Capítulo 3: Percepción robótica y representación espacial

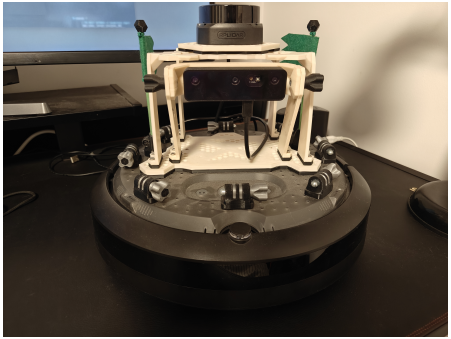
Sensores, marcos de referencia y TF en ROS 2

Francisco Martín Rico y Rodrigo Pérez Rodríguez

Sensores crudos

Mediciones directas y encapsulación

- Único contacto físico con el entorno: **mediciones digitalizadas** sin estructura ni interpretación.
- Responsabilidad: **adquirir, encapsular y publicar** datos tal como los entrega el hardware.
- Gestiona ciclo de vida y configuración (resolución, rango, frecuencia), aislando detalles del dispositivo.
- Debe mantener un **contrato de datos estable** para permitir sustituir hardware.



Robot iCreate con LIDAR 2D y cámara RGB-D.

Sensores crudos

Calibración intrínseca y extrínseca

- Los sensores introducen **sesgos sistemáticos**; calibrar es estimar y compensar esos parámetros.
- **Intrínseca:** propiedades internas (escala, distorsión óptica, ruido característico).
- **Extrínseca:** transformación rígida entre marco del sensor y marco del robot.
- Arquitectónicamente, los parámetros deben ser **configuración explícita** de la capa sensorial.

Calibración de cámara monocular (ROS).

Percepción como transformación de datos

De $\mathcal{D}_0$ a representaciones estructuradas

- Percepción: **transforma** datos sensoriales en representaciones con mayor estructura.
- Operaciones típicas: cambio de marco, extracción de características, agrupamiento, estimación de estado, mapas.
- Suele organizarse como **pipeline modular** (validable por etapas) y a veces incluye realimentación.
- Reduce dimensionalidad para hacer la información **computacionalmente tratable**.

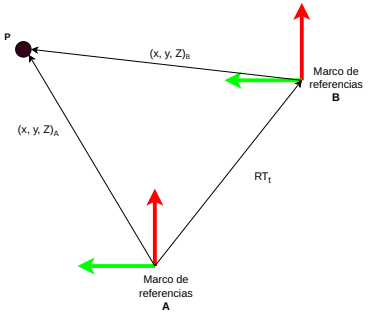
$$\mathcal{T} : \mathcal{D}_0 \rightarrow \mathcal{D}_1$$

La salida tiene más organización interna que la entrada.

Marco espacial y referencias

Coordenadas dependen del marco

- El razonamiento espacial exige **marcos de referencia** explícitos (origen + ejes ortogonales).
- El mismo punto físico se expresa con **coordenadas distintas** según el marco.
- La arquitectura debe evitar suposiciones implícitas: las referencias son parte del contrato geométrico.



Marco espacial y referencias

Transformación rígida y forma homogénea

- Entre marcos rígidos: rotación $R \in SO(3)$ + traslación t.
- Representación homogénea (4×4) permite composición y cambio de marco consistente.
- Cambio de referencia: multiplicación matricial.

$$p_B = T_{B \leftarrow A}\, p_A$$

$$T = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Marco espacial y referencias

Composición, tiempo y árbol

- **Composición:** garantiza coherencia geométrica.
- Transformaciones **dependen del tiempo**: $T(t)$ se conoce en instantes discretos e interpola entre muestras.
- Organización preferible: **árbol dirigido** con camino único entre marcos (sin ambigüedad).
- Debe existir un **servicio estructural transversal** para mantener el árbol y resolver consultas.

