Robotic Systems I

Lecture 2: Integrators, Mobile Robots and Motion Planning

Konstantinos Chatzilygeroudis - costashatz@upatras.gr

Department of Electrical and Computer Engineering
University of Patras

Template made by Panagiotis Papagiannopoulos

Logistics

Laboratory of Automation & Robotics

- **Lectures**:
  - 10 + 1 Recitation Lecture
  - ⩾8 Lab Exercises
- **Examination**:
  - Lab code **(20% of total grade)**
  - Intermediate exams **(30% of total grade)**
  - For **50% of total grade** you select **one of the two**:
    - Team Project (max 2 per team) — only for the June exams
    - Final exam
  - **IMPORTANT:** All exams are hybrid: multiple choice + code
- **Office Hours**:
  - **Tue-Wed** *(10:00-12:00)*
  - 24/7 by email (`costashatz@upatras.gr`, subject: *ECE_RSI_AM*)
- **Material and Announcements**

What is this course about?

- 3D Rigid Body Motions
- Dynamics and Control of Manipulators
- Mobile Robots, Simulators
- Orientation Representations
- Lie Algebra
- Splines
- Optimization
- Localization, SLAM
- Optimization-based Control of Complex Robots/Whole Body Controllers
- **We focus on implementation/robotics applications. You are required to write code!** Not a theory course!

Lectures/Lab Format

- **3-hour lectures:**
  - A lot of theory! Hints for personal reading!
  - Live code examples + analyses!
  - Ask questions please!
- **2-hour lab exercises:**
  - **20% of total grade**
  - Small code examples that you need to fill
  - You need to deliver code
  - Ask questions, experiment!

Modeling Systems via Differential Equations

Continuous Dynamics:

$$\dot{\boldsymbol{x}} = f(\boldsymbol{x}, \boldsymbol{u})$$

$\boldsymbol{x} \in \mathbb{R}^n$: "State space"

$\boldsymbol{u} \in \mathbb{R}^m$: "Control/Input space"

f : "Dynamics Function"

In robotics we usually have:

$$\boldsymbol{x} = \begin{bmatrix} \boldsymbol{q} \\ \boldsymbol{v} \end{bmatrix}$$

$\boldsymbol{q}$: "Configuration"

$\boldsymbol{v}$: "Velocity"

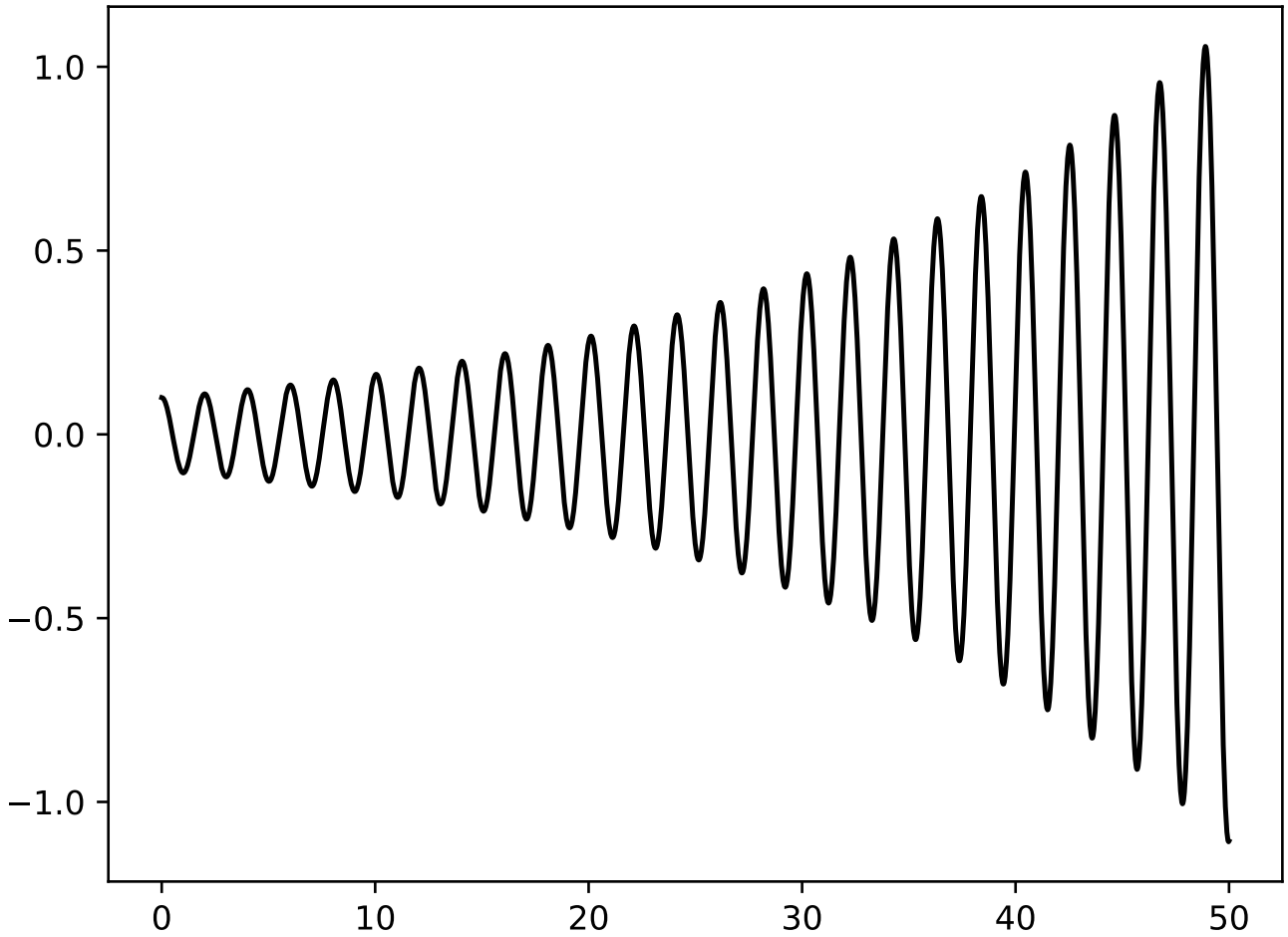
Discretization of Dynamical Systems

- **Why do we care about discretization?**
  - We can't "solve" $\dot{\boldsymbol{x}} = f(\boldsymbol{x}, \boldsymbol{u})$ in the general case,
  - Computers cannot handle continuous t in representing $\boldsymbol{x}(t)$,
  - Sometimes discrete time dynamics can describe better the situation (e.g. contact-based dynamics).
- **How?**

Discretization of Dynamical Systems

- **Why do we care about discretization?**
  - We can't "solve" $\dot{\boldsymbol{x}} = f(\boldsymbol{x}, \boldsymbol{u})$ in the general case,
  - Computers cannot handle continuous t in representing $\boldsymbol{x}(t)$,
  - Sometimes discrete time dynamics can describe better the situation (e.g. contact-based dynamics).
- **How?**
  - $\boldsymbol{x}_{k+1} = f_{\text{discrete}}(\boldsymbol{x}_k, \boldsymbol{u}_k)$
  - **Euler Integration:** $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \underbrace{\underbrace{f(\boldsymbol{x}_k, \boldsymbol{u}_k)}_{\text{"Continuous Dynamics"}} \underbrace{dt}_{\text{"Timestep"}}}_{f_{\text{discrete}}(\boldsymbol{x}_k, \boldsymbol{u}_k)}$

Discretization - Pendulum Code/Simulation



Runge Kutta 4th Order (RK4)

- **Do not use Euler Integration!**
- **What else?**
  - **Runge Kutta 4th Order (RK4):**

$$\boldsymbol{f}_1 = f(\boldsymbol{x}_k, \boldsymbol{u}_k)$$

$$\boldsymbol{f}_2 = f(\boldsymbol{x}_k + \boldsymbol{f}_1 \frac{dt}{2}, \boldsymbol{u}_k)$$

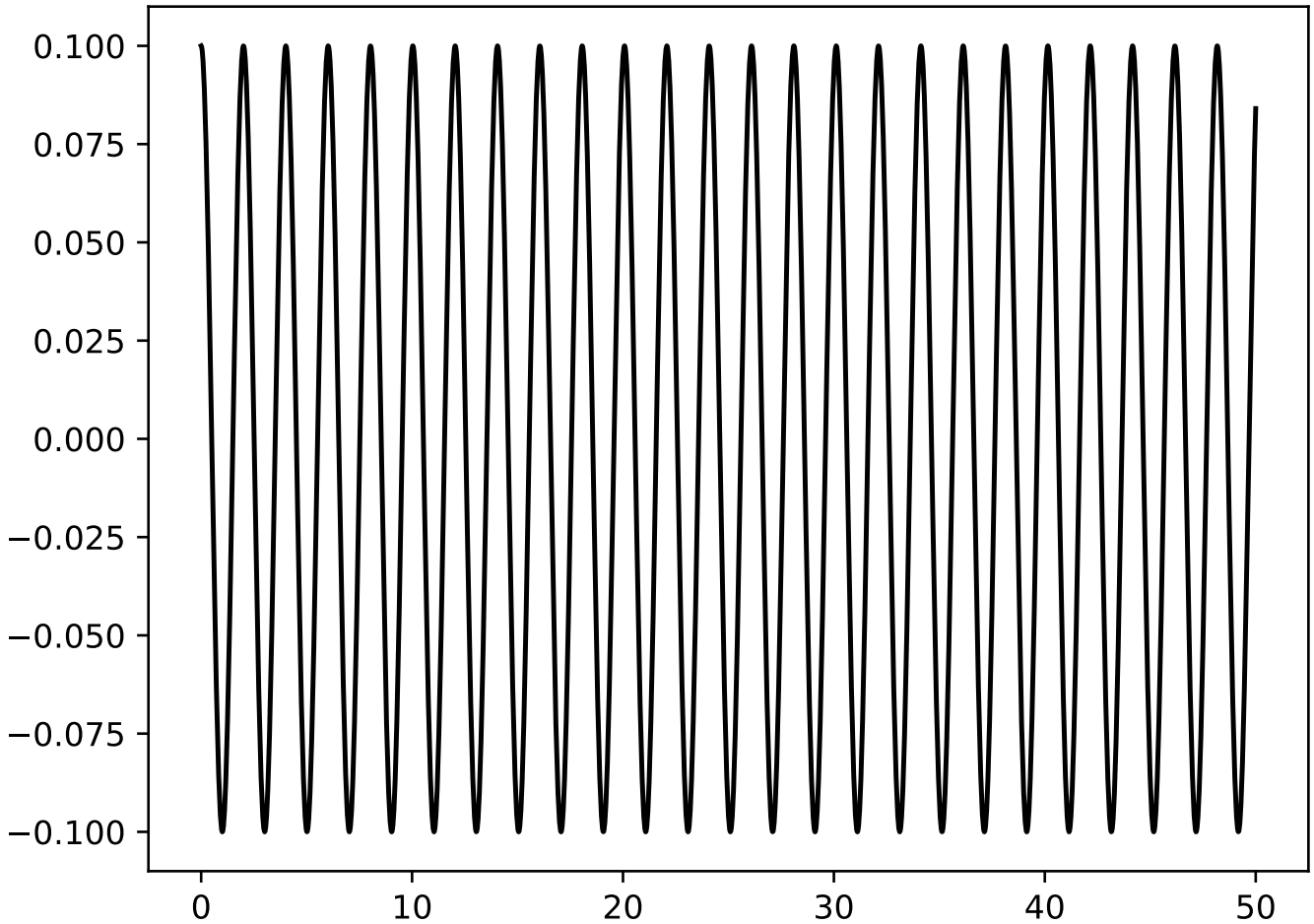
$$\boldsymbol{f}_3 = f(\boldsymbol{x}_k + \boldsymbol{f}_2 \frac{dt}{2}, \boldsymbol{u}_k)$$

$$\boldsymbol{f}_4 = f(\boldsymbol{x}_k + \boldsymbol{f}_3 dt, \boldsymbol{u}_k)$$

$$\boldsymbol{x}_{k+1} = \boldsymbol{x} + \frac{dt}{6}\left(\boldsymbol{f}_1 + 2\boldsymbol{f}_2 + 2\boldsymbol{f}_3 + \boldsymbol{f}_4\right)$$

