

Robots Móviles

(Temas Selectos de Control y Robótica)

(Temas Selectos de Mecatrónica I y II)

Dr. Marco Negrete

Facultad de Ingeniería, UNAM

Semestre 2025-1

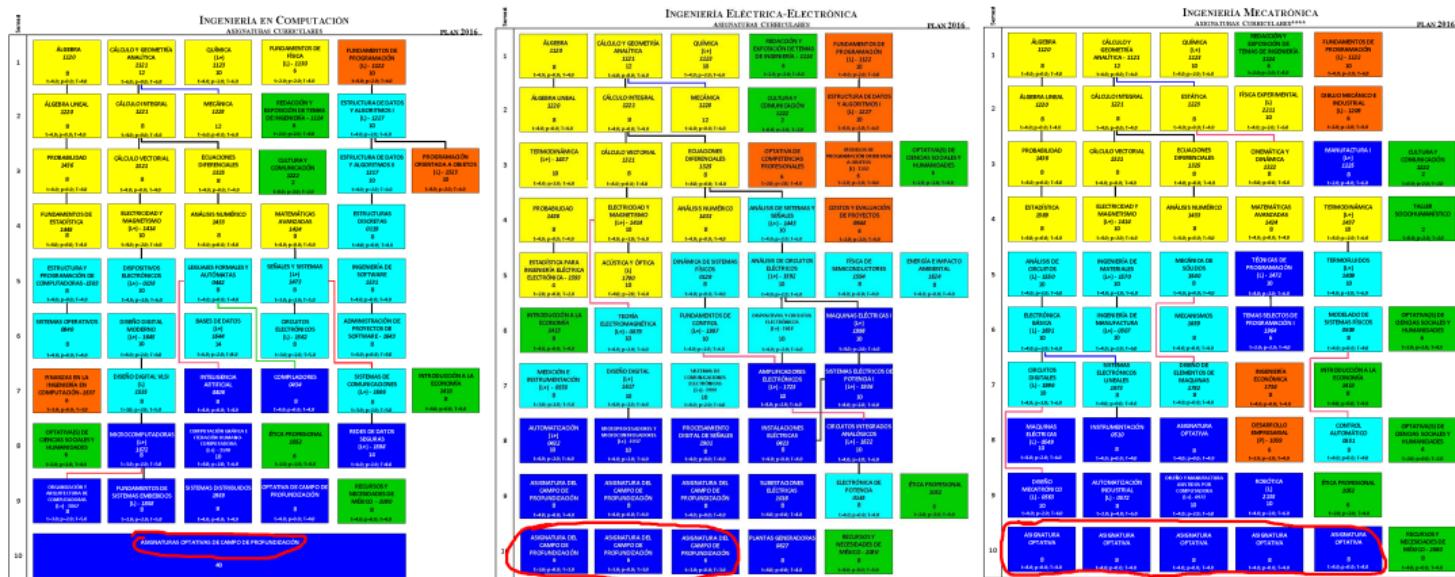
Material elaborado con apoyo del proyecto PAPIME PE105524

Objetivos:

- ▶ Aprender los conceptos básicos para operar un robot móvil autónomo
- ▶ Implementar dichos conceptos en un ambiente simulado
- ▶ Familiarizar al estudiante con la plataforma ROS

Ubicación en el plan de estudios

Bloque de ingeniería aplicada:



El curso integra diversos conceptos vistos a lo largo de la carrera: cálculo, álgebra, geometría, probabilidad, estructuras de datos, control, entre otras.

1. Introducción

- ▶ Componentes básicos de un robot móvil
- ▶ Herramientas de software para el desarrollo de robots móviles

2. Planeación de movimientos

- ▶ El problema de la planeación de movimientos
- ▶ Mapas geométricos y topológicos
- ▶ Descomposición en celdas, celdas de ocupación y diagramas de Voronoi
- ▶ Planeación de rutas mediante búsqueda en grafos
- ▶ Modelos cinemáticos: diferencial, omnidireccional, Ackermann
- ▶ Control de posición y seguimiento de trayectorias

3. Modelos reactivos

- ▶ Máquinas de estados finitas
- ▶ Campos potenciales artificiales
- ▶ Navegación mediante redes neuronales
- ▶ Navegación mediante algoritmos genéticos

4. Mapeo y localización

- ▶ Localización mediante filtro de Kalman extendido
- ▶ Localización mediante filtros de partículas
- ▶ Localización mediante Modelos Ocultos de Markov
- ▶ Creación de mapas mediante agrupamiento
- ▶ Localización y mapeo simultáneos

5. Conceptos básicos de visión artificial

- ▶ Imágenes y espacios de color
- ▶ Operadores morfológicos
- ▶ Extracción de características geométricas
- ▶ Visión 3D mediante imágenes RGB-D
- ▶ Redes neuronales artificiales

6. Conceptos básicos de manipulación

- ▶ Movimiento de cuerpo rígido
- ▶ Cinemáticas directa e inversa
- ▶ Planeación de trayectorias
- ▶ Seguimiento de trayectorias

7. Planeación de acciones

- ▶ Sistemas basados en reglas
- ▶ Procesos de Decisión de Markov

8. Herramientas para la interacción humano-robot

- ▶ Síntesis de voz con la biblioteca Festival
- ▶ Reconocimiento de palabras aisladas
- ▶ Reconocimiento de voz con la biblioteca CMU Sphinx

Bibliografía recomendada:

- ▶ <https://drive.google.com/drive/folders/1gb7VQJG5eUkCvCginRHHGn5lez6VASBJ?usp=sharing>
- ▶ <https://drive.google.com/drive/folders/1Epl2b51xEJzCvzfugBD1i7xGdKYdJucy?usp=sharing>

Forma de trabajo

- ▶ Horario: martes y jueves de 16:00 a 17:30.
- ▶ Para la realización de ciertas tareas y prácticas se utilizará el simulador Gazebo junto con la plataforma ROS Noetic. Este software corre en el sistema operativo Ubuntu 20.04.4. Varias tareas y prácticas implicarán escribir código para lo cual se utilizará el repositorio
<https://github.com/mnegretev/Mobile-Robots-2025-1>
Si no se desea instalar el sistema operativo de forma nativa, en el repositorio hay una máquina virtual con todo el software ya instalado.
- ▶ Para el uso del material del curso, se recomienda manejar las siguientes herramientas:
 - ▶ Sistema operativo Ubuntu
 - ▶ Lenguajes Python y C++
 - ▶ Software de control de versiones Git

Un conocimiento a nivel introductorio es suficiente. En caso de conocer las herramientas mencionadas, se recomienda buscar cursos gratuitos en www.udacity.com y www.edx.org

- ▶ El proyecto, las prácticas y algunas tareas incluyen la escritura de código. El alumno deberá completar el código utilizando las plantillas que ya se encuentran en el repositorio en línea en
<https://github.com/mnegretev/Mobile-Robots-2025-1>

Forma de trabajo

- ▶ Se creará una rama del repositorio para cada estudiante donde podrá subir códigos. Para ello el estudiante debe tener cuenta en GitHub (gratuita) y se le darán permisos de escritura en el repositorio.
- ▶ Las instrucciones para cada entregable (tareas, prácticas, proyecto y examen) así como las fechas de entrega se publicarán en el aula virtual en Classroom:

kyyetyi

- ▶ Algunas actividades serán individuales y otras en equipo. En el Classroom se indicará la modalidad para cada actividad.
- ▶ Las tareas que impliquen la escritura de código, se subirán al repositorio en la rama asignada para cada estudiante.
- ▶ Las tareas que no impliquen código se entregarán en papel al inicio de la clas en la fecha indicada.

Forma de evaluar

Rubros a evaluar:

Prácticas	40 %
Examen	30 %
Tareas	25 %
Proyecto	15 %

El examen final se exenta con 6.0. Si un alumno con calificación aprobatoria presenta el examen final, se entiende que renuncia a su calificación y el examen final contará el 100 %.

Todo comportamiento antiético causará una calificación de 0 en el entregable correspondiente.

Copiar y pegar texto de internet es un ejemplo de esto.

Forma de evaluar

- ▶ Las tareas se entregarán en papel al inicio de la clase en la fecha de entrega asignada. Si la tarea incluye la escritura de código, éste deberá subirse al repositorio en la rama correspondiente.
- ▶ Las prácticas consistirán en la escritura de código, que se subirá al repositorio en la rama correspondiente, y en la elaboración de un reporte escrito, que también se entregará impreso al inicio de la clase en la fecha asignada.
- ▶ El proyecto final consistirá en la integración de los conceptos vistos en el curso para resolver una tarea simple relacionada con robots de servicio doméstico, por lo que también consistirá en la escritura de código y en un reporte escrito.
- ▶ Para las tareas, prácticas y proyecto se publicará en el Classroom el instrumento de evaluación correspondiente. Para el caso de prácticas y proyectos, se utilizará una rúbrica y para el caso de tareas, una lista de cotejo.

Evaluación de reportes de prácticas y proyecto

- ▶ Los reportes de prácticas y proyecto deberán contener al menos los siguientes puntos:
 - ▶ Introducción (que incluya contexto, motivación, planteamiento del problema y objetivos)
 - ▶ Marco teórico (donde se expliquen los conceptos utilizados en la práctica o proyecto)
 - ▶ Desarrollo (donde se expliquen la implementación y las pruebas realizadas)
 - ▶ Resultados (donde se reporten las diferentes pruebas de funcionamiento)
 - ▶ Conclusiones (donde se discutan los resultados obtenidos y se plantee un trabajo futuro)
 - ▶ Referencias
- ▶ No hay extensión mínima ni máxima.
- ▶ No es necesario que los reportes sean extensos. Se prefiere que sean concisos.
- ▶ Las referencias deben ser sólo **publicaciones arbitradas** (libros, artículos de revista, memorias de congresos, entre otros). Se recomienda usar el buscador de la Dirección General de Bibliotecas de la UNAM (<https://www.dgb.unam.mx/>)
- ▶ No es necesario incluir código en el reporte, para eso está el repositorio en línea.
- ▶ No es necesario reproducir las instrucciones de ejecución de cada práctica en el reporte.
- ▶ Las conclusiones deben estar basadas en los resultados presentados en el reporte. Las opiniones personales **NO** son conclusiones.

Condición para continuar el curso

Se deberá enviar un correo electrónico a la dirección marco.negrete@ingenieria.unam.edu donde se indiquen los siguientes puntos:

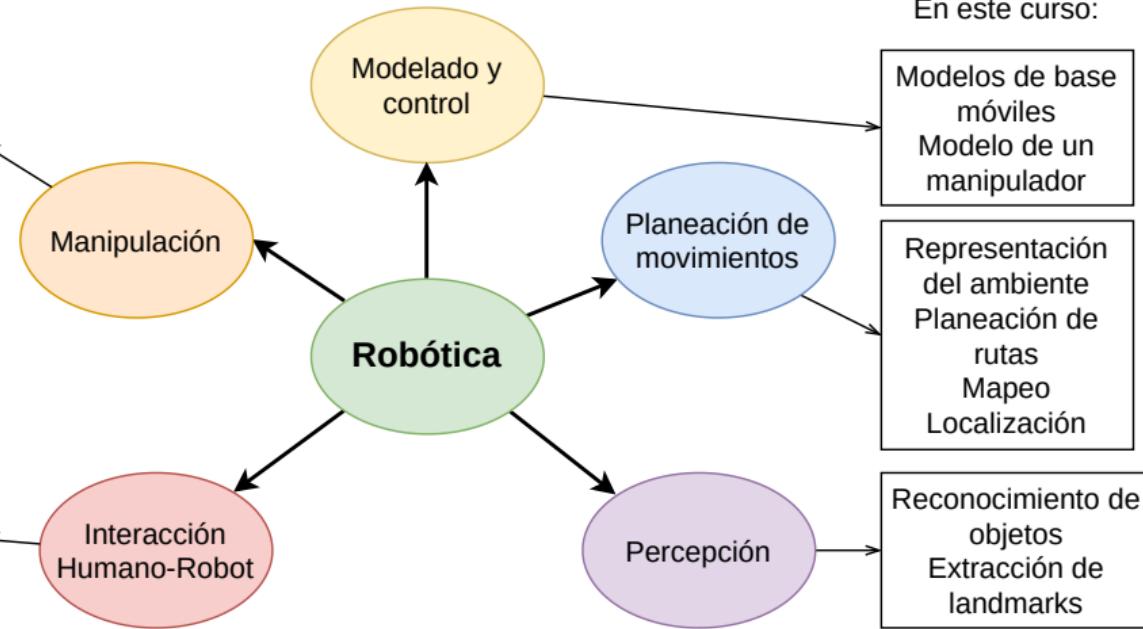
- ▶ Manifestar que se conocen los siguientes puntos:
 - ▶ Los objetivos del curso
 - ▶ Los criterios de evaluación
 - ▶ La forma de trabajo y la bibliografía recomendada
 - ▶ Sanciones en caso de conductas antiéticas
 - ▶ Video sobre la UIG
- ▶ Manifestar la aceptación de todos los puntos anteriores

- ▶ La palabra *robot* tiene su origen en la obra *Rossum's Universal Robots* del escritor checo Karel Čapek, publicada en 1921, y su significado es “trabajo duro”.
- ▶ Latombe (1991) define un robot como un dispositivo mecánico versátil equipado con sensores y actuadores bajo el control de un sistema de cómputo [3].
- ▶ Arkin (1998) propone que un robot inteligente es una máquina capaz de extraer información de su ambiente y usar el conocimiento acerca de su mundo para moverse de manera segura y significativa, con un propósito específico [1].
- ▶ Robótica es la ciencia que estudia la conexión inteligente entre la percepción y la acción.

Áreas de la Robótica

En este curso:

Cinemáticas
directa e inversa
Planeación de
trayectorias



En este curso:

Modelos de base
móviles
Modelo de un
manipulador

Representación
del ambiente
Planeación de
rutas
Mapeo
Localización

Reconocimiento de
objetos
Extracción de
landmarks

Robot

Sensores:

Son transductores que obtienen información del ambiente útil para la toma de decisiones.

Propioceptivos: sensan el estado interno del robot.

Exteroceptivos: sensan el ambiente externo.

Activos: emiten energía para realizar la medición.

Pasivos: no emiten energía.

Ejemplos de Sensores:

- Cámaras (RGB, RGB-D)
- Micrófonos
- Lidar
- Encoders
- Sensores de batería

Procesadores:

Es el hardware que se utiliza para procesar información. Reciben información de los sensores y envían comandos a los actuadores.

Ejemplos de procesadores:

- CPUs
- GPUs
- FPGA
- Microcontrolador
- DSP

Actuadores:

Son dispositivos que realizan alguna modificación al ambiente. Se pueden clasificar según su principio de actuación:

- Eléctricos
- Hidráulicos
- Neumáticos

Ejemplos de actuadores

- Motores (DC, Brushless)
- Bocinas
- Pistones
- Manipuladores

- ▶ **Configuración:** es la descripción de la posición en el espacio de todos los puntos del robot. Se denota con q .
- ▶ **Espacio de configuraciones:** es el conjunto Q de todas las posibles configuraciones.
- ▶ **Grados de libertad:** número mínimo de variables independientes para describir una configuración. En este curso, la base móvil del robot tiene 3 GdL, la cabeza tiene 2 GDL y cada brazo tiene 7 GDL más 1 GdL para el gripper. En total, el robot tiene 21 GdL.

Propiedades del robot:

- ▶ **Holonómico:** el robot puede moverse instantáneamente en cualquier dirección del espacio de configuraciones. Comunmente se logra mediante ruedas de tipo *Mecanum* u *Omnidireccionales*.
- ▶ **No holonómico:** existen restricciones de movimiento en velocidad pero no en posición. Son restricciones que solo se pueden expresar en términos de la velocidad pero no pueden integrarse para obtener una restricción en términos de posición. Ejemplo: un coche sólo puede moverse en la dirección que apuntan las llantas delanteras, sin embargo, a través de maniobras puede alcanzar cualquier posición y orientación. El robot de este curso es no holonómico.

Propiedades de los algoritmos:

- ▶ **Complejidad:** cuánta memoria y cuánto tiempo se requiere para ejecutar un algoritmo, en función del número de datos de entrada (número de grados libertad, número de lecturas de un sensor, entre otros).
- ▶ **Optimalidad:** un algoritmo es óptimo cuando encuentra una solución que minimiza una función de costo.
- ▶ **Completitud:** un algoritmo es completo cuando garantiza encontrar la solución siempre que ésta exista. Si la solución no existe, indica falla en tiempo finito.
 - ▶ Completitud de resolución: la solución existe cuando se tiene una discretización.
 - ▶ Completitud probabilística: la probabilidad de encontrar la solución tiende a 1.0 cuando el tiempo tiende a infinito.

Una explicación más detallada se puede encontrar en el Cap. 3 de [2].

Primitivas de la robótica

Las tareas que puede llevar a cabo un robot se pueden clasificar en tres grandes conjuntos conocidos como primitivas de la robótica: **sensar**, **planear** y **actuar**.

- ▶ **Sensar:** extracción de información del ambiente interno o externo del robot.
- ▶ **Planear:** generación de subtareas y toma decisiones a partir de la información de los sensores y/o de alguna Representación interna del ambiente.
- ▶ **Actuar:** modificación del ambiente con alguno de los dispositivos del robot.

Paradigma jerárquico. Las tres primitivas se realizan en forma secuencial.



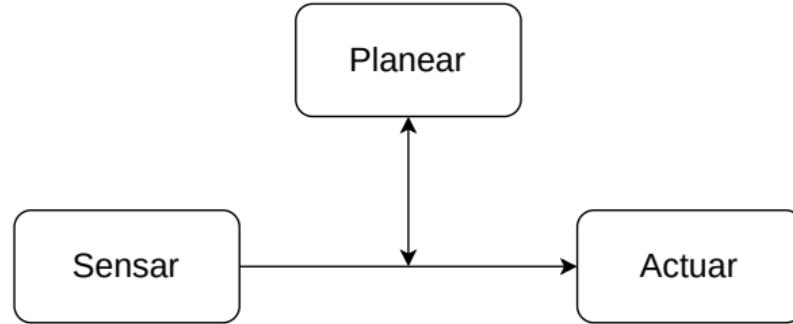
- ▶ Fuerte dependencia de una representación interna del ambiente
- ▶ Tiempo de respuesta lento comparado con el paradigma reactivo
- ▶ Alto costo computacional
- ▶ Se pueden resolver tareas con alto nivel cognitivo
- ▶ Alta capacidad de predicción

Paradigma reactivo. El sensado y la actuación se conectan directamente sin que haya de por medio una planeación.



- ▶ No requiere una representación interna del ambiente
- ▶ Tiempo de respuesta rápido comparado con el paradigma jerárquico
- ▶ Bajo costo computacional
- ▶ En general, no se pueden resolver tareas con alto nivel cognitivo
- ▶ Baja capacidad de predicción

Paradigma híbrido. Tiene como objetivo utilizar las ventajas de ambos paradigmas, es decir, emplear comportamientos reactivos para que el robot responda rápidamente ante cambios en el ambiente sin perder la alta capacidad cognitiva y de predicción que brinda el paradigma jerárquico



Tarea 01 - Herramientas de software

Ver instrucciones en Classroom

Deadline: 2024-08-13 al inicio de la clase.



ROS (Robot Operating System) es un *middleware* de código abierto para el desarrollo de robots móviles.

- ▶ Implementa funcionalidades comúnmente usadas en el desarrollo de robots como el paso de mensajes entre procesos y la administración de paquetes.
- ▶ Muchos drivers y algoritmos ya están implementados.
- ▶ Es una plataforma distribuida de procesos (llamados *nodos*).
- ▶ Facilita el reuso de código.
- ▶ Independiente del lenguaje (Python y C++ son los más usados).
- ▶ Facilita el escalamiento para proyectos de gran escala.

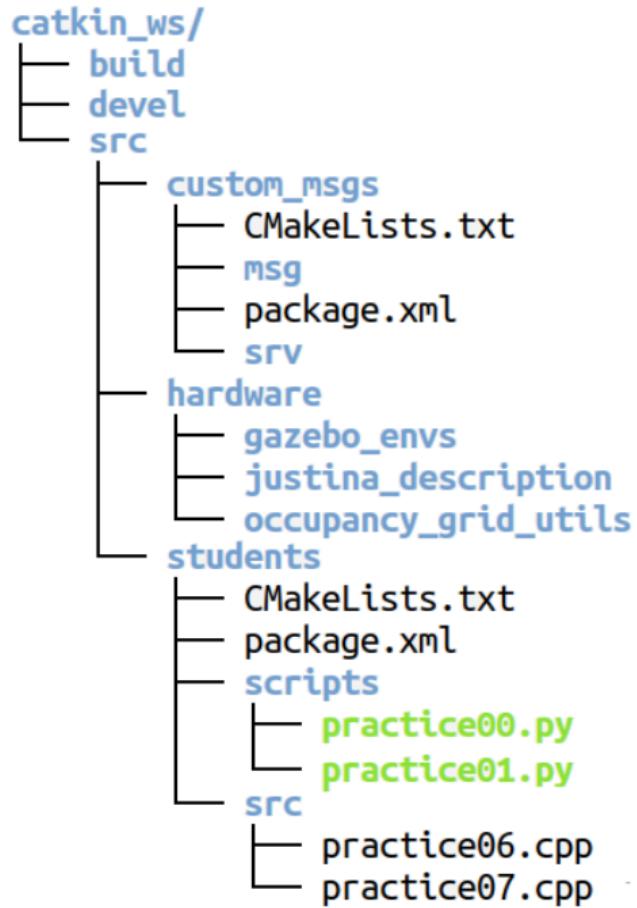
ROS se puede entender en dos grandes niveles conceptuales:

- ▶ **Sistema de archivos:** Recursos de ROS en disco
- ▶ **Grafo de procesos:** Una red *peer-to-peer* de procesos (llamados nodos) en tiempo de ejecución.

