Advanced Techniques for Mobile Robotics

Robot Software Architectures

Wolfram Burgard, Cyrill Stachniss, Kai Arras, Maren Bennewitz
How to Program a Robot

- Robots are rather complex systems
- Often, a large set of individual capabilities is needed
- Flexible composition of different capabilities for different tasks

In this lecture, we discuss:

- What are important aspects of a robot architecture?
- What are good design decisions?
Discussion

- What do you think is important?

- Consider you want to build your own robot control software. What are relevant design decisions for that software?
Requirements from a Academic Perspective

- Support for multiple components
- Communication between components
- Easy way to write own components
- Possibility to replace individual components
- Easy to extend
- Means for data logging and debugging
- Support for decentralized components
Desired Features from a Academic Perspective

- Robustness
- Hardware abstractions
- Open access (ideal case: open source)
- Hardware/OS independent
- Means for time stamping
- Means for visualization
- ...
Example

user interface

path planning

collision avoidance

base interface

base driver

sensor interface(s)

sensor driver(s)

localization

hardware

user interface

path planning

collision avoidance

base interface

base driver

sensor interface(s)

sensor driver(s)

localization

hardware
Communication Examples

- Message-based systems

- Direct (shared) memory access
Forms of Communication

- Push
- Pull
- Publish/Subscribe
- Publish to blackboard
Push

- One-way communication
- Send as the information is generated by the producer P

![Diagram: P (producer) sends data to C (consumer)]
Pull

- Data is delivered upon request by the consumer \( C \)
- Useful if the consumer \( C \) controls the process and the data is not required at high frequency
Publish/Subscribe

- The consumer C requests a subscription for the data by the producer P
- The producer P sends the subscribed data as it is generated to C
- Data generated according to a trigger (e.g., sensor data, pose corrections, ...)

\[
\begin{align*}
\text{P} & \quad \text{subscription request} \quad \text{C} \\
data (t=1) & \\
data (t=2) & \\
data (\ldots) & 
\end{align*}
\]
Publish to Blackboard

- The producer P send data to a blackboard
- A consumer C pulls data from the blackboard B
- Only the last instance of the data is stored in the blackboard B
- New data from P overrides previously sent data
Example: Laser Range Sensor

- Driver for the LRF reads the data from the hardware device (serial, USB, ...)
- LRF driver offers a subscription to the topic “laser data”
- The localization module subscribes to the topic “laser data”
- The LRF driver will send every new laser range information to the localization module
Communication Infrastructure

- A communication infrastructure/robotic middleware is needed that provides such forms of communication
- There exists a large set of such infrastructures (not only for robotics)

Examples (used in robotics)
- IPC by Reid Simmons (used in Carmen)
- MOOSDB by Paul Newman
- ROS-Master by Willow Garage
- ...
IPC, MOOS, ROS, and Friends

- Are created for easy data exchange
- Communication within and among processes ("programs")
- Transparent network support
- Designed for "friendly environments"
ROS Master

P advertises to ROS Master

C
ROS Master subscribes P C
ROS Master

P → ROS Master → C

data
IPC Central & MOOSDB

CENTRAL / MOOSDB

P

Msg definition

C
IPC Central & MOOSDB

CENTRAL / MOOSDB

P

C

subscribes
IPC Central & MOOSDB

CENTRAL / MOOSDB

P

data

C
Messages Through the Central

- **Discuss:** pros and cons
Messages Through the Central

Pro
- Better control over message flow
- Transparent logging (time-stamping)
- No dead processes
- Centralized “health monitoring”

Con
- Slower/bigger delays
- Higher network traffic in decentralized systems
Other Differences

- Need to share header files
- Typed vs. non-typed messages
- Binary data vs. human readable strings
- Platform/OS independence
- Time synchronization
- ...

Messages for Communication

- Each module provides a list of messages it sends (e.g., via publish) or wants to receive (via pull)
- This list of messages is the only way of communication (black box)

- Example:
A message definition:

```c
typedef struct {
    double x, y, theta;
    double tv, rv;
    double acceleration;
    double timestamp;
    char *host;
} carmen_base_odometry_message;
```

A helper function to subscribe the message:

```c
void
carmen_base_subscribe_odometry_message(
    carmen_base_odometry_message *odometry,
    carmen_handler_t handler,
    carmen_subscribe_t subscribe_how);
```
typedef struct {
    double x, y, theta;
    double tv, rv;
    double acceleration;
    double timestamp;
    char *host;
} carmen_base_odometry_message;

