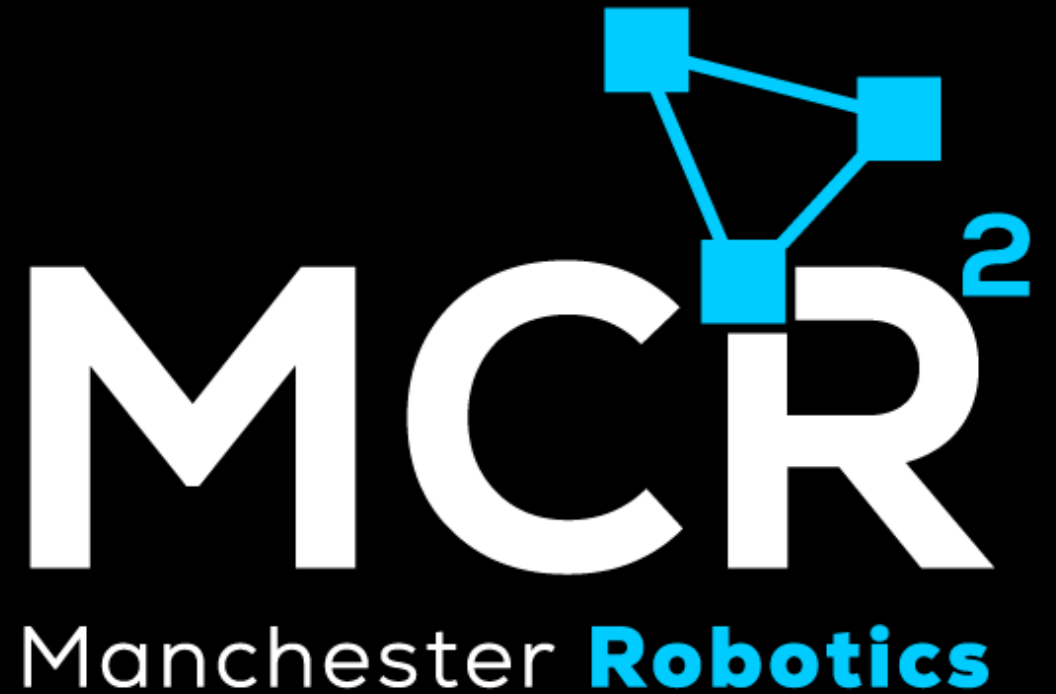


*{Learn, Create, Innovate};*

# Mobile robots

*Localisation in presence  
of uncertainties*



# Motion-based Localisation (Dead Reckoning)

$$\begin{bmatrix} s_{x,k} \\ s_{y,k} \\ s_{\theta,k} \end{bmatrix} = \begin{bmatrix} s_{x,k-1} \\ s_{y,k-1} \\ s_{\theta,k-1} \end{bmatrix} + \begin{bmatrix} \Delta d \cos(s_{\theta,k-1}) \\ \Delta d \sin(s_{\theta,k-1}) \\ \Delta \theta \end{bmatrix}$$





# Motion-based Localisation (Dead Reckoning)



The pose estimation of a mobile robot is always associated with some uncertainty with respect to its state parameters.

From a geometric point of view, the error in differential-drive robots is classified into three groups:

- **Range error:** it is associated with the computation of  $\Delta d$  over time.
- **Turn error:** it is associated with the computation of  $\Delta \theta$  over time.
- **Drift error:** it is associated with the difference between the angular speed of the wheels and it affects the error in the angular rotation of the robot.



# Motion-based Localisation (Dead Reckoning)

Kinematic model for a differential robot model

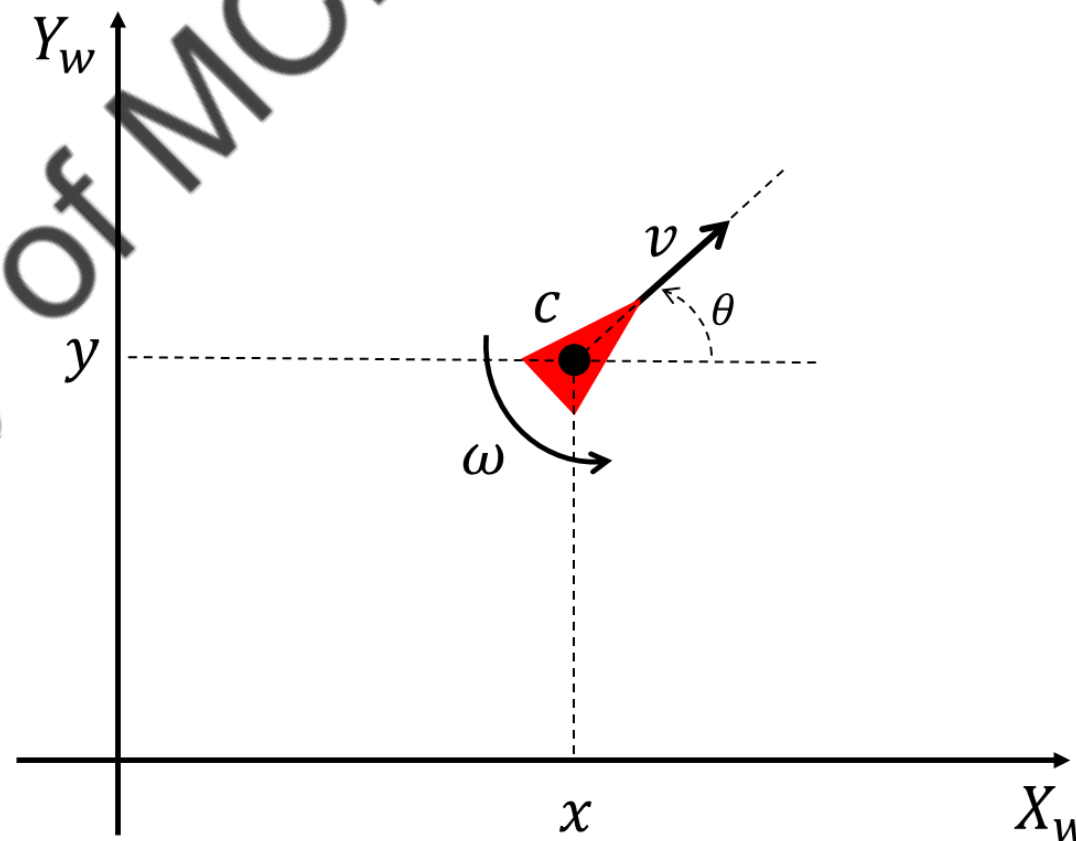
$$\frac{d}{dt} \begin{bmatrix} s_x \\ s_y \\ s_\theta \end{bmatrix} = \begin{bmatrix} \cos(s_\theta) & 0 \\ \sin(s_\theta) & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix}$$

The robot pose

$$\mathbf{s}_k = [s_x \quad s_y \quad s_\theta]^T$$

The robot inputs

$$\mathbf{u}_k = [v \quad \omega]^T$$



# Motion-based Localisation (Dead Reckoning)

- If  $\Delta t$  is the sampling time, then it is possible to compute the incremental linear and angular displacements,  $\Delta d$  and  $\Delta \theta$ , as follows:

$$\Delta d = v \cdot \Delta t$$

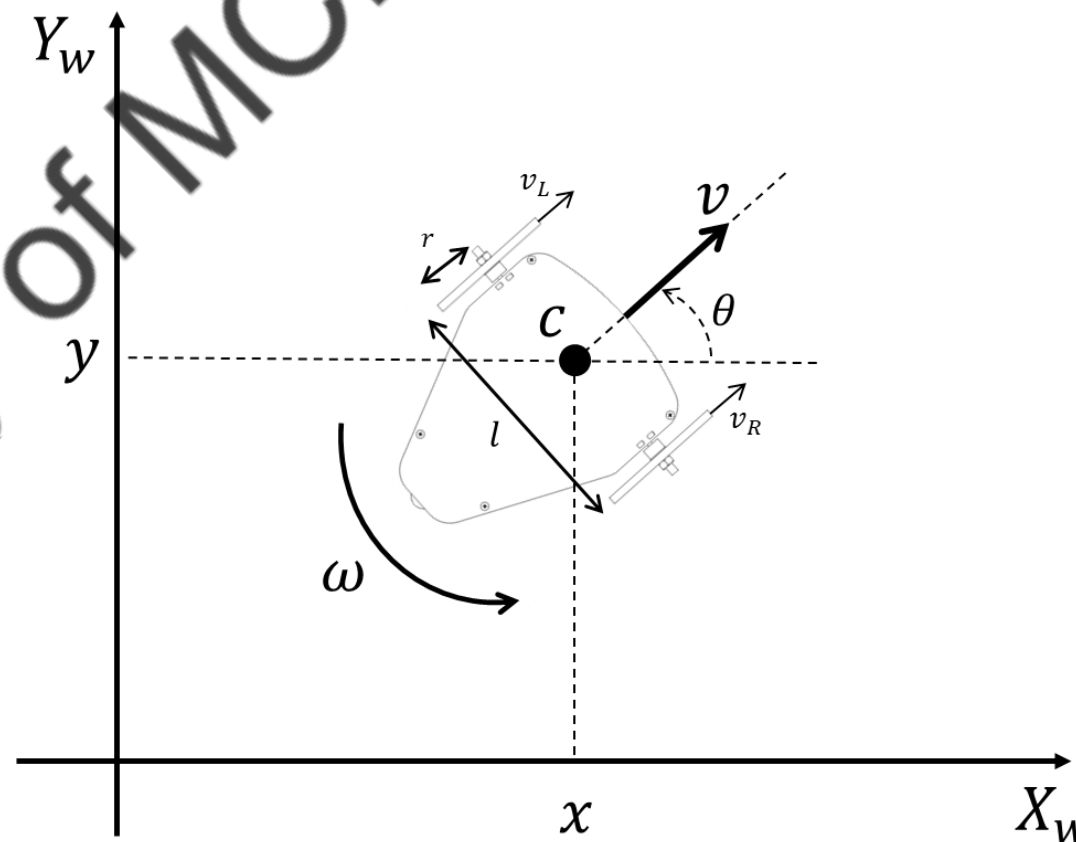
$$\Delta \theta = \omega \cdot \Delta t$$

$$\begin{bmatrix} \Delta d \\ \Delta \theta \end{bmatrix} = \begin{bmatrix} 1/2 & 1/2 \\ 1/l & -1/l \end{bmatrix} \begin{bmatrix} \Delta d_r \\ \Delta d_l \end{bmatrix}$$

To compute the pose of the robot at any given time step, the kinematic model must be numerically integrated.

This approximation follows the **Markov assumption** where the current robot pose depends only on the previous pose and the input velocities.

$$\begin{bmatrix} s_{x,k} \\ s_{y,k} \\ s_{\theta,k} \end{bmatrix} = \begin{bmatrix} s_{x,k-1} \\ s_{y,k-1} \\ s_{\theta,k-1} \end{bmatrix} + \begin{bmatrix} \Delta d \cos(s_{\theta,k-1}) \\ \Delta d \sin(s_{\theta,k-1}) \\ \Delta \theta \end{bmatrix}$$





# Motion-based Localisation (Dead Reckoning)



So, in order to make a realistic simulation, we must include the uncertainties in the state-space model.

$$\mathbf{s}_k = \mathbf{h}(\mathbf{s}_{k-1}, \mathbf{u}_k) + \mathbf{q}_k$$

- what is  $\mathbf{q}_k$ ???



# Motion-based Localisation (Dead Reckoning)



So, in order to make a realistic simulation, we must include the uncertainties in the state-space model.

$$\mathbf{s}_k = \mathbf{h}(\mathbf{s}_{k-1}, \mathbf{u}_k) + \mathbf{q}_k$$

- what is  $\mathbf{q}_k$ ???

Probabilities, a suitable mathematical tool to represent the uncertainties???



# Fundamentals in probabilities

---



- What is probability?

Property of MCR2



# History of probability

The theory of probability has always been associated with gambling and many most accessible examples still come from that activity.

Although gambling dates back thousands of years, the birth of modern probability is considered to be a 1654 letter from the Flemish aristocrat and notorious gambler Chevalier de Méré to the mathematician and philosopher Blaise Pascal. In essence the letter said:

“I used to bet even money that I would get at least one 6 in four rolls of a fair die. The probability of this is 4 times the probability of getting a 6 in a single die, i.e.,  $\frac{4}{6} = \frac{2}{3}$ ; clearly, I had an advantage and indeed I was making money. Now I bet even money that within 24 rolls of two dice I get at least one double 6. This has the same advantage ( $\frac{24}{6^2} = \frac{2}{3}$ ), but now I am losing money. Why?”

As Pascal discussed in his correspondence with Pierre de Fermat, de Méré’s reasoning was faulty; after all, if the number of rolls were 7 in the first game, the logic would give the nonsensical probability  $\frac{7}{6}$ .



# Random variables

---



- Discrete random variables
- Continuous random variables

Property of MCR2





# Discrete random variables

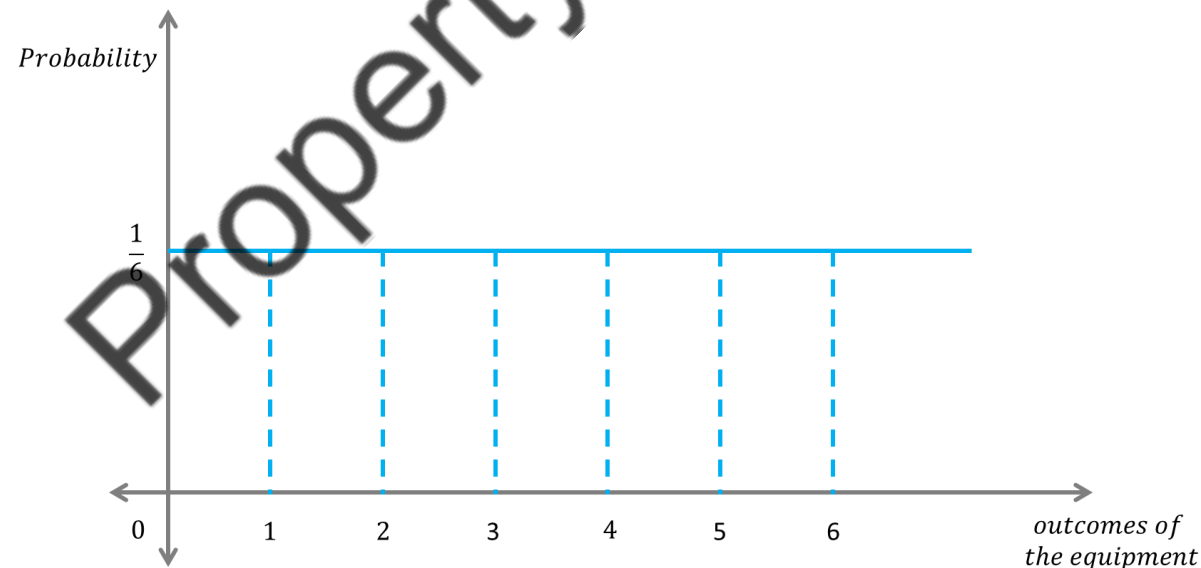


A discrete random variable takes only discrete values.

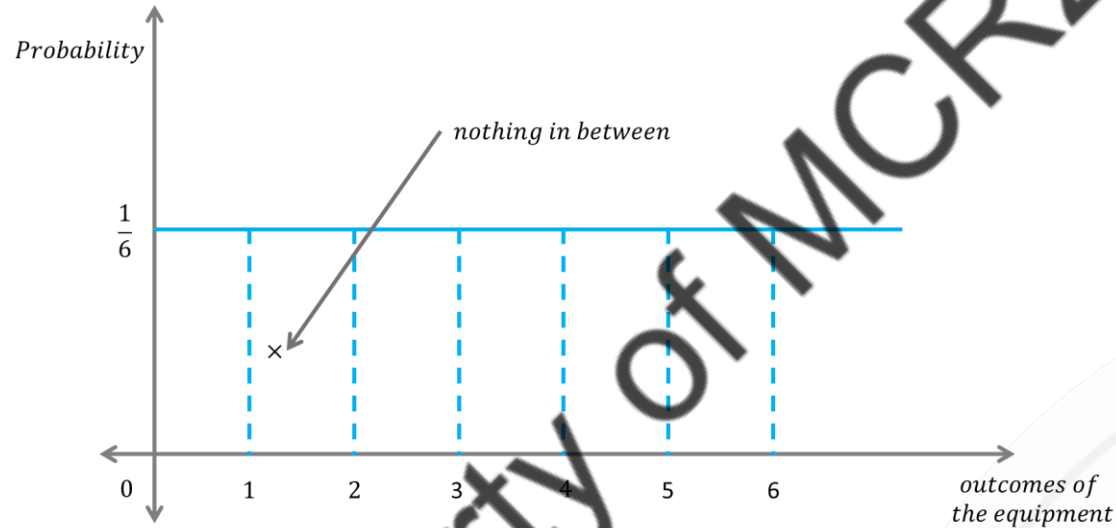
Example. Let's do the following experiment:

We throw (roll) a dice.

The results of this experiment can be 6 different outcomes, i.e., 1, 2, 3, 4, 5, and 6.



# Discrete random variables



The probability of getting one value is the outcome divided by the total number of possibilities. So, probability of getting '1' is  $\frac{1}{6}$ .

Which is the probability to get all values???



# Discrete random variables



It is 1:

$$\frac{1}{6} + \frac{1}{6} + \frac{1}{6} + \frac{1}{6} + \frac{1}{6} + \frac{1}{6} = \frac{6}{6} = 1$$

So, if we roll a dice, the probability to obtain any of the six outcomes is 1. This is because we don't have any other possible outcomes.