Sistema de archivos

Recursos en disco:

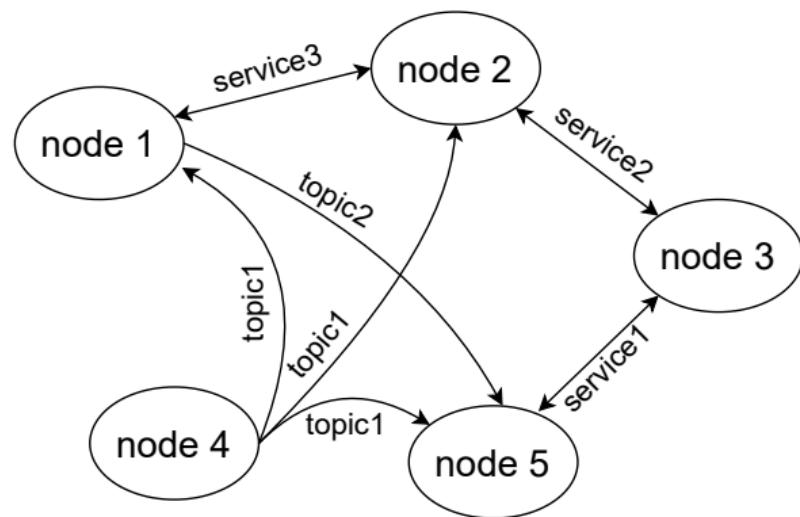
- ▶ **Workspace:** carpeta que contiene los paquetes desarrollados
- ▶ **Paquetes:** Principal unidad de organización del software en ROS (concepto heredado de Linux)
- ▶ **Manifiesto:** (`package.xml`) provee metadatos sobre el paquete (dependencias, banderas de compilación, información del desarrollador)
- ▶ **Mensajes (msg):** Archivos que definen la estructura de un *mensaje* en ROS.
- ▶ **Servicios (srv):** Archivos que definen las estructuras de la petición (*request*) y respuesta (*response*) de un servicio.



Grafo de procesos

El grafo de procesos es una red *peer-to-peer* de programas (nodos) que intercambian información entre sí. Los principales componentes del este grafo son:

- ▶ master
- ▶ servidor de parámetros
- ▶ nodos
- ▶ mensajes
- ▶ servicios



Tópicos y servicios

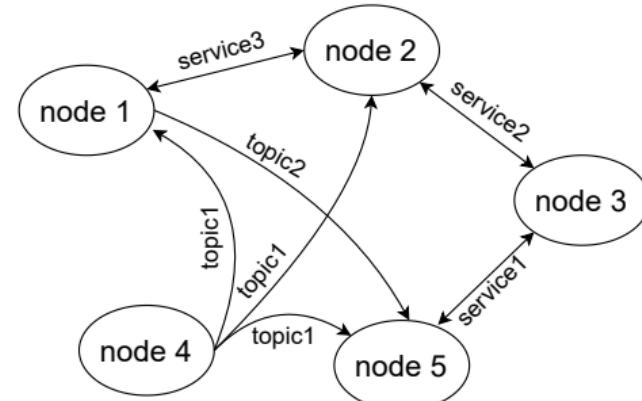
Los nodos (procesos) en ROS intercambian información a través de dos grandes patrones:

► Tópicos

- ▶ Son un patrón $1:n$ de tipo *publicador/suscriptor*
- ▶ Son no bloqueantes
- ▶ Utilizan estructuras de datos definidas en archivos `*.msg` para el envío de información

► Servicios

- ▶ Son un patrón $1:1$ de tipo *petición/respuesta*
- ▶ Son bloqueantes
- ▶ Utilizan estructuras de datos definidas en archivos `*.srv` para el intercambio de información.

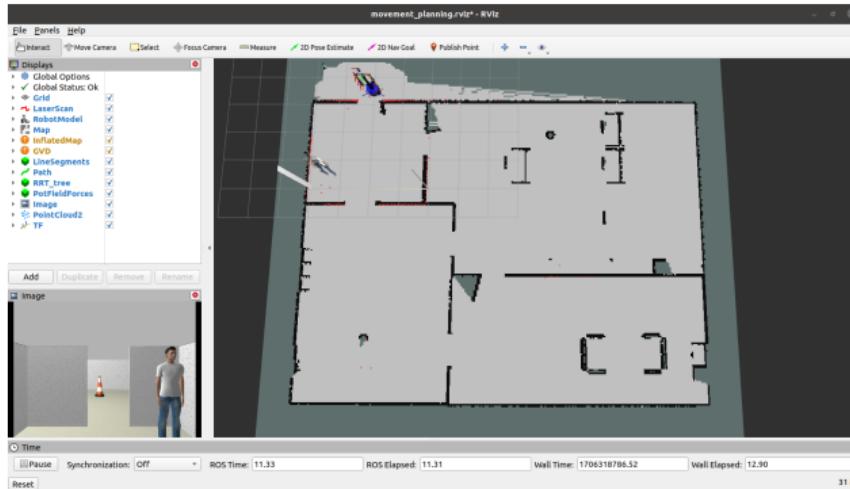


Para mayor información:

- ▶ Tutoriales <http://wiki.ros.org/ROS/Tutorials>
- ▶ Koubâa, A. (Ed.). (2020). Robot Operating System (ROS): The Complete Reference. Springer Nature

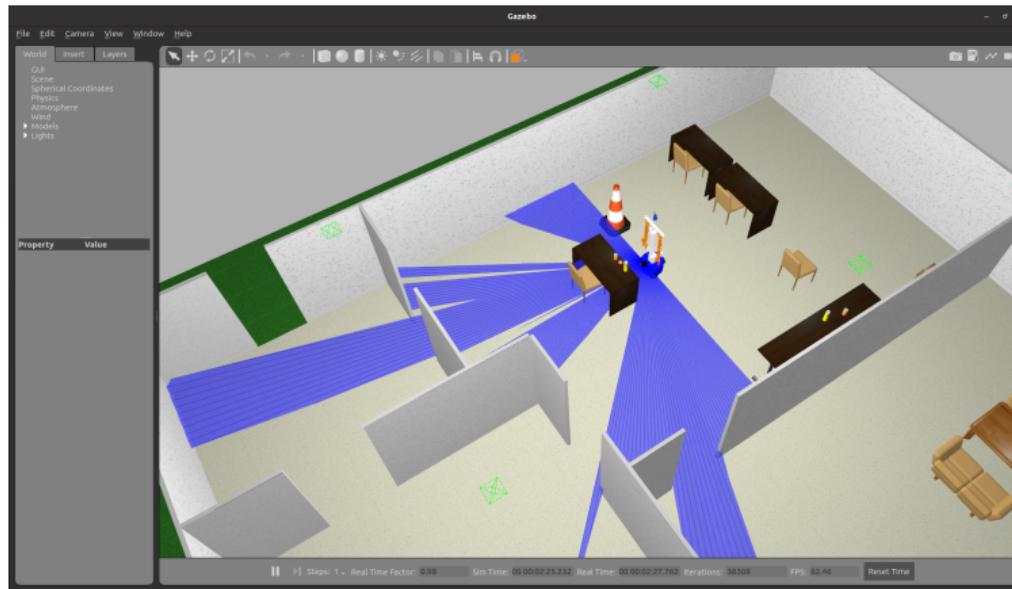
El visualizador RViz

- ▶ Es un nodo de ROS que tiene *plugins* para dibujar en pantalla diversos tipos de mensajes: lidas, mapas, robots, nubes de puntos, rutas, entre muchos otros.
- ▶ Funciona suscribiéndose a los tópicos indicados por el usuario.
- ▶ Utiliza el paquete *tf* para dibujar todo con respecto a un sistema de referencia.



El simulador Gazebo

- ▶ Acceso a múltiples motores de físicas de alto rendimiento como: ODE, Bullet, Simbody y DART
- ▶ Existen *plugins* para los sensores y actuadores más comunes en robots móviles (lidar, cámara RGB-D, motores, base diferencial, entre otros)
- ▶ Los componentes básicos son los *modelos* (archivos *.sdf) y los *mundos* (archivos *.world)
- ▶ Fácil integración con ROS



Tarea 02 - La plataforma ROS

1. Investigar dos aplicaciones de ROS y el simulador Gazebo. Se pueden consultar las siguientes páginas:
 - ▶ <https://www.openrobotics.org/markets>
 - ▶ <https://vimeo.com/649649866/37198994b5>
 - ▶ <https://robots.ros.org/robonaut2/>
 - ▶ <https://github.com/nasa/astrobe>
2. Buscar para qué sirven los comandos: `rosrun`, `rostopic list` y `rostopic echo`
3. Abra el archivo `catkin_ws/src/navigation/simple_move/scripts/ros_basics.py` y agregue el siguiente código en la línea 25:

```
25 n = int((msg.angle_max - msg.angle_min)/msg.angle_increment/2)
26 obstacle_detected = msg.ranges[n] < 1.0
27
```

En el mismo archivo, en la línea 45, agregue el siguiente código:

```
45 msg_cmd_vel = Twist()
46 msg_cmd_vel.linear.x = 0 if obstacle_detected else 0.3
47 pub_cmd_vel.publish(msg_cmd_vel)
48
```

Tarea 02 - La plataforma ROS

Ver instrucciones en Classroom

Deadline: 2024-08-15 al inicio de la clase.

El problema de la planeación de movimientos comprende cuatro tareas principales:

- ▶ Navegación: encontrar un conjunto de puntos $q \in Q_{free}$ que permitan al robot moverse desde una configuración inicial q_{start} a una configuración final q_{goal} .
- ▶ Mapeo: construir una representación del ambiente a partir de las lecturas de los sensores y la trayectoria del robot.
- ▶ Localización: determinar la configuración q dado un mapa y lecturas de los sensores.
- ▶ Barrido: pasar un actuador por todos los puntos $q \in Q_b \subset Q$.

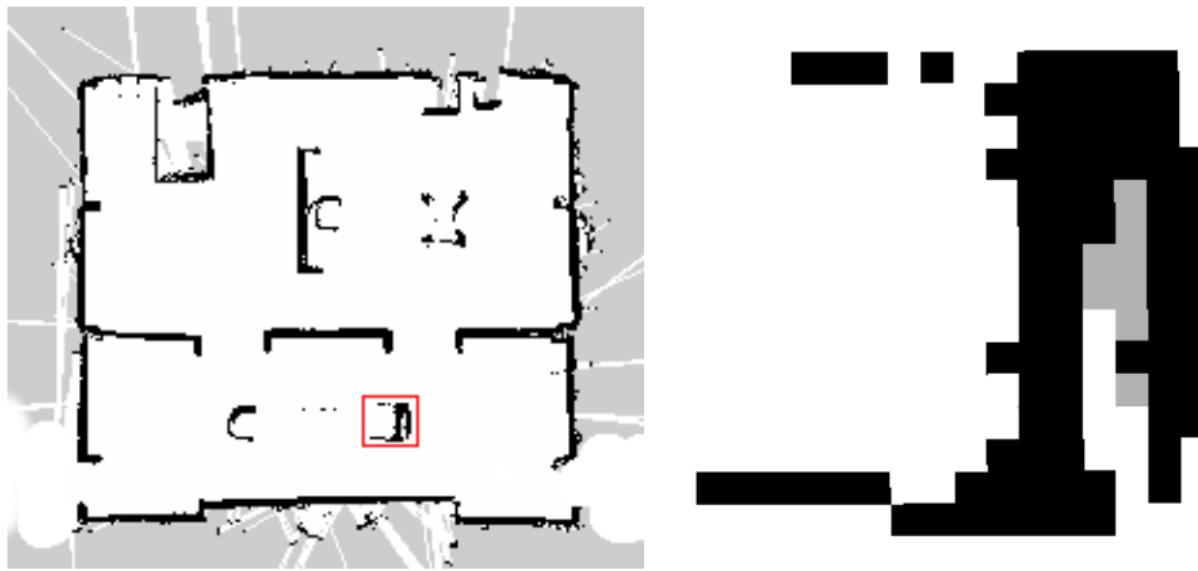
Representación del ambiente

Un mapa es cualquier representación del ambiente útil en la toma de decisiones.

- ▶ Interiores (se suelen representar en 2D)
 - ▶ Celdas de ocupación
 - ▶ Mapas de líneas
 - ▶ Mapas topológicos: Diagramas de Voronoi generalizados.
 - ▶ Mapas basados en *Landmarks*
- ▶ Exteriores (suelen requerir una representación 3D)
 - ▶ Celdas de elevación
 - ▶ Celdas de ocupación 3D
 - ▶ Octomaps

Celdas de ocupación

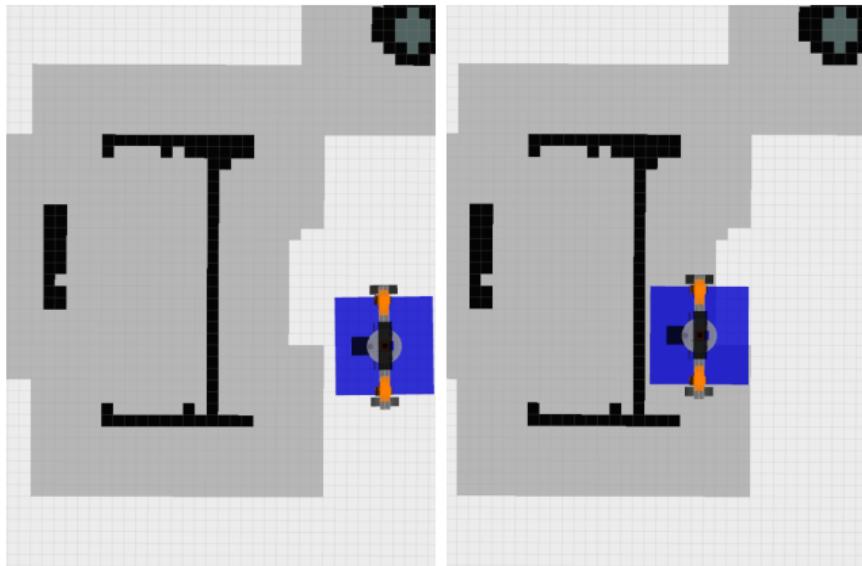
Es un tipo de mapa geométrico. El espacio se discretiza con una resolución determinada y a cada celda se le asigna un número $p \in [0, 1]$ que indica su nivel de ocupación. En un enfoque probabilístico este número se puede interpretar como la certeza que se tiene de que una celda esté ocupada.



El mapa resultante se representa en memoria mediante una matriz de valores de ocupación. En ROS, los mapas utilizan el mensaje `nav_msgs/OccupancyGrid`.

Inflado de celdas de ocupación

Aunque las celdas de ocupación representan el espacio donde hay obstáculos y donde no, en realidad, el robot no puede posicionarse en todas las celdas libres, debido a su tamaño, como se observa en la figura:



- ▶ Celdas blancas: espacio libre.
- ▶ Celdas negras: espacio con obstáculos.
- ▶ Celdas grises: espacio sin obstáculos donde el robot no puede estar debido a su tamaño.

- ▶ Un mapa de celdas de ocupación debe *inflarse* antes de usarse para planear rutas.
- ▶ Esta operación se conoce como *dilatación* y es un operador morfológico como se verá en la sección de conceptos de visión.
- ▶ El inflado se usa para planeación de rutas, no para localización.

Inflado de celdas de ocupación

Algoritmo 1: Algoritmo de inflado de mapas

Data:

Mapa M de celdas de ocupación

Radio de inflado r_i

Result: Mapa inflado M_{inf}

M_{inf} = Copia de M

foreach $i \in [0, \dots, \text{rows}]$ **do**

foreach $j \in [0, \dots, \text{cols}]$ **do**

 //Si la celda está ocupada, marcar como ocupadas las r_i celdas de alrededor.

if $M[i, j] == 100$ **then**

foreach $k_1 \in [-r_i, \dots, r_i]$ **do**

foreach $k_2 \in [-r_i, \dots, r_i]$ **do**

$M_{inf}[i + k_1, j + k_2] = 100$

end

end

end

end

end

Mapas de líneas

También son mapas geométricos, pero al almacenar *features* requieren mucho menos memoria. La desventaja es la dificultad para extraer líneas del ambiente y la poca precisión en el empateado.



Algunos métodos para extraer líneas:

- ▶ *Split and merge*
- ▶ Transformada Hough (se verá en la sección de visión computacional)
- ▶ RANSAC

Algoritmo *Split and Merge*

Se utiliza principalmente cuando los datos provienen de un sensor Lidar y por lo tanto, los puntos están en secuencia.

Algoritmo 2: *Split*

Data: Conjunto de puntos P

Result: Conjunto de líneas en forma normal (ρ, θ)

Ajustar una recta L al conjunto P por mínimos cuadrados

Encontrar el punto p_i más lejano a la recta

if $d(p_i, L) > d_0$ **then**

 Dividir P en dos subconjuntos P_1 y P_2 usando p_i como pivote

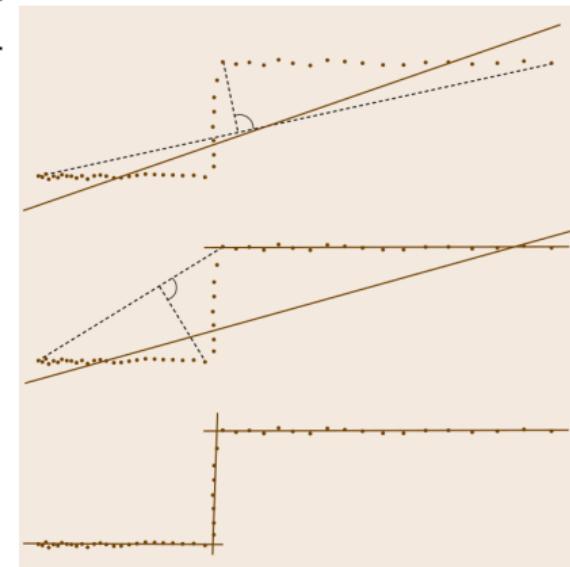
 Aplicar este algoritmo recursivamente para P_1 y P_2

 Devolver las rectas de ambos subconjuntos

else

 Devolver la recta L en forma normal (ρ, θ)

end



Algoritmo *Split and Merge*

El algoritmo anterior realiza la partición (*split*) del conjunto de puntos en posibles rectas y obtiene la ecuación para cada conjunto por mínimos cuadrados. La etapa de mezcla (*merge*) consiste simplemente en considerar como una sola recta, dos segmentos con parámetros (ρ, θ) similares. Los parámetros a sintonizar en este algoritmo son:

- ▶ d_0 : umbral de distancia entre un punto p_i y la recta L
- ▶ N_{min} : número mínimo de puntos para considerar que un subconjunto puede ser una recta.
- ▶ ρ_{tol} : valor máximo de diferencia entre dos rectas con ρ_1 y ρ_2 para considerarlas como una sola recta.
- ▶ θ_{tol} : valor máximo de diferencia entre dos rectas con θ_1 y θ_2 para considerarlas como una sola recta.

Mínimos cuadrados

Este método busca minimizar las distancias entre los puntos (x_i, y_i) y la recta en forma normal dada por los parámetros (ρ, θ) .

