CS 3630!

Lecture 11:
An omnidirectional
Logistics Robot

Logistics Robots

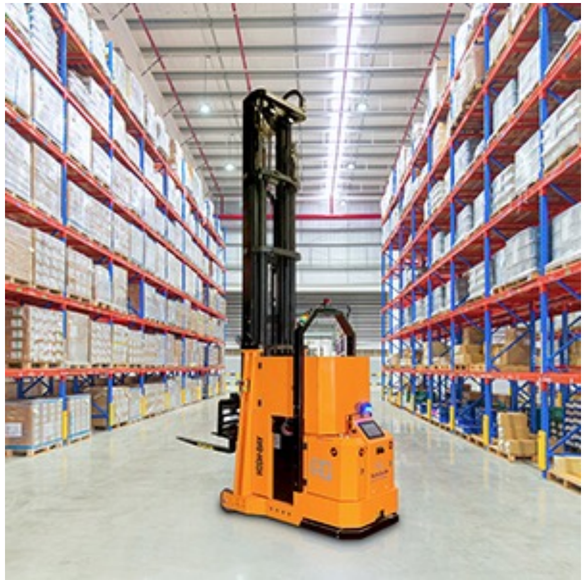
What exactly is logistics?

- Logistics is the management of the flow of things from point of origin to point of consumption [Wikipedia].
- This typically involves multiple stages of packaging, routing, transport.
- There are plenty of robotics applications:
  - Loading/unloading
  - Palletizing
  - Cargo container transport
  - Packaging
  - Last mile delivery
  - Warehouse operations

➢ For now, we will consider the narrow problem of warehouse operations, in particular, the problem of moving inventory from Point A to Point B in a warehouse.

Robots in Warehouses

A Few Warehouse Robots



Autoguide Mobile Robots

GreyOrange Inc.

Tompkins Robotics

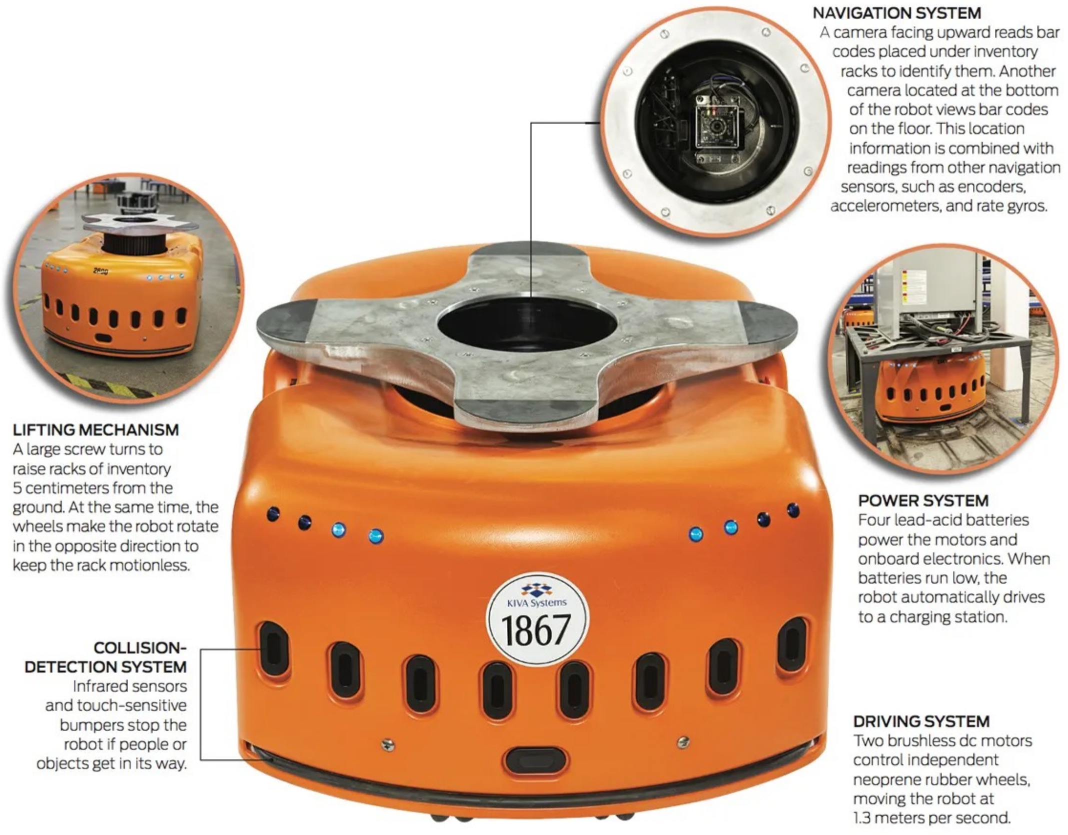
Milvus Robotics

A few mobile robots whose purpose in life is to move inventory from place to place in large warehouses.

From Kiva to Amazon Robotics

- 2003: Kiva Systems founded
- 2009: Rank #6 in Inc. 500 list of fastest growing co's in America
- 2012: Acquired by Amazon for $775M
- 2015: Name change: Amazon Robotics LLC
- 2019: *More than 200,000 robots deployed in Amazon warehouses*

Peter Wurman, Mick Mountz, Raff D'Andrea



IEEE Spectrum, Jul 2008

Amazon’s Warehouse Robots

Fetch Robotics

- Cloud robotics platform (claim to be the first)
- Mobile manipulation
- Sponsored competition at ICRA (GT won, and took home a shiny new robot).
- 2014: Founded (after Willow Garage ended)
- 2019: AI Breakthrough Award (best overall robotics company)
- 2021: Acquired by Zebra for $305M

CEO, Melonee Wise

Autonomous Mobile Robots

In the world of warehouse robotics, we there are two main categories or mobile robot platform:

- Automated Guided Vehicles (AGVs)
  - Follow fixed routes
  - Rely on wires or magnets embedded in the floor to track routes
  - Simple sensing to avoid collisions (typically, simply stop when an obstacle appears)
  - Rely on predictable and known environment
  - Train the humans to avoid the robots

- Autonomous Mobile Robots (AMRs)
  - Capable of planning general motion
  - Typically require a map of the environment
  - Can navigate based on obstacles (i.e., more than simple collision avoidance)
  - Robots know how to avoid the humans

In this chapter…

- Omnidirectional mobile robots
  - Can move in arbitrary directions
  - Control input is wheel angular velocity
  - Easy to convert wheel angular velocity to robot velocity
  - Forces and torques aren't important
- Continuous state space
  - Robot position is specified by x-y coordinates
  - Coordinates are real numbers, not discrete grid points or names of rooms
- Discrete time system
  - No real need for continuous time
  - Provides access to nice tools from Bayesian inference
- Fairly simple sensors
  - Proximity (binary sensor that detects obstacles)
  - Range (using RFID tags)
  - Pseudo-GPS (mainly to introduce conditional Gaussian models)

Continuous State

In this chapter, we level up to continuous state for the very first time.