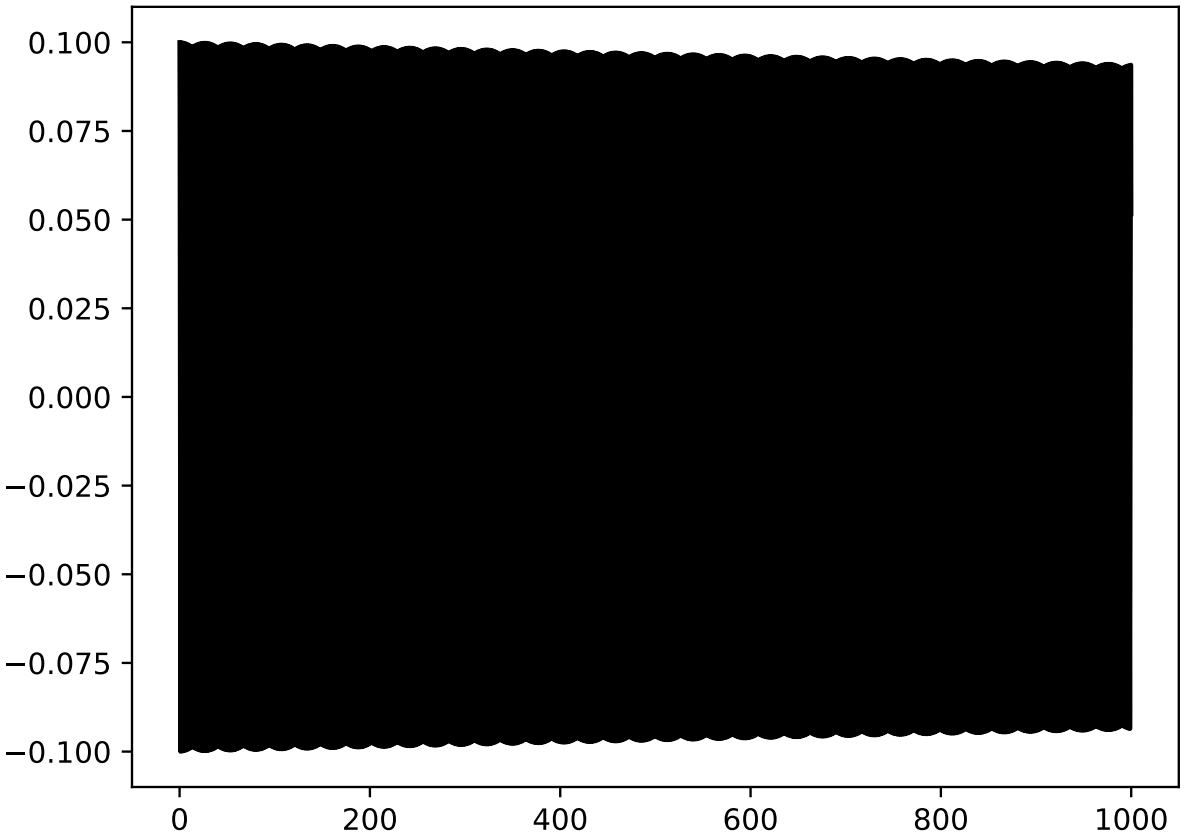
  - RK4 basically fits a cubic polynomial.

RK4 - Pendulum Code/Simulation



RK4 - Pendulum Code/Simulation (2)

The simulation is damped!



Semi-Implicit Euler Integration

- **RK4 is nice, but not perfect!**
- **What else?**
  - **Semi-Implicit Euler Integration:**

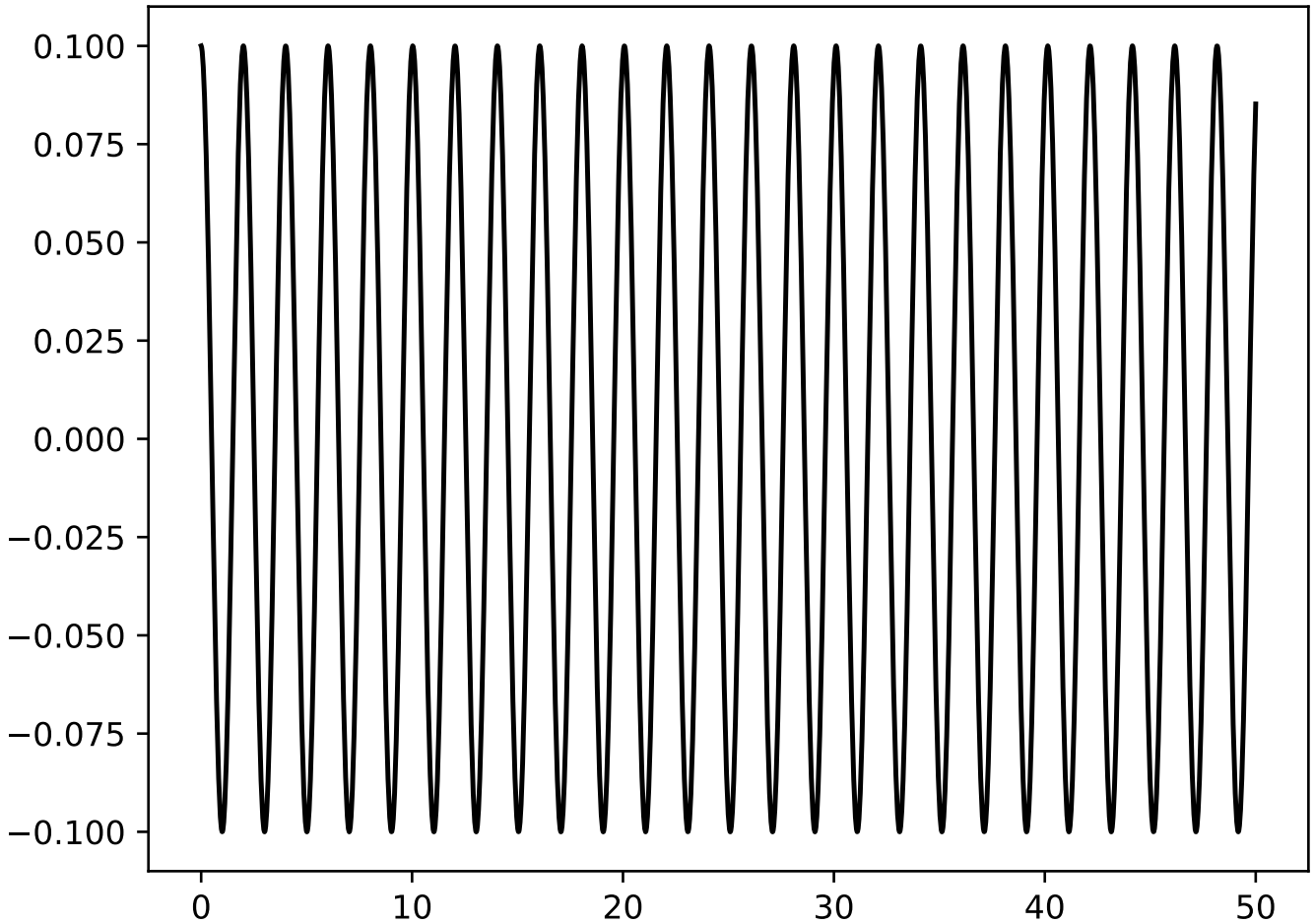
$$\begin{bmatrix} \dot{\boldsymbol{q}}_k \\ \dot{\boldsymbol{v}}_k \end{bmatrix} = f(\boldsymbol{x}_k, \boldsymbol{u}_k)$$

$$\boldsymbol{v}_{k+1} = \boldsymbol{v}_k + \dot{\boldsymbol{v}}_k dt$$

$$\boldsymbol{q}_{k+1} = \boldsymbol{q}_k + \boldsymbol{v}_{k+1} dt$$

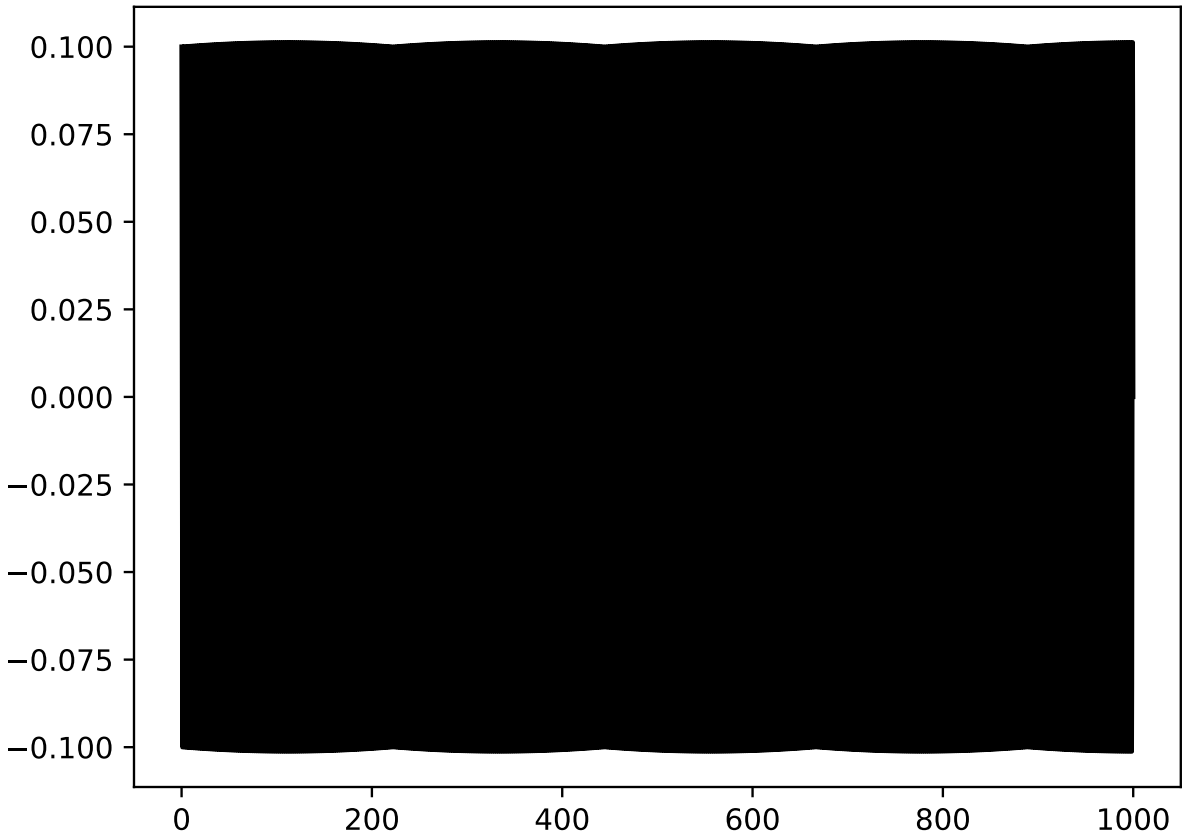
  - We basically update first the velocity of the system, and then use the $k+1$ velocity to update the position part.

Semi-Implicit Euler - Pendulum Code/Simulation



Semi-Implicit Euler - Pendulum Code/Simulation (2)

Undamped simulation! Semi-Implicit Euler preserves energy!



Implicit Integrators

So far we have been discussing explicit integrators. Implicit integrators are defined as:

$$f_{\text{discrete}}(\boldsymbol{x}_{k+1}, \boldsymbol{x}_k, \boldsymbol{u}_k) = \mathbf{0}$$

We solve this as a root-finding problem. We know $\boldsymbol{x}_k, \boldsymbol{u}_k$ and we need to solve for $\boldsymbol{x}_{k+1}$.

