Kinematics in ROS

Stefano Carpin

Department of Computer Science and Engineering
School of Engineering
University of California, Merced
https://sites.ucmerced.edu/scarpin
https//robotics.ucmerced.edu

Kinematics in ROS

- various packages devoted to kinematics
- `geometry_msgs` offers messages to represent and process quantities related to kinematics
- `tf2` implements transformation trees (and more)

ROS messages for Kinematics

Message Type	Quantity
`geometry_msgs::msg::Point`	Point in $\mathbb{R}^3$
`geometry_msgs::msg::PointStamped`	Point with time stamp
`geometry_msgs::msg::Quaternion`	Orientation
`geometry_msgs::msg::QuaternionStamped`	Orientation with time stamp
`geometry_msgs::msg::Pose`	Position and orientation (as quat.)
`geometry_msgs::msg::PoseStamped`	Pose with time stamp
`geometry_msgs::msg::Pose2D`	Pose in the plane, i.e., x, y, θ
`geometry_msgs::msg::Transform`	Transformation matrix
`geometry_msgs::msg::TransformStamped`	Transformation matrix with t.s.
`geometry_msgs::msg::Vector3`	Direction in space
`geometry_msgs::msg::Vector3Stamped`	Direction in space with time stamp
`geometry_msgs::msg::Twist`	Velocity (linear and angular)
`geometry_msgs::msg::TwistStamped`	Velocity with time stamp

Table: ROS messages to represent geometric data.

Time Stamps

- many messages have *stamped* version (e.g., `Vector3` and `Vector3Stamped`)
- useful for quantities changing over time
- stamped version includes a `std_msgs/Header` message in the beginning, e.g., `Vector3Stamped`) is

```
std_msgs/Header header
Vector3 vector
```

Headers

- `stamp`: temporal time stamp
- `frame_id`: name of the frame with respect to which this quantity is referred to

```
# Two-integer timestamp that is expressed as seconds and nanoseconds
builtin_interfaces/Time stamp

# Transform frame with which this data is associated.
string frame_id
```

geometry_msgs::msg::Pose2D

- convenience message type introduced specifically for robots moving on a plane (e.g., differential drive or skid-steer.)

```
float64 x
float64 y
float64 theta
```

- however, some mobile robots broadcast their pose as `geometry_msgs::msg::Pose`.
- use `ros2 topic list` to determine the format of the pose

Controlling a Differential Drive in ROS

- messages of type `geometry_msgs::msg::Twist` are used to specify velocities (e.g., to `cmd_vel` topics)

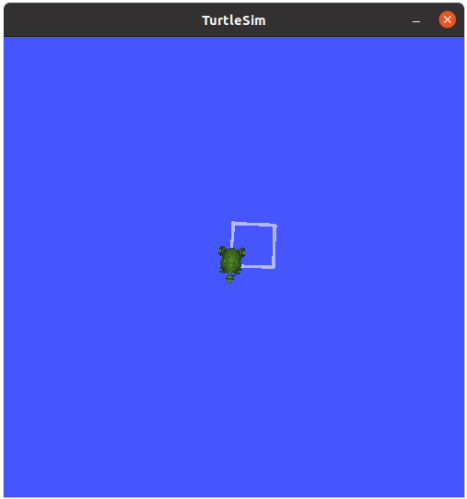
```
#This expresses velocity in free space broken into
# its linear and angular parts.

Vector3  linear
Vector3  angular
```

Recall how frames are attached to differential drives: `linear.x` is the translational speed, `angular.z` is the rotational speed

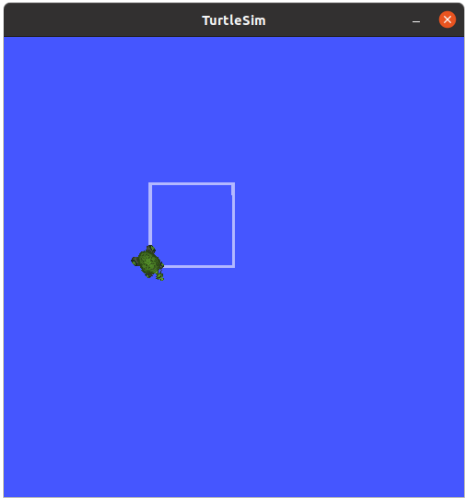
Controlling a Differential Drive

- Drawing a square in open loop: `drawsquare.cpp` ☹
- Drawing a square in closed loop: `drawsquarefb.cpp` ☺

