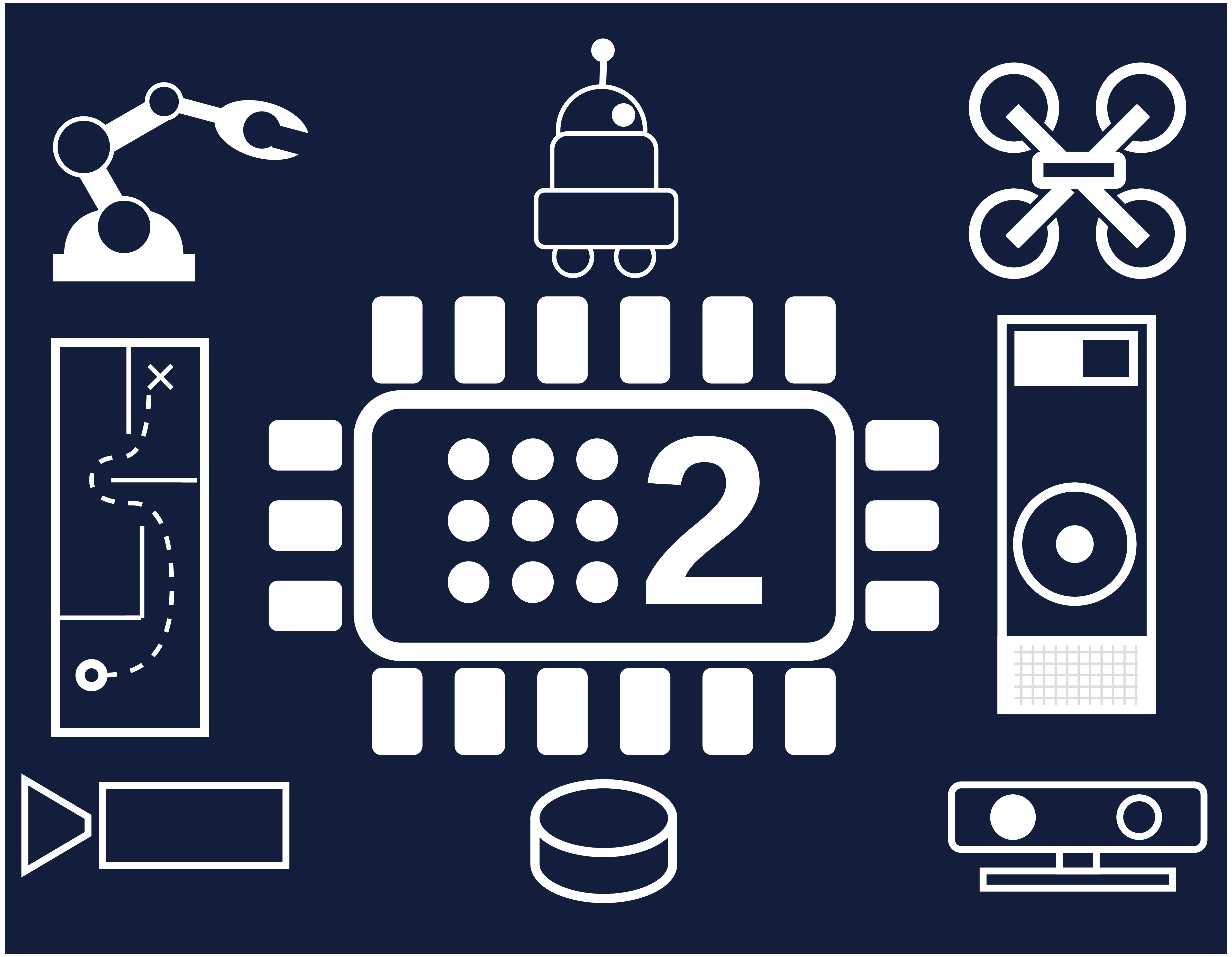


A Concise Introduction to Robot Programming in ROS2

Prof. Dr. Francisco Martín Rico

Chapter 3: Avoiding Obstacles with FSMs

francisco.rico@urjc.es
2022 @fmrico

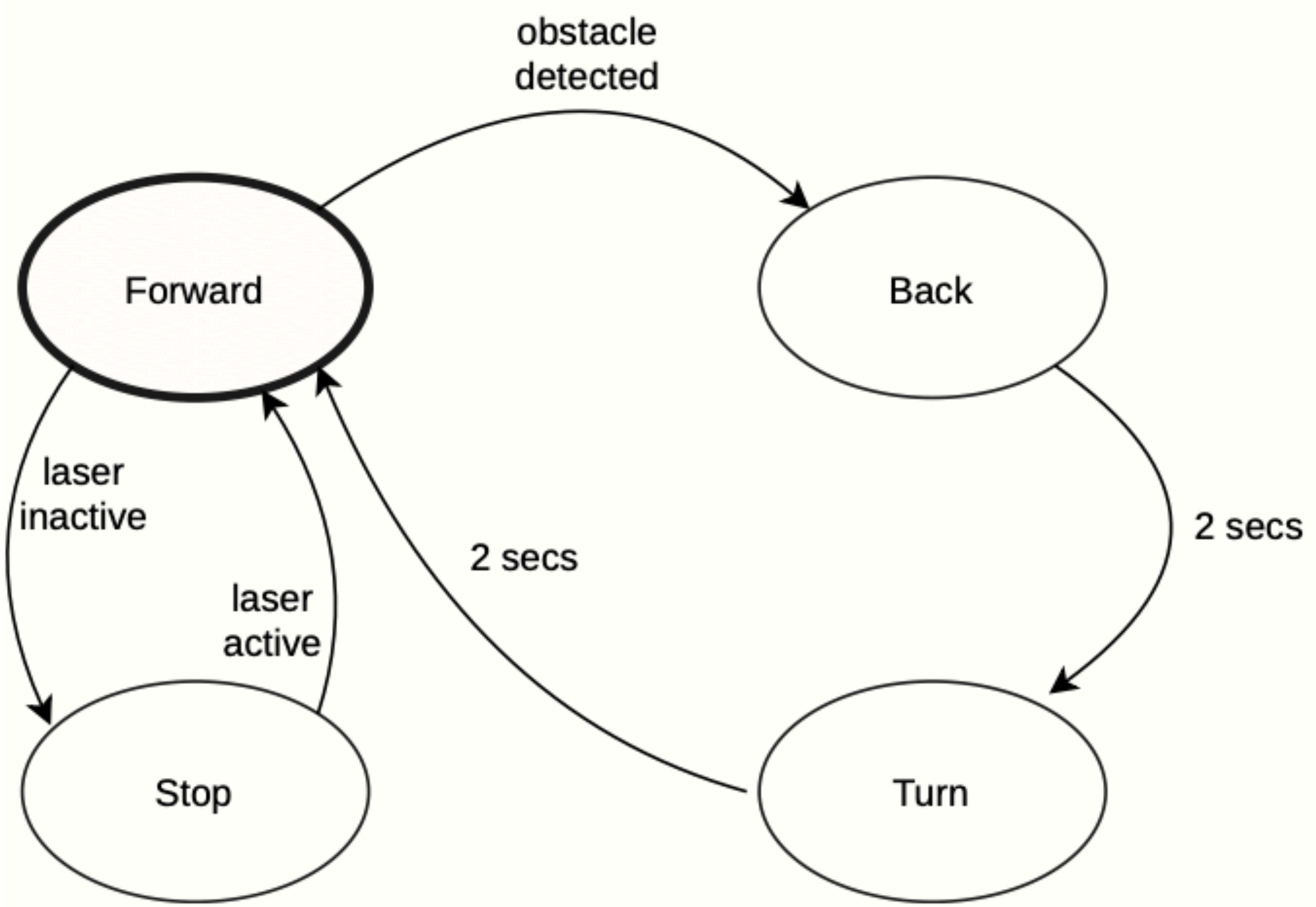
Universidad Rey Juan Carlos