Property of MCR2



# Discrete random variables

The **mean** of a discrete random variable  $X$  is its weighted average. Each value of  $X$  is weighted by its probability. To find the mean of  $X$ , multiply each value of  $X$  by its probability, then add all the products. The mean of a random variable is called the **expected value** of  $X$ .

The expected value of  $X$ :

$$E[X] = \mu_X = \sum_{i=1}^{\infty} P_X(x_i) x_i$$


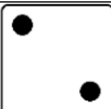




The variance of  $X$ :

$$\sigma^2 = \text{Var}(X) = \sum_{i=1}^{\infty} (x_i - \mu)^2 P_X(x_i)$$

# Discrete random variables

- Example for the die roll:

**Expected Value Of A 6-Sided Die Roll**

| <u>Visual</u><br>(Die Face)   | <u>Value</u><br>X | <u>Probability</u><br>P(X) | <u>Product</u><br>X*P(X) |
|---|-------------------|----------------------------|--------------------------|
|    | 1                 | 1/6                        | $1 * 1/6$<br>$= 1/6$     |
|    | 2                 | 1/6                        | $2 * 1/6$<br>$= 2/6$     |
|    | 3                 | 1/6                        | $3 * 1/6$<br>$= 3/6$     |
|    | 4                 | 1/6                        | $4 * 1/6$<br>$= 4/6$     |
|   | 5                 | 1/6                        | $5 * 1/6$<br>$= 5/6$     |
|  | 6                 | 1/6                        | $6 * 1/6$<br>$= 6/6$     |

**Expected Value (sum of last column)**

$$= 1/6 + 2/6 + 3/6 + 4/6 + 5/6 + 6/6$$

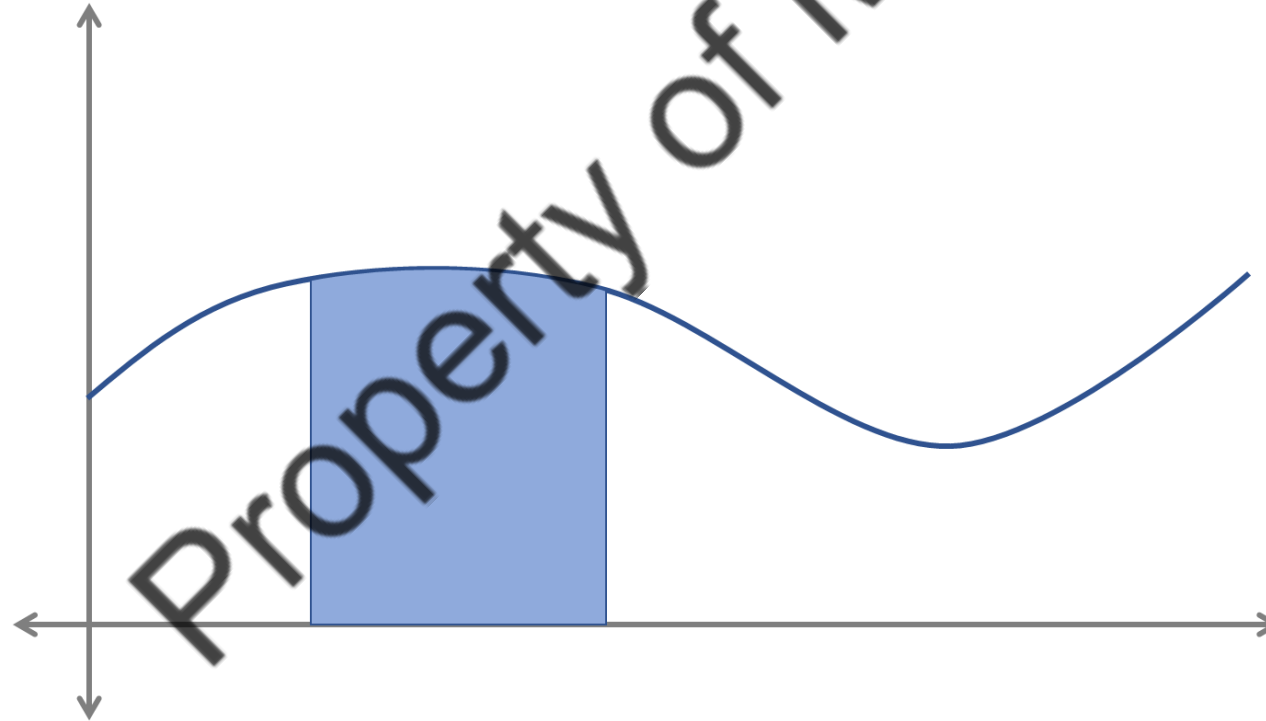
$$= 21/6 \text{ OR } 3.5$$



# Continuous random variables



In this case we have infinity values, infinite outcomes. So, continuous probability distributions.







# Continuous random variables



In this case, we cannot sum up the probabilities as we did in the case of Discrete Random Variables.

In discrete random variables:  $P(X=1) = 1/6$ ,  $P(X=1 \text{ OR } X=3) = 1/6 + 1/6$

We cannot sum up infinite numbers. So, we have to integrate. If we integrate, then we obtain the Area.

So, in case of continuous random variables, if we want to compute the probability of the area out of all possibilities, we have to integrate the respective area.

Property of MCR²





# Distributions for Continuous random variables

---



There are many distributions for continuous random variables. We'll study only two:

- Uniform distribution
- Gaussian distribution

Property of MCR2





# Uniform Distribution



- Uniform distribution means “all the same”

Experiment: Receiving emails

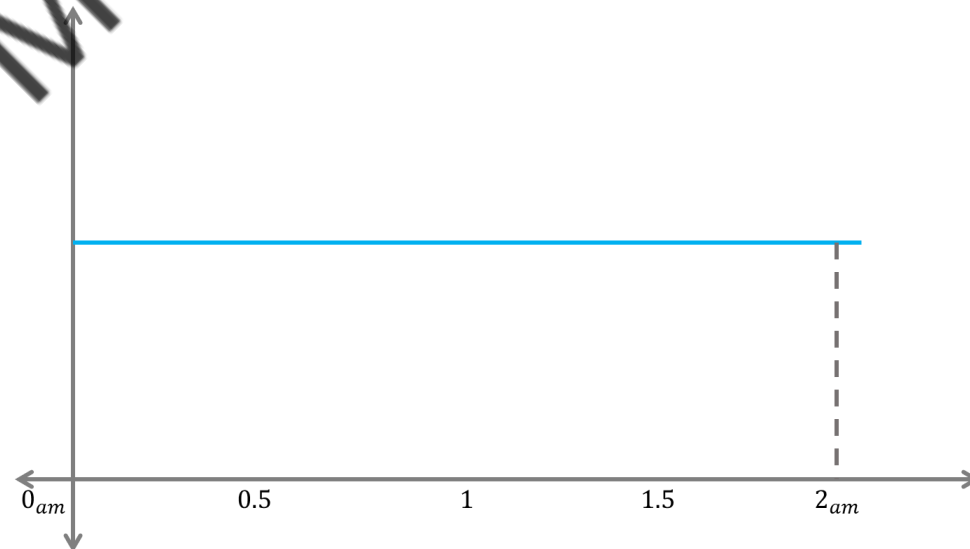
Let  $X$  be a continuous random variable which is the time interval until we receive the next email, this can be  $(0, \infty)$ .

Property of MCR2

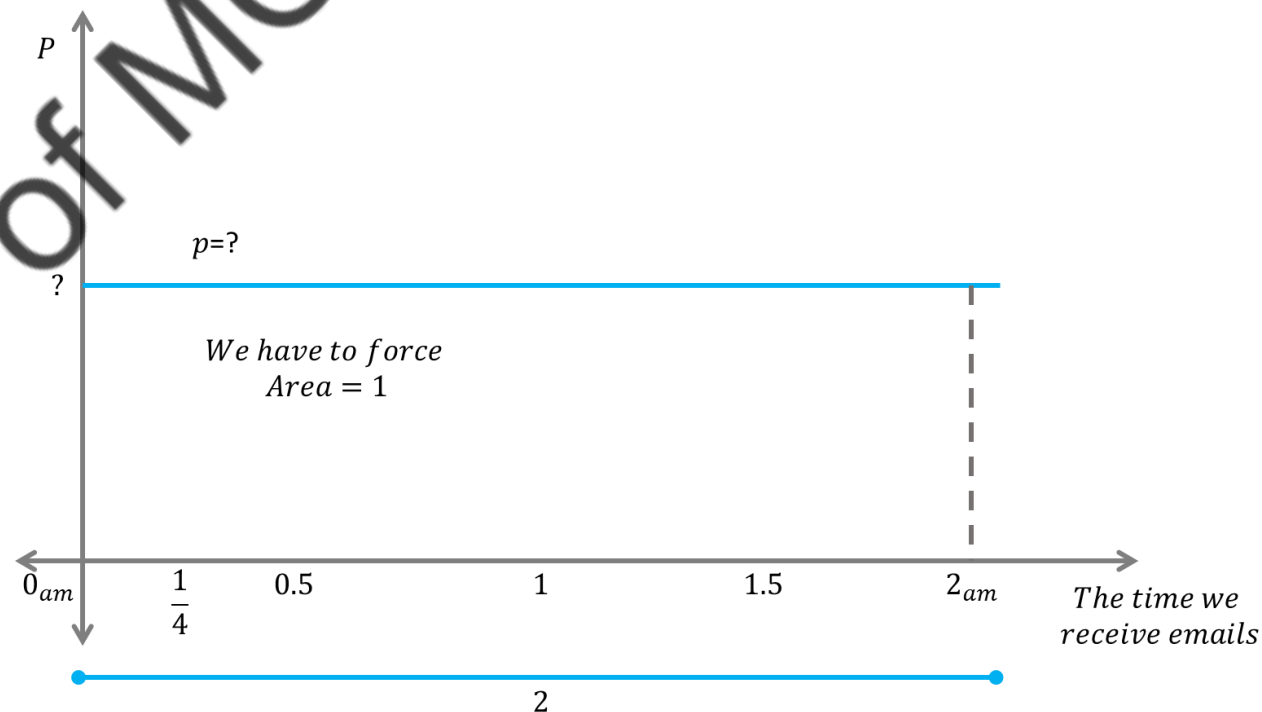


# Uniform Distribution

- We are interested in the probability of receiving emails between 12am and 2am (0 and 2).
- Only based on this information, we cannot say that receiving an email at 1am has bigger probability or less probability than receiving an email at other moments. So, the distribution is Uniform (the same). We did not say that at 1am is more likely to receive emails.

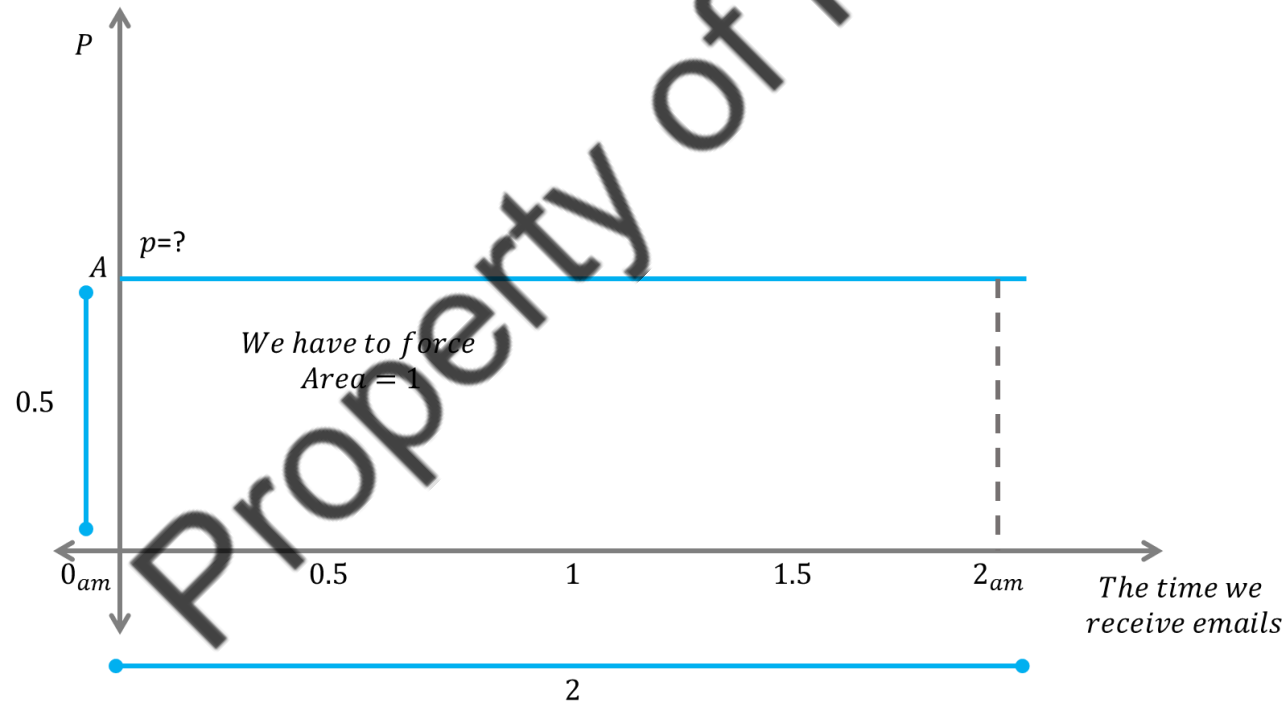


- We already agreed in case of discrete random variables that the probability must sum up at 1. Because the probability of all possible outcomes must be 1.
- However, for continuous random variables, the probability is related with area.

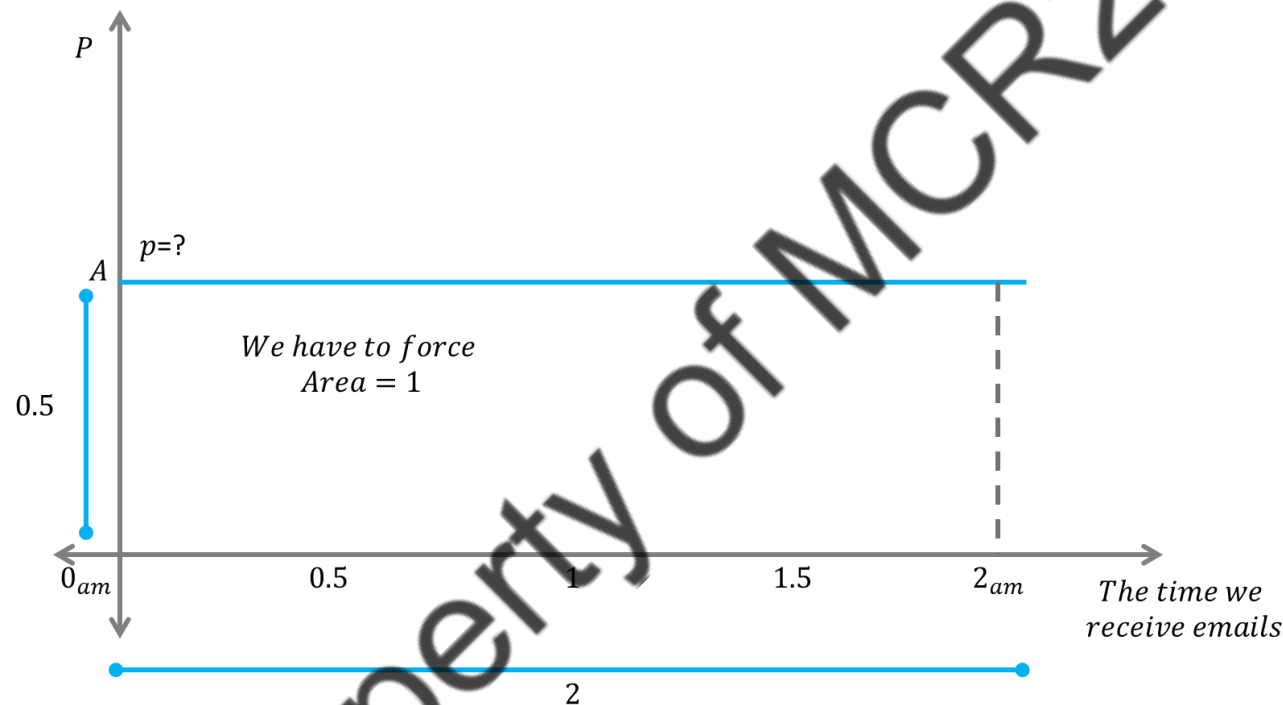


# Uniform Distribution

- Question: Is the meaning of point A in the figure that we have a probability  $p=0.5$ ?



