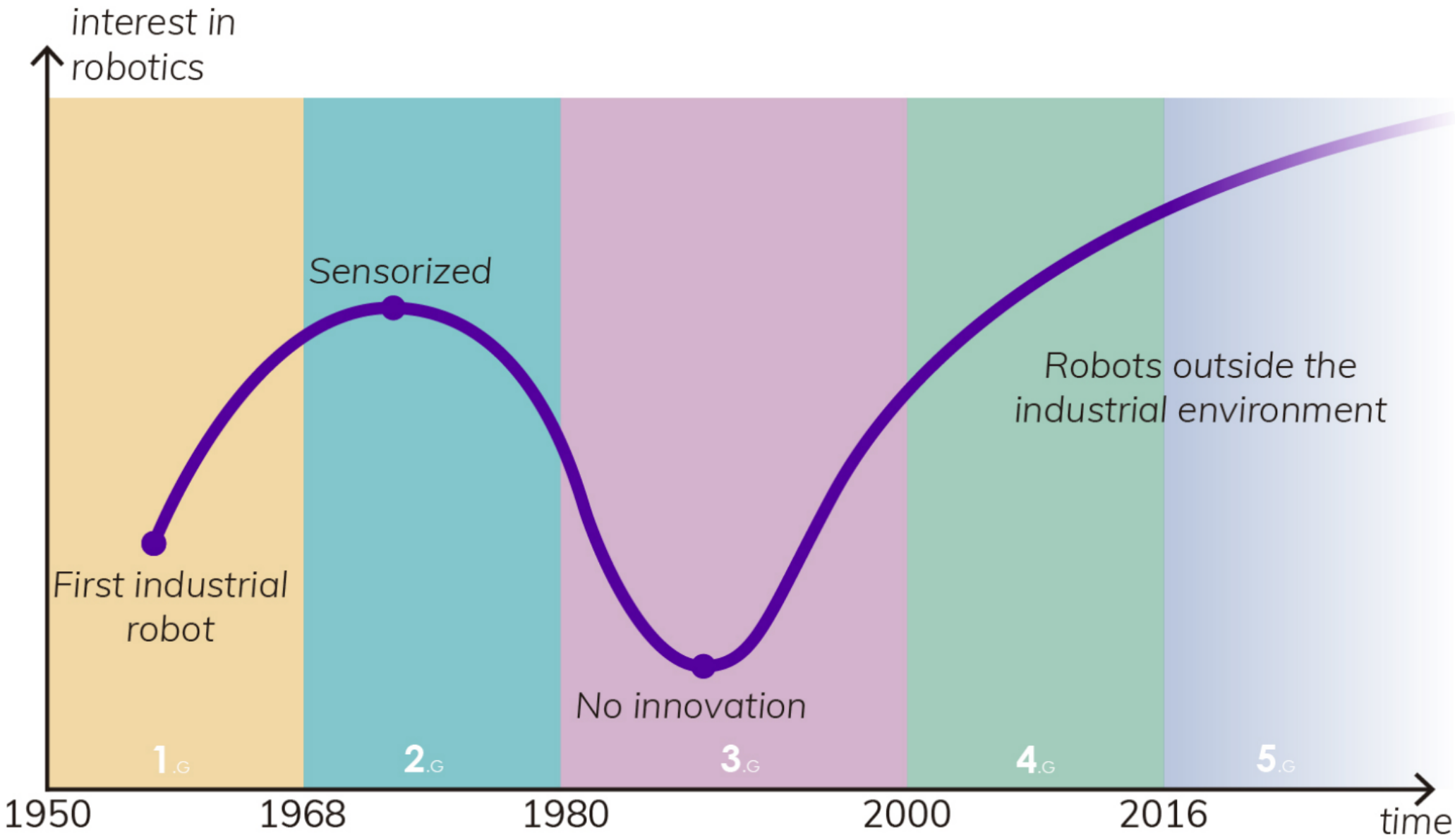
2. A robot must obey the orders given to it by human beings, except where such orders would conflict with the First Law.