The problem with open loop control



(Simple) closed loop control



A first glimpse into topic remapping

- `turtlesim` is controlled as a differential drive. Can I use the same code to control a differential drive robot?
- Yes, and it is not even necessary to recompile the code, thanks to *remapping*

```
ros2 run examples drawsquare --ros-args --remap
     turtle1/cmd_vel:=p2dx/cmd_vel
```

or

```
ros2 run examples drawsquare --ros-args --remap
     turtle1/cmd_vel:=p3at/cmd_vel
```

The transform library (package)

- two versions used, `tf` and `tf2` (the latter phasing out the former)
- Provides:
  - classes supporting geometric/kinematic concepts
  - static functions to perform geometric/kinematic operations
  - tools to track multiple coordinate frames changing over time (transform trees)
  - names standardization for common frames
- package `tf2_msgs` provides two messages: `TF2Error` and `TFMessage`

tf2 classes

Class	Description
tf2::Matrix3x3	Rotation Matrix
tf2::Quaternion	Quaternion
tf3::Vector3	Point or vector
tf2::Transform	Rigid transformation, i.e., rotation and translation

Table: tf2 classes to represent geometric data.

tf2::msg::TFMessage

- Structure of `tf2::msg::TFMessage`

  `geometry_msgs/TransformStamped[] transforms`
- Structure of `geometry_msgs::msg::TransformStamped`

```
#The frame id in the header is used as the reference frame
# of this transform.
std_msgs/Header header

#The frame id of the child frame to which
# this transform points.
string child_frame_id

#Translation and rotation in 3-dimensions of child_frame_id
# from header.frame_id
Transform transform
```

`tf2::msg::TFMessage` (cont'd)

- Structure of `geometry_msgs::msg::Transform`

```
# This represents the transform between two coordinate frames in

Vector3 translation
Quaternion rotation
```

- ${}^{B}_{A}\mathbf{T}$ will be represented by a message `TransformStamped` with `child_frame_id` set to A and `frame_id` set to B.

tf2::msg::TF2Error

```
uint8 NO_ERROR = 0
uint8 LOOKUP_ERROR = 1
uint8 CONNECTIVITY_ERROR = 2
uint8 EXTRAPOLATION_ERROR = 3
uint8 INVALID_ARGUMENT_ERROR = 4
uint8 TIMEOUT_ERROR = 5
uint8 TRANSFORM_ERROR = 6

uint8 error
string error_string
```

Quaternions and Rotations in ROS

- `geometry_msgs` does not provide a message for rotation matrices
  - rotations are always sent as quaternions
- `tf2::Quaternion` represents a quaternion
  - not a duplicate: the message has just data, the class includes useful methods, too
  - `getAngle` returns the angle associated with the quaternion
  - `getAxis` returns the axis associated with the quaternion
  - `setEulerZYZ` sets the quaternion to the rotation associated with a given triplet of Euler angles
  - `setRPY` sets the quaternion to the rotation associated with a given triplet of roll-pitch-yaw angles

Quaternions and Rotations in ROS

Useful static functions

- `tf2::getYaw`: accepts as parameter an instance of `tf2::Quaternion` and returns the associated yaw angle. This is extremely useful for mobile robots moving in the plane.
- `tf2::toMsg`: converts an instance of `tf2::Quaternion` into a message of type `geometry_msgs::Quaternion` so that it can be published (see also listing **??** for more details.)
- `tf2::fromMsg`: converts a message of type `geometry_msgs::Quaternion` into the equivalent class of the `tf2` package.

