UNIVERSITÉ DE TECHNOLOGIE DE BELFORT-MONTBÉLIARD

System Integration

RO51 - Introduction to Mobile Robotics

Zhi Yan
May 22, 2024

https://yzrobot.github.io/

www.utbm.fr

utbm
université de technologie
Belfort-Montbéliard

What?

- Bringing together the physical components (hardware) and the controlling programs (software) to create a single, functional system.
- **Software-hardware integration** for mobile robotics.
- Like having two actors work together seamlessly in a play.

What?

- **Hardware:** thermometer, microcontroller, led display, knob
- **Software:** turn the heating or cooling on and off based on the settings
- **Integration:** understand the data from the temperature sensor and control the heating system

Credits: Olivier Le Moal - adobe.stock.com

Why?

- Ensure performance and optimize resource utilization: $C(software) > or < C(hardware)$
- Reduce system redundancy
- Reduce development costs
- Improve iteration efficiency

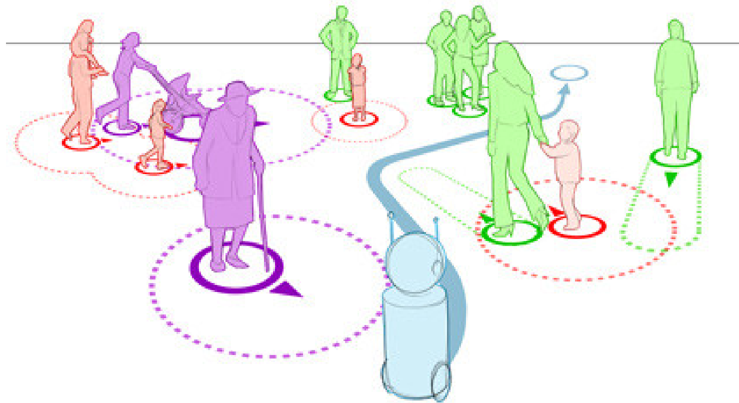
How?

- **Define requirements and system architecture**
  - Understand the purpose
  - Identify hardware and software components
  - Plan communication protocols
- **Hardware selection and configuration**
  - Choose compatible hardware
  - Set up hardware properly
- **Software development and integration**
  - Write device drivers (if needed)
  - Develop the main software program
  - Implement data exchange protocols
- **Testing and validation**
  - Thorough testing is crucial
  - Validate functionality

=> Successful software-hardware integration is an iterative process!

Integration in practice

Goal: Build a robot that can navigate crowds

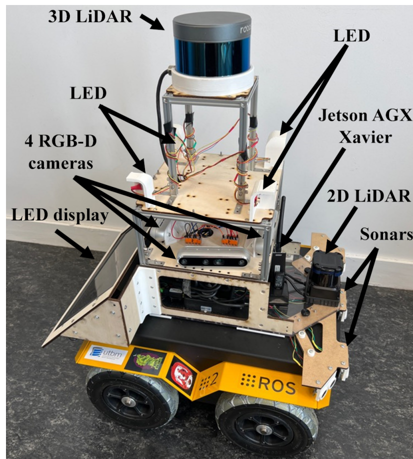


Copyright: EU project SPENCER (2013-2016)

Integration in practice

Hardware[1]:

- **Sensors:** a 3D lidar, a 2D lidar, four RGB-D cameras and a sonar belt
- **Computing Units:** a CPU-based computing unit, a CPU-GPU-based computing unit
- **Communications:** USB, Ethernet
- **Peripherals:** a set of LED lights, a LED display