3. A robot must protect its own existence as long as such protection does not conflict with the First or Second Law.

0. A robot may not harm humanity, or, by inaction, allow humanity to come to harm.

History

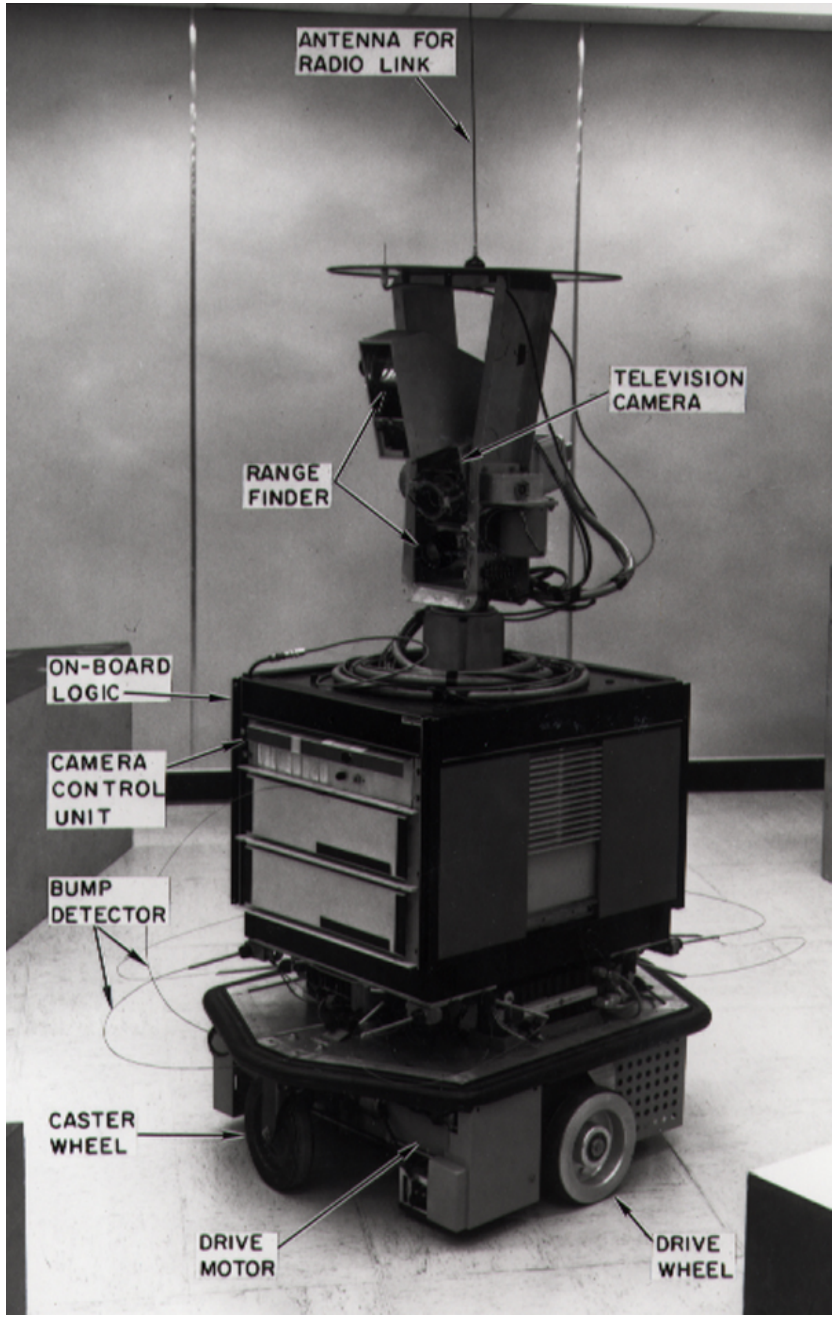
- First electronic autonomous robots 1949 in England (William Grey Walter, Burden Neurological Institute at Bristol)
  - three-wheeled robots: drive to recharging station using light source (phototaxis)
- Turing Test: 1950 (British mathematician Alan Turing)
- Unimate: 1961 lift hot pieces of metal from a die casting machine and stack them. First industry robot. Inventor: George Devol, user: General Motors.
- Lunokhod 1: 1970, lunar vehicle on the moon (Soviet Union)
- Shakey the robot: 1970
- 1989: Chess programs from Carnegie Mellon University defeat chess masters
- Aibo: 1999 Sony Robot Dog
- ASIMO: 2000 Honda (humanoid robot)



https://arxiv.org/pdf/1704.08617.pdf

Shakey the robot (1970)

- First general-purpose mobile robot to be able to reason about its own actions
- Advanced hardware:
  - radio communication
  - sonar range finders
  - television camera
  - on-board processors
  - bump detectors
- Advanced software:
  - Sensing and reasoning
- Very big impact
- https://robotics.shanghaitech.edu.cn/static/videos/Shakey.mp4



SOFTWARE

Robot Software: Tasks/ Modules/ Programs (ROS: node)

Support

- Communication with Micro controller
- Sensor drivers
- Networking
  - With other PCs, other Robots, Operators
- Data storage
  - Store all data for offline processing and simulation and testing
- Monitoring/ Watchdog

