Perception

Stefano Carpin

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

Why sensors?

- robots do not have all information available upfront and must acquire it on the go
- outcome of actions is not predictable; perception can assess deviations from desired outcomes
- time invariant, discrete, error-free sensor abstractions

$$\mathbf{z}_t = h(\mathbf{x}_t, \mathbf{u}_t) \qquad \mathbf{z}_t = h(\mathbf{x}_t)$$

- sensor reading at time t is a function of the state $\mathbf{x}$ at time t and implicitly of the environment.
- if the environment is static (an assumption we will often make), then it can be modeled into h

Sensor models

- sensors are noisy, too
- different models for sensor readings

$$\mathbf{z}_t = h(\mathbf{x}_t, \mathbf{u}_t, \psi_t)$$

- additive noise model; h_c is the *correct* sensor reading that we cannot access

$$h(\mathbf{x}_t, \mathbf{u}_t, \psi_t) = h_c(\mathbf{x}_t, \mathbf{u}_t) + \psi_t$$

- In estimation theory we will study how to handles these error sources; here we focus on the ROS API to retrieve sensor reading values.

Dead Reckoning

- class of sensors producing an estimate at time t by combining initial condition with temporal integration

$$\mathbf{x}(t) = \mathbf{x}(0) + \int_0^t \dot{\mathbf{x}}(\nu) d\nu$$

- not strictly adherent to the model we saw before
- initial condition must be given (often arbitrarily set)
- error grows over time
- typical example: wheel encoders

Types of Sensors

Proprioceptive return information about the robot itself

- encoders, accelerometers, gyroscopes, inertial measurement units, etc.

Exteroceptive return information about the environment

- sonars, laser range finders, 3D scanners/3D cameras, cameras, GPS, contact sensors

Sensors in ROS

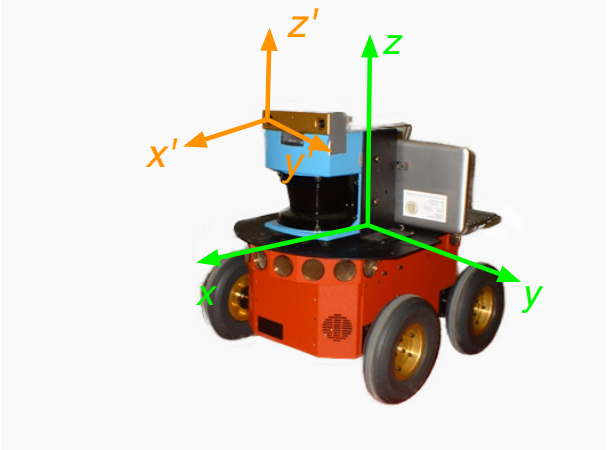
- most sensors come with ROS nodes to interact with them (*drivers*)
- information coming from a sensor is made available through topics
- your code typically does not interact with the sensor (the driver does), but rather gets its information from the topic)
- messages sent through sensor related topics are part of the `sensor_msgs` package

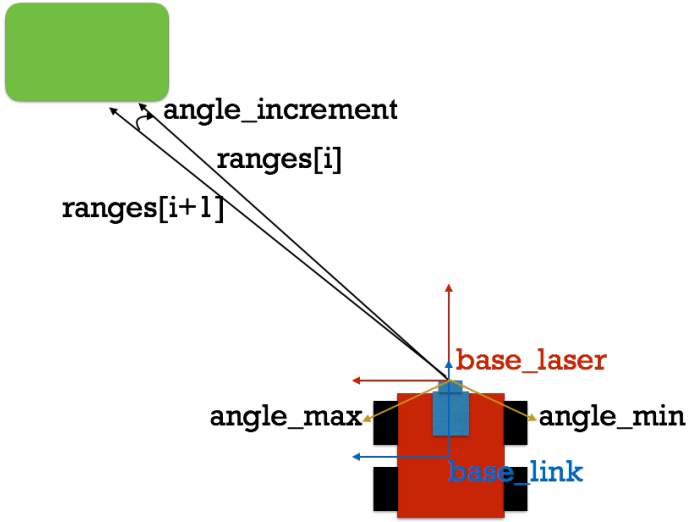
Sensors in ROS

Message Type	Sensor
`BatteryState`	Status of battery (e.g., voltage and other info)
`Image`	Camera image
`Imu`	Inertial Measurement Unit
`JointState`	Status of set of joints (position, velocity, torque)
`LaserScan`	Planar range finder (e.g., laser)
`NavSatFix`	GPS
`PointCloud`	Collection of 3D points returned by a 3D scanner
`PointCloud2`	Collection of 3D points returned by a 3D scanner (2nd version)
`Range`	Single range reading (e.g., sonar)