The question: how to represent knowledge? Three options:

- Gaussian density
- Finite elements
- Sampling-based

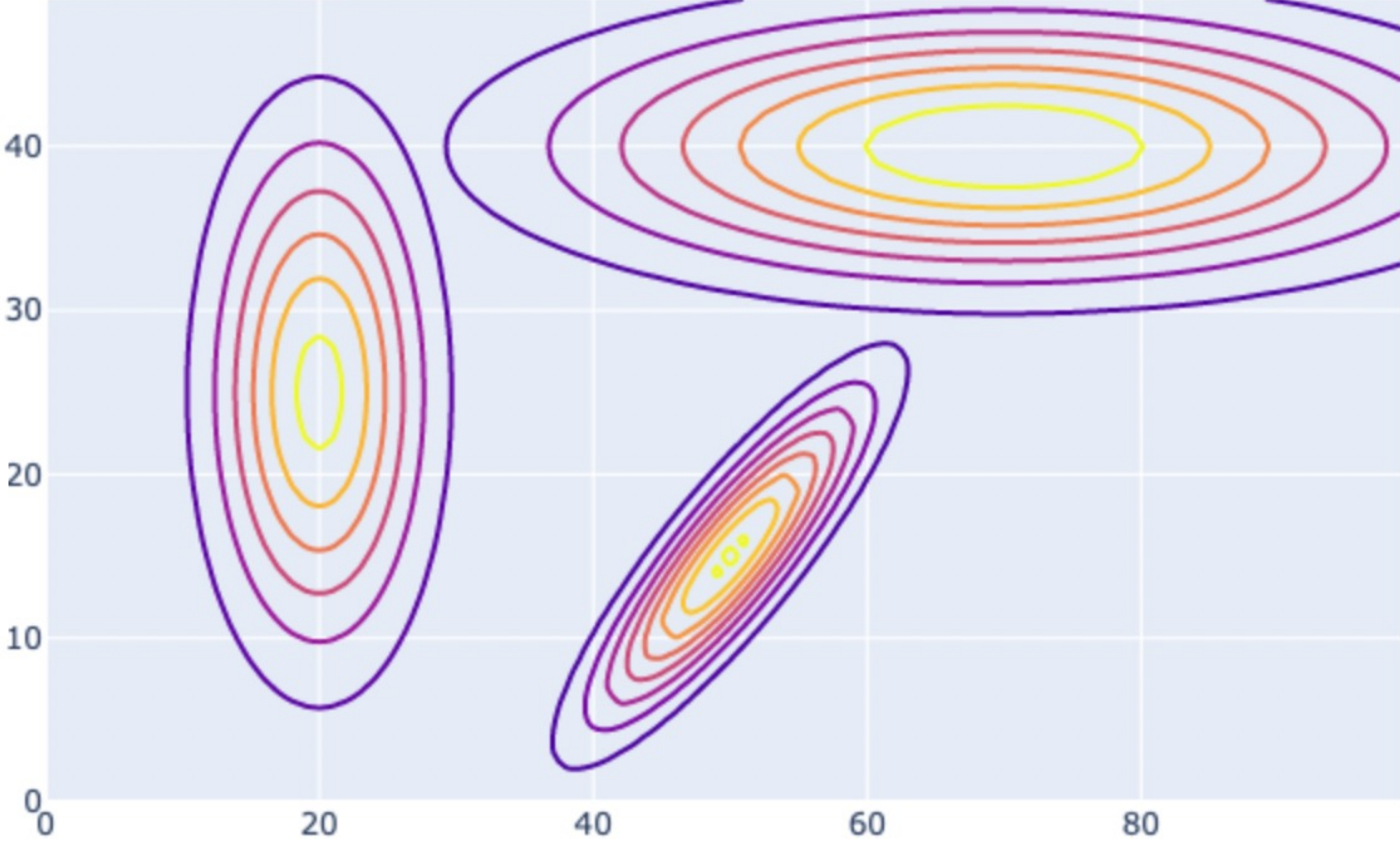
Continuous state

- Part of the 2D plane:

$$x \in \mathcal{D} \subset \mathbb{R}^2$$

- No orientation yet:
  - omni-directional movement

Gaussian Densities



- Remember 1D:

$$\mathcal{N}(x;\mu,\sigma^2) \doteq \frac{1}{k}\exp\{-\frac{1}{2}\frac{\|x-\mu\|^2}{\sigma^2}\}$$

- Just a quadratic inside!
- Rewrite as:

$$\mathcal{E}(x;\mu,\sigma^2) \doteq \frac{1}{2}(x-\mu)\sigma^{-2}(x-\mu)$$

- Generalize to:

$$\mathcal{E}(x;\mu,\Sigma) \doteq \frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)$$

$$\mathcal{N}(x;\mu,\Sigma) \doteq \frac{1}{k}\exp\{-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)\}$$

Multivariate Gaussians, the detail…

- Until now, we have considered Gaussian distributions for scalar random variables.
- For univariate Gaussians, η is a scalar, and it appears in the exponent:

$$f_{\mathrm{H}}(\eta) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(\eta-\mu)^2}{2\sigma^2}}$$

- Note that H is the uppercase version of Greek letter η.
- For a multivariate Gaussian, the random variable is a vector:

$$\eta = \begin{bmatrix} \eta_x \\ \eta_y \end{bmatrix}$$

- How do we put a vector in an exponent??

Multivariate Gaussians

- Let’s take a look at the exponent in the Gaussian distribution:

    1. The term $|x - \mu|$ is the distance from x to the mean.
    2. The term $(x - \mu)^2$ is the squared distance to the mean.
    3. The term $\sigma^{-2}(x - \mu)^2$ is a ***scaled squared distance to the mean***.

➢ This idea – computing a scaled squared distance to the mean – is the key to extending Gaussians to the multivariate case.

➢ Instead of scalar scaling, we can actually apply scaling along different axes, e.g., we can treat motion in the direction of the x-axis as being more uncertain than motion in the direction of the y-axis.

Multivariate Gaussians

First let's define the relevant vectors:

$$\mu = \begin{bmatrix} \mu_x \\ \mu_y \end{bmatrix}, \qquad \vec{x} = \begin{bmatrix} x \\ y \end{bmatrix}$$

<u>**NOTE**</u>:

- For the next few slides, we'll use $\vec{x}$ to denote a vector in $\mathbb{R}^2$.
- There's a possibility of confusion, because most of the time use x to denote a vector $x \in \mathbb{R}^2$.
- For the next derivations, we will use $\mathrm{x}, \mathrm{y}, \in \mathbb{R}$ to denote the scalar coordinates of the point $\vec{x}$.
- Don't lose track of this!

Quadratic Forms

- The squared distance between vectors $\vec{x}$ and μ can be conveniently written as:

$$(\vec{x}-\mu)^T(\vec{x}-\mu) = \begin{bmatrix} x-\mu_x & y-\mu_y \end{bmatrix} \begin{bmatrix} x-\mu_x \\ y-\mu_y \end{bmatrix} = (x-\mu_x)^2 + \left(y-\mu_y\right)^2$$

- Note that this term evaluates to a scalar value!
- The term $(x-\mu_x)^2$ gives the squared distance along the x-axis, and the term $\left(y-\mu_y\right)^2$ gives the squared distance along the y-axis.
- We can scale these simply by multiplying each by a scalar coefficients, say k_x and k_y:

$$k_x(x-\mu_x)^2 + k_y\left(y-\mu_y\right)^2$$

- We can incorporate these scaling values directly into a nice matrix equation:

$$\begin{bmatrix} x-\mu_x & y-\mu_y \end{bmatrix} \begin{bmatrix} k_x & 0 \\ 0 & k_y \end{bmatrix} \begin{bmatrix} x-\mu_x \\ y-\mu_y \end{bmatrix} = k_x(x-\mu_x)^2 + k_y\left(y-\mu_y\right)^2$$

➢ If you understand this, multivariate Gaussians are easy!

Quadratic Forms

- Let's generalize this just a bit

$$\|\vec{x} - \mu\|^2_{\Sigma^{-1}} = (\vec{x} - \mu)^T \Sigma^{-1} (\vec{x} - \mu) = \begin{bmatrix} x - \mu_x & y - \mu_y \end{bmatrix} \begin{bmatrix} a & b \\ b & c \end{bmatrix} \begin{bmatrix} x - \mu_x \\ y - \mu_y \end{bmatrix}$$

- If you multiply this out (a bit tedious), you'll arrive to the general equation for an ellipse:
  - Center of the ellipse is at μ
  - The matrix Σ^{-1}encodes the major and minor axes (direction and length).
  - Check back to your old geometry books to refresh your memory.

Comments:

- We say that the matrix Σ is positive definite if $\vec{x}^T \Sigma \vec{x} > 0$ for all $\vec{x} \neq 0$.
- If a matrix Σ is positive definite, then Σ^{-1} exists, and $\vec{x}^{\mathrm{T}} \Sigma^{-1} \vec{x} = k$ defines an ellipse, for $k > 0$.