Robotics

- Control
- Navigation
- Planning
- Sensor data processing
  - e.g. Stereo processing, Image rectification
- Mapping
- Localization
- Object Recognition
- Mission Execution
- Task specific computing, e.g.:
  - View planning, Victim search, Planning for robot arm, …

Software Design

- Modularization:
  - Keep different software components separated
  - ☺ Keep complexity low
  - ☺ Easily exchange a component (with a different, better algorithm)
  - ☺ Easily exchange multiple components with simulation
  - ☺ Easily exchange dada from components with replay from hard disk instead of live sensor data
  - ☺ Multiple programming teams working on different components easier
  - Need: Clean definition of interfaces or exchange messages!
  - Allows: Multi-Process (vs. Single-Process, Multi-Thread) robot software system
  - Allows: Distributing computation over multiple computers

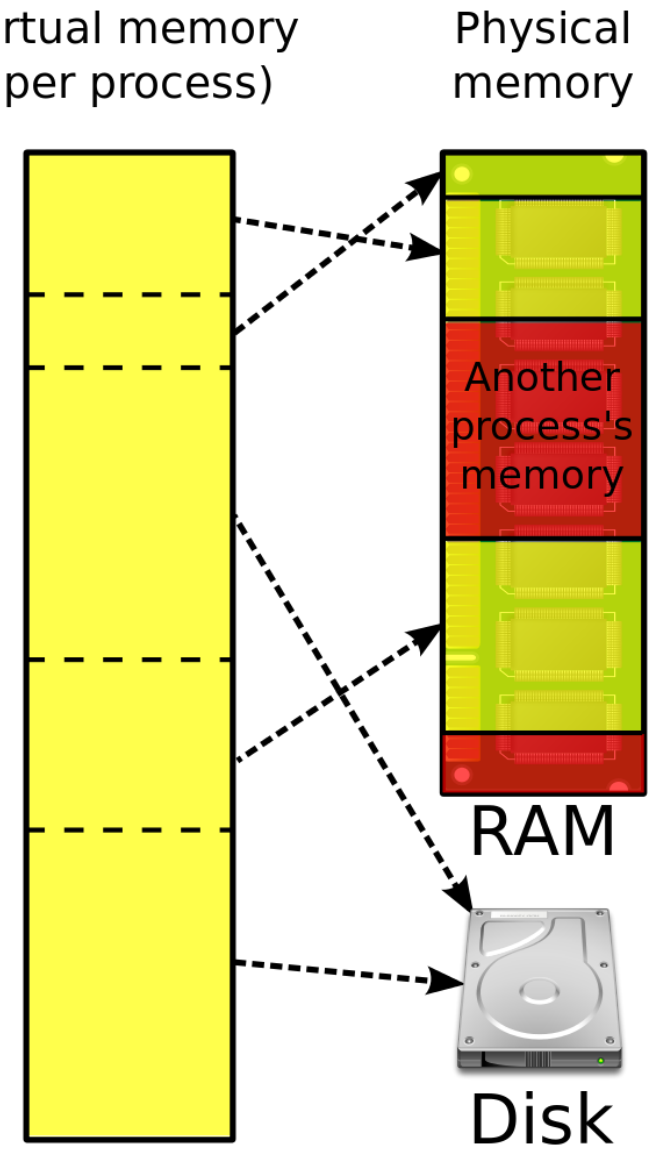
Programming review

- Process vs. Thread
- C++ Object Orientation
- Constant Variables
  - const-correctness
- C++ Templates
- Shared Pointer

- Objective:
  - Prerequisites for understanding ROS.
  - Understand how we can efficiently retrieve and transfer data in ROS.

