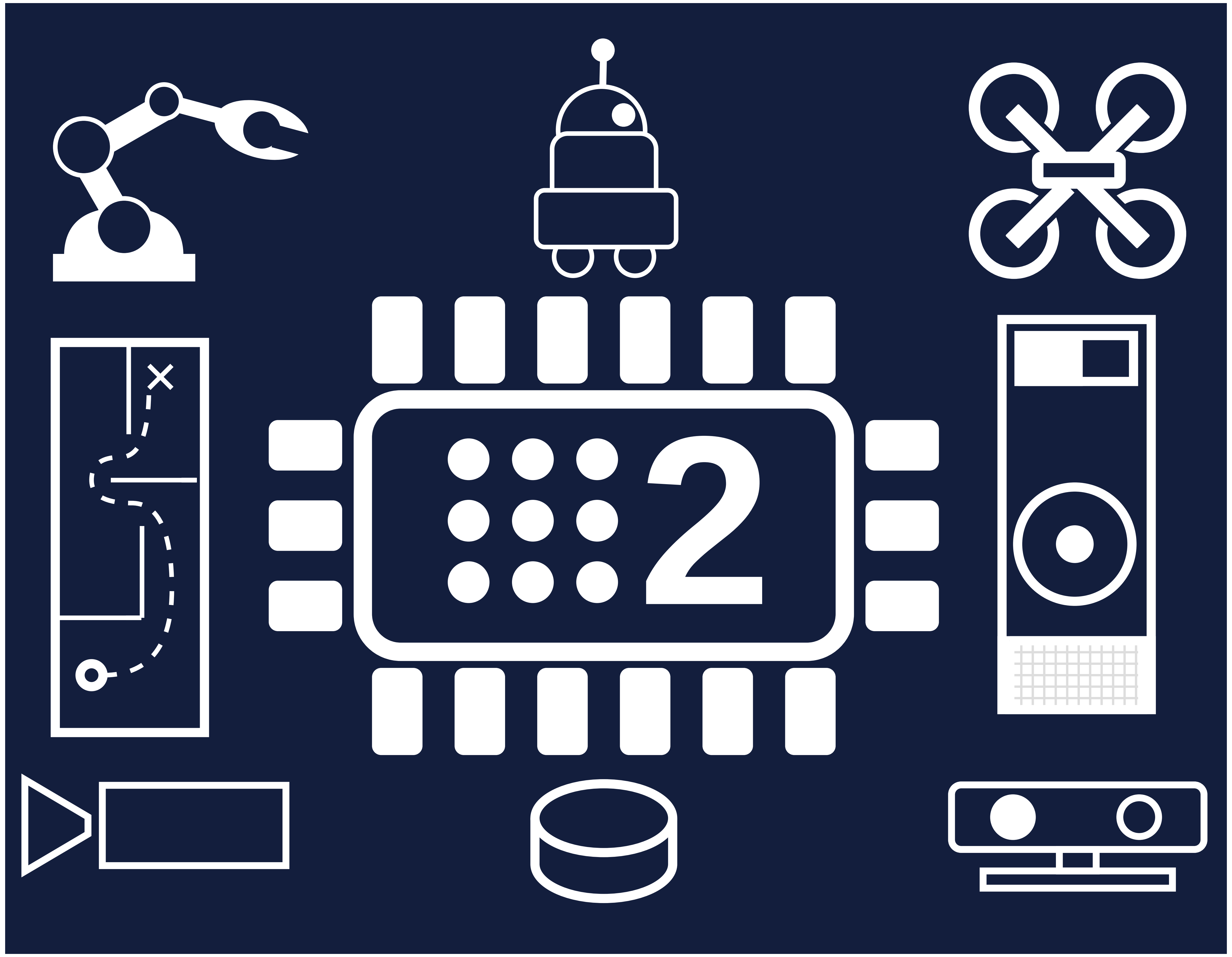


A Concise Introduction to Robot Programming in ROS2

Prof. Dr. Francisco Martín Rico

Chapter 4: The TF Subsytem

francisco.rico@urjc.es
2022 @fmrico

Universidad Rey Juan Carlos

Intelligent Robotics Lab

Introduction

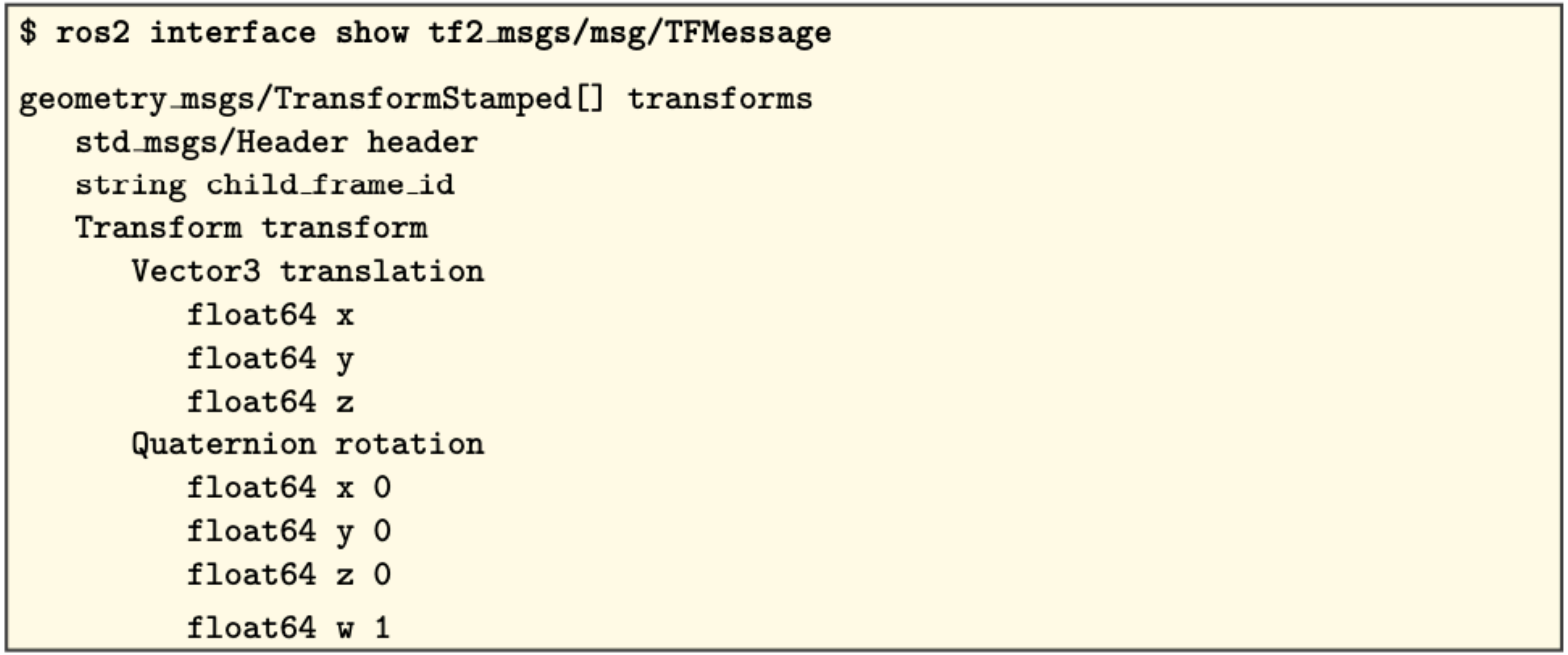
- One of the greatest treasures in ROS
- It allows to use and transform coordinates between different reference axes (**frames**)
- The robot perceives through sensors placed somewhere in the robot, even in moving parts, and performs actions specifying spatial positions

$$P_B = RT_{A \to B} * P_A$$

$$\begin{pmatrix} x_B \\ y_B \\ z_B \\ 1 \end{pmatrix} = \begin{pmatrix} R^{xx}_{A \to B} & R^{xy}_{A \to B} & R^{xz}_{A \to B} & T^{x}_{A \to B} \\ R^{yx}_{A \to B} & R^{yy}_{A \to B} & R^{yz}_{A \to B} & T^{y}_{A \to B} \\ R^{zx}_{A \to B} & R^{zy}_{A \to B} & R^{zz}_{A \to B} & T^{z}_{A \to B} \\ 0 & 0 & 0 & 1 \end{pmatrix} * \begin{pmatrix} x_A \\ y_A \\ z_A \\ 1 \end{pmatrix}$$

Introduction

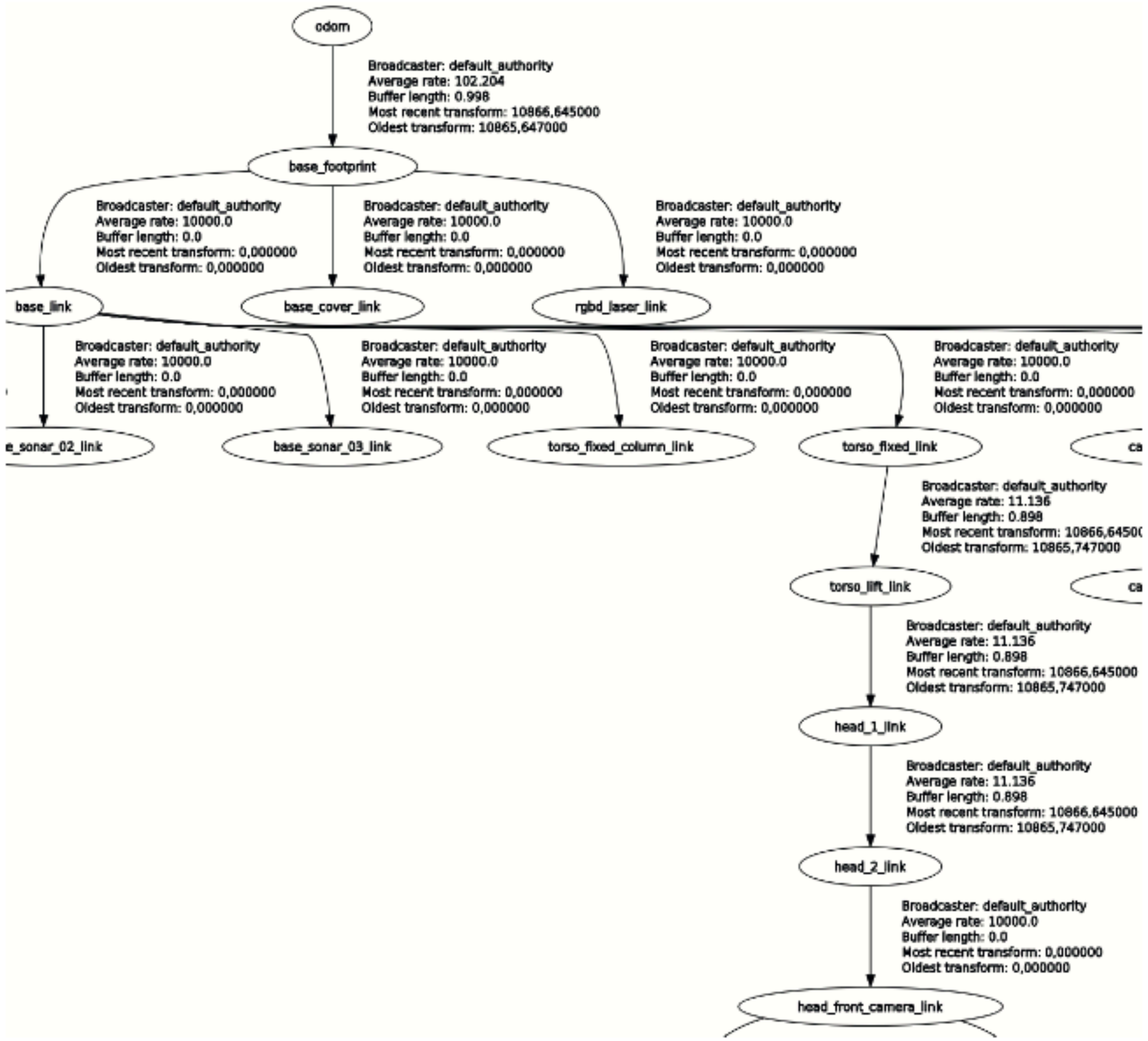
- Topics /tf and /tf_static (tf2 msgs/msg/TFMessage)



```
$ ros2 interface show tf2_msgs/msg/TFMessage

geometry_msgs/TransformStamped[] transforms
   std_msgs/Header header
   string child_frame_id
   Transform transform
      Vector3 translation
         float64 x
         float64 y
         float64 z
      Quaternion rotation
         float64 x 0
         float64 y 0
         float64 z 0
         float64 w 1
```

Introduction

```
$ ros2 run rqt_tf_tree rqt_tf_tree
```



Broadcaster: default_authority
Average rate: 102.204
Buffer length: 0.998
Most recent transform: 10866,645000
Oldest transform: 10865,647000

Broadcaster: default_authority
Average rate: 10000.0
Buffer length: 0.0
Most recent transform: 0,000000
Oldest transform: 0,000000

Broadcaster: default_authority
Average rate: 10000.0
Buffer length: 0.0
Most recent transform: 0,000000
Oldest transform: 0,000000

Broadcaster: default_authority
Average rate: 10000.0
Buffer length: 0.0
Most recent transform: 0,000000
Oldest transform: 0,000000