Example: `republishpose.cpp`

Quaternions and Rotations in ROS

- `tf2::Matrix3x3` represents a rotation matrix
- `setRotation` converts a quaternion into a rotation matrix
- `getRotation` returns the quaternion equivalent to a rotation matrix
- various other methods to compute inverse, transpose, etc, (see documentation)
- Examples: `geom.cpp` and `republishpose.cpp`

Frames in ROS

- exchanged as messages of type `geometry_msgs/TransformStamped`
- stamped because they can change over time
- two strings: `frame_id` in the header and `child_frame_id` in the message
  - ${}^{B}_{A}\mathbf{T}$ will be represented by a message `TransformStamped` with `child_frame_id` set to A and `frame_id` set to B.

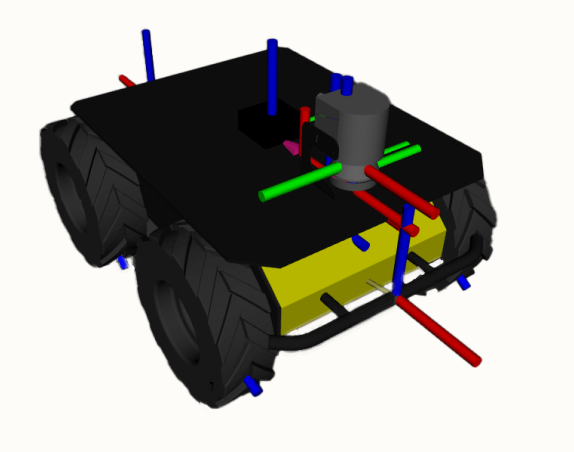
Transformation Trees in ROS

- handled by `tf2`
- allows to look at transformation *buffered in time*, i.e., at present time or in the past (up to 10 secs; configurable).
- to take advantage of the infrastructure, nodes must listen to or *broadcast* transformations
- often you can take advantage by just listening

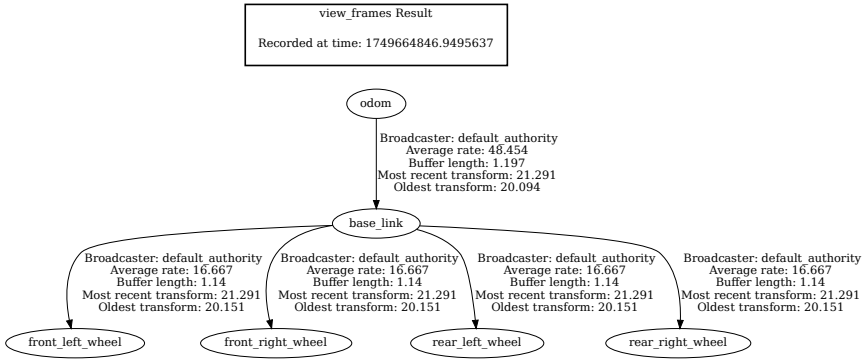
Transformation Trees in ROS

- Frames are exchanged through two topics: `/tf` and `/tf_static`
- as the name suggests, `/tf_static` is used for transformations that do not change over time
- Example: placement of frames on a robot

Transformation Trees in ROS



Transformation Trees in ROS



Interfacing with `tf2`

- use object of type `TransformListener` and `TransformBroadcaster`
- can also subscribe directly to topics, but more complex
- Example:`tflistner.cpp`
- To lookup in the past:
  `rclcpp::Time`
  `pastlookup=nodeh->get_clock()->now()-tf2::durationFromSec(3);`
- can also lookup transformation between arbitrary frames, provided the tree is connected
- Example: `tfbroadcaster.cpp`
- Visualizing the transformation tree:
  `ros2 run tf2_tools view_frame.py`

Broadcasting frames

```
transforms:
- header:
stamp:
sec: 1673894532
nanosec: 133698111
frame_id: base_link
child_frame_id: myframe
transform:
translation:
x: 4.0
y: 0.0
z: 2.0
rotation:
x: 0.0
y: 0.0
z: 0.0
w: 1.0
```

Standard Frames in ROS

`base_link` : rigidly attached to the robot (typically center of mass); ***x*** points forward, ***z*** points up;

`odom` : world fixed frame; pose expressed in this frame is continuous and can drift over time;

`map` : world fixed frame; pose expressed in this frame can be discontinuous and should not drift over time;

`earth` : world frame with origin at the center of the earth; used only in special cases

- *world frames* do not change over time