Simplest form is "Backward Euler" (uncontrolled system):

$$f_{\text{discrete}}(\boldsymbol{x}_{k+1}, \boldsymbol{x}_k) = \boldsymbol{x}_{k+1} - \boldsymbol{x}_k - f(\boldsymbol{x}_{k+1})dt = \mathbf{0}$$

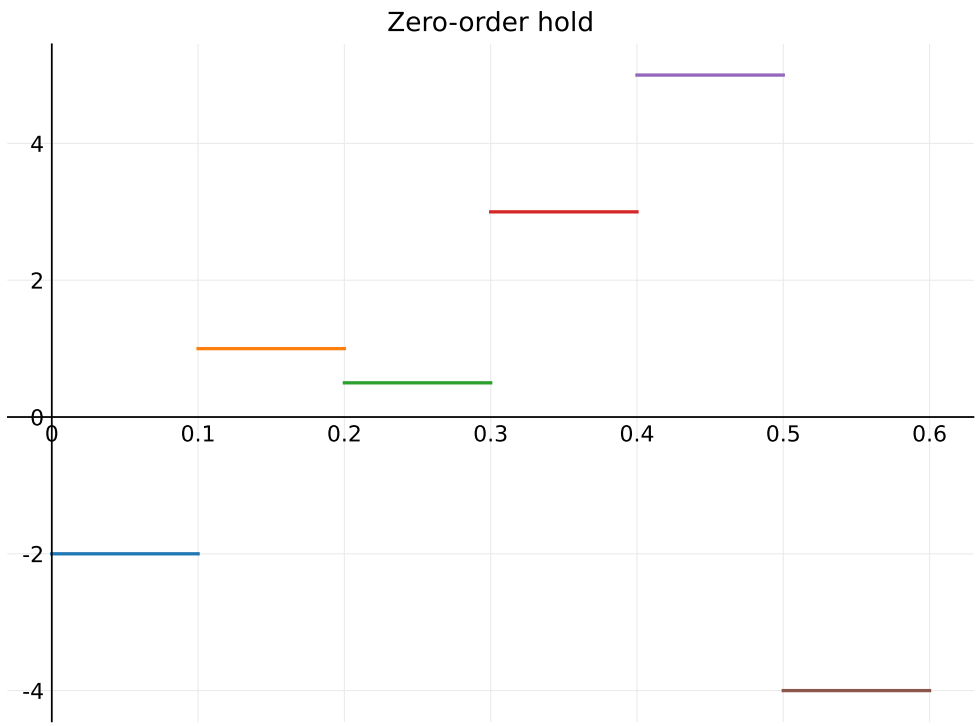
Integrators - Summary

- **Do not use Forward Euler!**
- RK4 is the "industry standard", but comes with damped behavior (unrealistic)
- Implicit methods are more accurate, but require optimization!
- Semi-Implicit Euler is a middle ground (not always good enough!)
- Be careful! Test! Test! Test!

Discretization of Controls

How can we discretize the control inputs?

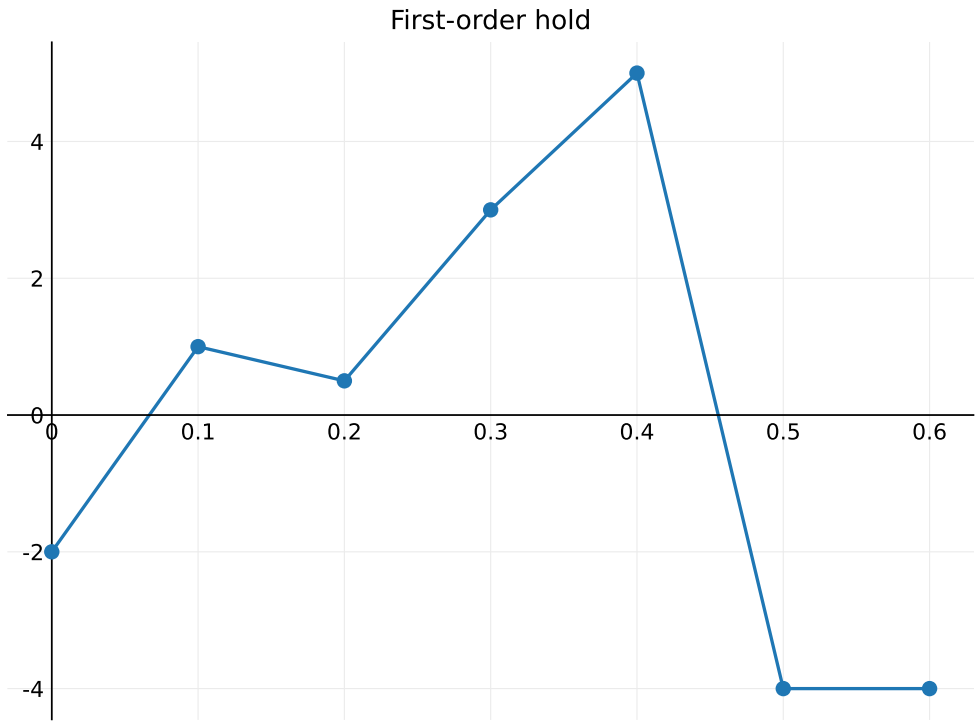
- **"Zero-order hold"**



Discretization of Controls (2)

How can we discretize the control inputs?

- **"First-order hold"**



- **Higher order polynomials/Splines!**

How can we describe the structure of our robots?

- Transformation Matrices
  - Too many parameters
  - *Body frames* for every link
- Denavit-Hartenberg (DH) parameters
  - Optimal number of parameters
  - *Body frames* for every link
  - "Equivalent" with transformation matrices
  - We need to define the joint frames in a particular way
- Product of Exponentials (PoE)
  - Less parameters than transformation matrices, but not optimal
  - We do not need *body frames* for every link
  - Straightforward to compute

Universal Robot Description Format (1)

A URDF file is basically an XML file that describes our robot(s):

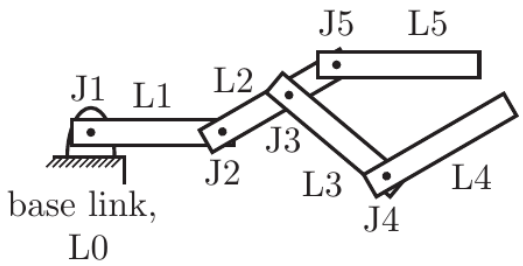
- It began as part of ROS, but now it is independent
- It allows modeling for the kinematic, dynamic and geometric parts of our robot
- It supports only open-chain structures (kinematic trees)
- Most robot softwares can read URDF files

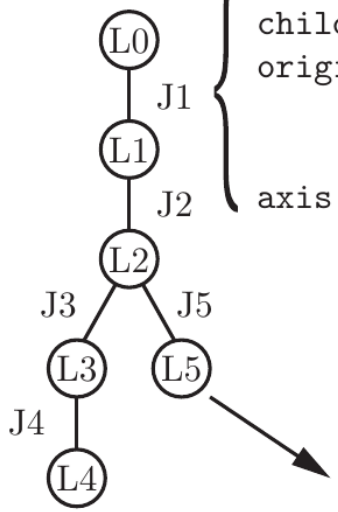
Universal Robot Description Format (2)

Every URDF file define joints and links:

- **Joints**:
  - Connect exactly 2 links: the *parent link* with the *child link*
  - Types: *revolute*, *prismatic* and *fixed*
  - *origin frame*: describes the pose of the child link with respect to the parent link in the *zero* (rest) configuration of the joint
  - *axis* (3D vector): descibes the axis of motion of the joint
  - ...
- **Links**:
  - *Intertial parameters*: mass, center of mass, ... (very important for accurate simulation)
  - Geometry for visualization
  - Geometry for collision detection (very important for simulation)
  - ...

Universal Robot Description Format (3)



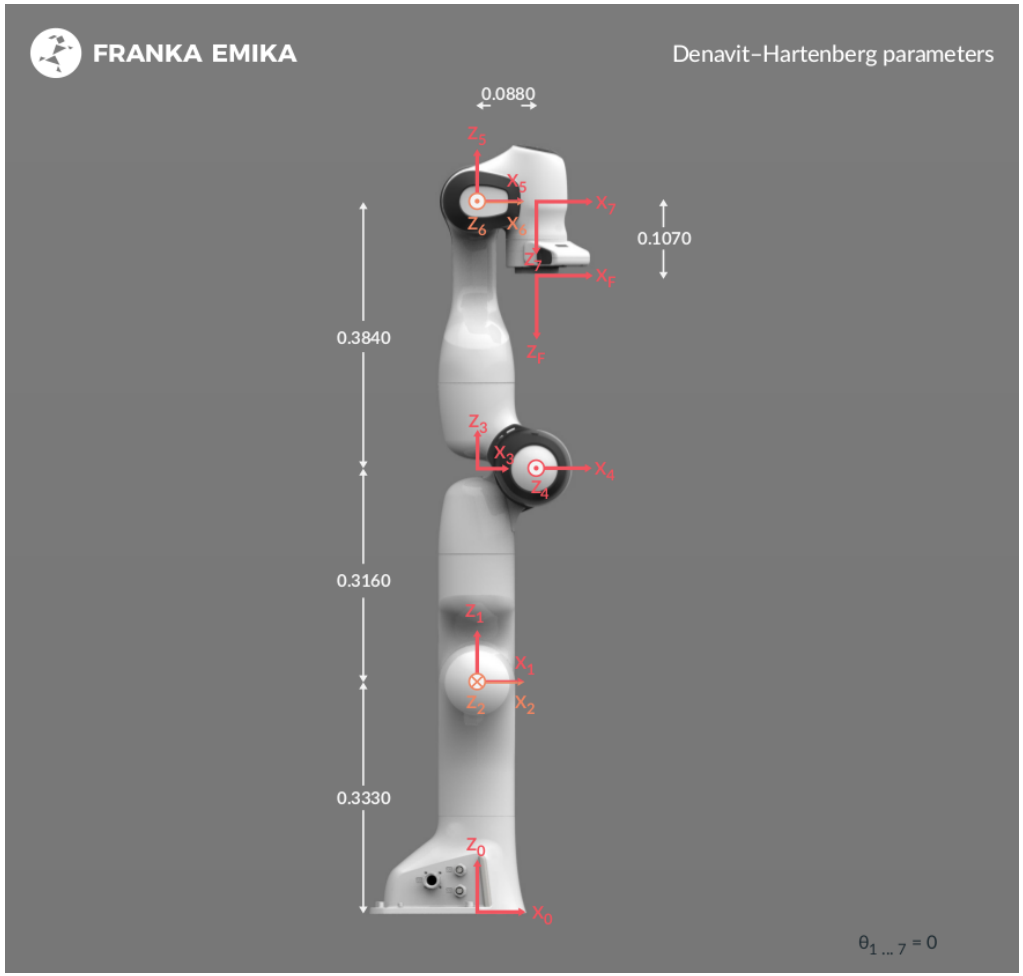


`parent:` J1's parent link, L0
`child:` J1's child link, L1
`origin:` the x–y–z and roll–pitch–yaw coords of the L1 frame relative to the L0 frame when J1 is zero
`axis:` the x–y–z unit vector along the rotation axis in the L1 frame

`mass:` L5's mass
`origin:` the x–y–z and roll–pitch–yaw coords of a frame at the center of mass of L5, relative to the L5 frame
`inertia:` six unique entries of inertia matrix in the `origin` frame

Source: Modern Robotics: Mechanics, Planning, and Control, *Kevin M. Lynch and Frank C. Park*, 2017, Cambridge University Press.

