Course of

Robot Programming with ROS 2

Day 1

4. Exercise: Bump&Go FSM

Robot Operating System

ikerlan

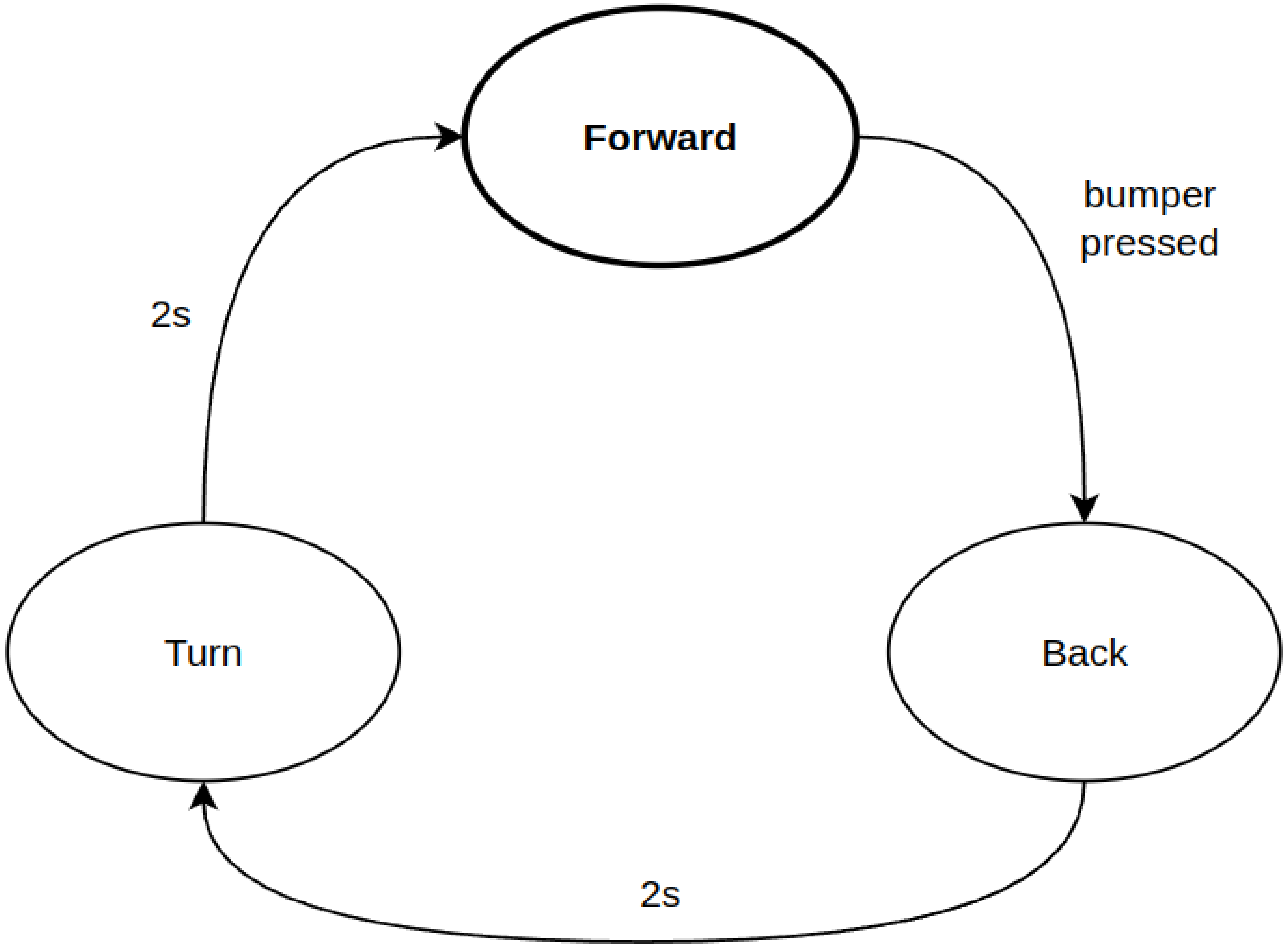
Universidad Rey Juan Carlos

Intelligent Robotics Lab

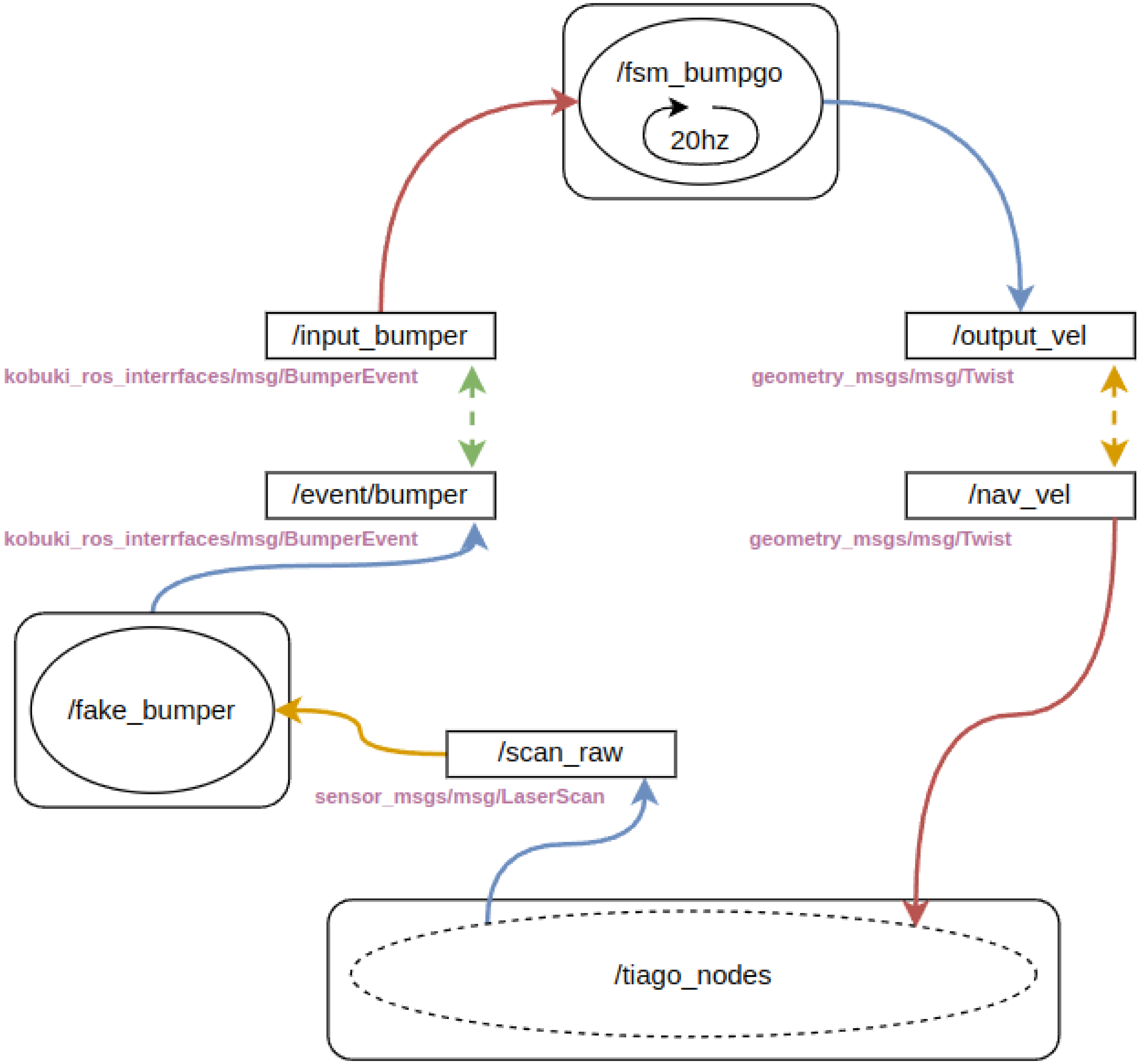
Finite State Machines (FSMs)