Broadcaster: default_authority
Average rate: 10000.0
Buffer length: 0.0
Most recent transform: 0,000000
Oldest transform: 0,000000

Broadcaster: default_authority
Average rate: 10000.0
Buffer length: 0.0
Most recent transform: 0,000000
Oldest transform: 0,000000

Broadcaster: default_authority
Average rate: 10000.0
Buffer length: 0.0
Most recent transform: 0,000000
Oldest transform: 0,000000

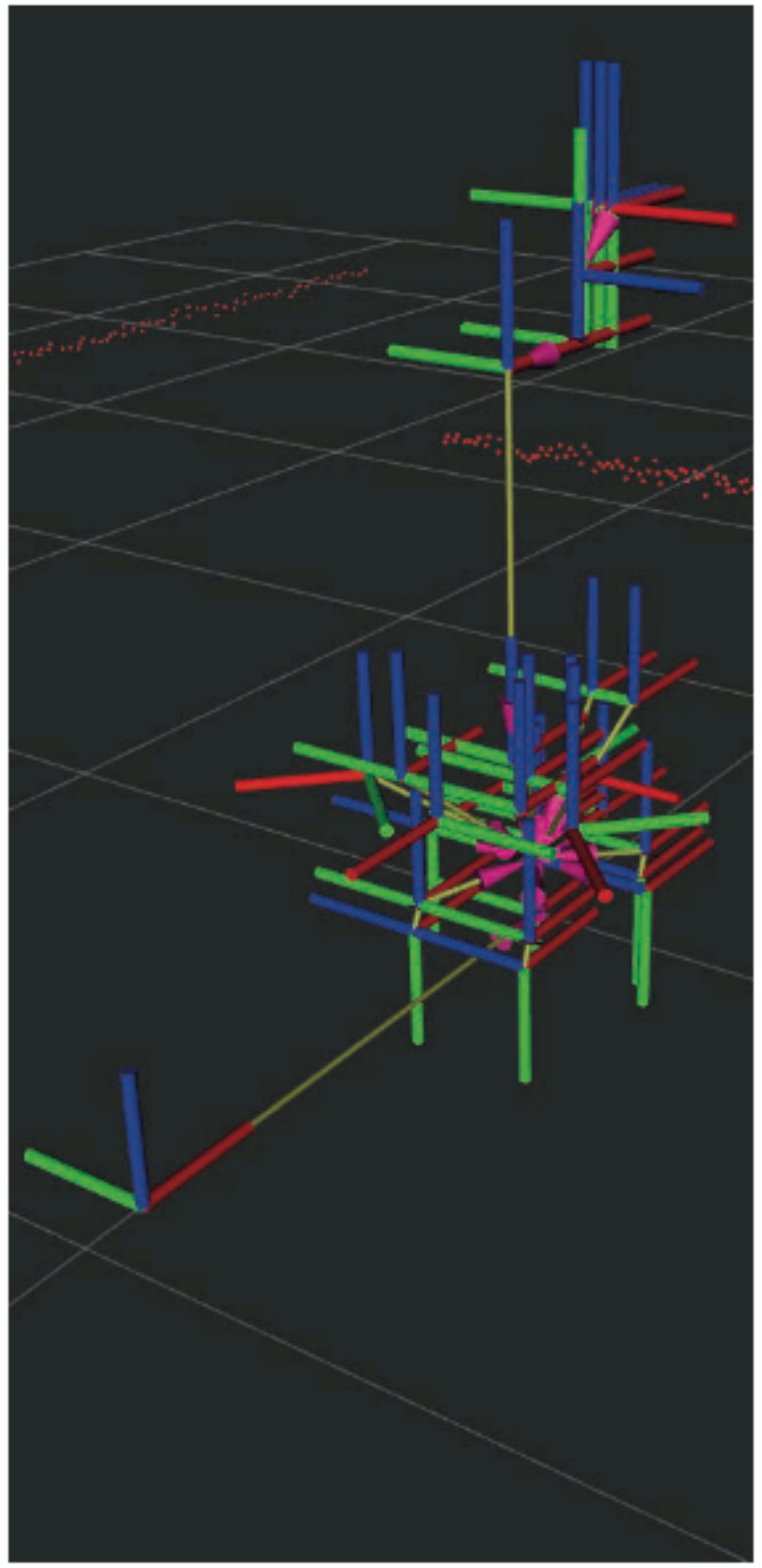
Broadcaster: default_authority
Average rate: 10000.0
Buffer length: 0.0
Most recent transform: 0,000000
Oldest transform: 0,000000

Broadcaster: default_authority
Average rate: 11.136
Buffer length: 0.898
Most recent transform: 10866,6450(
Oldest transform: 10865,747000

Broadcaster: default_authority
Average rate: 11.136
Buffer length: 0.898
Most recent transform: 10866,645000
Oldest transform: 10865,747000

Broadcaster: default_authority
Average rate: 11.136
Buffer length: 0.898
Most recent transform: 10866,645000
Oldest transform: 10865,747000

Broadcaster: default_authority
Average rate: 10000.0
Buffer length: 0.0
Most recent transform: 0,000000
Oldest transform: 0,000000

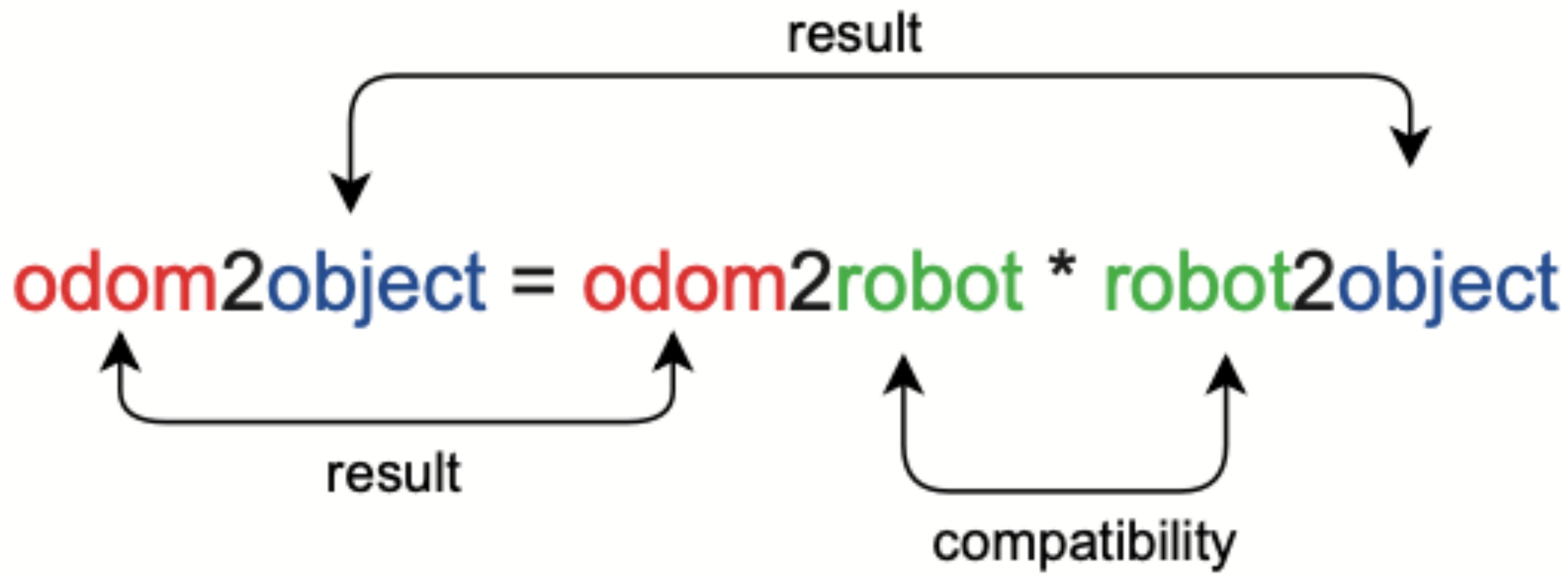


Introduction

TF Listeners and Publishers

```
geometry_msgs::msg::TransformStamped detection_tf;

detection_tf.header.frame_id = "base_footprint";
detection_tf.header.stamp = now();
detection_tf.child_frame_id = "detected_obstacle";
detection_tf.transform.translation.x = 1.0;

tf_broadcaster_->sendTransform(detection_tf);
```

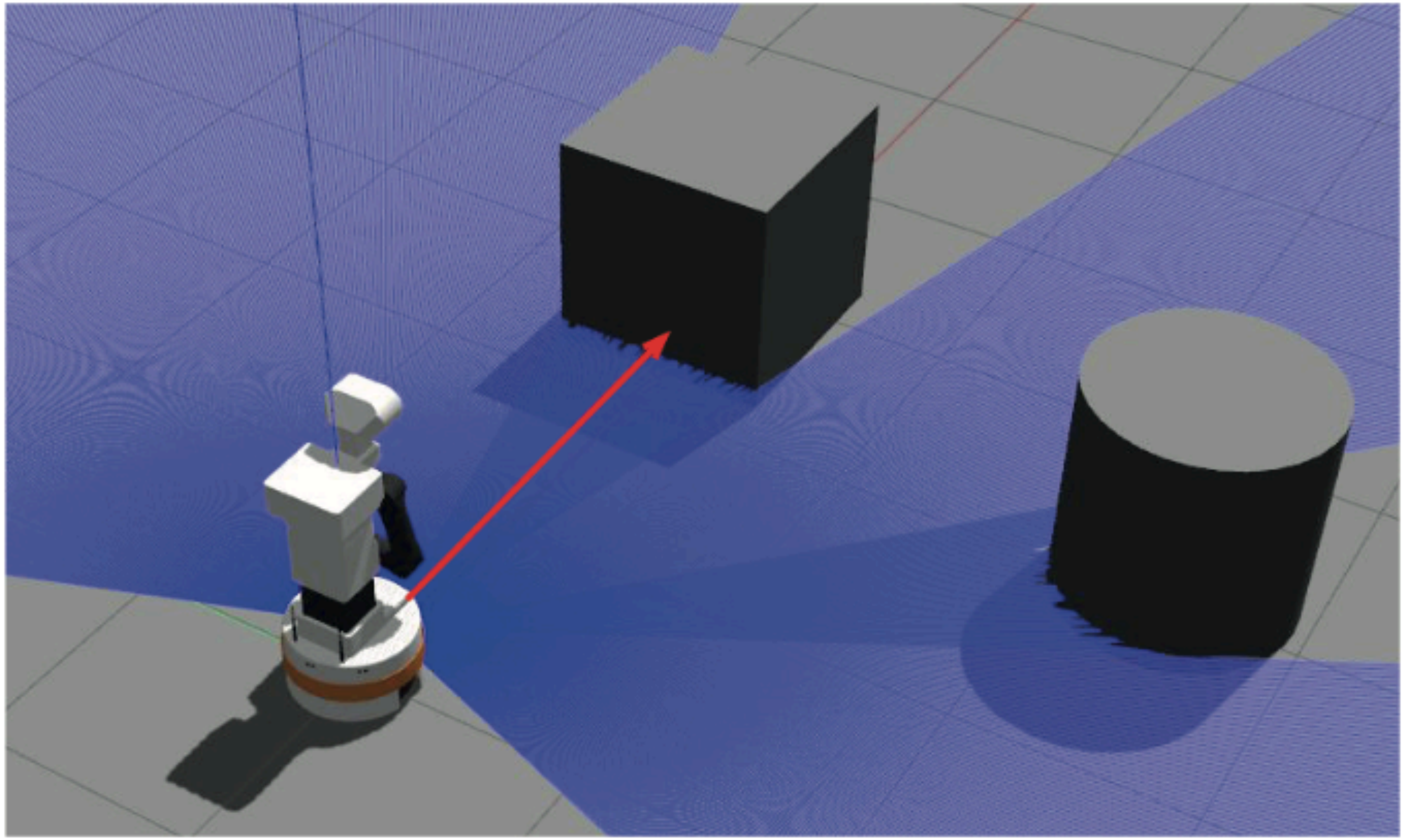
```
tf2_ros::Buffer tfBuffer;
tf2_ros::TransformListener tfListener(tfBuffer);

...

geometry_msgs::msg::TransformStamped odom2obstacle;
odom2obstacle = tfBuffer_.lookupTransform("odom", "detected_obstacle", tf2::TimePointZero);
```