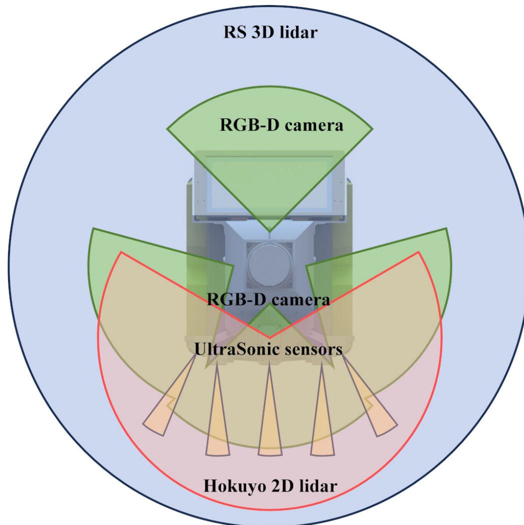


[1] https://github.com/Nedzhaken/human_aware_navigation

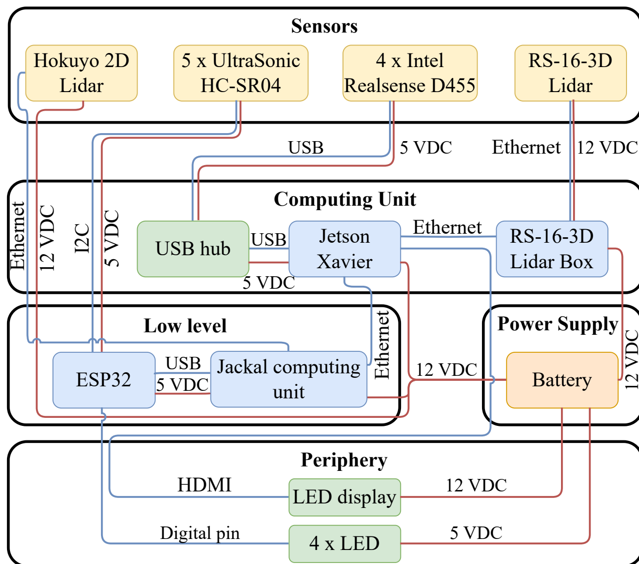
Integration in practice

Sensor	Data release frequency	Measure. distance	Horizontal FoV	Vertical FoV
3D lidar	10 Hz	150 m	360°	30°
2D lidar	40 Hz	30 m	270°	-
RGB-D cam.	30 Hz	6 m	87°	58°
Sonar	40 Hz	4 m	15°	-

Integration in practice

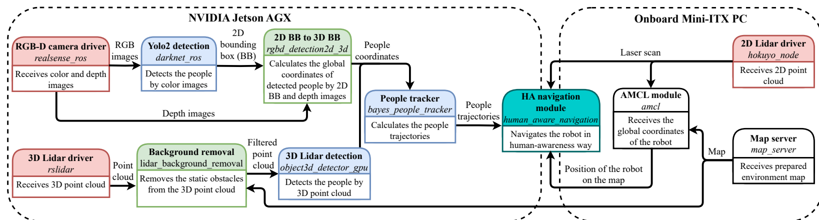


Integration in practice



Integration in practice

Software:



Module	Input	Output	Freq.
3D lidar driver	Raw data	3D point cloud	10 Hz
Background removal	3D point cloud, map	3D point cloud	10 Hz
3D object detector	3D point cloud	3D bounding box	20 Hz
RGB-D camera driver	Raw data	Color and depth images	30 Hz
YOLOv2 2D object detector	Color image	2D bounding box	44 Hz
2D bounding box to 3D	2D bounding box, depth image	3D bounding box	25 Hz
Multi-target tracker	3D bounding box	Human trajectory, etc.	30 Hz
2D lidar driver	Raw data	2D point	40 Hz
Localization	2D point, map	Robot pose	40 Hz

Integrating a robot system using ROS packages

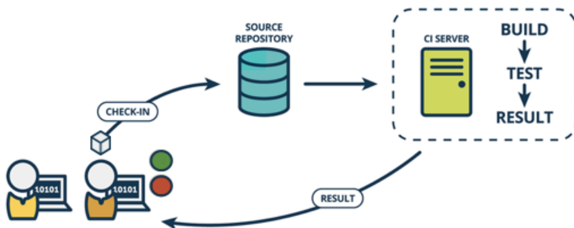
A few tips:

1. Draw a functional block diagram representing the system that needs to be integrated.
2. Determine dependencies of ROS packages (e.g. *rosdep*, *colcon*).
3. Define enabling elements for communication between packages such as topics, services, actions, etc.
4. Use the tools provided by ROS (e.g. *rqt_graph*, *rqt_monitor*) for debugging.