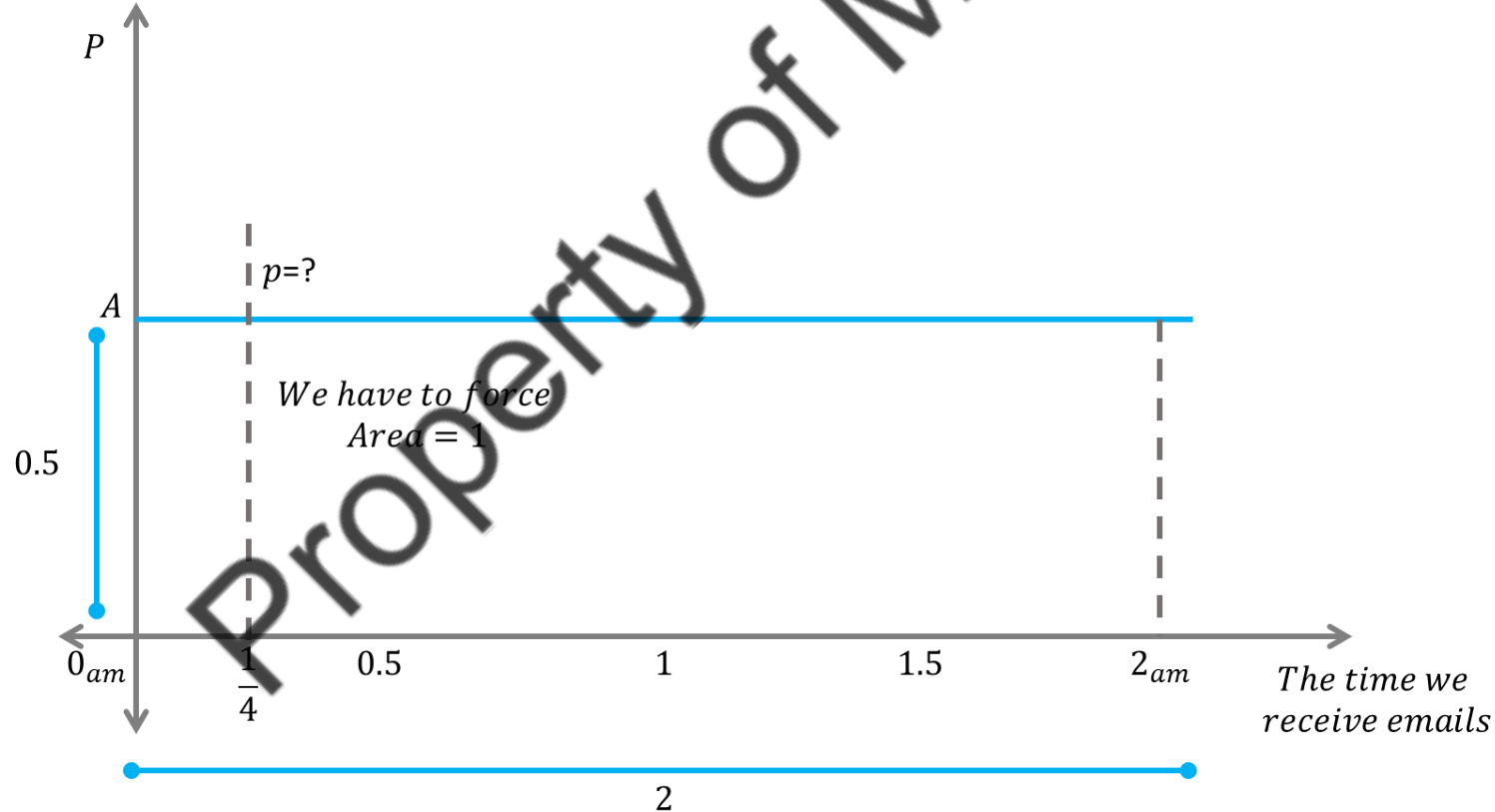
# Uniform Distribution



**Question:** Is the meaning of point A in the figure that we have a probability  $p=0.5$ ?

No. This is just a value such that the area=1. The area is the probability for all possible outcomes.

- Question: Which is the probability to receive an email at EXACT 12.15am?







# Uniform Distribution

---



The probability of receiving an email at EXACT 12.15am is zero

$$p(x = x_i) = 0$$

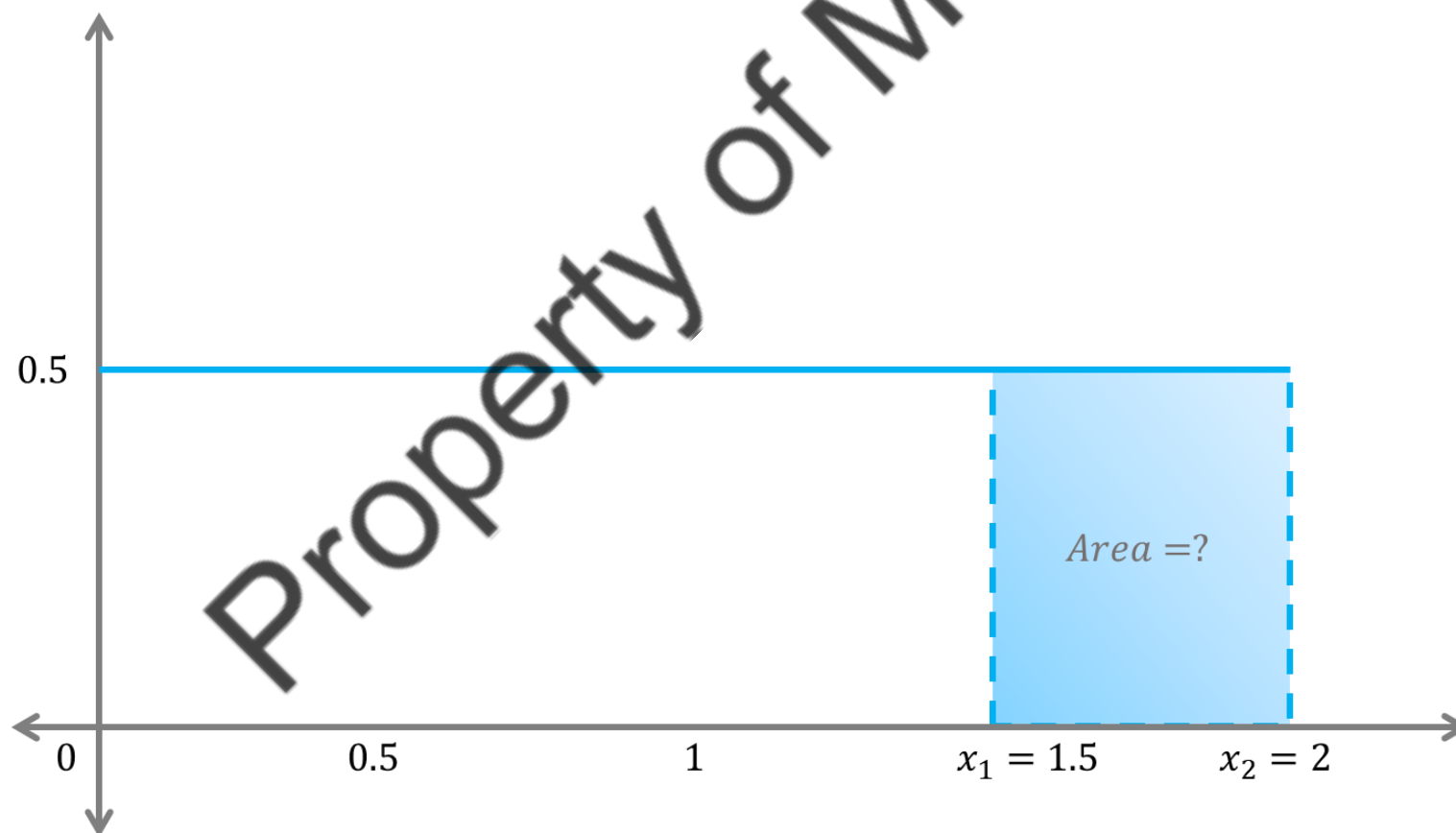
$$p\left(x = \frac{1}{4}\right) = 0$$

The probability of a continuous random variable to be equal with an outcome is zero.

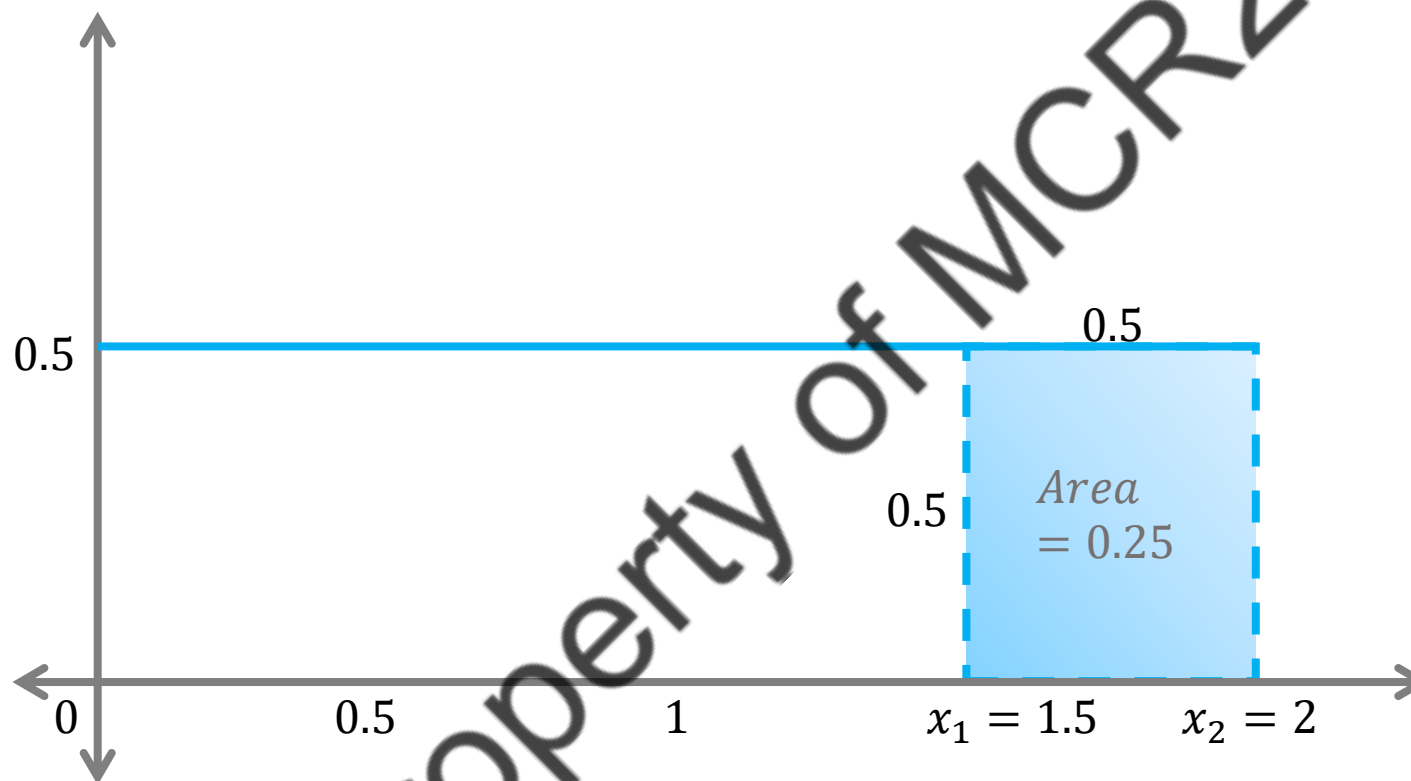
How many possible outcomes we have between 0am and 2am? We have infinite possible outcomes. So, the probability of a single outcome is  $\frac{1}{\infty} = 0$ .

# Uniform Distribution

Which is the probability of receiving emails between 1.30am and 2am?



# Uniform Distribution



$$p(x_1 < x < x_2) = 0.25 \quad (0.5 * 0.5)$$

So, the probability of receiving emails between 1.30am and 2am is 0.25 (25%).



# Uniform Distribution

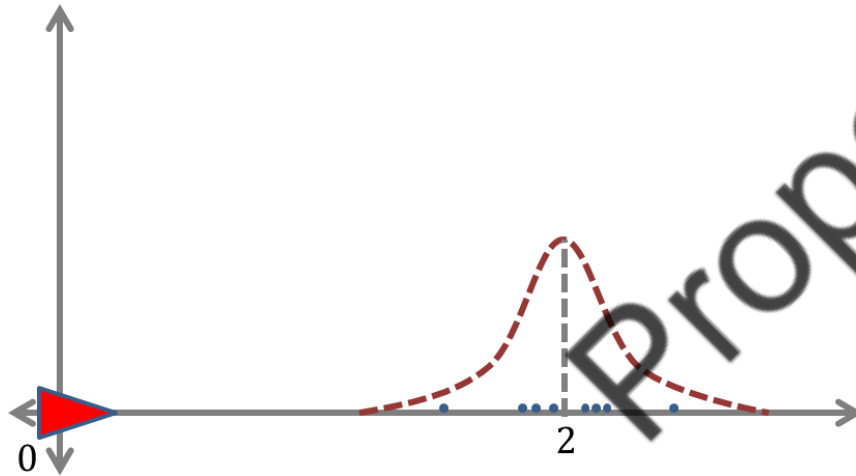
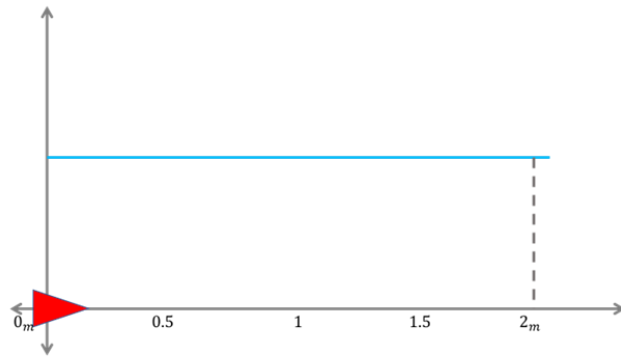
---



- In robotics we don't use uniform distributions. When we use sensors to measure a value, the measurements are distributed around the "true" value.
- For instance, we cannot say that moving a mobile robot for 2 meters we have a uniform distribution for the robot position between 0m and 2m.

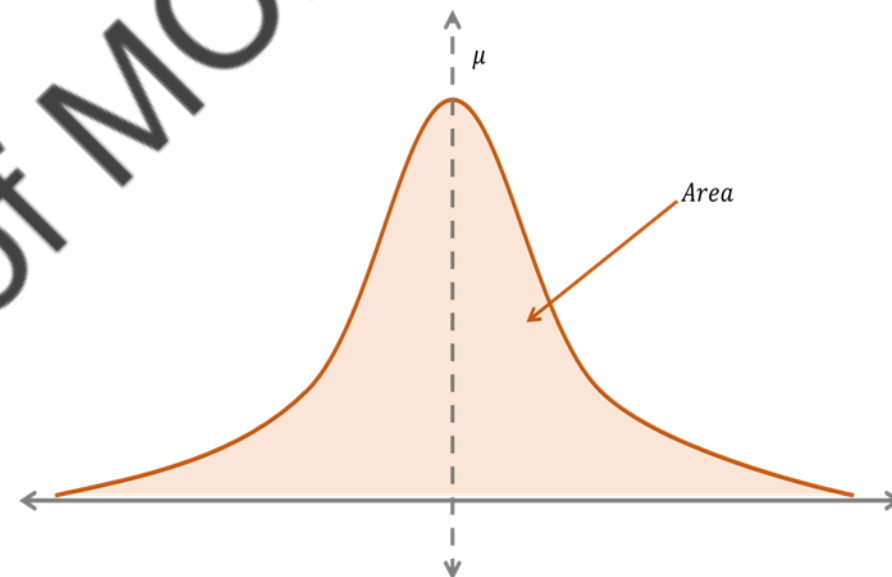
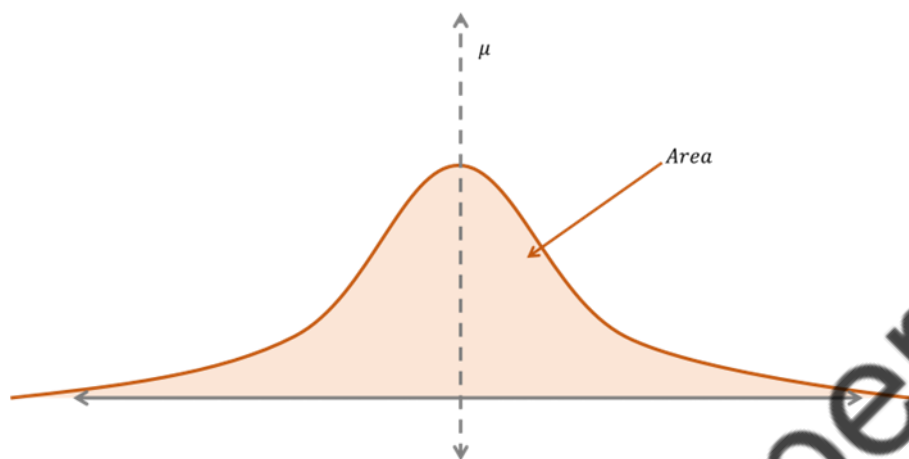
Property of MCR2

# Uniform Distribution vs Gaussian Distribution



In this case we'll use a Gaussian distribution (normal distribution) to represent the uncertainty in the robot position. This distribution is a symmetric bell shape.

$$p_X(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x - \mu)^2}{2\sigma^2}\right)$$