Manipulators

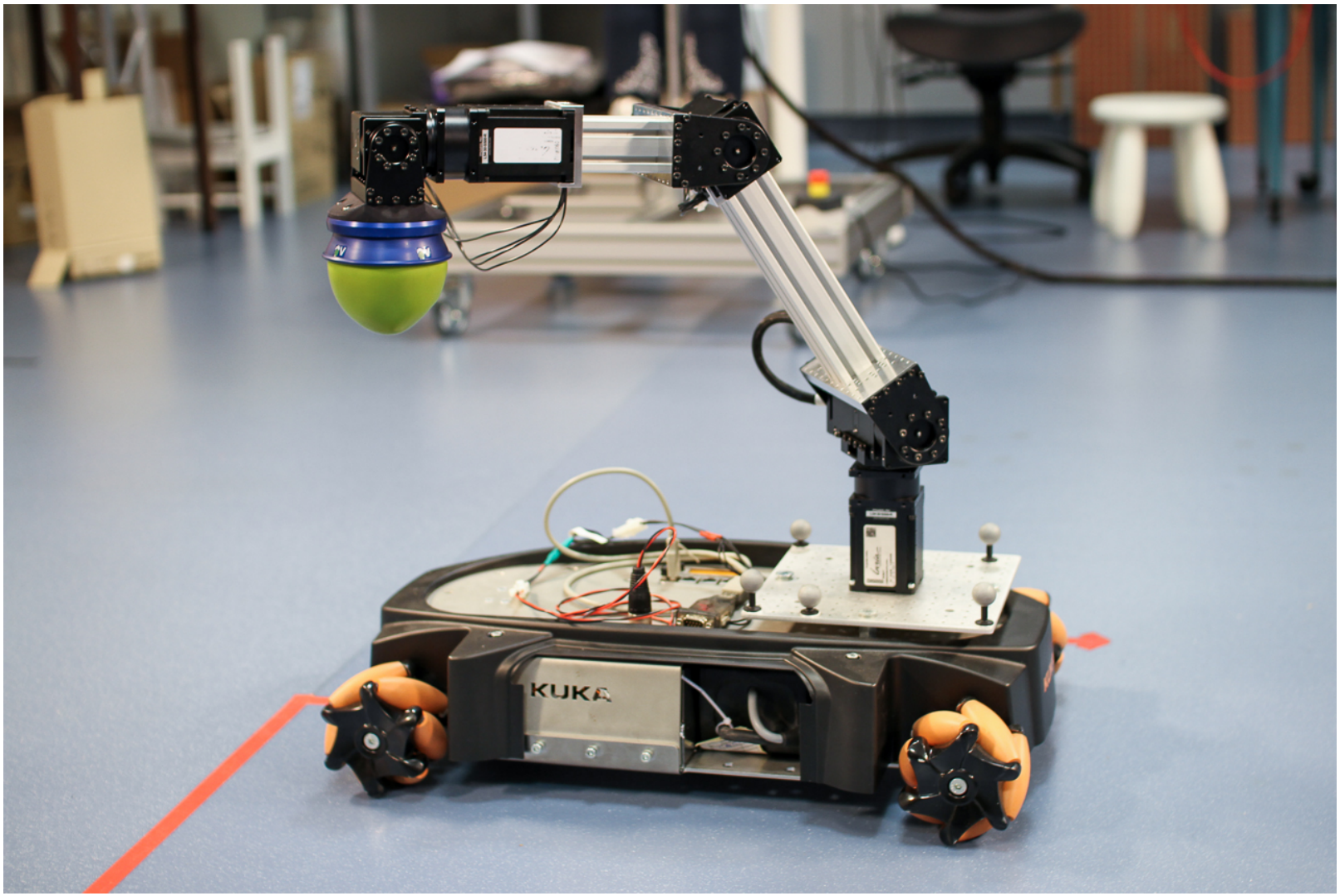


Source: `https://petercorke.com/robotics/franka-emika-panda-kinematics-and-singularities/`

Manipulators - Representation

- Usually base/root is fixed to world
- The configuration space is the joint position space
- $\boldsymbol{x} = (\boldsymbol{q}, \dot{\boldsymbol{q}})$ where $\boldsymbol{q} \in \mathbb{R}^n, \dot{\boldsymbol{q}} \in \mathbb{R}^n$, n degrees of freedom
- What happens if we move the base of a manipulator?
- Or if we mount a manipulator on a wheeled robot?

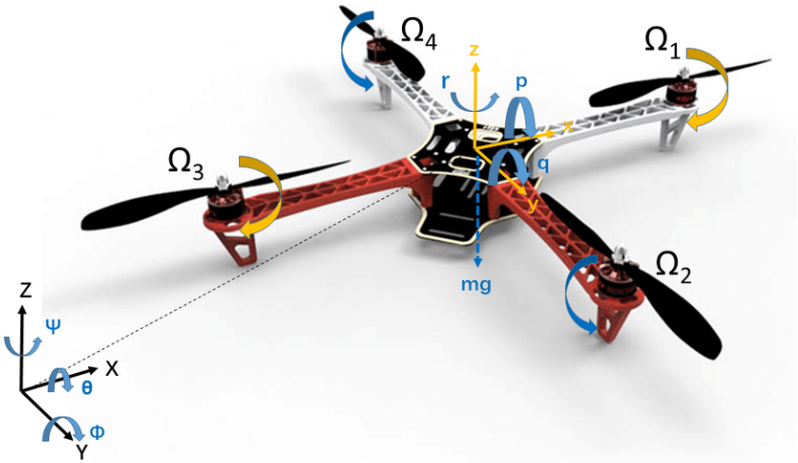
Manipulators on Mobile Robots

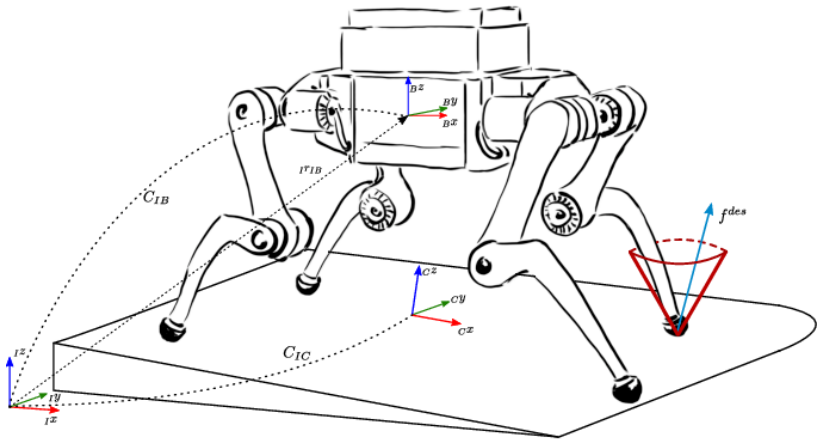


Source: `https://www.resibots.eu/photos.html#kuka-youbot-versaball`

Mobile Robots (1)

Mobile Robots (2)





Mobile Robots - Representation

- Joint positions and velocities
- AND 3D pose and velocity of the root/base body (base link)

Mobile Robots - Representation

- Joint positions and velocities
- AND 3D pose and velocity of the root/base body (base link)
- So we have:

$$\boldsymbol{x} = \begin{bmatrix} \boldsymbol{\omega}_w \\ \boldsymbol{T}_{wb} \\ \boldsymbol{q} \\ \dot{\boldsymbol{q}} \end{bmatrix}$$

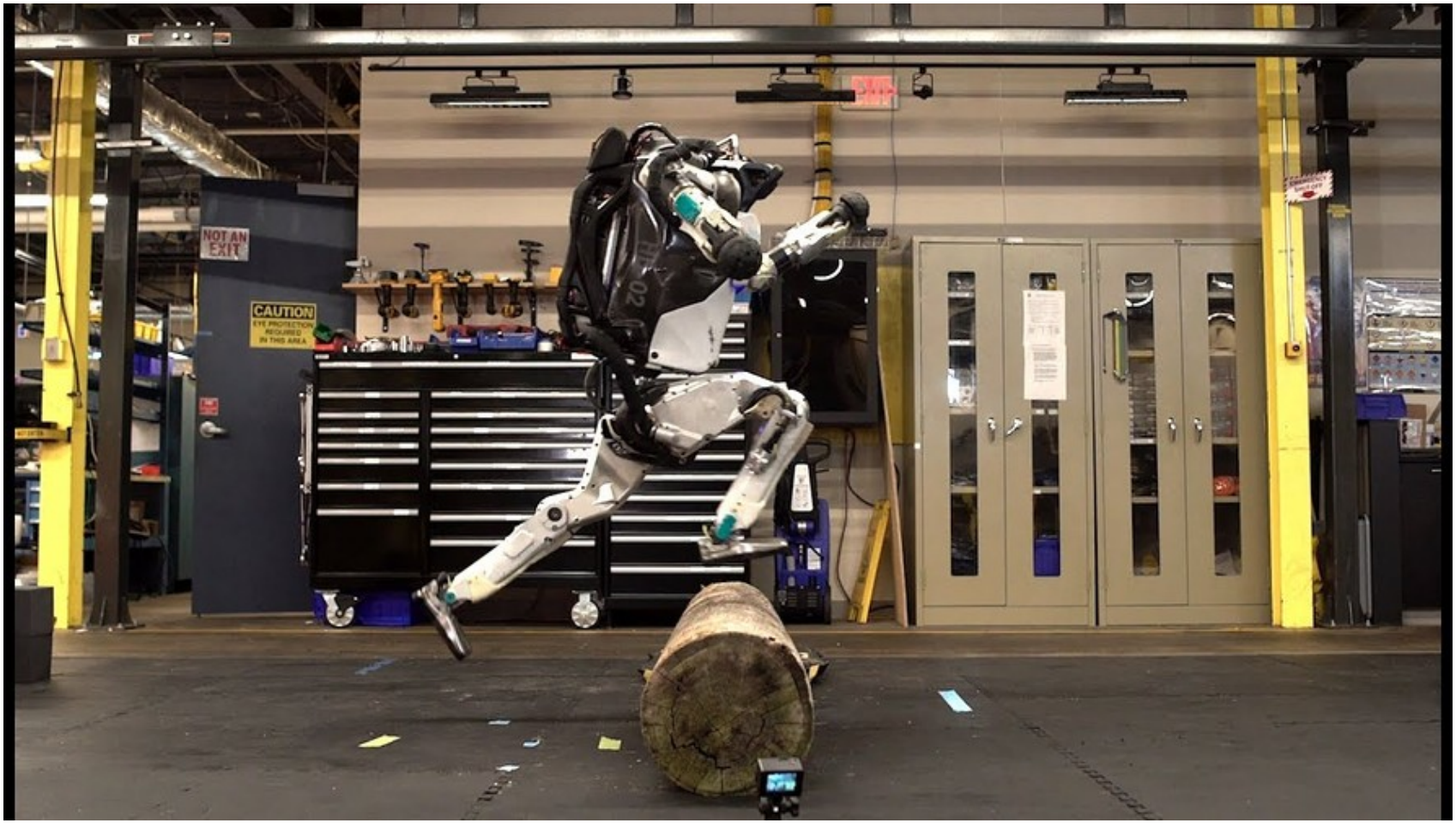
  where $\boldsymbol{q} \in \mathbb{R}^n, \dot{\boldsymbol{q}} \in \mathbb{R}^n$, $\boldsymbol{T}_{wb}$ and $\boldsymbol{\omega}_w$ are the transformation matrix of the base link and its angular velocity.