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

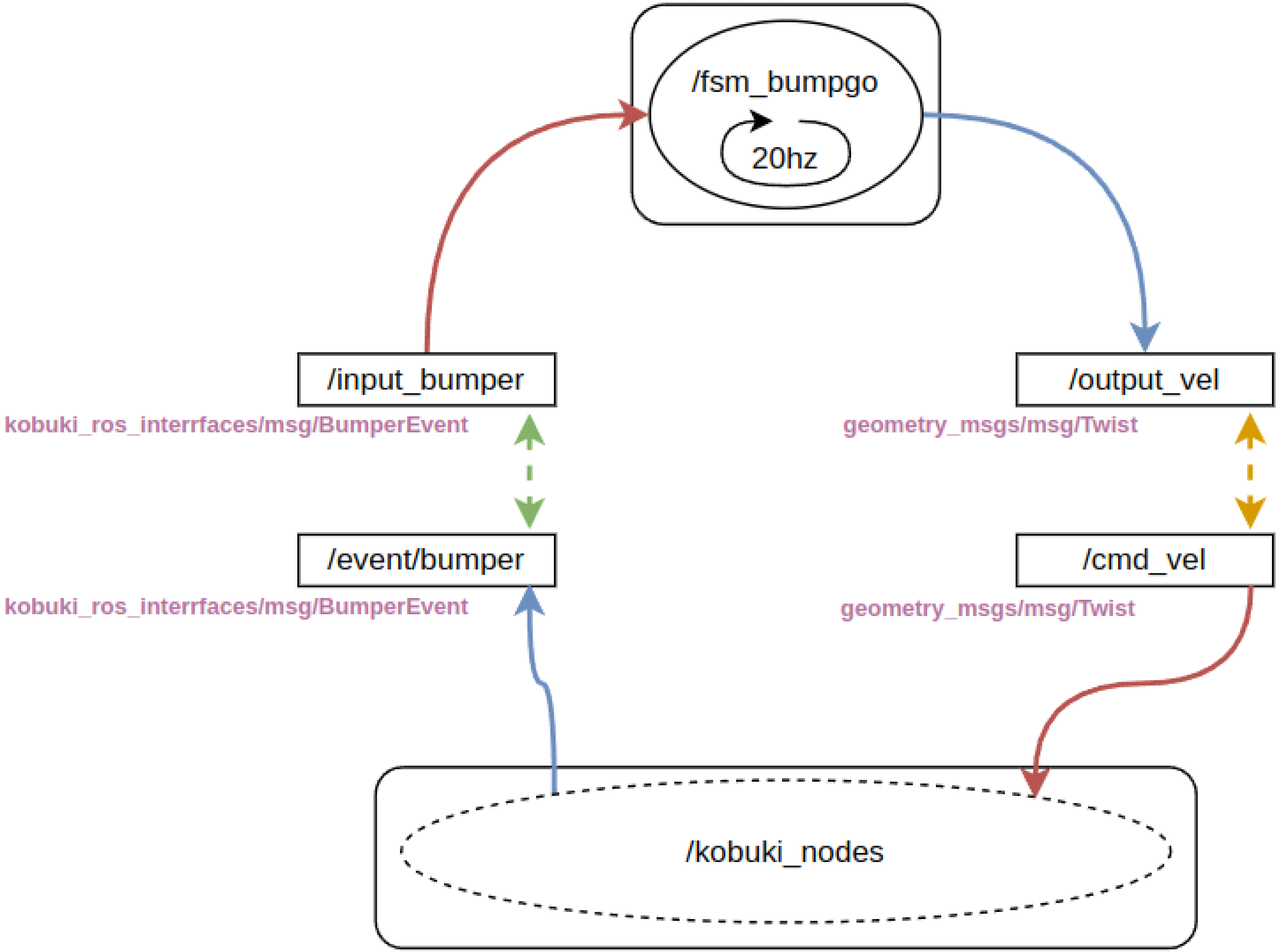
- **Goal**: a Bump&Go behavior
- New concepts:
  - **FSMs**
  - **Robot Motion**