Intelligent Robotics Lab

Finite State Machines (FSMs)

- It a simple hay to encode behaviors
- States and transitions

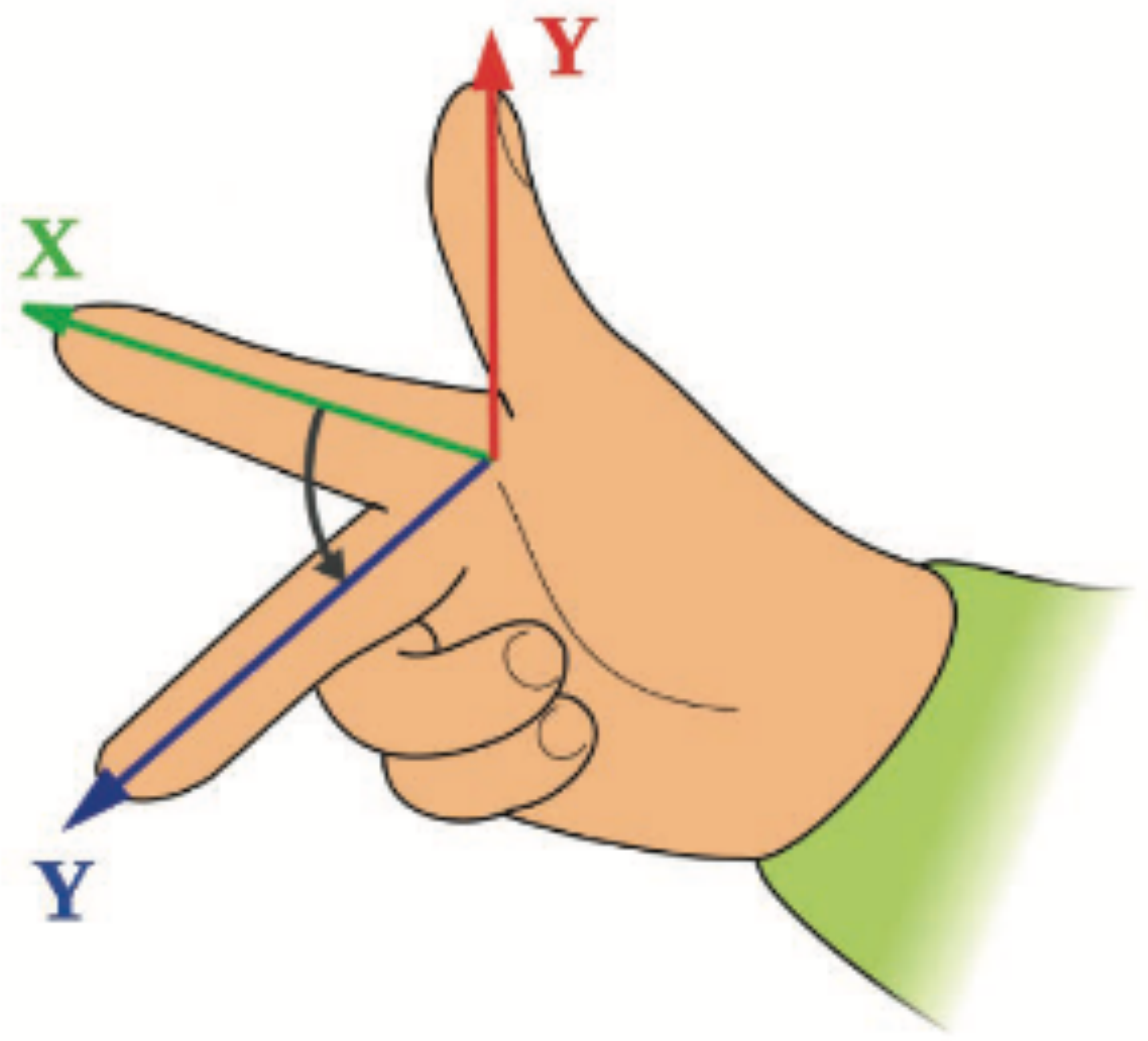
- **Goal**: a Bump&Go behavior
- New concepts:
  - **FSMs**
  - **Laser Processing**
  - **Robot Motion**
  - **Nodes in C++ and Python**