- We totally have (at most) $n + 6$ degrees of freedom

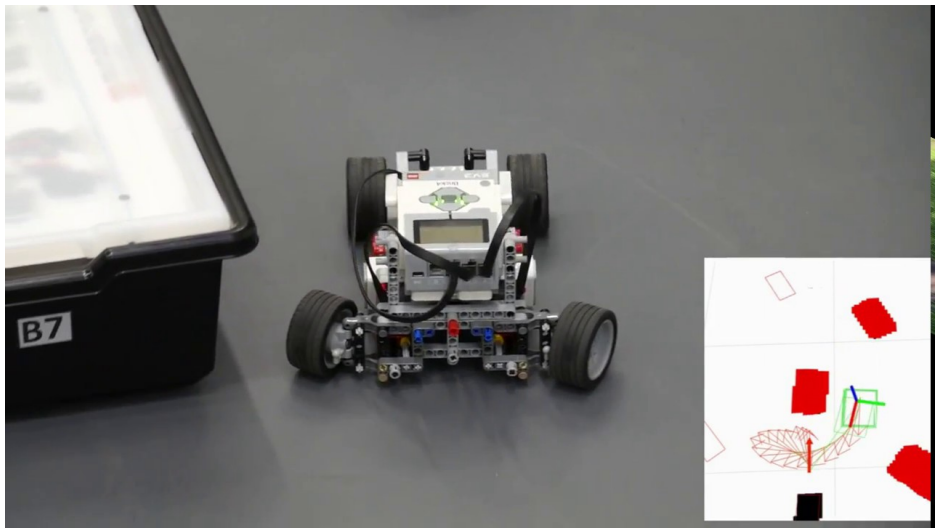
Types of Mobile Robots

1. Free-floating robots
   - 6 extra degrees of freedom
   - Humanoids, drones, ...
2. Planar/classical mobile robots
   - Robots with wheels
   - 3 extra degrees of freedom
   - Cars, bicycles, omnidirectional, ...
3. ...

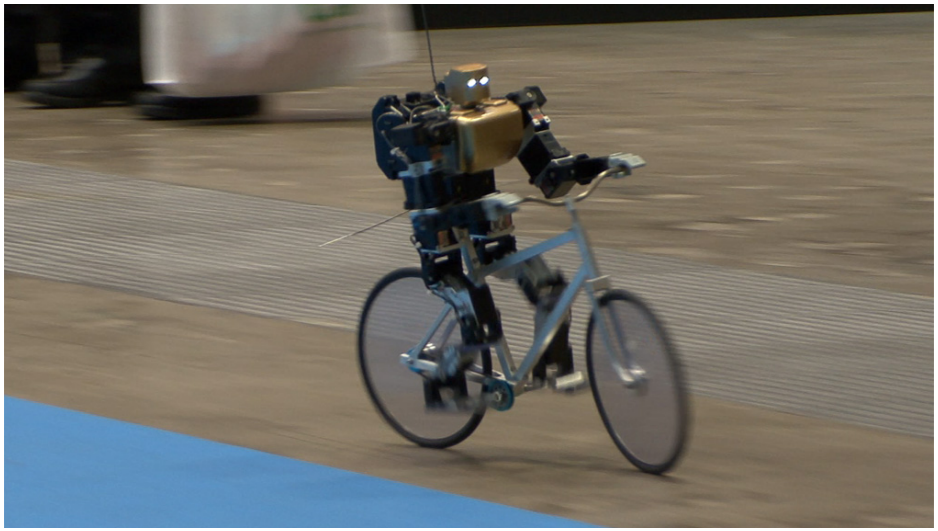
Free-floating robots



Planar/classical mobile robots (1)



Planar/classical mobile robots (2)



Assumptions:

- We usually ignore the dynamics and focus on the kinematics
- Robots roll on hard, flat ground without skidding
- Single rigid-body chassis with a transformation $\boldsymbol{T}_{wb} \in SE(2)$
- We denote $\boldsymbol{x} = [\theta, x, y]^T$.
- Planar Twist:

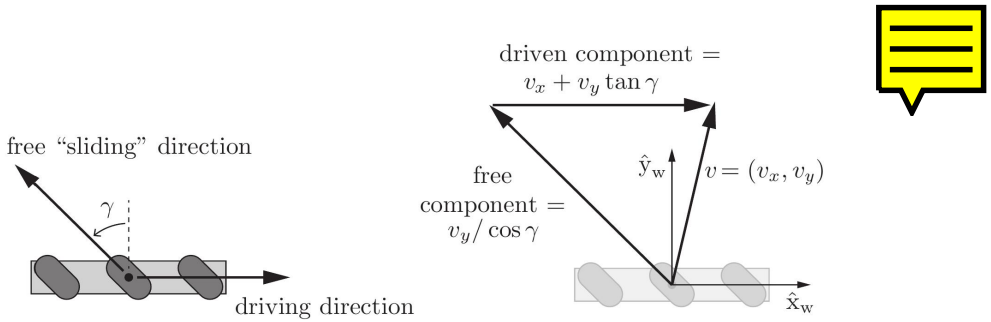
$$\mathcal{V}_b = \begin{bmatrix} \omega \\ v_x \\ v_y \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & \sin\theta \\ 0 & -\sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} \dot{\theta} \\ \dot{x} \\ \dot{y} \end{bmatrix}$$

$$\dot{\boldsymbol{x}} = \begin{bmatrix} \dot{\theta} \\ \dot{x} \\ \dot{y} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} \omega \\ v_x \\ v_y \end{bmatrix}$$

Types of Wheeled Mobile Robots

- Omnidirectional
  - No equality constraints on $\dot{\boldsymbol{x}}$
  - It can move freely! Even sideways!
- Nonholonomic
  - A single constraint of the form $A(\boldsymbol{x})\dot{\boldsymbol{x}} = 0$
  - Car-like motion
  - Can still reach any point in space!

Omnidirectional Mobile Robots



Source: Modern Robotics: Mechanics, Planning, and Control, *Kevin M. Lynch and Frank C. Park*, 2017, Cambridge University Press.

Kinematics:

In the wheel frame $(\hat{x}_w, \hat{y}_w)$ we can describe the motion as:

$$\begin{bmatrix} v_x \\ v_y \end{bmatrix} = v_{\text{drive}} \begin{bmatrix} 1 \\ 0 \end{bmatrix} + v_{\text{slide}} \begin{bmatrix} -\sin\gamma \\ \cos\gamma \end{bmatrix}$$

where γ (typically 0 or ± 45 degrees) is the angle of "free-sliding", v_{drive} is the velocity of the wheel center in the $\hat{x}_w$ direction and v_{slide} is the velocity in the "sliding" direction.

Omnidirectional Mobile Robots (2)

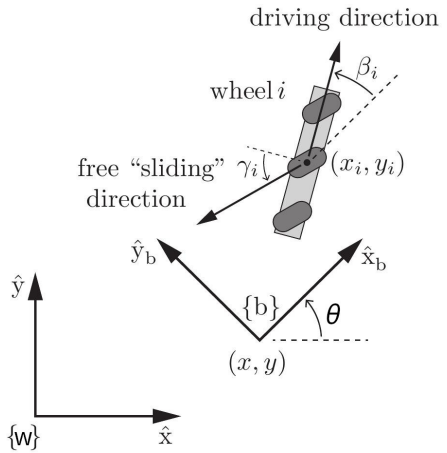
Let's re-write the previous equation solving for v_{slide} and v_{drive}:

$$v_{\text{drive}} = v_x + v_y \tan\gamma$$

$$v_{\text{slide}} = \frac{v_y}{\cos\gamma}$$

Assuming a radius r for the wheel and u is the angular velocity of the wheel (aka how fast it is turning):

$$u = \frac{v_{\text{drive}}}{r} = \frac{v_x + v_y \tan\gamma}{r}$$



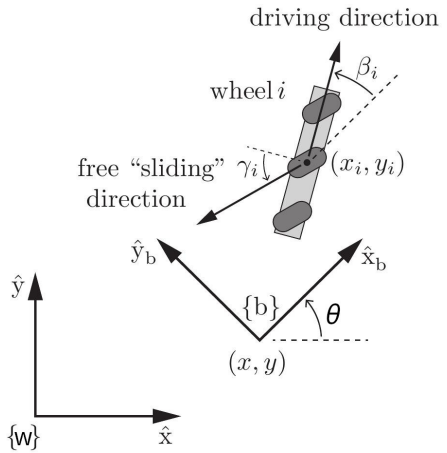
Source: Modern Robotics: Mechanics, Planning, and Control, *Kevin M. Lynch and Frank C. Park*, 2017, Cambridge University Press.

Omnidirectional Mobile Robots (3)

Notation! The center of the wheel and its driving direction are given by (β_i, x_i, y_i) expressed in the body frame of the robot. Now we can find the relation between u_i (of wheel i) and the full body twist $\dot{\boldsymbol{x}}$. We will derive an equation that looks like this:

$$u_i = h_i(\theta)\dot{\boldsymbol{x}}$$

What does this reminds us of?



Source: Modern Robotics: Mechanics, Planning, and Control, *Kevin M. Lynch and Frank C. Park*, 2017, Cambridge University Press.

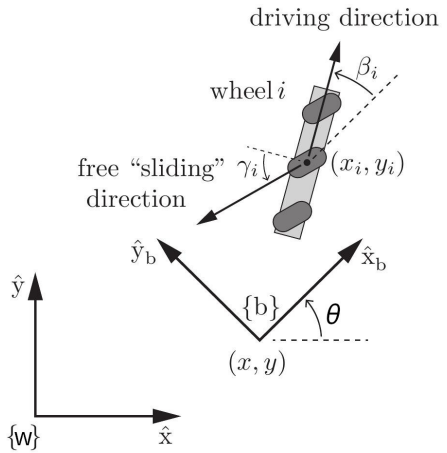
Omnidirectional Mobile Robots (3)

Notation! The center of the wheel and its driving direction are given by (β_i, x_i, y_i) expressed in the body frame of the robot. Now we can find the relation between u_i (of wheel i) and the full body twist $\dot{\boldsymbol{x}}$. We will derive an equation that looks like this:

$$u_i = h_i(\theta)\dot{\boldsymbol{x}}$$

What does this reminds us of?

Jacobians!



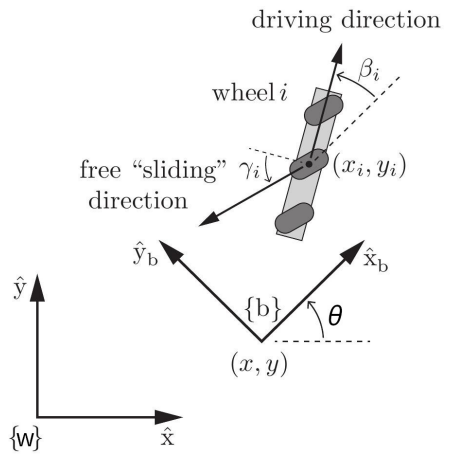
Source: Modern Robotics: Mechanics, Planning, and Control, *Kevin M. Lynch and Frank C. Park*, 2017, Cambridge University Press.

Omnidirectional Mobile Robots (3)

So let's start by:

$$\dot{\boldsymbol{x}} = \begin{bmatrix} \dot{\theta} \\ \dot{x} \\ \dot{y} \end{bmatrix}$$

What do we need to do first?



Source: Modern Robotics: Mechanics, Planning, and Control, *Kevin M. Lynch and Frank C. Park*, 2017, Cambridge University Press.

Omnidirectional Mobile Robots (3)

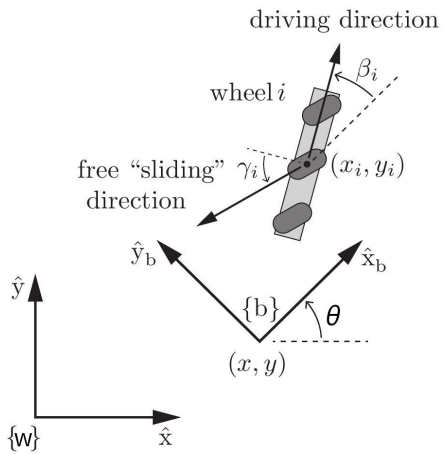
So let's start by:

$$\dot{\boldsymbol{x}} = \begin{bmatrix} \dot{\theta} \\ \dot{x} \\ \dot{y} \end{bmatrix}$$

What do we need to do first? Transform the world velocity into body frame:

$$\mathcal{V}_b = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & \sin\theta \\ 0 & -\sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} \dot{\theta} \\ \dot{x} \\ \dot{y} \end{bmatrix}$$

Then?