Process

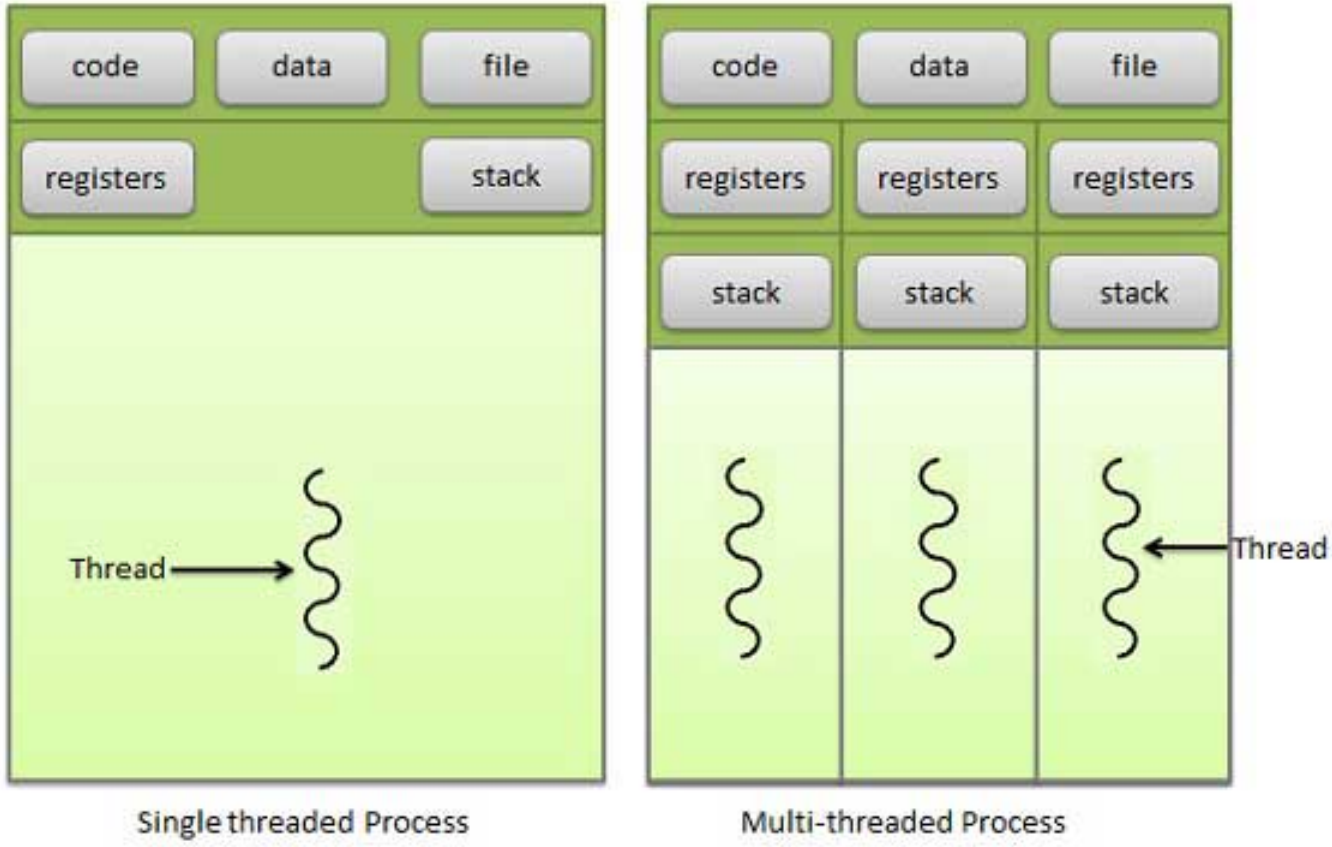
- Execution of one instance of a computer program
- Virtual memory:
  - Contains only code and data from this program, the libraries and the operating system
  - Other processes (programs) can not access this memory (shared memory access is possible but complicated)
- Operating system gives each process equal amount of processing time (scheduling) – if the processes need it
  - Good support from the operating system to give certain processes higher or lower priority
  - Linux console program to see processes: **top**



(From Wikipedia)

Multi-Threading

- In one process, multiple threads => parallel execution
- ☺ Code and Memory is shared => easy exchange of data, save mem.
- 😐 Synchronization can be tricky (mutex, dead lock, race condition)
- ☹ If one thread crashes, the whole process (all threads) die



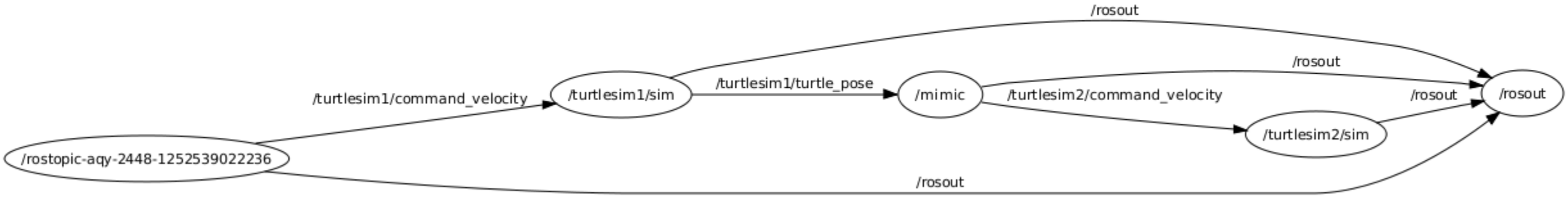
(from http://www.tutorialspoint.com)