Dado un conjunto de puntos (x_i, y_i) , la recta (ρ, θ) que mejor se ajusta se puede obtener con:

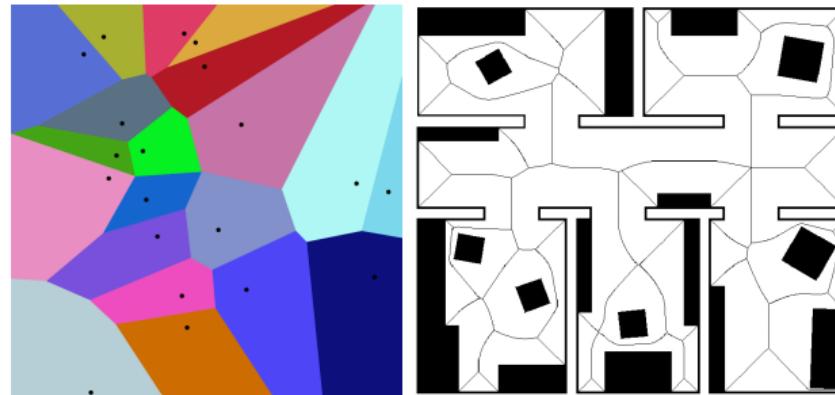
$$\begin{aligned}\theta &= \frac{1}{2} \operatorname{atan2} \left(-2 \sum_i (\bar{x} - x_i)(\bar{y} - y_i) , \sum_i [(\bar{y} - y_i)^2 - (\bar{x} - x_i)^2] \right) \\ \rho &= \bar{x} \cos \theta + \bar{y} \sin \theta\end{aligned}$$

con

$$\begin{aligned}\bar{x} &= \frac{1}{n} \sum_i x_i \\ \bar{y} &= \frac{1}{n} \sum_i y_i\end{aligned}$$

Diagrama de Voronoi Generalizado

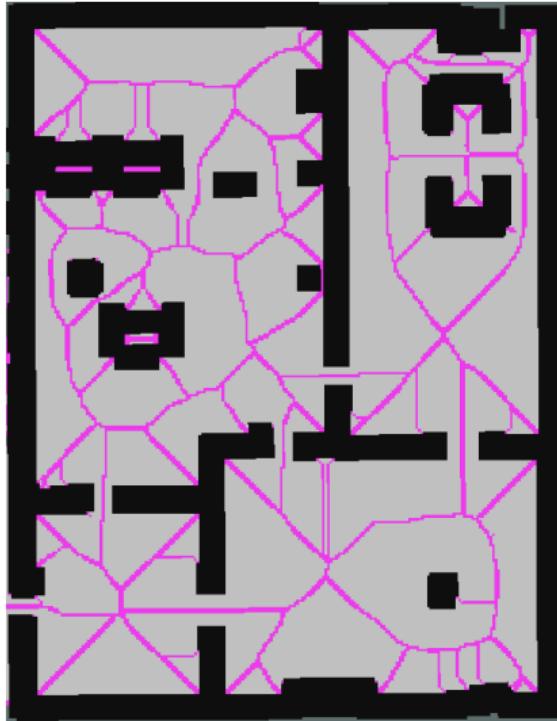
- ▶ A diferencia de los mapas geométricos, donde se busca reflejar la forma exacta del ambiente, los **mapas topológicos** buscan representar solo las relaciones espaciales de los puntos de interés.
- ▶ Los Diagramas de Voronoi dividen el espacio en regiones. Cada región está asociada a un punto llamado semilla, sitio o generador. Una región asociada a una semilla x contiene todos los puntos p tales que $d(x, p)$ es menor o igual que la distancia $d(x', p)$ a cualquier otra semilla x' .
- ▶ Un diagrama de Voronoi generalizado (GVD) considera que las semillas pueden ser objetos con dimensiones y no solo puntos.



- ▶ La forma de las regiones depende de la función de distancia que se utilice.

El algoritmo *Brushfire*

- ▶ Obtener un GVD es aún un problema abierto
- ▶ Se simplifica el problema si se asume que el espacio está representado por Celdas de Ocupación
- ▶ En este caso el GVD se puede obtener mediante el algoritmo *Brushfire*
- ▶ El mapa de rutas mostrado en la figura se forma con las celdas que son máximos locales en el mapa de distancias devuelto por Brushfire, es decir, son las celdas que son fronteras entre las regiones de Voronoi.
- ▶ Estas celdas también son aquellas equidistantes a los dos obstáculos más cercanos.



El algoritmo *Brushfire*

Algoritmo 3: Brushfire

Data: Mapa de celdas de ocupación M

Result: Distancias de cada celda al objeto más cercano

Fijar $d(p) = 0$ para toda celda p en los obstáculos

Fijar $d(p) = -1$ para toda celda p en el espacio libre

Crear una cola Q y agregar toda p en los obstáculos

while Q no esté vacía **do**

x = desencolar de Q

forall celdas p vecinas de x **do**

if $d(p) == -1$ **then**

 Aregar p a Q

 Fijar $d(p) = x + d(p, x)$

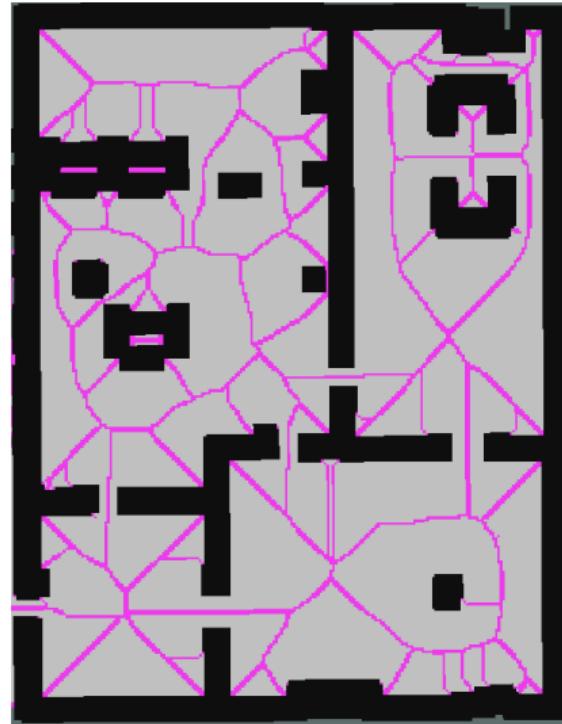
else

 Fijar $d(p) = \min(d(p), x + d(p, x))$

end

end

end



Tarea 03 - Representación del ambiente

Ver instrucciones en Classroom

Deadline: 2024-08-20 al inicio de la clase.

Planeación de rutas

La planeación de rutas consiste en encontrar una secuencia de puntos $q \in Q_{free}$ que permitan al robot moverse desde una configuración inicial q_{start} hasta una configuración final q_{goal} .

- ▶ Una **ruta** es solo la secuencia de configuraciones para llegar a la meta.
- ▶ Cuando la secuencia de configuraciones se expresa en función del tiempo, entonces se tiene una **trayectoria**.

En este curso solo vamos a hacer planeación de rutas, no de trayectorias (para navegación).

Existen varios métodos para planear rutas. La mayoría de ellos se pueden agrupar en:

- ▶ Métodos basados en muestreo
- ▶ Métodos basados en grafos

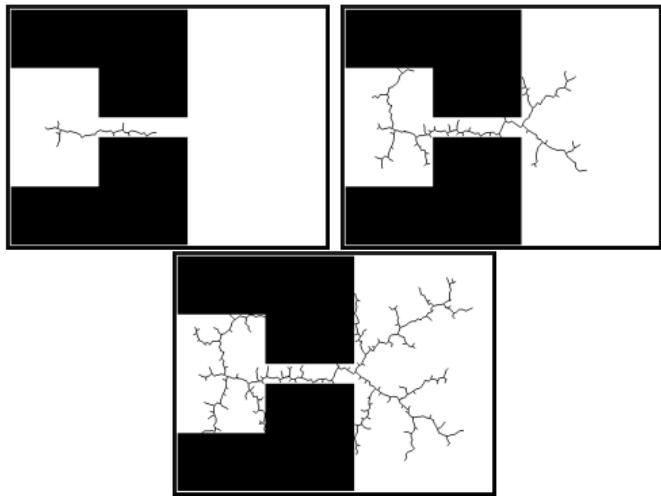
Como su nombre lo indica, consisten en tomar muestras aleatorias del espacio libre. Si es posible llegar en línea recta de la configuración actual al punto muestrado, entonces se agrega a la ruta. Ejemplos:

- ▶ RRT (Rapidly-exploring Random Trees)
- ▶ RRT-Bidireccional
- ▶ RRT-Extendido

Rapidly-exploring Random Trees

Consiste en construir un árbol a partir de muestras aleatorias del espacio libre. Los pasos generales se pueden resumir en:

1. Construir un árbol T con nodo raíz en el punto inicial
2. Elegir una posición aleatoria $p = (x, y)$ dentro del espacio libre
3. Encontrar el nodo n , del árbol T , más cercano al punto p
4. Si la distancia $d(n, p) > \epsilon$, cambiar p por un punto en la misma dirección \vec{np} pero a una distancia ϵ
5. Si no hay colisión entre n y p , agregar a p como nodo hijo de n
6. Si no hay colisión entre p y el punto meta p_g , agregar p_g como nodo hijo de p , terminar la exploración y anunciar éxito
7. Repetir desde el punto 2 un máximo de N veces.



Rapidly-exploring Random Trees

Algoritmo 4: RRT

Data: Mapa, p_s = Punto origen, p_g = Punto meta

Result: Ruta de p_s a p_g

Árbol.Raíz = p_s

while $\text{padre}(p_g) == \text{Null} \wedge \text{intentos} < N$ do

$p \leftarrow$ Punto aleatorio en el espacio libre

$n \leftarrow$ Nodo más cercano a p

$p_n \leftarrow \text{ObtenerNuevoNodo}(n, p, \epsilon)$

 if No hay colisión de n a p_n then

 Aregar p_n como nodo hijo de n

 if No hay colisión de p a p_g then

 Aregar p_g como nodo hijo de p_n

 Detener exploración

 end

 end

 intentos ++

end

Obtener ruta P a partir del árbol de exploración

return P

ObtenerNuevoNodo:

Data: Nodo n , punto p y distancia ϵ

Result: Nuevo punto p_n

if $d(n, p) > \epsilon$ then

$$p_n \leftarrow n + \epsilon \frac{p - n}{|p - n|}$$

else

$$p_n \leftarrow p$$

end

return p_n

Obtener Ruta del Árbol de Exploración:

Data: Árbol de exploración T , nodo meta p_g

Result: Ruta P

while $\text{padre}(p_g) \neq \text{NULL}$ do

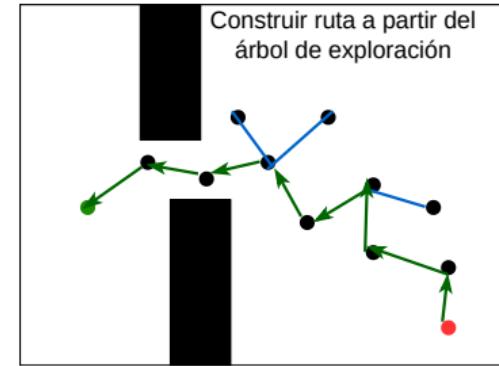
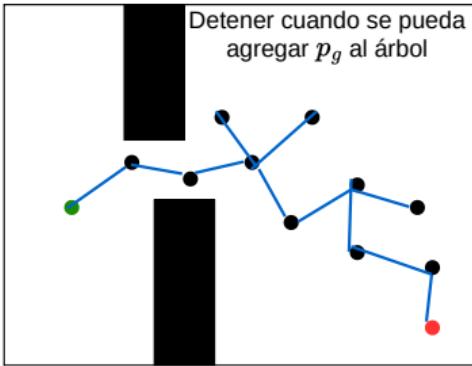
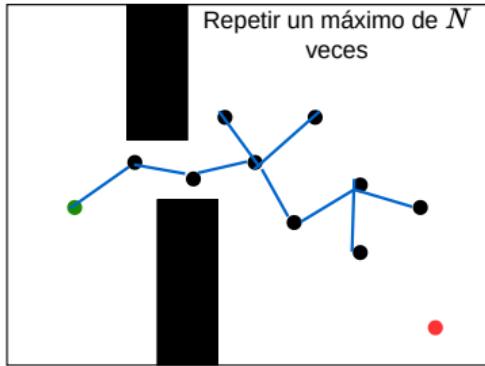
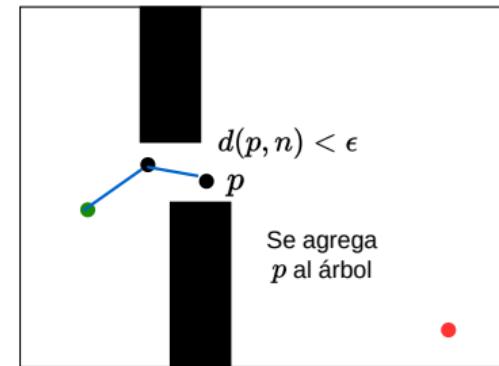
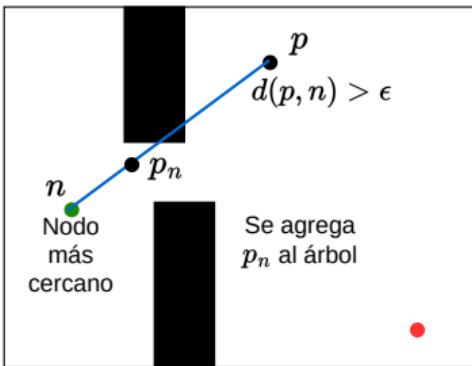
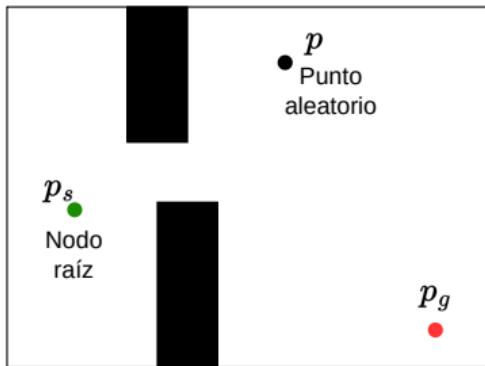
 Insertar p_g al inicio de la ruta P

$$p_g \leftarrow \text{parent}(p_g)$$

end

return P

Rapidly-exploring Random Trees



Tarea 04 - Planeación de rutas mediante RRT

Ver instrucciones en Classroom

Deadline: 2024-08-22 al inicio de la clase.

Métodos basados en grafos

Estos métodos consideran el ambiente como un grafo. En el caso de celdas de ocupación, cada celda libre es un nodo que está conectado con las celdas vecinas que también estén libres. Los pasos generales de este tipo de algoritmos se pueden resumir en:

Data: Mapa M de celdas de ocupación, configuración inicial q_{start} , configuración meta q_{goal}

Result: Ruta $P = [q_{start}, q_1, q_2, \dots, q_{goal}]$

Obtener los nodos n_s y n_g correspondientes a q_{start} y q_{goal}

Lista abierta $OL = \emptyset$ y lista cerrada $CL = \emptyset$

Agregar n_s a OL

Nodo actual $n_c = n_s$

while $OL \neq \emptyset$ y $n_c \neq n_g$ **do**

 | Seleccionar n_c de OL bajo algún criterio

 | Agregar n_c a CL

 | Expandir n_c

 | Agregar a OL los vecinos de n_c que no estén ya en OL ni en CL

end

if $n_c \neq n_g$ **then**

 | Anunciar Falla

end

Obtener la configuración q_i para cada nodo n_i de la ruta

Métodos basados en grafos

El criterio para seleccionar el siguiente nodo a expandir n_c de la lista abierta, determina el tipo de algoritmo:

- ▶ Criterio FIFO: Búsqueda a lo ancho BFS (la lista abierta es una cola)
- ▶ Criterio LIFO: Búsqueda en profundidad DFS (la lista abierta es una pila)
- ▶ Menor valor g : Dijkstra (la lista abierta es una cola con prioridad)
- ▶ Menor valor f : A* (la lista abierta es una cola con prioridad)

Si el costo g para ir de una celda a otra es siempre 1, entonces Dijkstra es equivalente a BFS. A* y Dijkstra siempre calculan la misma ruta (óptima) pero A* lo hace más rápido.

Mapas de costo

- ▶ Los métodos como Dijkstra y A* minimizan una función de costo. Esta función podría ser distancia, tiempo de recorrido, número de giros, energía gastada, entre otras.
- ▶ En este curso se empleará como costo una combinación de distancia recorrida más peligro de colisión (cercanía a los obstáculos).
- ▶ De este modo, las rutas serán un equilibrio entre rutas cortas y rutas seguras.
- ▶ Ejemplo: la ruta azul es más larga pero más segura.



Mapas de costo

- Se utilizará como costo una función de cercanía.
- Se calcula de forma similar al algoritmo Brushfire, pero la función decrece conforme nos alejamos de los objetos.

Algoritmo 5: Mapa de costo

Data:

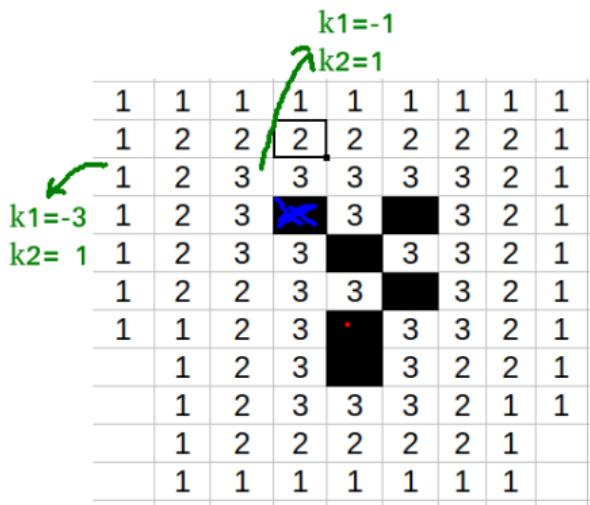
Mapa M de celdas de ocupación

Radio de costo r_c

Result: Mapa de costo M_c

M_c = Copia de M

```
foreach i ∈ [0, ..., rows) do
    foreach j ∈ [0, ..., cols) do
        //Si está ocupada, calcular el costo de  $r_c$  celdas alrededor.
        if M[i, j] == 100 then
            foreach k1 ∈ [-rc, ..., rc] do
                foreach k2 ∈ [-rc, ..., rc] do
                    C = rc - max(|k1|, |k2|) + 1
                    Mc[i + k1, j + k2] = max(C, Mc[i + k1, j + k2])
                end
            end
        end
    end
end
```



El algoritmo A*

- ▶ Es un algoritmo completo, es decir, si la ruta existe, seguro la encontrará, y si no existe, lo indicará en tiempo finito.
- ▶ Al igual que Dijkstra, A* encuentra una ruta que minimiza una función de costo, es decir, es un algoritmo óptimo.
- ▶ Es un algoritmo del tipo de búsqueda informada, es decir, utiliza información sobre el estimado del costo restante para llegar a la meta para priorizar la expansión de ciertos nodos.
- ▶ El nodo a expandir se selecciona de acuerdo con la función:

$$f(n) = g(n) + h(n)$$

donde

- ▶ $g(n)$ es el costo acumulado del nodo n
- ▶ $h(n)$ es una función heurística que **subestima** el costo de llegar del nodo n al nodo meta n_g .
- ▶ Se tienen los siguientes conjuntos importantes:
 - ▶ Lista abierta: conjunto de todos los nodos en la frontera (visitados pero no conocidos). Es una cola con prioridad donde los elementos son los nodos y la prioridad es el valor $f(n)$.
 - ▶ Lista cerrada: conjunto de nodos para los cuales se ha calculado una ruta óptima.
- ▶ A cada nodo se asocia un valor $g(n)$, un valor $f(n)$ y un nodo padre $p(n)$.

El algoritmo A*

Data: Mapa M , nodo inicial n_s con configuración q_s , nodo meta n_g con configuración q_g

Result: Ruta óptima $P = [q_s, q_1, q_2, \dots, q_g]$

Lista abierta $OL = \emptyset$ y lista cerrada $CL = \emptyset$

Fijar $f(n_s) = 0$, $g(n_s) = 0$ y $prev(n_s) = NULL$

Agregar n_s a OL y fijar nodo actual $n_c = n_s$

while $OL \neq \emptyset$ y $n_c \neq n_g$ **do**

 Remover de OL el nodo n_c con el menor valor f y agregar n_c a CL

forall n vecino de n_c **do**

$g = g(n_c) + costo(n_c, n)$

if $g < g(n)$ **then**

$g(n) = g$

$f(n) = h(n) + g(n)$

$prev(n) = n_c$

end

end

 Agregar a OL los vecinos de n_c que no estén ya en OL ni en CL

end

if $n_c \neq n_g$ **then**

 | Anunciar Falla

end

while $n_c \neq NULL$ **do**

 | Insertar al inicio de la ruta P la configuración correspondiente al nodo n_c

 | $n_c = prev(n_c)$

end

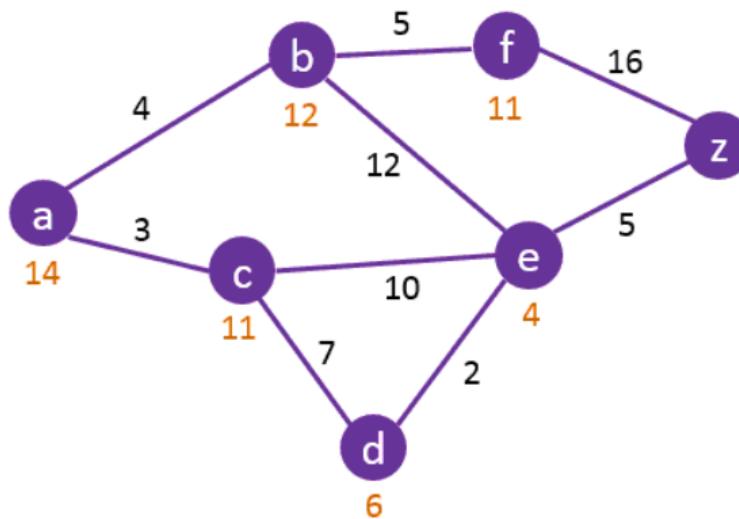
Devolver ruta óptima P

El algoritmo A*

- ▶ La función de costo será el número de celdas más el mapa de costo obtenido anteriormente.
- ▶ Puesto que el mapa está compuesto por celdas de ocupación, los nodos vecinos se pueden obtener usando conectividad 4 o conectividad 8.
- ▶ Si se utiliza conectividad 4, la distancia de Manhattan es una buena heurística.
- ▶ Si se utiliza conectividad 8, se debe usar la distancia Euclídea.
- ▶ La lista abierta se puede implementar con una *Heap*, de este modo, la inserción de los nodos n se puede hacer en tiempo logarítmico y la selección del nodo con menor f se hace en tiempo constante.