Multivariate Gaussians

- We can use this idea to build an n-dimensional Gaussian distribution:

$$f_{\vec{X}}(\vec{x}) = \frac{1}{\sqrt{(2\pi)^n|\Sigma|}} e^{-\frac{1}{2}\|\vec{x}-\mu\|^2_{\Sigma^{-1}}} = \frac{1}{\sqrt{(2\pi)^n|\Sigma|}} e^{-\frac{1}{2}(\vec{x}-\mu)^T\Sigma^{-1}(\vec{x}-\mu)}$$

- As usual, the action is in the exponent; the constant $\sqrt{(2\pi)^n|\Sigma|}$ is merely to scale the pdf so that $\int f_{\vec{X}}(\vec{x})d\vec{x} = 1$.

- ***The value of $f_{\vec{X}}(\vec{x})$ decreases exponentially with the square of the scaled distance $\|\vec{x} - \mu\|_{\Sigma^{-1}}$.***

- The matrix Σ is called the covariance matrix. In the two-dimensional case, it is defined as:

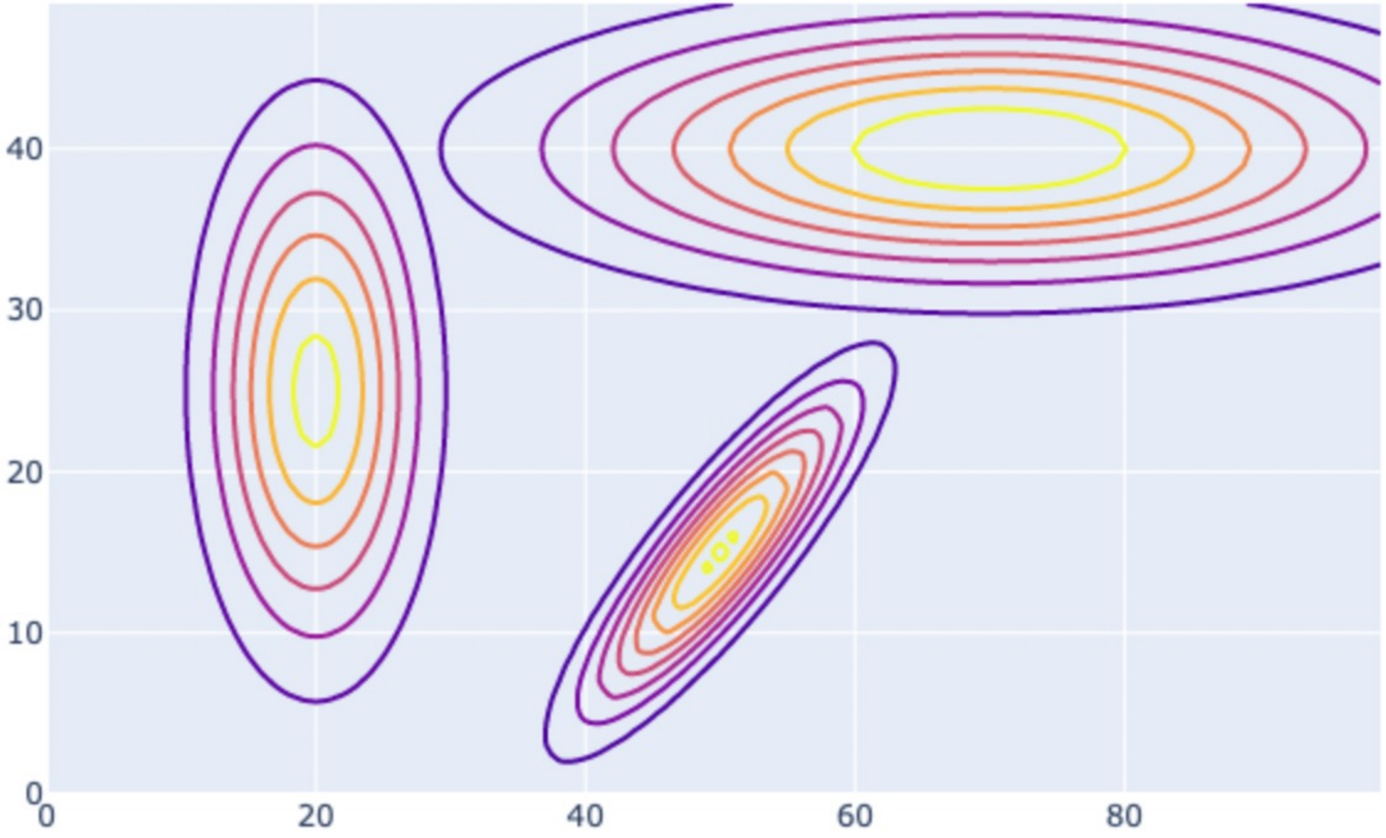
$$\Sigma = \begin{bmatrix} E[(X-\mu_x)^2] & E\left[(X-\mu_x)(Y-\mu_y)\right] \\ E\left[(X-\mu_x)(Y-\mu_y)\right] & E\left[\left(Y-\mu_y\right)^2\right] \end{bmatrix}$$

Multivariate Gaussians in code:

```python
def gaussian(x:np.array, mean=np.zeros((2,)), cov=np.eye(2)):
    """Evaluate multivariate Gaussian at x of shape(m,n), yields (m,) vector."""
    assert x.shape[-1]==2, f"error: x has shape {x.shape}"
    k = math.sqrt(np.linalg.det(2*math.pi*cov))
    e = x - mean
    E = np.sum(0.5 * (e @ np.linalg.inv(cov) * e), axis=-1)
    return np.exp(-E)/k
```

$$\mathcal{N}(x;\mu,\Sigma) \doteq \frac{1}{k}\exp\{-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)\}$$

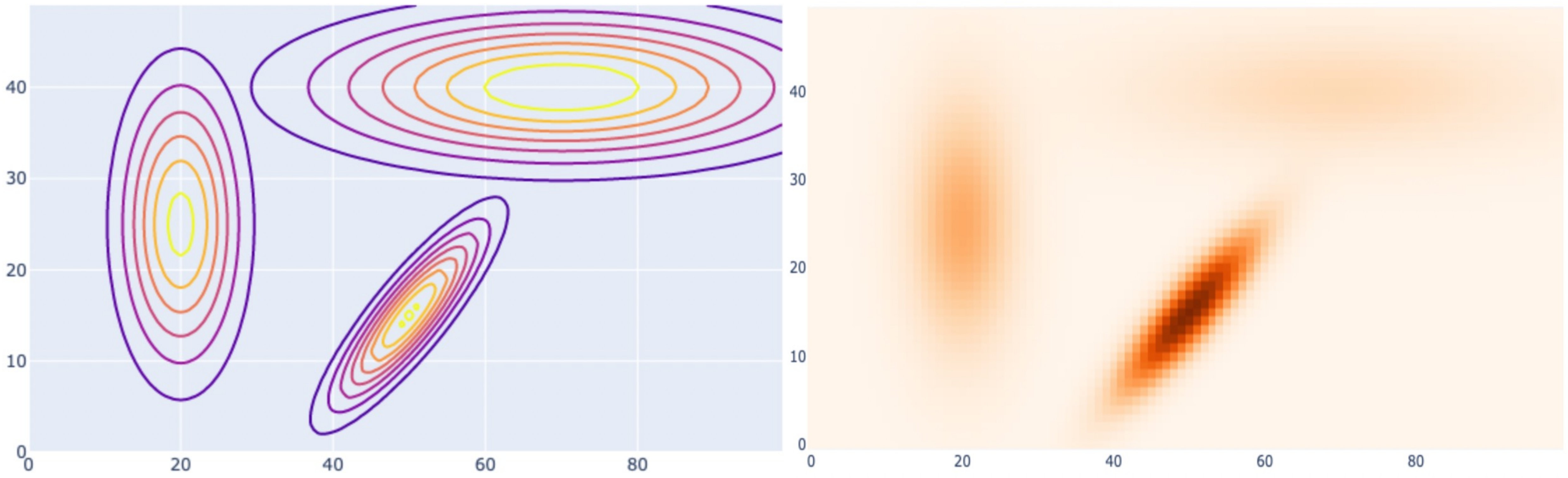
$$k = \sqrt{(2\pi)^n|\Sigma|} = \sqrt{|2\pi\Sigma|}.$$




```python
means = [gtsam.Point2(x,y) for x,y in [(20,25),(70,40),(50,15)]]
covariances = [np.diag([sx**2,sy**2]) for sx,sy in [(5,10),(20,5)]]
covariances.append(np.array([[40,35],[35,40]]))
```