Processes and Threads in Robotics - Messages

- Both approaches have been implemented!
- Both are used and important!
- Robot Operating System (ROS): Multiple Processes:
  - Each component runs in its own process: called **node**
  - A node can have multiple threads => faster computation
  - Nodes communicate using **messages**
  - A node can send (**publish**) **messages** under different names called **topic**
  - Nodes can listen to (**subscribe**) **messages** under different **topics**
  - The messages are transferred over the network (TCP/IP) => multiple computers work together transparently
  - ☹ Messages are serialized, copied and de-serialized even if both nodes on the same computer => slow (compared to pointer passing)
    - Optimization: **Nodelet**: run different nodes in the SAME process => pointer passing => fast

ROS nodes

- **ROS core**: keep track which **nodes** are running and their **topics**
- Show all nodes and topics in a graph: `rosrun rqt_graph rqt_graph`
  - `/rosout` : special node for output on console (standard out)
  - `/turtlesim1/sim, /turtlesim2/sim` : simulated robots (**nodes**) (multiple nodes per simulated robot)
  - `/command_velocity` : set the speed of a robot (**topic**)
  - **Node** `/turtlesim1/sim` **publishes** on **topic** `/turtlesim1/turtle_pose`
  - **Node** `/mimic` **subscribes** to **topic** `/turtlesim1/turtle_pose`



Constant Variables

- Declare variables that do not change (anymore) in the code: `const`
- Works for variables and objects
- Const Objects:
  - Only methods that do not change any variable of the object may be called =>
  - Those methods have to be declared const
- Used for program-correctness
- Especially for multi-threading:
  - Share the data (e.g. image)
  - Make it read only via const
  - => no side-effects between different threads

1. `const int x = 5; // x may not be changed`
2. `int * someValue = &x; // pointer – compilation error!!`
3. `const int * pointy = &x; // good`
4. `*pointy = 8; // error – pointing to const!`
5. `int y = 4;`
6. `pointy = &y; // from non const to const is always possible!`
7. `const int * p2 const = &y; // pointing to const variable and p2 is also const`
8. `p2 =&x; // error – p2 is const`

C++ Templates

- Functions and classes that operate with generic types
- Function or class works on many different data types without rewrite
  - `template <typename T> int compare( T v1, T v2);`
  - Type of T is determined during compile time => errors during compilation (and not run-time)
  - Any type (type == class) that offers the needed methods & variables can be used
  - Usage: `compare<string>( string("string number one"), "hello world" );`
    - Explicit declaration: typename T = string
    - typename T can (most often) deducted by the compiler from the argument types
- Class template:
  - ```
    template <typename T> class myStuff{
        T v1, v2;
        myStuff(T var1, T var2){  v1 = var2; v2 = var2; }
    };
    ```
    ```

Template example

```
//This example throws the following error : call of overloaded 'max(double, double)' is ambiguous
template <typename Type>
Type max(Type a, Type b) {
    return a > b ? a : b;
}
```

```
#include <iostream>

int main(int, char**)
{
  // This will call max <int> (by argument deduction)
  std::cout << max(3, 7) << std::endl;
  // This will call max<double> (by argument deduction)
  std::cout << max(3.0, 7.0) << std::endl;
  // This type is ambiguous, so explicitly instantiate max<double>
  std::cout << max<double>(3, 7.0) << std::endl;
  return 0;
}
```

Shared Pointer

- C++ Standard Library (std): heavily templated part of C++ Standard (many parts used to be in boost library)
- Pointer: address of some data in the heap – in the virtual address space
- Space for data has to be allocated (reserved) with: `new`
- After usage of data it has to be destroyed to free the memory: `delete`
- Problem: Data (e.g.) image is shared among different modules/ components/ threads. Who is the last user – who has to delete the data?
  - Shared pointer: counts the number of users (smart pointers); upon destruction of last user (smart pointer) the object gets destroyed : called “Reference counting”
  - Problem: Shared pointer needs to know the destructor method for the pointer =>
  - Shared pointer is a templated class: Template argument: class type of the object pointed to
  - Shared pointer can also point to const object!

Shared pointer example

```
std::shared_ptr<int> p1(new int(5));
std::shared_ptr<int> p2 = p1; //Both now own the memory.

p1.reset(); //Memory still exists, due to p2.
p2.reset(); //Deletes the memory, since no one else owns the memory.
```

- Earlier, shared_ptr used to be in boost
- Excerpt from ROS message of type “String” :