El algoritmo A*

Ejemplo: ¿Cuál es la ruta óptima del nodo A al nodo Z?



El algoritmo A*

Paso	Nodo actual	Lista Cerrada	Lista abierta
0	NULL	\emptyset	{A}
1	A	{A}	{B, C}
2	C	{A, C}	{B, D, E}
3	B	{A, C, B}	{D, E, F}
4	D	{A, C, B, D}	{E, F}
5	E	{A, C, B, D, E}	{F, Z}
6	Z	{A, C, B, D, E, Z}	{F }

A g,f,p	B g,f,p	C g,f,p	D g,f,p	E g,f,p	F g,f,p	Z g,f,p
0,0,NULL	$\infty, \infty, \text{NULL}$					
0,0,NULL	4, 16, A	3, 14, A	$\infty, \infty, \text{NULL}$	$\infty, \infty, \text{NULL}$	$\infty, \infty, \text{NULL}$	$\infty, \infty, \text{NULL}$
0,0,NULL	4, 16, A	3, 14, A	10, 16, C	13, 17, C	$\infty, \infty, \text{NULL}$	$\infty, \infty, \text{NULL}$
0,0,NULL	4, 16, A	3, 14, A	10, 16, C	13, 17, C	9, 20, B	$\infty, \infty, \text{NULL}$
0,0,NULL	4, 16, A	3, 14, A	10, 16, C	12, 16, D	9, 20, B	$\infty, \infty, \text{NULL}$
0,0,NULL	4, 16, A	3, 14, A	10, 16, C	12, 16, D	9, 20, B	17, 17, E
0,0,NULL	4, 16, A	3, 14, A	10, 16, C	12, 16, D	9, 20, B	17, 17, E

Práctica 01 - Planeación de rutas

Ver instrucciones en Classroom

Deadline: 2024-02-29 al inicio de la clase.

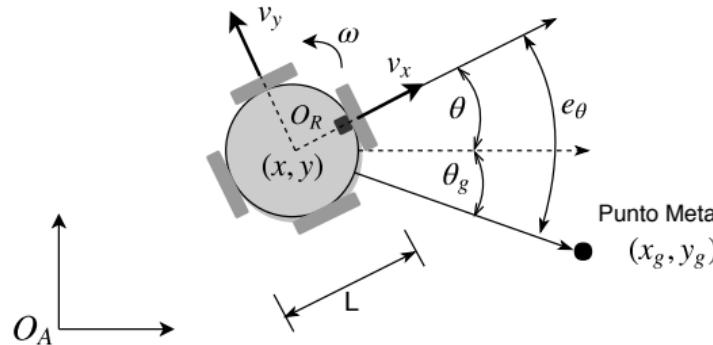
Seguimiento de rutas

Hasta el momento ya se tiene una representación del ambiente y una forma de planear rutas. Ahora falta diseñar las leyes de control que hagan que el robot se mueva por la ruta calculada. Este control se hará bajo los siguientes supuestos:

- ▶ Se conoce la posición del robot (más adelante se abordará el problema de la localización)
- ▶ El modelo cinemático es suficiente para modelar el movimiento del robot
- ▶ Las dinámicas no modeladas (parte eléctrica y mecánica de los motores) son lo suficientemente rápidas para poder despreciarse

Modelo cinemático

Considere la base móvil omnidireccional de la figura con configuración $q = (x, y, \theta)$.



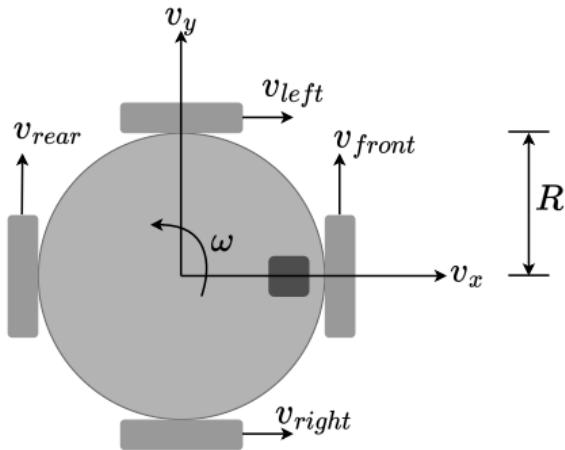
El modelo cinemático está dado por

$$\begin{aligned}\dot{x} &= v_x \cos \theta - v_y \sin \theta \\ \dot{y} &= v_x \sin \theta + v_y \cos \theta \\ \dot{\theta} &= \omega,\end{aligned}$$

- ▶ (v_x, v_y, ω) se consideran como señales de control
- ▶ Corresponden a las velocidades lineales frontal y lateral, y la velocidad angular, con respecto al robot.
- ▶ La forma de convertir (v_x, v_y, ω) a velocidades de cada motor varía dependiendo del número de motores y de su posición.

Base omnidireccional de 4 ruedas

Dada una base omnidireccional con las ruedas colocadas como se muestra en la figura, las velocidades de cada llanta se pueden obtener como:



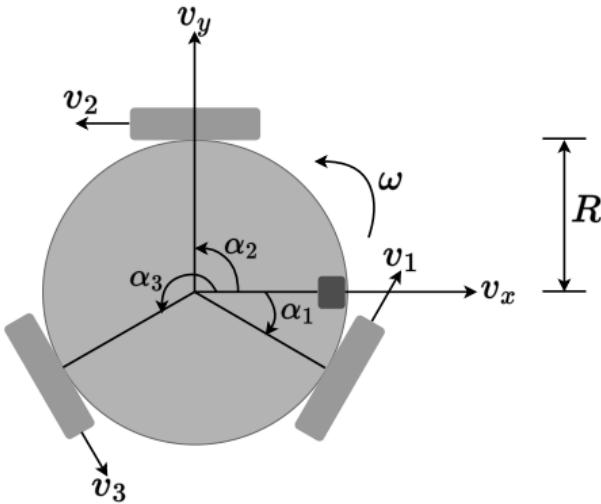
$$\begin{bmatrix} v_{left} \\ v_{right} \\ v_{front} \\ v_{rear} \end{bmatrix} = \begin{bmatrix} 1 & 0 & -R \\ 1 & 0 & +R \\ 0 & 1 & +R \\ 0 & 1 & -R \end{bmatrix} \begin{bmatrix} v_x \\ v_y \\ \omega \end{bmatrix}$$

- ▶ Las velocidades de las llantas son lineales.
- ▶ Para obtener las velocidades angulares, basta con dividir entre el radio de las llantas.

- ▶ Como se puede observar, la matriz anterior no tiene inversa.
- ▶ Se tienen cuatro velocidades de llantas en función de tres variables (v_x, v_y, ω)
- ▶ Esto significa que dadas tres velocidades de llantas, **la cuarta no puede ser cualquier valor**

Base omnidireccional de 3 ruedas

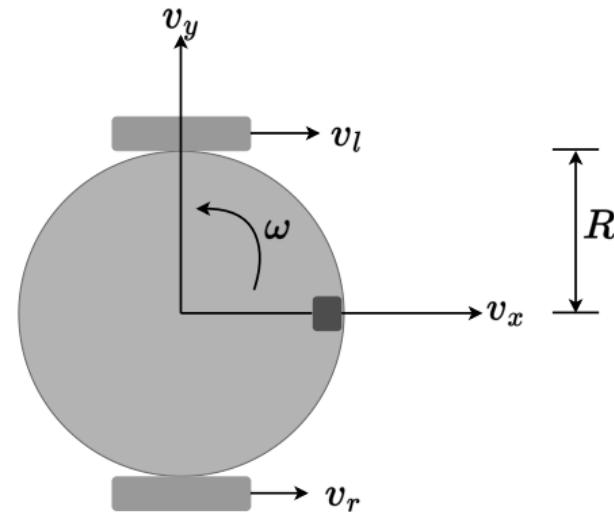
- ▶ La base de 4 ruedas omnidireccionales tiene la ventaja de tener mejor tracción y de lograr movimientos rectos más fácilmente.
- ▶ Tiene la desventaja de que las velocidades pueden indeterminarse si no están bien calculadas.
- ▶ Una base omnidireccional también puede lograrse con 3 ruedas:



$$\begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix} = \begin{bmatrix} -\sin \alpha_1 & \cos \alpha_1 & R \\ -\sin \alpha_2 & \cos \alpha_2 & R \\ -\sin \alpha_3 & \cos \alpha_3 & R \end{bmatrix} \begin{bmatrix} v_x \\ v_y \\ \omega \end{bmatrix}$$

Base diferencial

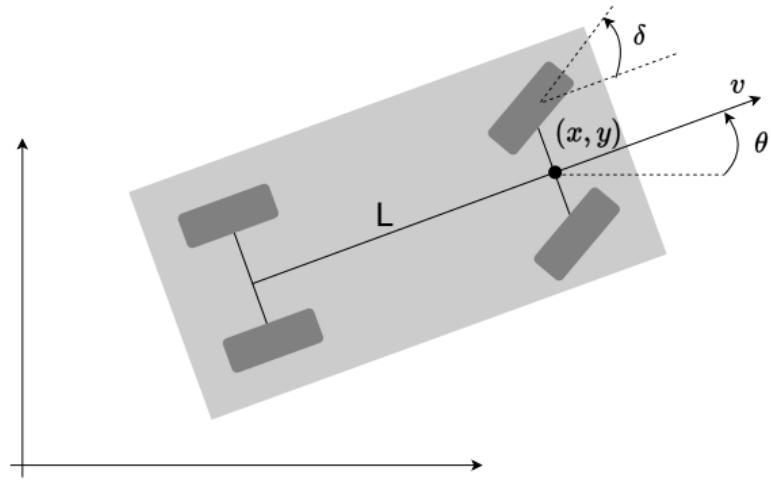
- ▶ Con una base diferencial ya no se tiene movimiento omnidireccional, es decir, se tiene movimiento no holonómico.
- ▶ El robot ya solo puede tener velocidad frontal v_x pero no velocidad lateral v_y .



$$v_l = v_x - R\omega$$

$$v_r = v_x + R\omega$$

Esta es la configuración más fácil de lograr debido a la simplicidad del hardware necesario.



$$\begin{aligned}\dot{x} &= v \cos(\theta + \delta) \\ \dot{y} &= v \sin(\theta + \delta) \\ \dot{\theta} &= \frac{v}{L} \sin(\delta)\end{aligned}$$

- ▶ Es el modelo simplificado de un coche (en un auto real, δ no es la misma para cada llanta)
- ▶ Las entradas de control son (v, δ) , es decir, la acción del acelerador y el freno y el ángulo del volante
- ▶ Las restricciones de movimiento se deben considerar para la planeación de rutas, ya que, por ejemplo, el auto no puede girar sobre su centro.

Tarea 05 - Modelos cinemáticos

Ver instrucciones en Classroom

Deadline: 2024-08-29 al inicio de la clase.

Control de posición

- ▶ Las leyes de control se diseñarán considerando una base diferencial
- ▶ Es mejor mover al robot así, pues los sensores están generalmente al frente

Si se quiere alcanzar el punto meta (x_g, y_g) , las siguientes leyes de control siguientes permiten alcanzar dicho punto meta:

$$\begin{aligned}v_x &= v_{max} e^{-\frac{e_\theta^2}{\alpha}} \\ \omega &= \omega_{max} \left(\frac{2}{1 + e^{-\frac{e_\theta}{\beta}}} - 1 \right)\end{aligned}$$

con

$$e_\theta = \text{atan2}(y_g - y, x_g - x) - \theta$$

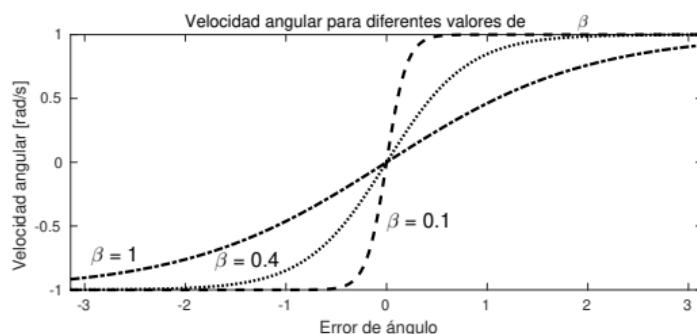
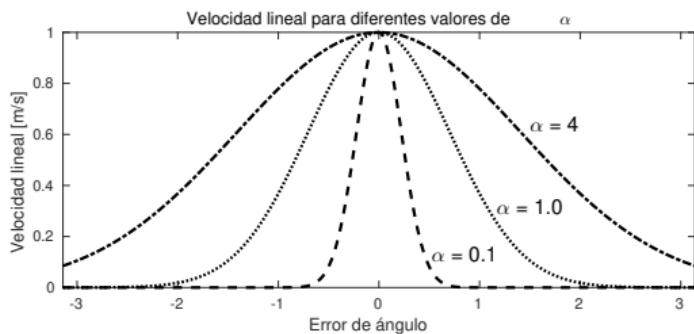
El error de ángulo e_θ debe estar siempre en el intervalo $(-\pi, \pi]$. Si la diferencia resulta en un valor fuera de este ángulo, se puede acotar mediante:

$$e_\theta \leftarrow (e_\theta + \pi) \% (2\pi) - \pi$$

donde $\%$ denota el operador módulo (residuo).

Control de posición

- ▶ v_{max} y ω_{max} son las velocidades linear y angular máximas y dependen de las capacidades físicas del robot.
- ▶ α y β determinan qué tan rápido varían dichas velocidades cuando cambia el error de ángulo.
- ▶ En general, valores pequeños de α y β logran que el robot alcance el punto meta casi en línea recta, sin embargo, valores muy pequeños pueden producir oscilaciones.
- ▶ Valores grandes de α y β producen un movimiento más suave pero pueden hacer que el robot describa curvas muy extensas.



Control de base omnidireccional

Seguimiento de rutas

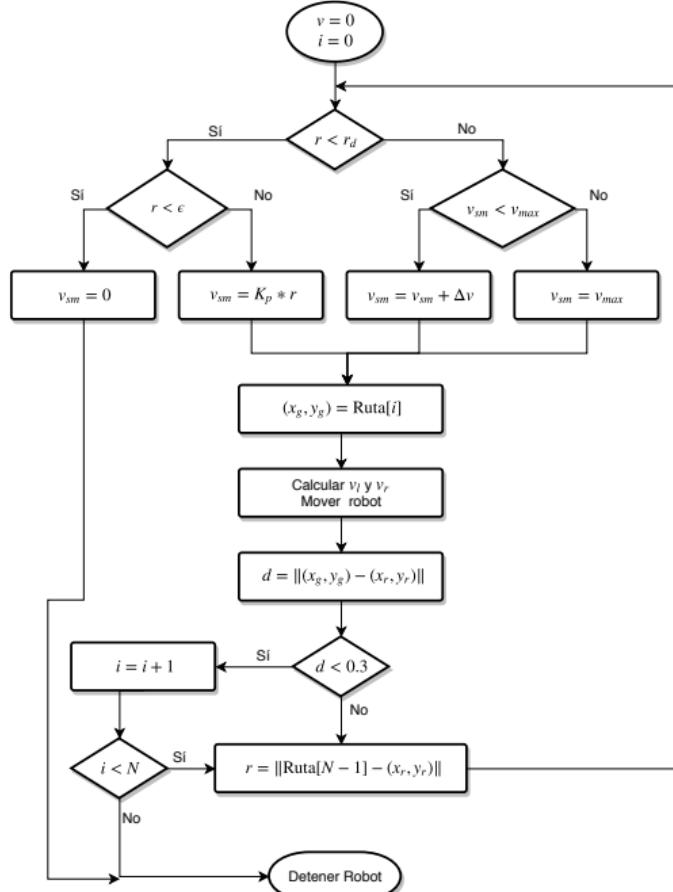
- ▶ Hasta el momento se ha planteado cómo alcanzar una posición, pero, ¿para una ruta?
- ▶ Las rutas son secuencias de puntos. Esta secuencia se podría parametrizar con respecto al tiempo para tener una trayectoria, sin embargo esto resulta muy complicado debido a la complejidad de las rutas.
- ▶ Una solución más sencilla es aplicar el control de posición para cada punto hasta recorrer toda la ruta.
- ▶ Las leyes de control solo dependen de e_θ por lo que el robot no desacelera al acercarse a la meta, provocando fuertes oscilaciones.
- ▶ Una forma de resolver este problema es ejecutar la ley de control sólo si la distancia al punto meta

$$d = \sqrt{(x_g - x_r)^2 + (y_g - y_r)^2}$$

es mayor que una tolerancia ϵ .

- ▶ En este caso, el robot se detendrá abruptamente cuando el error de distancia sea menor que ϵ , lo cual tampoco es deseable
- ▶ Una forma fácil de hacer que el robot acelere y desacelere, o en general, obtener un perfil de velocidad, es mediante el uso de una máquina de estados

Perfil de velocidad



Considere una máquina de estados que calcule v_{max} en el control. Sea v_{sm} la nueva velocidad lineal máxima, de modo que ahora se tiene:

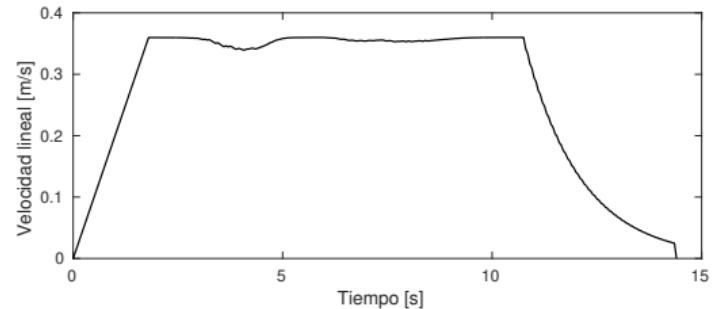
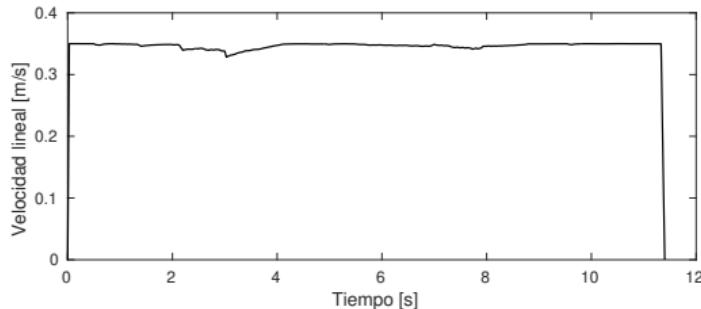
$$v = v_{sm} e^{-\frac{e^2 \theta}{\alpha}}$$
$$\omega = \omega_{max} \left(\frac{2}{1 + e^{-\frac{e \theta}{\beta}}} - 1 \right)$$

con

- ▶ r : Distancia a la meta global
- ▶ ϵ : Distancia a la que se considera que el robot alcanzó la meta global
- ▶ r_d : Distancia a la meta global para desacelerar
- ▶ Δv : Aceleración

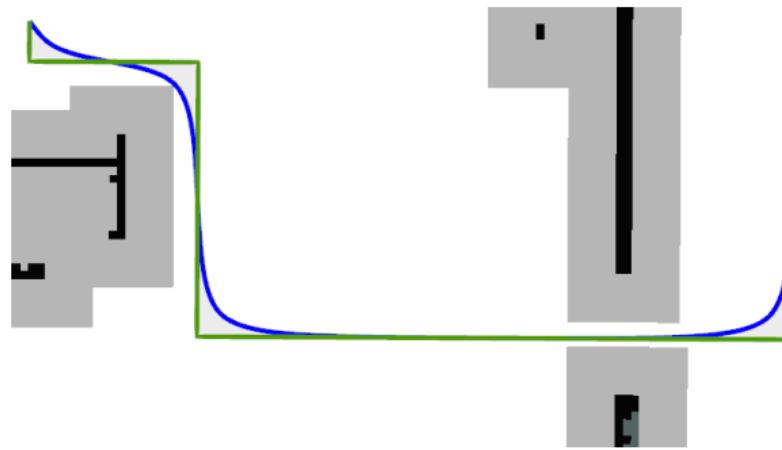
Perfil de velocidad

La siguiente figura muestra un ejemplo de una ruta y las velocidades lineales generadas usando solo las leyes de control (izquierda) y usando la máquina de estados para un perfil de velocidad (derecha).



Suavizado de rutas

- ▶ Puesto que las rutas se calcularon a partir de celdas de ocupación, están compuestas de esquinas.
- ▶ La esquinas no son deseables, pues suelen generar cambios bruscos en las señales de control.
- ▶ La ruta verde de la imagen es una muestra de una ruta calculada por A*.
- ▶ Es preferible una ruta como la azul.



