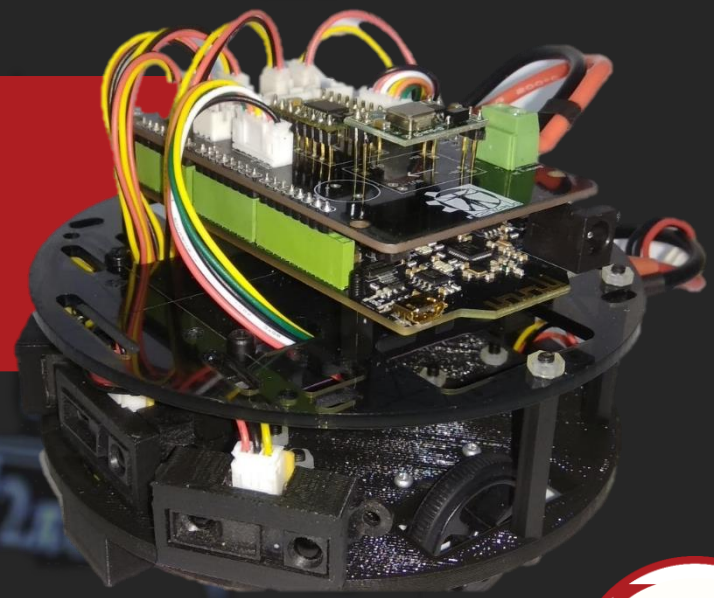
CRAFT #4

Mobile Robotics Programming

Kinematics & Control



Micael S. Couceiro micael@ingeniarius.pt

22/07/2024

CONTENT

- **Kinematics**
  - Initial concepts
    - Differential robots
- **Control**
  - Initial concepts
    - PID

