EMC 2014
Software Design

Sjoerd van den Dries

Eindhoven University of Technology

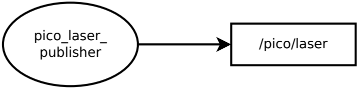
Department of Mechanical Engineering

May 8, 2014

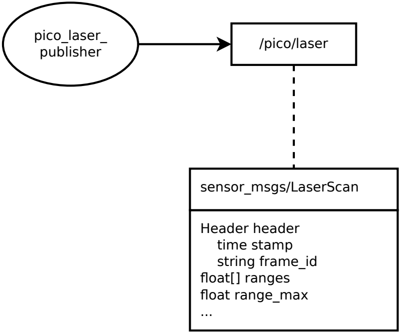
ROS Communication Example

pico_laser_
publisher

ROS Communication Example



ROS Communication Example

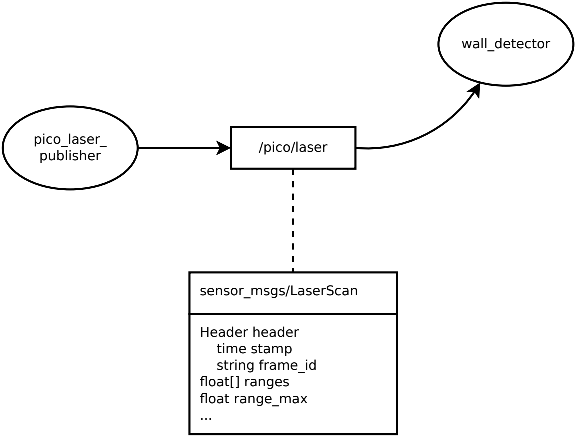

pico_laser_ publisher

/pico/laser

sensor_msgs/LaserScan

Header header
    time stamp
    string frame_id
float[] ranges
float range_max
...

ROS Communication Example



wall_detector

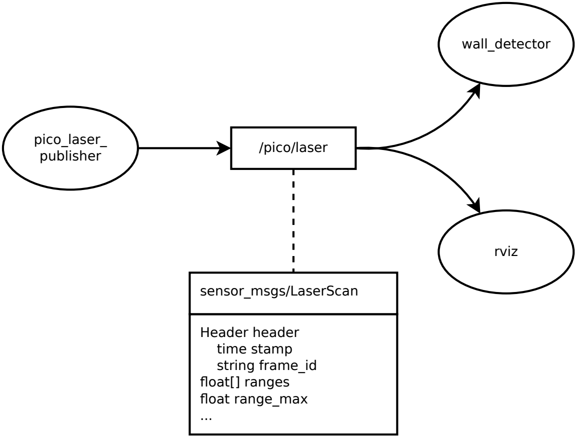
pico_laser_ publisher

/pico/laser

sensor_msgs/LaserScan

```
Header header
    time stamp
    string frame_id
float[] ranges
float range_max
...
```

ROS Communication Example



pico_laser_
publisher

/pico/laser

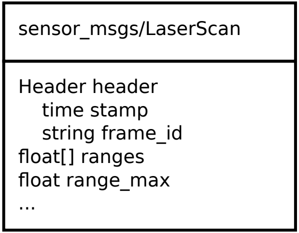
wall_detector

rviz

sensor_msgs/LaserScan

Header header
    time stamp
    string frame_id
float[] ranges
float range_max
...

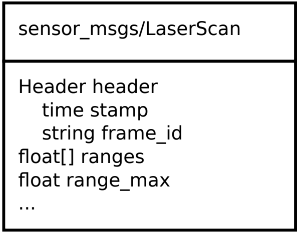
ROS Message Types



```
sensor_msgs/LaserScan

Header header
   time stamp
   string frame_id
float[] ranges
float range_max
...
```

ROS Message Types

Is in fact:



sensor_msgs/LaserScan

Header header
    time stamp
    string frame_id
float[] ranges
float range_max
...

```
struct LaserScan {
    Header header;
    float range_max;
    std::vector<float> ranges;
}

struct Header {
    std::string frame_id;
    Time stamp;
}

struct Time {
    int secs;
    int nsecs;
}
```

ROS Message Types

```
sensor_msgs/LaserScan

Header header
   time stamp
   string frame_id
float[] ranges
float range_max
...
```

Is in fact:

```
struct LaserScan {
    Header header;
    float range_max;
    std::vector<float> ranges;
}

struct Header {
    std::string frame_id;
    Time stamp;
}

struct Time {
    int secs;
    int nsecs;
}
```

ROS Message Types

sensor_msgs/LaserScan

- Header header
  - time stamp
  - string frame_id
- float[] ranges
- float range_max
- ...