Existen varias formas de suavizar la ruta generada:

- ▶ Splines
- ▶ Descenso del gradiente

Suavizado mediante splines

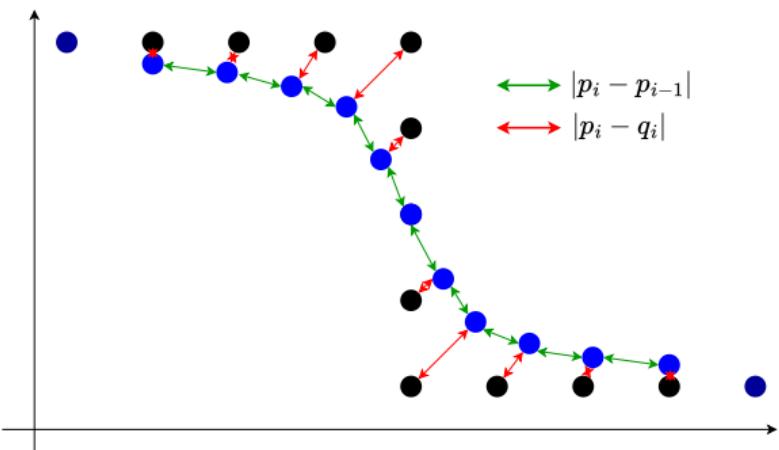
- ▶ Un *spline* es una función definida a tramos por polinomios.
- ▶ La forma más común son los splines de tercer grado o *cubic splines*
- ▶ Se ajusta un polinomio de tercer grado por cada par de puntos
- ▶ La derivada al final de un tramo debe ser igual a la derivada al inicio del siguiente tramo.
- ▶ Aplicando estas condiciones para cada par

Suavizado mediante descenso del gradiente

Otra forma de suavizar la ruta es planteando una función de costo y encontrando el mínimo. Los puntos negros representan la ruta de A* compuesta por los puntos $Q = \{q_0, q_1, \dots, q_n\}$ y los puntos azules representan una ruta suave $P = \{p_0, p_1, \dots, p_n\}$.

Considere la función de costo:

$$J = \alpha \frac{1}{2} \sum_{i=1}^{n-1} (p_i - p_{i-1})^2 + \beta \frac{1}{2} \sum_{i=1}^{n-1} (p_i - q_i)^2$$



- ▶ J es la suma de distancias entre un punto y otro de la ruta suavizada, y entre la ruta suavizada y la original.
- ▶ Si la ruta es muy suave, J es grande.
- ▶ Si la ruta es muy parecida a la original, J también es grande.
- ▶ Una ruta ni muy suave ni muy parecida a la original, logrará minimizar J .

Suavizado mediante descenso del gradiente

- ▶ Una forma de encontrar el mínimo es resolviendo $\nabla J(p) = 0$, y luego evaluando la matriz Hessiana para determinar si el punto crítico p_c es un mínimo.
- ▶ Esto se puede complicar debido al alto número de variables en p .
- ▶ Una forma más sencilla, es mediante el descenso del gradiente.

Algoritmo 6: Descenso del gradiente

Data: Función $J(p) : \mathbb{R}^n \rightarrow \mathbb{R}$ a minimizar

Result: Vector p que minimiza la función J

$p \leftarrow p_{init}$ //Fijar una estimación inicial

while $|\nabla J(p)| > tol$ **do**

$| \quad p \leftarrow p - \epsilon \nabla J(p)$ // p se modifica un poco en sentido contrario al gradiente.

end

Devolver p

El descenso del gradiente devuelve el mínimo local más cercano a la condición inicial p_0 . Pero la función de costo J tiene solo un mínimo global. El gradiente de la función de costo J se calcula como:

$$\left[\underbrace{\alpha(p_0 - p_1) + \beta(p_0 - q_0), \dots, \alpha(2p_i - p_{i-1} - p_{i+1}) + \beta(p_i - q_i), \dots, \alpha(p_{n-1} - p_{n-2}) + \beta(p_{n-1} - q_{n-1})}_{\frac{\partial J}{\partial p_0}}, \underbrace{\dots, \frac{\partial J}{\partial p_i}, \dots, \frac{\partial J}{\partial p_{n-1}}} \right]$$

Suavizado mediante descenso del gradiente

Para no variar los puntos inicial y final de la ruta, la primer y última componentes de ∇J se dejarán en cero. El algoritmo de descenso del gradiente queda como:

Algoritmo 7: Suavizado de rutas mediante descenso del gradiente

Data: Conjunto de puntos $Q = \{q_0 \dots q_i \dots q_{n-1}\}$ de la ruta original, parámetros α y β , ganancia ϵ y tolerancia tol

Result: Conjunto de puntos $P = \{p_0 \dots p_i \dots p_{n-1}\}$ de la ruta suavizada

$P \leftarrow Q$

$\nabla J_0 \leftarrow 0$

$\nabla J_{n-1} \leftarrow 0$

while $\|\nabla J(p_i)\| > tol \wedge steps < max_steps$ **do**

foreach $i \in [1, n - 1]$ **do**

$\nabla J_i \leftarrow \alpha(2p_i - p_{i-1} - p_{i+1}) + \beta(p_i - q_i)$

end

$P \leftarrow P - \epsilon \nabla J$

$steps \leftarrow steps + 1$

end

regresar P

Tarea 06 - Seguimiento de rutas

Ver instrucciones en Classroom

Deadline: 2024-09-03 al inicio de la clase.

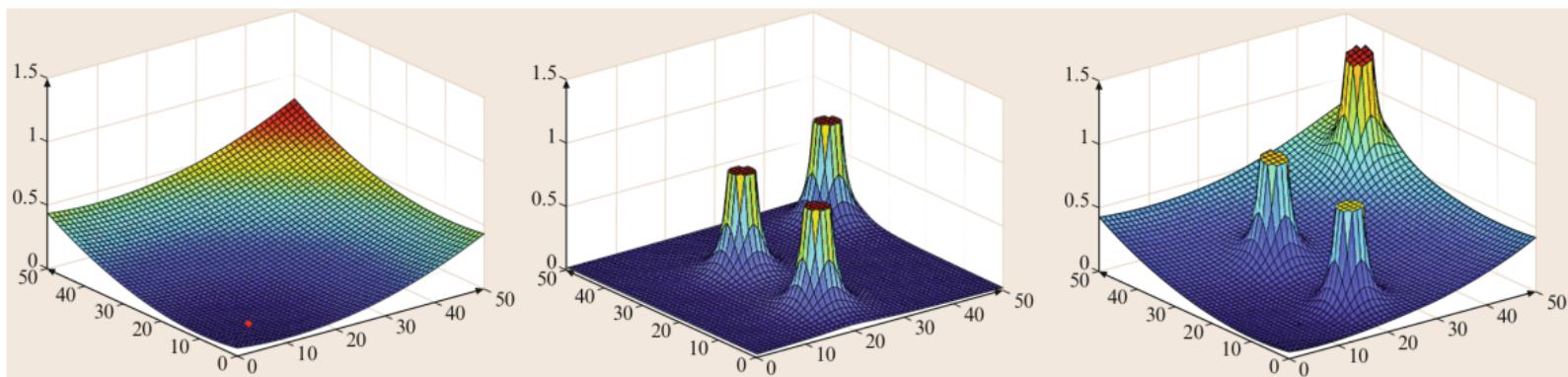
Evasión de obstáculos

- ▶ Hasta el momento se tiene una manera de representar el ambiente, planear una ruta y seguirla
- ▶ ¿Qué pasa si en el ambiente hay un obstáculo que no estaba en el mapa?
- ▶ Se requiere de una técnica reactiva para evadir obstáculos
- ▶ Una posible solución es el uso de campos potenciales artificiales

El objetivo de esta técnica es diseñar una función $U(q) : \mathbb{R}^n \rightarrow \mathbb{R}$ que represente energía potencial.

- ▶ El gradiente $\nabla U(q) = \left[\frac{\partial U}{\partial q_1}, \dots, \frac{\partial U}{\partial q_n} \right]$ es una fuerza.
- ▶ Se debe diseñar de modo que tenga un mínimo global en el punto meta y máximos locales en cada obstáculo.
- ▶ Si el robot se mueve siempre en sentido contrario al gradiente ∇U llegará al punto meta siguiendo una ruta alejada de los obstáculos.
- ▶ Hay varias formas de diseñar la función $U(q)$, algunas son:
 - ▶ Algoritmo *wavefront*, requiere una discretización del espacio (requiere mapa previo), pero no presenta mínimos locales.
 - ▶ Campos atractivos y repulsivos, no requieren mapa previo, pero pueden presentar mínimos locales.

Potenciales atractivos y repulsivos



- ▶ **Campos repulsivos:** Por cada obstáculo se diseña una función $U_{rej_i}(q)$ con un máximo local en la posición q_{o_i} del obstáculo.
- ▶ **Campo atractivo:** Se diseña una función $U_{att}(q)$ con un mínimo global en el punto meta q_g .
- ▶ La función potencial total $U(q)$ se calcula como

$$U(q) = U_{att}(q) + \frac{1}{N} \sum_{i=1}^N U_{rej_i}(q)$$

Fuerzas atractiva y repulsivas

Puesto que el gradiente es un operador lineal, se pueden diseñar directamente las fuerzas atractiva $F_{att}(q) = \nabla U_{att}(q)$ y repulsivas $F_{rej_i}(q) = \nabla U_{rej_i}(q)$, de modo que la fuerza total será:

$$\nabla U(q) = F(q) = F_{att}(q) + \frac{1}{N} \sum_{i=1}^N F_{rej_i}(q)$$

Una propuesta de estas fuerzas es:

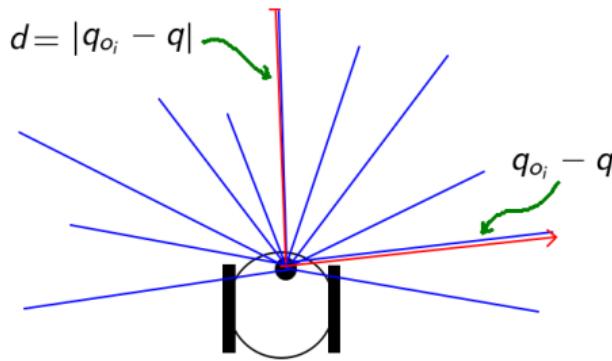
$$F_{att} = \eta \frac{(q - q_g)}{\|q - q_g\|}, \quad \zeta > 0$$
$$F_{rej} = \begin{cases} \zeta \left(\sqrt{\frac{1}{d} - \frac{1}{d_0}} \right) \frac{q_{o_i} - q}{d} & \text{si } d < d_0 \\ 0 & \text{en otro caso} \end{cases}$$

donde

- ▶ $q = (x, y)$ es la posición del robot
- ▶ $q_g = (x_g, y_g)$ es el punto que se desea alcanzar
- ▶ $q_{o_i} = (x_{o_i}, y_{o_i})$ es la posición del i -ésimo obstáculo
- ▶ d_0 es una distancia de influencia. Más allá de d_0 los obstáculos no producen efecto alguno
- ▶ ζ y η , junto con d_0 , son constantes de sintonización

Evasión de obstáculos por campos potenciales

- ▶ Aunque las ecuaciones anteriores suponen que se conoce la posición de cada obstáculo q_{oi} , en realidad ésta aparece siempre en la diferencia $q_{oi} - q$, es decir, solo se requiere su posición relativa al robot.
- ▶ Los campos potenciales se implementan utilizando el lidar, donde cada lectura se considera un obstáculo.



Las lecturas del lidar generalmente son pares distancia-ángulo (d_i, θ_i) expresados con respecto al robot, por lo que, si se conoce la posición del robot (x_r, y_r, θ_r) , la posición de cada obstáculo se puede calcular como:

$$\begin{aligned}x_{oi} &= x_r + d_i \cos(\theta_i + \theta_r) \\y_{oi} &= y_r + d_i \sin(\theta_i + \theta_r)\end{aligned}$$

Evasión de obstáculos por campos potenciales

Finalmente, para que el robot alcance el punto de menor potencial, se puede emplear el descenso del gradiente:

Algoritmo 8: Descenso del gradiente para mover al robot a través de un campo potencial.

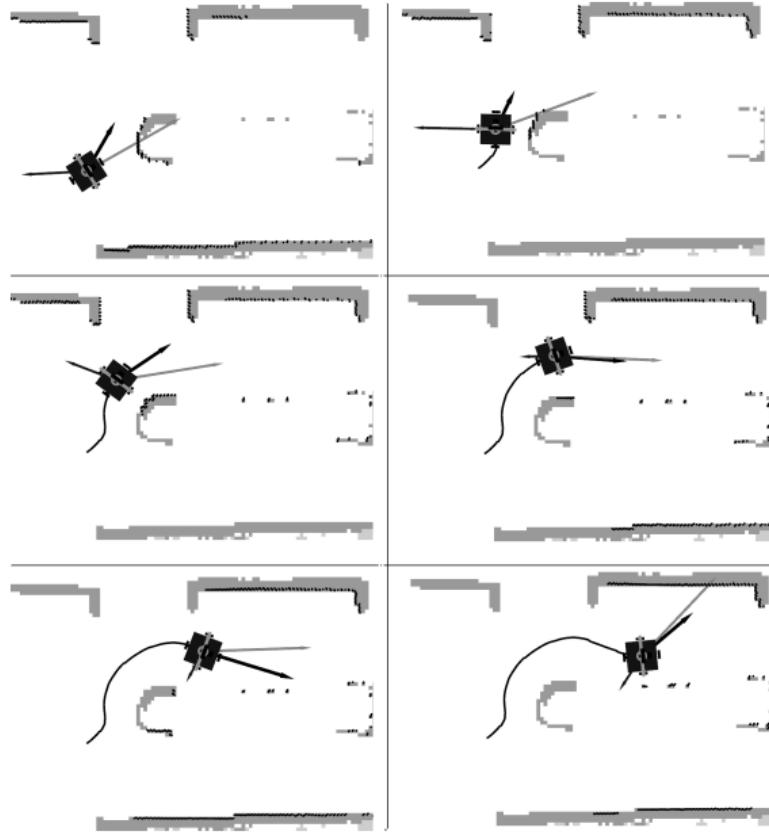
Data: Posición inicial q_s , posición meta q_g , posiciones q_{oi} de los obstáculos y tolerancia tol

Result: Secuencia de puntos $\{q_0, q_1, q_2, \dots\}$ para evadir obstáculos y alcanzar el punto meta

```
q ←  $q_s$ 
while  $\|\nabla U(q)\| > tol$  do
    |   q ←  $q - \epsilon F(q)$ 
    |   [ $v, \omega$ ] ← leyes de control con  $q$  como posición deseada
end
```

Evasión de obstáculos por campos potenciales

Ejemplo de movimiento:



Tarea 08 - Campos potenciales

Ver instrucciones en Classroom

Deadline: 2024-03-12 al inicio de la clase.

Práctica 02 - Navegación autónoma

Ver instrucciones en Classroom

Deadline: 2024-09-10 al inicio de la clase.

El problema de la localización consiste en determinar la configuración q del robot dada un mapa y un conjunto de lecturas de los sensores.

- ▶ La localización se podría lograr simplemente integrando los comandos de velocidad del robot.
- ▶ Si se conoce perfectamente la configuración inicial y el robot ejecuta perfectamente los comandos de movimiento, entonces la simple integración de la velocidad de los motores sería suficiente.
- ▶ Esto por supuesto no es posible. Se tiene incertidumbre tanto en la estimación inicial de la posición como en la ejecución de cada movimiento.
- ▶ Es decir, el robot pierde información sobre su posición en cada movimiento.

Existen principalmente dos tipos:

Localización local:

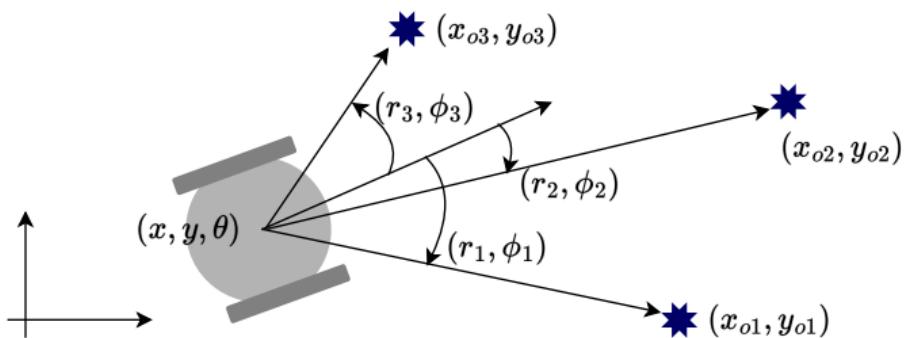
- ▶ Requiere una estimación inicial *cercana* a la posición real del robot, de otro modo, no converge.
- ▶ Suele ser menos costosa computacionalmente.
- ▶ Un método común es el Filtro de Kalman Extendido.

Localización global:

- ▶ La estimación inicial puede ser cualquiera.
- ▶ Suele ser computacionalmente costosa.
- ▶ Un método común son los Filtros de Partículas.

- ▶ En la localización probabilística, en lugar de llevar una sola estimación sobre la posición del robot, se mantiene una *distribución de probabilidad* sobre todo el espacio de hipótesis.
- ▶ El enfoque probabilístico permite manejar las incertidumbres inherentes al movimiento y al sensado.
- ▶ El reto es obtener una distribución de densidad de probabilidad (PDF) sobre todas las posibles posiciones del robot.
- ▶ En general, los métodos probabilísticos de estimación se componen de dos pasos:
 1. **Predicción:** Se modifica la PDF de la posición del robot con base en los comandos y el modelo de movimiento.
 2. **Actualización:** Se corrige la predicción mezclando la información de PDF predicha con información de los sensores. Se obtiene una PDF de la posición y se repite el proceso.

El Filtro de Kalman Extendido



Modelo de observación:

Modelo de transición de estados:

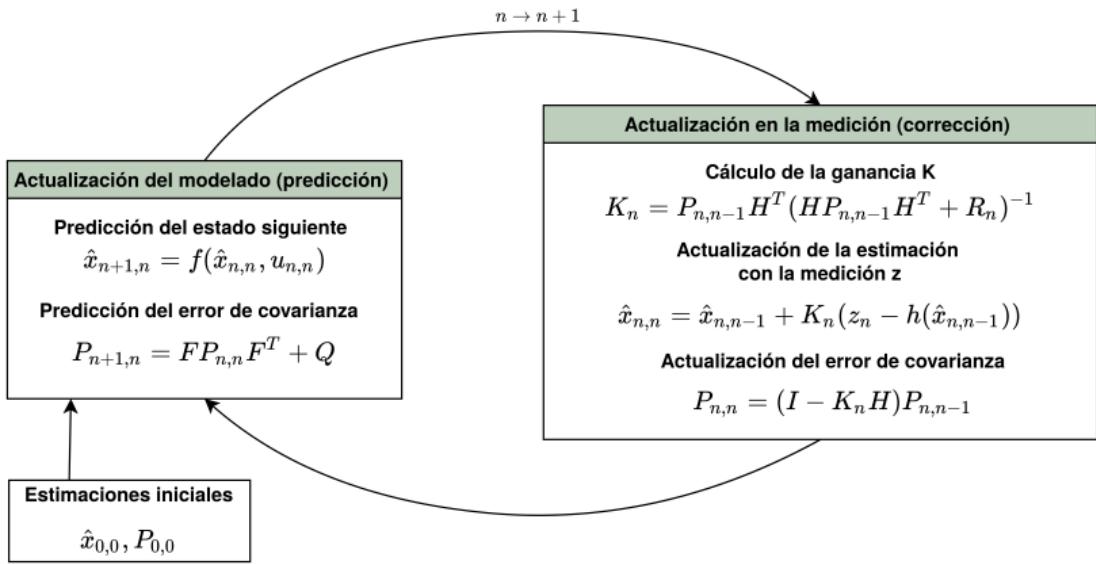
$$\begin{aligned} x_{k+1} &= x_k + (\Delta t)v \cos(\theta_k) & +n_{p1} \\ y_{k+1} &= y_k + (\Delta t)v \sin(\theta_k) & +n_{p2} \\ \theta_{k+1} &= \theta_k + (\Delta t)\omega & +n_{p3} \end{aligned} \left. \right\} f(x)$$

donde $n_p \in \mathbb{R}^3$ es ruido de proceso que es distribuye normalmente con media cero y matriz de covarianza conocida $Q \in \mathbb{R}^{3 \times 3}$

$$\begin{aligned} r_i &= \sqrt{(x_k - x_{oi})^2 + (y_k - y_{oi})^2} & +n_{m1} \\ \phi_i &= \text{atan2}(y_k - y_{oi}, x_k - x_{oi}) - \theta_k & +n_{m2} \\ &\vdots & \end{aligned} \left. \right\} h(x)$$

donde M es el número de marcas observadas, $n_m \in \mathbb{R}^{2M}$ es ruido de medición que se distribuye normalmente con media cero y matriz de covarianza conocida $R \in \mathbb{R}^{2M \times 2M}$

Algoritmo de estimación del EKF



con:

- ▶ P : Matriz de covarianza del error de estimación
- ▶ F : Jacobiano de $f(x)$
- ▶ H : Jacobiano de $h(x)$

El EKF como un filtro Bayesiano

- ▶ El EKF obtiene una distribución de probabilidad *a posteriori* $P(A|B)$: la probabilidad de que el robot esté en una posición dadas las lecturas de los sensores.
- ▶ El modelo de observación representa la verosimilitud $P(B|A)$: la probabilidad de obtener ciertas lecturas dado un cierto estado del robot.
- ▶ La estimación dada por \hat{x}_{k+1} es la distribución *a priori* $P(A)$
- ▶ EKF supone una distribución normal, es decir \hat{x}_{k+1} es una variable aleatoria que se distribuye normalmente con media en $f(x_k, u_k)$ y covarianza Q