An Obstacle Detector that uses TF2

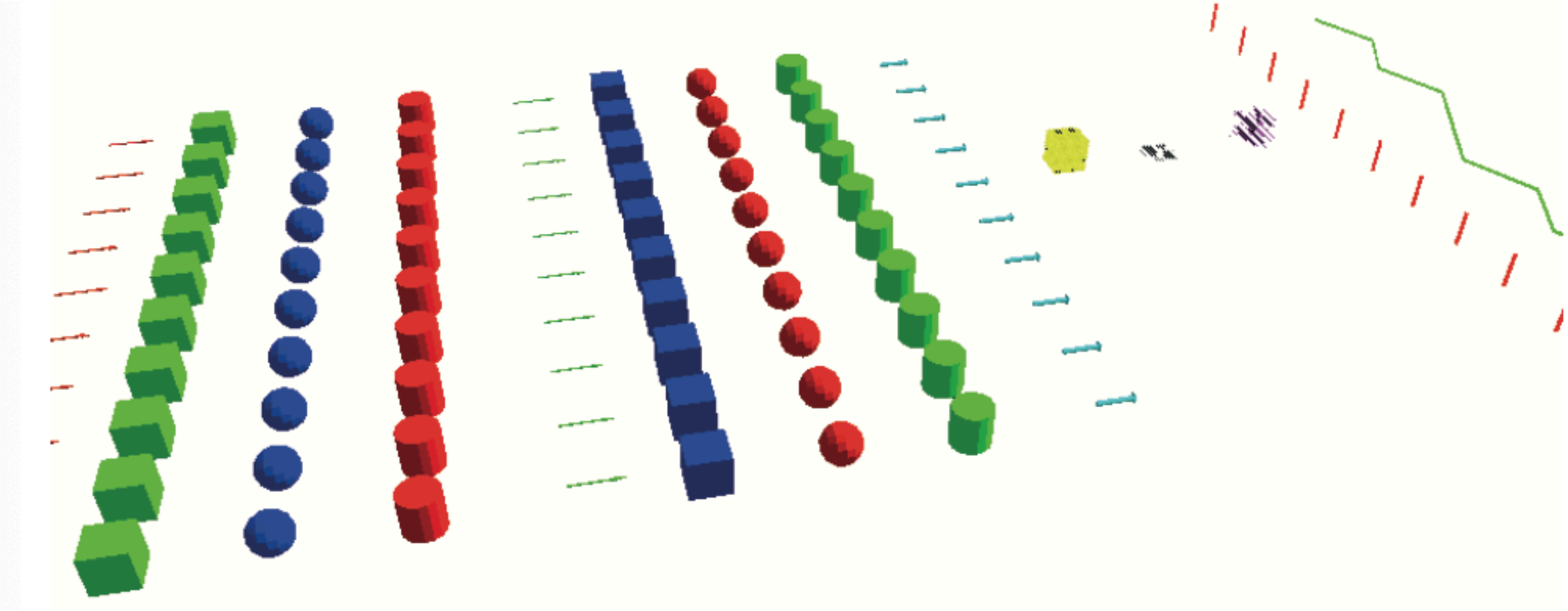
- The goal is detecting obstacles in from of the robot and debug it with visual tools

- New concepts:
  - **Use of TFs**
  - **Visual Markers**



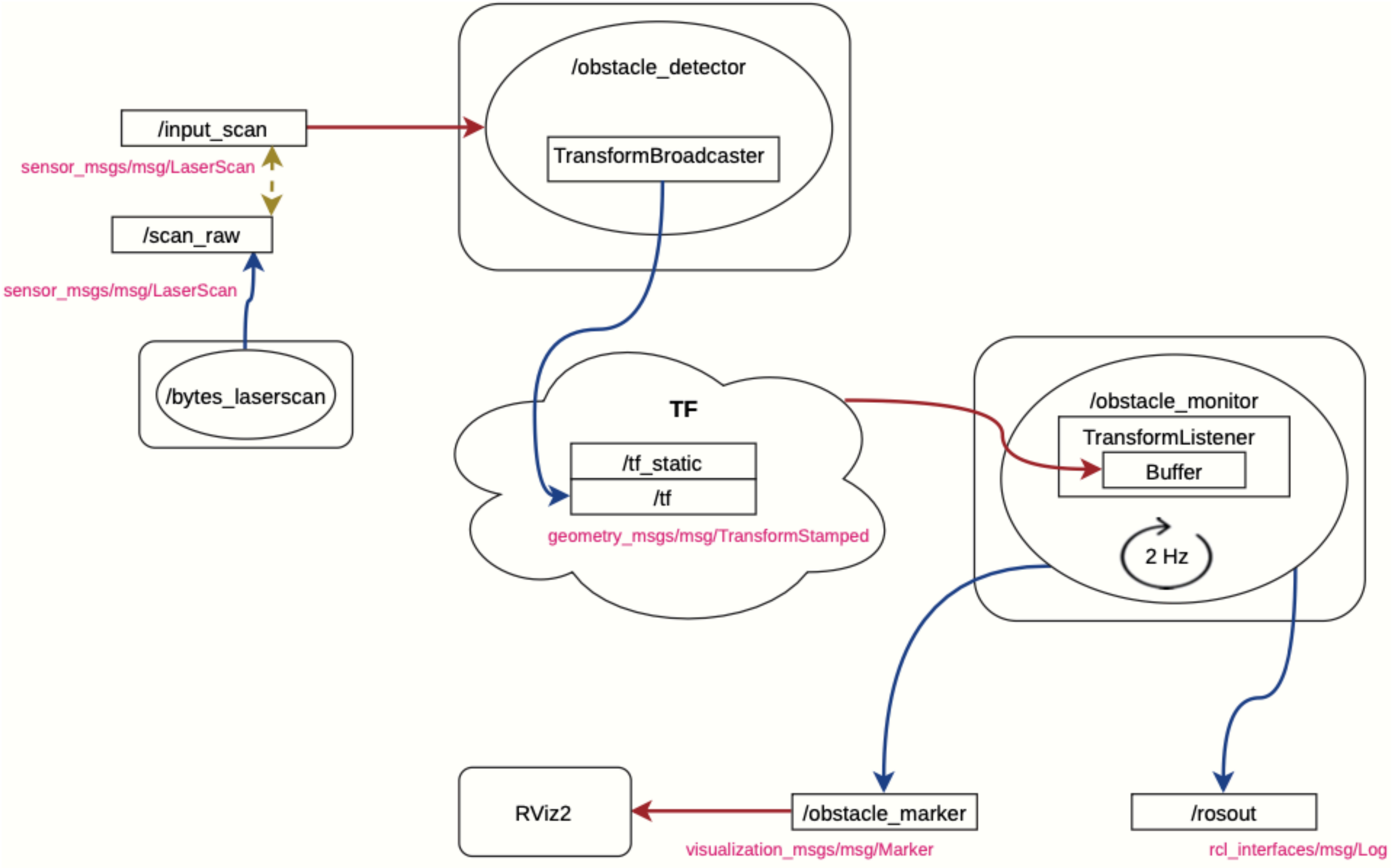
An Obstacle Detector that uses TF2

Use of Visual Markers for debugging



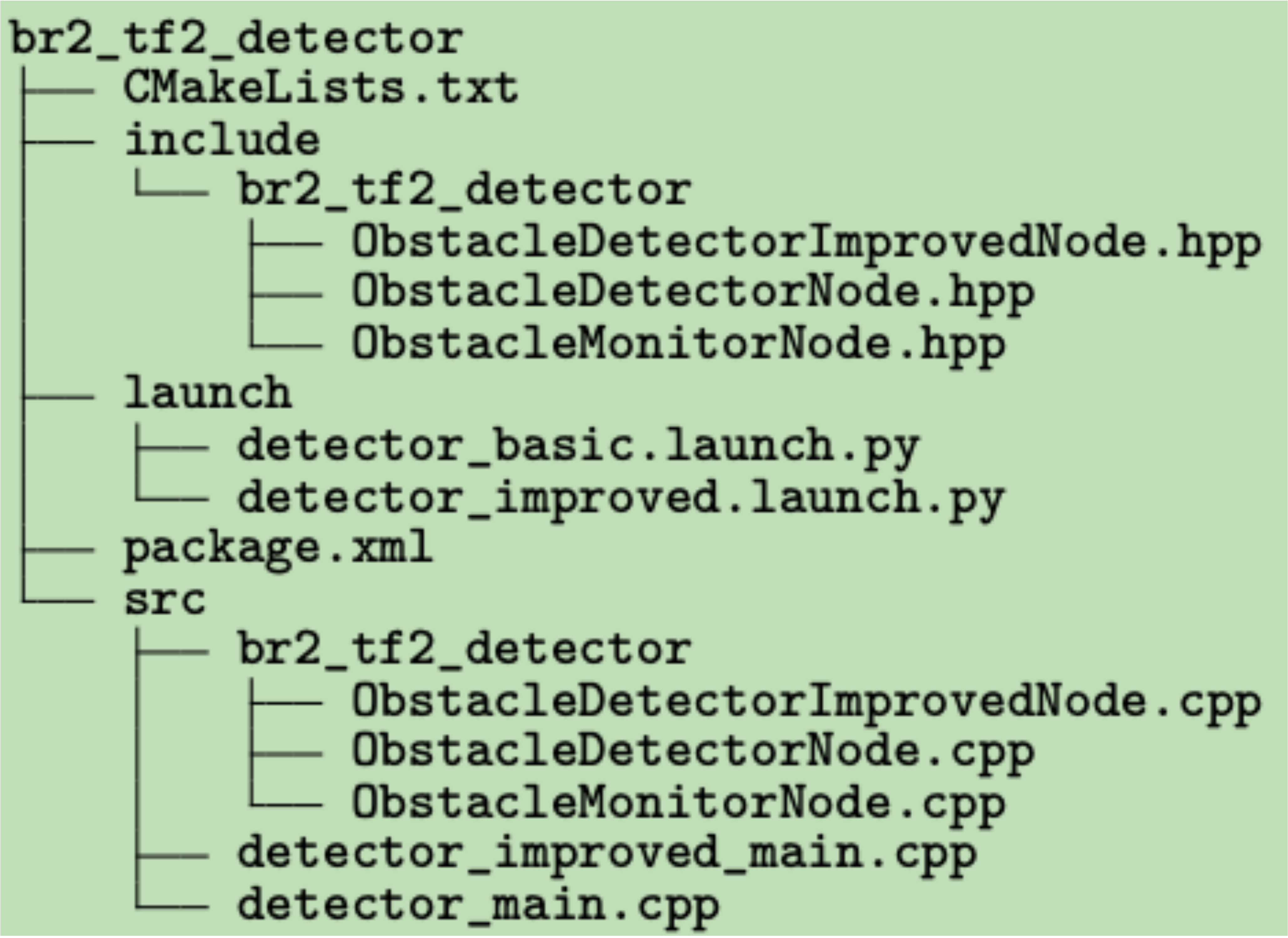
An Obstacle Detector that uses TF2

Computation Graph



An Obstacle Detector that uses TF2

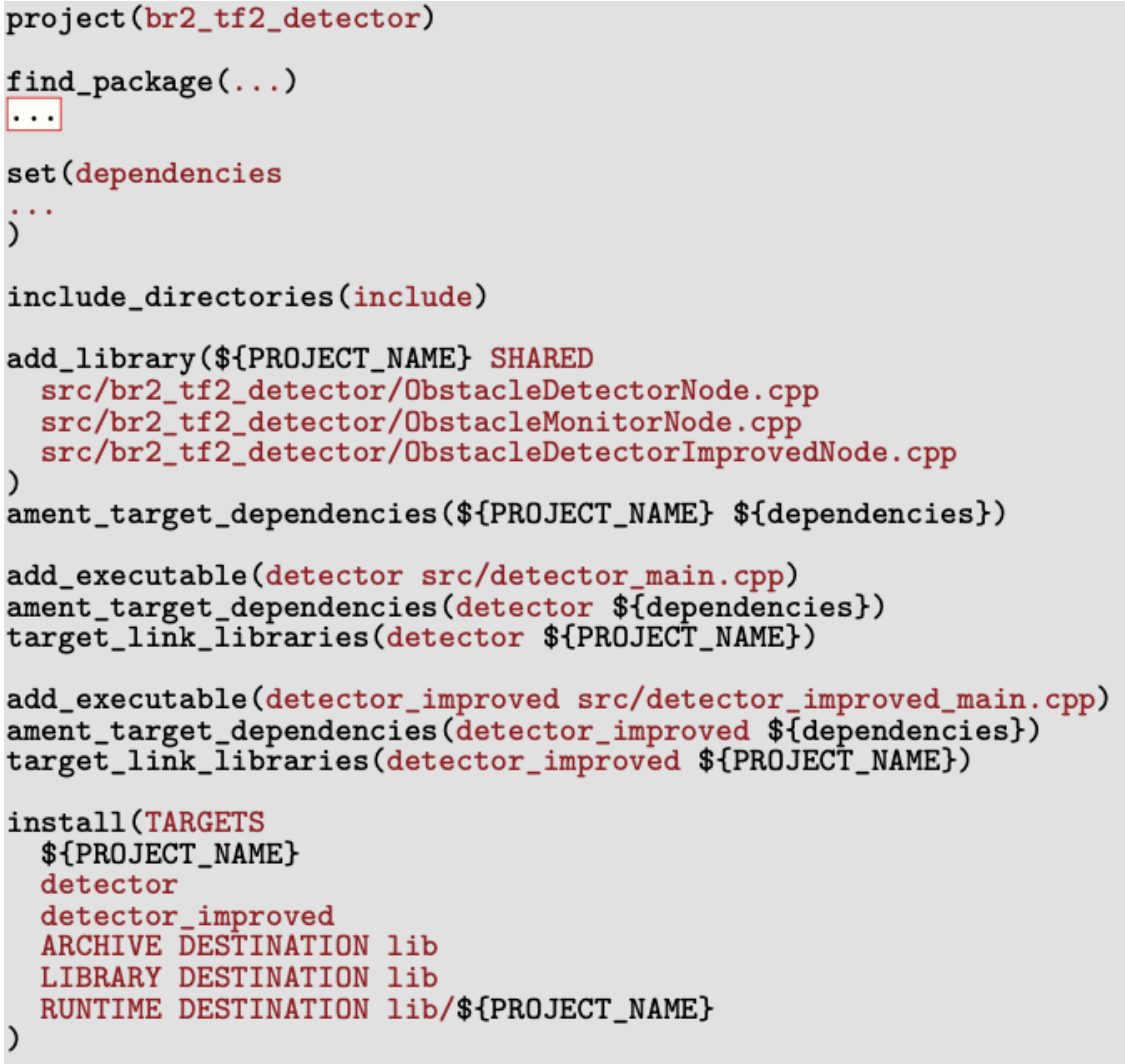
Package content



```
br2_tf2_detector
├── CMakeLists.txt
├── include
│   └── br2_tf2_detector
│       ├── ObstacleDetectorImprovedNode.hpp
│       ├── ObstacleDetectorNode.hpp
│       └── ObstacleMonitorNode.hpp
├── launch
│   ├── detector_basic.launch.py
│   └── detector_improved.launch.py
├── package.xml
└── src
    ├── br2_tf2_detector
    │   ├── ObstacleDetectorImprovedNode.cpp
    │   ├── ObstacleDetectorNode.cpp
    │   └── ObstacleMonitorNode.cpp
    ├── detector_improved_main.cpp
    └── detector_main.cpp
```