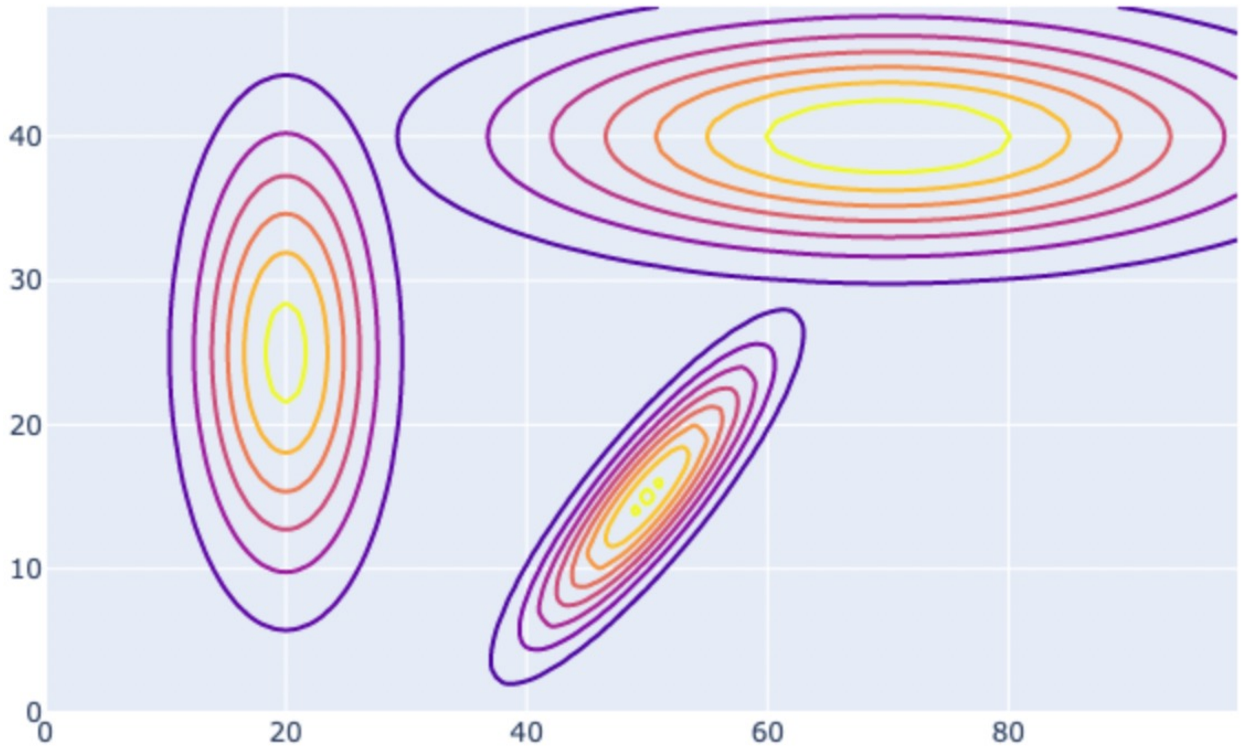
Finite Elements

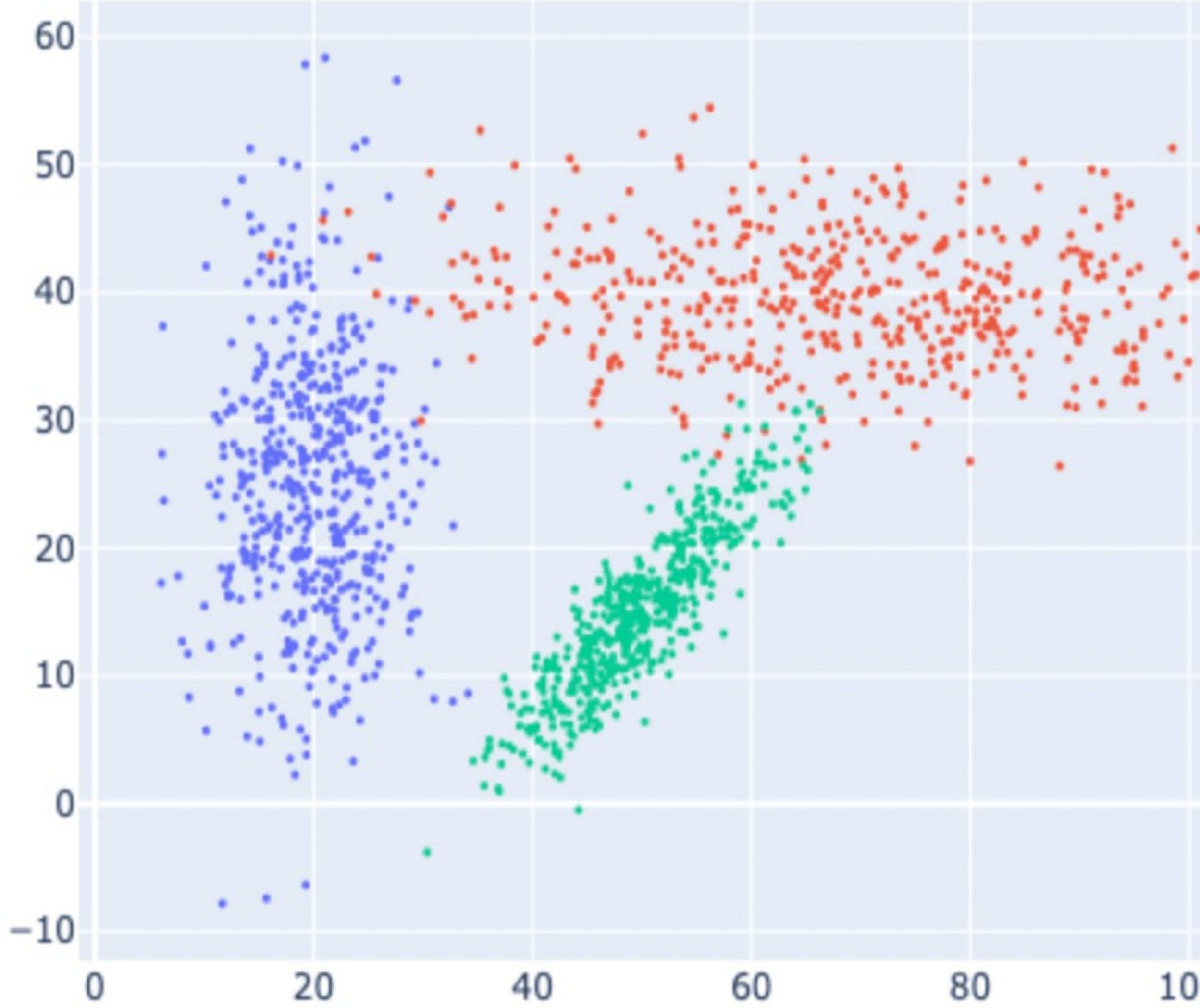
- Just chop up 2D spaces into a 2D grid of finite cells or “elements”
- How does this scale with dimension?