Tarea 08 - Filtro de Kalman Extendido

Ver instrucciones en Classroom

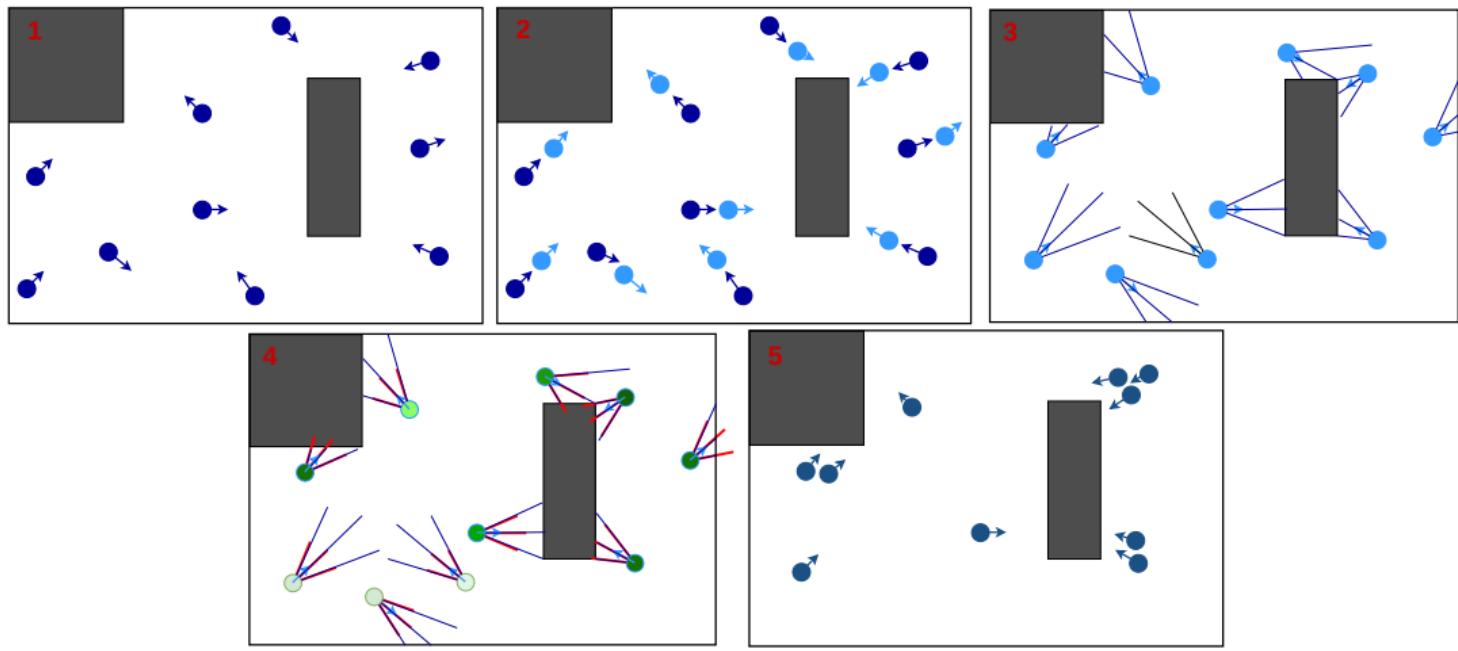
Deadline: 2024-03-12 al inicio de la clase.

- ▶ También son un filtro Bayesiano
- ▶ A diferencia del EKF que supone una distribución normal, los filtros de partículas suponen una distribución no paramétrica.
- ▶ Se consideran N suposiciones de la posición del robot. A cada suposición (x, y, θ) se le llama partícula.
- ▶ La distribución *a priori* se calcula realizando simulaciones de movimiento para cada partícula.
- ▶ El modelo de observación también se realiza mediante simulaciones de los sensores para cada partículas.
- ▶ Para obtener la distribución *a posteriori* se comparan las lecturas simuladas con la lectura del sensor real y se realiza un muestreo aleatorio con reemplazo. Cada partícula tiene una probabilidad de ser muestreada, proporcional a la similitud de su lectura simulada con el sensor real.

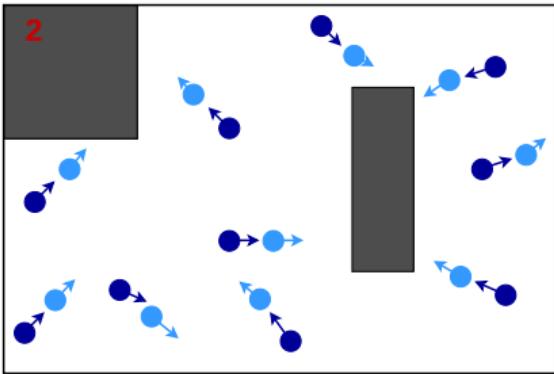
Un filtro de partículas para localización consta de 5 pasos generales:

1. Generar N partículas con (x, y, θ) aleatorias con distribución uniforme (distribución inicial equivalente a \hat{x}_0 en el EKF)
2. Mover cada partícula un desplazamiento $(\Delta x, \Delta y, \Delta\theta)$ más ruido Gaussiano (equivalente a estimar $\hat{x}_{k+1} = f(x_k, u_k)$ en el EKF)
3. Simular las lecturas del sensor láser para cada partícula (equivalente a predecir las salidas $h(x_k)$ en el EKF)
4. Comparar cada lectura simulada con la lectura del sensor real. Las similitudes representan una distribución de probabilidad
5. Remuestrear N partículas con reemplazo usando la distribución anterior, y agregar ruido Gaussiano (equivalente a la actualización de la estimación de \hat{x} en el EKF)

Filtros de Partículas



Desplazamiento de partículas



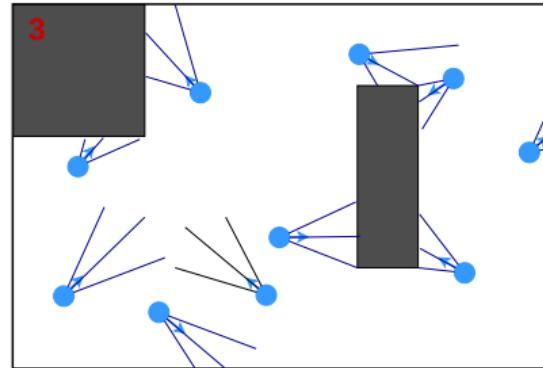
- ▶ Se puede suponer que todas las partículas se desplazan una cantidad ($\Delta x, \Delta y, \Delta\theta$) (obtenida mediante odometría) con respecto al sistema del robot
- ▶ Para calcular la nueva pose de cada partícula, basta con transformar los desplazamientos al sistema de referencia (giro sobre Z de un ángulo θ)
- ▶ Para modelar la incertidumbre del movimiento, a cada partícula se le suma ruido Gaussiano n con media cero y varianza σ_m^2 :

Para cada partícula, hacer:

$$\begin{aligned}x_{k+1} &= x_k + \Delta x \cos \theta_k - \Delta y \sin \theta_k + n_x \\y_{k+1} &= y_k + \Delta x \sin \theta_k + \Delta y \cos \theta_k + n_y \\\theta_{k+1} &= \theta_k + \Delta\theta + n_\theta\end{aligned}$$

Simulación del sensor

- ▶ Se puede simular con técnicas de *ray tracing* a partir de un mapa y una pose del sensor
- ▶ Se puede utilizar el paquete `occupancy_grid_utils`. Tiene una biblioteca que simula las lecturas del láser dado:
 - ▶ Un mapa de celdas de ocupación (`OccupancyGrid`)
 - ▶ Una posición y orientación del sensor (`Pose`). Esta posición y orientación corresponde con el (x, y, θ) de cada partícula.
 - ▶ Especificaciones del sensor a simular. Esto lo obtiene de un mensaje `LaserScan`
- ▶ Puesto que esta simulación es computacionalmente costosa, solo se simulan **1 de cada N** lecturas del sensor real.



Comparación de simulación con real

Para obtener una medida de comparación se obtiene una media de diferencias en las distancias:

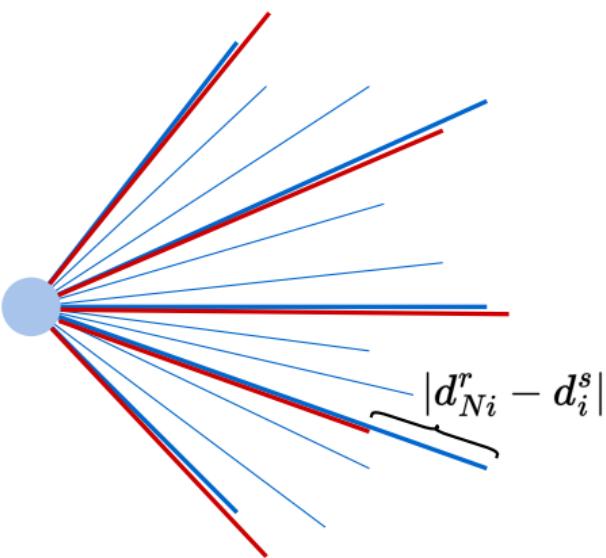
$$\delta = \sum_{i=0}^{M_s-1} |d_{Ni}^r - d_i^s|$$

$$s = e^{-\delta^2/\sigma^2}$$

con:

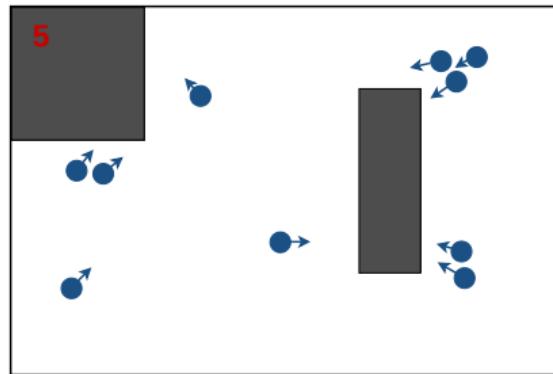
- ▶ d_{Ni}^r : es la (Ni)-ésima lectura de distancia del sensor real
- ▶ d_i^s : es la i -ésima lectura de distancia del sensor simulado (de una partícula)
- ▶ δ : media de los valores absolutos de las diferencias
- ▶ σ^2 : Varianza del sensor (medida de incertidumbre)
- ▶ s : medida de similitud entre el sensor real y el sensor simulado
- ▶ M_s : número de distancias en el sensor simulado ($1/N$ del número de distancias en el sensor real).

Recuerde que se simularon solo 1 de cada N lecturas del sensor real



Remuestreo con reemplazo

- ▶ Las similitudes obtenidas en el paso anterior pueden usarse como distribución de probabilidad si se normalizan para que sumen 1
- ▶ La distribución de probabilidad anterior determina la probabilidad de que una partícula sea remuestreada
- ▶ Para evitar tener partículas repetidas, a cada partícula remuestreada se le suma ruido Gaussiano



Práctica 03 - Localización probabilística

Ver instrucciones en Classroom

Deadline: 2024-03-19 al inicio de la clase.

¿Qué es la visión computacional?

- ▶ **Visión Humana:** Se puede concebir como una tarea de procesamiento de información, que obtiene significado a partir de los estímulos percibidos por los ojos.
- ▶ **Visión Computacional:** Desarrollo de programas de computadora que puedan *interpretar* imágenes. Es decir, realizar la visión humana por medios computacionales.



"Visión es un proceso que produce, a partir de imágenes del mundo externo, una descripción que es útil para el observador y que está libre de información irrelevante." (Marr, 1976).

El fenómeno de la visión lo podemos considerar como el producto de un sistema de procesamiento de información.

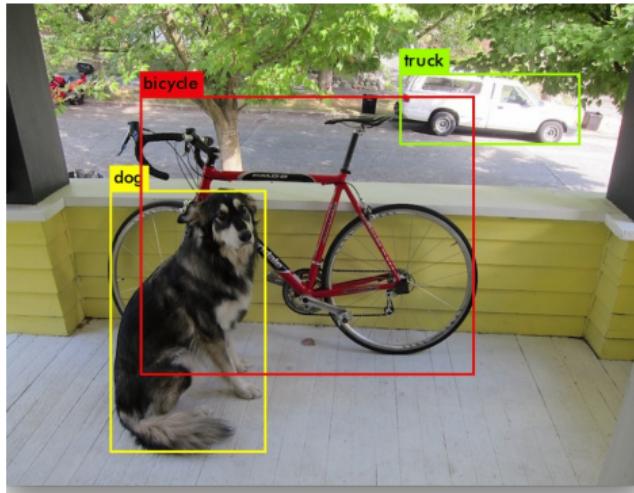
Marr propone los siguientes tres niveles de construcción de un sistema de procesamiento de información:

1. Teoría Computacional (¿Cuál es el problema por resolver?)
2. Representación y algoritmos (Estrategia usada para resolverlo)
3. Implementación (Realización física, software y hardware)

Es decir, la visión computacional sería un proceso parecido a la visión humana, similar en los niveles computacionales y de algoritmos, pero implementado de forma diferente: en hardware de procesamiento con sensores de visión.

Visión Computacional

Por lo tanto, la tarea de la Visión por computadora es la construcción de descriptores de la escena con base en características relevantes contenidas en una imagen:



- ▶ Objetos
- ▶ Formas de Superficies
- ▶ Colores
- ▶ Texturas
- ▶ Movimientos
- ▶ Iluminación
- ▶ Reflejos

Vision Computacional vs Proc de Imágenes

1. Procesamiento de imágenes: Es cualquier forma de procesamiento de señales donde la entrada es una imagen, la salida puede ser otra imagen o un conjunto de características o parámetros relacionados con la misma.
2. Visión Computacional: Estudio y aplicación de métodos que permiten a las computadoras “entender” el contenido de una imagen.
3. Visión Máquina: Es la aplicación de la visión por computadora en la industria y procesos de manufactura.

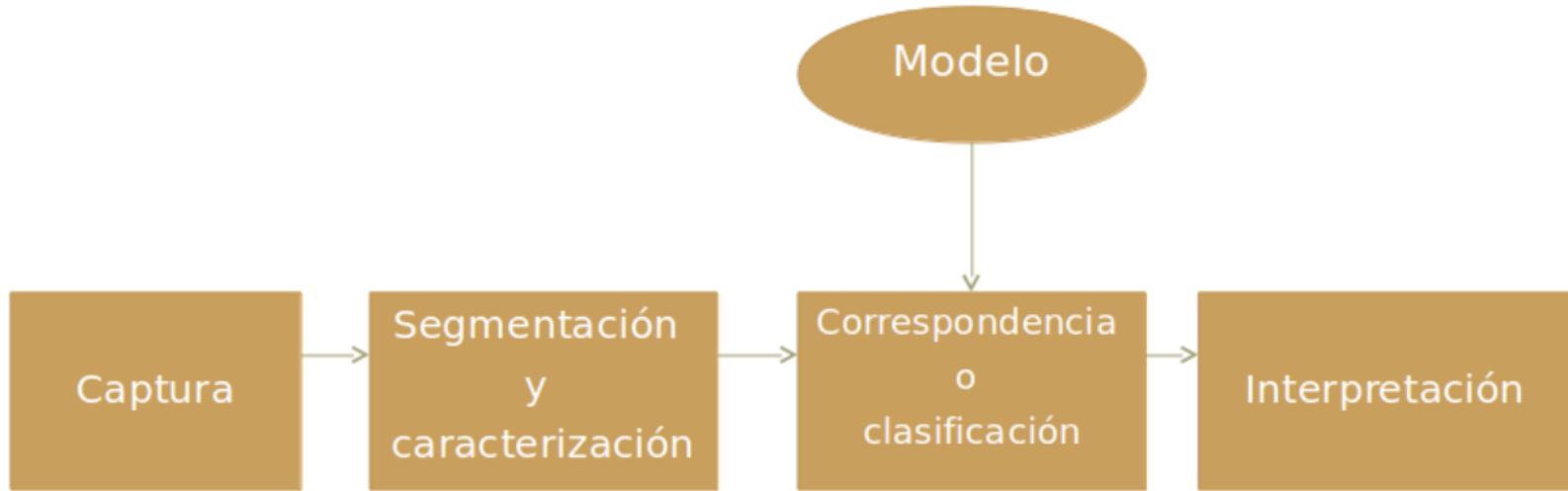
Tareas que se pueden hacer con visión computacional (con aplicaciones a la robótica):

- ▶ OCR (Optical Character Recognition)
- ▶ Detección e identificación de rostros
- ▶ Reconocimiento de objetos
- ▶ Percepción para vehículos sin conductor
- ▶ Reconocimiento de gestos

Otras aplicaciones:

- ▶ Vigilancia
- ▶ Imagenología médica
- ▶ Consultas a bases de datos de imágenes.
- ▶ Percepción remota

Esquema de Visión



Dificultades

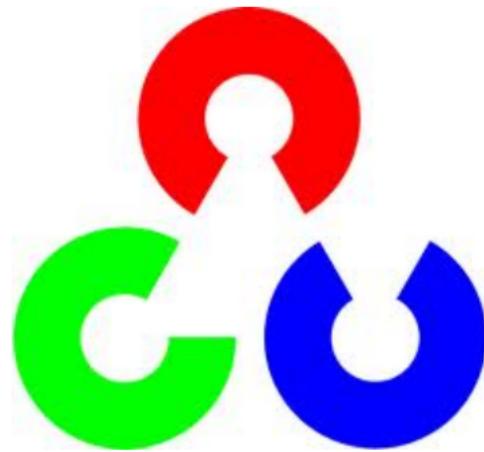
El entorno real tiene una gran cantidad de variaciones en las imágenes de entrada.



1. Iluminación
2. Orientación
3. Oclusión
4. Escala
5. Ruido
6. Desenfoque

La computación con imágenes tiene mas de 30 años, sin embargo, en los últimos años, se ha incrementado considerablemente su desarrollo debido a:

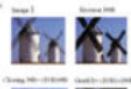
1. Decremento en los precios
2. Memoria con gran capacidad
3. Procesadores de propósito general de alta velocidad.
4. Existen scanners o camaras digitales que pueden ser utilizados para procesar imágenes propias.
5. Existen bibliotecas de software que contienen subrutinas de procesamiento de imágenes (opencv).



OpenCV es una biblioteca libre de visión artificial, originalmente desarrollada por Intel.
Programación en código Python, C y C++
Existen versiones para GNU/Linux, Mac OS X y Windows

OpenCV Overview: > 500 functions

opencv.willowgarage.com



General Image Processing Functions



Segmentation



Transforms



Machine Learning: • Detection, • Recognition



Geometric descriptors



Features



Tracking



Matrix Math

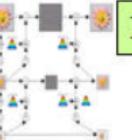
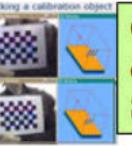
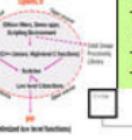


Image Pyramids



Camera calibration, Stereo, 3D



Utilities and Data Structures



Fitting



Robot support

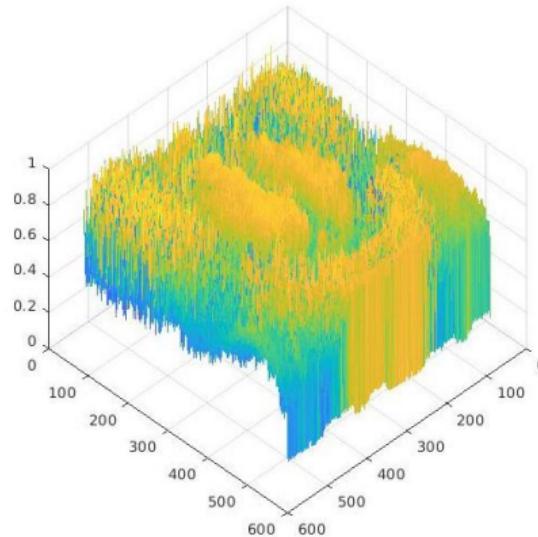
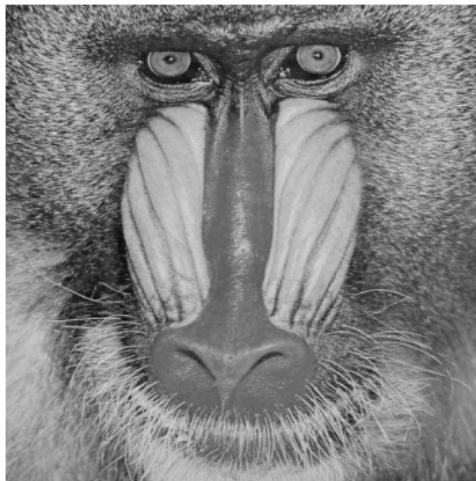
Tarea 11 - La biblioteca OpenCV

Ver instrucciones en Classroom

Deadline: 2024-02-20 al inicio de la clase.