Source: Modern Robotics: Mechanics, Planning, and Control, *Kevin M. Lynch and Frank C. Park*, 2017, Cambridge University Press.

Omnidirectional Mobile Robots (3)

So let's start by:

$$\dot{\boldsymbol{x}} = \begin{bmatrix} \dot{\theta} \\ \dot{x} \\ \dot{y} \end{bmatrix}$$

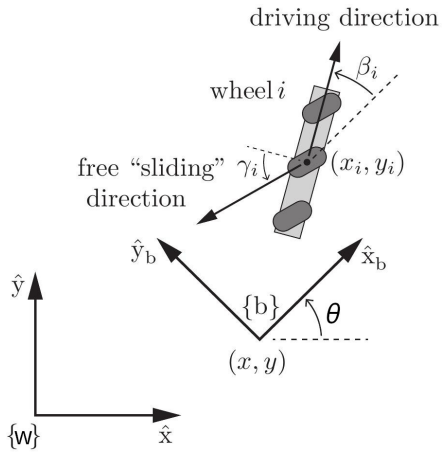
What do we need to do first? Transform the world velocity into body frame:

$$\mathcal{V}_b = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & \sin\theta \\ 0 & -\sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} \dot{\theta} \\ \dot{x} \\ \dot{y} \end{bmatrix}$$

Then? Compute the linear velocity of the wheel in the body frame:

$$\boldsymbol{v}_b = \begin{bmatrix} -y_i & 1 & 0 \\ x_i & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & \sin\theta \\ 0 & -\sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} \dot{\theta} \\ \dot{x} \\ \dot{y} \end{bmatrix}$$

And?



Source: Modern Robotics: Mechanics, Planning, and Control, *Kevin M. Lynch and Frank C. Park*, 2017, Cambridge University Press.

Omnidirectional Mobile Robots (3)

So let's start by:

$$\dot{\boldsymbol{x}} = \begin{bmatrix} \dot{\theta} \\ \dot{x} \\ \dot{y} \end{bmatrix}$$

What do we need to do first? Transform the world velocity into body frame:

$$\mathcal{V}_b = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & \sin\theta \\ 0 & -\sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} \dot{\theta} \\ \dot{x} \\ \dot{y} \end{bmatrix}$$

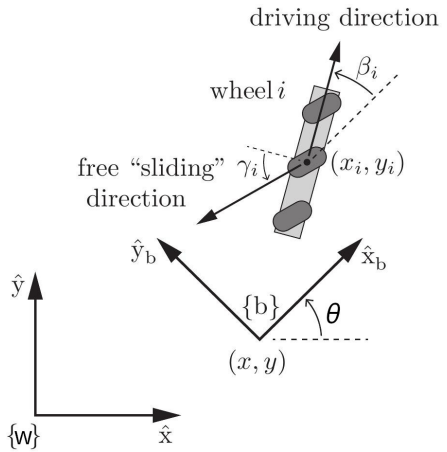
Then? Compute the linear velocity of the wheel in the body frame:

$$\boldsymbol{v}_b = \begin{bmatrix} -y_i & 1 & 0 \\ x_i & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & \sin\theta \\ 0 & -\sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} \dot{\theta} \\ \dot{x} \\ \dot{y} \end{bmatrix}$$

And? Transform (aka rotate) this velocity into the wheel frame:

$$\begin{bmatrix} v_x \\ v_y \end{bmatrix} = \begin{bmatrix} \cos\beta_i & \sin\beta_i \\ -\sin\beta_i & \cos\beta_i \end{bmatrix} \begin{bmatrix} -y_i & 1 & 0 \\ x_i & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & \sin\theta \\ 0 & -\sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} \dot{\theta} \\ \dot{x} \\ \dot{y} \end{bmatrix}$$

Finally?



Source: Modern Robotics: Mechanics, Planning, and Control, *Kevin M. Lynch and Frank C. Park*, 2017, Cambridge University Press.

Omnidirectional Mobile Robots (3)

So let's start by:

$$\dot{\boldsymbol{x}} = \begin{bmatrix} \dot{\theta} \\ \dot{x} \\ \dot{y} \end{bmatrix}$$

What do we need to do first? Transform the world velocity into body frame:

$$\mathcal{V}_b = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & \sin\theta \\ 0 & -\sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} \dot{\theta} \\ \dot{x} \\ \dot{y} \end{bmatrix}$$

Then? Compute the linear velocity of the wheel in the body frame:

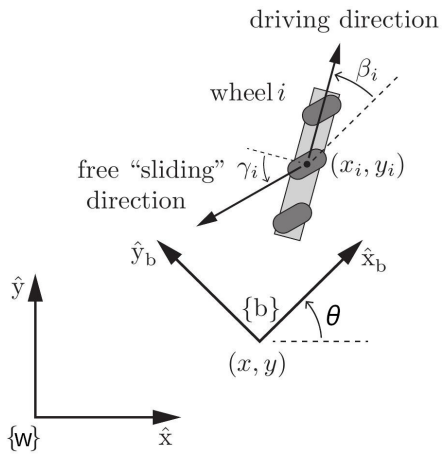
$$\boldsymbol{v}_b = \begin{bmatrix} -y_i & 1 & 0 \\ x_i & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & \sin\theta \\ 0 & -\sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} \dot{\theta} \\ \dot{x} \\ \dot{y} \end{bmatrix}$$

And? Transform (aka rotate) this velocity into the wheel frame:

$$\begin{bmatrix} v_x \\ v_y \end{bmatrix} = \begin{bmatrix} \cos\beta_i & \sin\beta_i \\ -\sin\beta_i & \cos\beta_i \end{bmatrix} \begin{bmatrix} -y_i & 1 & 0 \\ x_i & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & \sin\theta \\ 0 & -\sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} \dot{\theta} \\ \dot{x} \\ \dot{y} \end{bmatrix}$$

Finally? Transform this into wheel angular velocity:

$$u_i = \begin{bmatrix} \frac{1}{r} & \frac{\tan\gamma_i}{r} \end{bmatrix} \begin{bmatrix} \cos\beta_i & \sin\beta_i \\ -\sin\beta_i & \cos\beta_i \end{bmatrix} \begin{bmatrix} -y_i & 1 & 0 \\ x_i & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & \sin\theta \\ 0 & -\sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} \dot{\theta} \\ \dot{x} \\ \dot{y} \end{bmatrix}$$



Source: Modern Robotics: Mechanics, Planning, and Control, *Kevin M. Lynch and Frank C. Park*, 2017, Cambridge University Press.

Omnidirectional Mobile Robots (4)

Putting all together, we get:

$$h_i(\theta) = \frac{1}{r_i \cos \gamma_i} \begin{bmatrix} x_i \sin(\beta_i + \gamma_i) - y_i \cos(\beta_i + \gamma_i) \\ \cos(\beta_i + \gamma_i + \theta) \\ \sin(\beta_i + \gamma_i + \theta) \end{bmatrix}^T$$

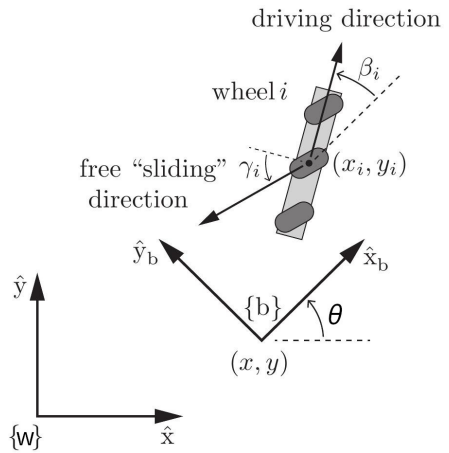
If we have m wheels, we can something as follows:

$$\boldsymbol{u} = \begin{bmatrix} u_1 \\ \vdots \\ u_m \end{bmatrix} = H(\theta)\dot{\boldsymbol{x}} = \begin{bmatrix} h_1(\theta) \\ \vdots \\ h_m(\theta) \end{bmatrix} \dot{\boldsymbol{x}}$$

Also:

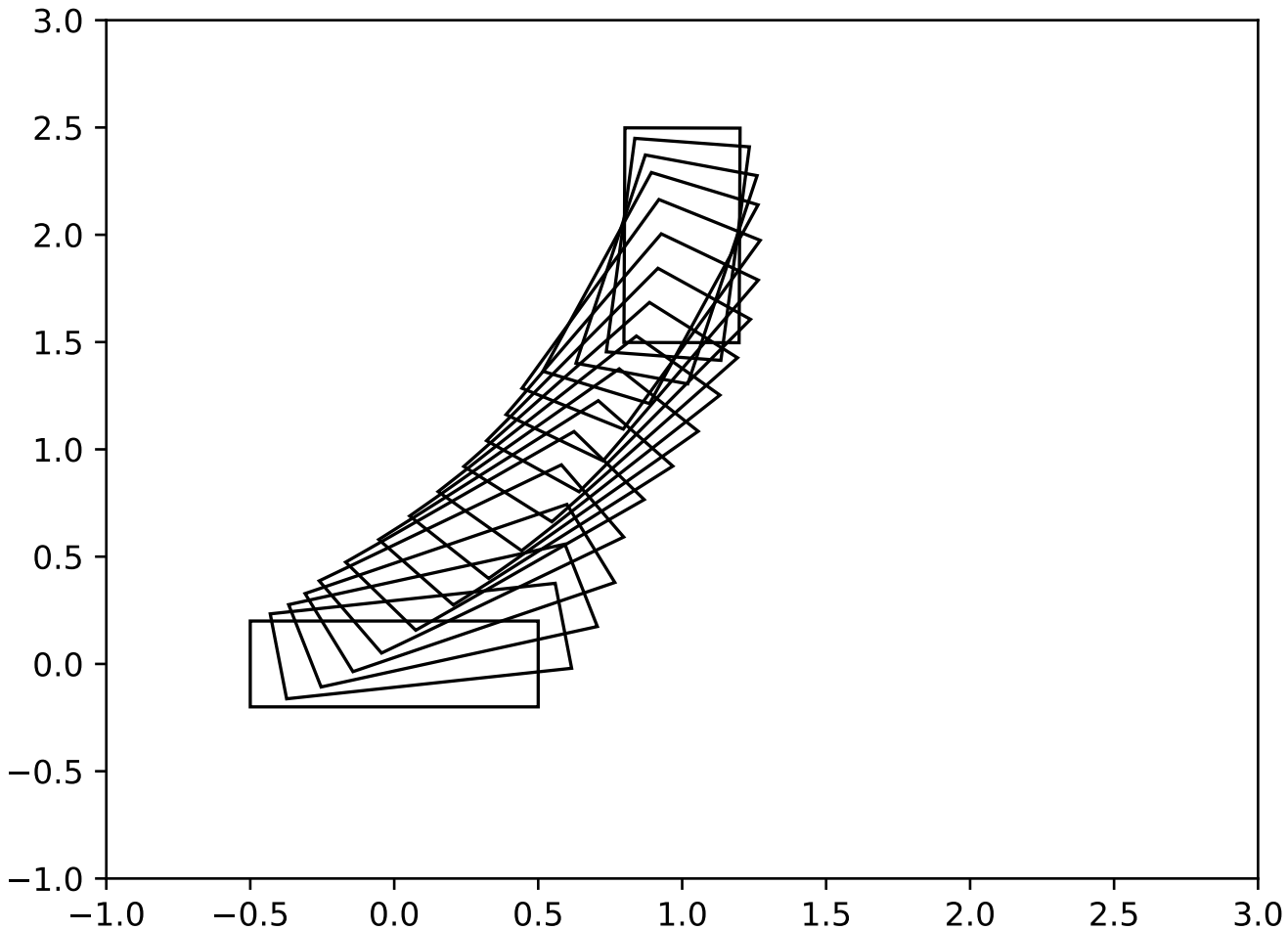
$$\boldsymbol{u} = H(0)\mathcal{V}_b = \begin{bmatrix} h_1(0) \\ \vdots \\ h_m(0) \end{bmatrix} \begin{bmatrix} \omega \\ v_x \\ v_y \end{bmatrix}$$

Independent of θ!