Sampling-based Representation

- Simple, efficient alternative
- Scales with “typical set”





Actions

Until this point, we have ignored the issues related to robot motion:

- The trash sorting robot had built-in sorting actions.
- The vacuuming robot had built-in motion primitives to navigate from room to room.
- We modeled uncertainty, but we really didn't do any work to develop these models, which really should be related to reliability of the robot's actions/motions.

In this chapter, we'll take a first look at robot motion:

- Rolling wheels induce motion of a mobile platform.
- Uncertainty in the effects of actions is modeled directly in terms of the robot's motion.

➢ We'll start with the kinematics of omni wheels…

Omni Wheels

Typical wheel:

- Rolls forward (the driving direction) without slipping
- Cannot slide perpendicular to the steering direction
- Wheel velocity is therefore always in the driving direction
- The inability to slide is a nonholonomic constraint

Omni wheel:

- Rolls forward (the driving direction) without slipping
- Can slide perpendicular to the steering direction
- ***Wheel velocity not constrained to be in the driving direction!***
- Sliding is passive, just the right amount to accommodate the wheel velocity.

Typical Omni-Wheel robot

The reason for three wheels:

- Steering directions of the three wheels positively spans the plane, plus stability.
- Can move in any direction instantaneously by an appropriate choice of wheel speed.

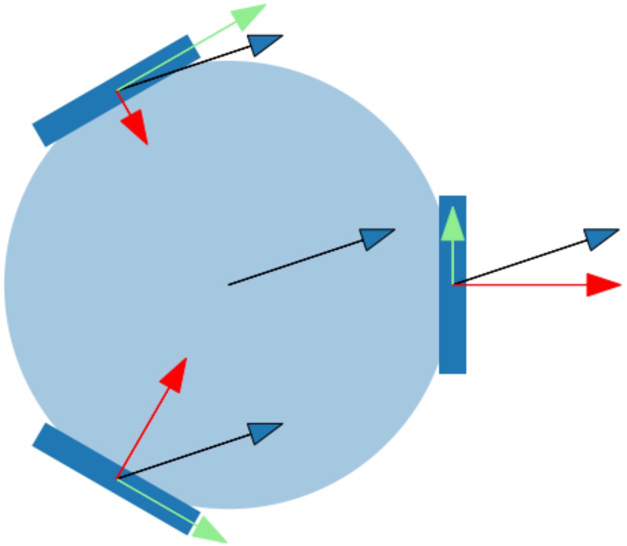
Wheel Kinematics

- In this chapter, we consider only pure translations (we'll consider orientation and rotation in a later chapter).
- If the robot moves with a pure translational velocity, then every point on the robot moves with the same velocity.
- Define the translational velocity of the robot to be

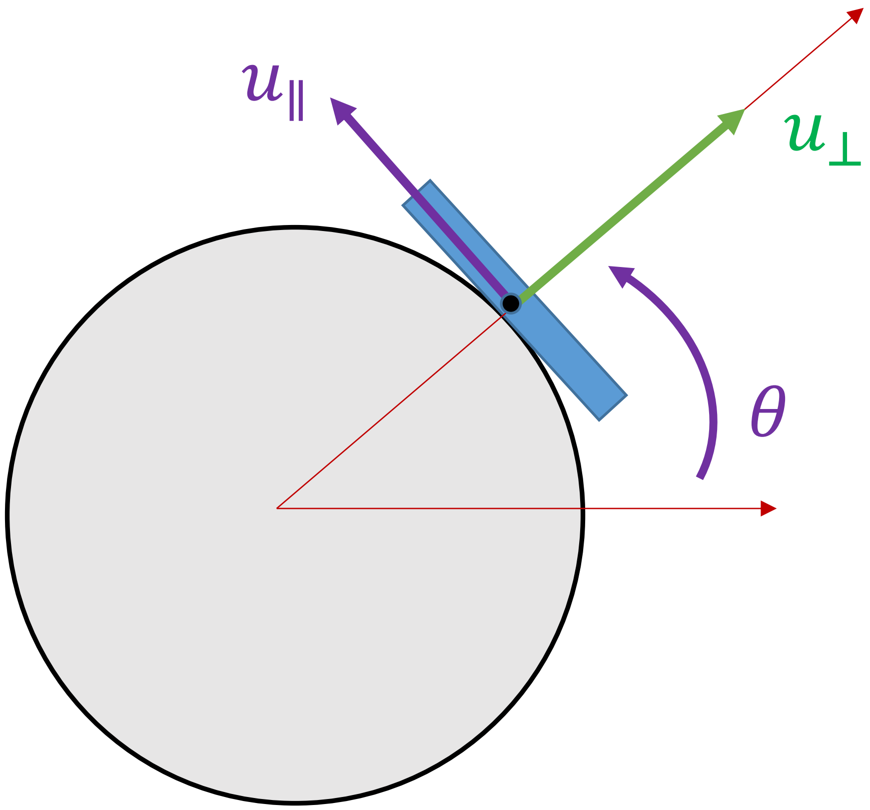
$$v = \begin{bmatrix} v_x \\ v_y \end{bmatrix}$$

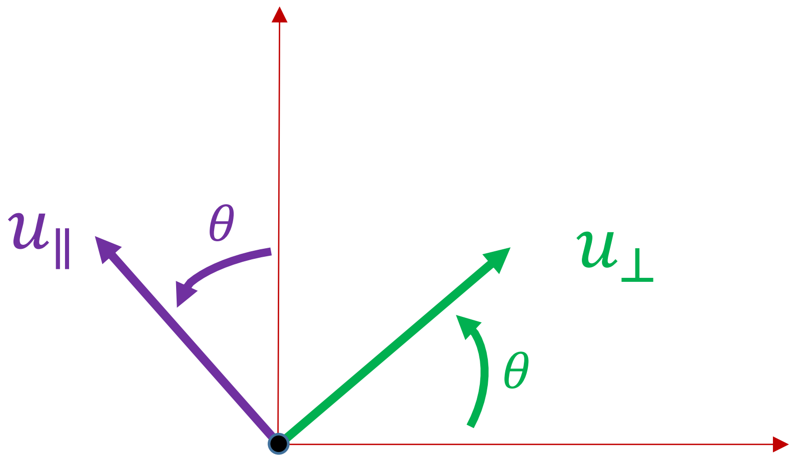
The velocity of each wheel can be decomposed into two components: $v_{\|}$ and $v_{\perp}$.

- $v_{\|}$ is the component of wheel velocity that is parallel to the driving direction.
- $v_{\perp}$ is the component of the wheel velocity that is perpendicular to the driving direction.