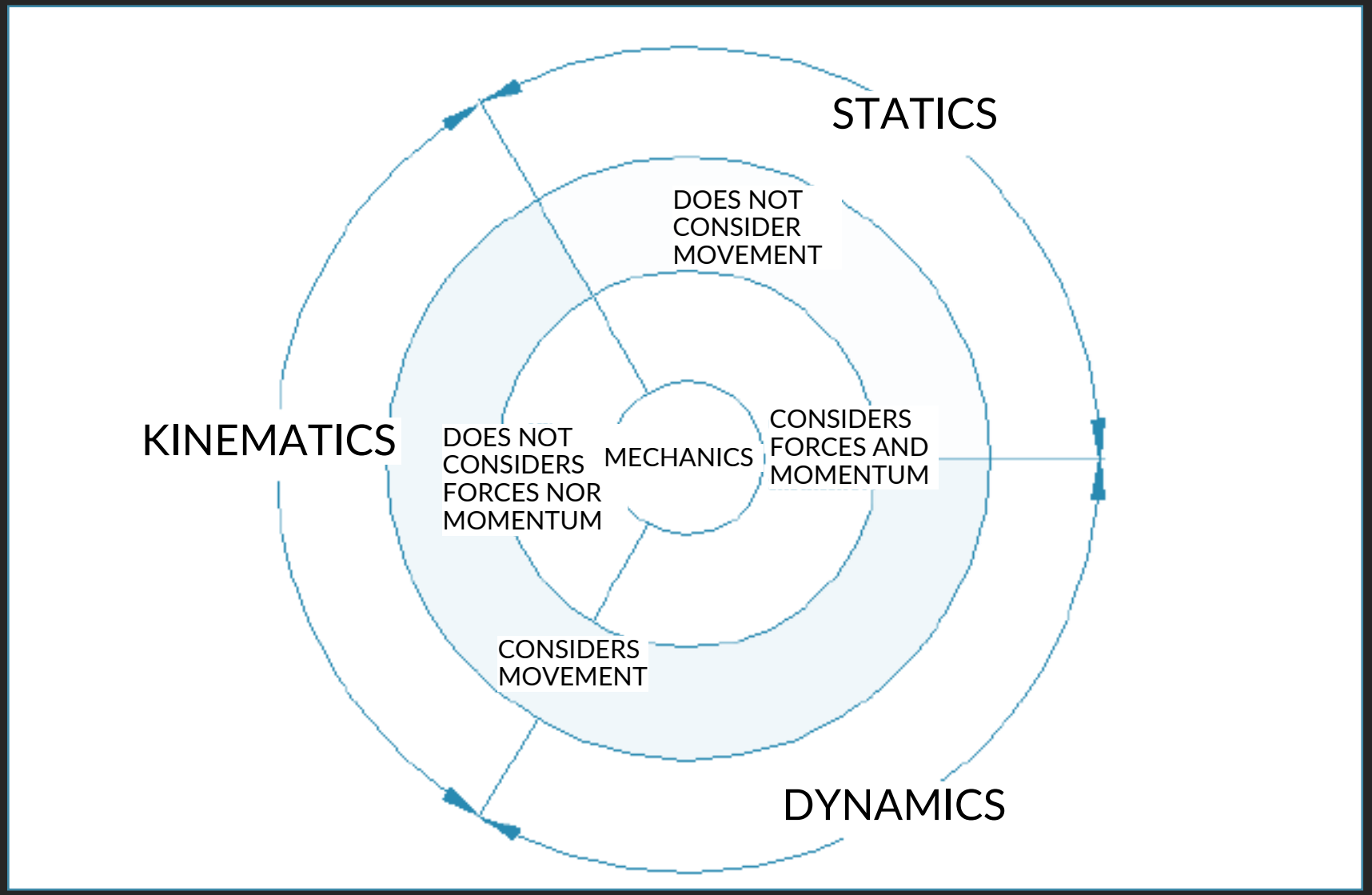
KINEMATICS

KINEMATICS

- Formal definition

> *"the branch of mechanics concerned with the motion of objects **without reference to the forces** which cause the motion."*

KINEMATICS


STATICS
DOES NOT CONSIDER MOVEMENT
KINEMATICS
DOES NOT CONSIDERS FORCES NOR MOMENTUM
MECHANICS
CONSIDERS FORCES AND MOMENTUM
CONSIDERS MOVEMENT
DYNAMICS