Is in fact:

```
struct LaserScan {
    Header header;
    float range_max;
    std::vector<float> ranges;
}

struct Header {
    std::string frame_id;
    Time stamp;
}

struct Time {
    int secs;
    int nsecs;
}
```

ROS and C++

ROS and C++

- To use ROS in your program:

```
#include <ros/ros.h>
```

ROS and C++

- To use ROS in your program:

```
#include <ros/ros.h>
```

- Register your program as a node to the ROS master:

```
ros::init(..., "your_node_name");
```

ROS and C++

- To use ROS in your program:

```
#include <ros/ros.h>
```

- Register your program as a node to the ROS master:

```
ros::init(..., "your_node_name");
```

- Let ROS know you want to listen to a certain topic:

```
ros::NodeHandle n;
ros::Subscriber sub = n.subscribe("/pico/laser", 1,
                                  callbackFunction);
```

ROS and C++

- Let ROS know you want to listen to a certain topic:

```
ros::NodeHandle n;
ros::Subscriber sub = n.subscribe("/pico/laser", 1,
                                  callbackFunction);
```

ROS and C++

- Let ROS know you want to listen to a certain topic:

```
ros::NodeHandle n;
ros::Subscriber sub = n.subscribe("/pico/laser", 1,
                                  callbackFunction);
```

- Start listening to the topics:

```
ros::spin();
```

ROS and C++

- Let ROS know you want to listen to a certain topic:

```
ros::NodeHandle n;
ros::Subscriber sub = n.subscribe("/pico/laser", 1,
                                  callbackFunction);
```

- Start listening to the topics:

```
ros::spin();
```

- This function is called every time the node receives a message:

```
void callbackFunction(sensor_msgs::LaserScan scan) {
    // do something
    std::cout << scan.header.stamp << std::endl;
}
```

PICO Safe Drive

PICO Safe Drive

```
// Include ROS framework (Publishers, Subcribers, init, etc)
#include <ros/ros.h>

// Include the LaserScan message type
#include <sensor_msgs/LaserScan.h>

// Include the Twist message type (used for sending velocity
  commands to the base)
#include <geometry_msgs/Twist.h>

// Global variables
bool drive = true;
ros::Publisher cmd_pub;
```

PICO Safe Drive

```
int main(int argc, char** argv) {
// Register your ROS node
ros::init(argc, argv, "pico_safe_drive");

// Create node handle
ros::NodeHandle n;

// Subscribe to topic '/pico/laser' topic
ros::Subscriber sub = n.subscribe("/pico/laser",1,laserCallback);

// Create 'cmd_vel' publisher
cmd_pub = n.advertise<geometry_msgs::Twist>("/pico/cmd_vel", 10);

// Program loop
while (ros::ok()) {
    ros::spinOnce(); // Check incoming messages
    sendVelocity();  // Publish velocity
    ros::Duration(0.1).sleep(); // Sleep 0.1 seconds
}

return 0;
```

PICO Safe Drive

```
void laserCallback(sensor_msgs::LaserScan scan) {

    // Default: drive
    drive = true;

    // Check all laser points
    for(unsigned int i = 0; i < scan.ranges.size(); i++) {
        // Check laser point distance
        if (scan.ranges[i] > 0.1 && scan.ranges[i] < 0.3) {
            // Oh no, something is near! Better stop driving...
            drive = false;
        }
    }
}
```

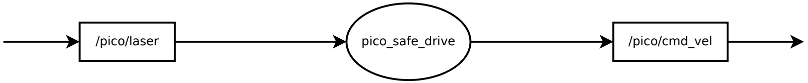
PICO Safe Drive

```
void sendVelocity() {
    // Create a ROS Twist message
    geometry_msgs::Twist cmd_msg;

    // Set forward velocity
    if (drive) {
        cmd_msg.linear.x = 0.2;
    } else {
        cmd_msg.linear.x = 0;
    }

    // Set all the other components to 0
    cmd_msg.linear.y = 0;
    cmd_msg.linear.z = 0;

    cmd_msg.angular.x = 0;
    cmd_msg.angular.y = 0;
    cmd_msg.angular.z = 0;

    // Send the command!
    cmd_pub.publish(cmd_msg);
}
```