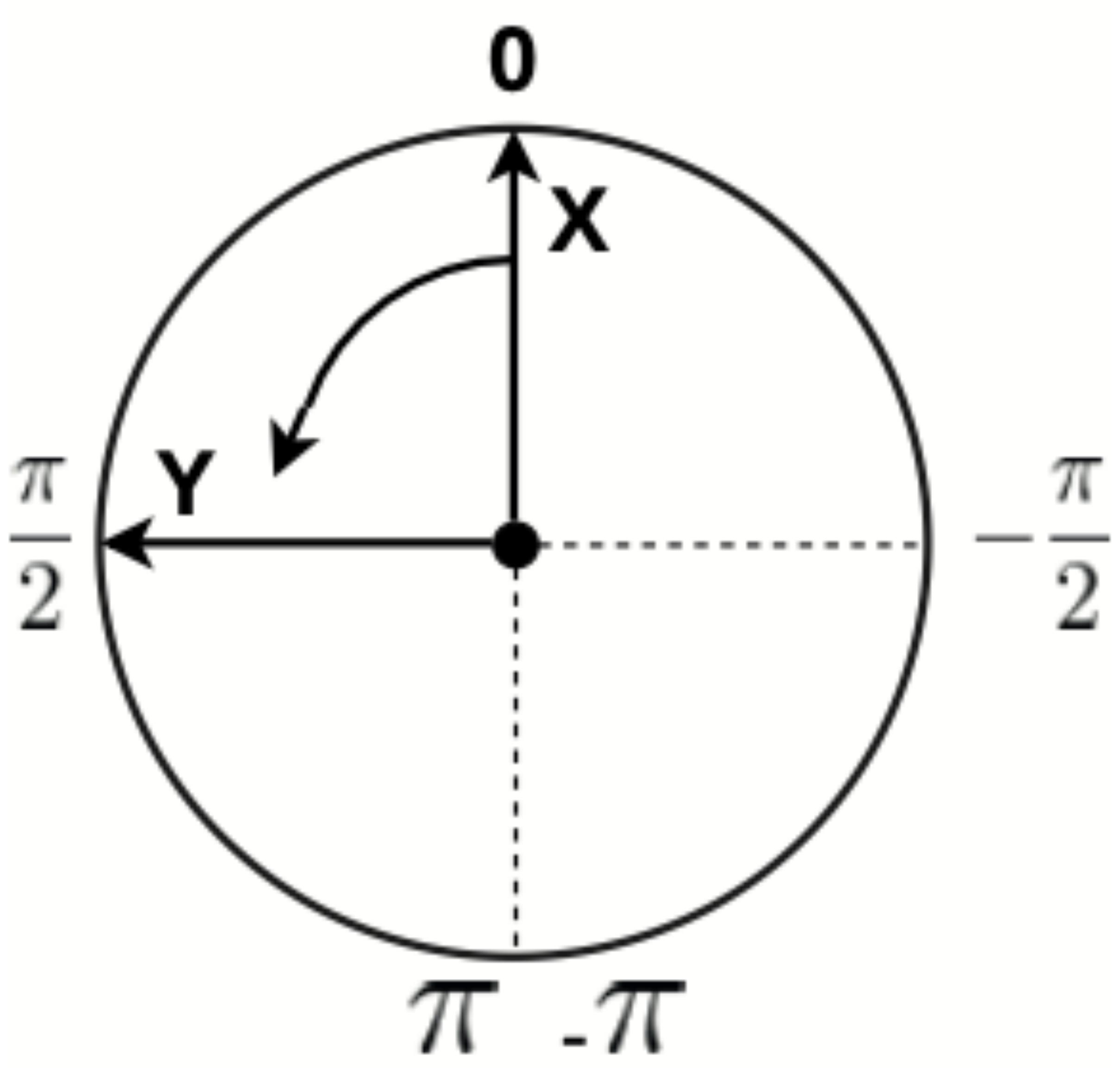


Perception and Actuation Models

Conventions

- International System of Measurements (SI)