KINEMATICS

kinematics

The effect of a robot's geometry on its motion.

If the motors move ***this*** much, where will the robot be?

Assumes that we control ***encoder readings***…

dynamics

The effect of all forces (internal and external) on a robot's motion.

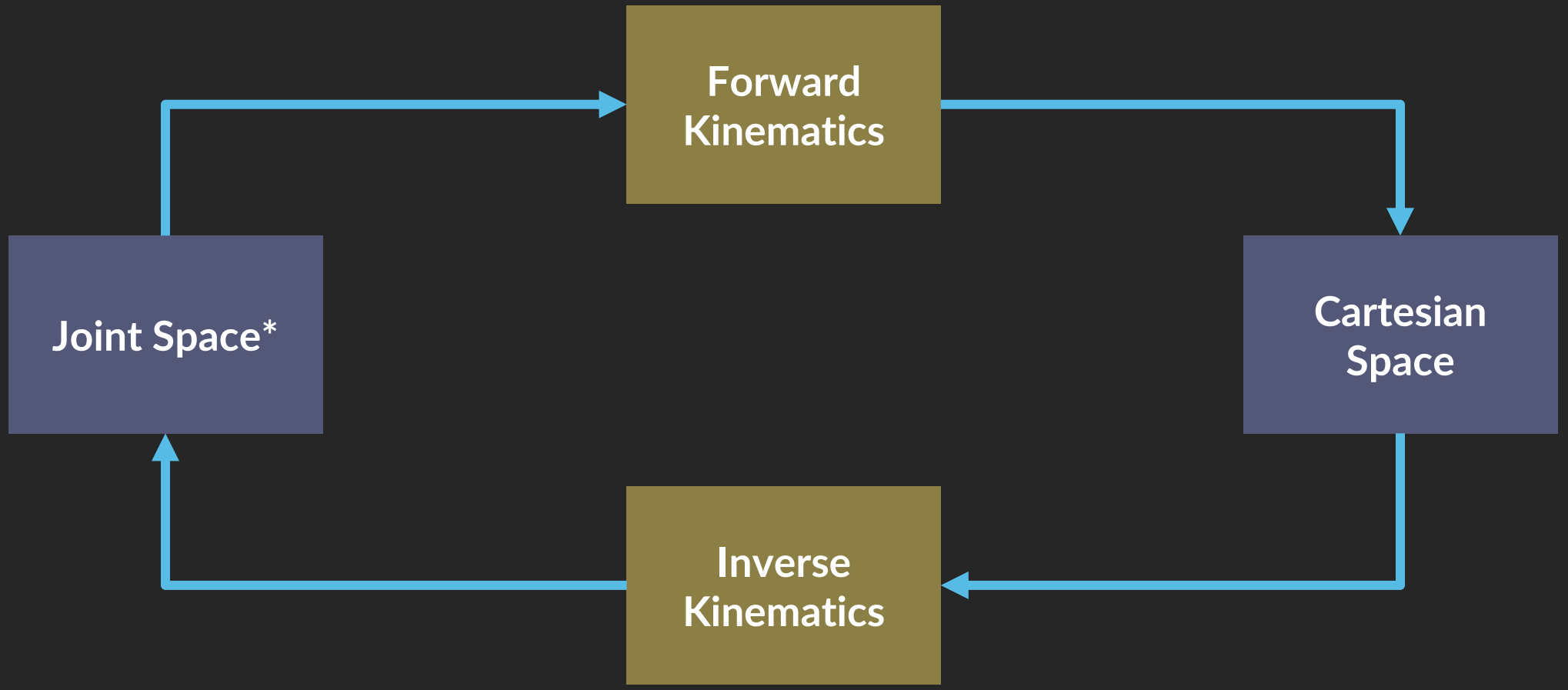
If the motors apply ***this*** much force, where will the robot be?