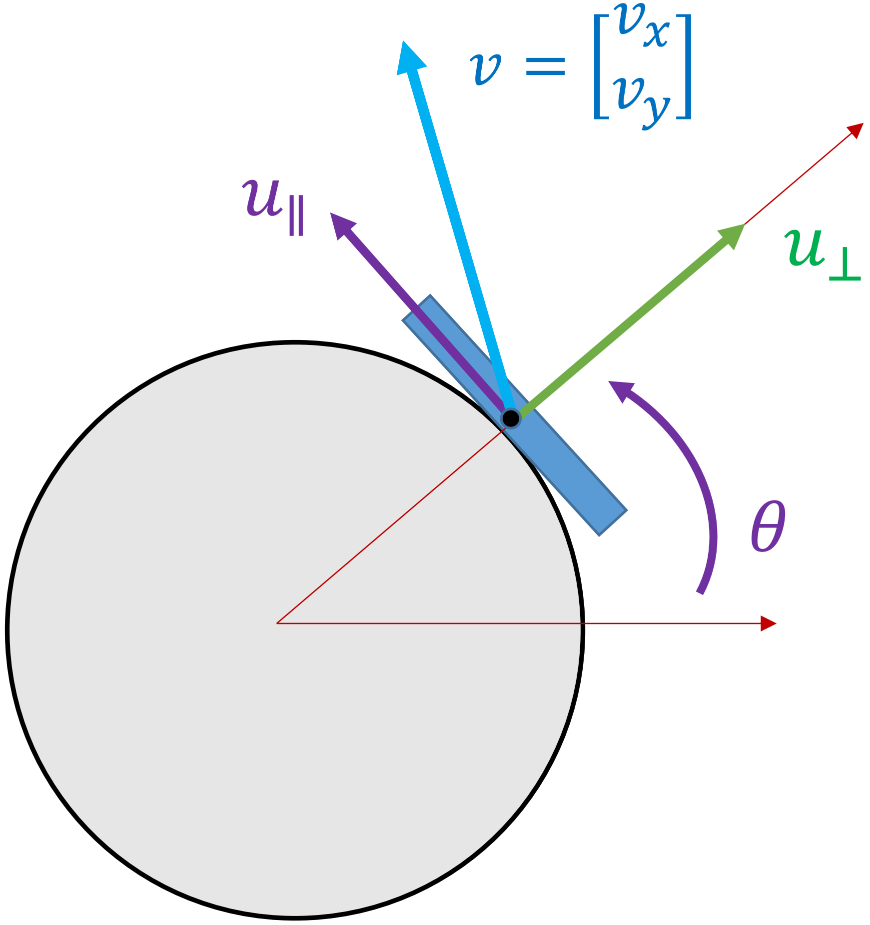
Decomposing Robot Velocity





$$u_{\perp} = \begin{bmatrix} \cos\theta \\ \sin\theta \end{bmatrix} \qquad u_{\parallel} = \begin{bmatrix} -\sin\theta \\ \cos\theta \end{bmatrix}$$

Decomposing Robot Velocity



- We can now decompose v into the components parallel to and perpendicular to the steering direction.
- This is done by projecting v onto $u_\parallel$ and $u_\perp$

$$v = (v \cdot u_\parallel)u_\parallel + (v \cdot u_\perp)u_\perp$$

which can be written as

$$v = v_\parallel u_\parallel + v_\perp u_\perp$$

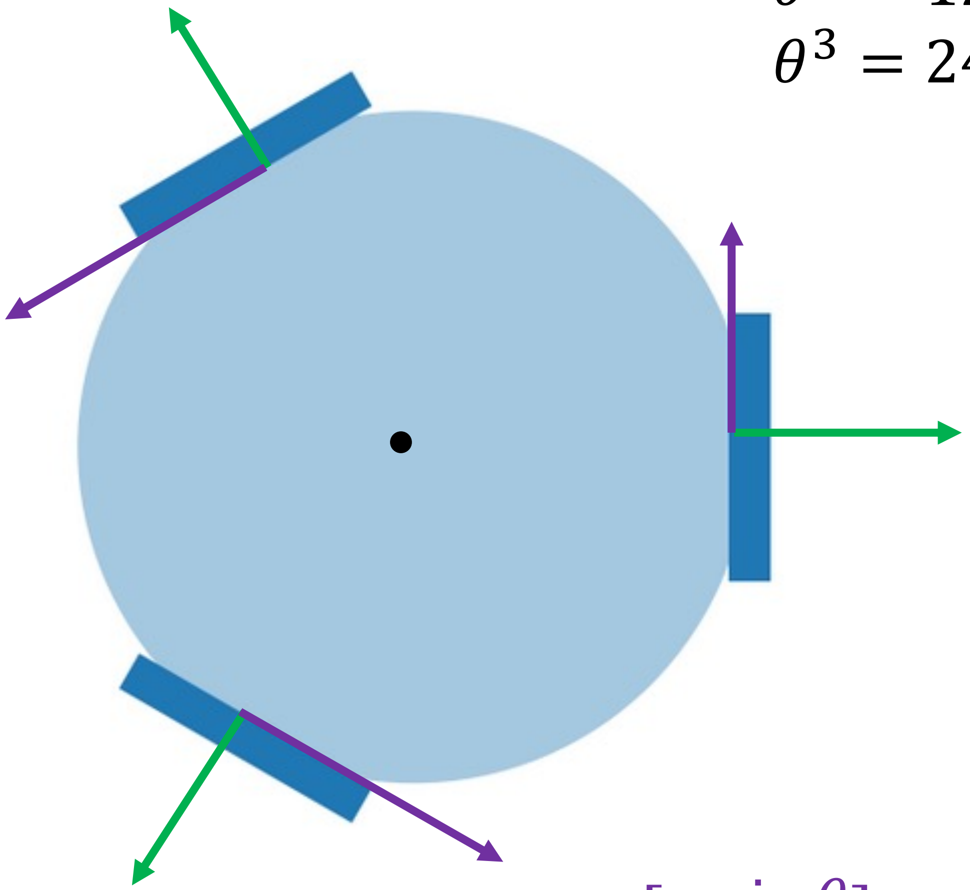
where

$$v_\parallel = -v_x \sin\theta + v_y \cos\theta$$
$$v_\perp = v_x \cos\theta + v_y \sin\theta$$

$$u_\perp = \begin{bmatrix} \cos\theta \\ \sin\theta \end{bmatrix} \qquad u_\parallel = \begin{bmatrix} -\sin\theta \\ \cos\theta \end{bmatrix}$$

Note that $v_\parallel$ and $v_\perp$ are scalars!

Three Uniformly Positioned Wheels



$$\theta^1 = 0$$
$$\theta^2 = 120$$
$$\theta^3 = 240$$

$$u_\parallel = \begin{bmatrix} -\sin\theta \\ \cos\theta \end{bmatrix}$$

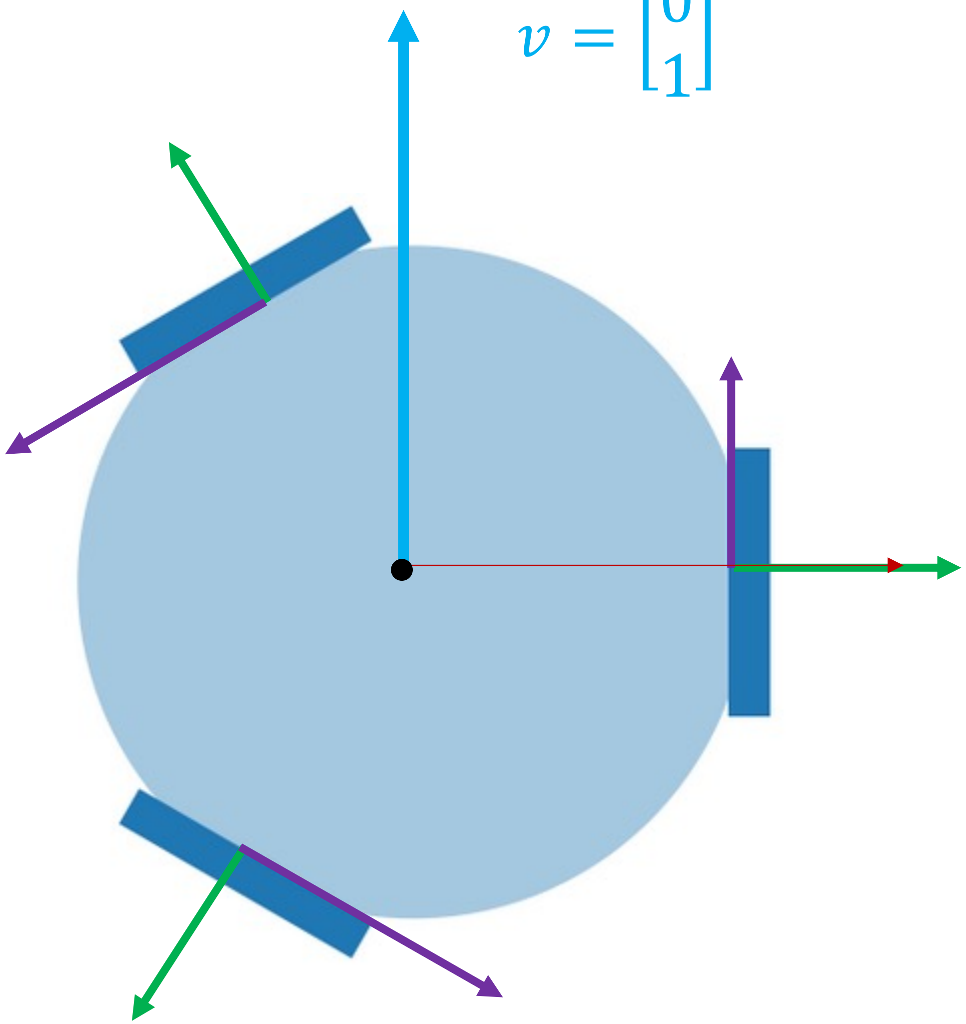
$$u_\parallel^1 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