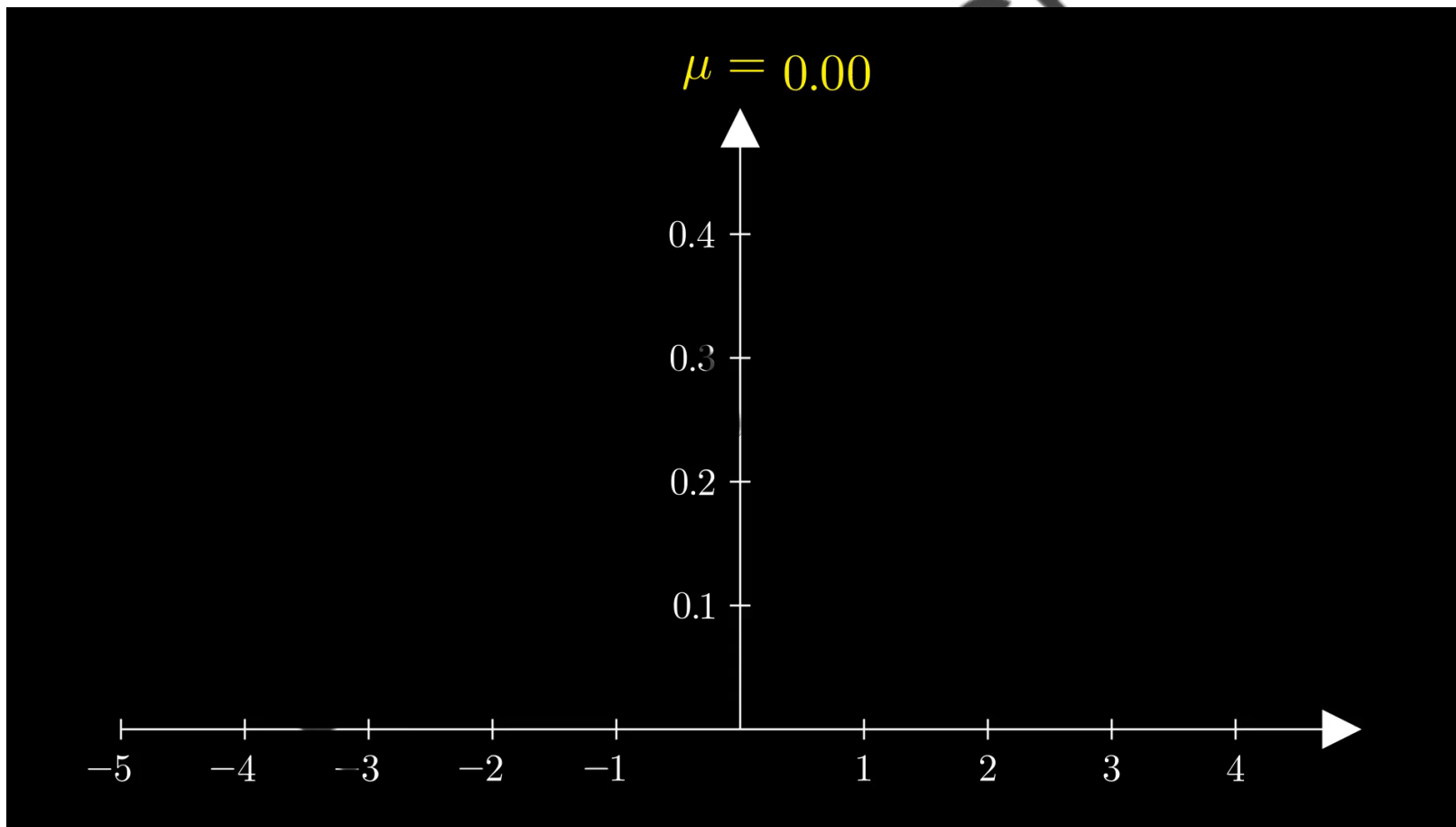
The beauty of the Gaussian Distribution is that we can describe it with just two parameters  $\mu$  and  $\sigma$ .

$x \sim \mathcal{N}(\mu, \sigma)$

$\sigma^2$  variance  
 $\sigma$  standard deviation  
 $\mu$  is the mean

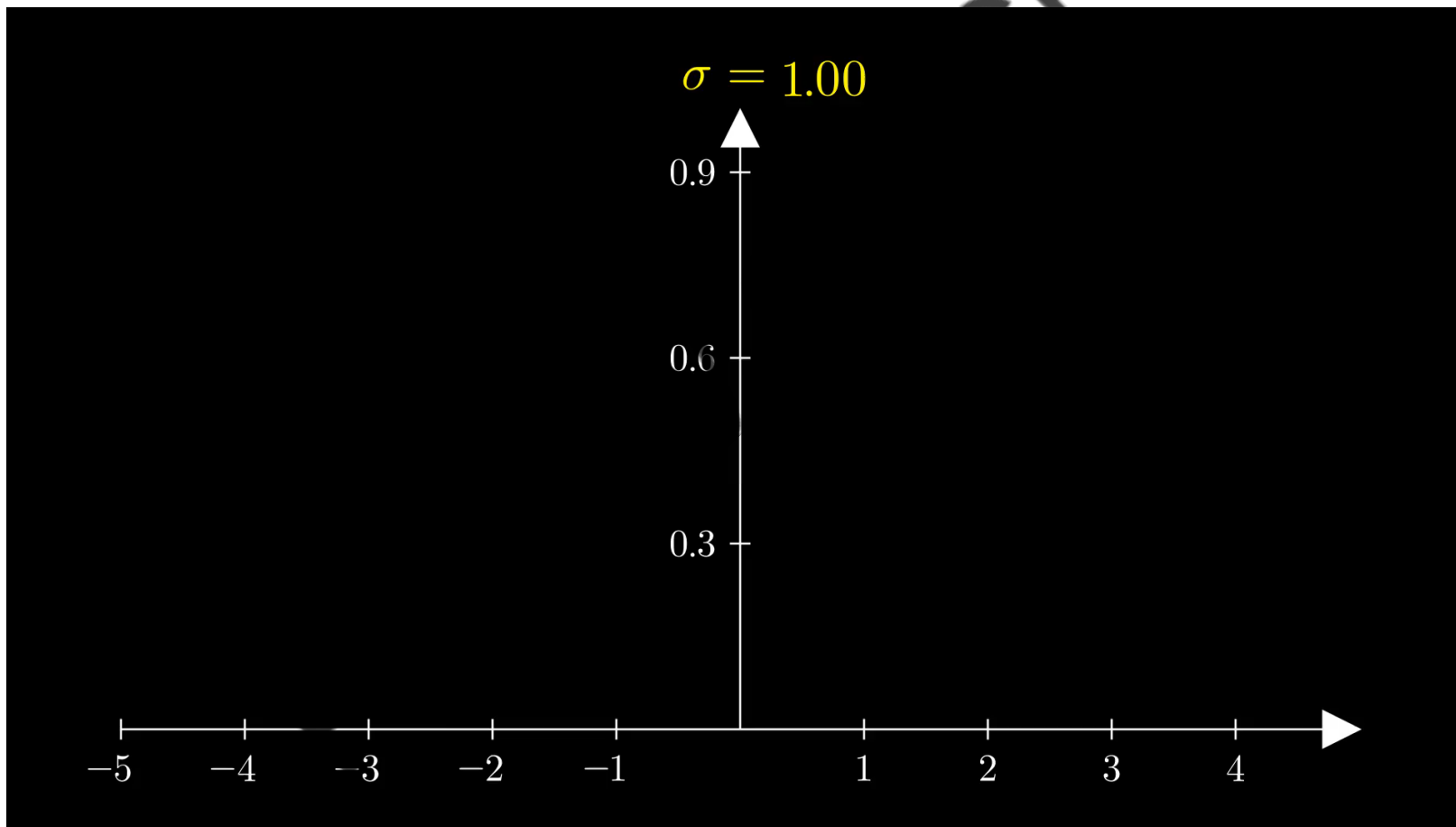


# Gaussian Distribution



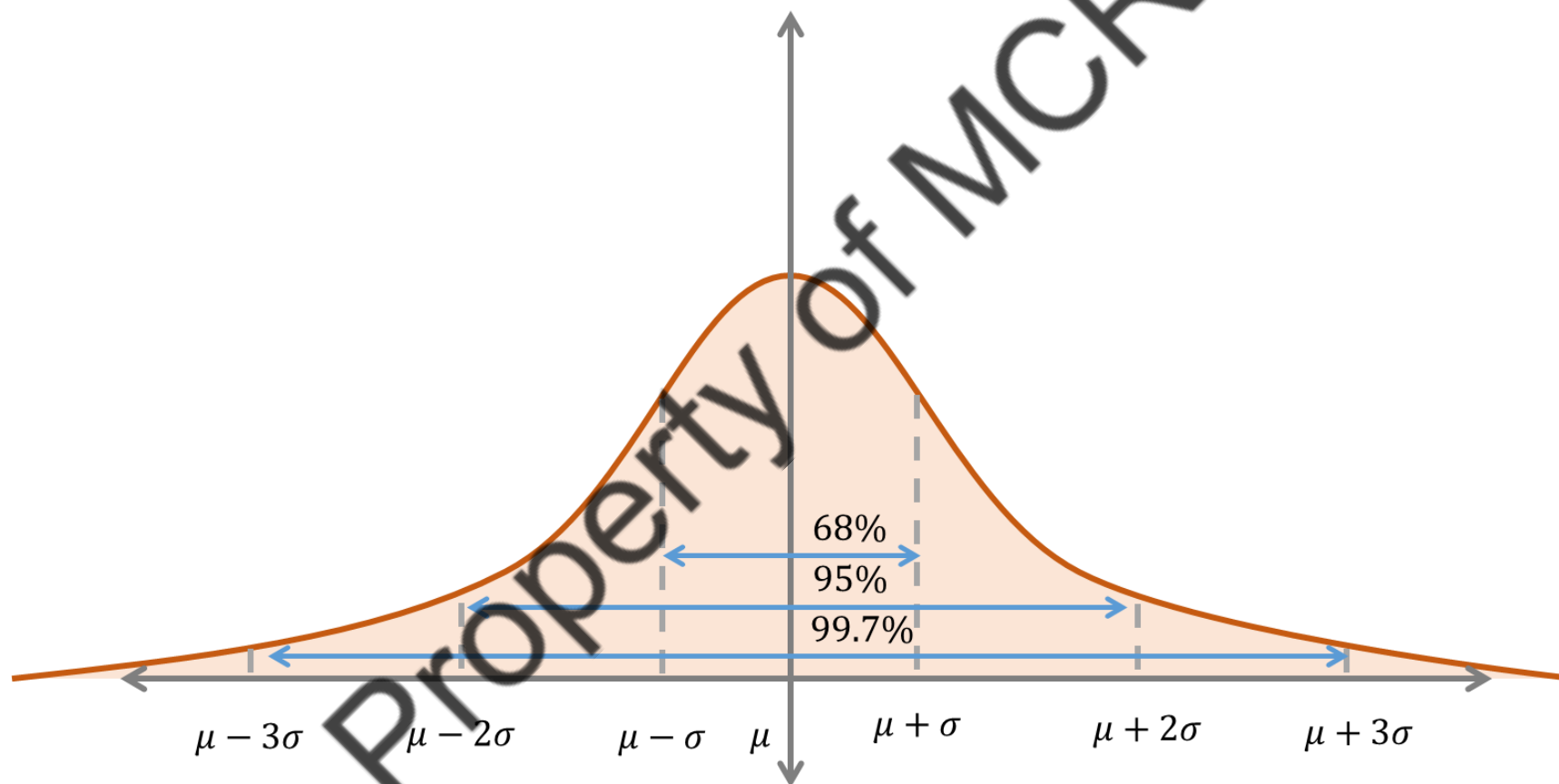


# Gaussian Distribution





# Gaussian Distribution



In robotics we use  $3\sigma$ .

# Kinematics of a differential drive

- The Kinematic model for a non-holonomic mobile robot is

$$\dot{x} = v \cos \theta$$

$$\dot{y} = v \sin \theta$$

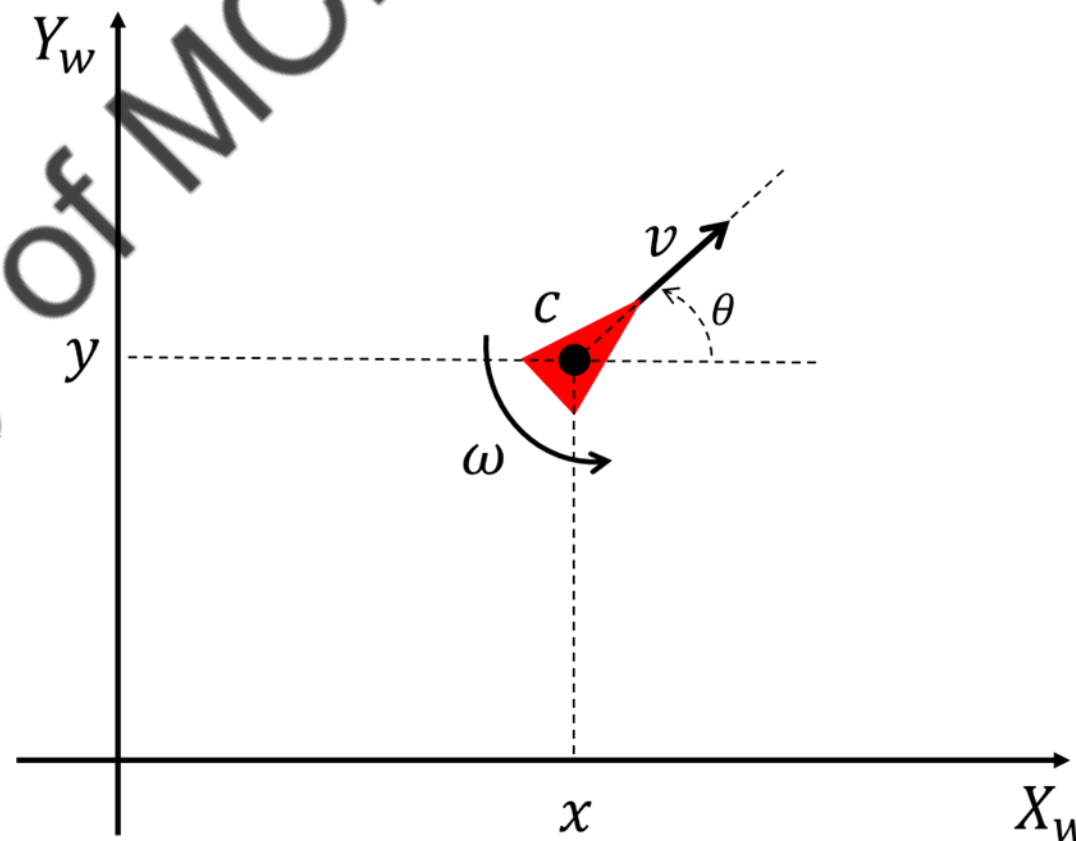
$$\dot{\theta} = \omega$$

- We use the notation  $s_x$ ,  $s_y$ , and  $s_\theta$  for the states of the mobile robot.

$$\dot{\mathbf{s}} = \mathbf{h}(\mathbf{s}, \mathbf{u})$$

$$\frac{d}{dt} \begin{bmatrix} s_x \\ s_y \\ s_\theta \end{bmatrix} = \begin{bmatrix} \cos(s_\theta) & 0 \\ \sin(s_\theta) & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix}$$

- The robot pose  $\mathbf{s}_k = [s_x \ s_y \ s_\theta]^T$
- The robot inputs  $\mathbf{u}_k = [v \ \omega]^T$





# Kinematics of a differential drive



- Using Euler, we discretise the continuous time model
- If  $\Delta t$  is the sampling time, then it is possible to compute the incremental linear and angular displacements,  $\Delta d$  and  $\Delta \theta$ , as follows:

$$\Delta d = v \cdot \Delta t$$

$$\Delta \theta = \omega \cdot \Delta t$$

- To compute the pose of the robot at any given time step, the kinematic model must be numerically integrated.

- The current robot pose depends only on the previous pose and the input velocities.

$$\begin{bmatrix} s_{x,k} \\ s_{y,k} \\ s_{\theta,k} \end{bmatrix} = \begin{bmatrix} s_{x,k-1} \\ s_{y,k-1} \\ s_{\theta,k-1} \end{bmatrix} + \begin{bmatrix} \Delta d \cos(s_{\theta,k-1}) \\ \Delta d \sin(s_{\theta,k-1}) \\ \Delta \theta \end{bmatrix} \Rightarrow \mathbf{s}_k = \mathbf{h}(\mathbf{s}_{k-1}, \mathbf{u}_{k-1})$$

- We can use this model to “estimate” the position of our robot in real life.



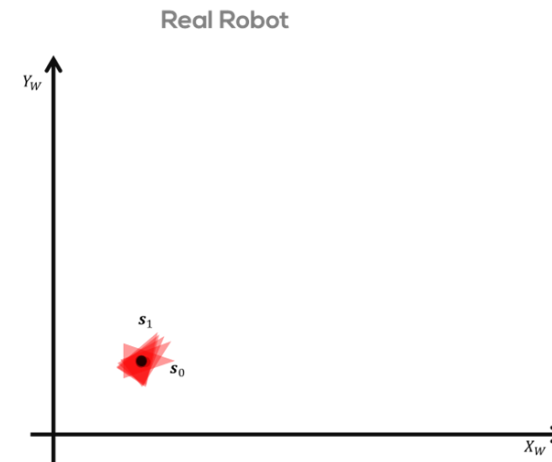
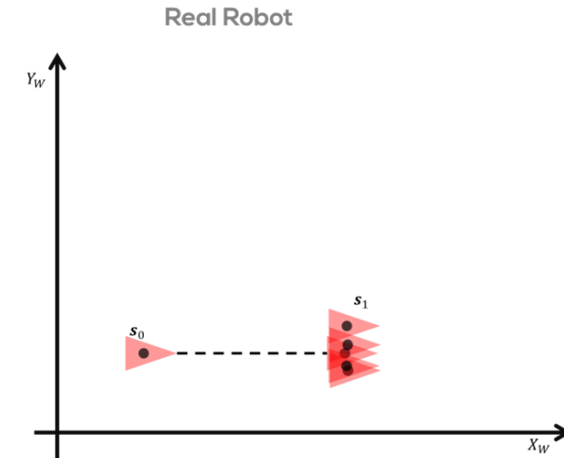
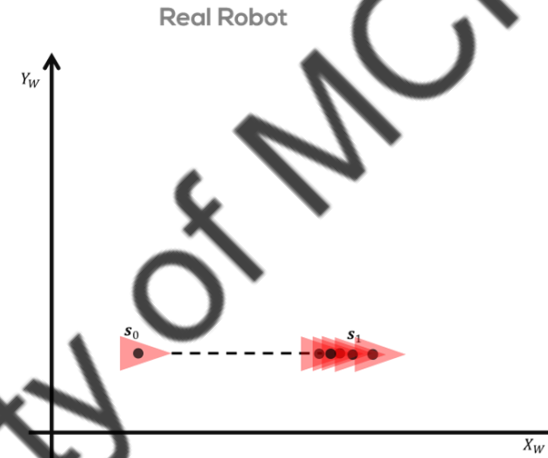
# Model-based Localisation



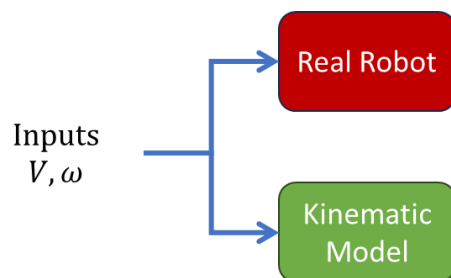
- The pose estimation of a mobile robot is **always associated with some uncertainty** with respect to its state parameters.