Francisco Martín Rico francisco.rico@urjc.es @fmrico

Perception and Actuation Models

Conventions

- Standard interfaces for sensors and actuators

```
$ ros2 interface show /sensor_msgs/msg/LaserScan

# Single scan from a planar laser range-finder
#
# If you have another ranging device with different behavior (e.g. a sonar
# array), please find or create a different message, since applications
# will make fairly laser-specific assumptions about this data

std_msgs/Header header  # timestamp in the header is the acquisition time of
                        # the first ray in the scan.
                        #
                        # in frame frame_id, angles are measured around
                        # the positive Z axis (counterclockwise, if Z is up)
                        # with zero angle being forward along the x axis

float32 angle_min       # start angle of the scan [rad]
float32 angle_max       # end angle of the scan [rad]
float32 angle_increment # angular distance between measurements [rad]

float32 time_increment  # time between measurements [seconds] - if your scanner
                        # is moving, this will be used in interpolating pos
                        # of 3d points
float32 scan_time       # time between scans [seconds]

float32 range_min       # minimum range value [m]
float32 range_max       # maximum range value [m]

float32[] ranges        # range data [m]
                        # (Note: values < range_min or > range_max should be
                        # discarded)
float32[] intensities   # intensity data [device-specific units]. If your
                        # device does not provide intensities, please leave
                        # the array empty.
```

```
$ ros2 topic echo /scan_raw --no-arr
---
header:
  stamp:
    sec: 11071
    nanosec: 445000000
  frame_id: base_laser_link
angle_min: -1.9198600053787231
angle_max: 1.9198600053787231
angle_increment: 0.005774015095084906
time_increment: 0.0
scan_time: 0.0
range_min: 0.05000000074505806
range_max: 25.0
ranges: '<sequence type: float, length: 666>'
intensities: '<sequence type: float, length: 666>'
---
```

Perception and Actuation Models

Conventions

- Standard interfaces for sensors and actuators

```
$ ros2 interface show /sensor_msgs/msg/LaserScan

# Single scan from a planar laser range-finder
#
# If you have another ranging device with different behavior (e.g. a sonar
# array), please find or create a different message, since applications
# will make fairly laser-specific assumptions about this data

std_msgs/Header header  # timestamp in the header is the acquisition time of
                        # the first ray in the scan.
                        #
                        # in frame frame_id, angles are measured around
                        # the positive Z axis (counterclockwise, if Z is up)
                        # with zero angle being forward along the x axis

float32 angle_min       # start angle of the scan [rad]
float32 angle_max       # end angle of the scan [rad]
float32 angle_increment # angular distance between measurements [rad]

float32 time_increment  # time between measurements [seconds] - if your scanner
                        # is moving, this will be used in interpolating pos
                        # of 3d points
float32 scan_time       # time between scans [seconds]

float32 range_min       # minimum range value [m]
float32 range_max       # maximum range value [m]

float32[] ranges        # range data [m]
                        # (Note: values < range_min or > range_max should be
                        # discarded)
float32[] intensities   # intensity data [device-specific units]. If your
                        # device does not provide intensities, please leave
                        # the array empty.
```