Computation Graph Tiago Sim

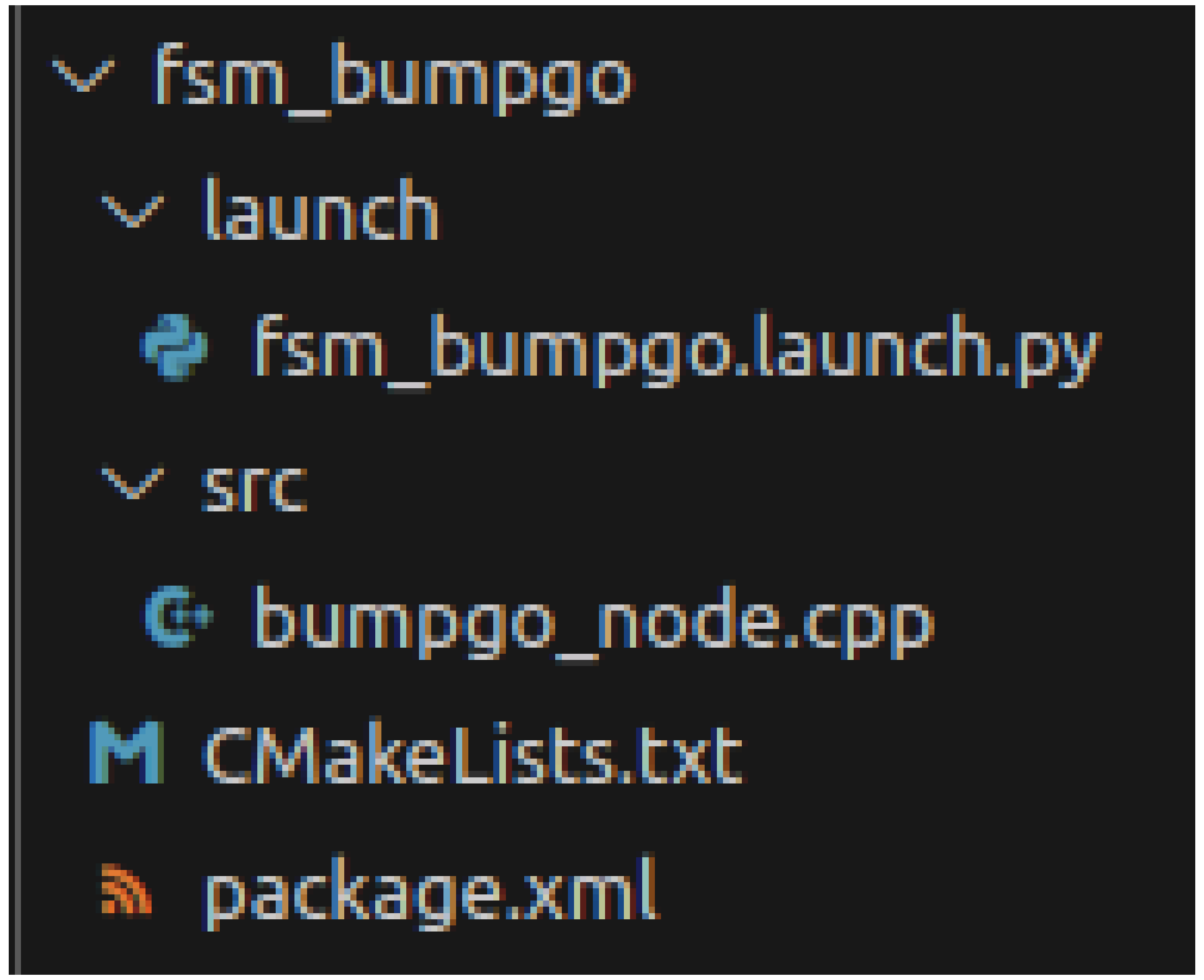


Computation Graph Kobuki



Bump&Go in C++

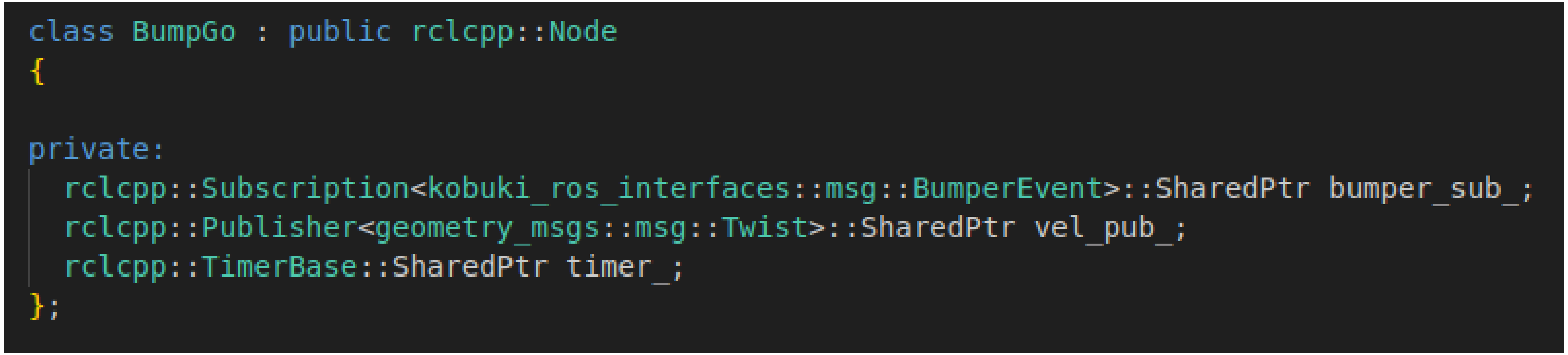
Package content



```
fsm_bumpgo
  launch
    fsm_bumpgo.launch.py
  src
    bumpgo_node.cpp
  CMakeLists.txt
  package.xml
```