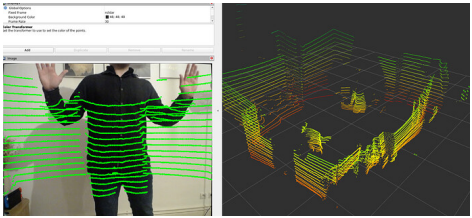
$$T_{C \leftarrow A} = T_{C \leftarrow B}\, T_{B \leftarrow A}$$

$$T_{B \leftarrow A} = T_{B \leftarrow W}\, T_{W \leftarrow A}$$

Fusión sensorial básica

Coherencia espacial y temporal

- Integra observaciones de **múltiples sensores** en una representación coherente.
- Requisitos: **marco común** (coherencia espacial) + **alineamiento temporal** (latencias/frecuencias).
- En nivel básico: alinear y superponer, sin inferencia estadística compleja.
- La arquitectura define políticas de sincronización: más reciente, interpolar o descartar fuera de ventana.



Gestión de sensores en ROS 2

Nodos, topics y mensajes estándar

- Cada sensor se encapsula en un **nodo** que adquiere del hardware y publica periódicamente en **topics**.
- Se separa adquisición de percepción: el nodo del sensor publica **datos crudos** sin interpretación.
- ROS promueve **tipos estándar** (p.ej. `sensor_msgs/LaserScan`, `Image`, `Imu`).
- La estandarización desacopla hardware y algoritmos consumidores.

Gestión de sensores en ROS 2

Calidad de Servicio (QoS): políticas y compatibilidad

- La QoS gobierna fiabilidad, histórico/cola, durabilidad y conectividad.
- Publicador **ofrece** un perfil; suscriptor **solicita** otro.
- La comunicación solo existe si son **compatibles**; la QoS efectiva es la intersección.
- En **Reliability**, el publicador fija el límite superior de garantías.

Publicador	Suscriptor	QoS efectiva
Best Effort	Best Effort	Best Effort
Best Effort	Reliable	Sin conexión
Reliable	Best Effort	Best Effort
Reliable	Reliable	Reliable

Gestión de sensores en ROS 2

QoS: Durability y efectos arquitectónicos

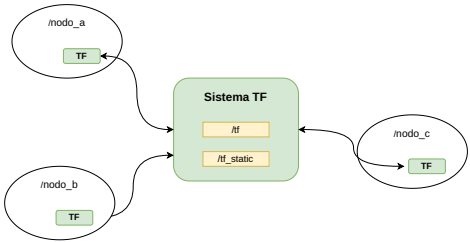
- **Volatile:** no almacena; suscriptores tardíos no reciben pasado.
- **Transient Local:** conserva últimos mensajes y los entrega a nuevos suscriptores.
- Útil para información estructural (p.ej., TFs estáticas); sensores de alta frecuencia suelen ser *volatile*.
- La QoS condiciona rendimiento y también **conectividad** efectiva.

Publicador	Suscriptor	QoS efectiva
Volatile	Volatile	Volatile
Volatile	Transient Local	Sin conexión
Transient Local	Volatile	Volatile
Transient Local	Transient Local	Transient Local

Sistema de transformaciones (TF) en ROS 2

Infraestructura común para marcos

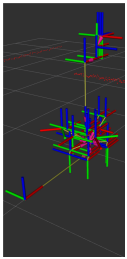
- Materializa matrices homogéneas, tiempo y árbol de marcos.
- Los nodos **no intercambian** TFs punto a punto: publican y consultan una infraestructura común.
- Objetivo: resolver transformaciones de forma consistente y reutilizable.

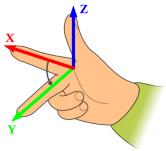


Sistema de transformaciones (TF) en ROS 2

Fuentes de transformaciones

- **Estructurales:** geometría del robot (URDF/SDF) + `robot_state_publisher`.
- **Dinámicas:** odometría, localización, estimadores y controladores.
- En simulación, el simulador publica TFs dinámicas igual que el hardware real.
- Todas las fuentes alimentan el **mismo árbol** TF.