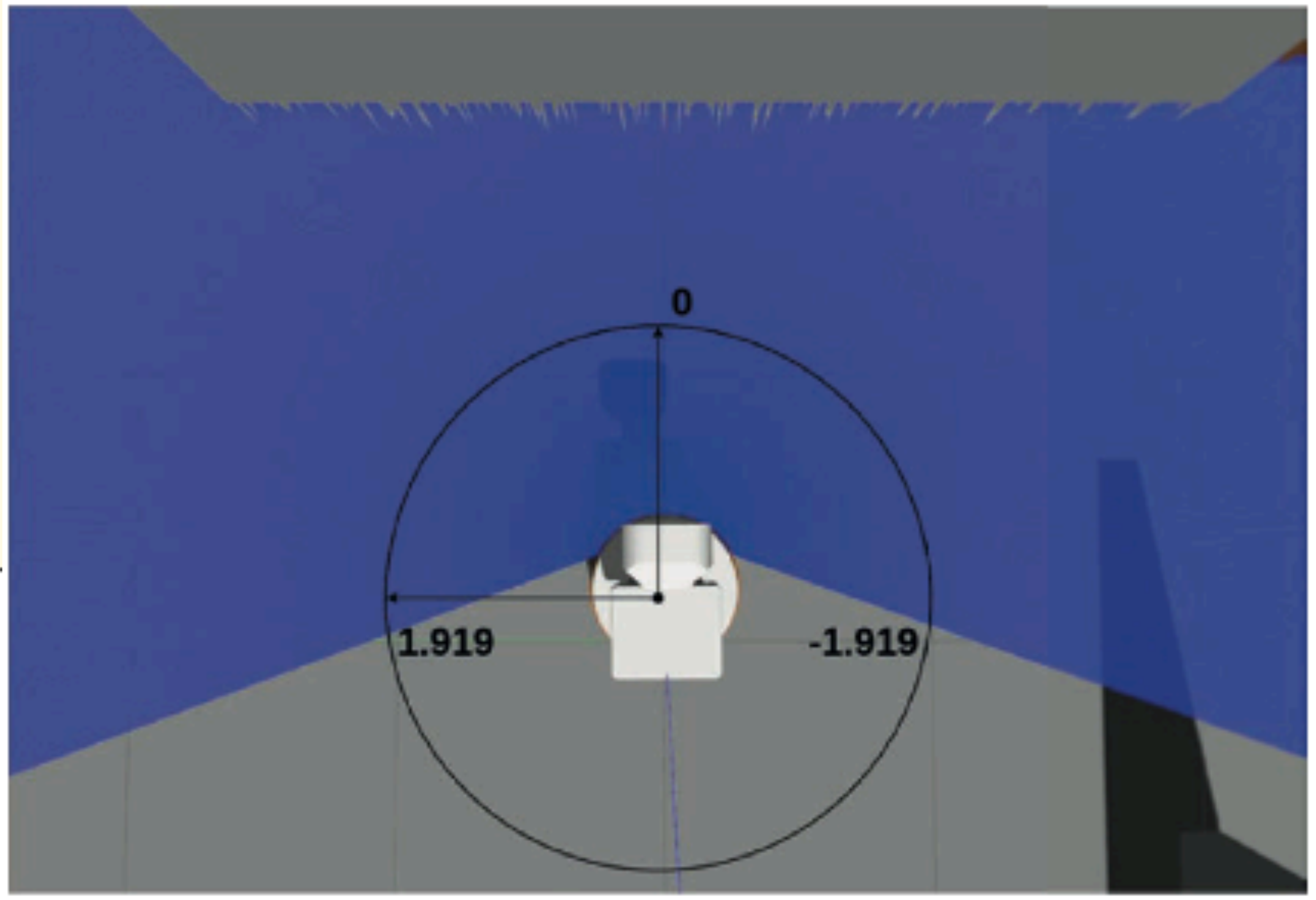

Francisco Martín Rico francisco.rico@urjc.es @fmrico

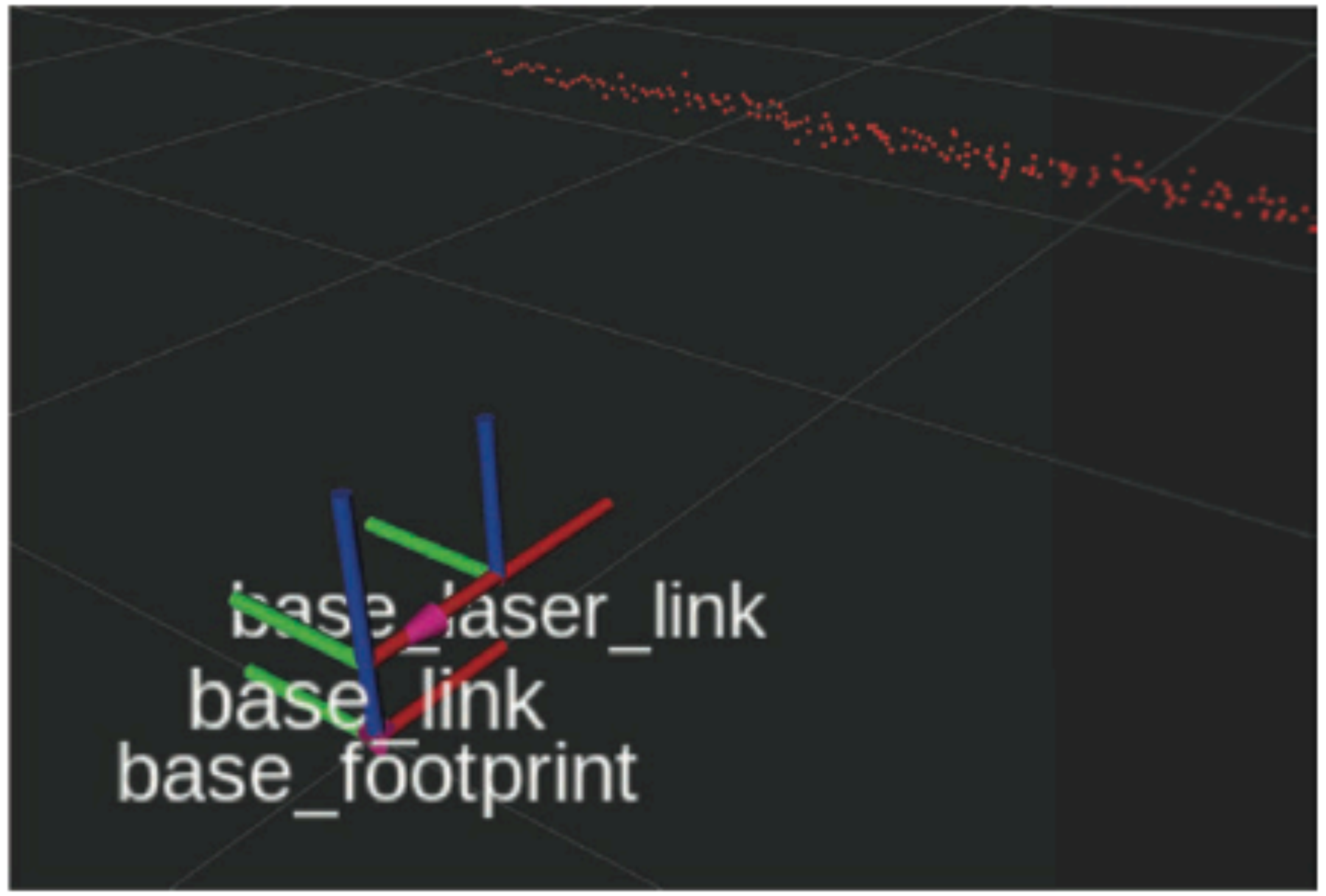
Perception and Actuation Models

Conventions

- Standard interfaces for sensors and actuators

```
$ ros2 topic echo /scan_raw --no-arr
---
header:
  stamp:
    sec: 11071
    nanosec: 445000000
  frame_id: base_laser_link
angle_min: -1.9198600053787231
angle_max: 1.9198600053787231
angle_increment: 0.005774015095084906
time_increment: 0.0