Assumes that we control ***motor current***…

KINEMATICS

- To localize a given rigid body in a 3D space, a reference coordinate system is associated to it (world reference or map reference)
- A reference associated to a rigid body is fixed on it – any point of the rigid body will have invariant coordinates in its own reference coordinate system
- The reference coordinate systems are defined by three orthogonal unitary vectors: $(\hat{x}, \hat{y}, \hat{z})$

KINEMATICS

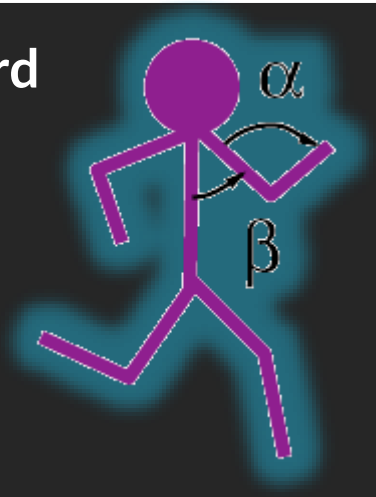


*A typical differential robot can be considered as a joint itself

KINEMATICS

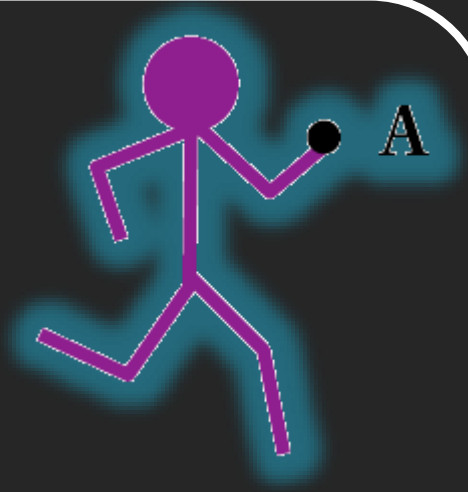
Kinematics