```
typedef boost::shared_ptr< ::std_msgs::String_<ContainerAllocator> > Ptr;
typedef boost::shared_ptr< ::std_msgs::String_<ContainerAllocator> const> ConstPtr;
```

- typedef: create another (shorter) name for a certain type
- Our type: a shared pointer that points to a (complicated) String object

```
void chatterCallback(const std_msgs::String::ConstPtr& msg)
{
  ROS_INFO("I heard: [%s]", msg->data.c_str());
}
```

Review for ROS

- Different components, modules, algorithms run in different processes: **nodes**
- Nodes communicate using **messages** (and **services** …)
- Nodes **publish** and **subscribe** to **messages** by using names (**topics**)
- **Messages** are often passed around as shared pointers which are
  - “write protected” using the const keyword
  - The shared pointers take the message type as template argument
  - Shared pointers can be accessed like normal pointers

```cpp
#include "ros/ros.h"
#include "std_msgs/String.h"
#include <sstream>

int main(int argc, char **argv){
  ros::init(argc, argv, "talker");
  ros::NodeHandle n;

  ros::Publisher chatter_pub = n.advertise<std_msgs::String>("chatter", 1000);

  ros::Rate loop_rate(10);
  int count = 0;
  while (ros::ok()){
    std_msgs::String msg;
    std::stringstream ss;
    ss << "hello world " << count;
    msg.data = ss.str();

    chatter_pub.publish(msg);

    ros::spinOnce();

    loop_rate.sleep();
    ++count;
  }
  return 0;
}
```

ROS Tutorial: Listener

```
1  #include "ros/ros.h"
2  #include "std_msgs/String.h"
3
4  void chatterCallback(const std_msgs::String::ConstPtr& msg){
5    ROS_INFO("I heard: [%s]", msg->data.c_str());
6  }
7
8  int main(int argc, char **argv){
9    ros::init(argc, argv, "listener");
10   ros::NodeHandle n;
11
12   ros::Subscriber sub = n.subscribe("chatter", 1000, chatterCallback);
13
14   ros::spin();
15
16   return 0;
17 }
```

Recourses:

- http://wiki.ros.org/ROS/Tutorials/
- https://en.wikipedia.org/wiki/Object-oriented_programming
- C++: http://www.cplusplus.com/doc/tutorial/
  - http://www.cplusplus.com/doc/tutorial/templates/
- https://en.wikipedia.org/wiki/Smart_pointer
  - http://en.cppreference.com/w/cpp/memory/shared_ptr
- http://www.cprogramming.com/tutorial/const_correctness.html

- Cheat sheets:
  - https://robotics.shanghaitech.edu.cn/static/cheatSheets/

Messages

- Publisher does not know about subscribers
- Subscribers do not know publishers
- One topic name: many subscribers and many publishers possible, BUT: same message type (determined by the first publisher)!
- List all topics in the current system:
  - rostopic list
  - Other commands: `rostopic echo`, `rostopic hz`, `rostopic pub` , `rostopic pub /test std_msgs/String “Hello World!"`

Create own message: Text format

- Types:
  - int8, int16, int32, int64 (plus uint*)
  - float32, float64
  - string
  - time, duration
  - other msg files
  - variable-length array[] and fixed-length array[C]
- Save in folder “msg”, start with big letter, end with “.msg”

```
string first_name
string last_name
uint8 age
uint32 score
```

Services

- ROS **service**: send a “message” or command to service provider, wait for reply
- Text format: First message for **request**
  - Separation: three dashes
  - Then message for **response**
- A call to a service blocks

```
float32 x
float32 y
float32 theta
string name
---
string name
```

```
#include "beginner_tutorials/AddTwoInts.h"

bool add(beginner_tutorials::AddTwoInts::Request  &req,
         beginner_tutorials::AddTwoInts::Response &res)
{
  res.sum = req.a + req.b;
  ROS_INFO("request: x=%ld, y=%ld", (long int)req.a, (long int)req.b);
  ROS_INFO("sending back response: [%ld]", (long int)res.sum);
  return true;
}
```