scan_time: 0.0
range_min: 0.05000000074505806
range_max: 25.0
ranges: '<sequence type: float, length: 666>'
intensities: '<sequence type: float, length: 666>'
---
```





Perception and Actuation Models

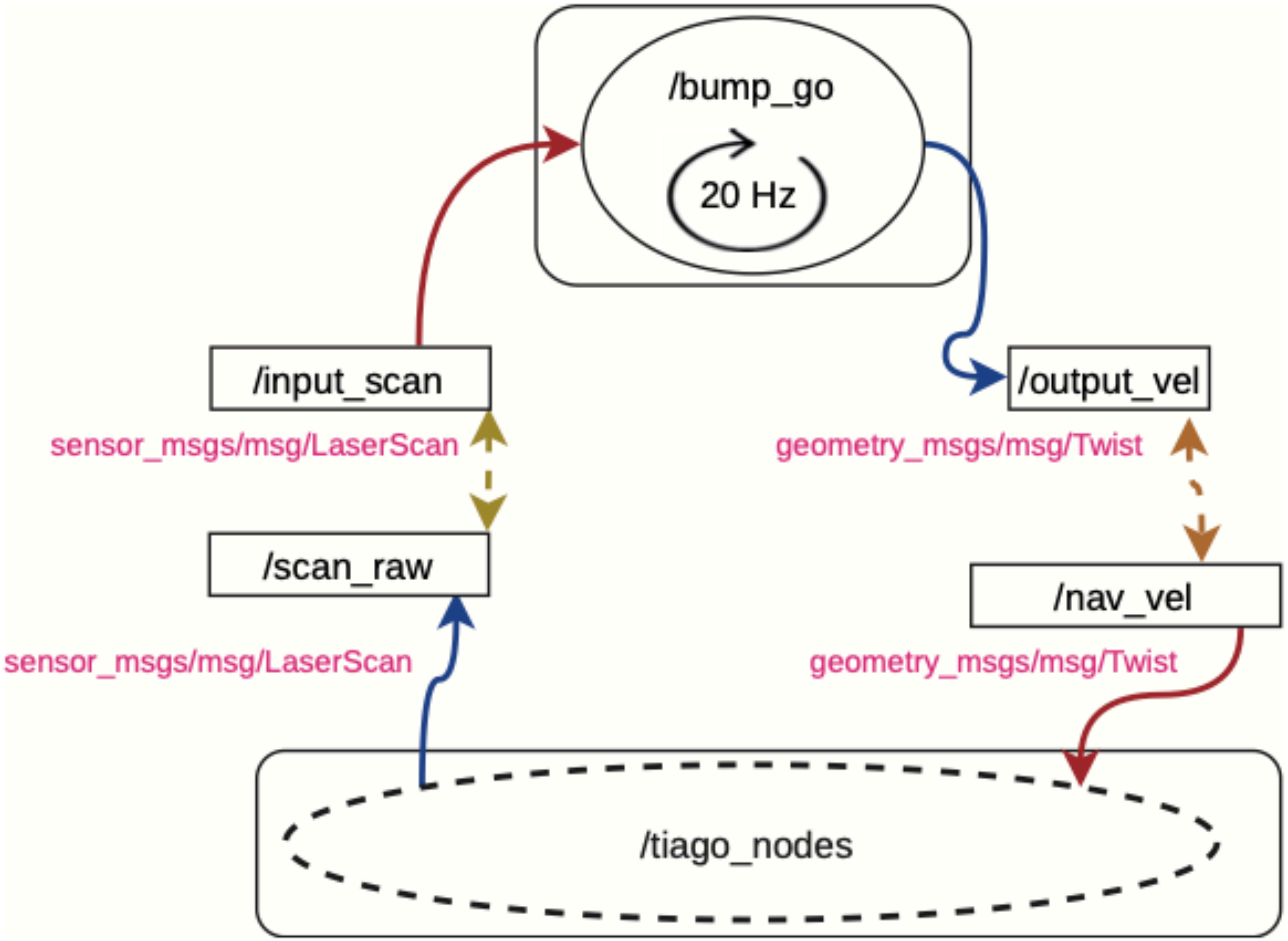
Conventions

- Standard interfaces for sensors and actuators

```
$ ros2 interface show geometry_msgs/msg/Twist

Vector3 linear
Vector3 angular


$ ros2 interface show geometry_msgs/msg/Vector3

float64 x
float64 y
float64 z
```