$$u_\parallel^2 = \begin{bmatrix} -0.8660 \\ -0.5 \end{bmatrix}$$

$$u_\parallel^3 = \begin{bmatrix} 0.866 \\ -0.5 \end{bmatrix}$$

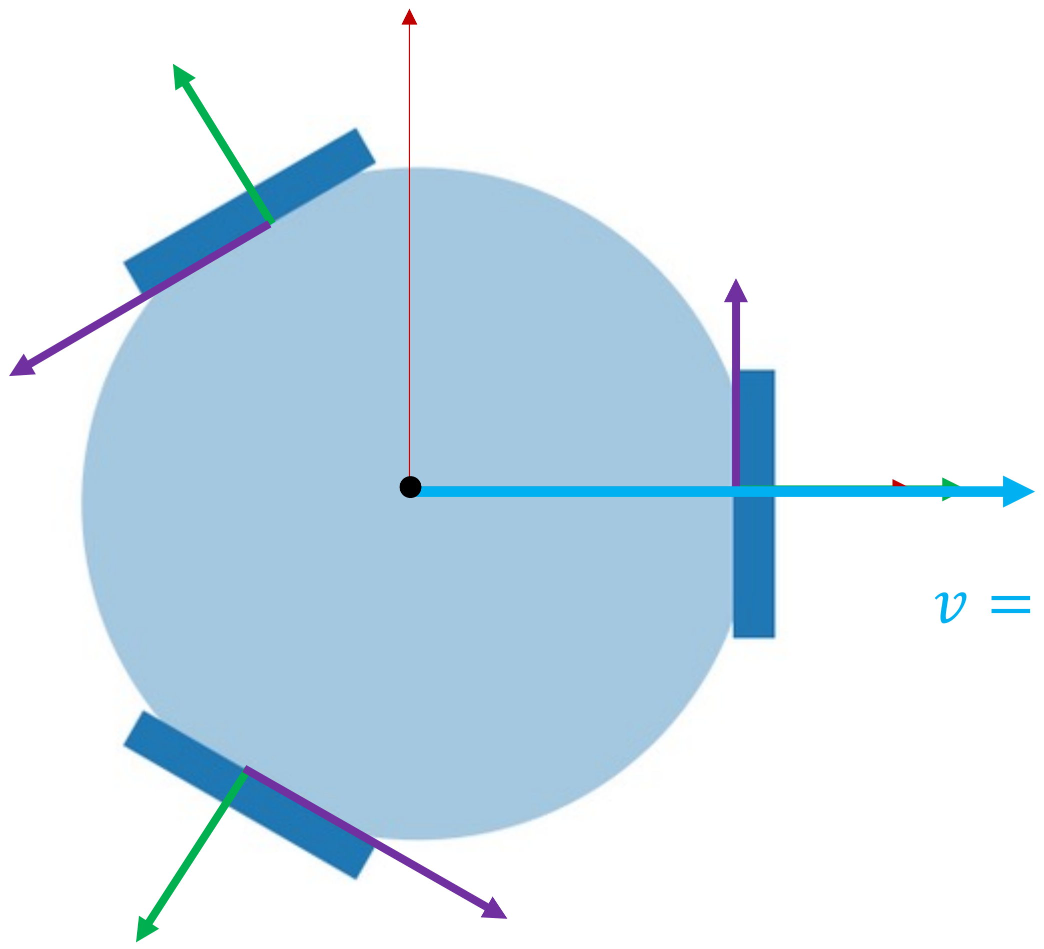
$$\begin{bmatrix} v_\parallel^1 \\ v_\parallel^2 \\ v_\parallel^3 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -0.866 & -0.5 \\ 0.866 & -0.5 \end{bmatrix} \begin{bmatrix} v_x \\ v_y \end{bmatrix}$$

Example



$$\begin{bmatrix} v_{\parallel}^1 \\ v_{\parallel}^2 \\ v_{\parallel}^3 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -0.866 & -0.5 \\ 0.866 & -0.5 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ -0.5 \\ -0.5 \end{bmatrix}$$

Example



$$v = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

$$\begin{bmatrix} v_{\parallel}^{1} \\ v_{\parallel}^{2} \\ v_{\parallel}^{3} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -0.866 & -0.5 \\ 0.866 & -0.5 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ -0.866 \\ 0.866 \end{bmatrix}$$

Wheel Jacobian

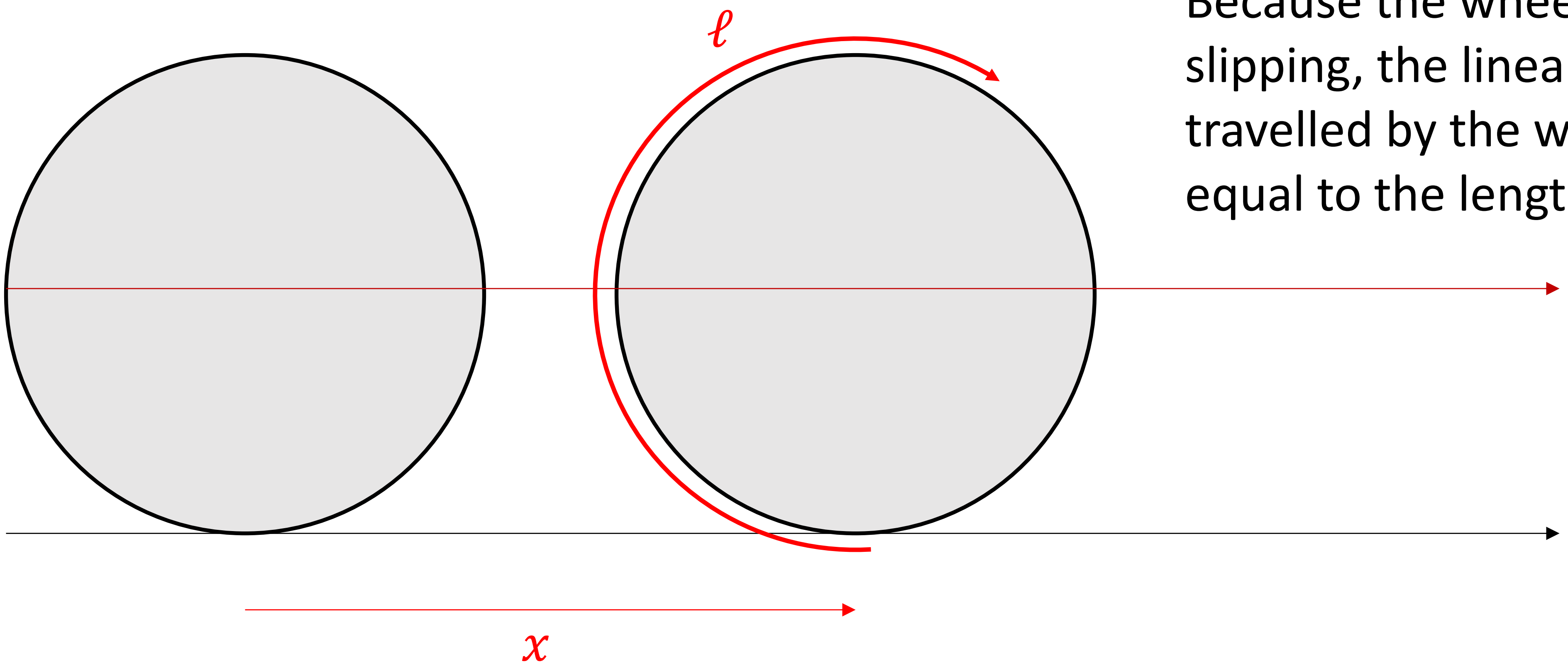
- A ***Jacobian*** matrix maps velocities in one coordinate system to velocities in another coordinate system.
- For our case, we want to map the velocity of the robot v to wheel rotation, specified as angular velocities ω^i for $i = 1,2,3$.
- The desired relationship is given by:

$$\begin{bmatrix} \omega^1 \\ \omega^2 \\ \omega^3 \end{bmatrix} = J \begin{bmatrix} v_x \\ v_y \end{bmatrix}$$

- We'll need to relate rotation of the wheel to translation in the driving direction.