An Obstacle Detector that uses TF2

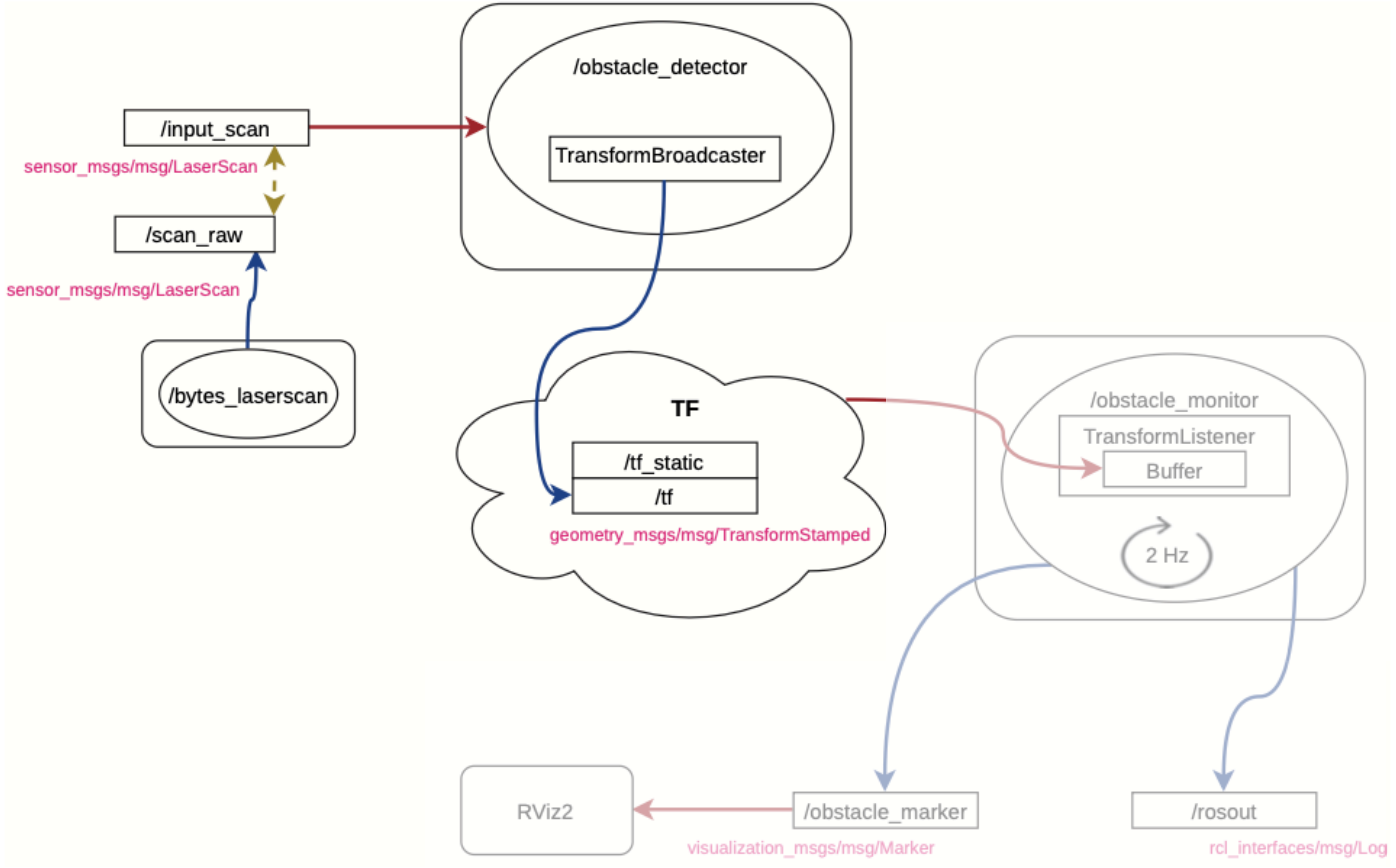
Build nodes as libraries



```
project(br2_tf2_detector)

find_package(...)
...

set(dependencies
...
)

include_directories(include)

add_library(${PROJECT_NAME} SHARED
  src/br2_tf2_detector/ObstacleDetectorNode.cpp
  src/br2_tf2_detector/ObstacleMonitorNode.cpp
  src/br2_tf2_detector/ObstacleDetectorImprovedNode.cpp
)
ament_target_dependencies(${PROJECT_NAME} ${dependencies})

add_executable(detector src/detector_main.cpp)
ament_target_dependencies(detector ${dependencies})
target_link_libraries(detector ${PROJECT_NAME})

add_executable(detector_improved src/detector_improved_main.cpp)
ament_target_dependencies(detector_improved ${dependencies})
target_link_libraries(detector_improved ${PROJECT_NAME})

install(TARGETS
  ${PROJECT_NAME}
  detector
  detector_improved
  ARCHIVE DESTINATION lib
  LIBRARY DESTINATION lib
  RUNTIME DESTINATION lib/${PROJECT_NAME}
)
```

An Obstacle Detector that uses TF2

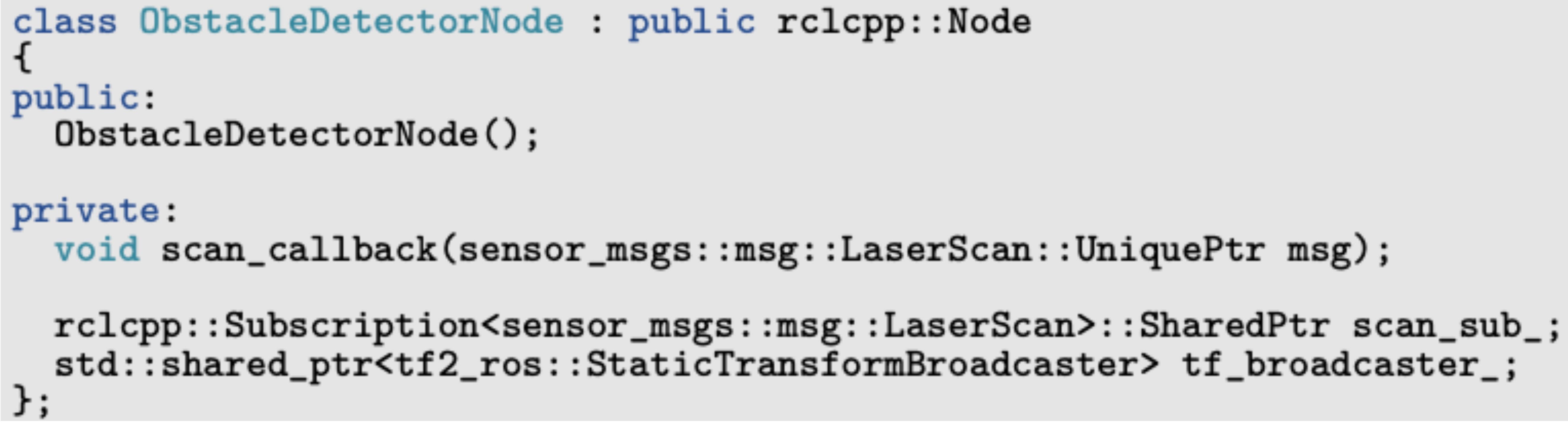
Obstacle Detector Node



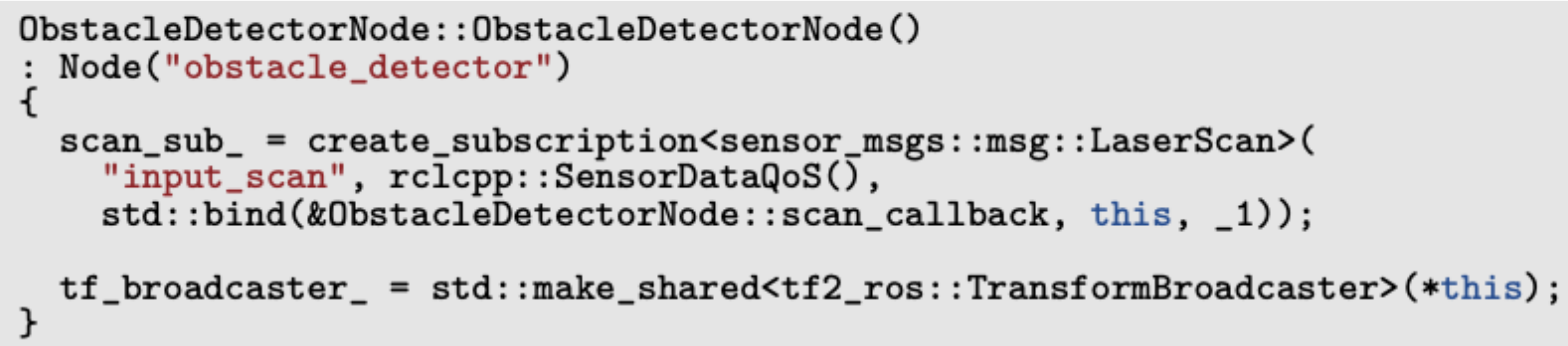
An Obstacle Detector that uses TF2

Obstacle Detector Node

- Use a TF broadcaster to send transforms to TF



```
class ObstacleDetectorNode : public rclcpp::Node
{
public:
  ObstacleDetectorNode();

private:
  void scan_callback(sensor_msgs::msg::LaserScan::UniquePtr msg);

  rclcpp::Subscription<sensor_msgs::msg::LaserScan>::SharedPtr scan_sub_;
  std::shared_ptr<tf2_ros::StaticTransformBroadcaster> tf_broadcaster_;
};
```

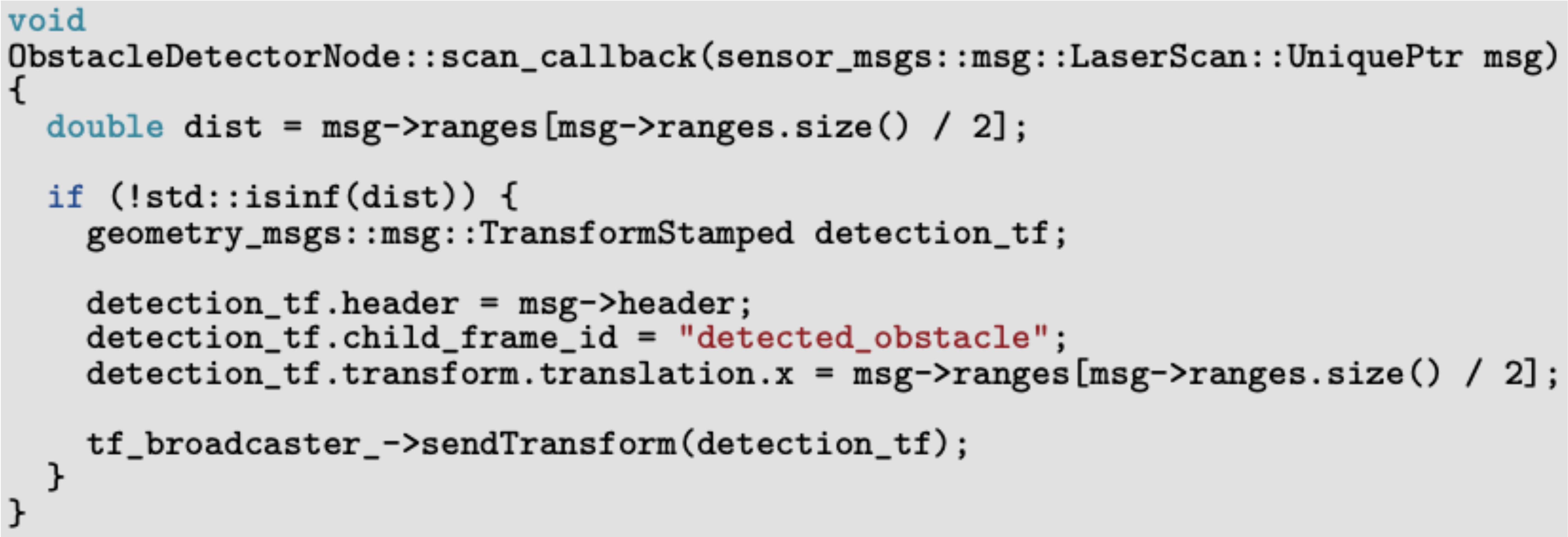


```
ObstacleDetectorNode::ObstacleDetectorNode()
: Node("obstacle_detector")
{
  scan_sub_ = create_subscription<sensor_msgs::msg::LaserScan>(
    "input_scan", rclcpp::SensorDataQoS(),
    std::bind(&ObstacleDetectorNode::scan_callback, this, _1));

  tf_broadcaster_ = std::make_shared<tf2_ros::TransformBroadcaster>(*this);
}
```