Bump&Go in C++

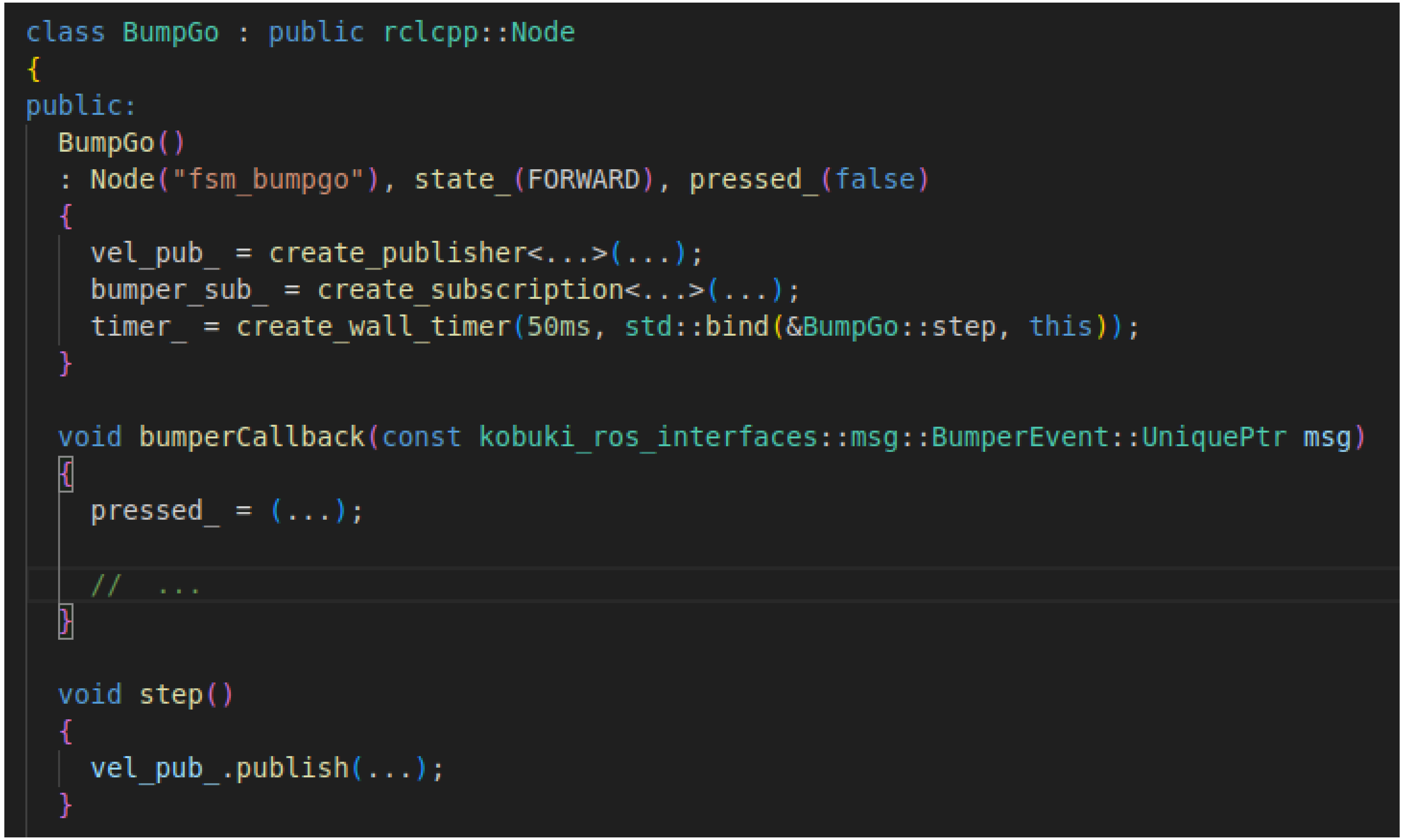
Execution Control



```
class BumpGo : public rclcpp::Node
{

private:
  rclcpp::Subscription<kobuki_ros_interfaces::msg::BumperEvent>::SharedPtr bumper_sub_;
  rclcpp::Publisher<geometry_msgs::msg::Twist>::SharedPtr vel_pub_;
  rclcpp::TimerBase::SharedPtr timer_;
};
```