ROS

Continuous Integration

- **What?** Continuous Integration (CI) is a software development practice where developers frequently merge their code changes into a central repository, triggering automated builds and tests.
- **Why?** It helps identify any (integration) issues or bugs early in the development lifecycle, improve software quality, and reduce time to release.
- **How?**



Continuous Integration

A breakdown of how CI typically works:

1. Developer makes changes
2. Commit and push (to the central repository)
3. CI triggered (by a CI server)
4. Automated build
5. Automated testing
6. Feedback and action: “tests pass” or “tests fail”

Continuous Integration

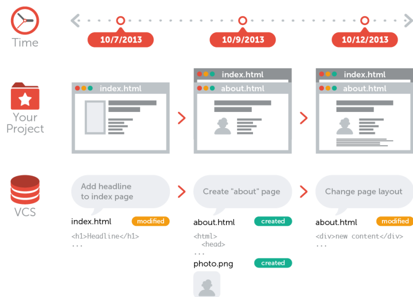
Some well-known CI tools

Continuous Integration

- **CircleCI**
  - A CI & Continuous Delivery (CD) platform for DevOps practices
  - Facebook, Coinbase, Sony, Kickstarter, GoPro, and Spotify used it
  - Automatically test builds in either **Docker** containers or virtual machines
- **Travis CI**
  - Closely connected with online repositories such as GitHub, Bitbucket, and GitLab
  - Free for open source projects until 2020
  - An example: `https://github.com/yzrobot/adaptive_clustering/blob/master/.travis.yml`
- **Jenkins**
  - Written in Java and fully open source
  - Can be deployed locally

Version Control

- **What?** Version Control (VC) is a system that keeps track of changes to a set of files (e.g. code, blueprint, manuscript, etc.) over time.
- **Why?** Prevent data loss (data security), make collaboration easy, improve code quality (problem tracing), try new ideas (low-risk experimentation).
- **How?** (Image Source: Tower)



Version Control

A breakdown of how VC typically works:

1. **Centralized repository:** A central location that stores all the files and their entire history, can be a server or a cloud-based storage solution.
2. **Snapshot:** Whenever someone makes a change to a file, VC takes a snapshot of that file and stores it in the repository, along with information like who made the change, why they made it, and when.
3. **Version history:** A history of all the changes made to a file.
4. **Reverting changes:** If there's a problem with the current version of a file, one can easily revert back to a previous version that worked.
5. **Branching:** Branches of the main project can be created for development of new features or bug fixes without affecting the main code base.

Version Control

SVN vs. Git:

- **SVN (Subversion):** A centralized VC, developers work on their local copies and need to connect to a central server to download updates or upload changes.
- **Git:** A distributed VC. Every developer has a complete copy of the project history on their machine, and can therefore work offline and collaborate by sharing their local copies with each other.

Feature	**SVN**	**Git**
Model	Centralized	Distributed
Offline work	Limited	Possible
Branching complexity	Simpler	More complex but powerful
Learning curve	Easier	Steeper
Conflict resolution	Can be trickier	More robust

Version Control

Which one is better?

- **SVN:** Small team, prioritize simplicity, and need fine-grained access control.
- **Git:** Larger team, value offline work, need powerful branching features, or use platforms like GitHub.

In summary, Git is the more popular choice nowadays due to its flexibility and scalability. However, SVN can still be a good option for specific workflows.

Docker

What?