Safe_drive Example

/pico/laser
/pico/cmd_vel

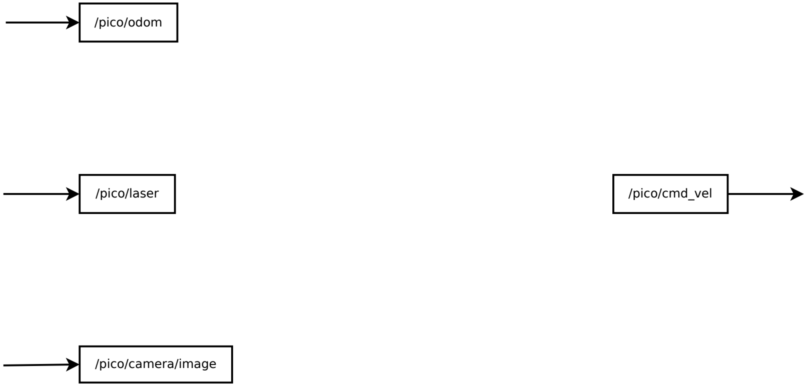
Safe_drive Example



System Input / Output

/pico/laser
/pico/cmd_vel

System Input / Output



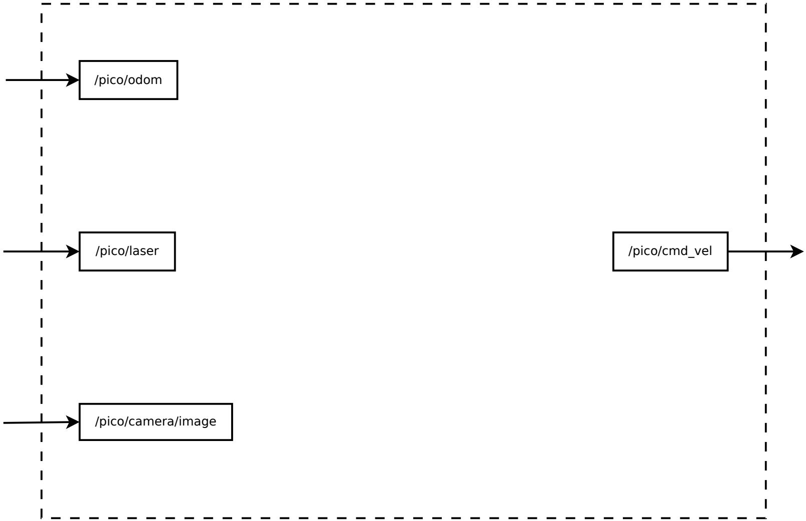
/pico/odom

/pico/laser

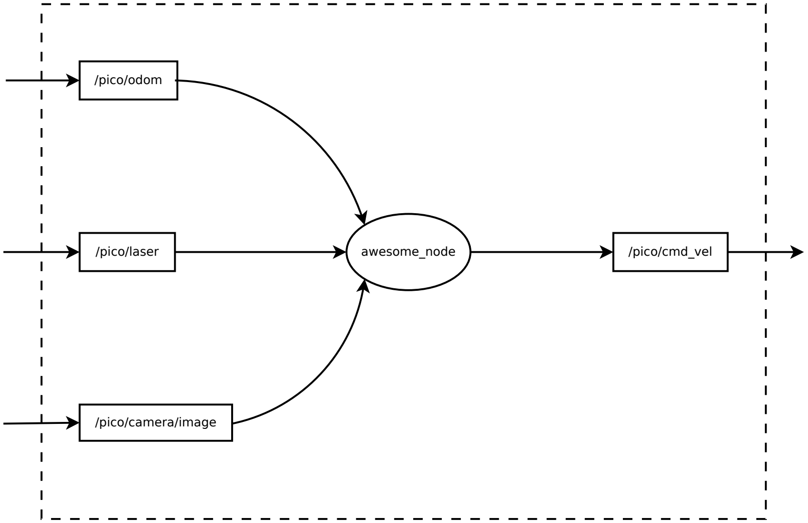
/pico/camera/image

/pico/cmd_vel

System Input / Output



Black Box


/pico/odom
/pico/laser
/pico/camera/image
awesome_node
/pico/cmd_vel

Black Box

Black Box

- Simple way of trying to fulfill the assignment:
  - ‘Hack into’ the current code

Black Box

- Simple way of trying to fulfill the assignment:
  - ‘Hack into’ the current code
  - Add features whenever you think of them

Black Box

- Simple way of trying to fulfill the assignment:
  - ‘Hack into’ the current code
  - Add features whenever you think of them
- May get you somewhere, but:

Black Box

- Simple way of trying to fulfill the assignment:
  - ‘Hack into’ the current code
  - Add features whenever you think of them
- May get you somewhere, but:
  - Hard to maintain
    - Found a bug: can be anywhere!