Bump&Go in C++

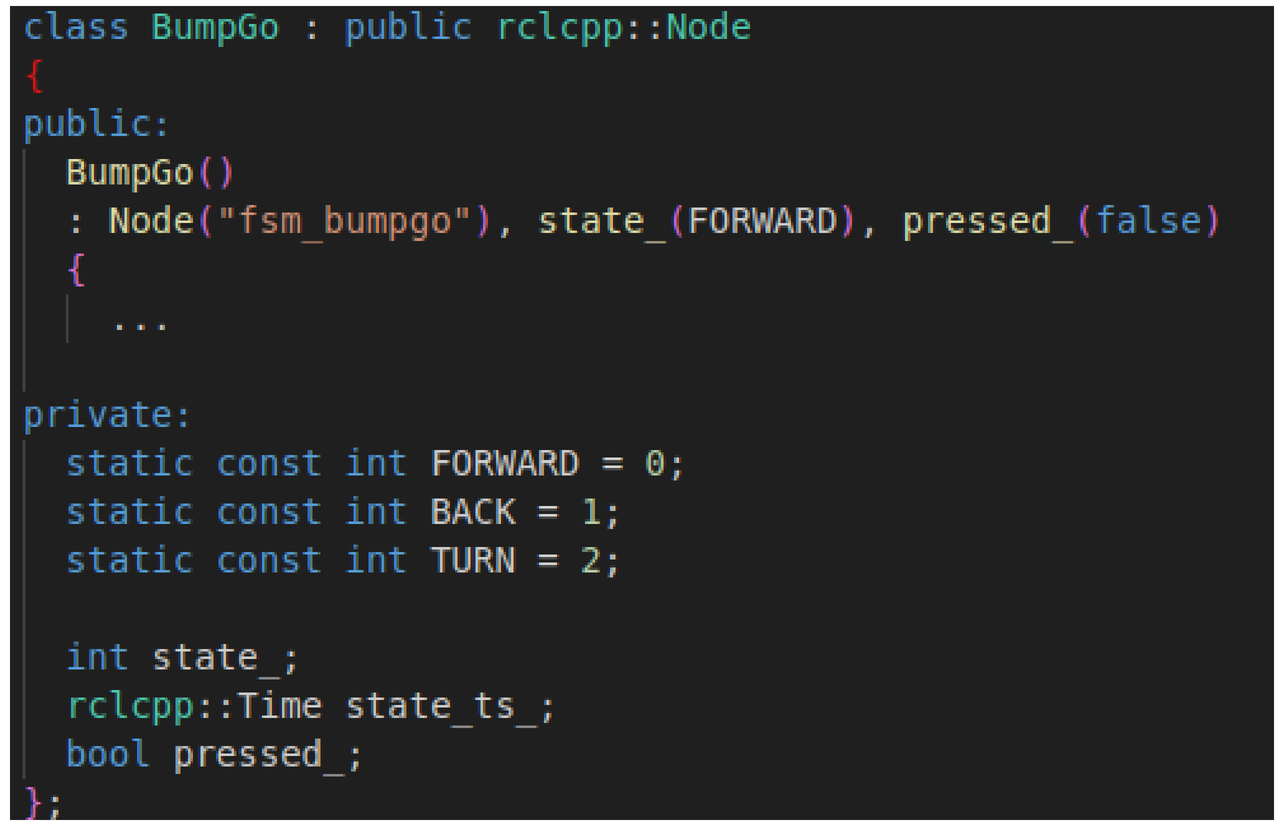
Subscriptions and publications



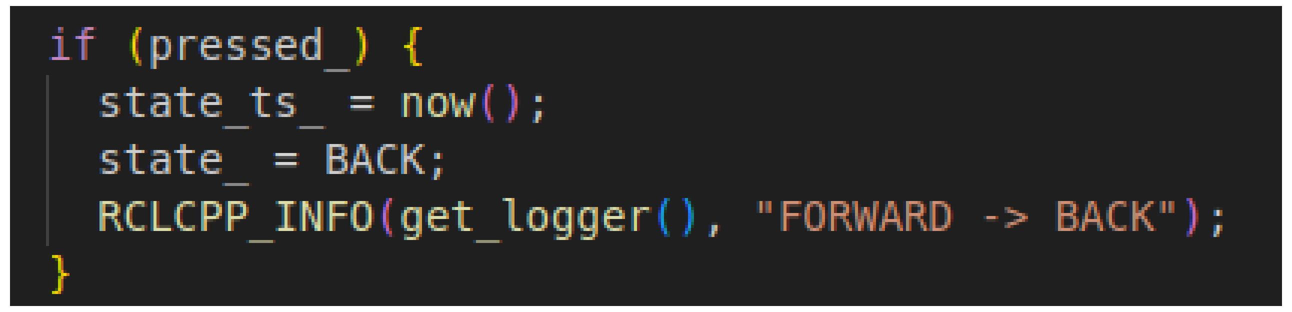
```cpp
class BumpGo : public rclcpp::Node
{
public:
  BumpGo()
  : Node("fsm_bumpgo"), state_(FORWARD), pressed_(false)
  {
    vel_pub_ = create_publisher<...>(...);
    bumper_sub_ = create_subscription<...>(...);
    timer_ = create_wall_timer(50ms, std::bind(&BumpGo::step, this));
  }

  void bumperCallback(const kobuki_ros_interfaces::msg::BumperEvent::UniquePtr msg)
  {
    pressed_ = (...);

    //  ...
  }

  void step()
  {
    vel_pub_.publish(...);
  }
```

Bump&Go in C++

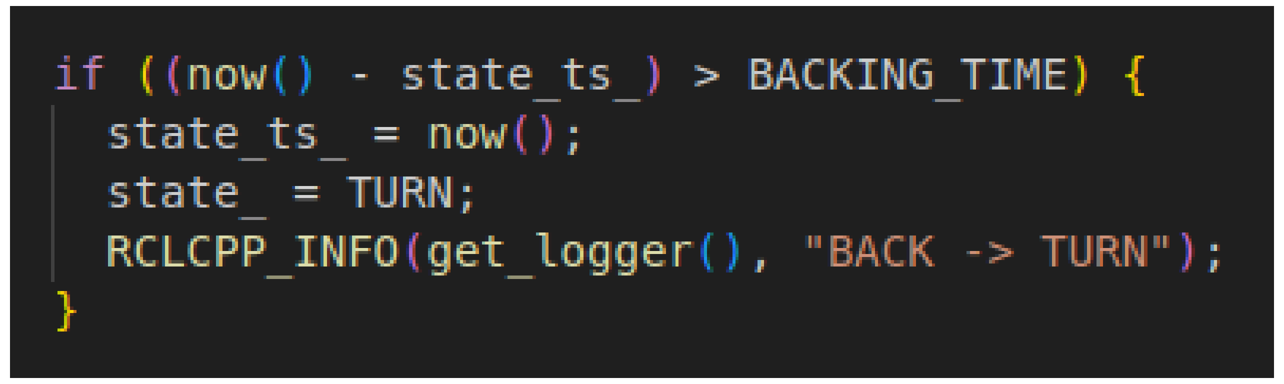
Implementing a FSM



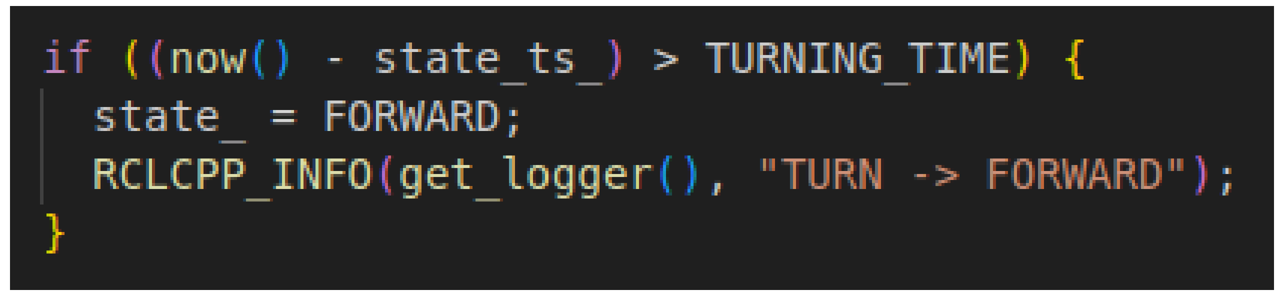
```
class BumpGo : public rclcpp::Node
{
public:
  BumpGo()
  : Node("fsm_bumpgo"), state_(FORWARD), pressed_(false)
  {
    ...

private:
  static const int FORWARD = 0;
  static const int BACK = 1;
  static const int TURN = 2;

  int state_;
  rclcpp::Time state_ts_;
  bool pressed_;
};
```