Exception: `nav_msgs::msg::Odometry`

LaserScan



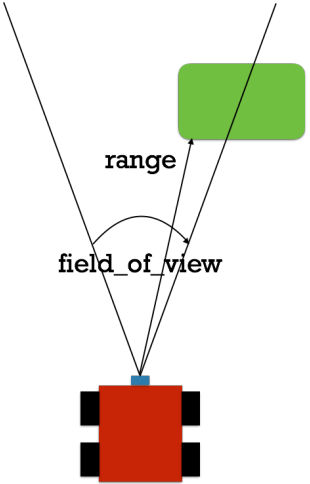

angle_increment
ranges[i]
ranges[i+1]
base_laser
angle_max
angle_min
base_link

sensor_msgs::msg::LaserScan

```
std_msgs/Header header
float32 angle_min
float32 angle_max
float32 angle_increment
float32 time_increment
float32 scan_time
float32 range_min
float32 range_max
float32[] ranges
float32[] intensities
```

Code example: `pubsubstl.cpp`

Sonar



sensor_msgs::msg::Range

```
uint8 ULTRASOUND=0
uint8 INFRARED=1
std_msgs/Header header
uint8 radiation_type
float32 field_of_view
float32 min_range
float32 max_range
float32 range
float variance
```

IMU

- wide variety of sensors in this category
- `sensor_msgs::msg::Imu` aims at including fields for most IMUs; not all may be used in practice
- covariance information included to assess precision

sensor_msgs::Imu

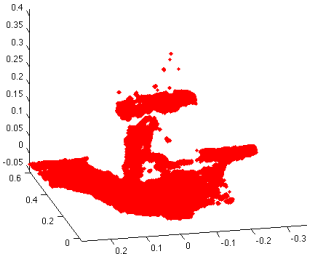
```
Header header

geometry_msgs/Quaternion orientation
float64[9] orientation_covariance

geometry_msgs/Vector3 angular_velocity
float64[9] angular_velocity_covariance

geometry_msgs/Vector3 linear_acceleration
float64[9] linear_acceleration_covariance
```

GPS: sensor_msgs::msg::NavSatFix

```
std_msgs/Header header
NavSatStatus status
float64 latitude
float64 longitude
float64 altitude
float64[9] position_covariance
uint8 COVARIANCE_TYPE_UNKNOWN=0
uint8 COVARIANCE_TYPE_APPROXIMATED=1
uint8 COVARIANCE_TYPE_DIAGONAL_KNOWN=2
uint8 COVARIANCE_TYPE_KNOWN=3
uint8 position_covariance_type
```

PointCloud

- two versions (PointCloud and PointCloud2)
- can be produced by a variety of sensors (e.g., Velodyne, RealSense, ...)
- ROS can be interfaced with external libraries (e.g., PCL) to utilize read-to-use algorithms



PointCloud

```
std_msgs/Header header
geometry_msgs/Point32[] points
  float32 x
  float32 y
  float32 z
sensor_msgs/ChannelFloat32[] channels
  string name
  float32[] values
```

PointCloud2

```
std_msgs/Header header
uint32 height
uint32 width
PointField[] fields
bool is_bigendian
uint32 point_step  # Length of a point in bytes
uint32 row_step  # Length of a row in bytes
uint8[] data    # Actual point data, size is (row_step*height)
bool is_dense  # True if there are no invalid points
```

Odometry

- derived from robot geometry and commands given to the robot
- provided by the navigation stack; hence via the `nav_msgs` package

nav_msgs::msg::Odometry

```
std_msgs/Header header

# Frame id the pose points to. The twist is in this coordinate frame
string child_frame_id

# Estimated pose that is typically relative to a fixed world frame.
geometry_msgs/PoseWithCovariance pose

# Estimated linear and angular velocity relative to child_frame_id.
geometry_msgs/TwistWithCovariance twist
```