Black Box

- Simple way of trying to fulfill the assignment:
  - ‘Hack into’ the current code
  - Add features whenever you think of them
- May get you somewhere, but:
  - Hard to maintain
    - Found a bug: can be anywhere!
  - Hard to extend:
    - Adding new features may break old ones

Black Box

- Simple way of trying to fulfill the assignment:
  - ‘Hack into’ the current code
  - Add features whenever you think of them
- May get you somewhere, but:
  - Hard to maintain
    - Found a bug: can be anywhere!
  - Hard to extend:
    - Adding new features may break old ones
  - Teamwork becomes hard:
    - No clear division of work
    - Practical example: all editing the same file

Modular Approach

- Another way: modular approach

Modular Approach

- Another way: modular approach
- Modules are pieces of software that:

Modular Approach

- Another way: modular approach
- Modules are pieces of software that:
  - Have a clearly defined function

Modular Approach

- Another way: modular approach
- Modules are pieces of software that:
  - Have a clearly defined function
  - Have a clearly defined input and output
    - Interface

Modular Approach

- Another way: modular approach
- Modules are pieces of software that:
  - Have a clearly defined function
  - Have a clearly defined input and output
    - Interface
- The idea: break down system into modules:

Modular Approach

- Another way: modular approach
- Modules are pieces of software that:
  - Have a clearly defined function
  - Have a clearly defined input and output
    - Interface
- The idea: break down system into modules:
  - Modules talk to each other through interfaces

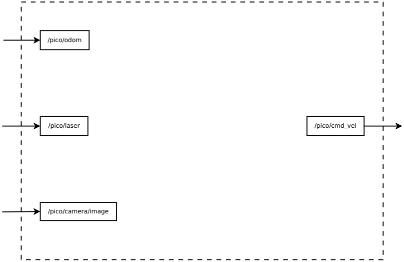
Modular Approach

- Another way: modular approach
- Modules are pieces of software that:
  - Have a clearly defined function
  - Have a clearly defined input and output
    - Interface
- The idea: break down system into modules:
  - Modules talk to each other through interfaces
  - Encapsulation of functionality and data

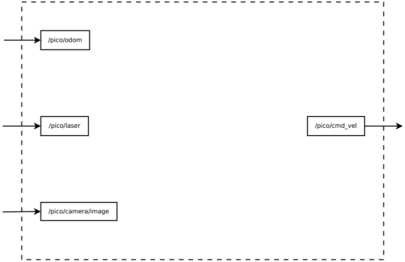
Modular Approach

- Another way: modular approach
- Modules are pieces of software that:
  - Have a clearly defined function
  - Have a clearly defined input and output
    - Interface
- The idea: break down system into modules:
  - Modules talk to each other through interfaces
  - Encapsulation of functionality and data
    - Easier to predict behavior
    - Easier to find errors
    - Easier to add new functionality

Identify Goal

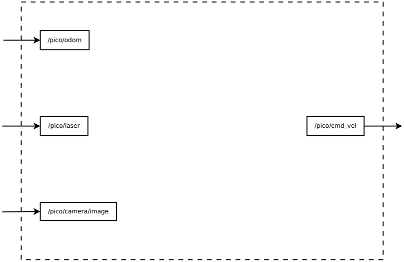


Identify Goal



- Let PICO navigate through a maze and find and go to the exit.

Identify Goal



- Let PICO navigate through a maze and find and go to the exit.
  - try to be as fast as possible
  - but avoid hitting obstacles at all cost!
  - ...

Break it down

- Let PICO navigate through a maze and find and go to the exit.

Break it down

- Let PICO navigate through a maze and find and go to the exit.
    - try to be as fast as possible
    - but avoid hitting obstacles at all cost!

Break it down

- Let PICO navigate through a maze and find and go to the exit.
    - try to be as fast as possible
    - but avoid hitting obstacles at all cost!
- Identify requirements and capabilities:

Break it down

- Let PICO navigate through a maze and find and go to the exit.
  - try to be as fast as possible
  - but avoid hitting obstacles at all cost!
- Identify requirements and capabilities:
  - Obstacle avoidance
  - Speed control
  - Recognize intersections
  - Drive straight through corridors
  - ...

Identify Modules

Identify Modules

- Requirements and capabilities:

Identify Modules

- Requirements and capabilities:
  - Obstacle avoidance
  - Speed control
  - Recognize intersections
  - Drive straight through corridors
  - ...

Identify Modules

- Requirements and capabilities:
  - Obstacle avoidance
  - Speed control
  - Recognize intersections
  - Drive straight through corridors
  - ...
- Identify Modules:

Identify Modules

- Requirements and capabilities:
  - Obstacle avoidance
  - Speed control
  - Recognize intersections
  - Drive straight through corridors
  - ...
- Identify Modules:
  - Wall Detector
  - Corner Detector
  - Corridor Navigation
  - ...