- Every message contains a timestamp and the name of the sending host
Often, modules provide helper functions that encapsulate sending messages. For example, calling

```c
void carmen_robot_velocity_command(  
    double tv, double rv);
```

Sends the message

```c
typedef struct {  
    double tv, rv;  
    double timestamp;  
    char *host;  
} carmen_robot_velocity_message;
```
Modules

- Most systems use the modules (or nodes)
- Often, each module represents one task (localization, path planning, a driver, ...)
- Each module runs as an own process

Discuss: why is this done like that?
Modules

- Most systems use the modules (or nodes)
- Often, each module represents one task (localization, path planning, a driver, ...)
- Each module runs as an own process
- Modules can be replaced easily
- Modules can be distributed between machines
- If a module dies, this does not affect the other components (at least they can react)
- Separation between module and its GUI

**Discuss:** why separating the GUI?
Separation of the User Interface

- It is a good advice to separate a component/module from is GUI
- GUIs can run remotely
- GUIs may require significant resources (on the robot, that can be critical)
- Often OpenGL GUIs for 3D visualizations will not run on the robot (graphics card)

...but often less nice to code...
Parameters

- “There should be only a single parameter file. No exceptions.”
- Parameters should be handled centrally
- Modules should only be allowed to read and write parameters via a centralized mechanism
Parameters in Carmen

- There is only one ini file
- It is read by one process (param_daemon)
- Modules can query and set parameters
- Modules get notified if a parameter has changed online

Examples:

```c
int carmen_param_get_int(char *variable, int *return_value);
int carmen_param_get_double(char *variable, double *return_value);
```
Example

- User Interface
- Path Planning
- Collision Avoidance
- Localization
- Base Interface
- Sensor Interface(s)
- Base Driver
- Sensor Driver(s)
- Hardware
- Parameter Server
- Middleware
Logging Data

- **Discuss:** why logging data?
Logging Data...

- for post-processing
- for documenting experiments
- for being independent from a robot running 24/7
- for debugging and reproducing failures
- for collecting training data
- ...
Logging Data

- Good systems provide an easy way to log data
- Time-stamped data
- Transparent logging and playback (no distinction between played back data and real robot generating data online)
- Ideal case: everything can be logged transparently
Log Formats

- Human readable formats vs. self-made binary formats

- Discuss: pros and cons!
Log Formats

- MOOS: Human readable formats, logs everything (String-based messages)
- Carmen (IPC): Human readable formats, newly defined message required changes in the logger/playback component
- ROS: binary format, can log transparently, logfile compatibility between versions
Logfile Example
Units & Coordinate Frames

- All modules should use the same units
- SI units (meter, kilogram, second, ...)

- All modules should use the same coordinate frame
- ... here the problems start in practice
- Different communities use different frames
- Especially in 3D, there are representations with different properties
Units & Coordinate Frames

- Using different units and/or different coordinate frames is one serious source of errors.

Example: Mars Climate Orbiter, 1999

- One company used English units and the other used SI for controlling the thrusters.
- This lead to a wrongly calculated orbit altitude and finally the orbiter entered the atmosphere and burned.
Most Commonly Used in 2D

\[ \theta = 0 \] looks along the x-axis
Simulations

- Simulations are always incomplete
- Simulations will never replace real world experiments

**Discuss**: Why are simulations useful?
Simulations are Useful

- Possibility to get ground truth
- Control the amount of noise
- Control over the time dimension
- Test of the communication flow
- Test software with the risk of ruining expensive hardware
- Useful for debugging
- No hardware/robot required
- ...
Summary – Important Issues

- Flexible communication architecture
- Message-based communication
- Network transparency
- Easy to use and transparent logging and playback capabilities
- Centralized parameter handling
- Abstracts higher level components from the actual hardware (robot/sensors)
- SI units and one reference frame
Part II: Carmen

...in action
More Details on Carmen

http://carmen.sourceforge.net
Install Carmen

- Download carmen from http://carmen.sourceforge.net

- tar xzf carmen.tgz ~/
- export CARMEN_HOME=~/carmen
- cd $CARMEN_HOME/src
- ./configure
- make
Running Carmen

- cd $CARMEN_HOME/bin
- ./central
- ./param_daemon -r p2d8+ ../data/freiburg.map
- ./simulator
- ./robot
- ./localize
- ./navigator
- ./navigator_panel
  (click place robot to initialize the simulator and the localization)
- ./robotgui