Intelligent Robotics Lab

Computation Graph



Bump&Go in C++

Package content

```
br2_fsm_bumpgo_cpp
├── CMakeLists.txt
├── include
│   └── br2_fsm_bumpgo_cpp
│       └── BumpGoNode.hpp
├── launch
│   └── bump_and_go.launch.py
├── package.xml
└── src
    ├── br2_fsm_bumpgo_cpp
    │   └── BumpGoNode.cpp
    └── bumpgo_main.cpp
```

Bump&Go in C++

Execution Control

```
class BumpGoNode : public rclcpp::Node
{
  ...
private:
  void scan_callback(sensor_msgs::msg::LaserScan::UniquePtr msg);
  void control_cycle();

  rclcpp::Publisher<geometry_msgs::msg::Twist>::SharedPtr vel_pub_;
  rclcpp::Subscription<sensor_msgs::msg::LaserScan>::SharedPtr scan_sub_;
  rclcpp::TimerBase::SharedPtr timer_;

  sensor_msgs::msg::LaserScan::UniquePtr last_scan_;
};
```

```
1. void scan_callback(const sensor_msgs::msg::LaserScan & msg);
2. void scan_callback(sensor_msgs::msg::LaserScan::UniquePtr msg);
3. void scan_callback(sensor_msgs::msg::LaserScan::SharedConstPtr msg);
4. void scan_callback(const sensor_msgs::msg::LaserScan::SharedConstPtr & msg);
5. void scan_callback(sensor_msgs::msg::LaserScan::SharedPtr msg);
```

Intelligent Robotics Lab

Bump&Go in C++

Subscriptions and publications

```
BumpGoNode::BumpGoNode()
: Node("bump_go")
{
  scan_sub_ = create_subscription<sensor_msgs::msg::LaserScan>(
    "input_scan", rclcpp::SensorDataQoS(),
    std::bind(&BumpGoNode::scan_callback, this, _1));

  vel_pub_ = create_publisher<geometry_msgs::msg::Twist>("output_vel", 10);
  timer_ = create_wall_timer(50ms, std::bind(&BumpGoNode::control_cycle, this));
}

void
BumpGoNode::scan_callback(sensor_msgs::msg::LaserScan::UniquePtr msg)
{
  last_scan_ = std::move(msg);
}

void
BumpGoNode::control_cycle()
{
  // Do nothing until the first sensor read
  if (last_scan_ == nullptr)
    return;

    vel_pub_->publish(...);
}
```

Bump&Go in C++

Implementing a FSM

```
class BumpGoNode : public rclcpp::Node
{
  ...
private:
  void control_cycle();

  static const int FORWARD = 0;
  static const int BACK = 1;
  static const int TURN = 2;
  static const int STOP = 3;
  int state_;
  rclcpp::Time state_ts_;
};
```

```
bool
BumpGoNode::check_forward_2_back()
{
  // going forward when deteting an obstacle
  // at 0.5 meters with the front laser read
  size_t pos = last_scan_->ranges.size() / 2;
  return last_scan_->ranges[pos] < OBSTACLE_DISTANCE;
}
```

```
bool
BumpGoNode::check_forward_2_stop()
{
  // Stop if no sensor readings for 1 second
  auto elapsed = now() - rclcpp::Time(last_scan_->header.stamp);
  return elapsed > SCAN_TIMEOUT;
}
```

```
bool
BumpGoNode::check_back_2_turn()
{
  // Going back for 2 seconds
  return (now() - state_ts_) > BACKING_TIME;
}
```

```
BumpGoNode::BumpGoNode()
: Node("bump_go"),
  state_(FORWARD)
{
  ...
  state_ts_ = now();
}

void
BumpGoNode::control_cycle()
{
  switch (state_) {
    case FORWARD:

      // Do whatever you should do in this state.
      // In this case, set the output speed.

      // Checking the condition to go to another state in the next iteration
      if (check_forward_2_stop())
        go_state(STOP);
      if (check_forward_2_back())
        go_state(BACK);

      break;
      ...
  }
}

void
BumpGoNode::go_state(int new_state)
{
  state_ = new_state;
  state_ts_ = now();
}
```


Francisco Martín Rico francisco.rico@urjc.es @fmrico

Bump&Go in C++

Running the code

```cpp
int main(int argc, char * argv[])
{
  rclcpp::init(argc, argv);

  auto bumpgo_node = std::make_shared<br2_fsm_bumpgo_cpp::BumpGoNode>();
  rclcpp::spin(bumpgo_node);

  rclcpp::shutdown();

  return 0;
}
```

```
$ ros2 launch br2_tiago sim.launch.py
```

```
$ ros2 run br2_fsm_bumpgo_cpp bumpgo --ros-args -r output_vel:=/nav_vel -r
input_scan:=/scan_raw -p use_sim_time:=true
```

Intelligent Robotics Lab

Bump&Go in C++

Running the code

```
int main(int argc, char * argv[])
{
  rclcpp::init(argc, argv);

  auto bumpgo_node = std::make_shared<br2_fsm_bumpgo_cpp::BumpGoNode>();
  rclcpp::spin(bumpgo_node);

  rclcpp::shutdown();

  return 0;
}
```

```
$ ros2 launch br2_tiago sim.launch.py
```

launch/bump_and_go.launch.py

```
bumpgo_cmd = Node(package='br2_fsm_bumpgo_cpp',
    executable='bumpgo',
    output='screen',
    parameters=[{
      'use_sim_time': True
    }],
    remappings=[
      ('input_scan', '/scan_raw'),
      ('output_vel', '/nav_vel')
    ])
```

```
$ ros2 launch br2_fsm_bumpgo_cpp bump_and_go.launch.py
```