Identify Interfaces

- For each module, define its interface

Identify Interfaces

- For each module, define its interface
  - Wall Detector:
  - Corner Detector
  - Corridor Navigation

Identify Interfaces

- For each module, define its interface
    - Wall Detector:
        - input: laser data
        - output: lines (x, y, x2, y2)
    - Corner Detector
    - Corridor Navigation

Identify Interfaces

- For each module, define its interface
  - Wall Detector:
    - input: laser data
    - output: lines (x, y, x2, y2)
  - Corner Detector
    - input: laser data
    - output: corner points (x, y)
  - Corridor Navigation

Identify Interfaces

- For each module, define its interface
  - Wall Detector:
    - input: laser data
    - output: lines (x, y, x2, y2)
  - Corner Detector
    - input: laser data
    - output: corner points (x, y)
  - Corridor Navigation
    - input: wall lines
    - output: wanted velocity

Identify Interfaces

- For each module, define its interface
  - Wall Detector:
    - input: laser data
    - output: lines (x, y, x2, y2)
  - Corner Detector
    - input: laser data
    - output: corner points (x, y)
  - Corridor Navigation
    - input: wall lines
    - output: wanted velocity

- Make it as clear and complete as possible:

Identify Interfaces

- For each module, define its interface
  - Wall Detector:
    - input: laser data
    - output: lines (x, y, x2, y2)
  - Corner Detector
    - input: laser data
    - output: corner points (x, y)
  - Corridor Navigation
    - input: wall lines
    - output: wanted velocity

- Make it as clear and complete as possible:
  - Helps you identify missing modules

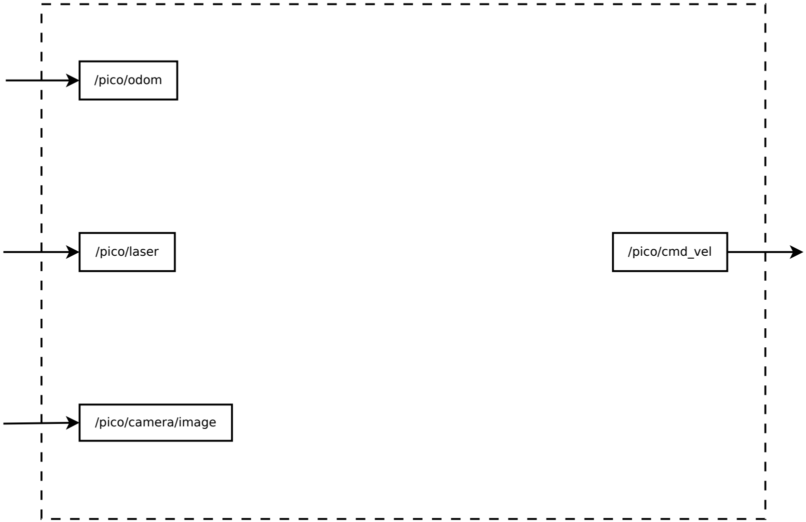
Identify Interfaces

- For each module, define its interface
  - Wall Detector:
    - input: laser data
    - output: lines (x, y, x2, y2)
  - Corner Detector
    - input: laser data
    - output: corner points (x, y)
  - Corridor Navigation
    - input: wall lines
    - output: wanted velocity

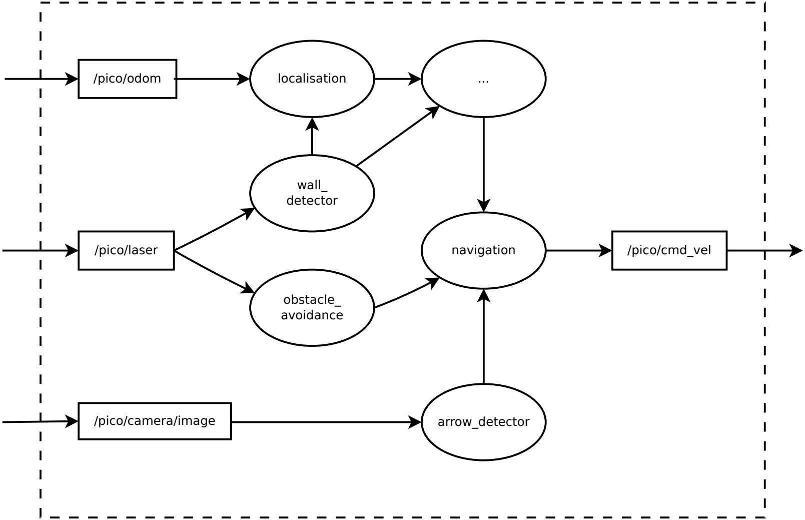
- Make it as clear and complete as possible:
  - Helps you identify missing modules
  - Teamwork becomes a lot easier