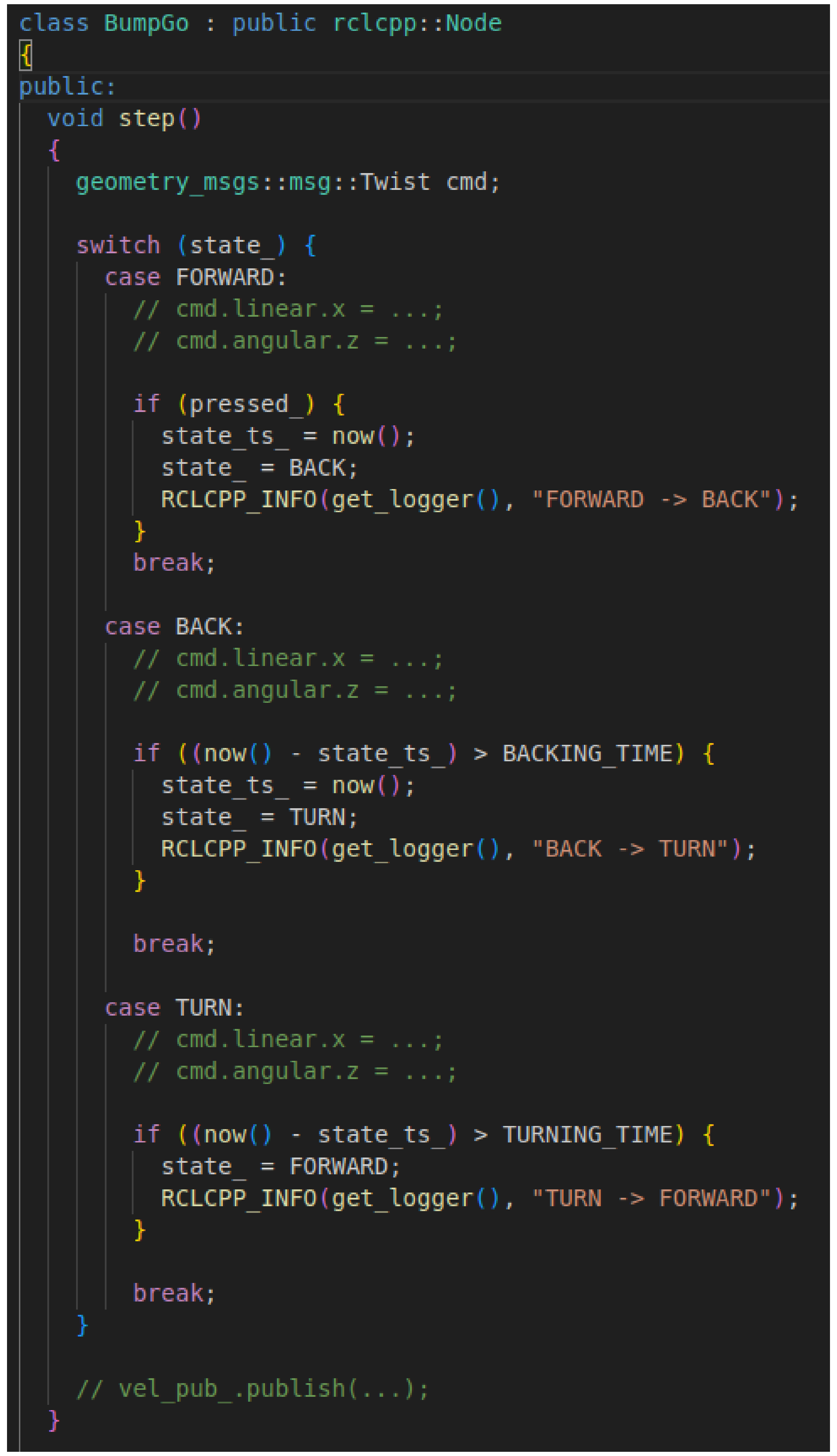
```
if (pressed_) {
  state_ts_ = now();
  state_ = BACK;
  RCLCPP_INFO(get_logger(), "FORWARD -> BACK");
}
```



```
if ((now() - state_ts_) > BACKING_TIME) {
  state_ts_ = now();
  state_ = TURN;
  RCLCPP_INFO(get_logger(), "BACK -> TURN");
}
```