```
ros::ServiceServer service = n.advertiseService("add_two_ints", add);
```

Compiler, Linker

- Standard in Linux: gcc: GNU Compiler Collection
- Compiler: Create machine code out of programming language
  - For C++ code: g++
  - g++ -o helloworld -I/homes/me/randomplace/include helloworld.cc
  - Options:
    - -g - turn on debugging (so GDB gives more friendly output)
    - -o <name> - name of the output file
    - -O to -O4 - turn on optimizations
    - -L<library path> - specify a lib directory
    - -Wall - turns on most warnings
    - -c - output an object file (.o)
    - -I<include path> - specify an include directory
    - -l<library> - link with library lib<library>.a
- Linker: Link the machine code with other machine code (provided by libraries)
  - Static link library: executable includes the statically linked library
  - Dynamic link library: upon execution the program is linked against the library: Multiple programs will use the same code => save memory
  - Program: ln
  - Show dynamic linked libraries used by a program: ldd

Makefile, CMake

- Avoid typing g++ and ln
- Makefile:
  - Commands for compiling and linking the program: “make” uses the file “Makefile”
  - May provide additional commands like “make clean”
  - Can be used to run arbitrary commands, e.g. to create pdf files from LaTeX
- Cmake
  - Cross-platform Makefile generator
  - Searches for dependencies (libraries, headers, etc.)
  - Autoconfigure with “cmake .”
  - “CMakeLists.txt”: specify which files to make, etc.

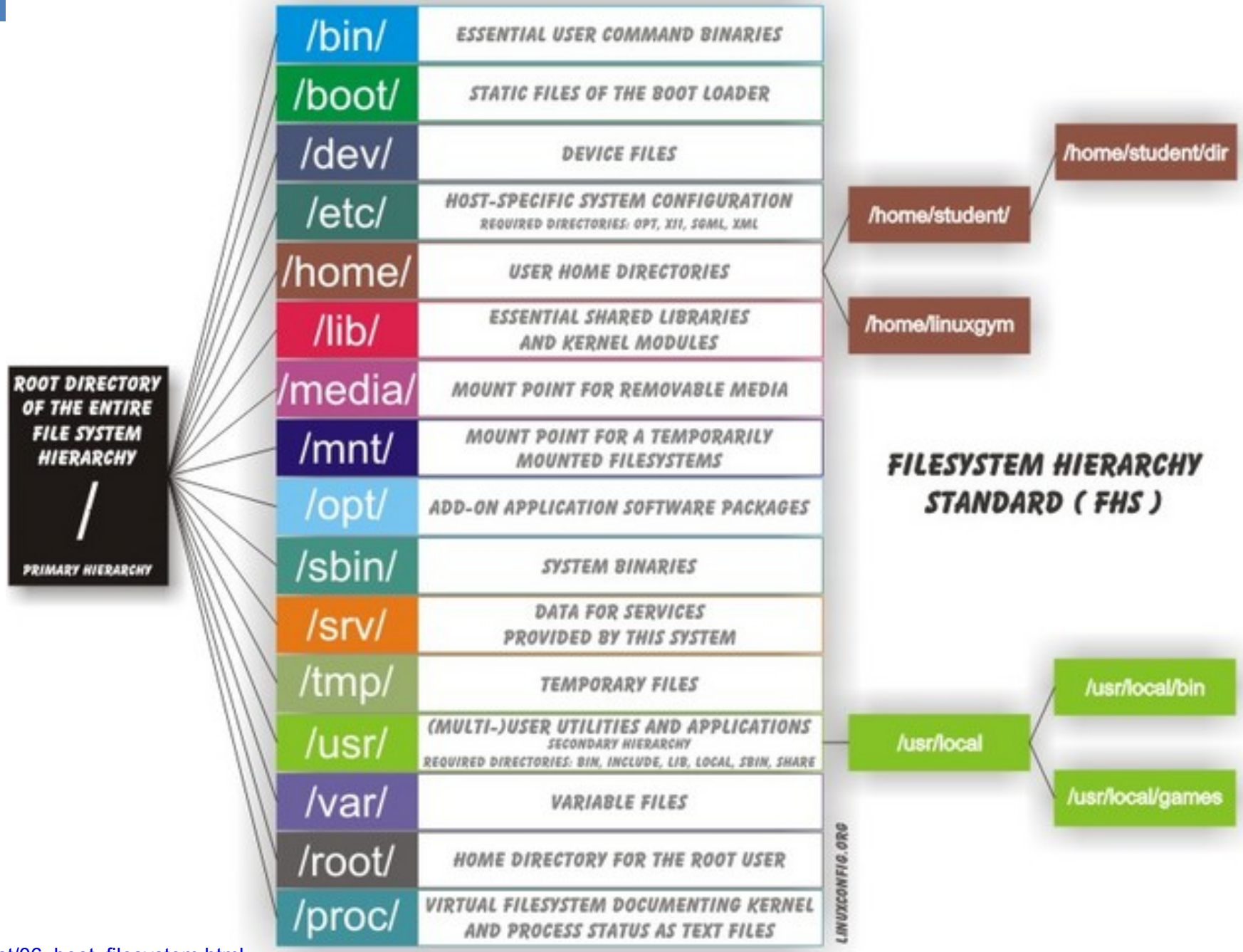
GIT: distributed revision control and source code management

- Every Git working directory is a full-fledged repository
  - => can work without server, two repos can pull/ push from each other
- Working directory has a hidden .git folder in its root
- Automatically merges common changes in same files
- Non-linear development:
  - Create branches, merge them
- Cryptographic authentication of history
- See Cheat Sheet

Unix File System

- File types: regular, directory, link, (sockets, named pipes, block devices)
- Slash “/” instead of backslash “\” for folders - distinction between small and big letters!
- One file system tree, beginning with root: “/”
  - Mount partitions (areas of the hard disk): any folder can be the mount point, e.g.: /media/<user_name>/usbDiskName
- Home folders of different users in “/home/<user_name>”
- Hidden files and folders: begin with a dot “.”
- In Unix/ Linux, (almost) everything is a file: devices, partitions, …in “/dev”, e.g. “/dev/video0”
- Show files: “ls”; more info: “ll”; human readable: “-h” – e.g. “ll -h”
- Free space: “df -h”
- Symbolic links (symlink): point to another file or folder. Create with “ln -s from to”

Overview



From: http://devopsbootcamp.readthedocs.org/en/latest/06_boot_filesystem.html

Misc

- Files have access rights: users and groups and others
  - r: read w: write x: execute (for directories: go in)
  - chmod a+w => all (three) are allowed to write