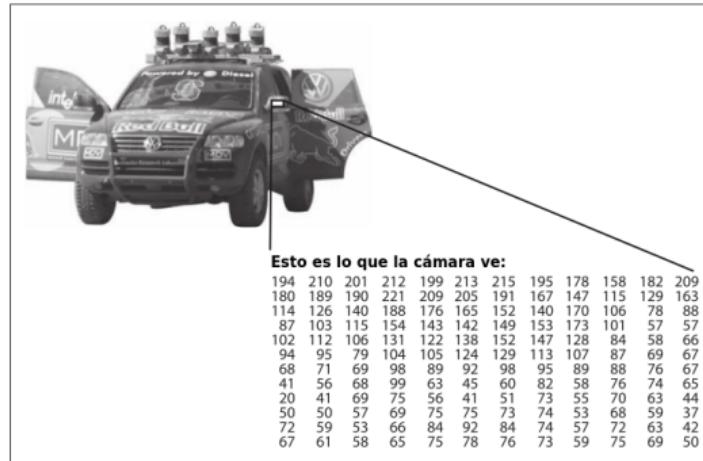
Imágenes como funciones

- ▶ Una imagen (en escala de grises) es una función $I(x, y)$ donde x, y son variables discretas en coordenadas de imagen y la función I es intensidad luminosa.
- ▶ Las imágenes también pueden considerarse como arreglos bidimensionales de números entre un mínimo y un máximo (usualmente 0-255).
- ▶ Las imágenes de color son funciones vectoriales $f : \mathbb{R}^2 \rightarrow \mathbb{R}^3$ donde cada componente de la función se llama canal.



Las imágenes como funciones

Aunque formalmente una imagen es un mapeo $f : \mathbb{R}^2 \rightarrow \mathbb{R}$, en la práctica, tanto x, y como I son variables discretas con valores entre un mínimo y un máximo.



Operaciones básicas

- ▶ Desfase
- ▶ Escalamiento
- ▶ Inversión en x, y
- ▶ Suma y Resta
- ▶ Multiplicación

Tipos de ruido

El ruido es una señal aleatoria $\eta(x, y)$, es decir, no sabemos cuánto vale para un punto determinado (x, y) pero sí podemos caracterizarla.

$$I_n(x, y) = I(x, y) + \eta(x, y)$$

Existen varios tipos de ruido:

- ▶ Sal y pimienta: aleatoriamente aparecen puntos ya sea blancos o negros
- ▶ Ruido de impulso: aleatoriamente aparecen puntos blancos
- ▶ Ruido gausiano: $\eta(x, y)$ se distribuye

Ver ejercicios en Python

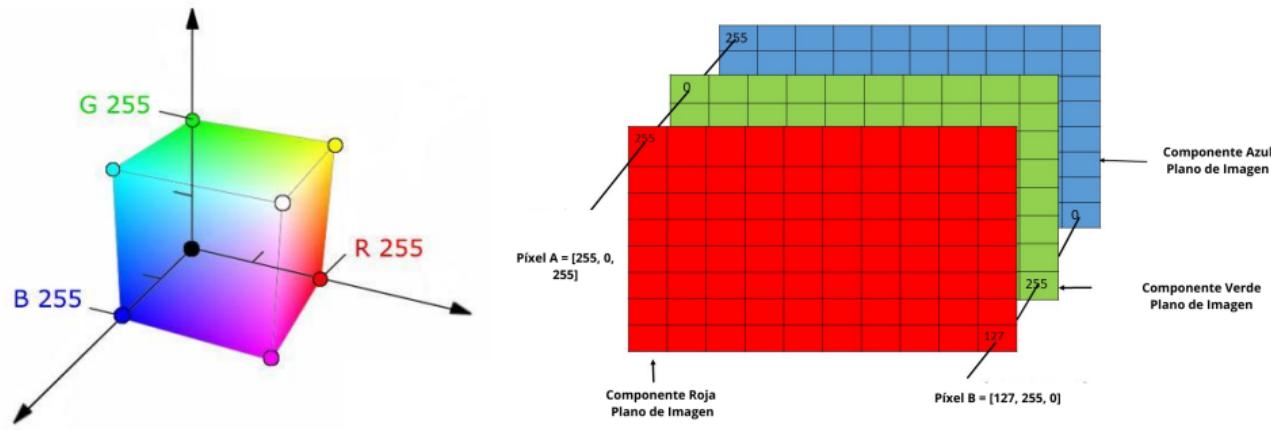
Espacios de color

Un espacio de color o modelo de color es una representación del color mediante un conjunto numérico de valores, generalmente tres valores. Existen varios espacios de color:

- ▶ Aditivos:
 - ▶ RGB
 - ▶ HSV
 - ▶ YCrCb
- ▶ Sustractivos
 - ▶ MCYK
- ▶ Lineales:
 - ▶ RGB
 - ▶ CIE XYZ
- ▶ No lineales
 - ▶ HSV
 - ▶ HSI

El espacio RGB

En este espacio cada color se forma mediante la suma de tres colores primarios: rojo, verde y azul.



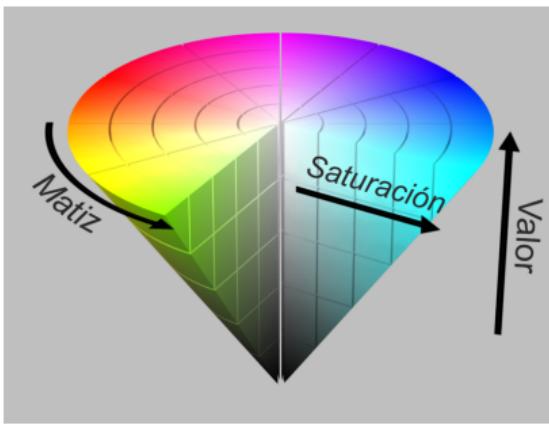
En memoria, las imágenes RGB se suelen representar con arreglos de $H \times W \times 3$ donde H y W son el alto y ancho de la imagen respectivamente.

El espacio HSV

Es un espacio de color diseñado para representar el color de una forma más similar a como lo percibe el ojo humano. El color se representa con tres valores:

- ▶ **Hue:** (Matiz) Es el atributo del color que hace que un estímulo se perciba como similar a alguno de los colores que el ojo puede percibir: rojo, amarillo, verde, azul, violeta, o una combinación de ellos.
- ▶ **Saturation:** (Saturación) Es el atributo que indica qué tan colorido es un estímulo con respecto a su propio brillo.
- ▶ **Value:** (Valor) Atributo que indica qué tanta luz emite un estímulo.

Para obtenerlo a partir de RGB:



$$M = \max(R, G, B) \quad m = \min(R, G, B) \quad C = M - m$$

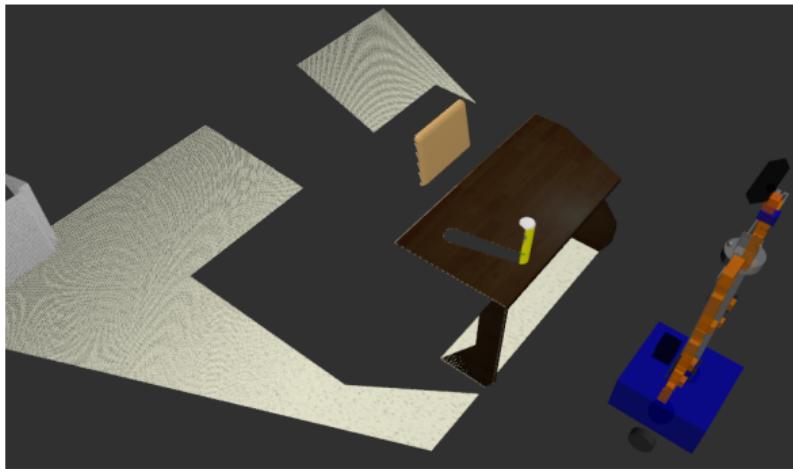
$$H = \begin{cases} \text{Indeterminado} & \text{si } C=0 \\ \frac{G-B}{C} \times 60 & \text{si } M = R \\ \frac{B-R}{C} \times 60 & \text{si } M = G \\ \frac{R-G}{C} \times 60 & \text{si } M = B \end{cases}$$

$$S = \begin{cases} 0 & \text{si } V = 0 \\ \frac{C}{V} & \text{en otro caso} \end{cases}$$

$$V = M$$

Nubes de puntos

Las nubes de puntos son conjuntos de vectores que representan puntos en el espacio. Estos vectores generalmente tienen información de posición (x, y, z). También pueden contener información de color (x, y, z, r, g, b).



Son útiles para determinar la posición en el espacio de los objetos reconocidos.

- ▶ OpenCV es un conjunto de bibliotecas que facilita la implementación de algoritmos de visión computacional.
- ▶ Se puede usar con diversos lenguajes: C++, Python, Java.
- ▶ En Python utiliza la biblioteca Numpy.
- ▶ Las imágenes se representan como matrices donde cada elemento puede ser un solo valor, o bien tres valores, dependiendo de si la imagen está en escala de grises o a color.
- ▶ La configuración más común es que cada pixel esté representado por tres bytes.
- ▶ Las nubes de puntos se representan también como matrices donde cada elemento es una terna de flotantes con la posición (x, y, z).

Segmentación por color

La segmentación de una imagen se refiere a obtener regiones significativas con ciertas características. En este caso, la característica es que estén en un cierto intervalo de color. Los pasos generales para esto son:

1. Transformación de la imagen del espacio BGR al HSV (función `cvtColor`)
2. Obtención de aquellos pixeles que están en un rango de color (función `inRange`)
3. Obtención del centroide de la región (funciones `findNonZero` y `mean`)
4. Si se dispone de una nube de puntos, se puede obtener la posición (x, y, z) del centroide de la región segmentada.

Tarea 07 - Segmentación por color

1. En el archivo `catkin_ws/src/students/scripts/assignment07.py`, en la función `segment_by_color`, realice lo siguiente:
 - 1.1 Defina dos límites superiores y dos límites inferiores, en el espacio de color HSV, para segmentar las latas que se encuentran sobre el escritorio simulado.
 - 1.2 Transforme la imagen del espacio BGR al espacio HSV mediante la función `cvtColor` de OpenCV.
 - 1.3 Determine los pixeles de la imagen que pertenecen al rango de color elegido mediante la función `inRange` de OpenCV.
 - 1.4 Encuentre los índices de los pixeles que pertenecen al rango de color con la función `findNonZero` de OpenCV.
 - 1.5 Utilizando los índices anteriores y la nube de puntos, determine el centroide del objeto con el color seleccionado.
2. Ejecute la simulación con el comando `roslaunch bring_up color_segmentation.launch`
3. Ejecute la práctica con el comando `rosrun students practice06.py`
4. En la pestaña *Simple Tasks* de la GUI, en el campo *Find Object* teclee `pringles` o `drink`
5. En el visualizador debe dibujarse una esfera morada sobre el centro del objeto seleccionado.

Tarea 07 - Segmentación por color

Entregables:

- ▶ Código modificado en la rama correspondiente del repositorio en línea.

Deadline: 2023-11-23 al inicio de la clase.

Un cuerpo rígido en el espacio puede tener una posición (x, y, z) y una orientación. La orientación se puede representar de varias formas:

- ▶ Mediante ángulos de Euler: roll, pitch y yaw $RPY = (\psi, \theta, \phi)$
- ▶ Mediante cuaterniones
- ▶ Mediante una matriz de rotación $R \in SO(3)$

Los ángulos RPY son rotaciones intrínsecas sobre los ejes X , Y , y Z respectivamente. Se llaman intrínsecas porque son rotaciones que ocurren sobre un sistema de referencia *atado* a un cuerpo rígido. Cualquier orientación se puede obtener mediante la composición de tres rotaciones básicas:

$$R = R_{z,\phi} R_{y,\theta} R_{x,\psi}$$

Es decir, primero una rotación de ϕ radianes sobre el eje Z , seguida de una rotación de θ radianes sobre el eje Y del sistema resultante y una rotación de ψ radianes sobre el eje X del sistema rotado.

Transformaciones Homogéneas

Una Transformación Homogénea es una matriz de la forma

$$T = \begin{bmatrix} & & & d_x \\ & R \in SO(3) & & d_y \\ & & & d_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Puede servir para

- ▶ Representar la posición y orientación de un cuerpo rígido
- ▶ Representar una transformación de coordenadas T_{ab} de un sistema de referencia b a un sistema a

Propiedades:

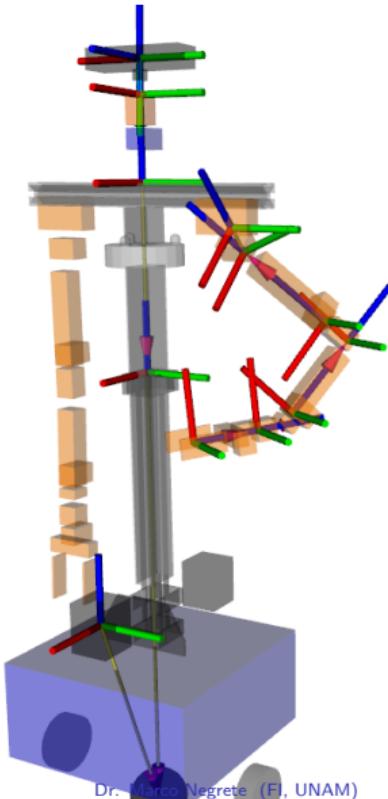
- ▶ Asociatividad: $(T_1 T_2) T_3 = T_1 (T_2 T_3)$
- ▶ Inversa:

$$T^{-1} = \begin{bmatrix} R^T & -R^T d \\ 0 & 1 \end{bmatrix}$$

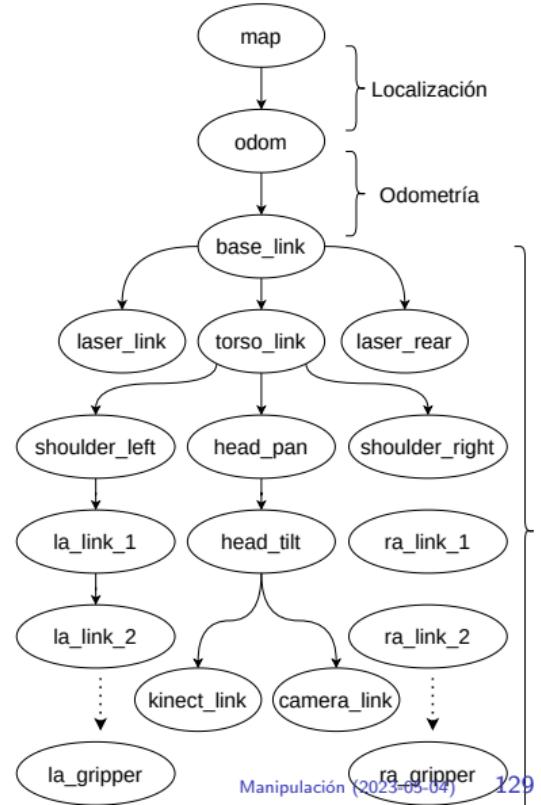
- ▶ Cancelación de índices: $T_{ab} = T_{ac} T_{cb}$

El árbol cinemático

Es útil tener una descripción de la forma en que están conectadas las diferentes articulaciones del robot. Se considera que sobre cada articulación hay un sistema de referencia (*frame*) que está trasladado y rotado con respecto al sistema anterior.



El sistema *absoluto* se suele llamar *map*
El sistema base del robot se suele
llamar *base_link*
Las transformaciones de *map* a
base_link las determina el sistema de
localización
El resto de las transformaciones se
determinan con la posición de cada
articulación
El árbol cinemático se traduce en una
cadena de multiplicaciones de
Transformaciones Homogéneas.



El formato URDF

El formato URDF permite describir el arbol cinemático del robot mediante etiquetas XML:

```
1 <robot>
2   <!-- Define la forma y tamano de la base movil-->
3   <link name="base_link">
4     <visual>
5       <origin xyz="0 0 0.235" rpy="0 0 0"/><material name="blue" />
6       <geometry> <box size="0.42 0.42 0.20" /></geometry>
7     </visual>
8   </link>
9   <!-- Define la forma y tamano del sensor laser-->
10  <link name="laser_link">
11    <visual>
12      <origin xyz="0 0 0" rpy="0 0 0"/><material name="black" />
13      <geometry> <box size="0.08 0.08 0.1" /></geometry>
14    </visual>
15  </link>
16  <!-- Define la posicion del laser con respecto a la base movil-->
17  <joint name="laser_connect" type="fixed">
18    <origin xyz="0.17 0 0.44" rpy="0 0 0"/>
19    <parent link="base_link"/><child link="laser_link" />
20  </joint>
21 </robot>
```

Cada etiqueta `<joint>` representará una Transformación Homogénea.

El formato Xacro

El formato Xacro es un lenguaje de *macros* que permite obtener archivos XML más cortos. Es útil para especificar parámetros físicos en el URDF como inercias y volúmenes:

```
1 <robot name="justina" xmlns:xacro="http://www.ros.org/wiki/xacro">
2   <xacro:property name="width" value="0.42"/>
3   <xacro:property name="depth" value="0.42"/>
4   <xacro:property name="height" value="0.2"/>
5   <xacro:property name="mass" value="30.0"/>
6
7   <link name="base_link">
8     <visual>
9       <origin xyz="0 0 0.235" rpy="0 0 0"/><material name="blue"/>
10      <geometry> <box size="${width} ${depth} ${height}" /></geometry>
11    </visual>
12    <collision>
13      <origin xyz="0 0 0.235" rpy="0 0 0"/>
14      <geometry> <box size="${width} ${depth} ${height}" /></geometry>
15    </collision>
16    <inertial>
17      <origin xyz="0 0 0.235" rpy="0 0 0"/><mass value="50.00"/>
18      <xacro:box_inertia m="${mass}" x="${depth}" y="${width}" z="${height}" />
19    </inertial>
20  </link>
21</robot>
```

La cinemática directa

La cinemática directa consiste en determinar la posición y orientación del efecto final del manipulador a partir de la posición de cada articulación. Esta se puede calcular con la ecuación:

$$P_1 = T_{12} T_{23} T_{34} T_{45} T_{56} T_{67} T_{7g} P_g$$

donde $P_g = [0, 0, 0, 1]^T$ es la posición del gripper con respecto al sistema del gripper, P_1 es la posición del gripper con respecto al sistema base y T_{ab} es la transformación homogénea que define la rotación y traslación producida por cada articulación. Las matrices T_{ab} tienen la forma:

$$T_{ab} = \begin{bmatrix} R_{ab} \in SO(3) & dx_{ab} \\ 0 & dy_{ab} \\ 0 & dz_{ab} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Donde R_{ab} representa la rotación del sistema b respecto al sistema a y $(dx_{ab}, dy_{ab}, dz_{ab})$ es la traslación del sistema b respecto al sistema a .

La rotación R_{ab} está definida en el URDF por el atributo “rpy” de la sub etiqueta origin de la etiqueta joint y por la posición de la articulación. La traslación $(dx_{ab}, dy_{ab}, dz_{ab})$ está definida por el atributo “xyz”.

La cinemática inversa

La cinemática inversa consiste en determinar las posiciones que debe tener cada articulación para que el efecto final tenga la posición y orientación deseadas.

- ▶ Mientras la cinemática directa siempre tiene solución, la cinemática inversa, no.
- ▶ Se puede resolver por métodos geométricos para obtener una solución cerrada, aunque el análisis puede ser muy complicado.
- ▶ Una solución más general se puede obtener mediante un método numérico.

Suponiendo que se tiene una configuración deseada $p_d \in \mathbb{R}^6$ ($xyz - RPY$), se desea encontrar el conjunto de posiciones articulares $q \in \mathbb{R}^7$ que satisfagan la ecuación

$$FK(q) - p_d = 0$$

donde la función FK representa la cinemática directa.