Rolling Without Slipping

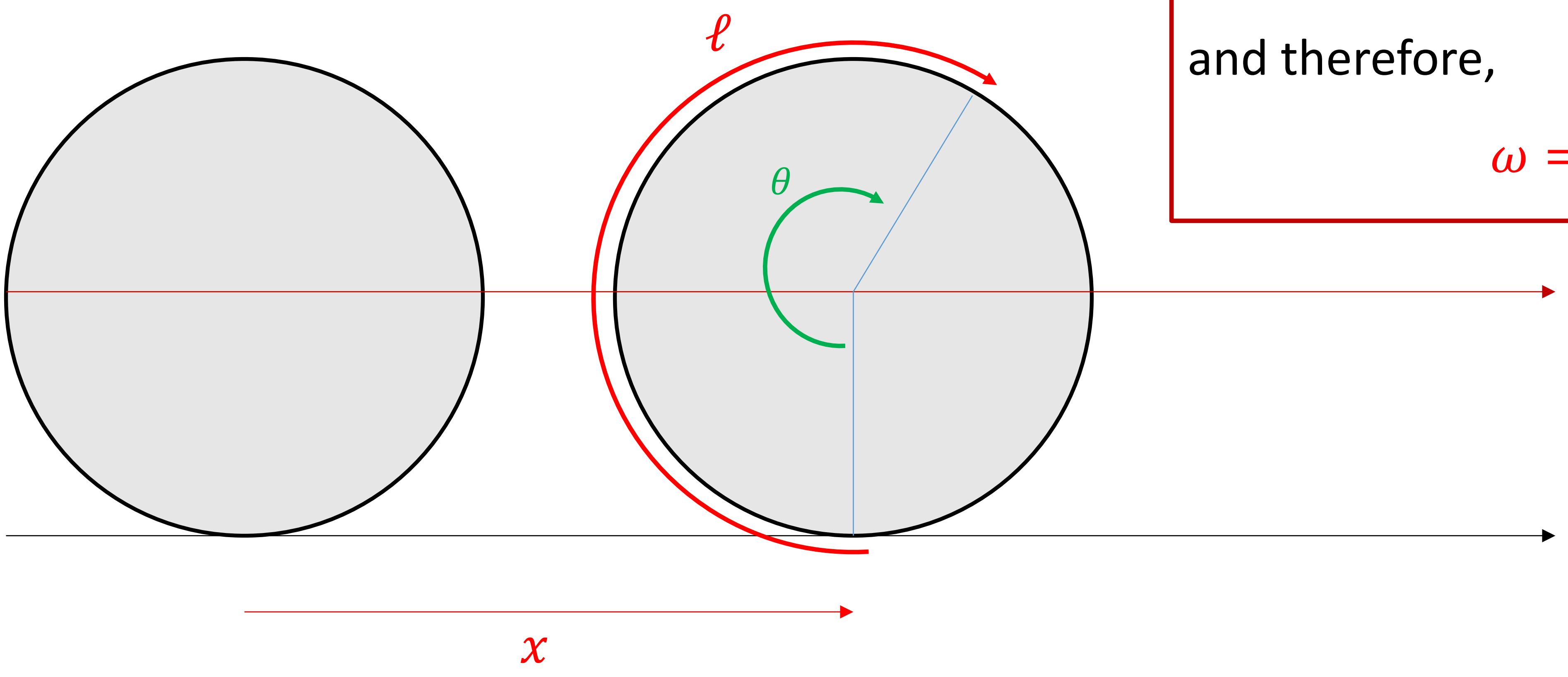
Suppose a wheel rolls without slipping a linear distance x.



Because the wheel rolls without slipping, the linear distance x travelled by the wheel center is equal to the length of the section ℓ.

Rolling Without Slipping

Using basic geometry, we know that $x = \ell = r\theta$.



Differentiating both sides, we obtain

$$\mathrm{v} = \frac{\mathrm{d}}{\mathrm{dt}}x = \ell = r\frac{d}{dt}\theta = r\omega$$

and therefore,

$$\omega = \frac{1}{r}v$$

Mapping Robot Velocity to Wheel Rotation

Combining these results, we obtain our final, Jacobian relationship:

$$v_{\parallel}^{i} = -v_x \sin\theta^i + v_y \cos\theta^i$$
$$v_{\perp}^{i} = v_x \cos\theta^i + v_y \sin\theta^i$$

$$\omega^i = \frac{1}{r} v_{\parallel}^{i}$$

$$\begin{bmatrix} \omega^1 \\ \omega^2 \\ \omega^3 \end{bmatrix} = \frac{1}{r} \underbrace{\begin{bmatrix} -\sin\theta^1 & \cos\theta^1 \\ -\sin\theta^2 & \cos\theta^2 \\ -\sin\theta^3 & \cos\theta^3 \end{bmatrix}}_{\text{This is the Jacobian matrix, } J} \begin{bmatrix} v_x \\ v_y \end{bmatrix}$$

$$\begin{bmatrix} \omega^1 \\ \omega^2 \\ \omega^3 \end{bmatrix} = \frac{1}{r} \begin{bmatrix} 0 & 1 \\ -0.866 & -0.5 \\ 0.866 & -0.5 \end{bmatrix} \begin{bmatrix} v_x \\ v_y \end{bmatrix}$$

Discrete Time Motion Model

- The control input for our robot is a linear velocity v (which is converted to angular velocities for each wheel).

- We could model the motion of the robot using a differential equation: $\dot{\boldsymbol{x}} = f(\boldsymbol{x}, \boldsymbol{u})$

$$\begin{bmatrix} \dot{x} \\ \dot{y} \end{bmatrix} = \begin{bmatrix} v_x \\ v_y \end{bmatrix}$$

- It's much simpler to use a discrete time model for the position of the robot:

$$\begin{bmatrix} x_{t+1} \\ y_{t+1} \end{bmatrix} = \begin{bmatrix} x_t + v_x \Delta T \\ y_t + v_y \Delta T \end{bmatrix} = \begin{bmatrix} x_t + u_x \\ y_t + u_y \end{bmatrix}$$

- If the motion of the robot happened to be deterministic and error-free, this would be all we need.
- We'll assume that the motion model is stochastic, and show how to model uncertainty using continuous probability density functions.

Limitations of our Model

The model we developed for omni-wheeled robots made several simplifications to what we might find in real applications:

- We conveniently aligned the robot's coordinate system to a global world coordinate frame. Specifying the angle θ^i was simple, because it was specified in a coordinate frame that was fixed w.r.t. to the robot.
- Real robots sometimes rotate. We could accomplish this with the exact same robot by adding a rotational component to the robot velocity (i.e., robot angular velocity):

$$v = \begin{bmatrix} \omega^{robot} & v_x & v_y \end{bmatrix}^T$$

- If the robot rotates, then we'll need to represent its orientation w.r.t. the global coordinate frame, since the steering directions of the wheels will change if the robot rotates.

Mecanum Wheels

- We can make the wheels a bit more interesting by changing the orientation of the “roller” wheels that allow sliding – ***Mecanum Wheels***.
- The math is (only) slightly more complex, but we won’t go further in this course.