Francisco Martín Rico francisco.rico@urjc.es @fmrico

Bump&Go in Python

Package content

```
br2_fsm_bumpgo_py
├── br2_fsm_bumpgo_py
│   ├── bump_go_main.py
│   └── __init__.py
├── launch
│   └── bump_and_go.launch.py
├── package.xml
├── resource
│   └── br2_fsm_bumpgo_py
├── setup.cfg
├── setup.py
└── test
    ├── test_copyright.py
    ├── test_flake8.py
    └── test_pep257.py
```

Bump&Go in Python

Execution Control

```python
from rclpy.duration import Duration
from rclpy.node import Node
from rclpy.qos import qos_profile_sensor_data
from rclpy.time import Time

from geometry_msgs.msg import Twist
from sensor_msgs.msg import LaserScan


class BumpGoNode(Node):
    def __init__(self):
        super().__init__('bump_go')

        ...

        self.last_scan = None
        self.scan_sub = self.create_subscription(
            LaserScan,
            'input_scan',
            self.scan_callback,
            qos_profile_sensor_data)

        self.vel_pub = self.create_publisher(Twist, 'output_vel', 10)
        self.timer = self.create_timer(0.05, self.control_cycle)

    def scan_callback(self, msg):
        self.last_scan = msg
```

```python
    def scan_callback(self, msg):
        self.last_scan = msg

    def control_cycle(self):
        if self.last_scan is None:
            return

        out_vel = Twist()

        # FSM

        self.vel_pub.publish(out_vel)

def main(args=None):
    rclpy.init(args=args)

    bump_go_node = BumpGoNode()

    rclpy.spin(bump_go_node)

    bump_go_node.destroy_node()
    rclpy.shutdown()

if __name__ == '__main__':
    main()
```

Bump&Go in Python

Implementing a FSM

```python
class BumpGoNode(Node):
    def __init__(self):
        super().__init__('bump_go')

        self.FORWARD = 0
        self.BACK = 1
        self.TURN = 2
        self.STOP = 3
        self.state = self.FORWARD
        self.state_ts = self.get_clock().now()

    def control_cycle(self):

        if self.state == self.FORWARD:
          out_vel.linear.x = self.SPEED_LINEAR

          if self.check_forward_2_stop():
            self.go_state(self.STOP)
          if self.check_forward_2_back():
            self.go_state(self.BACK)

        self.vel_pub.publish(out_vel)

    def go_state(self, new_state):
        self.state = new_state
        self.state_ts = self.get_clock().now()
```

```python
def check_forward_2_back(self):
    pos = round(len(self.last_scan.ranges) / 2)
    return self.last_scan.ranges[pos] < self.OBSTACLE_DISTANCE

def check_forward_2_stop(self):
    elapsed = self.get_clock().now() - Time.from_msg(self.last_scan.header.stamp)
    return elapsed > Duration(seconds=self.SCAN_TIMEOUT)

def check_back_2_turn(self):
    elapsed = self.get_clock().now() - self.state_ts
    return elapsed > Duration(seconds=self.BACKING_TIME)
```


Francisco Martín Rico francisco.rico@urjc.es @fmrico

Bump&Go in Python

Running the code

- setup.py

```
import os
from glob import glob

from setuptools import setup

package_name = 'br2_fsm_bumpgo_py'

setup(
    name=package_name,
    version='0.0.0',
    packages=[package_name],
    data_files=[
        ('share/ament_index/resource_index/packages',
            ['resource/' + package_name]),
        ('share/' + package_name, ['package.xml']),
        (os.path.join('share', package_name, 'launch'), glob('launch/*.launch.py'))
    ],
    install_requires=['setuptools'],
    zip_safe=True,
    maintainer='johndoe',
    maintainer_email='john.doe@evilrobot.com',
    description='BumpGo in Python package',
    license='Apache 2.0',
    tests_require=['pytest'],
    entry_points={
        'console_scripts': [
          'bump_go_main = br2_fsm_bumpgo_py.bump_go_main:main'
        ],
    },
)
```


Francisco Martín Rico francisco.rico@urjc.es @fmrico

Bump&Go in Python

Running the code

```
$ colcon build --symlink-install
```

```
$ ros2 launch br2_tiago sim.launch.py
```

```
$ ros2 run br2_fsm_bumpgo_py bump_go_main --ros-args -r output_vel:=/nav_vel -r
input_scan:=/scan_raw -p use_sim_time:=true
```

Bump&Go in Python

Running the code

```
$ colcon build --symlink-install
```

```
$ ros2 launch br2_tiago sim.launch.py
```

```
$ ros2 launch br2_fsm_bumpgo_py bump_and_go.launch.py
```

Intelligent Robotics Lab

Proposed Exercises

1. Modify the *Bump and Go* project so that the robot perceives an obstacle in front, on its left and right diagonal. Instead of always turning to the same side, it turns to the side with no obstacle.

2. Modify the *Bump and Go* project so that the robot turns exactly to the angle with no obstacles or the more far perceived obstacle. Try two approaches:
   - Open-loop: Calculate before turning time and speed to turn.
   - Closed-loop: Turns until a clear space in front is detected.

Intelligent Robotics Lab