From a geometric point of view, the error in differential-drive robots is classified into three groups:

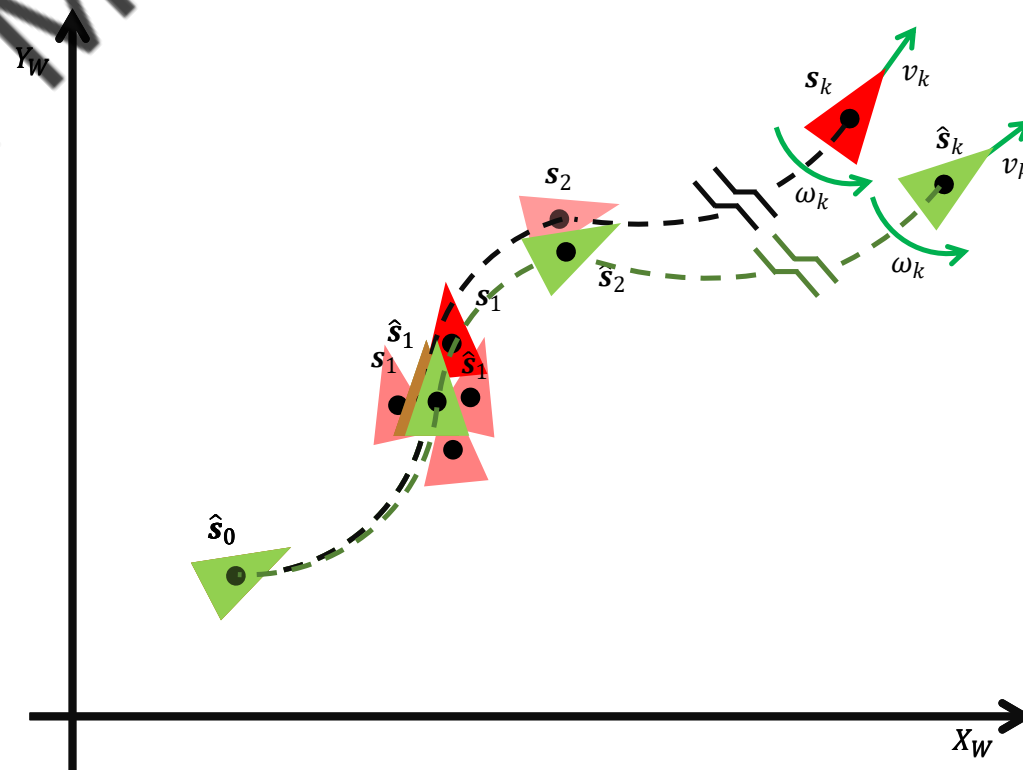
- Range error:** it is associated with the computation of  $\Delta d$  over time.
- Turn error:** it is associated with the computation of  $\Delta \theta$  over time.
- Drift error:** it is associated with the difference between the angular speed of the wheels and it affects the error in the angular rotation of the robot.
- These Errors are related to each other!!



- By comparing the estimated pose using the model with the real robot, it is possible to see that the model will diverge very quickly from the real robot's position.



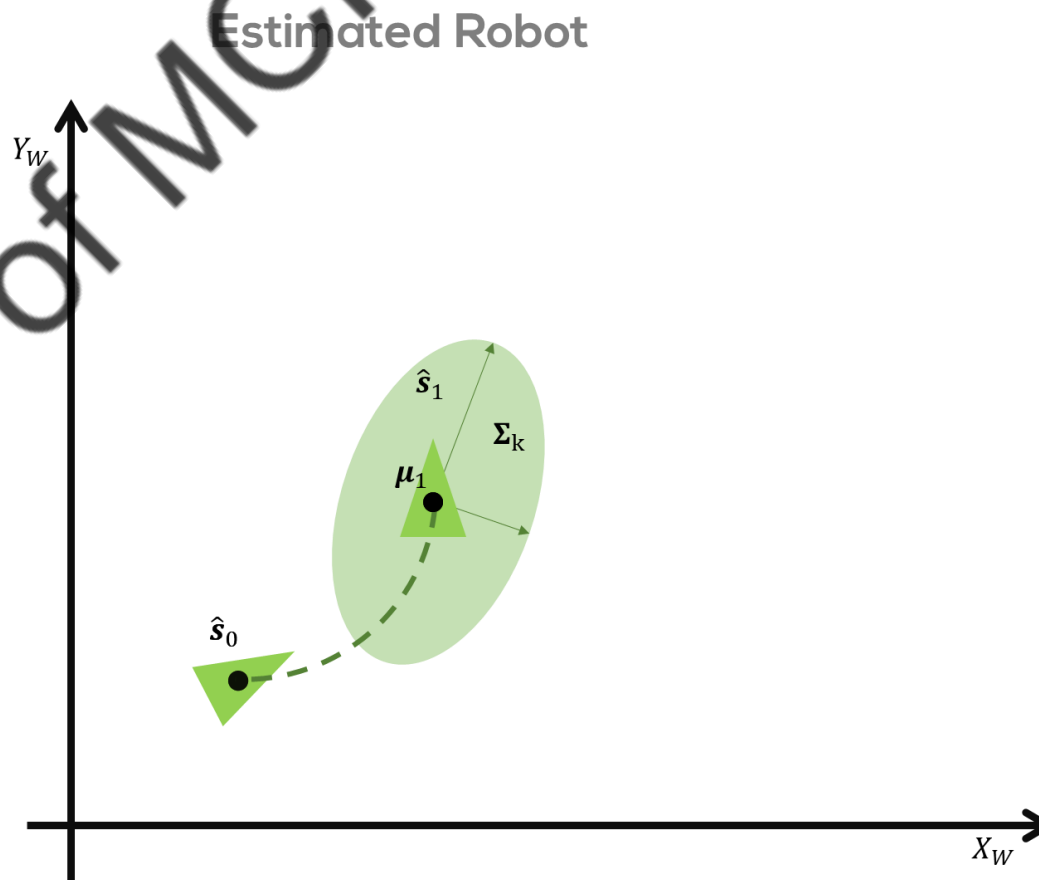
- So, which is the obvious way to represent the uncertainty in robot position?



Due to such uncertainty, it is possible to represent the belief of the robot pose by a Gaussian distribution, where

- the mean vector  $\mu_k$  is the best estimate of the pose, and
- the covariance matrix  $\Sigma_k$  is the uncertainty of the pose that encapsulates the errors presented in the previous slide.
- The Gaussian distribution (or normal distribution) is denoted by

$$s_k \sim \mathcal{N}(\mu_k, \Sigma_k)$$

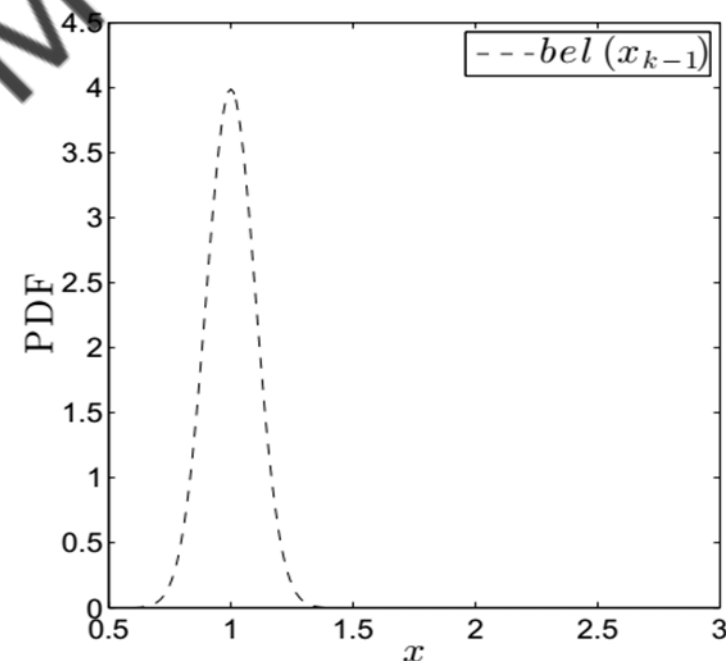


- A random variable  $X$  is *normally distributed*, or *Gaussian*, if its probability density function is defined as:

$$p_X(x) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(x - \mu_X)^2}{2\sigma^2}\right)$$

where,  $\mu_X$ ,  $\sigma^2$  are the *mean* and *variance*, respectively; they are the distribution parameters.

- The notation  $X \sim \mathcal{N}(\mu_X, \Sigma_X)$  means that the random variable  $X$  is Gaussian.



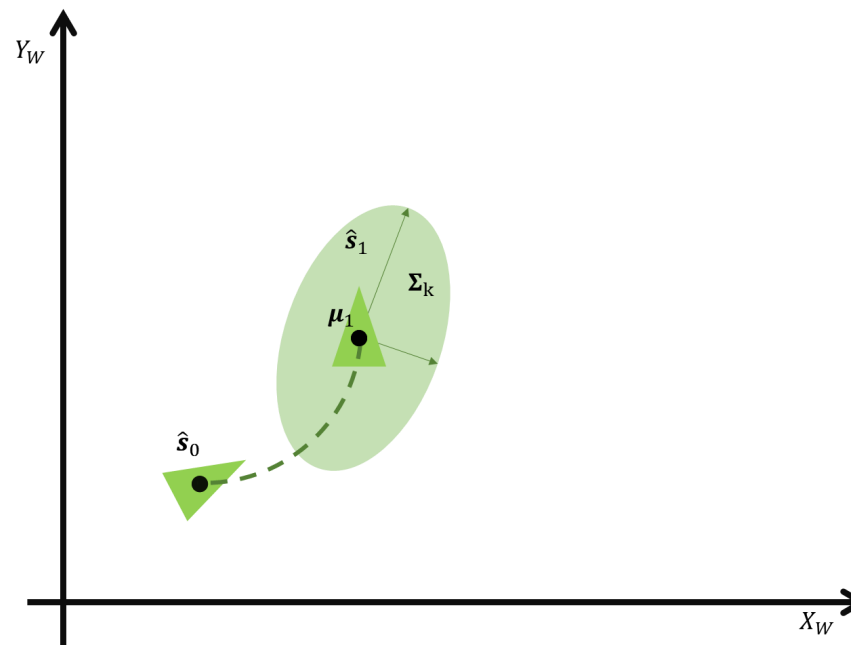
- In the context of probability, the robot pose at time step  $k$ , denoted by  $\mathbf{s}_k$ , can be described as function of previous robot pose  $\mathbf{s}_{k-1}$  and the current control input  $\mathbf{u}_k = [v_k \ \omega_k]^T$ . This process is called the *robot motion model*.
- At this stage we assume that we already have obtained  $\mathbf{Q}_k$  experimentally. However, we'll come back later and do the experiments to calibrate  $\mathbf{Q}_k$ .

$$\mathbf{s}_k = \mathbf{h}(\mathbf{s}_{k-1}, \mathbf{u}_k) + \mathbf{q}_k$$

where  $\mathbf{q}_k$  is an additive Gaussian noise such that  $\mathbf{q}_k \sim \mathcal{N}(\mathbf{0}, \mathbf{Q}_k)$ , and  $\mathbf{Q}_k$  is a positive semidefinite covariance matrix.

- This noise is directly related to the error sources described in previous slides ...

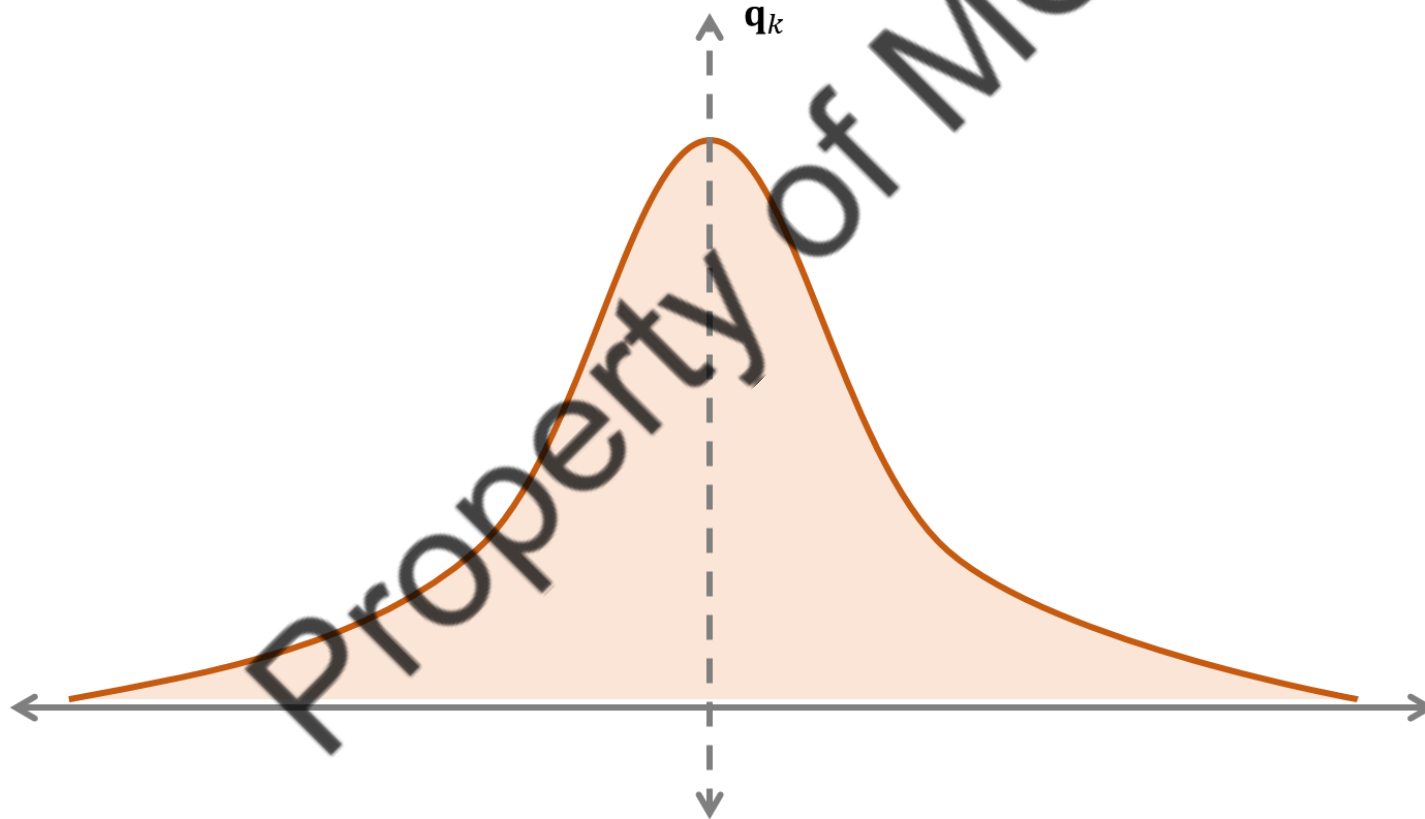
Estimated Robot





$$\mathbf{s}_k = \mathbf{h}(\mathbf{s}_{k-1}, \mathbf{u}_k) + \mathbf{q}_k$$

$\mathbf{q}_k$

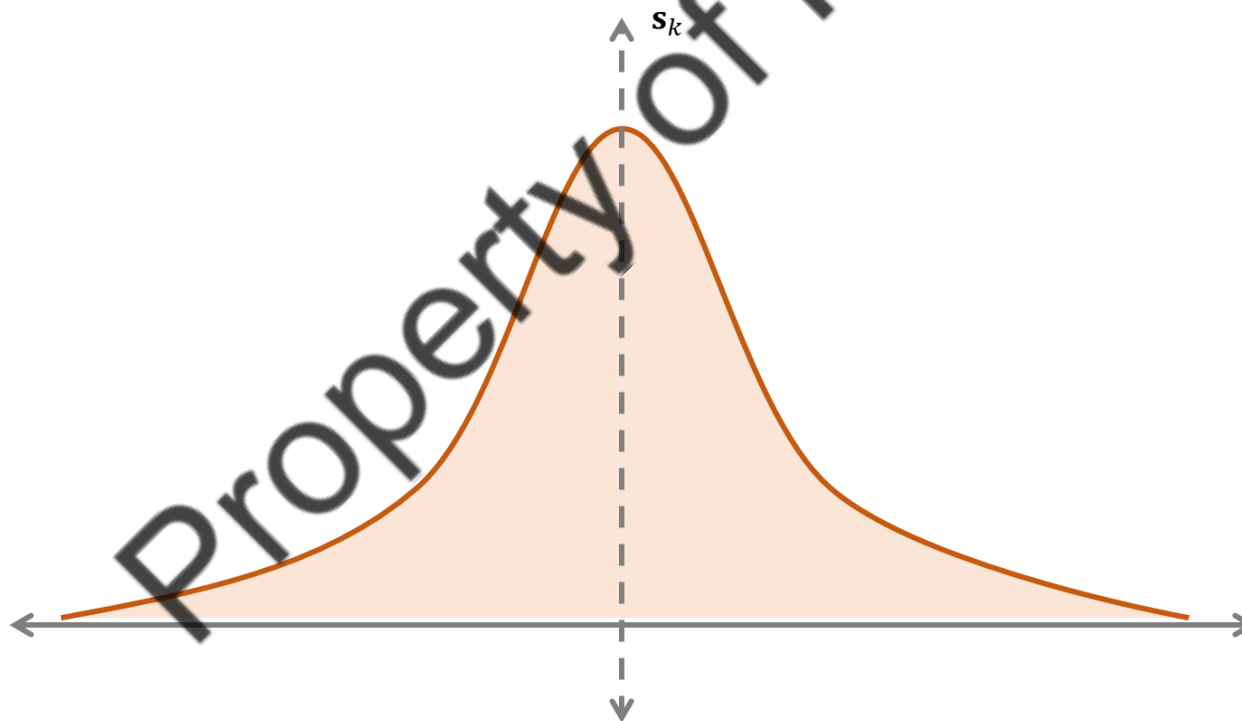


But ...