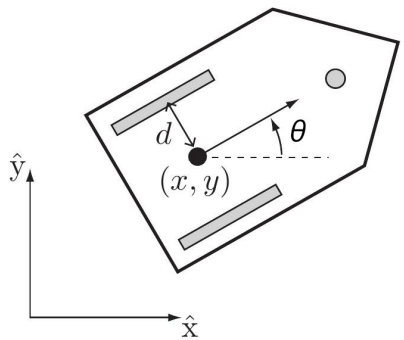


Source: Modern Robotics: Mechanics, Planning, and Control, *Kevin M. Lynch and Frank C. Park*, 2017, Cambridge University Press.

Omnidirectional Mobile Robots - Code Example



Differential Drive Robots



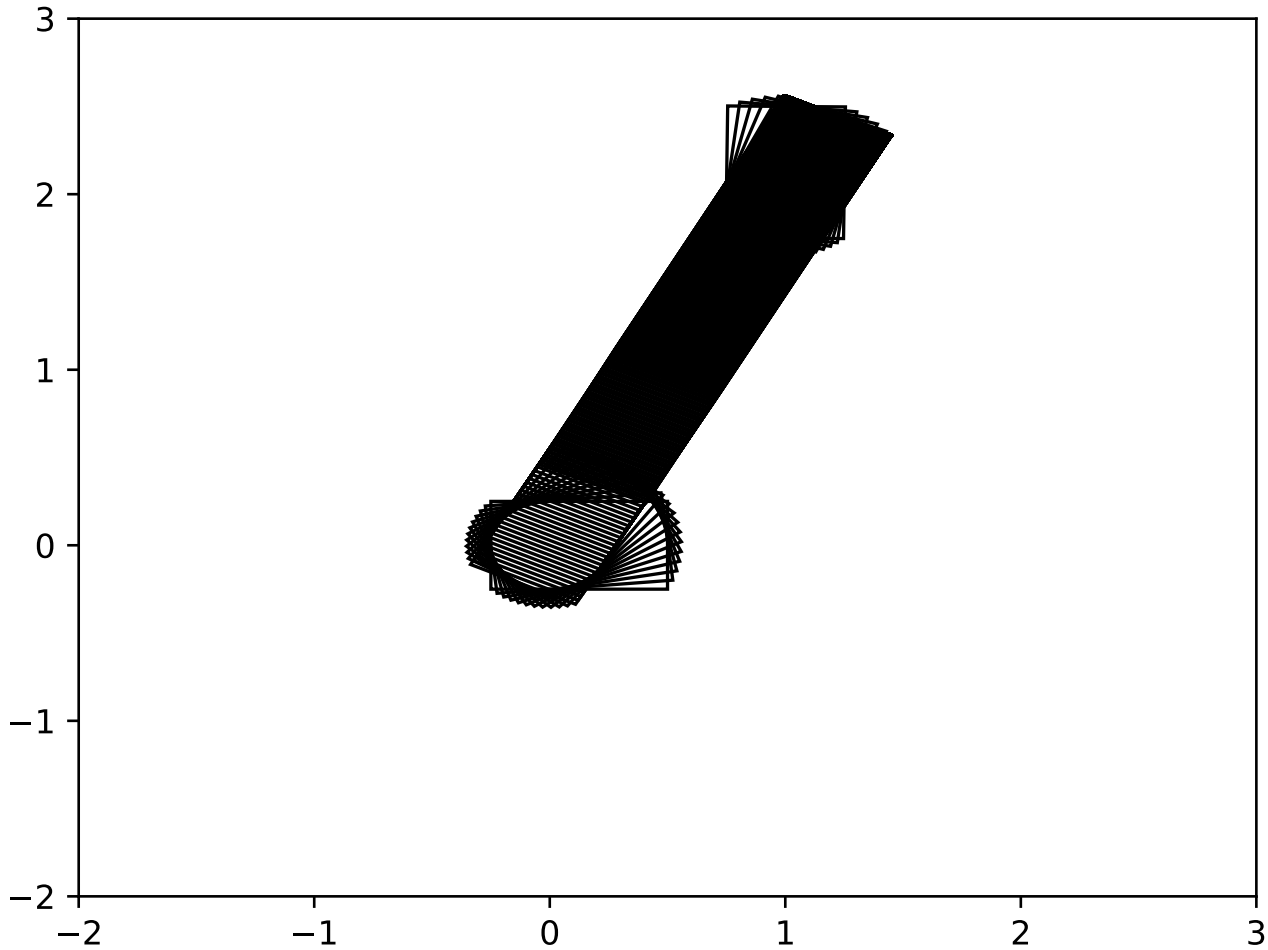
Source: Modern Robotics: Mechanics, Planning, and Control, *Kevin M. Lynch and Frank C. Park*, 2017, Cambridge University Press.

Kinematics:
The state of a differential drive robot is $\boldsymbol{x} = [\theta, x, y, \theta_L, \theta_R]^T$. And the kinematic equations are given by:

$$\begin{bmatrix} \dot{\theta} \\ \dot{x} \\ \dot{y} \\ \dot{\theta_L} \\ \dot{\theta_R} \end{bmatrix} = \begin{bmatrix} -\frac{r}{2d} & \frac{r}{2d} \\ \frac{r}{2}\cos\theta & \frac{r}{2}\cos\theta \\ \frac{r}{2}\sin\theta & \frac{r}{2}\sin\theta \\ 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} u_L \\ u_R \end{bmatrix}$$

where (θ, x, y) is the pose of the robot in world space, r are the wheel radii, d is the distance of the wheels from the robot center, θ_L, θ_R are the wheel rotation angles (values from encoder of the motors) and u_L, u_R are the wheel angular velocities (commands).

Differential Drive Robots - Code Example



Motion Planning (1)

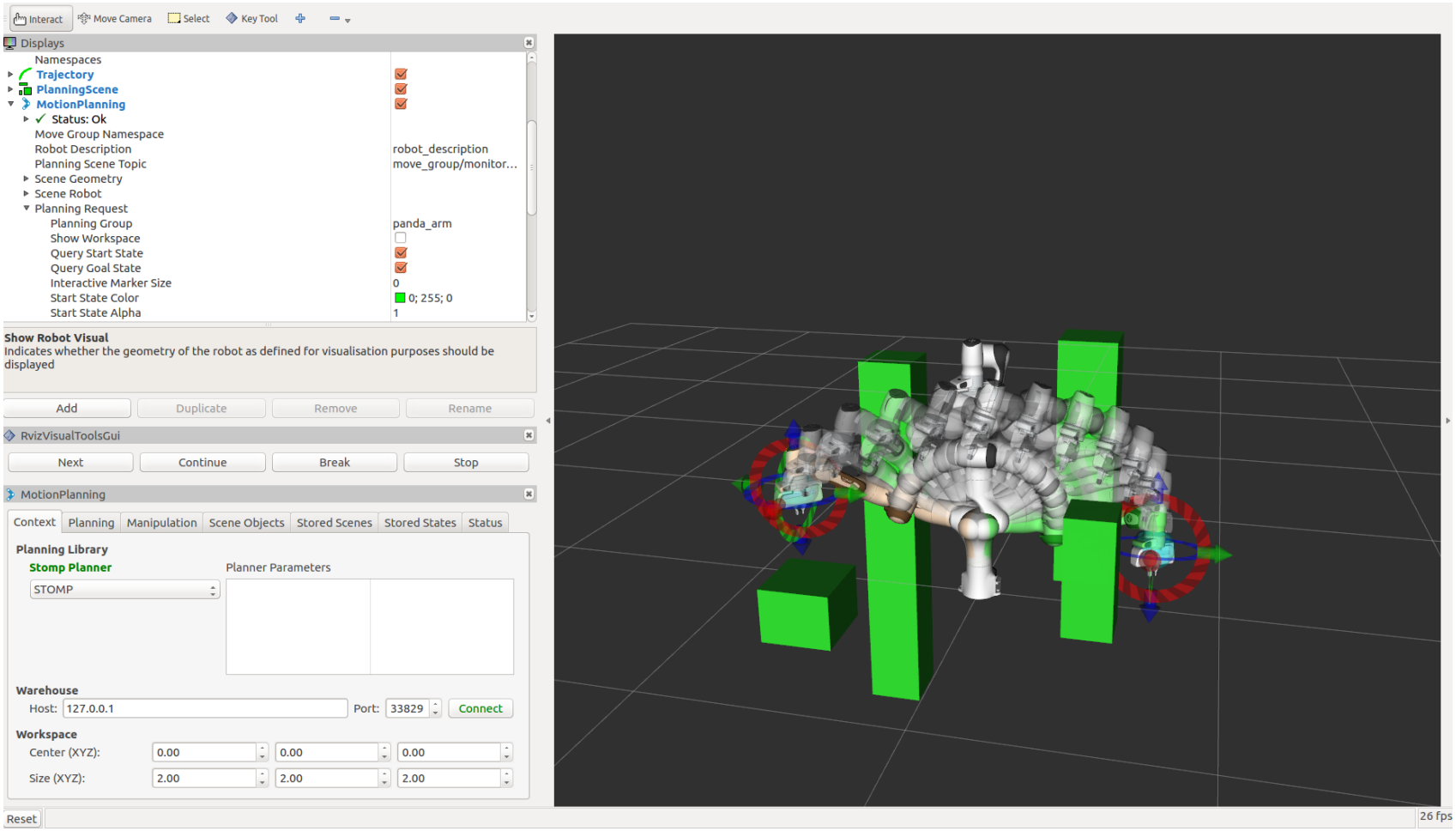
Theorem (General Motion Planning Problem)

Assuming that we have an initial state $\boldsymbol{x}(0) = \boldsymbol{x}_{start}$ *and a final state (goal)* $\boldsymbol{x}_{goal}$*, we would like to find the total time* T_f *and controls* $\boldsymbol{u} : [0, T_f] \rightarrow \mathcal{U}$ *such that when following the model of the robot/environment* $\dot{\boldsymbol{x}} = f(\boldsymbol{x}, \boldsymbol{u})$*, we have* $\boldsymbol{x}(T_f) = \boldsymbol{x}_{goal}$ *and* $\forall t \in [0, T_f]$ *the state* $\boldsymbol{x}(t)$ *is a valid state (no collisions or bad state values).*

Motion Planning (2)

- We assume that we know the robot and the environment
- Model f gives us the dynamics of our system (usually in the kinematic level)
- We assume that we have a low-level controller so that we can follow $\boldsymbol{x}(t)$