Forward



$A = f(\alpha, \beta)$

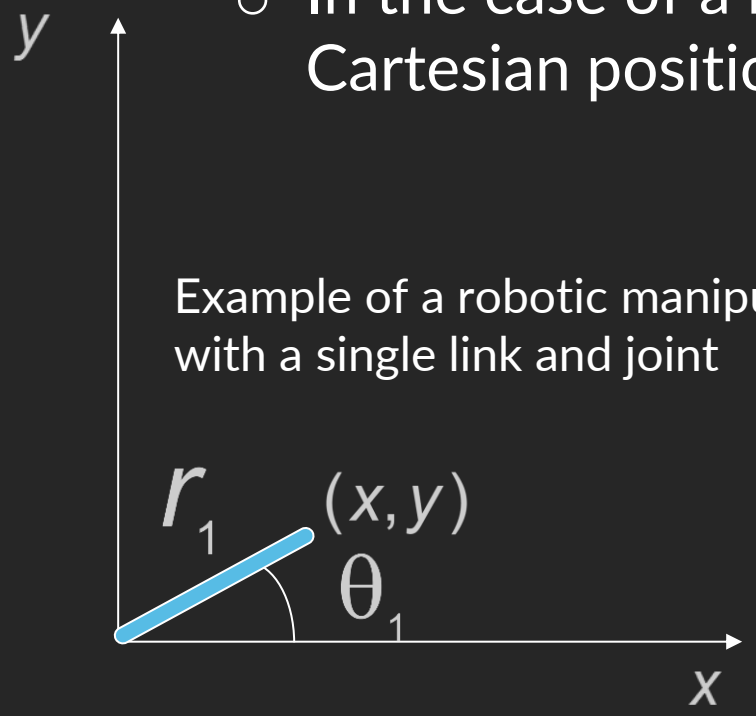
Inverse



$[\alpha, \beta] = f^{-1}(A)$

FORWARD KINEMATICS

- The forward kinematics allows to determine the Cartesian position of the links by knowing the angular orientation of the joints
- In the case of a mobile robot, it allows to determine its Cartesian position considering the rotation of the wheels



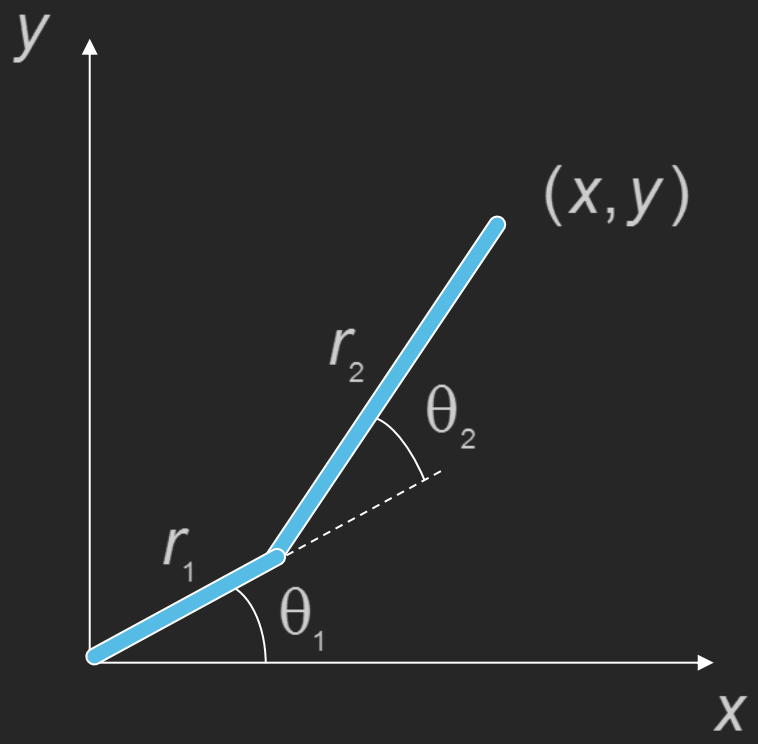
Example of a robotic manipulator with a single link and joint

$$x = r_1 \cos\theta_1$$

$$y = r_1 \sin\theta_1$$

FORWARD KINEMATICS

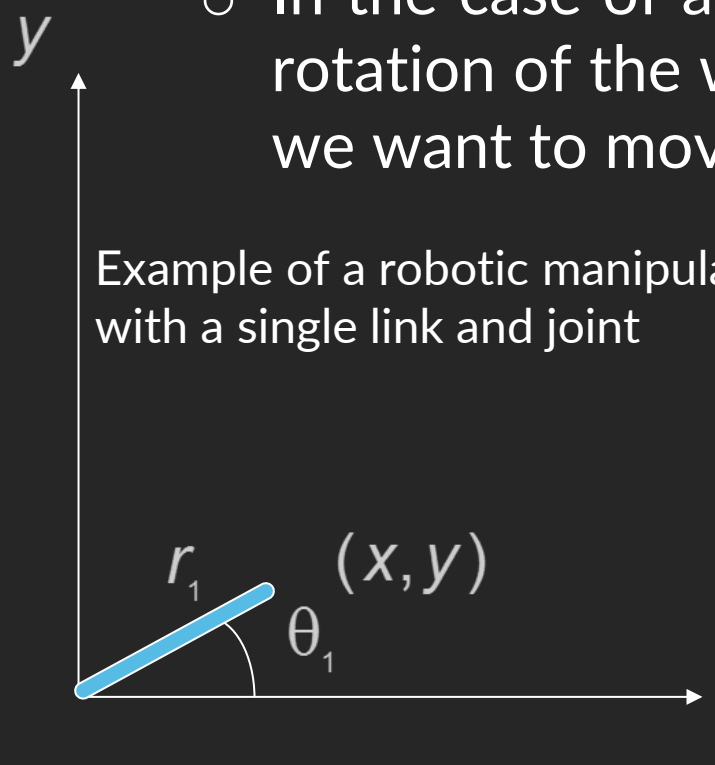
- How would it be for 2 links (and 2 joints)?