Draw the Picture

Draw the Picture

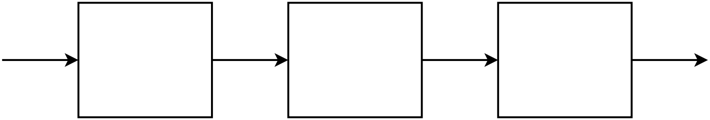


Draw the Picture

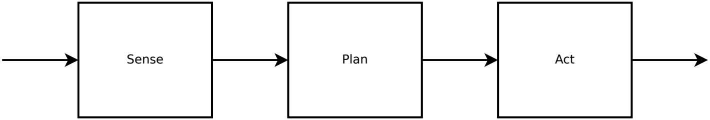


Combining Modules

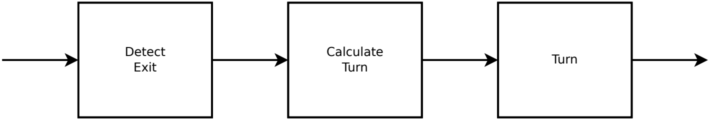
Combining Modules



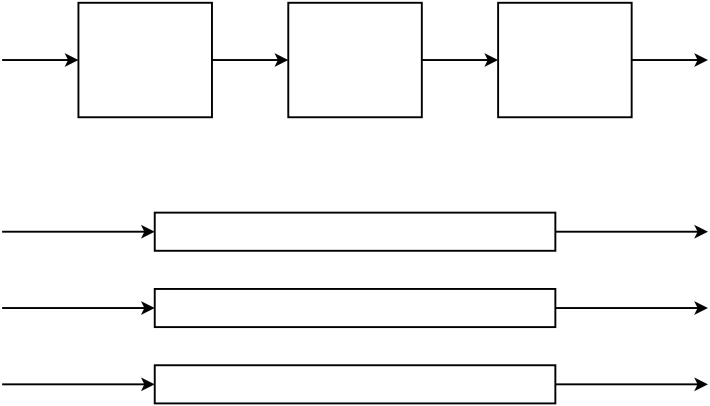
Combining Modules



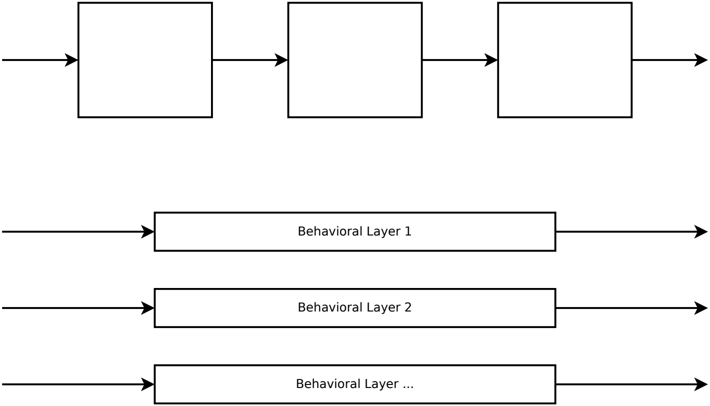
Combining Modules



Combining Modules



Combining Modules

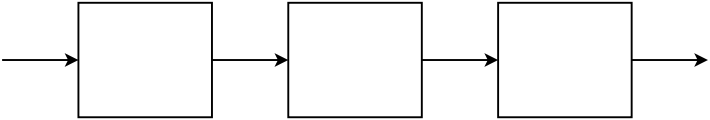


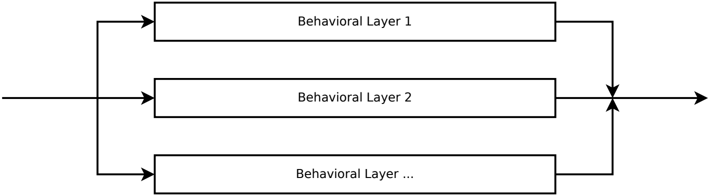
Behavioral Layer 1

Behavioral Layer 2

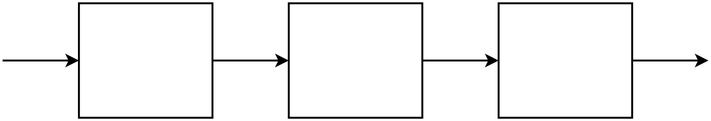
Behavioral Layer ...