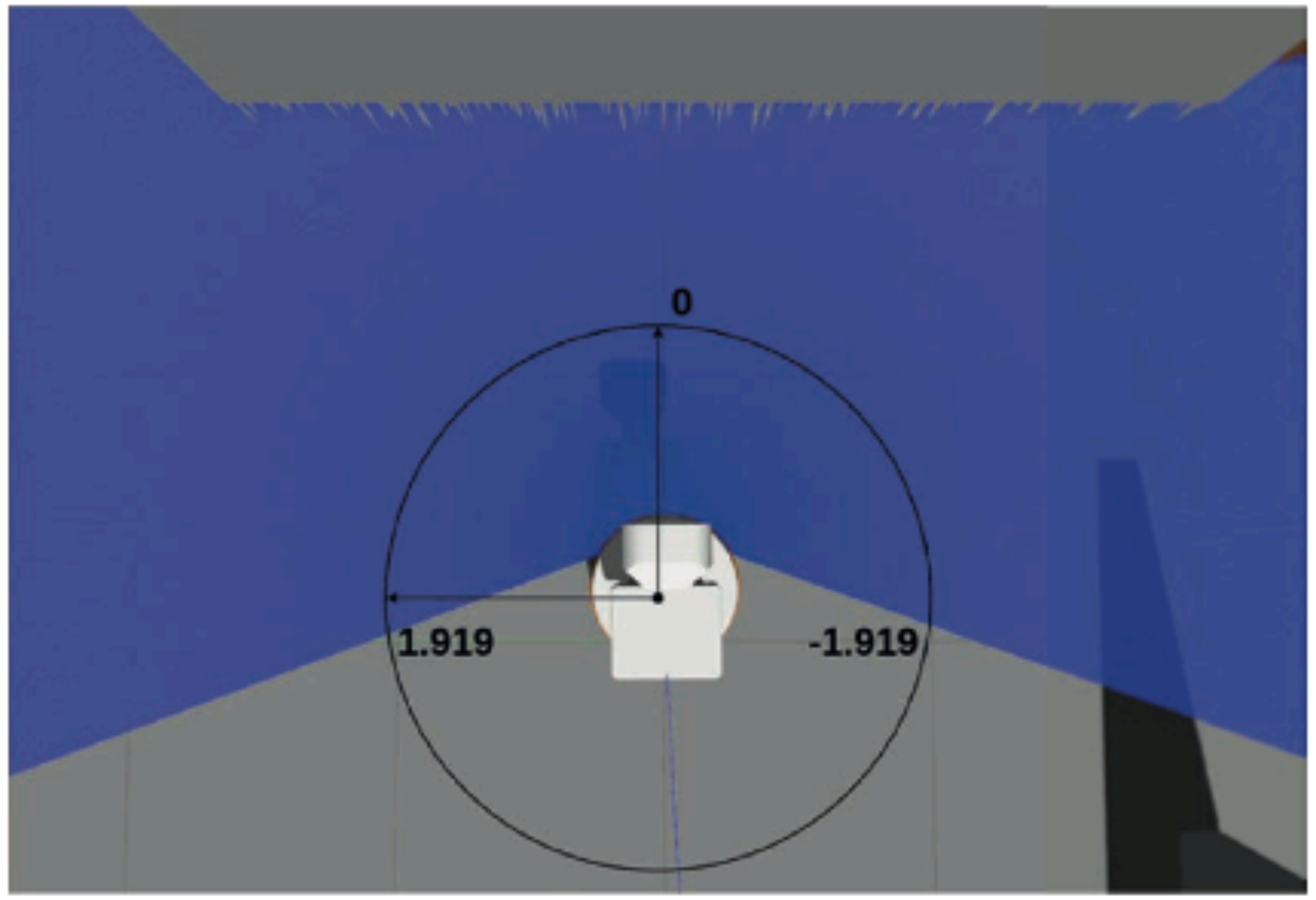
An Obstacle Detector that uses TF2

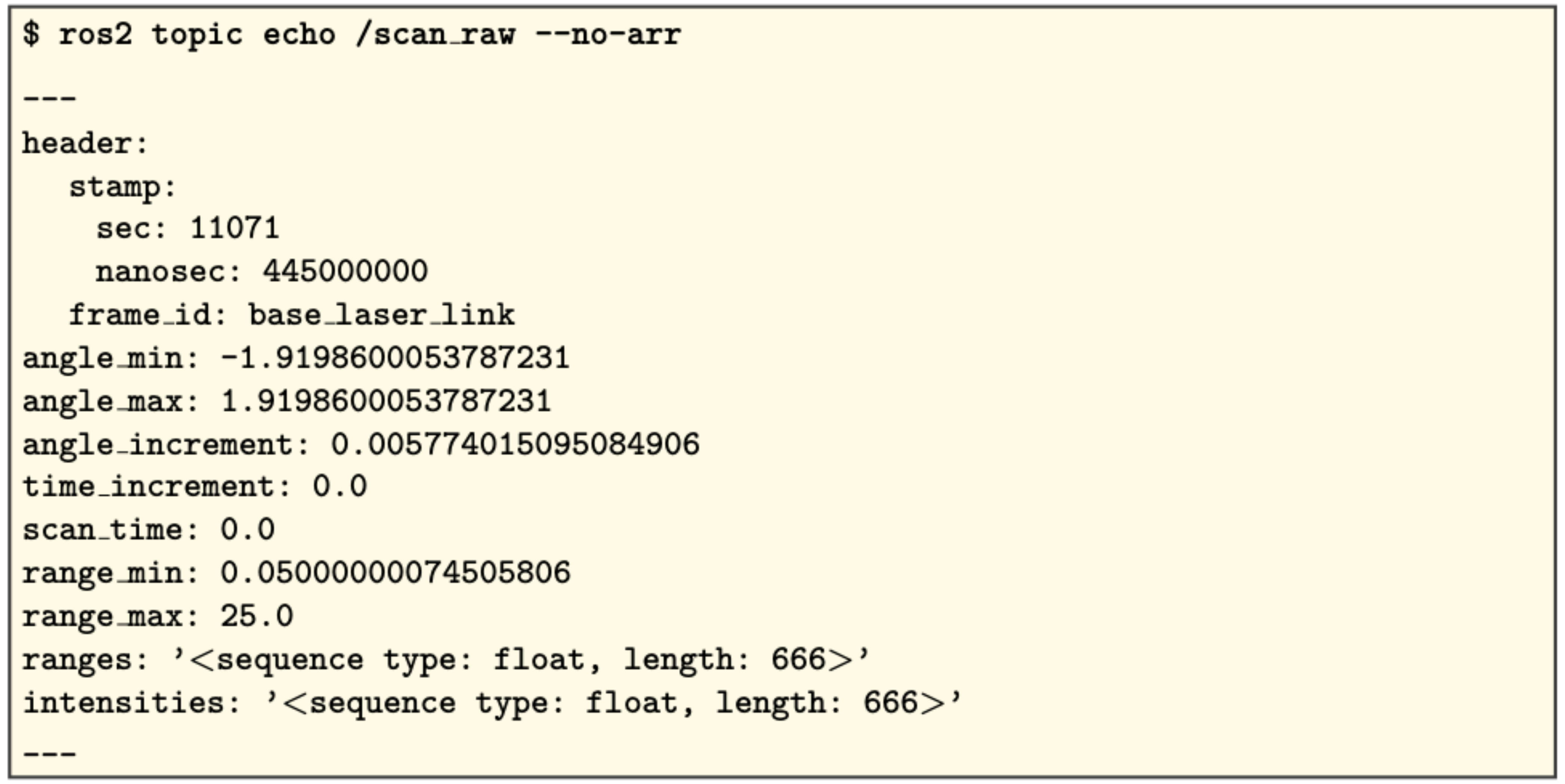
Obstacle Detector Node

- Obstacle is detected in front of the robot



```
void
ObstacleDetectorNode::scan_callback(sensor_msgs::msg::LaserScan::UniquePtr msg)
{
  double dist = msg->ranges[msg->ranges.size() / 2];

  if (!std::isinf(dist)) {
    geometry_msgs::msg::TransformStamped detection_tf;

    detection_tf.header = msg->header;
    detection_tf.child_frame_id = "detected_obstacle";
    detection_tf.transform.translation.x = msg->ranges[msg->ranges.size() / 2];

    tf_broadcaster_->sendTransform(detection_tf);
  }
}
```