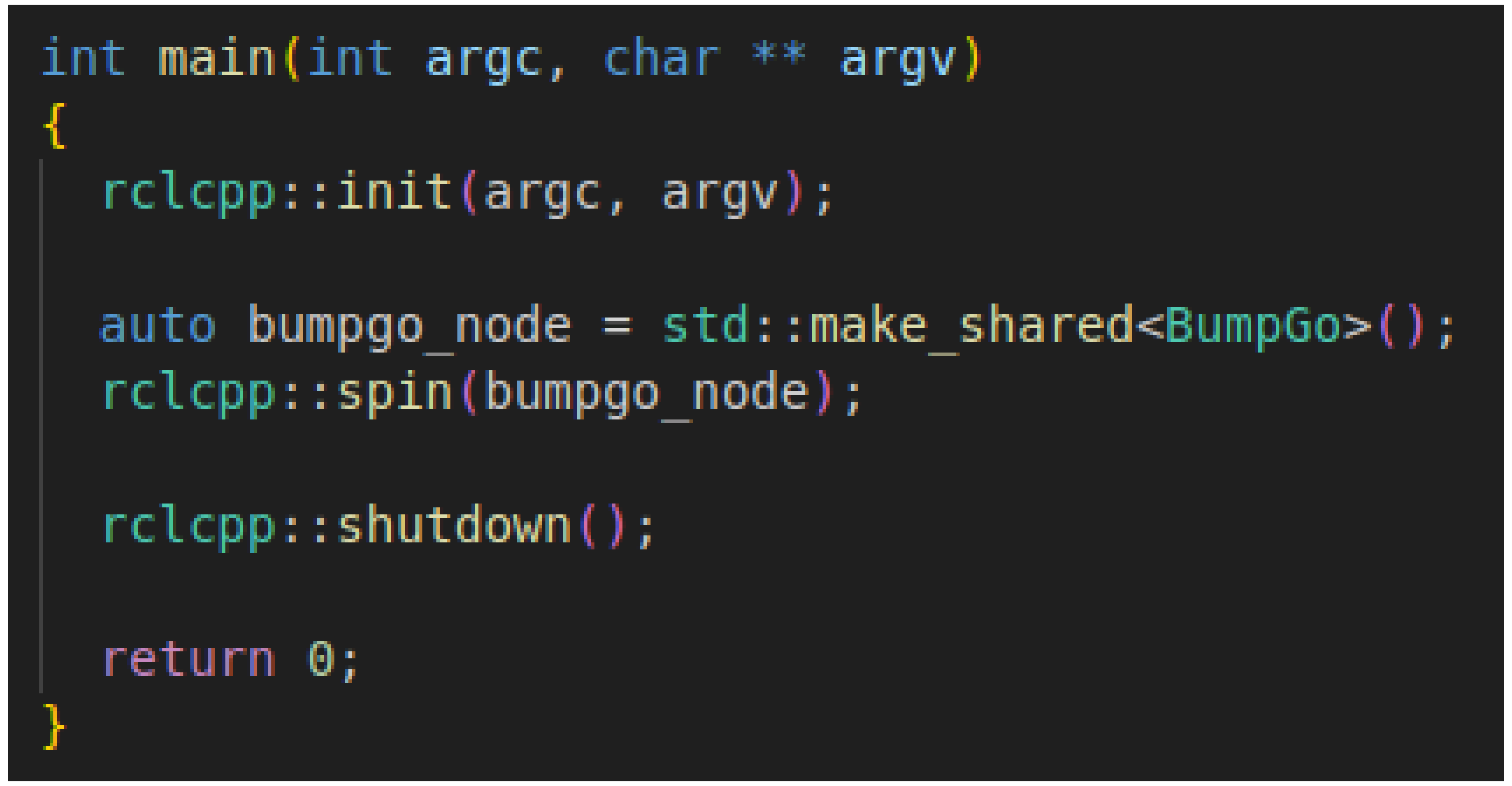
```
if ((now() - state_ts_) > TURNING_TIME) {
  state_ = FORWARD;
  RCLCPP_INFO(get_logger(), "TURN -> FORWARD");
}
```



```
class BumpGo : public rclcpp::Node
{
public:
  void step()
  {
    geometry_msgs::msg::Twist cmd;

    switch (state_) {
      case FORWARD:
        // cmd.linear.x = ...;
        // cmd.angular.z = ...;

        if (pressed_) {
          state_ts_ = now();
          state_ = BACK;
          RCLCPP_INFO(get_logger(), "FORWARD -> BACK");
        }
        break;

      case BACK:
        // cmd.linear.x = ...;
        // cmd.angular.z = ...;

        if ((now() - state_ts_) > BACKING_TIME) {
          state_ts_ = now();
          state_ = TURN;
          RCLCPP_INFO(get_logger(), "BACK -> TURN");
        }

        break;

      case TURN:
        // cmd.linear.x = ...;
        // cmd.angular.z = ...;

        if ((now() - state_ts_) > TURNING_TIME) {
          state_ = FORWARD;
          RCLCPP_INFO(get_logger(), "TURN -> FORWARD");
        }

        break;
    }

    // vel_pub_.publish(...);
  }
```

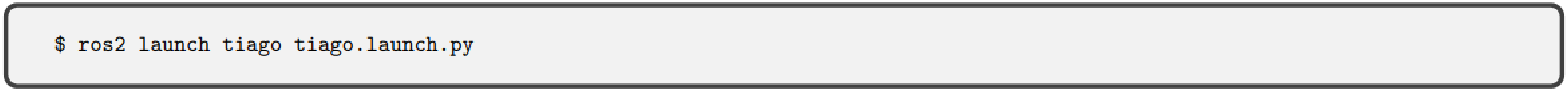
Bump&Go in C++

Running the code



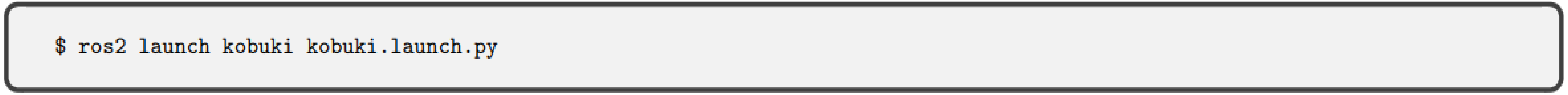
```cpp
int main(int argc, char ** argv)
{
  rclcpp::init(argc, argv);

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

  rclcpp::shutdown();

  return 0;
}
```