  - chmod o-r => others are not allowed to read
- chown user:group file_name_or_dir change ownership
- Super user: root: can access all files
- sudo <command>: execute a command as root
- sudo su: (one way) to become root
- Compress files: zip + rar for Windows => no support for permissions/ symbolic links
  - tar : tape archive (lol) – sequentially store files and folders (no compression)
  - gzip : compress one file
  - combine: tar gzip: archive.tar.gz

Bash: GNU Unix Shell

- Program that runs in your terminal – executes your commands
- Keyboard up: go through history of last commands
- Tab-complete: any time, press tab to complete the command/ path/ file-name/ … - if a unique solution exists; double tab for list of possible options
- Control C to tell program to stop; Control | to quit;
- Control Z to stop (pause) program: fg to run in foreground again, bg to run in background, kill %1 to kill the last program (in background)
- Start program in background: command &
- Pipe: send output of program 1 as input to program 2: prog1 | prog2; e.g. “ll /dev | less”
- Send standard output to file use “>” e.g.: “ll > file.txt”
- Wildcards: “*” matches anything with any length, “?” matches any one char, e.g. “ll /dev/tty*”

.bashrc

- .bashrc is executed every time a new shell (terminal) is opened
- Execute by hand: “source ~/.bashrc” or “. ~/.bashrc”
- “~” is replaced by your home directory
- Setup variables, e.g.:
  - alias df='df -h' # when calling df, acutally "df -h“ is called – human readable
  - alias ..='cd ..' # executing "..“ will go one level up in the file tree
  - Option: setup ros path always here: “source ~/my_ws/devel/setup.bash”
- Edit input.rc to search history of commands with page up, down:
  - “sudo vi /etc/inputrc” – uncomment “# alternate mappings for "page up" and "page down" to search the history”

vi: editor for the console

- Command mode (press escape) and input mode (press i)
- Install vim for more comfort: sudo apt-get install vim
- Command mode:
  - Press escape to enter command mode
  - “: w” write file
  - “: q” quit
  - “: wq” write file and quit
  - “: q!” quit without writing changes to file
  - Press “d” to delete a char; press “dd” to delete a line
  - Press “/” and enter a regular expression to search
  - Press “n” or “N” for next, previous search result

ssh: secure shell (to connect to a robot)

- Login to remote computer, using encrypted communication
- sudo apt-get install ssh : Installs the ssh server
- Usage: ssh user@host e.g.: ssh schwerti@robotics.shanghaitech.edu.cn
- Option: -X forward X-server: see GUI of remote application on your screen (-Y without encryption)
- ssh-keygen : generate authentication keys – public and private keyfile in .ssh
- ssh-copy-id : copy your public key to remove hose => no login needed anymore!
- Copy files: scp [-r] <from> <to>
  - Either from or to can be remote host: [user@]host:path, e.g. scp hw2.tar.gz test@robotics:homeworks/
  - -r: recursive – copies whole directories