# Model-based Localisation (Dead Reckoning)

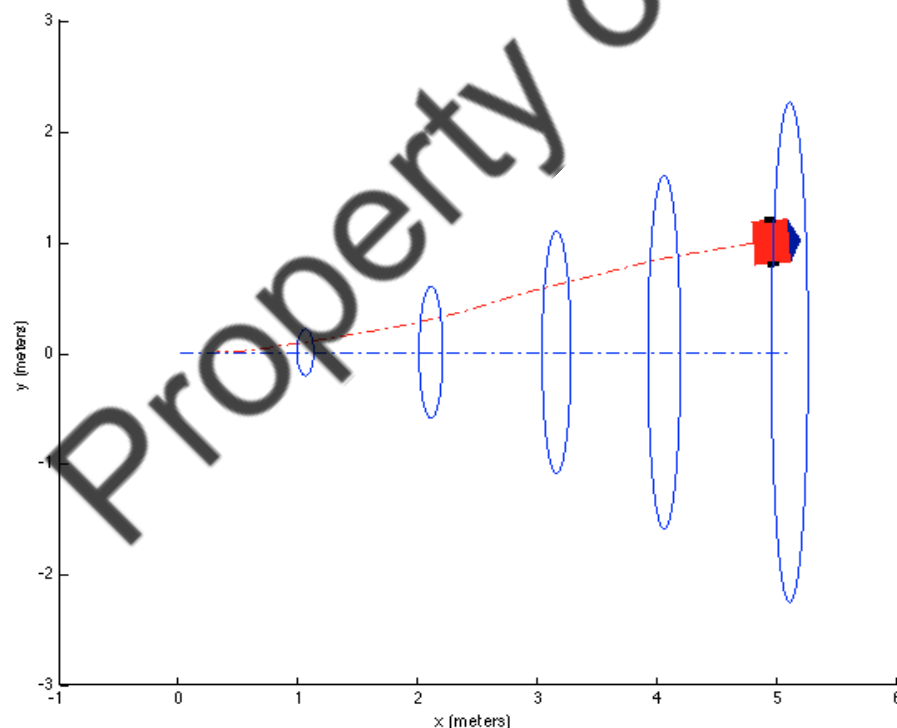
$$\mathbf{s}_k = \mathbf{h}(\mathbf{s}_{k-1}, \mathbf{u}_k) + \mathbf{q}_k$$

But, we want  $\mathbf{s}_k$  to be a Gaussian distribution as well.

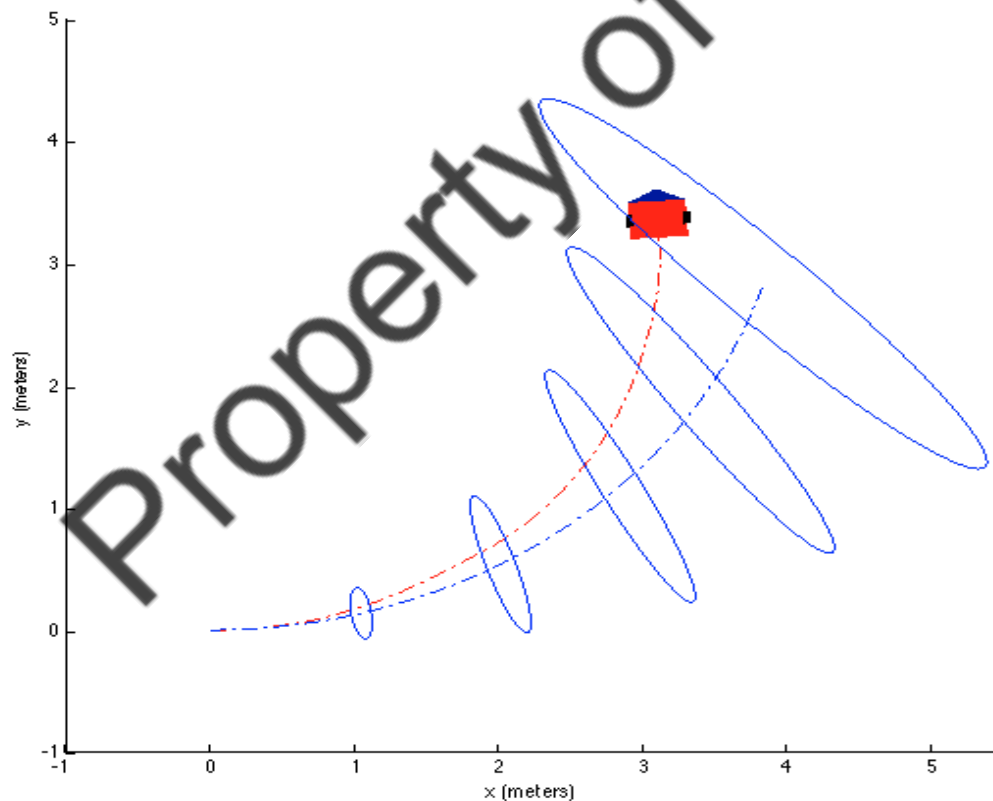


How can we do this???

The joint uncertainty of  $s_x$  and  $s_y$  is represented by an ellipsoid around the robot. This ellipsoid is named **Ellipsoid of Confidence**. As the robot moves along the  $x$ -axis, its uncertainty along the  $y$ -axis increases faster than the  $x$ -axis due to the drift error.



The uncertainty ellipsoid is no longer perpendicular to the motion direction as soon as the robot starts to turn.



# Model Based Localisation



$$\mathbf{s}_1 \sim \mathcal{N}(\boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1)$$

$$\mathbf{s}_2 \sim \mathcal{N}(\boldsymbol{\mu}_2, \boldsymbol{\Sigma}_2)$$

$\vdots$

$$\mathbf{s}_{k-1} \sim \mathcal{N}(\boldsymbol{\mu}_{k-1}, \boldsymbol{\Sigma}_{k-1})$$

$$\mathbf{s}_k \sim \mathcal{N}(\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$$

We said that the beauty of a Gaussian distribution is that it is a closed form, it is represented by only two parameters,  $\boldsymbol{\mu}$  and  $\boldsymbol{\Sigma}$ .

So, we just have to propagate, at every time instant, two parameters and we have the Gaussian distribution at the next time step.



# Model Based Localisation



We propagate the mean  $\mu$ .

$$\mu_k = h(\mu_{k-1}, u_k)$$

We propagate the covariance  $\Sigma$ . But we have a problem here... If we propagate  $\Sigma$  using a nonlinear function, then the new  $\Sigma$  will not represent the covariance for a Gaussian distribution anymore.

Property of MCR²



To propagate  $\Sigma$  we use the following result:

## *Affine Transformation*

Consider  $X \sim \mathcal{N}(\mu_X, \Sigma_X)$  in  $\mathbb{R}^n$ , and let  $Y = AX + \mathbf{b}$  be the affine transformation, where  $\mathbf{A} \in \mathbb{R}^{m \times n}$  and  $\mathbf{b} \in \mathbb{R}^m$ . Then, the random vector  $Y \sim \mathcal{N}(\mu_Y, \Sigma_Y)$  such that:

$$\mu_Y = \mathbf{A}\mu_X + \mathbf{b}$$

$$\Sigma_Y = \mathbf{A}\Sigma_X\mathbf{A}^T$$





# Model-based Localisation

---



So, to propagate  $\Sigma$  we need a linear map, a linear function. However, the kinematic model of the non-holonomic robot is nonlinear.

Solution: Linearisation

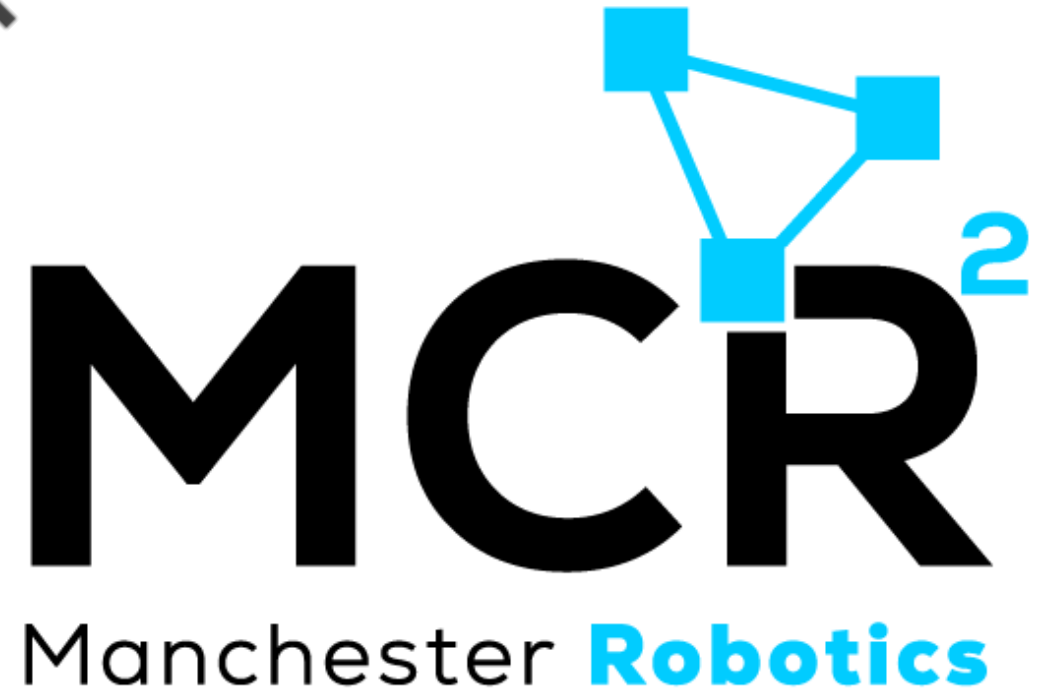
Property of MCR2



Linearisation

Property of MCR<sup>2</sup>

*{Learn, Create, Innovate};*





# Linearisation of nonlinear systems

---



- Linearisation around equilibrium points
- Linearisation around an operating point

Property of MCR2





# Linearisation of nonlinear systems



- As we stated in the first lecture, the state-space representation of a system can be used for both, linear and nonlinear systems.

If for any  $x, y, k \in \mathbb{R}$

$$f(kx) = kf(x)$$

$$f(x + y) = f(x) + f(y)$$

then the function  $f$  is linear

A function which does not satisfy these conditions is a nonlinear function.

Property of MCR2



# Linearisation around equilibrium points

Let us consider the dynamical system (autonomous):

$$\dot{x}(t) = f(x(t)), \quad x(0) = x_0$$

$$f: \mathbb{R}^n \rightarrow \mathbb{R}^n$$

Definition 1 (equilibrium point): The point  $x_e \in \mathbb{R}^n$  is an equilibrium point of the system if  $f(x_e) = 0$ , i.e.,

$$\dot{x}(t) \Big|_{x=x_e} = f(x_e) = 0$$

Property of MCR2



# Linearisation around equilibrium points



Definition 2 (Stability of equilibrium points): An equilibrium point  $x_e$  is said to be stable if for any initial condition around the equilibrium point, the distance from the solution at any instant and the equilibrium point is bounded. Mathematically, for any  $\epsilon > 0$  there exist  $\delta$  such that:

$$\|x_0 - x_e\| < \delta \quad \text{then} \quad \|x(t) - x_e\| < \epsilon \quad \forall t \geq 0$$

Moreover, is said to be asymptotically stable if  $\lim_{t \rightarrow \infty} x(t) = 0$ .

If the equilibrium point  $x_e$  is not stable, it is said to be unstable.





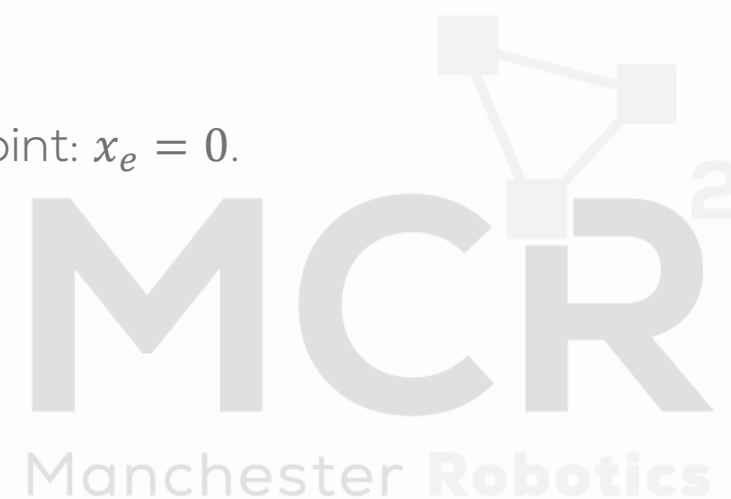
# Linearisation around equilibrium points



An equilibrium point is stable if the trajectory of the state tries to recover or stay close to the equilibrium point when the system slightly disturbed. In other words, it is able to keep this position even though small disturbances will disturb the system. When the equilibrium point is unstable, any small perturbation will lead to the loss of equilibrium and the equilibrium position will not be recovered.

In the linear case, we are only interested in the trivial equilibrium point:  $x_e = 0$ .

Property of MCR2





# Linearisation around equilibrium points



A nonlinear system behaves approximately as a linear system near an equilibrium point (of course if the function  $f$  is continuous). This can be proved using Taylor expansion around the equilibrium point:

$$f(x) \cong f(x_e) + J_f(x_e)(x - x_e)$$

where  $J_f$  is the Jacobian matrix. It is clear that if  $f(x_e) = 0$  and we define a new state as  $\Delta x = x - x_e$ , then  $f(x)$  becomes the matrix  $J_f$  times the vector  $\Delta x$ . If  $x - x_e$  is close to zero, hence the nonlinear system becomes linear.

$$\frac{d}{dt}(\Delta x) \cong A\Delta x$$

where  $A = J_f(x_e)$  and if  $\Delta x = x - x_e$ , then  $\frac{d}{dt}(\Delta x) = \dot{x}$ .





# Linearisation around equilibrium points

Definition 3 (Jacobian): Given a vectorial function  $f: \mathbb{R}^n \rightarrow \mathbb{R}^n$ , then the Jacobian matrix  $\mathcal{J}_f \in \mathbb{R}^{n \times n}$  is defined by:

$$\mathcal{J}_f = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \dots & \frac{\partial f_1}{\partial x_n} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \dots & \frac{\partial f_2}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial x_1} & \frac{\partial f_n}{\partial x_2} & \dots & \frac{\partial f_n}{\partial x_n} \end{bmatrix}$$



# Linearisation around equilibrium points



So, we can analyse the equilibrium point by performing a linearisation and studying the properties of the Jacobian matrix at the equilibrium point.

The stability of the equilibrium point  $x_e$  of the system  $\dot{x}(t) = f(x(t))$ ,  $x(0) = x_0$ , is equivalent to the stability of the linear system defined by  $\dot{x} = Ax$ , where  $A = J_f(x_e)$ .

Property of MCR2





# Linearisation around equilibrium points



Example: Pendulum system

$$\ddot{\theta} + \frac{g}{l} \sin(\theta) = 0$$

Let us define the states:

$$x_1 = \theta$$

$$x_2 = \dot{\theta}$$

So, the state-space representation of the pendulum system is:

$$\begin{cases} \dot{x}_1 = x_2 \\ \dot{x}_2 = -\frac{g}{l} \sin(x_1) \end{cases}$$

Property of MCR²





# Linearisation around equilibrium points



Hence, the vectorial function  $f(x_1, x_2)$  is given by:

$$f(x_1, x_2) = \begin{bmatrix} x_2 \\ -\frac{g}{l} \sin(x_1) \end{bmatrix}$$

At the equilibrium points satisfies  $f(x_1, x_2) = 0$ , i.e.,

$$x_2 = 0$$

$$\sin(x_1) = 0$$

Property of MCR2



# Linearisation around equilibrium points

For  $\theta \in (-\pi, \pi]$ , the equilibrium points of the pendulum system are  $x_{e1} = (0, 0)$  and  $x_{e2} = (\pi, 0)$ .