1. Launch your robot:



```
$ ros2 launch tiago tiago.launch.py
```

or



```
$ ros2 launch kobuki kobuki.launch.py
```

Bump&Go in C++

Running the code

2. Run your implementation:

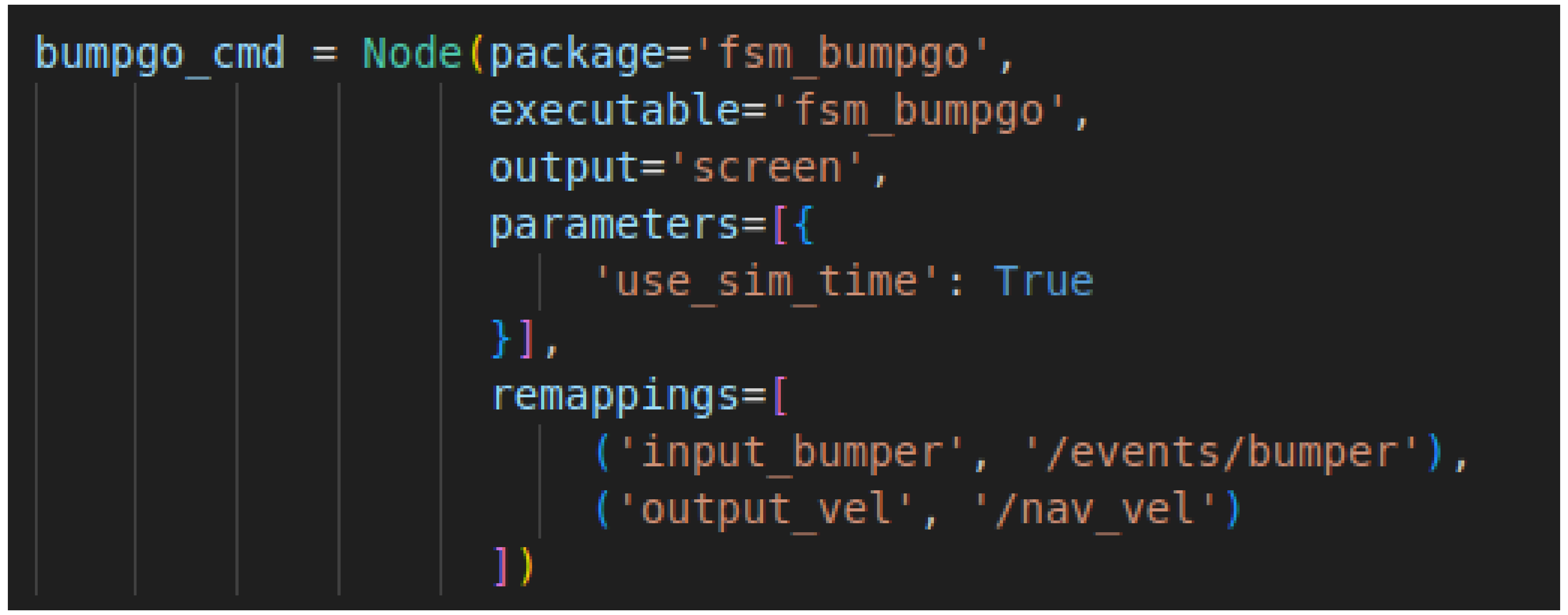
```
$ ros2 run fsm_bumpgo fsm_bumpgo --ros-args -r output_vel:=/nav_vel -r input_bumper:/events/bumper -p use_sim_time:=true
```

or

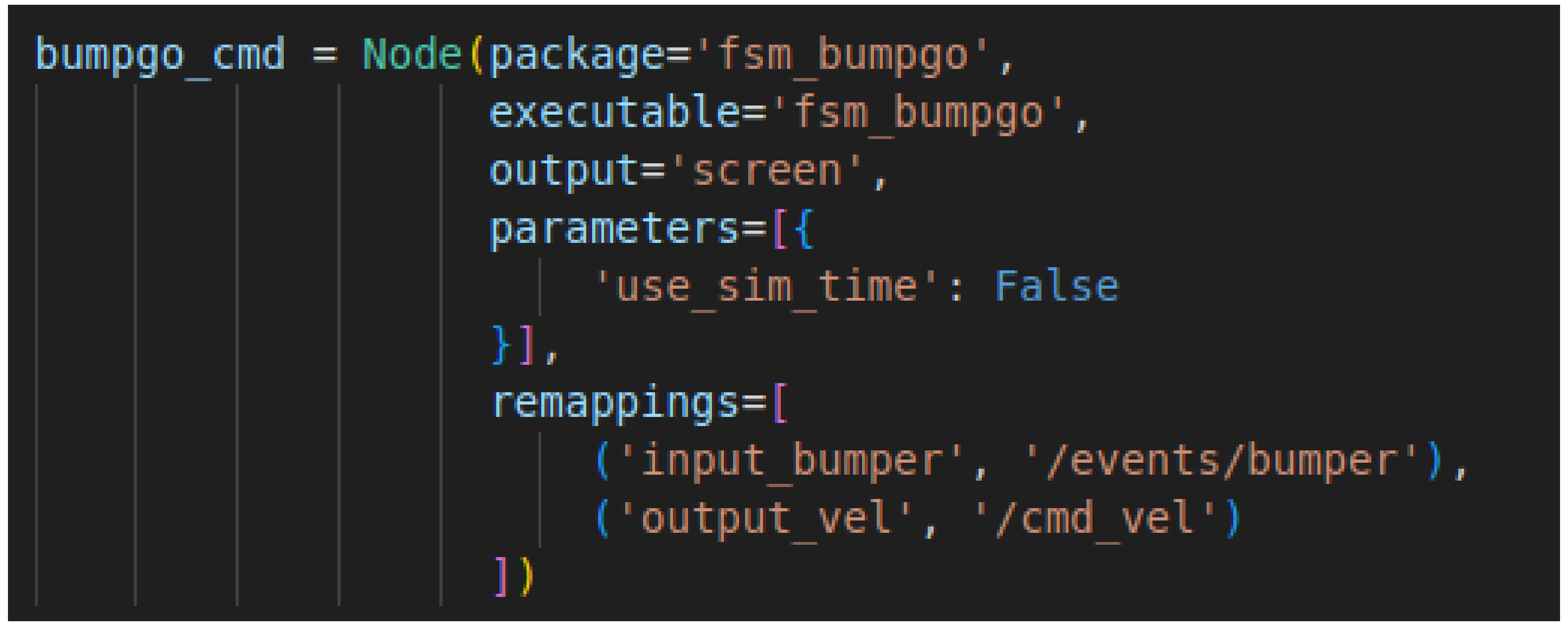
```
$ ros2 run fsm_bumpgo fsm_bumpgo --ros-args -r output_vel:=/cmd_vel -r input_bumper:/events/bumper -p use_sim_time:=false
```

or use a launcher

```
$ ros2 launch fsm_bumpgo fsm_bumpgo.launch.py
```



```
bumpgo_cmd = Node(package='fsm_bumpgo',
                  executable='fsm_bumpgo',
                  output='screen',
                  parameters=[{
                    'use_sim_time': True
                  }],
                  remappings=[
                    ('input_bumper', '/events/bumper'),
                    ('output_vel', '/nav_vel')
                  ])
```



```
bumpgo_cmd = Node(package='fsm_bumpgo',
                  executable='fsm_bumpgo',
                  output='screen',
                  parameters=[{
                    'use_sim_time': False
                  }],
                  remappings=[
                    ('input_bumper', '/events/bumper'),
                    ('output_vel', '/cmd_vel')
                  ])
```