$$x = r_1 \cos\theta_1 + r_2 \cos(\theta_1 + \theta_2)$$
$$y = r_1 \sin\theta_1 + r_2 \sin(\theta_1 + \theta_2)$$

INVERSE KINEMATICS

- The inverse kinematics allows to determine the angular rotation of joints if one knows the Cartesian position of the link
- In the case of a mobile robot, it allows to determine the rotation of the wheels considering the Cartesian position we want to move the robot to (target position)



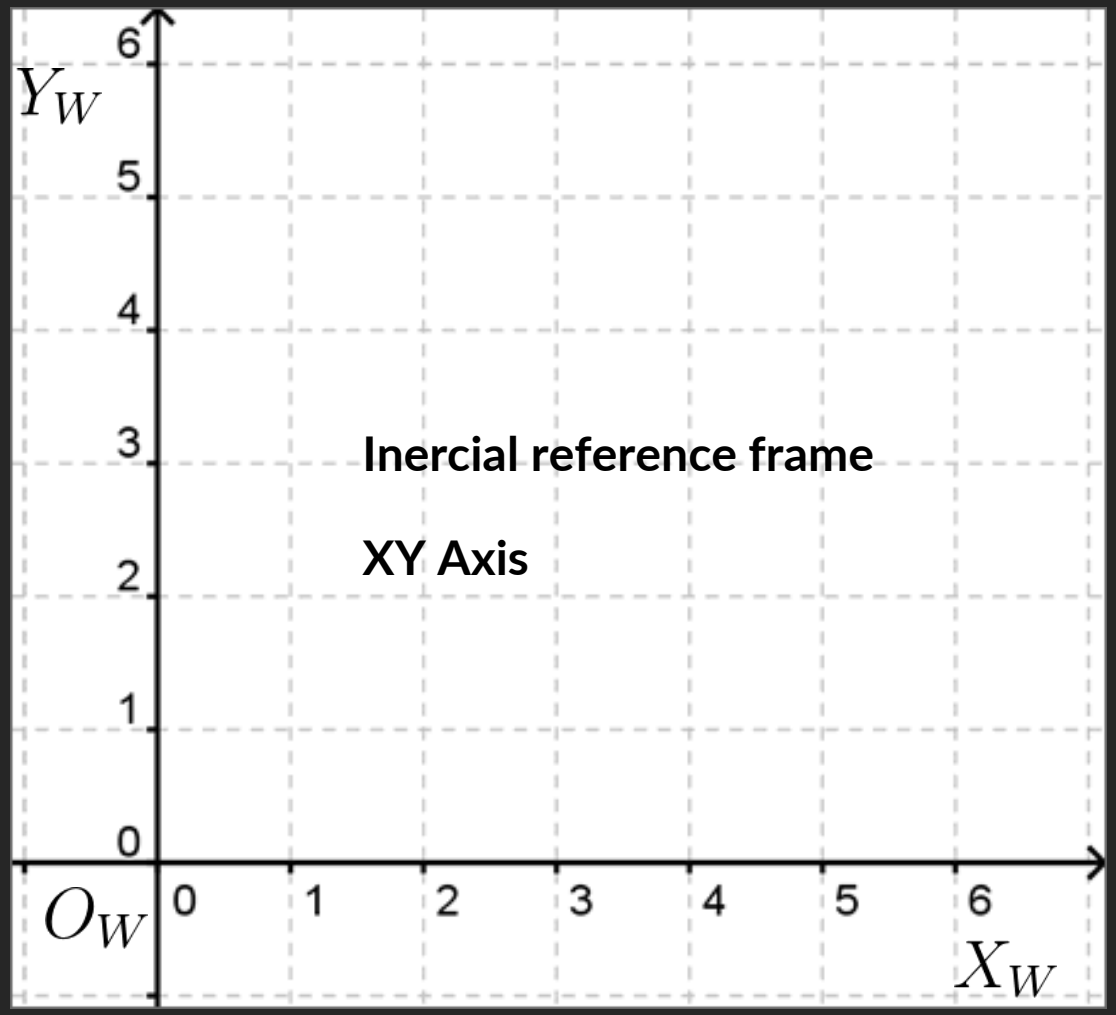
Example of a robotic manipulator with a single link and joint