The Jacobian matrix is:

$$\mathcal{J}_f(x_1, x_2) = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} \end{bmatrix}$$

$$\mathcal{J}_f = \begin{bmatrix} \frac{\partial}{\partial x_1} x_2 & \frac{\partial f_1}{\partial x_2} x_2 \\ \frac{\partial}{\partial x_1} \left( -\frac{g}{l} \sin(x_1) \right) & \frac{\partial}{\partial x_2} \left( -\frac{g}{l} \sin(x_1) \right) \end{bmatrix}$$

$$\mathcal{J}_f = \begin{bmatrix} 0 & 1 \\ -\frac{g}{l} \cos(x_1) & 0 \end{bmatrix}$$

# Linearisation around equilibrium points

At equilibrium point  $x_{e1} = (0,0)$ , the pendulum system can be linearised as:

$$\dot{x} = J_f(0,0)x = \begin{bmatrix} 0 & 1 \\ -\frac{g}{l} & 0 \end{bmatrix} x$$

Question: How can we verify mathematically if the equilibrium point is stable???

Property of MCR²

# Linearisation around equilibrium points

Answer: By checking the eigenvalues of  $J_f(0,0)$ , i.e., the eigenvalues of the  $A$  matrix of the linearised pendulum system.

The eigenvalues of  $J_f(0,0)$  are given by:

$$\det(J_f(0,0) - \lambda I) = \begin{vmatrix} -\lambda & 1 \\ -\frac{g}{l} & -\lambda \end{vmatrix} = \lambda^2 + \frac{g}{l} = 0$$



$$\lambda = \pm \sqrt{\frac{g}{l}} j$$

So??? How is the equilibrium point???

Property of MCR2



# Linearisation around equilibrium points



The real part of the eigenvalues is zero and they are on the imaginary axis, then the equilibrium point  $x_{e1}$  is marginally stable.

At equilibrium point  $x_{e2} = (\pi, 0)$  we need to use a transformation of the space since the equilibrium point is not the origin, hence  $\Delta x = x - x_2$ . Thus the pendulum systems can be linearised around  $x_{e2}$  as:

$$\Delta \dot{x} = J_f(\pi, 0)\Delta x = \begin{bmatrix} 0 & 1 \\ \frac{g}{l} & 0 \end{bmatrix} \Delta x$$

Property of MCR2





# Linearisation around equilibrium points

The eigenvalues of  $J_f(\pi, 0)$  are given by:

$$\det(J_f(\pi, 0) - \lambda I) = \begin{bmatrix} -\lambda & 1 \\ \frac{g}{l} & -\lambda \end{bmatrix} = \lambda^2 - \frac{g}{l} = 0$$



$$\lambda = \pm \sqrt{\frac{g}{l}}$$

So??? How is the equilibrium point???

# Linearisation around equilibrium points

The eigenvalues of  $J_f(\pi, 0)$  are given by:

$$\det(J_f(\pi, 0) - \lambda I) = \begin{bmatrix} -\lambda & 1 \\ \frac{g}{l} & -\lambda \end{bmatrix} = \lambda^2 - \frac{g}{l} = 0$$



$$\lambda = \pm \sqrt{\frac{g}{l}}$$

The real part of one eigenvalue is positive, therefore the equilibrium point  $x_{e2}$  is unstable.



# Linearisation around equilibrium points

---



Animation for pendulum system:

<https://www.youtube.com/watch?v=ovDWmGhMKyl&list=PLqCuMQTwnIP99CrzdPEroGhdAhzVfvWgR&index=11>

Property of MCR²





# Linearisation around an operating point

---



When the nonlinear system is required to be operating with an input that is different to zero, then we can no longer refer to as an equilibrium point of the system since this concept is related with the autonomous systems. In this case, we use the concept of operating point.

Property of MCR2





# Linearisation around an operating point



Let us consider the nonlinear system given by:

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

$$y = h(x, u)$$

Then if  $f(x_0, u_0) = 0$ , the point  $(x_0, u_0)$  is referred to as an operating point. Under this condition, we can perform a linearisation of the system. Let us define the new input, state, and output as the variation around  $x_0, u_0$ , and  $y_0$ .

$$\Delta u = u - u_0$$

$$\Delta x = x - x_0$$

$$\Delta y = y - y_0$$

Property of MCR2



# Linearisation around an operating point

We apply a Taylor expansion of  $f(x, u)$  and  $h(x, u)$  around the point  $(x_0, u_0)$  and we get:

$$f(x, u) \simeq f(x_0, u_0) + J_f^x(x_0, u_0)(x - x_0) + J_f^u(x_0, u_0)(u - u_0)$$

$$h(x, u) \simeq h(x_0, u_0) + J_h^x(x_0, u_0)(x - x_0) + J_h^u(x_0, u_0)(u - u_0)$$

where  $f(x_0, u_0) = 0$  and  $h(x_0, u_0) = y_0$ .

The linearised system is given by:

$$\frac{d}{dt}(\Delta x) \simeq J_f^x(x_0, u_0)\Delta x + J_f^u(x_0, u_0)\Delta u$$

$$\Delta y \simeq J_h^x(x_0, u_0)\Delta x + J_h^u(x_0, u_0)\Delta u$$

where the superscripts  $x$  and  $u$  in the Jacobian matrices indicate the parameter that is considered as a variable.



# Linearisation for uncertainty propagation



- In this case the linearisation is performed with respect to the states only.

$$\mathbf{s}_{k+1} = \mathbf{h}(\mathbf{s}_k, \mathbf{u}_k) = \begin{bmatrix} s_{x,k} + \Delta t \cdot v_k \cdot \cos(s_{\theta,k}) \\ s_{y,k} + \Delta t \cdot v_k \cdot \sin(s_{\theta,k}) \\ s_{\theta,k} + \Delta t \cdot \omega_k \end{bmatrix}$$

- The following equation represents the Jacobian matrix of  $\mathbf{h}(\mathbf{s}_k, \mathbf{u}_k)$  with respect to each variable in  $\mathbf{s}_k$ , evaluated at  $\mathbf{s}_k = \boldsymbol{\mu}_k$ :

$$\mathbf{H}_k = \nabla_{\mathbf{s}_k} \mathbf{h}(\mathbf{s}_k, \mathbf{u}_k) \Big|_{\mathbf{s}_k = \boldsymbol{\mu}_k}$$

$$\mathbf{H}_k = \begin{bmatrix} \frac{\partial h_1}{\partial s_{x,k}} & \frac{\partial h_1}{\partial s_{y,k}} & \frac{\partial h_1}{\partial s_{\theta,k}} \\ \frac{\partial h_2}{\partial s_{x,k}} & \frac{\partial h_2}{\partial s_{y,k}} & \frac{\partial h_2}{\partial s_{\theta,k}} \\ \frac{\partial h_3}{\partial s_{x,k}} & \frac{\partial h_3}{\partial s_{y,k}} & \frac{\partial h_3}{\partial s_{\theta,k}} \end{bmatrix}$$

$$\mathbf{H}_k = \begin{bmatrix} 1 & 0 & -\Delta t \cdot v_k \cdot \sin(\mu_{\theta,k}) \\ 0 & 1 & \Delta t \cdot v_k \cdot \cos(\mu_{\theta,k}) \\ 0 & 0 & 1 \end{bmatrix}$$

- What about the inputs, this is a non autonomous system!

# Linearisation for uncertainty propagation

- If a linear model is required to derive the control law for the robot navigation the linearisation is computed with respect to the states as well as to the control signals.

$$\mathbf{s}_{k+1} = \begin{bmatrix} s_{x,k+1} \\ s_{y,k+1} \\ s_{\theta,k+1} \end{bmatrix} = \mathbf{h}(\mathbf{s}_k, \mathbf{u}_k) = \begin{bmatrix} s_{x,k} + \Delta t \cdot v_k \cdot \cos(s_{\theta,k}) \\ s_{y,k} + \Delta t \cdot v_k \cdot \sin(s_{\theta,k}) \\ s_{\theta,k} + \Delta t \cdot \omega_k \end{bmatrix}$$

- Kinematic model linearisation:

$$\mathbf{s}_{k+1} = \frac{\partial \mathbf{h}}{\partial \mathbf{s}_k} \cdot \mathbf{s}_k + \frac{\partial \mathbf{h}}{\partial \mathbf{U}_k} \cdot \mathbf{U}_k$$

- Kinematic model linearisation:

$$\mathbf{s}_{k+1} = \begin{bmatrix} 1 & 0 & -\Delta t \cdot v_k \cdot \sin(s_{\theta,k}) \\ 0 & 1 & \Delta t \cdot v_k \cdot \cos(s_{\theta,k}) \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} s_{x,k} \\ s_{y,k} \\ s_{\theta,k} \end{bmatrix} + \begin{bmatrix} \Delta t \cdot \cos(s_{\theta,k}) & 0 \\ \Delta t \cdot \sin(s_{\theta,k}) & 0 \\ 0 & \Delta t \end{bmatrix} \begin{bmatrix} v_k \\ \omega_k \end{bmatrix}$$

- In state space form

$$\mathbf{s}_{k+1} = \mathbf{A}_k \cdot \mathbf{s}_k + \mathbf{B}_k \cdot \mathbf{U}_k$$

Where:

$$\mathbf{A}_k = \begin{bmatrix} 1 & 0 & -v_k \cdot \sin(s_{\theta,k}) \\ 0 & 1 & -v_k \cdot \cos(s_{\theta,k}) \\ 0 & 0 & 1 \end{bmatrix} \cdot \Delta t$$

$$\mathbf{B}_k = \begin{bmatrix} \cos(s_{\theta,k}) & 0 \\ \sin(s_{\theta,k}) & 0 \\ 0 & 1 \end{bmatrix} \cdot \Delta t$$



To propagate  $\Sigma$  we use the following result:

## *Affine Transformation*

Consider  $X \sim \mathcal{N}(\mu_X, \Sigma_X)$  in  $\mathbb{R}^n$ , and let  $Y = AX + \mathbf{b}$  be the affine transformation, where  $\mathbf{A} \in \mathbb{R}^{m \times n}$  and  $\mathbf{b} \in \mathbb{R}^m$ . Then, the random vector  $Y \sim \mathcal{N}(\mu_Y, \Sigma_Y)$  such that:

$$\mu_Y = \mathbf{A}\mu_X + \mathbf{b}$$

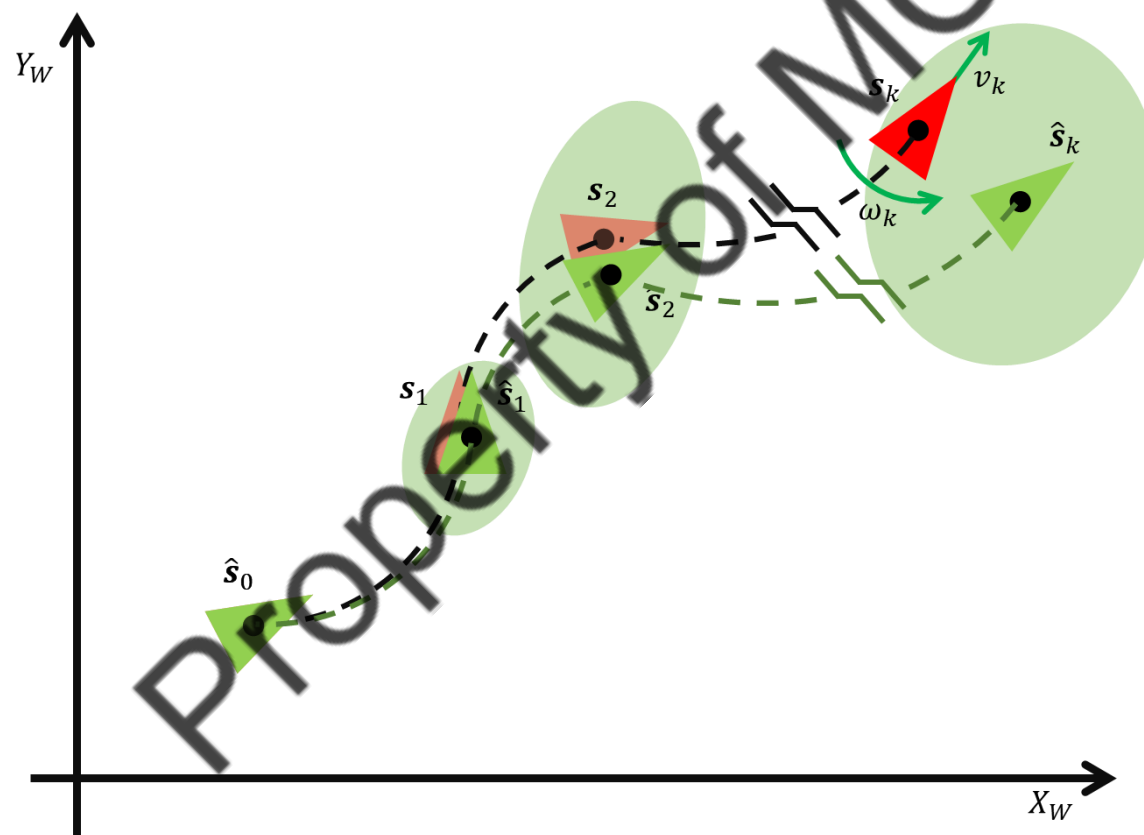
$$\Sigma_Y = \mathbf{A}\Sigma_X\mathbf{A}^T$$

Property of MCR2

- Since the robot motion model is linearised and all uncertainties are Gaussians, it is possible to compute the covariance  $\Sigma_k$  associated with the robot pose at time step  $k$  using the properties of Gaussians:
$$\Sigma_k = \mathbf{H}_k \Sigma_{k-1} \mathbf{H}_k^T + \mathbf{Q}_k$$
- Thus, the estimated pose at time step  $k$  is Gaussian such that  $\mathbf{s}_k \sim \mathcal{N}(\mu_k, \Sigma_k)$ , and it is computed recursively using the pose at time step  $k - 1$  and the input vector  $\mathbf{u}_k$ .
- The initial robot pose is assumed known such that  $\mu_k = \mathbf{0}$ , and  $\Sigma_k = \mathbf{0}$ .
- The pose uncertainty will always increase every time the robot moves due to the addition of the nondeterministic error represented by  $\mathbf{Q}_k$ , which is positive semi-definite.

# Model-based Localisation

Real/Estimated Robot





Property of MCR2

Example

# Pose covariance matrix $\mathbf{Q}_k$

- How can we obtain, experimentally, the covariance matrix  $\mathbf{Q}_k$ ???
- We do experiments with the real mobile robot (or Gazebo model).

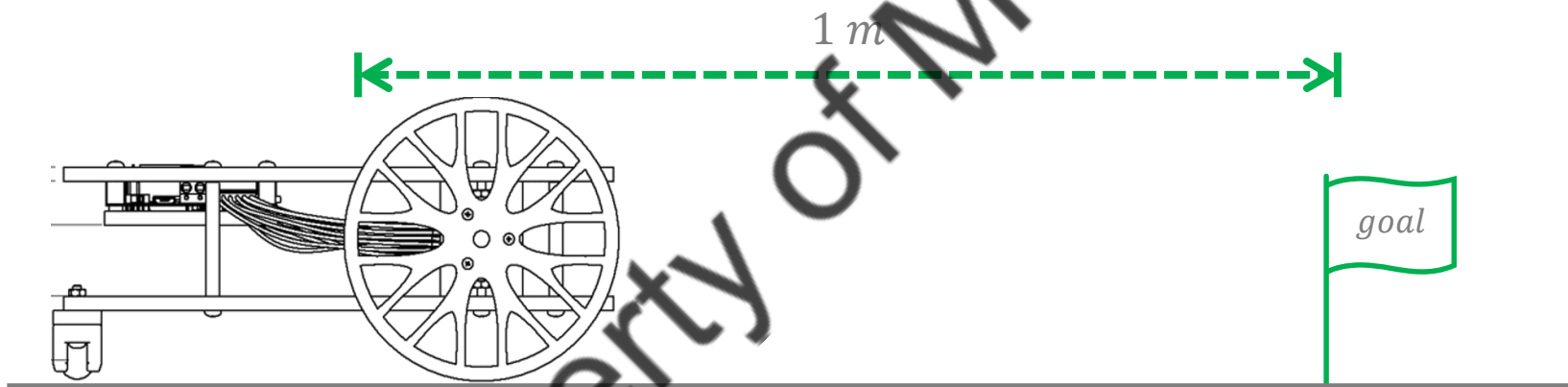
$$\Sigma_k = \mathbf{H}_k \Sigma_{k-1} \mathbf{H}_k^T + \mathbf{Q}_k$$

- **Method 1:**

- Setup in x direction the starting point and the goal (e.g. measure a 1m distance).
- For some selected speeds ( $v$  only), compute the time needed for the robot to go from the starting point to the goal.
- Drive the robot with the selected speed ( $v$  only) and for the computed time (open loop). Repeat the experiment for 10 times.
- Do the same for the  $\omega$  input only
- Save the end position of the robot for each experiment.
- Find the worst-case difference in x and y
- Compute the covariance  $\mathbf{Q}_k$

# Pose covariance matrix $Q_k$

- Setup in x direction the starting point and the goal (e.g. measure a 1m distance).