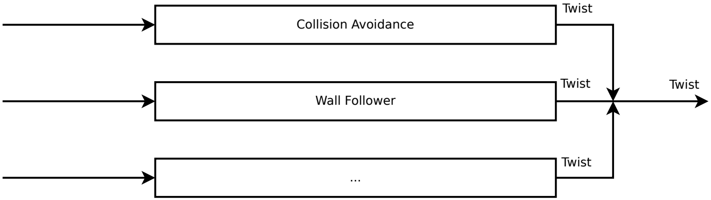
Combining Modules



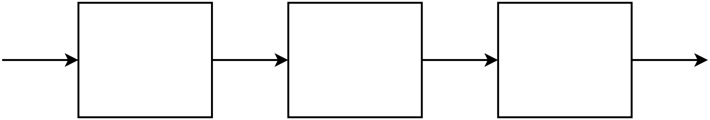


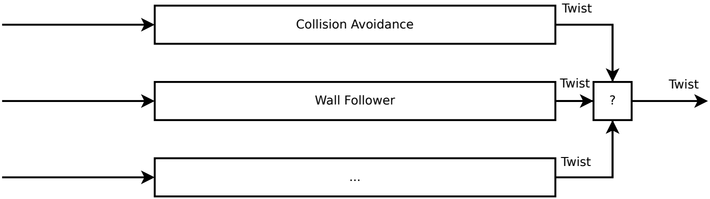
Combining Modules





Combining Modules





Modular Implementation

This is all nice and all, but pretty abstract.
How to actually implement this modular design?

Modular Implementation

- Use functions
    - Modularity within a process

Modular Implementation

- Use functions
  - Modularity within a process
  - Can split up over different files

Modular Implementation

- Use functions
  - Modularity within a process
  - Can split up over different files
- Use ROS nodes
  - Each node is a process

Modular Implementation

- Use functions
  - Modularity within a process
  - Can split up over different files
- Use ROS nodes
  - Each node is a process
- Use ROS packages
  - Way of grouping nodes together

Modular Implementation

- Use functions
  - Modularity within a process
  - Can split up over different files
- Use ROS nodes
  - Each node is a process
- Use ROS packages
  - Way of grouping nodes together
  - Can define dependencies

Modular Implementation

- Use functions
  - Modularity within a process
  - Can split up over different files
- Use ROS nodes
  - Each node is a process
- Use ROS packages
  - Way of grouping nodes together
  - Can define dependencies
- Use C++ Classes:

Modular Implementation

- Use functions
  - Modularity within a process
  - Can split up over different files
- Use ROS nodes
  - Each node is a process
- Use ROS packages
  - Way of grouping nodes together
  - Can define dependencies
- Use C++ Classes:
  - ’Data types’ that have encapsulated data and functionality

Modular Implementation

- Use functions
  - Modularity within a process
  - Can split up over different files
- Use ROS nodes
  - Each node is a process
- Use ROS packages
  - Way of grouping nodes together
  - Can define dependencies
- Use C++ Classes:
  - ’Data types’ that have encapsulated data and functionality
  - Object-oriented programming (OOP)

Modular Implementation

- Use functions
  - Modularity within a process
  - Can split up over different files
- Use ROS nodes
  - Each node is a process
- Use ROS packages
  - Way of grouping nodes together
  - Can define dependencies
- Use C++ Classes:
  - ’Data types’ that have encapsulated data and functionality
  - Object-oriented programming (OOP)
  - Don’t have to use it, but know that it is a very powerful paradigm

Modular Implementation

- Use functions
  - Modularity within a process
  - Can split up over different files
- Use ROS nodes
  - Each node is a process
- Use ROS packages
  - Way of grouping nodes together
  - Can define dependencies
- Use C++ Classes:
  - ’Data types’ that have encapsulated data and functionality
  - Object-oriented programming (OOP)
  - Don’t have to use it, but know that it is a very powerful paradigm
  - Examples: std::vector, ros::Subscriber, ros::NodeHandle

Modular Approach Using Functions

- Split up in functions

Modular Approach Using Functions

- Split up in functions
- Each function implements a re-usable, ’small’ but complete calculation, action, ...

Modular Approach Using Functions

- Split up in functions
- Each function implements a re-usable, ’small’ but complete calculation, action, ...
- The function arguments and return types are the interfaces

Modular Approach Using Functions

- Split up in functions
- Each function implements a re-usable, ’small’ but complete calculation, action, ...
- The function arguments and return types are the interfaces
    - Avoid global variables as much as possible!

Modular Approach Using Functions

- Split up in functions
- Each function implements a re-usable, ’small’ but complete calculation, action, ...
- The function arguments and return types are the interfaces
  - Avoid global variables as much as possible!

```
while(ros::ok()) {
    ros::spinOnce();  // get sensor data
    ... walls = detectWalls(laser_data, ...);
    ... corners = detectCorners(laser_data, ...);
    ... vel = navToGoal(walls, corners, ...);
    ... vel_safe = avoidCollision(vel, laser_data, ...);
    publishVel(vel_safe);
}
```