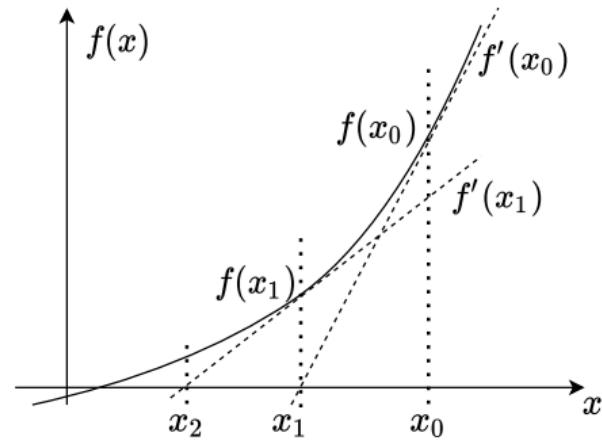
El método Newton-Raphson

El método numérico de Newton-Raphson sirve para encontrar raíces, es decir, para resolver ecuaciones de la forma

$$f(x) = 0$$

El algoritmo es el siguiente:

```
xi ← x0
while |f(x)| > ε do
    |   xi+1 ← xi -  $\frac{f(x_i)}{f'(x_i)}$ 
end
```



La función cuyas raíces se desea encontrar es:

$$f(q) = FK(q) - p_d = 0$$

Por lo que, sustituyendo $f(x)$ el algoritmo quedaría:

```
qi ← q0
while |f(q)| > ε do
    qi+1 ← qi -  $\frac{FK(q_i) - p_d}{\frac{d}{dq}(FK(q) - p_d)}$ 
end
```

- ▶ Sin embargo, dado que $f(q) : \mathbb{R}^7 \rightarrow \mathbb{R}^6$, no podemos hablar de derivada sino que debemos usar un Jacobiano $J(q) \in \mathbb{R}^{6 \times 7}$.
- ▶ Newton-Raphson incluye una división de la función entre su derivada, pero no existe la división entre matrices, por lo que debemos multiplicar por la inversa del Jacobiano
- ▶ Pero el Jacobiano $J(q)$ no es una matriz cuadrada por lo que **no tiene inversa**

La matriz pseudoinversa

Para una matriz $A \in \mathbb{R}^{m \times n}$, la matriz pseudoinversa A^\dagger se puede calcular a partir de la descomposición en valores singulares:

$$A = U \begin{pmatrix} S & 0 \\ 0 & 0 \end{pmatrix} V^T$$

donde U y V son ambas matrices ortogonales y S está es una matriz diagonal formada por los valores singulares (positivos) de A . Con esta descomposición, la pseudoinversa se puede calcular como:

$$A^\dagger = V \begin{pmatrix} S^{-1} & 0 \\ 0 & 0 \end{pmatrix} U^T$$

La matriz pseudoinversa

La pseudoinversa tiene varias propiedades:

- ▶ $AA^\dagger A = A$ y $A^\dagger AA^\dagger = A^\dagger$
- ▶ AA^\dagger es simétrica y también $A^\dagger A$
- ▶ Si A tiene elementos reales, también A^\dagger
- ▶ Si A es de rango completo por columnas, es decir $\text{rank}(A) = n \leq m$, esto es, $A^T A$ es no singular, entonces:

$$A^\dagger = (A^T A)^{-1} A^T \quad A^\dagger A = I_n$$

- ▶ Si A es de rango completo por renglones, es decir $\text{rank}(A) = m \leq n$, esto es, AA^T es no singular, entonces:

$$A^\dagger = A^T (AA^T)^{-1} \quad AA^\dagger = I_m$$

- ▶ La solución al problema de mínimos cuadrados:

$$\min_x \|Ax - y\|_2$$

está dada por

$$x = A^\dagger y$$

El Jacobiano de un manipulador

El Jacobiano es una matriz que relaciona la velocidad articular \dot{q} con la velocidad en el espacio cartesiano $[v \omega]^T$ (velocidad lineal y angular):

$$\dot{p} = \begin{bmatrix} v \\ \omega \end{bmatrix} = J\dot{q} \quad p = [x, y, z, \phi, \theta, \psi] \in \mathbb{R}^6, \quad J \in \mathbb{R}^{6 \times 7}, \quad q \in \mathbb{R}^7$$

con (ϕ, θ, ψ) , los ángulos Roll, Pitch y Yaw respectivamente

$$J = \begin{bmatrix} \frac{\partial x}{\partial q_1} & \dots & \frac{\partial x}{\partial q_7} \\ \frac{\partial y}{\partial q_1} & \dots & \frac{\partial y}{\partial q_7} \\ \frac{\partial z}{\partial q_1} & \dots & \frac{\partial z}{\partial q_7} \\ \vdots & \ddots & \vdots \\ \frac{\partial \phi}{\partial q_1} & \dots & \frac{\partial \phi}{\partial q_7} \\ \frac{\partial \theta}{\partial q_1} & \dots & \frac{\partial \theta}{\partial q_7} \\ \frac{\partial \psi}{\partial q_1} & \dots & \frac{\partial \psi}{\partial q_7} \end{bmatrix}$$

- ▶ La matriz J se puede obtener analíticamente, sin embargo, dado el número de grados de libertad, resulta muy complicado
- ▶ Se puede obtener approximando las derivadas parciales con diferencias finitas

El Jacobiano de un manipulador

De la definición de derivada parcial:

$$\frac{\partial f}{\partial q_i} = \lim_{\Delta q \rightarrow 0} \frac{f(q_1, q_2, \dots, q_i, \dots, q_7) - f(q_1, q_2, \dots, q_i - \Delta q, \dots, q_7)}{\Delta q}$$

Podemos aproximar la deriva como:

$$\frac{\partial f}{\partial q_i} \approx \frac{f(q_1, q_2, \dots, q_i, \dots, q_7) - f(q_1, q_2, \dots, q_i - \Delta q, \dots, q_7)}{\Delta q}$$

O bien

$$\frac{\partial f}{\partial q_i} \approx \frac{f(q_1, q_2, \dots, q_i + \Delta q, \dots, q_7) - f(q_1, q_2, \dots, q_i, \dots, q_7)}{\Delta q}$$

Promediando ambas aproximaciones:

$$\frac{\partial f}{\partial q_i} \approx \frac{f(q_1, q_2, \dots, q_i + \Delta q, \dots, q_7) - f(q_1, q_2, \dots, q_i - \Delta q, \dots, q_7)}{2\Delta q}$$

El Jacobiano del manipulador

Utilizando la aproximación de la derivada parcial, el Jacobiano del manipulador se puede calcular como:

$$J = \begin{bmatrix} \frac{FK(q_+^1) - FK(q_-^1)}{2\Delta q} & \dots & \frac{FK(q_+^7) - FK(q_-^7)}{2\Delta q} \end{bmatrix}$$

donde $FK(q) \in \mathbb{R}^6$ es la cinemática directa del manipulador y

$$\begin{aligned} q_+^i &= [q_1, \dots, q_i + \Delta q, \dots, q_7] \\ q_-^i &= [q_1, \dots, q_i - \Delta q, \dots, q_7] \end{aligned}$$

con Δq , un valor lo suficientemente pequeño para una buena aproximación de la derivada.

Cinemática Inversa por Newton-Raphson

Sintetizando, aplicamos Newton-Raphson para resolver la ecuación:

$$FK(q) - p_d = 0$$

Se tiene:

```
q_k ← q_0 //Una estimación inicial que puede ser la posición articular actual
p ← FK(q_k) //La posición cartesiana que tendría el gripper con la estimación inicial
while |p - p_d| > ε do
    J ← Jacobiano(q_k)
    q_{k+1} ← q_k - J†(p - p_d)
    p ← FK(q_{k+1})
end
```

- ▶ Es importante que las variables angulares siempre estén en el intervalo $(-\pi, \pi]$:
 - ▶ Las posiciones articulares
 - ▶ Los ángulos roll, pitch, yaw
 - ▶ Las componentes angulares del error $p - p_d$

Tarea 08 - Cinemática Inversa

1. En el archivo `catkin_ws/src/students/scripts/assignment08.py`, en la función `forward_kinematics`, implemente la el cálculo de la cinemática directa. Siga las instrucciones dadas en los comentarios.
2. En la función, `jacobian`, calcule el jacobiano del manipulador para la configuración q utilizando la aproximación por diferencias finitas. Siga las instrucciones de los comentarios del código.
3. En la función `inverse_kinematics_xyzrpy`, implemente el cálculo de la cinemática inversa por el método de Newton-Raphson. Siga las instrucciones de los comentarios del código.
4. Ejecute la simulación con el comando `roslaunch surge_et_ambula justina_gazebo.launch`
5. Ejecute la tarea con el comando `rosrun students assingment08.py`

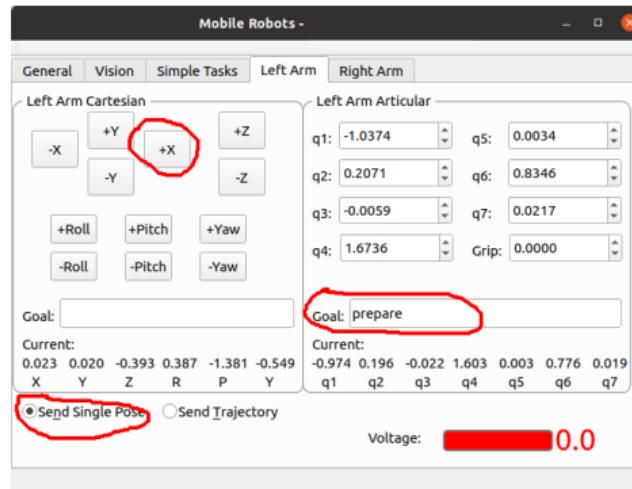
Tarea 08 - Cinemática Inversa

6. En las pestañas *Left Arm* o *Right Arm* de la GUI realice lo siguiente:

6.1 Seleccione la opción *Send Single Pose*

6.2 Escriba *prepare* en el campo *Goal* del grupo *Arm Articular* y presione enter

6.3 Presione el botón *+X* y verifique que el brazo se mueva unos 10 cm hacia adelante



Entregables:

- Código modificado en la rama correspondiente del repositorio en línea

Deadline: 2023-11-28 al inicio de la clase.

Modelo dinámico de un manipulador

El modelo dinámico de un manipulador está dado por:

$$M(q)\ddot{q} + C(q, \dot{q})\dot{q} + B\dot{q} + G(q) = \tau$$

donde

- ▶ $M(q)$ es la matriz de inercias del manipulador
- ▶ $C(q, \dot{q})$ es la matriz de fuerzas de Coriolis
- ▶ B es la matriz de coeficientes de fricción
- ▶ $G(q)$ es el vector de pares debidos a la gravedad
- ▶ τ es el vector de pares de entrada

El control PID

El control Proporcional-Integral-Derivativo es un tipo de control lineal en lazo cerrado que calcula la acción de control mediante una combinación lineal del error, la integral del error y la derivada del error.

- ▶ Para el manipulador, el ángulo deseado q_d está dado por el resultado de la cinemática inversa.
- ▶ La posición angular q se obtiene de los motores o del simulador.
- ▶ La salida del controlador es el torque τ que se envía a los motores.

En la versión continua:

$$\tau(t) = K_p e(t) + K_I \int e(t) dt + K_d \dot{e}(t)$$

con $e = q_d - q$ En la versión discreta:

$$\tau_i = K_p e_i + K_I \sum_{j=0}^i e_j + K_d \frac{e_i - e_{i-1}}{\Delta t}$$

con Δt , el periodo de muestreo.

Aunque la interacción de las tres señales (error, integral del error y derivada del error) es compleja y depende mucho del sistema, de manera intuitiva se pueden indicar las siguientes funciones de cada componente:

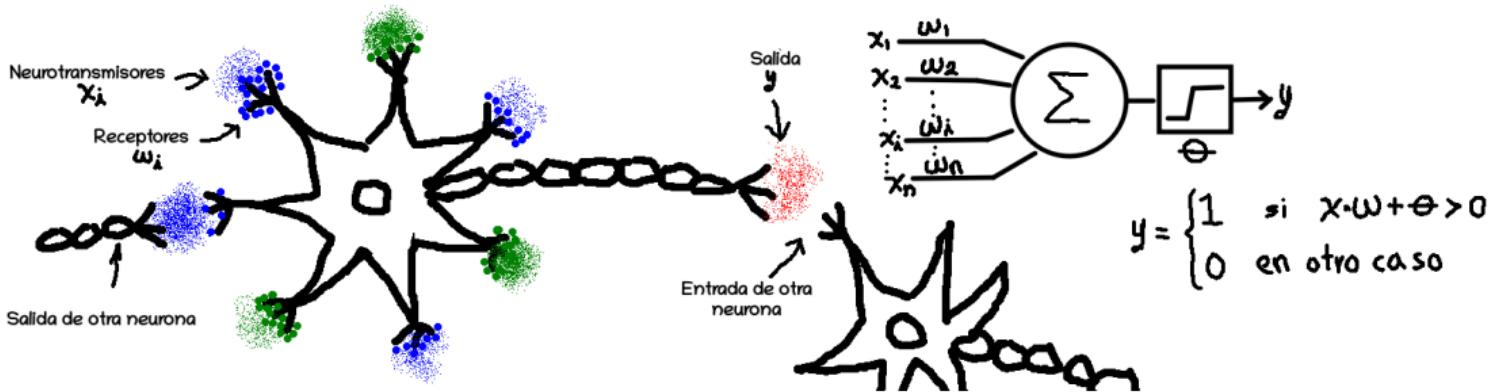
- ▶ **Proporcional:** Aumenta o disminuye el tiempo de asentamiento.
- ▶ **Integral:** Reduce el error en estado estable, aunque puede producir inestabilidad.
- ▶ **Derivativa:** Funciona como amortiguamiento y ayuda a disminuir el sobreceso.

Los *stacks* ros_control y ros_controllers

Son un conjunto de paquetes que implementan controladores PID y varias interfaces de hardware.

- ▶ El stack `ros_control` implementa varias interfaces de hardware. En este curso, la interfaz usada es la que interactúa con la simulación de Gazebo. De este stack, el paquete `controller_manager` es importante porque utiliza un archivo `yaml` para lanzar otros nodos que implementan controladores PID.
- ▶ El stack `ros_controllers` implementa varios algoritmos de control para diferentes tipos de actuadores.

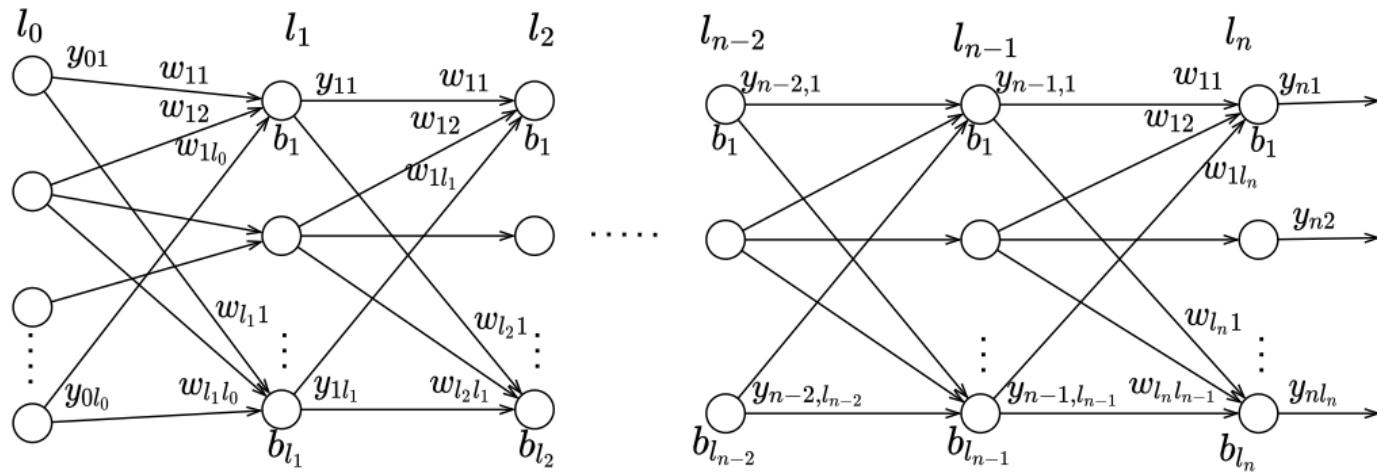
Modelo de una neurona



- ▶ Una neurona tiene una diferencia de potencial entre el interior y exterior de la membrana de aprox -70 mV
- ▶ Los neurotransmisores abren canales para dejar pasar iones positivos o negativos por lo que hacen que el voltaje aumente o disminuya
- ▶ La efectividad del neurotransmisor depende de la cantidad de receptores en la membrana
- ▶ La cantidad de neurotransmisor se puede considerar como una señal de entrada x_i que se pondra por la cantidad de receptores w_i
- ▶ Si el efecto de todos los neurotransmisores hace que el voltaje supere los -50 mV aprox, se produce un pulso eléctrico que hace que la neurona libere neurotransmisor

Backpropagation

La red neuronal está definida por la cantidad de neuronas en cada capa $[l_0, l_1, \dots, l_{n-1}, l_n]$:



$$W_1 \in R^{l_1 \times l_0}$$

$$W_2 \in R^{l_2 \times l_1}$$

$$B_1 \in R^{l_1}$$

$$B_2 \in R^{l_2}$$

$$W_n \in R^{l_{n-1} \times l_{n-2}}$$

$$W_n \in R^{l_n \times l_{n-1}}$$

$$B_n \in R^{l_{n-1}}$$

$$B_n \in R^{l_n}$$

El conjunto de pesos w se puede agrupar en un conjunto de n matrices $W = [W_1, W_2, \dots, W_n]$ con los órdenes indicados en la figura. El conjunto de biases se puede agrupar en n vectores $B = [B_1, B_2, \dots, B_n]$.

Backpropagation

Para la capa salida, el gradiente con respecto a cada uno de los pesos $w \in W_n \in \mathbb{R}^{I_n \times I_{n-1}}$ es también una matriz $\nabla y_n \in \mathbb{R}^{I_n \times I_{n-1}}$:

$$\begin{bmatrix} (y_{n1} - t_1)(y_{n1} - y_{n1}^2)y_{n-1,1} & (y_{n1} - t_1)(y_{n1} - y_{n1}^2)y_{n-1,2} & \dots & (y_{n1} - t_1)(y_{n1} - y_{n1}^2)y_{n-1,I_{n-1}} \\ (y_{n2} - t_2)(y_{n2} - y_{n2}^2)y_{n-1,1} & (y_{n2} - t_2)(y_{n2} - y_{n2}^2)y_{n-1,2} & \dots & (y_{n2} - t_2)(y_{n2} - y_{n2}^2)y_{n-1,I_{n-1}} \end{bmatrix}$$

Tarea 09 - Redes neuronales

Síntesis de voz con SoundPlay

- ▶ Es un paquete que permite reproducir archivos .wav o .ogg, sonidos predeterminados y síntesis de voz.
- ▶ La síntesis de voz se hace utilizando Festival (<http://www.cstr.ed.ac.uk/projects/festival/>).
- ▶ Para sintetizar voz, basta con correr el nodo soundplay_node y publicar un mensaje de tipo sound_play/SoundRequest con lo siguiente:
 - ▶ msg_speech.sound = -3
 - ▶ msg_speech.command = 1
 - ▶ msg_speech.volume = 1.0
 - ▶ msg_speech.arg2 = "voz a utilizar"
 - ▶ msg_speech.arg = "texto a sintetizar"

Tarea 10 - Síntesis de voz

Ejecute el comando:

```
1      roslaunch surge-et_ambula speech_synthesis.launch  
2
```

En otra terminal, ejecute el comando:

```
1      rosrun students assignment10.py "my first synthesized voice"  
2
```

Para instalar más voces:

- ▶ Ejecute el comando `sudo apt-get install festvox-<voz deseada>`
- ▶ Para ver qué voces se tienen instaladas: `ls /usr/share/festival/voices/english/`

Tarea 10 - Síntesis de voz

Modifique el archivo `catkin_ws/src/students/scripts/assignment10.py` y cambie la voz a utilizar en el mensaje `SoundRequest`.

```
17 msg_speech = SoundRequest()
18 msg_speech.sound    = -3
19 msg_speech.command = 1
20 msg_speech.volume  = 1.0
21 #
22 # EJERCICIO
23 # Cambie la voz por alguna de las voces instaladas
24 #
25 msg_speech.arg2     = "voice_kal_diphone"
26 msg_speech.arg      = text_to_say
```

El nombre de la voz se compone de `voice_` más el nombre que aparece en la carpeta `/usr/share/festival/voices/english/`.

Reconocimiento de voz con Pocketsphinx

Pocketsphinx es un *toolkit* open source desarrollado por la Universidad de Carnegie Mellon (<https://cmusphinx.github.io/>).

- ▶ Aunque el toolbox original no está hecho específicamente para ROS, ya existen varios repositorios con nodos ya implementados que integran ROS y Pocketsphinx:
 - ▶ <https://github.com/mikeferguson/pocketsphinx>
 - ▶ <https://github.com/Pankaj-Baranwal/pocketsphinx>
- ▶ El usuario debe estar agregado al grupo *audio* para el correcto funcionamiento: `sudo usermod -a -G audio <user_name>`
- ▶ Se puede hacer reconocimiento usando una lista de palabras, un modelo de lenguaje o una gramática.
- ▶ Se utilizarán gramáticas y sus correspondientes diccionarios.
- ▶ Para construir diccionarios, visitar <https://cmusphinx.github.io/wiki/tutorialdict/>
- ▶ Para construir gramáticas, visitar <https://www.w3.org/TR/2000/NOTE-jsgf-20000605/>

1. Verifique el volumen del micrófono
2. Inspeccione el archivo `catkin_ws/src/hri/sprec_pocketsphinx/vocab/final_project.gram` para ver las frases que se pueden reconocer de acuerdo con la gramática.
3. Ejecute el comando `rosrun surge_ambula speech_recognition`
4. En otra terminal, ejecute el comando `rostopic echo /hri/sp_rec/recognized`
5. Pruebe el reconocimiento de voz con alguna de las siguientes frases:
 - 5.1 Robot, take the pringles to the table
 - 5.2 Robot, take the drink to the table
 - 5.3 Robot, take the pringles to the kitchen
 - 5.4 Robot, take the drink to the kitchen

Proyecto final: robot de servicio

Haga que el robot lleve un objeto de un lugar a otro



R. Arkin.

Behavior-Based Robotics.

MIT Press, 1998.



H. M. Choset, K. M. Lynch, S. Hutchinson, G. Kantor, W. Burgard, L. Kavraki, S. Thrun, and R. C. Arkin.

Principles of robot motion: theory, algorithms, and implementation.

MIT press, 2005.



J. Latombe.

Robot Motion Planning.

Kluwer Academic, 1991.

Contacto

Dr. Marco Negrete
Profesor Asociado C
Departamento de Procesamiento de Señales
Facultad de Ingeniería, UNAM.

marco.negrete@ingenieria.unam.edu