*Move the robot*

$N = 10$  #Number of repetitions

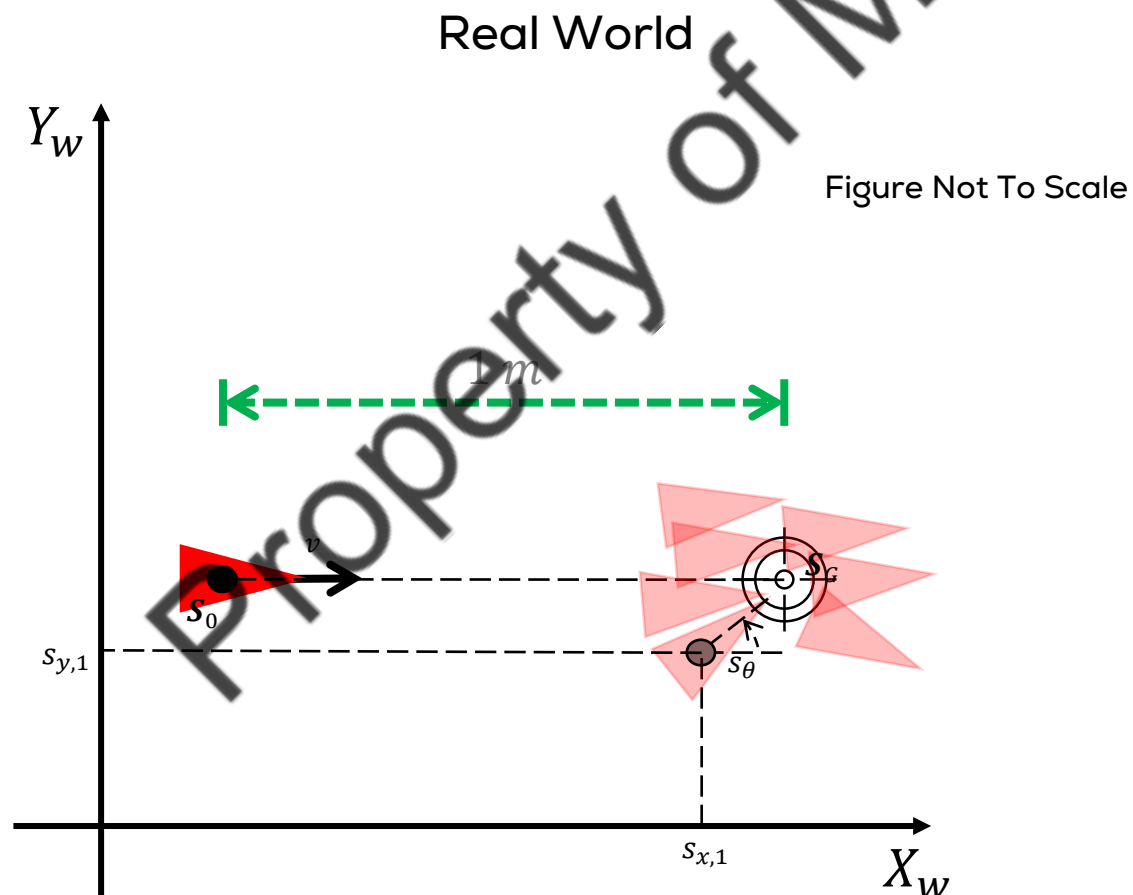


$$\begin{cases} v = 0.2 \frac{m}{s}, & t_1 = 5 s \\ v = 0.1 \frac{m}{s}, & t_2 = 10 s \\ v = 0.25 \frac{m}{s}, & t_3 = 4 s \end{cases}$$

- For some selected speeds ( $v$  only), compute the time needed for the robot to go from the starting point to the goal.

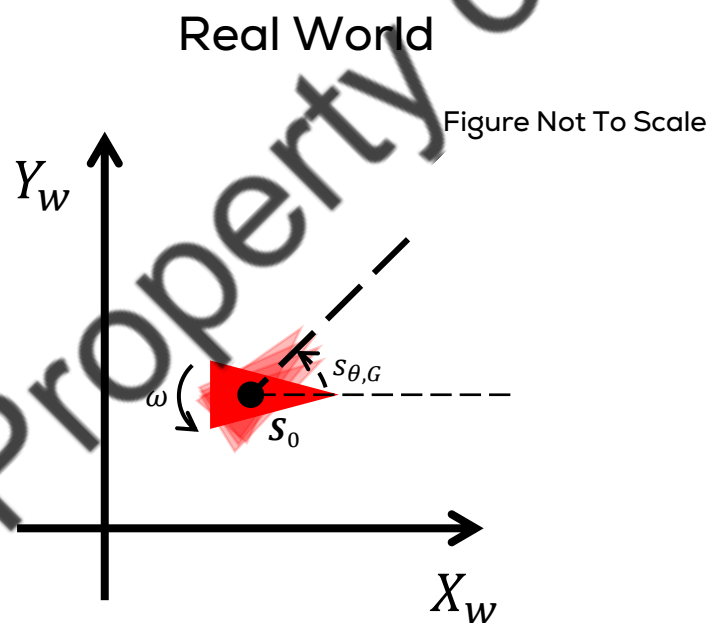
# Pose covariance matrix $Q_k$

- Drive the robot with the selected speed ( $v$  only) and for the computed time (open loop). Repeat the experiment for 10 times.
- Save the end position of the robot for each experiment.



# Pose covariance matrix $Q_k$

- Repeat the experiment for the input  $\omega$  only.



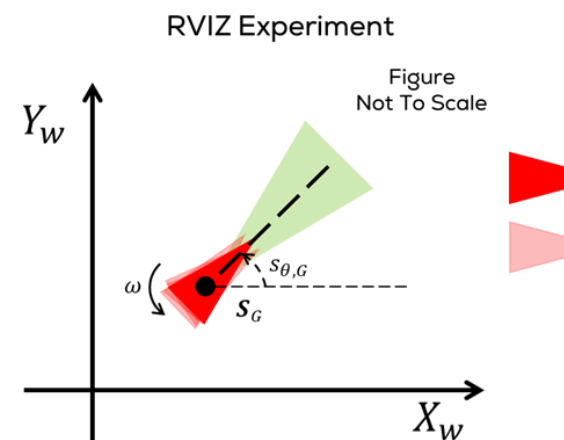
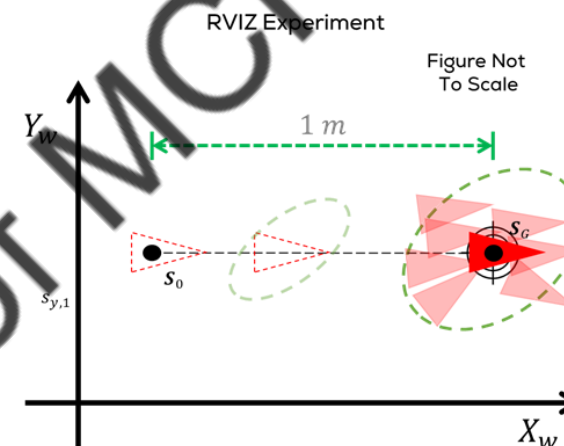


# Pose covariance matrix $Q_k$

- Tune the motion model covariance matrix.
- Use a software like RVIZ to verify if the parameters that you set are correct and compare the results to the real robot results

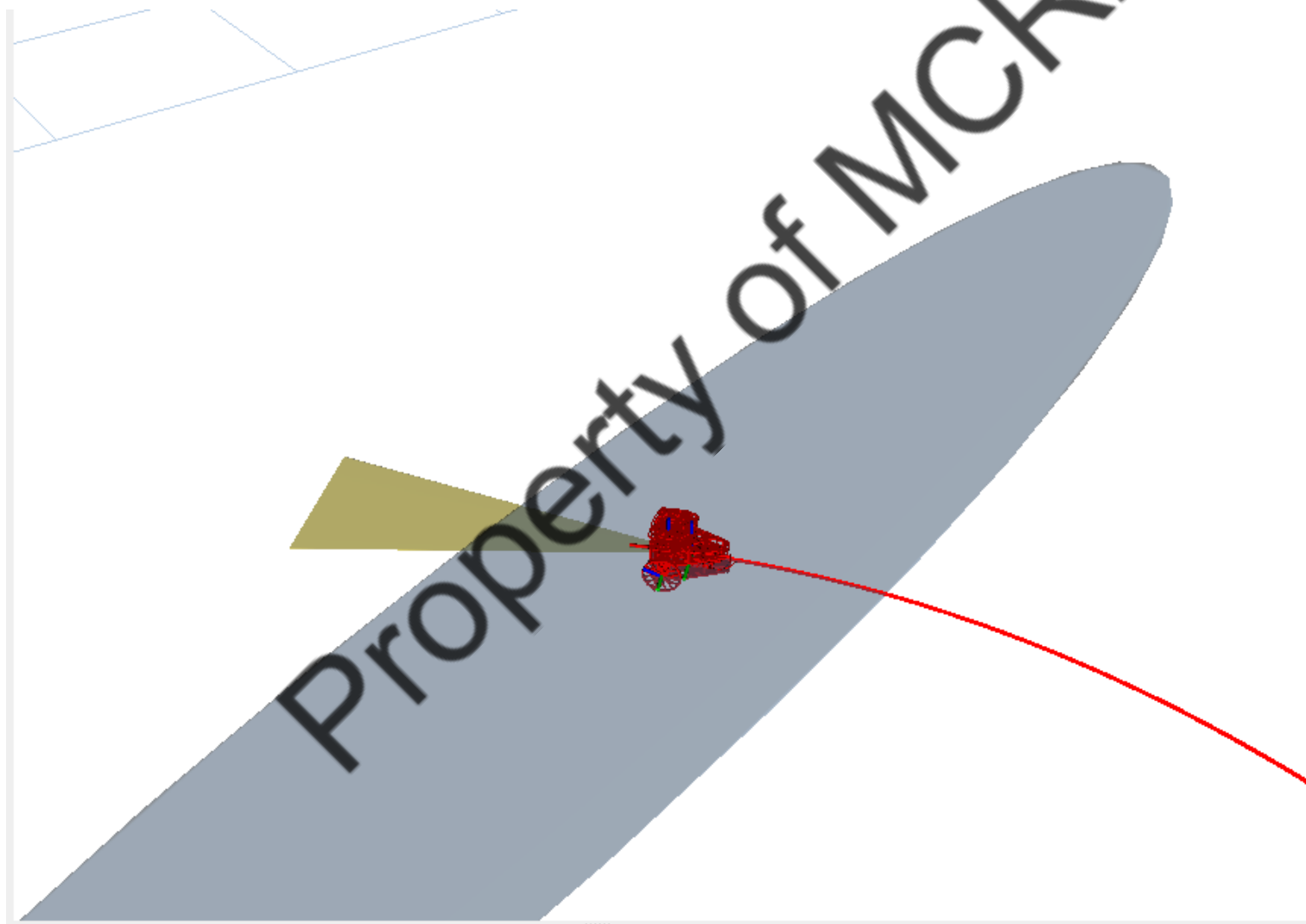
$$Q_k = \begin{bmatrix} 0.5 & 0.01 & 0.01 \\ 0.01 & 0.5 & 0.01 \\ 0.01 & 0.01 & 0.2 \end{bmatrix}$$

motion model covariance matrix





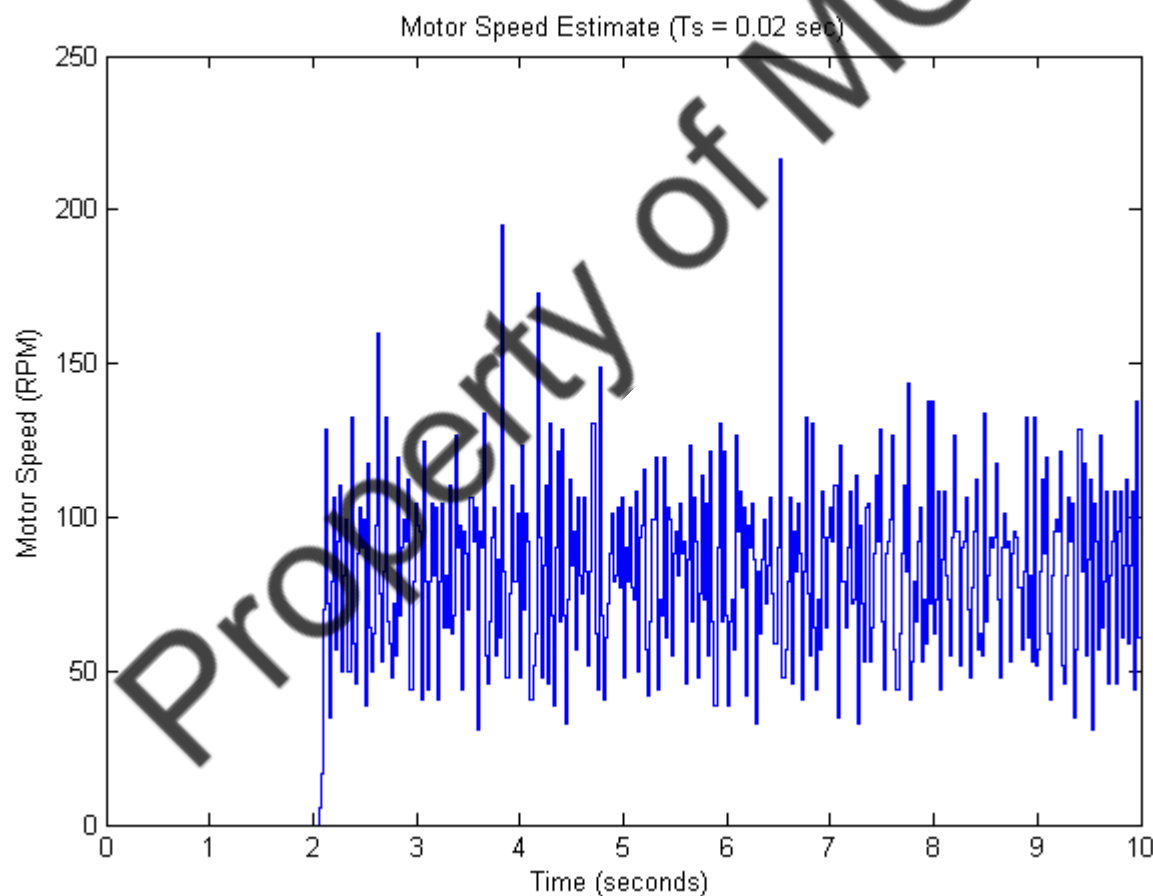
# Pose covariance matrix $Q_k$



# Pose covariance matrix $\mathbf{Q}_k$

## Method 2:

The Encoder noise:



# Pose covariance matrix $Q_k$ (2<sup>nd</sup> Method)

Consider the following motion model for a differential drive robot:

$$\mathbf{h}(\mathbf{s}_k, \omega_{r,k}, \omega_{l,k}) = \begin{bmatrix} s_{x,k-1} + r\Delta t \frac{\omega_{r,k} + \omega_{l,k}}{2} \cos(s_{\theta,k-1}) \\ s_{y,k-1} + r\Delta t \frac{\omega_{r,k} + \omega_{l,k}}{2} \sin(s_{\theta,k-1}) \\ s_{\theta,k-1} + r\Delta t \frac{\omega_{r,k} - \omega_{l,k}}{l} \end{bmatrix}$$

where  $\omega_{r,k}$  and  $\omega_{l,k}$  are the right and left wheel angular velocity at time step  $k$ .

# Pose covariance matrix $Q_k$ (2<sup>nd</sup> Method)

- Now assume the noise in both right and left wheel angular velocities to be zero-mean Gaussian distribution such that:

$$\begin{bmatrix} \omega_{r,k} \\ \omega_{l,k} \end{bmatrix} \sim \mathcal{N}(0, \Sigma_{\Delta,k}) \quad \Sigma_{\Delta,k} = \begin{bmatrix} k_r |\omega_{r,k}| & 0 \\ 0 & k_l |\omega_{l,k}| \end{bmatrix}$$

where  $k_r$  and  $k_l$  are constants representing the error associated with computing the angular velocity by each wheel.

- These constants are related to the traction between the wheels and the floor surface or the encoder noise used to compute the wheel displacements.
- Larger angular speed of the right motor  $|\omega_{r,k}|$  will lead to a larger variance of that motor  $k_r |\omega_{r,k}|$ .

# Pose covariance matrix $Q_k$ (2<sup>nd</sup> Method)

It is possible to propagate this noise  $\Sigma_{\Delta,k}$  to be seen from the robot state prospective using Taylor series expansion as follows:

$$Q_k = \nabla_{\omega_k} \cdot \Sigma_{\Delta,k} \cdot \nabla_{\omega_k}^T$$

$$\nabla_{\omega_k} = \begin{bmatrix} \frac{\partial h_1}{\partial \omega_{r,k}} & \frac{\partial h_1}{\partial \omega_{l,k}} \\ \frac{\partial h_2}{\partial \omega_{r,k}} & \frac{\partial h_2}{\partial \omega_{l,k}} \\ \frac{\partial h_3}{\partial \omega_{r,k}} & \frac{\partial h_3}{\partial \omega_{l,k}} \end{bmatrix} = \frac{1}{2} r \Delta t \begin{bmatrix} \cos(s_{\theta,k-1}) & \cos(s_{\theta,k-1}) \\ \sin(s_{\theta,k-1}) & \sin(s_{\theta,k-1}) \\ \frac{2}{l} & -\frac{2}{l} \end{bmatrix}$$