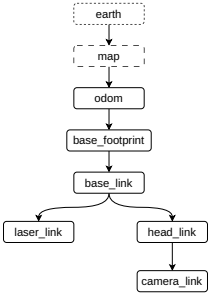




Sistema de transformaciones (TF) en ROS 2

Convenciones estándar: REP 105

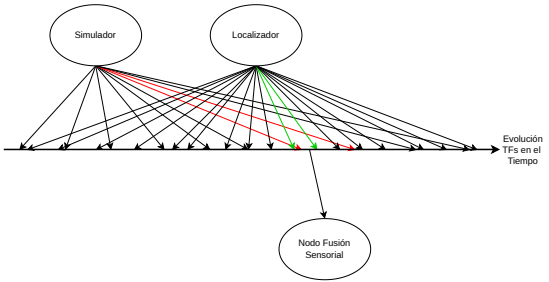
- Convención semántica de marcos para interoperabilidad.
- `map`: global (corrección absoluta) | `odom`: local continuo (acumula error).
- `base_link`: pose 3D completa | `base_footprint`: proyección planar ($SE(2)$).
- Reduce ambigüedades y acoplamientos implícitos entre módulos.



Sistema de transformaciones (TF) en ROS 2

Publicación y recepción de TF

- Topics: **/tf** (dinámicas) y **/tf_static** (estáticas, una sola vez).
- Cada nodo mantiene un **buffer temporal** (histórico finito) para consultas con marca temporal.
- Consulta: camino único en el árbol + **interpolación por tramo** si falta muestra exacta.
- No se permite **extrapolar al futuro** para evitar hipótesis dinámicas implícitas.



Sistema de transformaciones (TF) en ROS 2

Publicación de TF propias

- Cualquier módulo puede publicar nuevos marcos (p.ej. **objetos detectados** o referencias externas).
- Añadir un marco integra la entidad en el árbol y habilita su uso por planificación/control.
- Convierte TF en una infraestructura **extensible** del modelo espacial del entorno.

Sensores específicos en ROS 2

Escáner láser 2D (LaserScan): geometría

- Un LIDAR 2D barre ángulos y produce distancias polares (r_i, θ_i) en el marco del sensor.
- En ROS 2 publica `sensor_msgs/LaserScan` con metadatos angulares, distancias y *timestamp*.
- El ángulo del índice i:

$$\theta_i = \texttt{angle\_min} + i \cdot \texttt{angle\_increment}$$

- Conversión a cartesiano:

$$x_i = r_i \cos(\theta_i)$$

$$y_i = r_i \sin(\theta_i)$$

Sensores específicos en ROS 2

Escáner láser 2D: dimensión temporal del barrido

- `header.stamp` es el instante del **primer rayo**; el barrido dura `scan_time`.
- `time_increment` da el intervalo entre rayos consecutivos.
- Cada medición tiene su propio instante:

$$t_i = t_0 + i \cdot \texttt{time\_increment}$$

- Si el robot se mueve, cada rayo corresponde a una pose distinta.
- Hay que consultar TF con resolución temporal suficiente para el instante t_i.
- Ignorarlo produce **distorsiones geométricas** (más a alta velocidad).

Sensores específicos en ROS 2

Escáner láser 2D: estructura de `sensor_msgs/LaserScan`

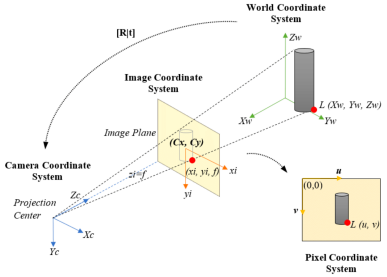
- `header.stamp` fija el tiempo de adquisición; `frame_id` fija el marco del escaneo (TF).
- La geometría del barrido se define con `angle_min/max` y `angle_increment`.
- La dimensión temporal se describe con `time_increment` y `scan_time`.
- Los límites físicos ayudan a filtrar medidas: `range_min/max`.
- Datos: `ranges` (m) y, si existe, `intensities`.

```
std_msgs/Header header
  builtin_interfaces/Time stamp
  string frame_id

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
```

Sensores específicos en ROS 2

Cámaras e imágenes: modelo pin-hole

- Modelo geométrico típico: **pin-hole** (proyección perspectiva 3D→2D).
- La calibración estima parámetros intrínsecos (focal, centro óptico, distorsión).
- En ROS 2, los intrínsecos se publican en `sensor_msgs/CameraInfo`.
- La imagen viaja en `sensor_msgs/Image` (píxeles + codificación).