Modular Approach Using Functions

- Split up in functions
- Each function implements a re-usable, ’small’ but complete calculation, action, ...
- The function arguments and return types are the interfaces
    - Avoid global variables as much as possible!

```
while(ros::ok()) {
    ros::spinOnce();  // get sensor data
    ... walls = detectWalls(laser_data, ...);
    ... corners = detectCorners(laser_data, ...);
    ... vel = navToGoal(walls, corners, ...);
    ... vel_safe = avoidCollision(vel, laser_data, ...);
    publishVel(vel_safe);
}
```

- *Whoah, it’s pretty clear what happens!*

Modular Approach Using Functions

- Split up in functions
- Each function implements a re-usable, ’small’ but complete calculation, action, ...
- The function arguments and return types are the interfaces
    - Avoid global variables as much as possible!

```
while(ros::ok()) {
    ros::spinOnce();  // get sensor data
    ... walls = detectWalls(laser_data, ...);
    ... corners = detectCorners(laser_data, ...);
    ... vel = navToGoal(walls, corners, ...);
    ... vel_safe = avoidCollision(vel, laser_data, ...);
    publishVel(vel_safe);
}
```

- *Whoah, it’s pretty clear what happens!*
- Can even split up in separate files

Modular Approach Using Nodes

- One step further: split up in ROS nodes

Modular Approach Using Nodes

- One step further: split up in ROS nodes
- Each node has own main function and main loop

Modular Approach Using Nodes

- One step further: split up in ROS nodes
- Each node has own main function and main loop
- Interface: ROS messages

Modular Approach Using Nodes

- One step further: split up in ROS nodes
- Each node has own main function and main loop
- Interface: ROS messages

- Advantages:
  - Clear separation of data

Modular Approach Using Nodes

- One step further: split up in ROS nodes
- Each node has own main function and main loop
- Interface: ROS messages
- Advantages:
  - Clear separation of data
  - Nodes are processes: run parallel

Modular Approach Using Nodes

- One step further: split up in ROS nodes
- Each node has own main function and main loop
- Interface: ROS messages

- Advantages:
  - Clear separation of data
  - Nodes are processes: run parallel
  - Can inspect communication at run time
    - `rostopic echo ...`
    - Visualization

Modular Approach Using Nodes

- One step further: split up in ROS nodes
- Each node has own main function and main loop
- Interface: ROS messages

- Advantages:
  - Clear separation of data
  - Nodes are processes: run parallel
  - Can inspect communication at run time
    - `rostopic echo ...`
    - Visualization
- Disadvantage:
  - Overhead of using Subscribers and Publishers

Modular Approach Using Nodes

- One step further: split up in ROS nodes
- Each node has own main function and main loop
- Interface: ROS messages

Modular Approach Using Nodes

- One step further: split up in ROS nodes
- Each node has own main function and main loop
- Interface: ROS messages

Directory Structure

```
src/
    wall_detector.cpp
    corner_detector.cpp
    ...
```

Modular Approach Using Nodes

- One step further: split up in ROS nodes
- Each node has own main function and main loop
- Interface: ROS messages

Directory Structure

```
src/
    wall_detector.cpp
    corner_detector.cpp
    ...
```

CmakeLists.txt

```
rosbuild_add_executable(wall_detector src/wall_detector.cpp)
rosbuild_add_executable(corner_detector src/corner_detector.cpp)
...
```

Recap

Recap

- Using ROS in C++

Recap

- Using ROS in C++
- PICO Safe Drive Example

Recap

- Using ROS in C++
- PICO Safe Drive Example
- Modular Programming

Recap

- Using ROS in C++
- PICO Safe Drive Example
- Modular Programming
    - Determine goal
    - Identify requirements
    - Identify modules
    - Determine interfaces

Recap

- Using ROS in C++
- PICO Safe Drive Example
- Modular Programming
  - Determine goal
  - Identify requirements
  - Identify modules
  - Determine interfaces
- Modular Implementation

Recap

- Using ROS in C++
- PICO Safe Drive Example
- Modular Programming
  - Determine goal
  - Identify requirements
  - Identify modules
  - Determine interfaces
- Modular Implementation
  - Using functions
  - Using ROS Nodes

Reminders

- Corridor Competition: Next week!
  - Location: GEM-N, Soccer Field
  - Time: 13.45

Reminders

- Corridor Competition: Next week!
  - Location: GEM-N, Soccer Field
  - Time: 13.45
- Weekly tutor appointments

Reminders

- Corridor Competition: Next week!
  - Location: GEM-N, Soccer Field
  - Time: 13.45
- Weekly tutor appointments
- Test schedule

Reminders

- Corridor Competition: Next week!
  - Location: GEM-N, Soccer Field
  - Time: 13.45
- Weekly tutor appointments
- Test schedule
- Wiki page:
  - Document every decision
  - Software design
  - Planning
  - In general: explain how you are going to tackle the problem!