$$\left.\begin{aligned} x &= r_1 \cos\theta_1 \\ y &= r_1 \sin\theta_1 \end{aligned}\right\} \Rightarrow \theta_1 = \operatorname{atan2}(y, x), \quad r = \sqrt{x^2 + y^2}$$

BUT HOW TO DEVELOP THE KINEMATIC MODEL OF A MOBILE ROBOT?

This is where odometry kicks in…

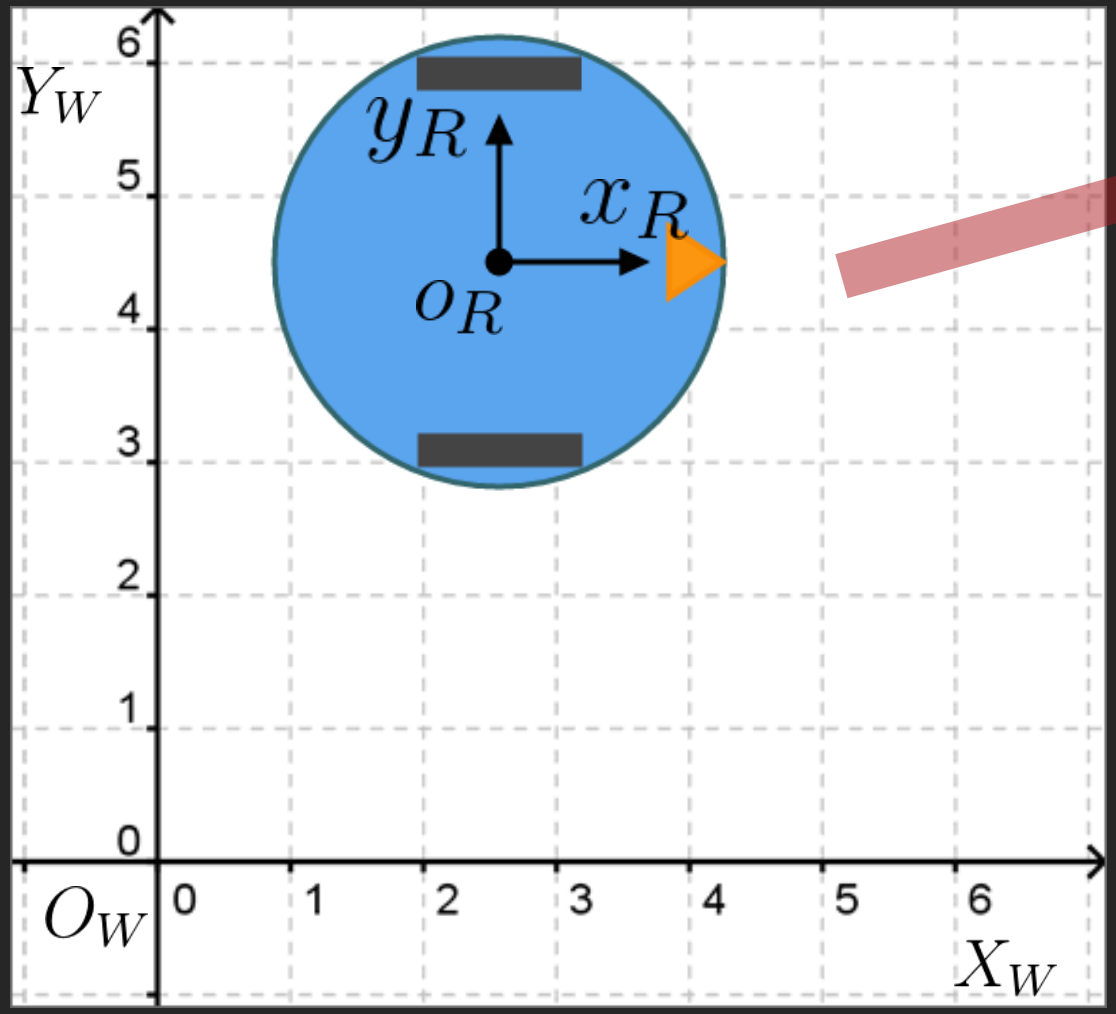
DIFFERENTIAL ROBOT



Workspace