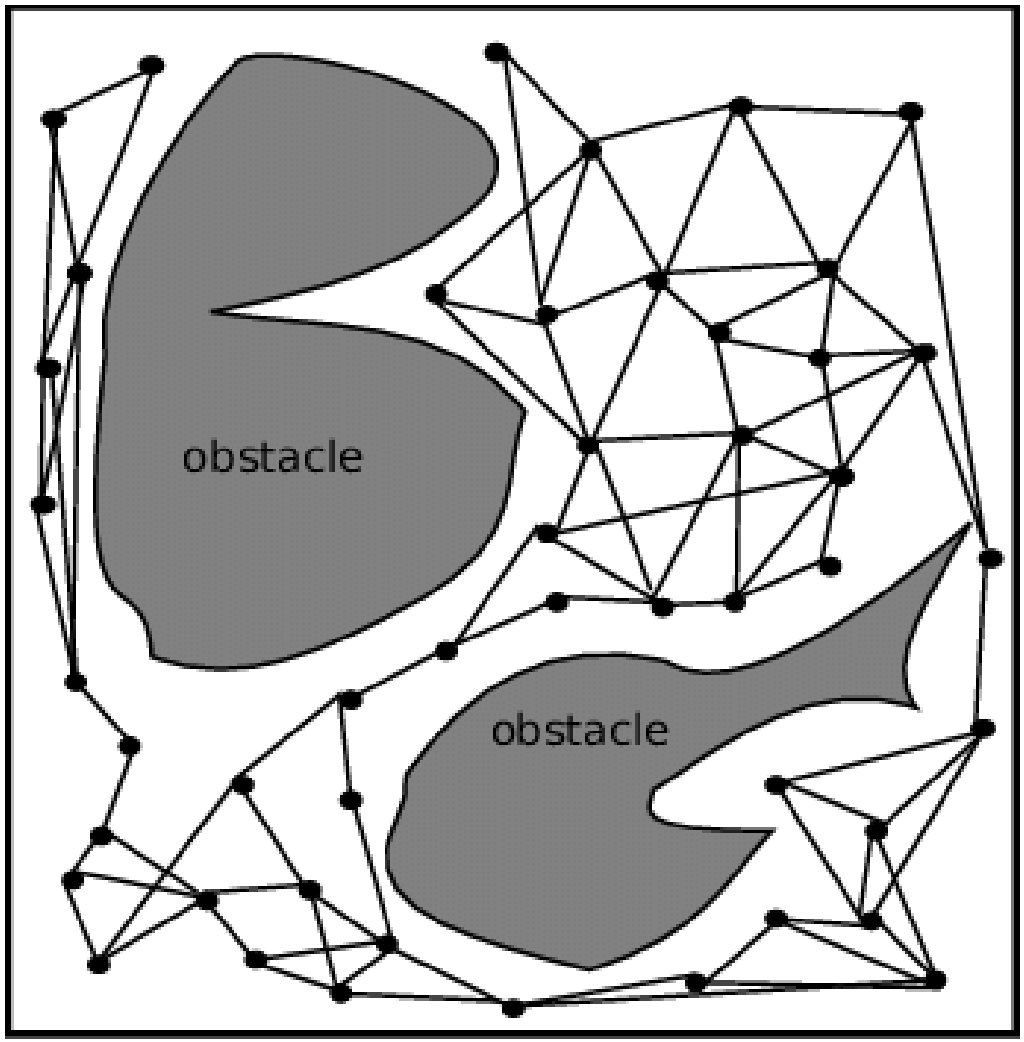
Motion Planning (3)



Motion Planning and Search (1)

- Search methods are very similar and important for motion planning
- Many problems of motion planning can be solved by search
- Most motion planning algorithms describe the state space by keeping a tree or a graph of states

Motion Planning and Search (2)

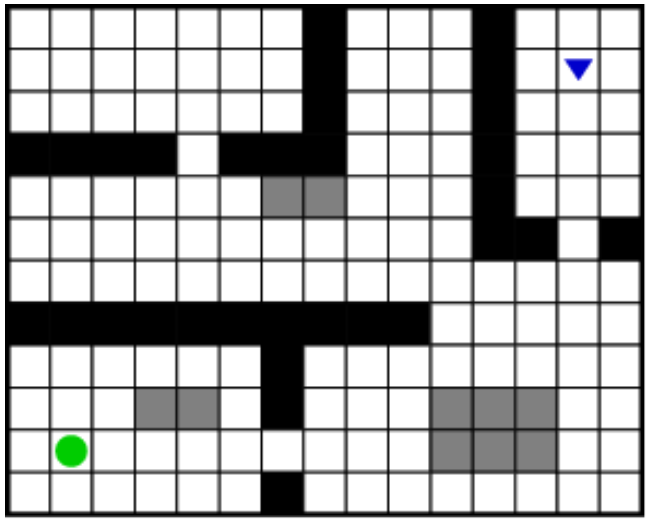


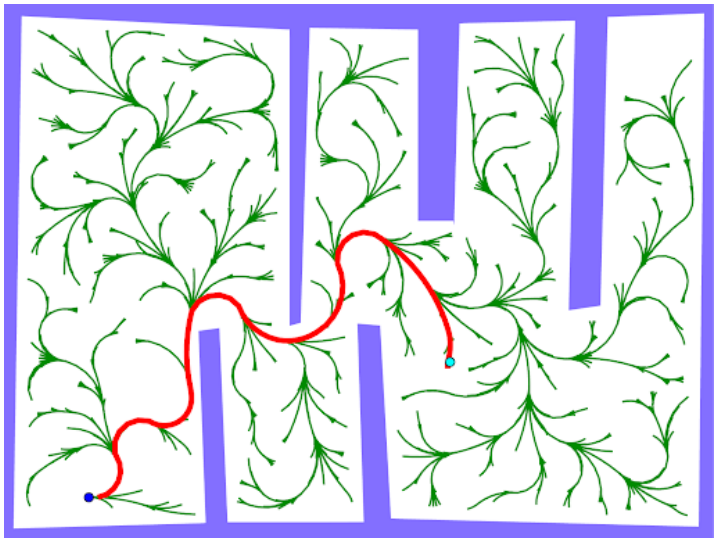
Source: Murray, S., Floyd-Jones, W., Qi, Y., Sorin, D.J. and Konidaris, G.D., 2016, June. Robot Motion Planning on a Chip. In Robotics: Science and Systems.

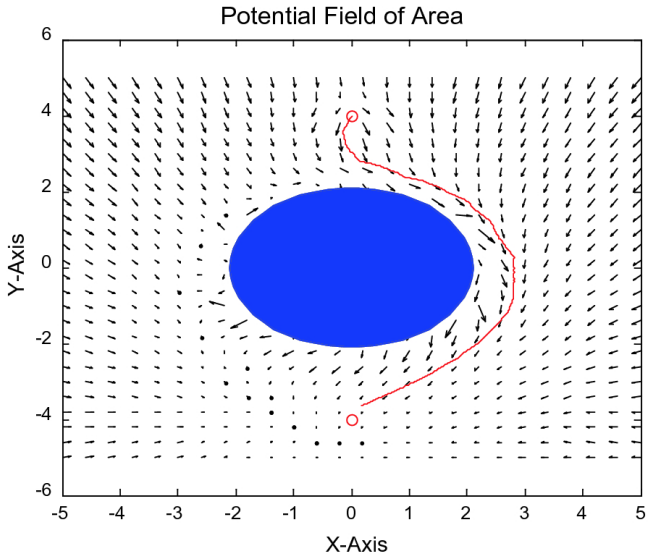
Types of Motion Planning Algorithms (1)

- Complete
- Grid-based
- Sampling-based
- Virtual Potential Fields
- Non-linear Optimization: *More on this later in the course!*

Types of Motion Planning Algorithms (2)





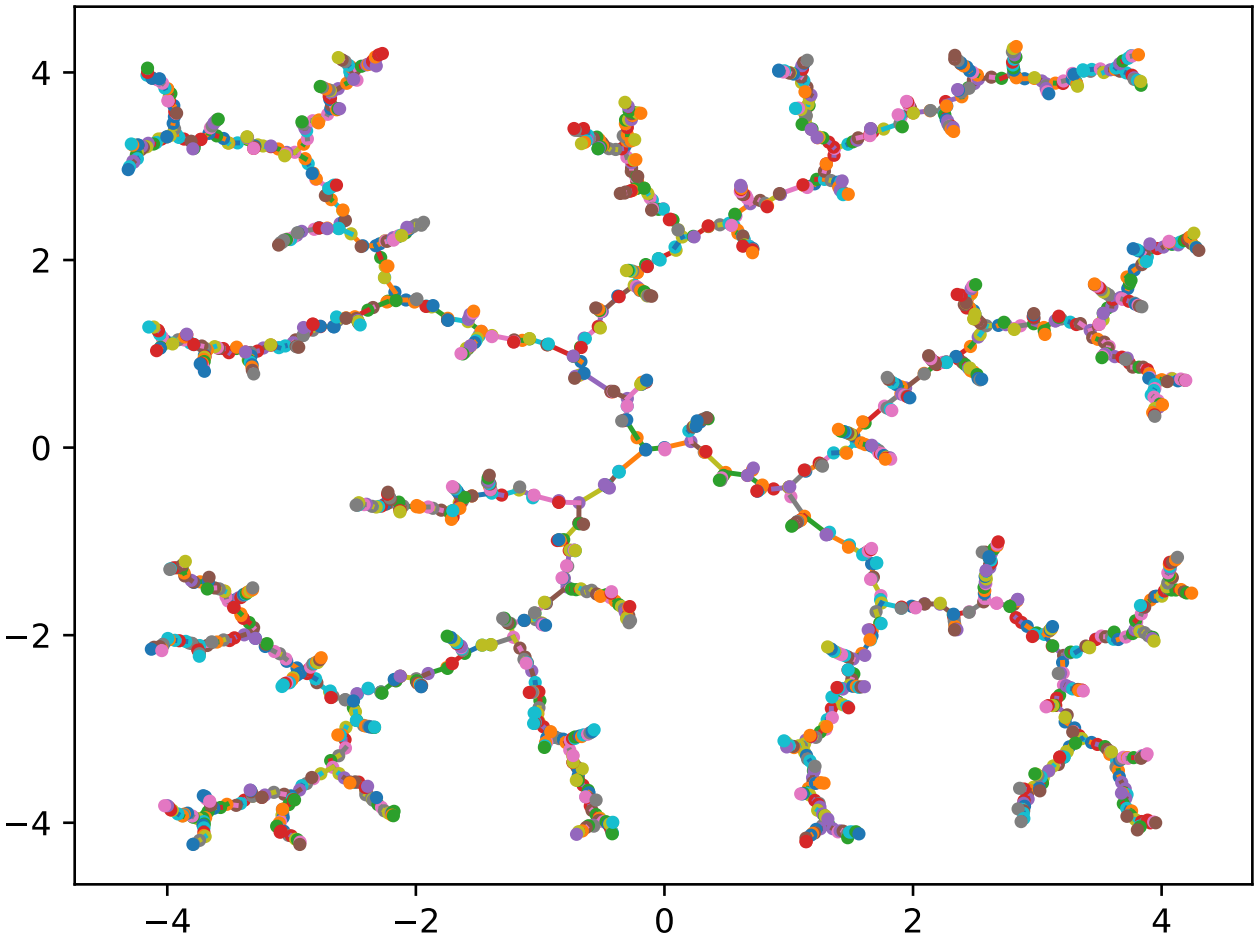


Rapidly exploring Random Trees (RRT)

RRT Algorithm:

1 : Initialize search tree with $\boldsymbol{x}_{\text{start}}$
2 : while *some stopping criteria is not met*
3 : sample $\boldsymbol{x}_{\text{sample}} \sim \mathcal{X}$
4 : find $\boldsymbol{x}_{\text{nearest}}$ nearest node of $\boldsymbol{x}_{\text{sample}}$ in tree
5 : connect $\boldsymbol{x}_{\text{nearest}}$ to $\boldsymbol{x}_{\text{new}}$ in direction of $\boldsymbol{x}_{\text{sample}}$
6 : **if** success
7 : add $\boldsymbol{x}_{\text{new}}$ to the tree with an edge from $\boldsymbol{x}_{\text{nearest}}$
8 : **if** $\boldsymbol{x}_{\text{new}} \in \mathcal{X}_{\text{goal}}$ return SUCCESS
9 : return FAILURE

RRT - Code Example



Bibliography

Chapters 10 and 13 from **Modern Robotics: Mechanics, Planning, and Control**, *Kevin M. Lynch and Frank C. Park*, 2017, Cambridge University Press. ebook

Thank you

- **Any Questions?**
- **Office Hours**:
  - **Tue-Wed** *(10:00-12:00)*
  - 24/7 by email (`costashatz@upatras.gr`, subject: *ECE_RSI_AM*)
- **Material and Announcements**

Laboratory of Automation & Robotics