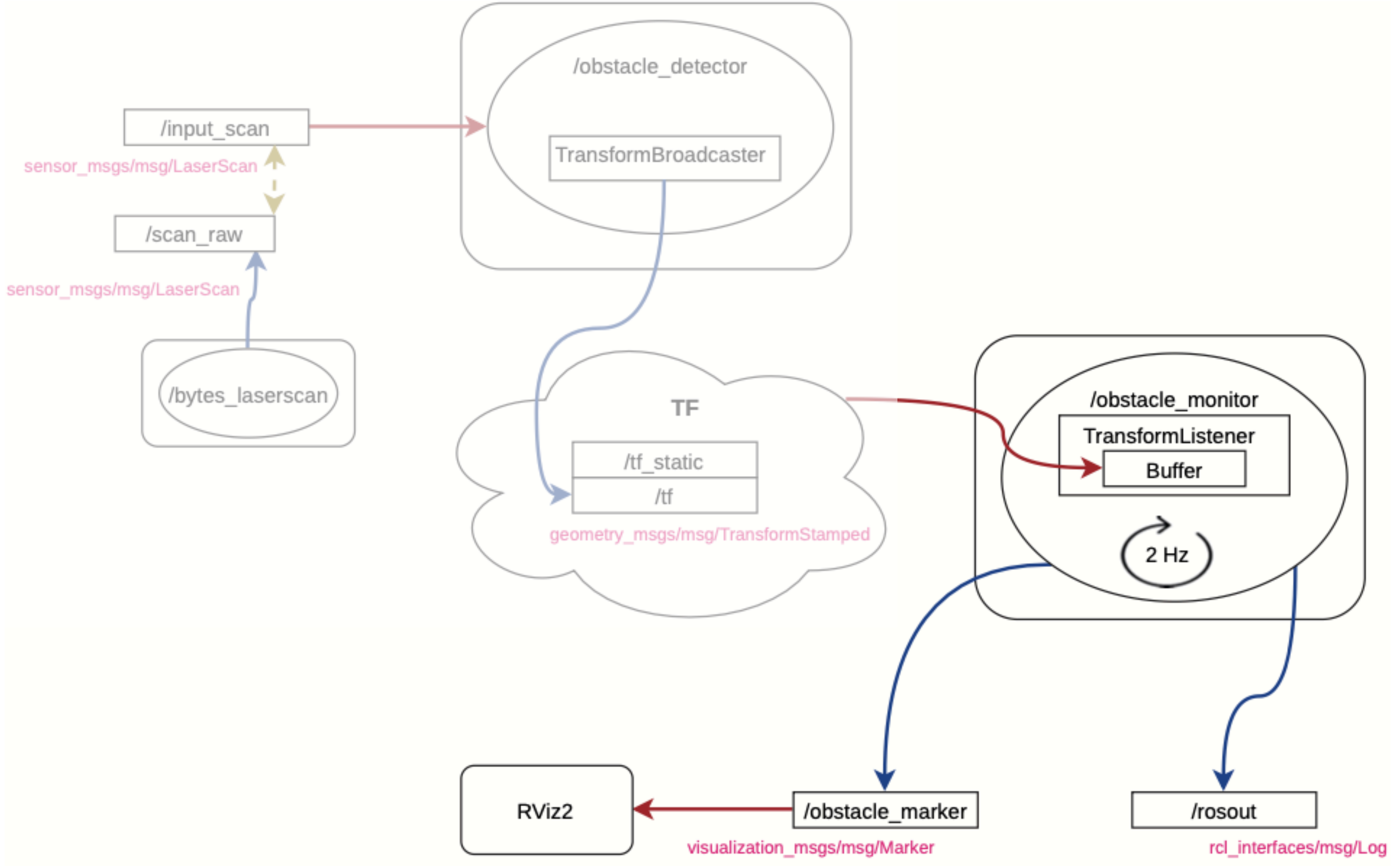




```
$ ros2 topic echo /scan_raw --no-arr
---
header:
  stamp:
    sec: 11071
    nanosec: 445000000
  frame_id: base_laser_link
angle_min: -1.9198600053787231
angle_max: 1.9198600053787231
angle_increment: 0.005774015095084906
time_increment: 0.0
scan_time: 0.0
range_min: 0.05000000074505806
range_max: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

An Obstacle Detector that uses TF2

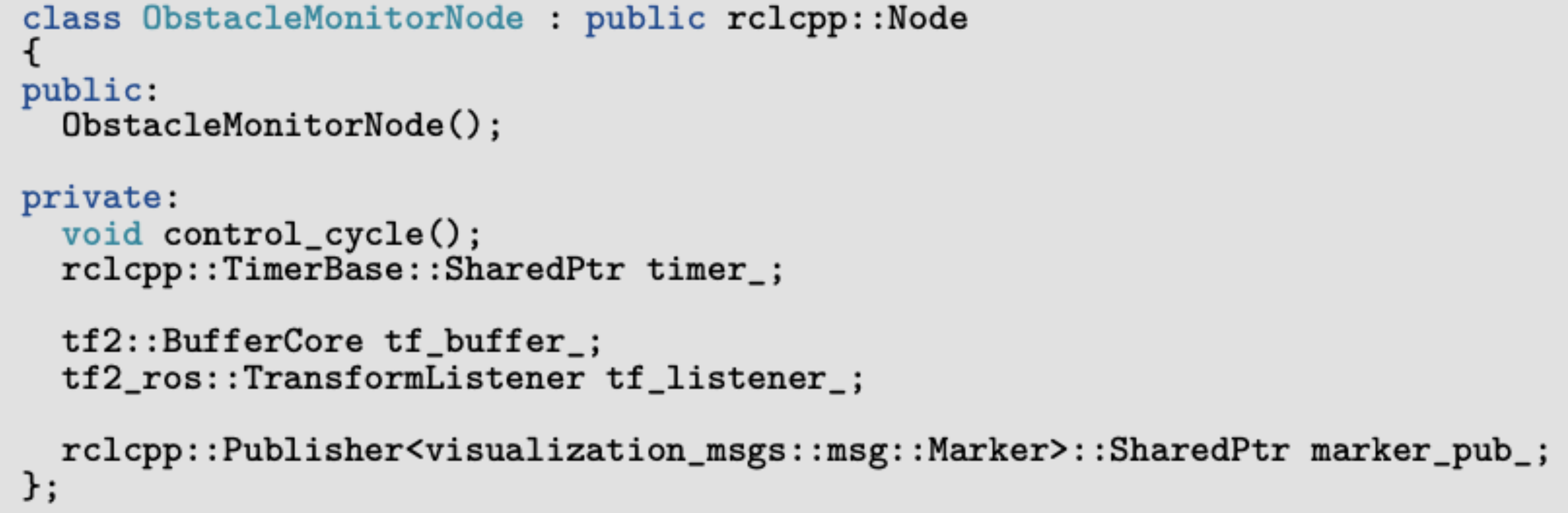
Obstacle Monitor Node



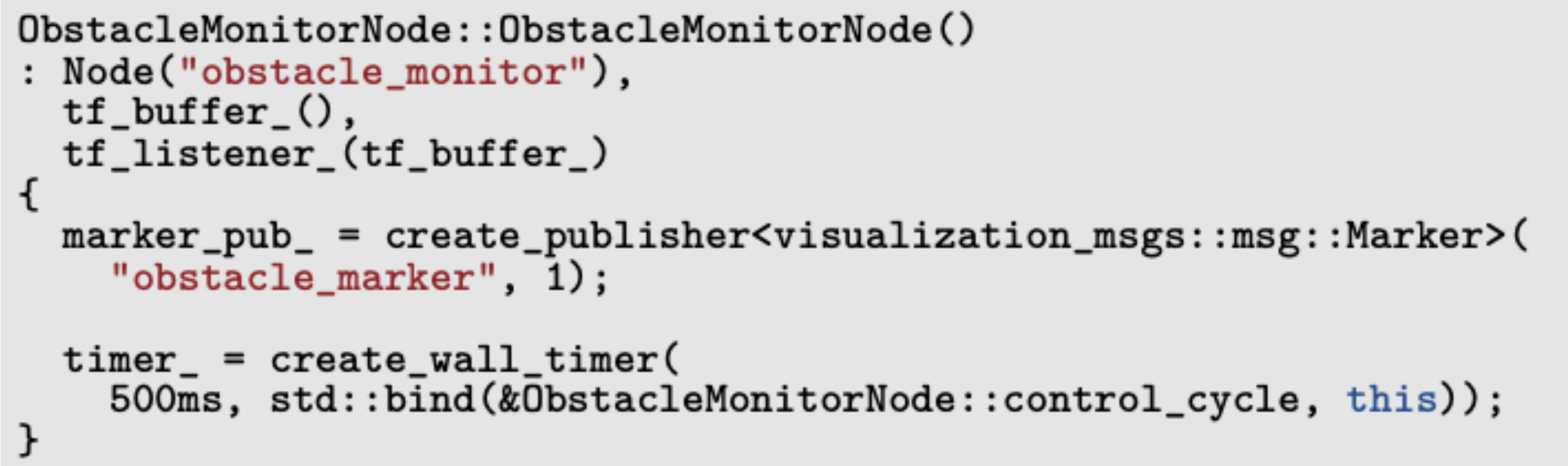
An Obstacle Detector that uses TF2

Obstacle Monitor Node

- It is necessary to declare a TF listener and a TF Buffer
- The listener updates the buffer
- Get the TF by querying the buffer



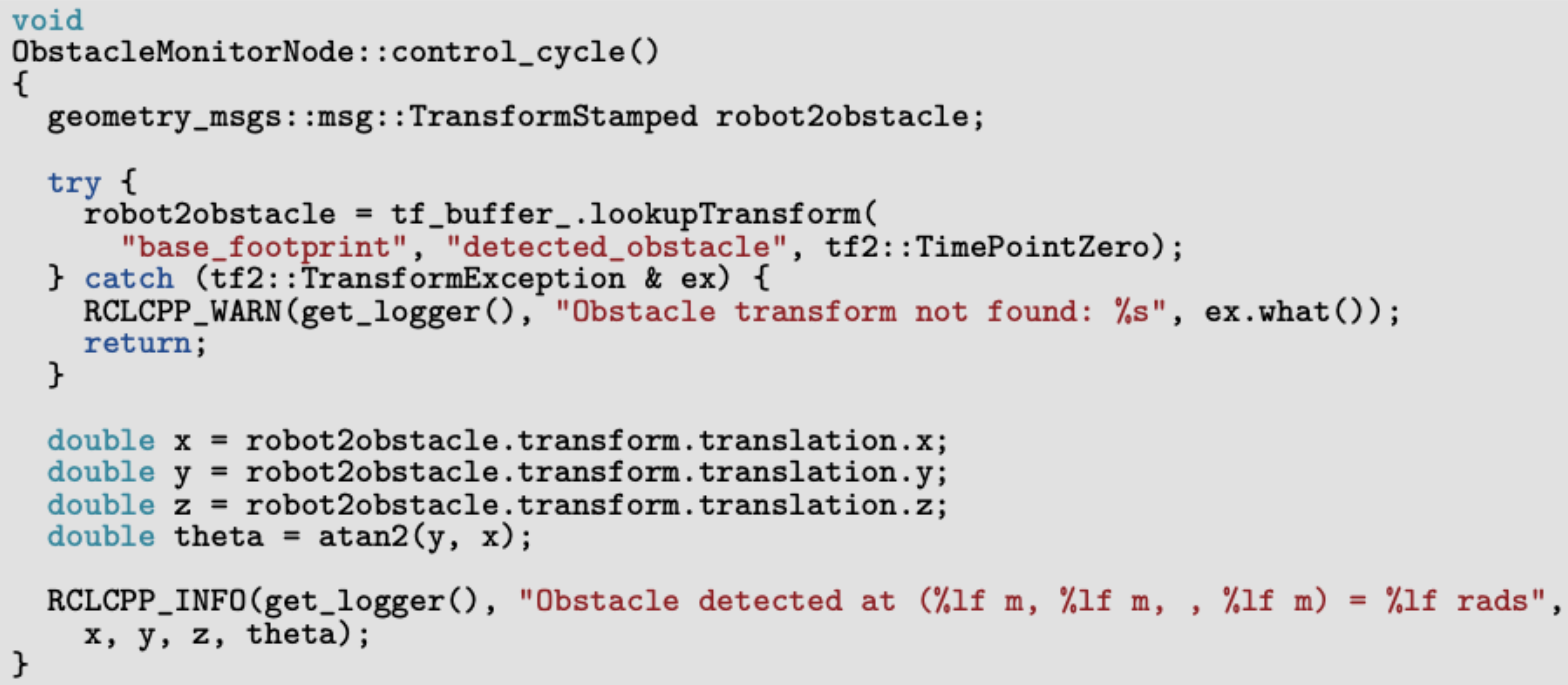
```
class ObstacleMonitorNode : public rclcpp::Node
{
public:
  ObstacleMonitorNode();

private:
  void control_cycle();
  rclcpp::TimerBase::SharedPtr timer_;

  tf2::BufferCore tf_buffer_;
  tf2_ros::TransformListener tf_listener_;

  rclcpp::Publisher<visualization_msgs::msg::Marker>::SharedPtr marker_pub_;
};
```



```
ObstacleMonitorNode::ObstacleMonitorNode()
: Node("obstacle_monitor"),
  tf_buffer_(),
  tf_listener_(tf_buffer_)
{
  marker_pub_ = create_publisher<visualization_msgs::msg::Marker>(
    "obstacle_marker", 1);

  timer_ = create_wall_timer(
    500ms, std::bind(&ObstacleMonitorNode::control_cycle, this));
}
```

An Obstacle Detector that uses TF2

Obstacle Monitor Node



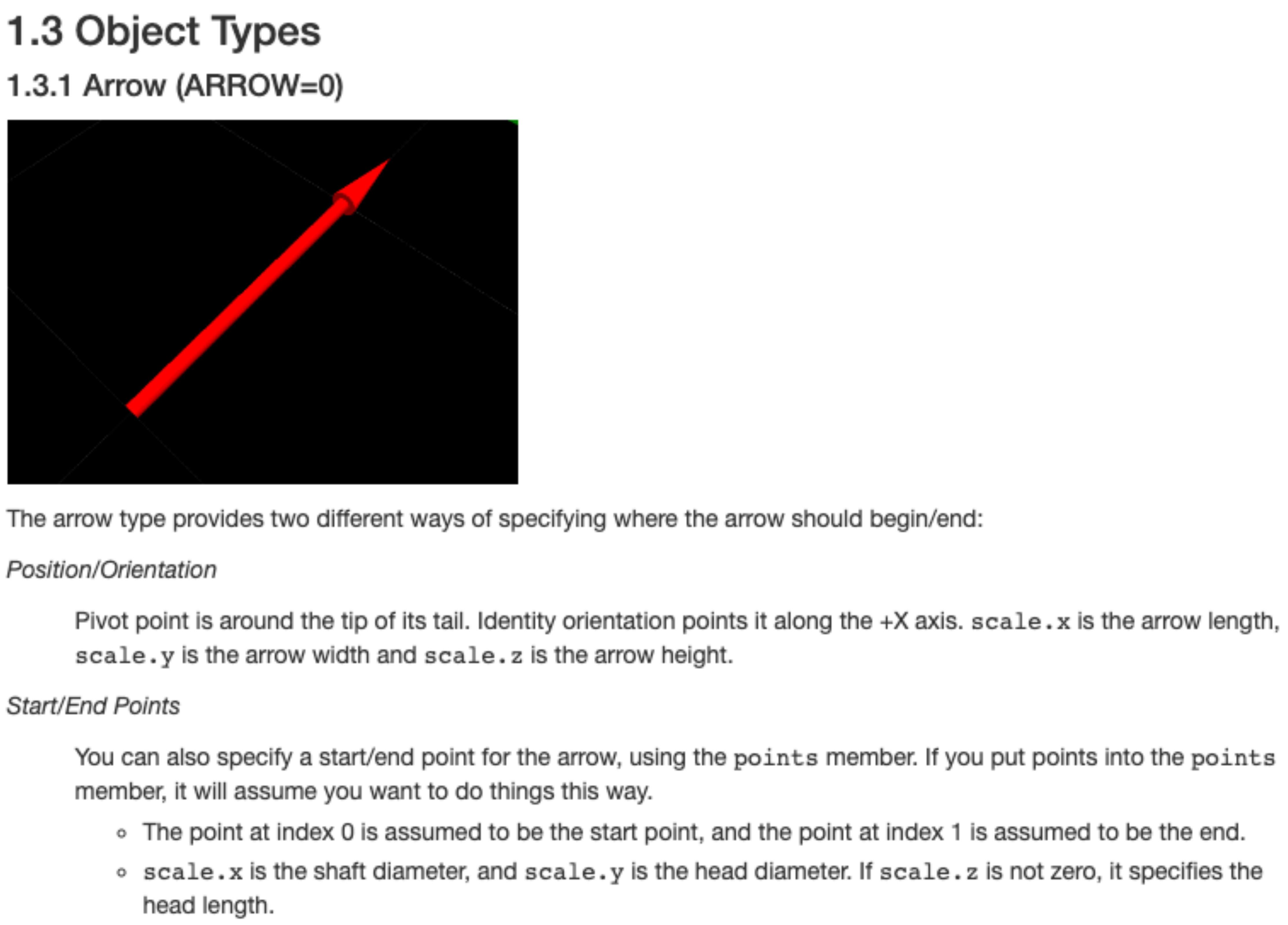
```
void
ObstacleMonitorNode::control_cycle()
{
  geometry_msgs::msg::TransformStamped robot2obstacle;

  try {
    robot2obstacle = tf_buffer_.lookupTransform(
      "base_footprint", "detected_obstacle", tf2::TimePointZero);
  } catch (tf2::TransformException & ex) {
    RCLCPP_WARN(get_logger(), "Obstacle transform not found: %s", ex.what());
    return;
  }

  double x = robot2obstacle.transform.translation.x;
  double y = robot2obstacle.transform.translation.y;
  double z = robot2obstacle.transform.translation.z;
  double theta = atan2(y, x);