Sensores específicos en ROS 2

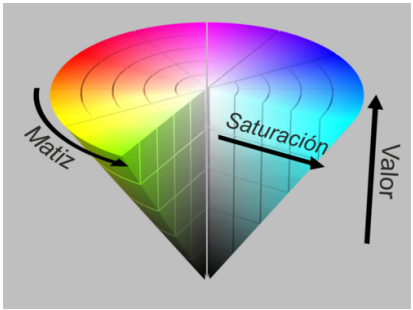
Cámaras e imágenes: ImageTransport

- Las imágenes `raw` son flujos de alta tasa: pueden saturar red/bus y aumentar latencias.
- **ImageTransport** (`image_transport`) desacopla **información** de **transporte**.
- Se publica un **topic base** (p.ej. `/camera/image`) y se ofrecen transportes alternativos mediante topics derivados: `/camera/image/compressed`, `/camera/image/theora`, etc.
- El consumidor elige transporte de forma **transparente** (por configuración) sin cambiar su lógica de percepción.
- Implicaciones: intercambio **CPU–ancho de banda**, latencia, despliegue distribuido y trazabilidad al grabar/depurar.
- Debe preservarse `header.stamp`/`frame_id` para mantener sincronización con `CameraInfo`, TF y otros sensores.

Sensores específicos en ROS 2

Cámaras e imágenes: espacios de color (HSV)

- En RGB/BGR, color y luminancia están acoplados: la iluminación cambia R/G/B a la vez.
- En **HSV**, H/S describen el "color" y V concentra el brillo.
- Umbralizar en H/S suele ser más robusto ante sombras y autoexposición.
- En práctica se convierte y procesa con **OpenCV**.



Sensores específicos en ROS 2

Cámaras e imágenes: estructura de `sensor_msgs/Image`

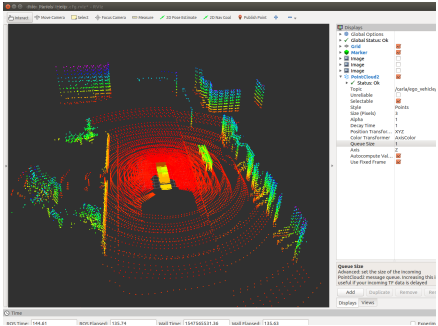
- `header.stamp` sincroniza con `CameraInfo` y TF; `frame_id` identifica el marco óptico.
- `height` y `width` definen la matriz de píxeles.
- `encoding` define canales/orden/profundidad (p.ej. `bgr8`, `mono8`, `16UC1`).
- `step` permite filas con padding; `data` contiene el *blob* de bytes.
- `is_bigendian` importa para codificaciones con palabras > 8 bits.

```
std_msgs/Header header
  builtin_interfaces/Time stamp
  string frame_id

uint32 height
uint32 width

string encoding

uint8  is_bigendian
uint32 step
uint8[] data
```

Sensores específicos en ROS 2

Sensores láser 3D y nubes de puntos

- Extienden LIDAR 2D a 3D (múltiples planos o rotación mecánica).
- Generan nubes de puntos: (x, y, z) + atributos (intensidad, color, etc.).
- En ROS 2, el tipo estándar es `sensor_msgs/PointCloud2`.
- Procesado con **PCL**: filtrado, segmentación, planos, registro.
- Son flujos muy pesados: frecuencia, filtrado previo y distribución computacional importan.



Sensores específicos en ROS 2

Sensores láser 3D: estructura de `sensor_msgs/PointCloud2`

- `header.stamp` y `frame_id`: sitúan la nube en tiempo y marco (TF/fusión).
- `height`/`width`: nube organizada (tipo imagen) o desordenada (típicamente `height=1`).
- `fields`: define el layout por punto (canales como `x,y,z,intensity`) con nombre/offset/tipo.
- `point_step`/`row_step` describen los *strides*; `data` es el blob binario.
- `is_dense` avisa de NaN/Inf; los algoritmos deben filtrar si es `false`.

```
std_msgs/Header header
uint32 height
uint32 width

PointField[] fields
bool    is_bigendian
uint32 point_step
uint32 row_step
uint8[] data
bool is_dense
```

Sensores específicos en ROS 2

ROSBag

- **rosbag** registra y reproduce mensajes de topics como si fueran sensores reales.
- Clave para depuración, experimentación reproducible y validación de algoritmos.
- Desacopla adquisición física de desarrollo: permite iterar sin hardware en tiempo real.
- Arquitectónicamente, actúa como **fuente alternativa** de datos.