- An **open-source** platform for developing, deploying, and running applications.
- Packages applications and their dependencies into standardized units called **containers**.
- Containers are lightweight, portable, and self-contained.

Docker

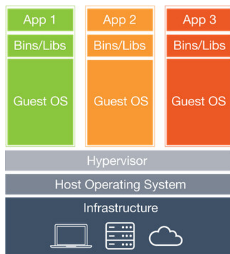
Why?

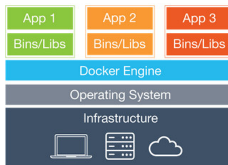
- **Consistency:** Applications run the same way everywhere.
- **Isolation:** Applications run in isolation, preventing conflicts.
- **Portability:** Containers run on any system with Docker installed.
- **Agility:** Faster development, testing, and deployment cycles.
- **Scalability:** Easily scale applications up or down by adding or removing containers.

Docker

How?

- **Images:** Read-only templates that define the contents of a container.
- **Containers:** Running instances of images.
- **Registries:** Stores for sharing and downloading Docker images.





Docker

Getting started with Docker:

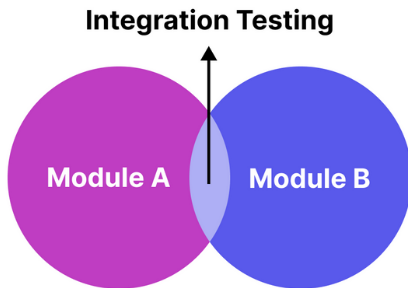
- Install Docker on your system: https://docs.docker.com/get-docker/
- Run a simple hello world container: `$ docker run hello-world`
- Explore Docker Hub: https://hub.docker.com/

```
$ docker run hello-world

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
    (amd64)
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.
```

Integration Testing

What?

- Focuses on interfaces between software modules.
- Verifies data flow and interaction between components.
- Checks if the combined modules fulfill the intended functionality.
- Often follows unit testing (but before system testing), which focuses on individual units.



Integration Testing

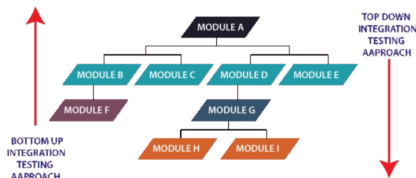
Why?

- **Early defect detection:** Identify issues arising from module interactions before system testing.
- **Improved system stability:** Ensure modules work together cohesively to prevent system crashes.
- **Enhanced reliability:** Verify data integrity and consistency as it flows between modules.
- **Reduced development costs:** Fixing integration issues early is cheaper than fixing system-wide problems later.

Integration Testing

How?

- **Top-Down Integration:**
  - Starts with high-level modules and progressively integrates lower-level modules.
  - Like assembling a tree, starting from the trunk and adding branches.
- **Bottom-Up Integration:**
  - Starts with low-level modules and integrates them into increasingly complex subsystems.
  - Like building a pyramid, starting with the base and adding layers on top.



Integration Testing

A few tips:

1. **Plan your tests:** Define clear objectives and identify modules to be integrated.
2. **Use mocks and stubs:** Simulate dependencies to isolate modules and control test environment.
3. **Automate your tests:** Use testing frameworks to streamline the testing process and ensure repeatability.
4. **Document your tests:** Clearly record test cases, expected results, and any assumptions made.

							Test Cases Template 1.0
Project Code				Created By:		Verified By:	
Test ID				Use Cases Reference			
Test Type		☐Unit Test		☐Integration		☐System	
Test Environment							
Quality Covered							
Pre-Conditions							
Post-Conditions							
Notes							
Testing Steps							
No.	Description	Test Data	Expected Results	Actual Results	Severity	Test Status	Remarks
				Test Date:			
				Total Test Time:			
				Test By:			

Summary

- Software-hardware integration
- Continuous integration
- Version control
- Docker
- Integration testing

The end

Thank you for your attention!

Any questions?