  RCLCPP_INFO(get_logger(), "Obstacle detected at (%lf m, %lf m, , %lf m) = %lf rads",
    x, y, z, theta);
}
```

An Obstacle Detector that uses TF2

Obstacle Monitor Node

http://wiki.ros.org/rviz/DisplayTypes/Marker

```
visualization_msgs::msg::Marker obstacle_arrow;
obstacle_arrow.header.frame_id = "base_footprint";
obstacle_arrow.header.stamp = now();
obstacle_arrow.type = visualization_msgs::msg::Marker::ARROW;
obstacle_arrow.action = visualization_msgs::msg::Marker::ADD;
obstacle_arrow.lifetime = rclcpp::Duration(1s);

geometry_msgs::msg::Point start;
start.x = 0.0;
start.y = 0.0;
start.z = 0.0;
geometry_msgs::msg::Point end;
end.x = x;
end.y = y;
end.z = z;
obstacle_arrow.points = {start, end};

obstacle_arrow.color.r = 1.0;
obstacle_arrow.color.g = 0.0;
obstacle_arrow.color.b = 0.0;
obstacle_arrow.color.a = 1.0;

obstacle_arrow.scale.x = 0.02;
obstacle_arrow.scale.y = 0.1;
obstacle_arrow.scale.z = 0.1;
```



1.3 Object Types

1.3.1 Arrow (ARROW=0)

The arrow type provides two different ways of specifying where the arrow should begin/end:

Position/Orientation

Pivot point is around the tip of its tail. Identity orientation points it along the +X axis. `scale.x` is the arrow length, `scale.y` is the arrow width and `scale.z` is the arrow height.

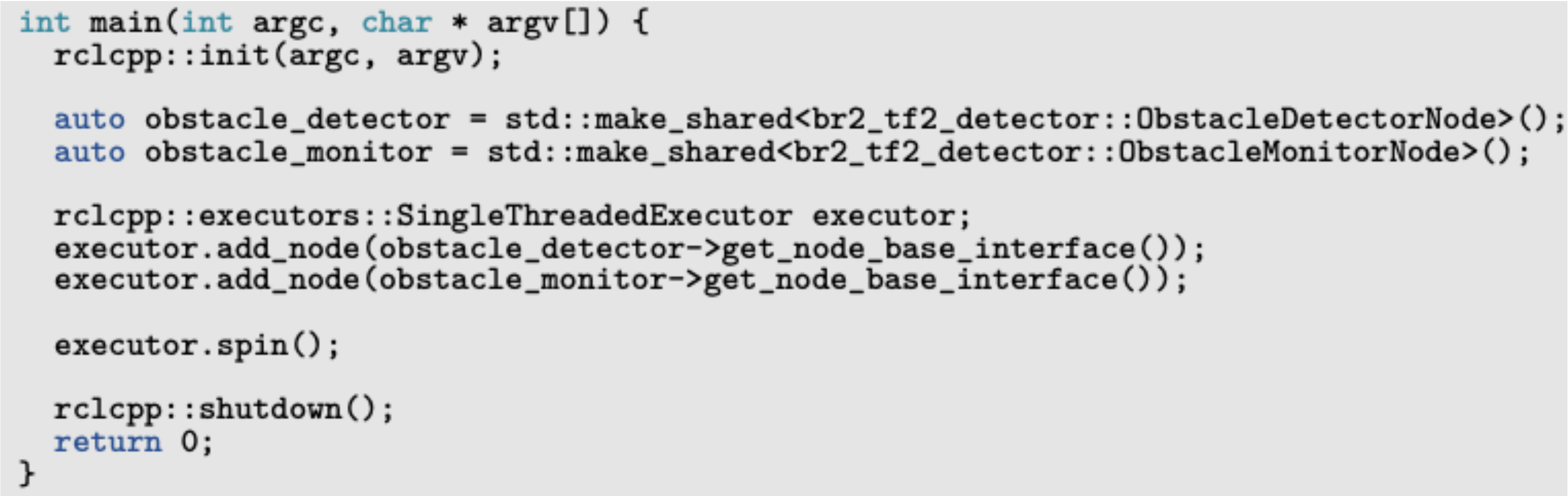
Start/End Points

You can also specify a start/end point for the arrow, using the `points` member. If you put points into the `points` member, it will assume you want to do things this way.

- The point at index 0 is assumed to be the start point, and the point at index 1 is assumed to be the end.
- `scale.x` is the shaft diameter, and `scale.y` is the head diameter. If `scale.z` is not zero, it specifies the head length.

An Obstacle Detector that uses TF2

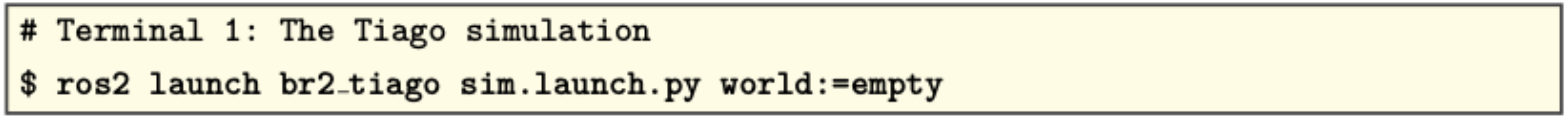
Running the code



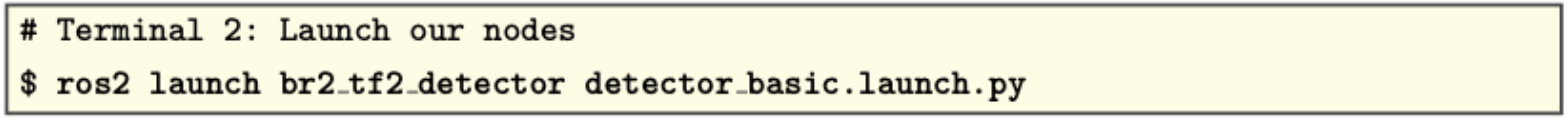
```
int main(int argc, char * argv[]) {
  rclcpp::init(argc, argv);

  auto obstacle_detector = std::make_shared<br2_tf2_detector::ObstacleDetectorNode>();
  auto obstacle_monitor = std::make_shared<br2_tf2_detector::ObstacleMonitorNode>();

  rclcpp::executors::SingleThreadedExecutor executor;
  executor.add_node(obstacle_detector->get_node_base_interface());
  executor.add_node(obstacle_monitor->get_node_base_interface());

  executor.spin();

  rclcpp::shutdown();
  return 0;
}
```

An Obstacle Detector that uses TF2

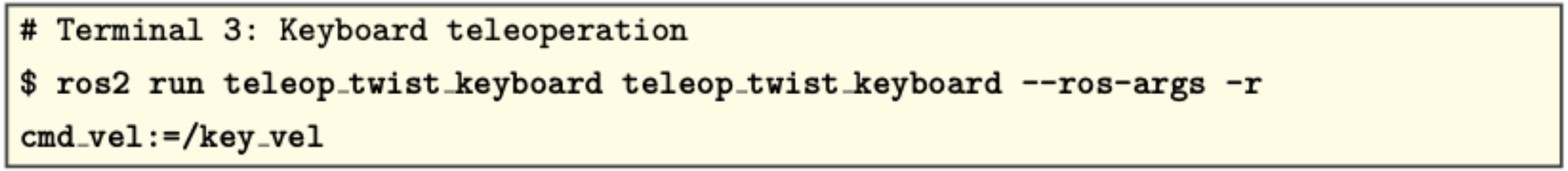
Running the code



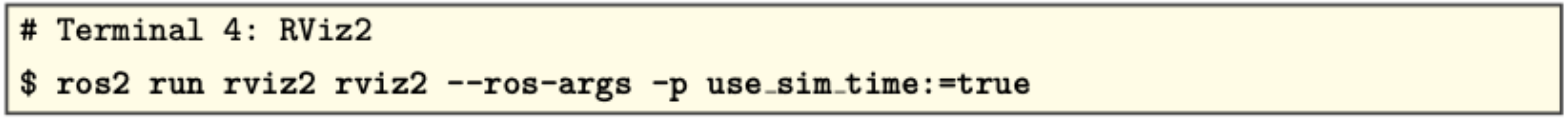
```
# Terminal 1: The Tiago simulation
$ ros2 launch br2_tiago sim.launch.py world:=empty
```



```
# Terminal 2: Launch our nodes
$ ros2 launch br2_tf2_detector detector_basic.launch.py
```



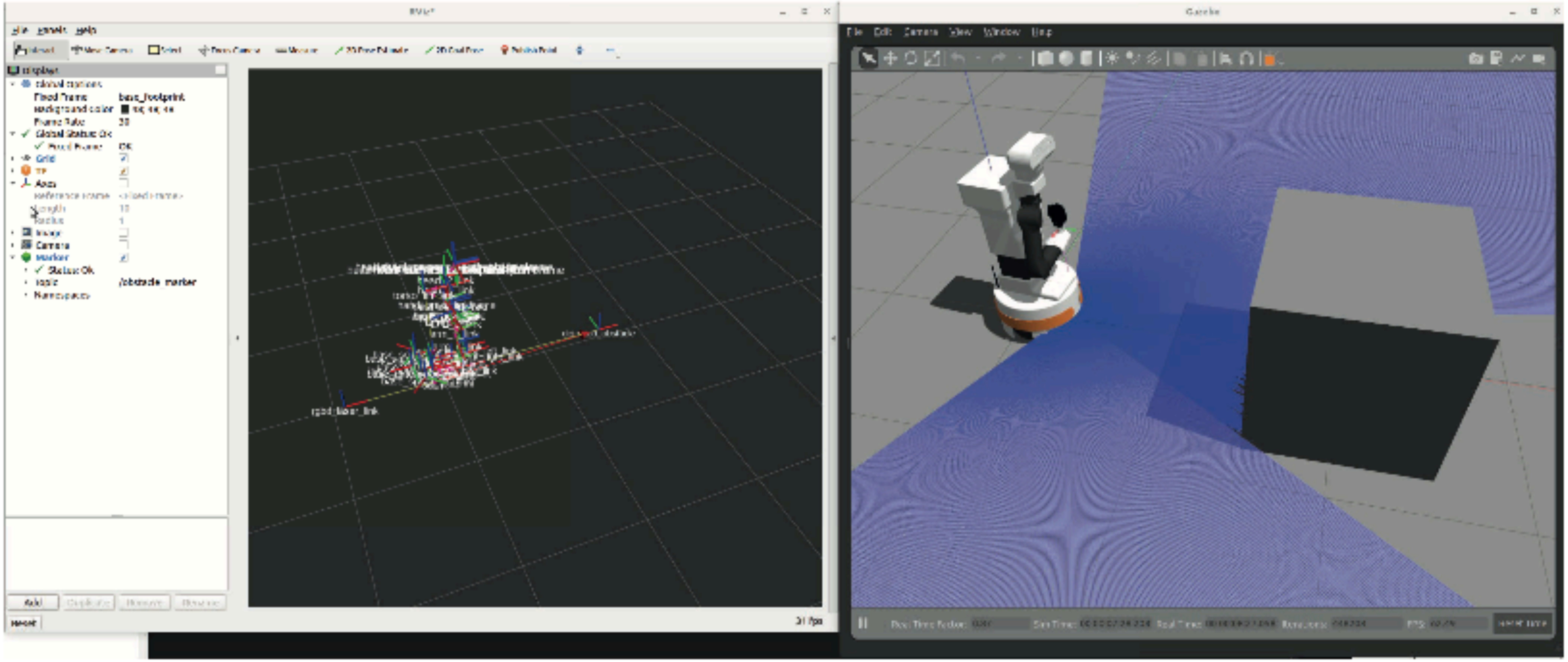
```
# Terminal 3: Keyboard teleoperation
$ ros2 run teleop_twist_keyboard teleop_twist_keyboard --ros-args -r
cmd_vel:=/key_vel
```