De sensores a detecciones

Reducción de complejidad para decisión

- Tras coherencia espacial/temporal, un paso natural es **reducir la complejidad** del flujo sensorial.
- Objetivo: convertir datos densos (imagen, nube de puntos) en **entidades compactas** consumibles por control o deliberación.
- Ejemplos: “hay un obstáculo”, “hay una persona”, “hay una señal” con localización y confianza.
- Esto estabiliza contratos entre módulos: la percepción asume el coste de interpretar el sensor.

De sensores a detecciones

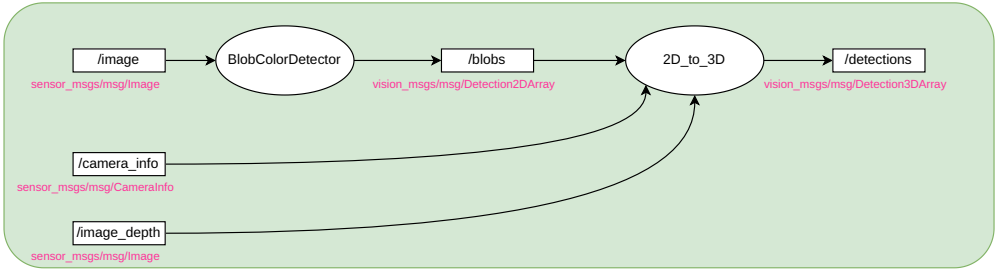
Mensajes `vision_msgs`: Detection2D/3D

- Contrato estándar para publicar **listas de detecciones** con clase, score y localización.
- 2D: cajas en imagen (`Detection2DArray`). 3D: pose + caja 3D (`Detection3DArray`).
- Facilita consumo por planificación/control y permite representar incertidumbre (`PoseWithCovariance`).
- Los mensajes no arrastran el dato denso: se relacionan con el origen por **marca temporal**.

```
vision_msgs/msg/Detection2DArray
  std_msgs/Header header
  Detection2D[] detections

vision_msgs/msg/Detection3DArray
  std_msgs/Header header
  Detection3D[] detections
```

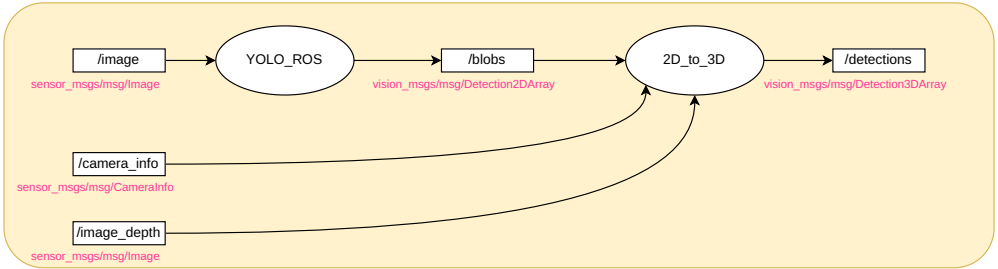
De sensores a detecciones

Ejemplos de subsistemas de percepción



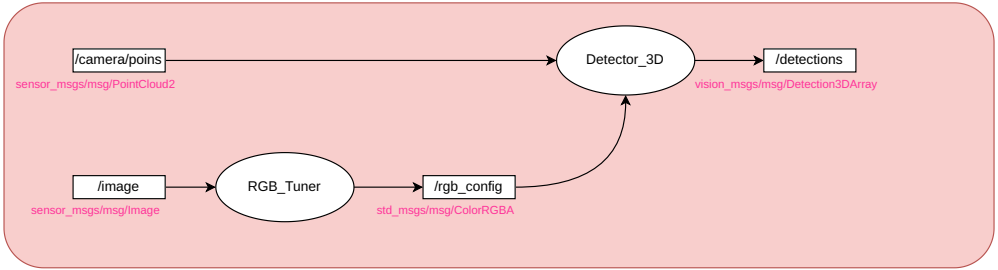
De sensores a detecciones

Ejemplos de subsistemas de percepción



De sensores a detecciones

Ejemplos de subsistemas de percepción



Preguntas