```
# Terminal 4: RViz2
$ ros2 run rviz2 rviz2 --ros-args -p use_sim_time:=true
```

An Obstacle Detector that uses TF2

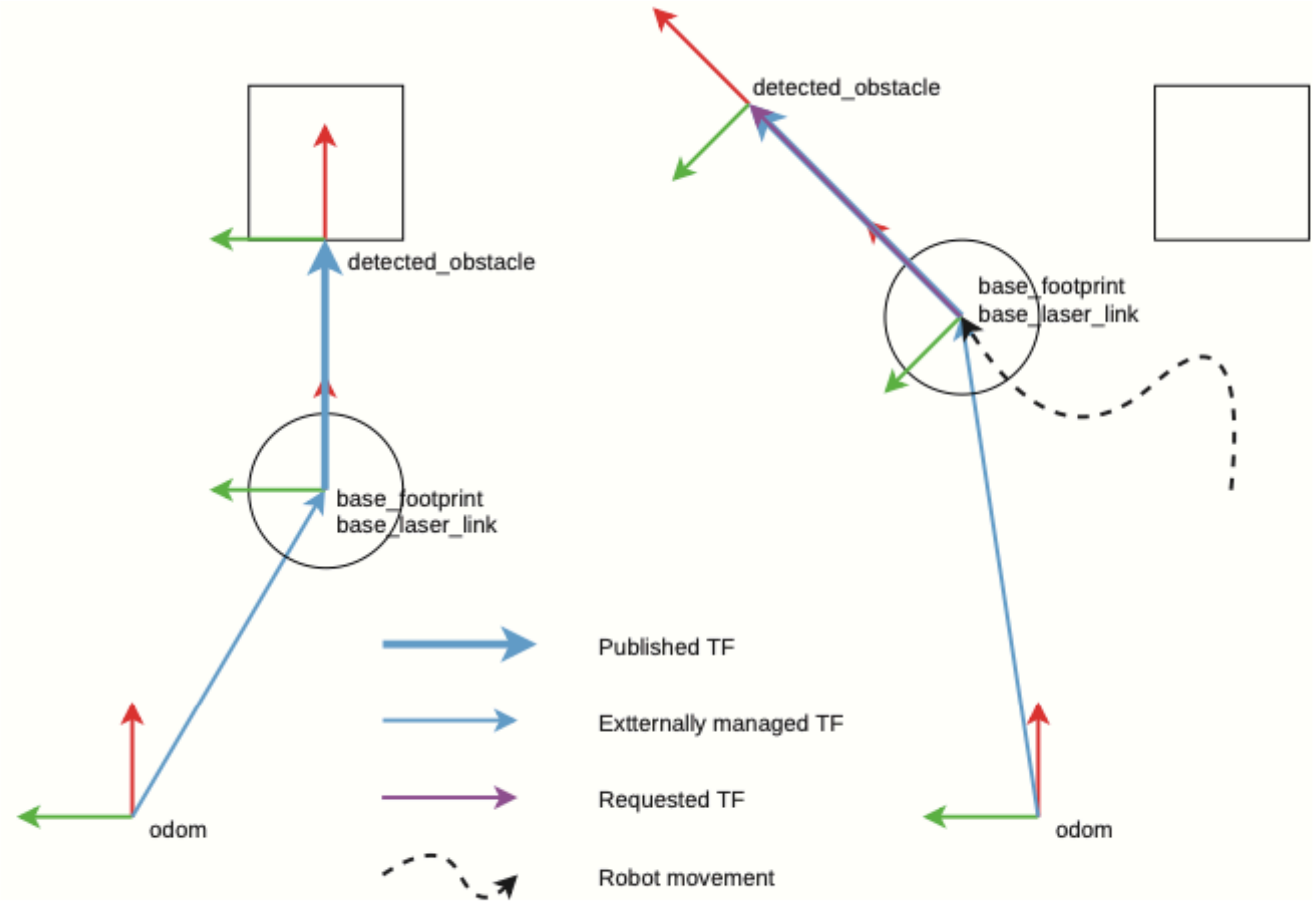
Running the code



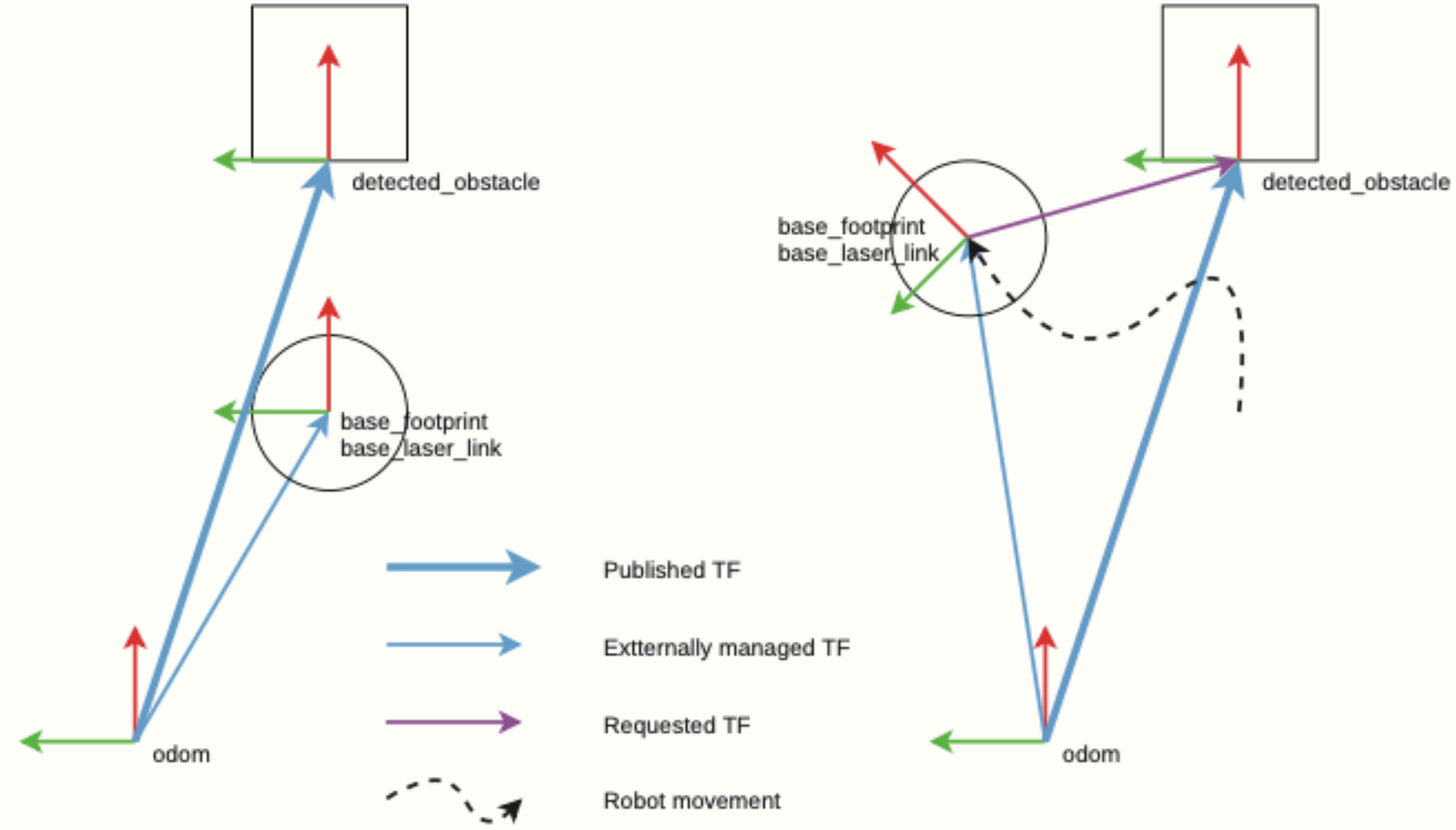
Fixed Frame base_footprint
Background Color 48; 48; 48
Frame Rate 30
Global Status: Ok
Fixed Frame OK
Grid ☑
TF ☑
Axes □
Reference Frame <Fixed Frame>
Length 10
Radius 1
Image □
Camera □
Marker ☑
Status: Ok
Topic /obstacle_marker
Namespaces

An Obstacle Detector that uses TF2

Problem

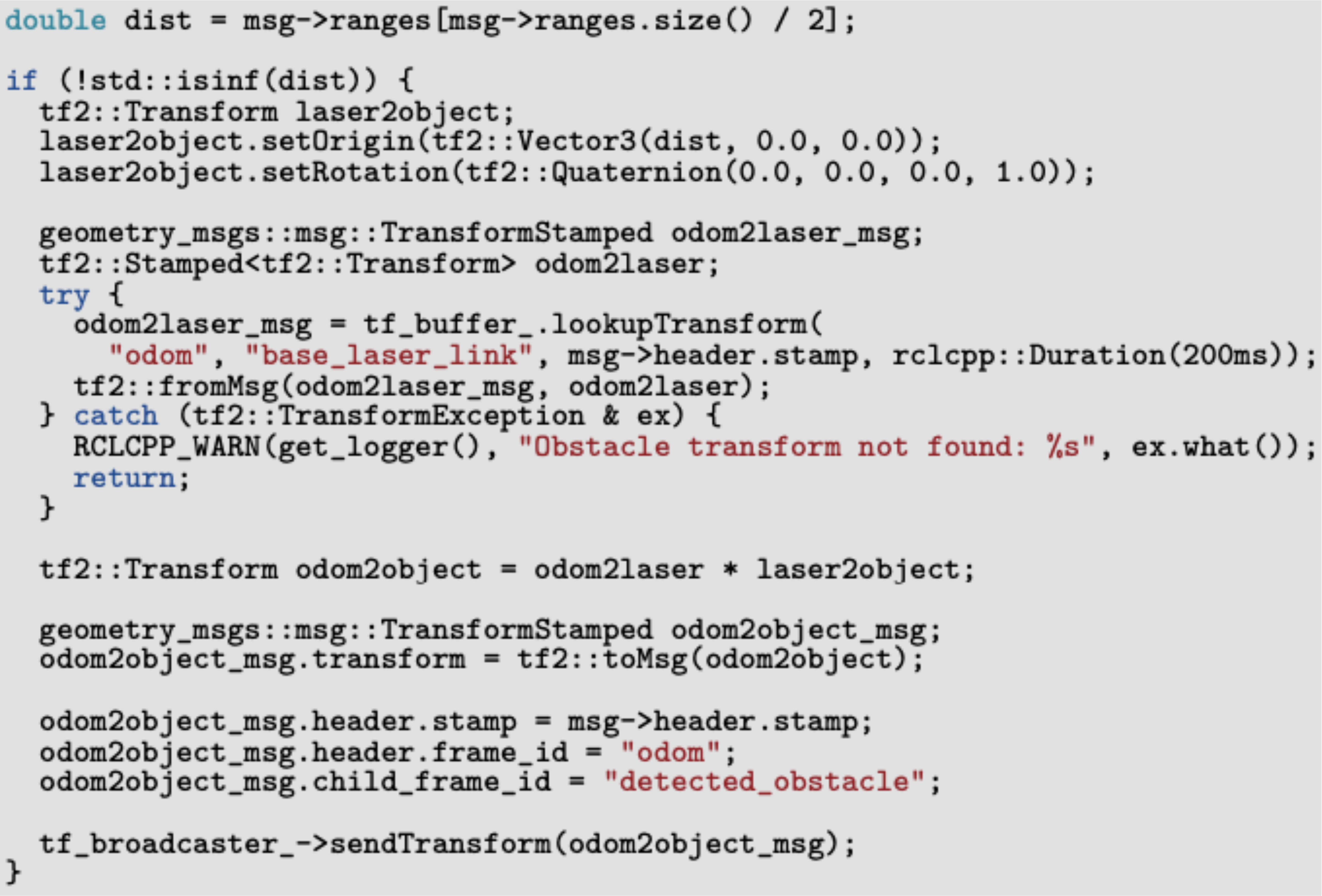


An improved Detector



An improved Detector

Msg types and TF2 library types



```
double dist = msg->ranges[msg->ranges.size() / 2];

if (!std::isinf(dist)) {
  tf2::Transform laser2object;
  laser2object.setOrigin(tf2::Vector3(dist, 0.0, 0.0));
  laser2object.setRotation(tf2::Quaternion(0.0, 0.0, 0.0, 1.0));

  geometry_msgs::msg::TransformStamped odom2laser_msg;
  tf2::Stamped<tf2::Transform> odom2laser;
  try {
    odom2laser_msg = tf_buffer_.lookupTransform(
      "odom", "base_laser_link", msg->header.stamp, rclcpp::Duration(200ms));
    tf2::fromMsg(odom2laser_msg, odom2laser);
  } catch (tf2::TransformException & ex) {
    RCLCPP_WARN(get_logger(), "Obstacle transform not found: %s", ex.what());
    return;
  }

  tf2::Transform odom2object = odom2laser * laser2object;

  geometry_msgs::msg::TransformStamped odom2object_msg;
  odom2object_msg.transform = tf2::toMsg(odom2object);

  odom2object_msg.header.stamp = msg->header.stamp;
  odom2object_msg.header.frame_id = "odom";
  odom2object_msg.child_frame_id = "detected_obstacle";

  tf_broadcaster_->sendTransform(odom2object_msg);
}
```

- **geometry msgs::msg::TransformStamped** is a message type, and is used to post TFs, and is the returned result of lookupTransform.

- **tf2::Transform** It is a data type of the TF2 library that allows to perform operations.

- **tf2::Stamped<tf2::Transform>** is similar to the previous one, but with a header that indicates a timestamp. It will be necessary to comply with the types in the transformation functions.

- **tf2::fromMsg/tf2::toMsg** are transformation functions that allow transforming from a message type to a TF2 type, and vice versa

Proposed Exercises

1. Make a node that shows every second how much the robot has moved. You can do this by saving $(\text{odom} \rightarrow \text{base footprint})_t$, and subtracting it from $(\text{odom} \rightarrow \text{base footprint})_{t+1}$

2. In ObstacleDetectorNode, change the arrow's color depending on the distance to the obstacle: green is far, and red is near.

3. In ObstacleDetectorNode, show in the terminal the obstacle's position in the odom frame, in base_footprint, and head_2_link.