Environment
World
Map

/map TF

DIFFERENTIAL ROBOT

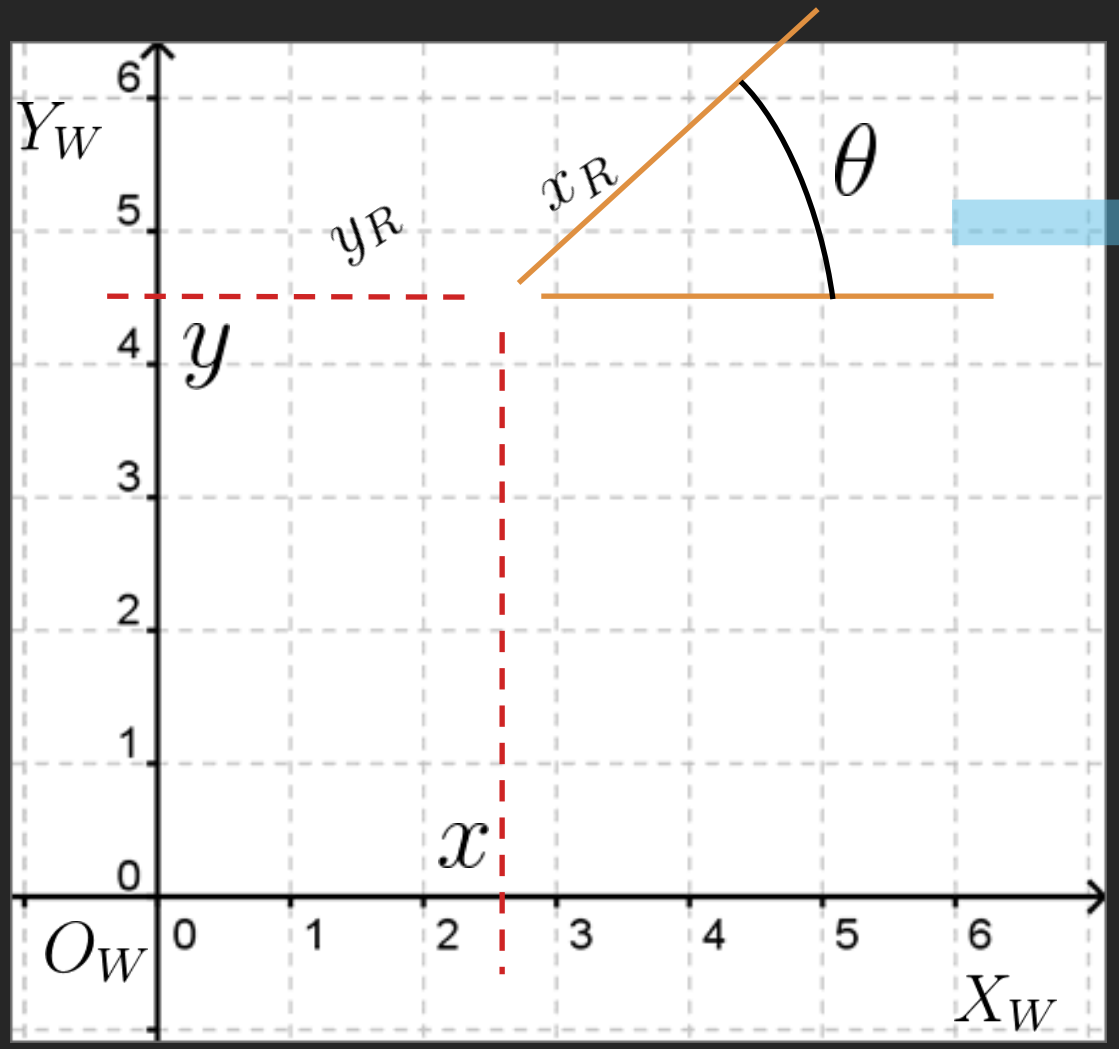


ROBOT

moving reference frame

/base_link TF

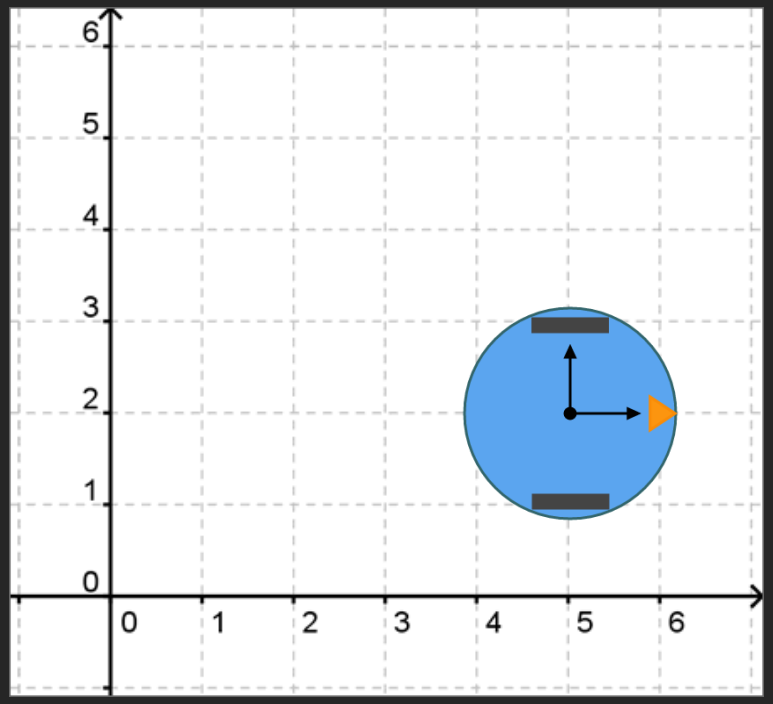
DIFFERENTIAL ROBOT

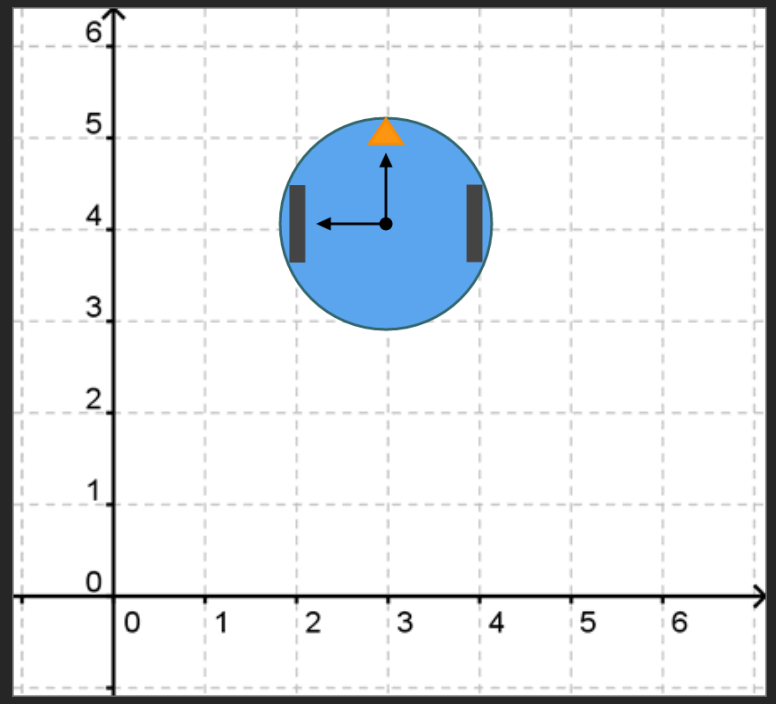


ROBOT
Pose

$$P_R = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix}$$

DIFFERENTIAL ROBOT

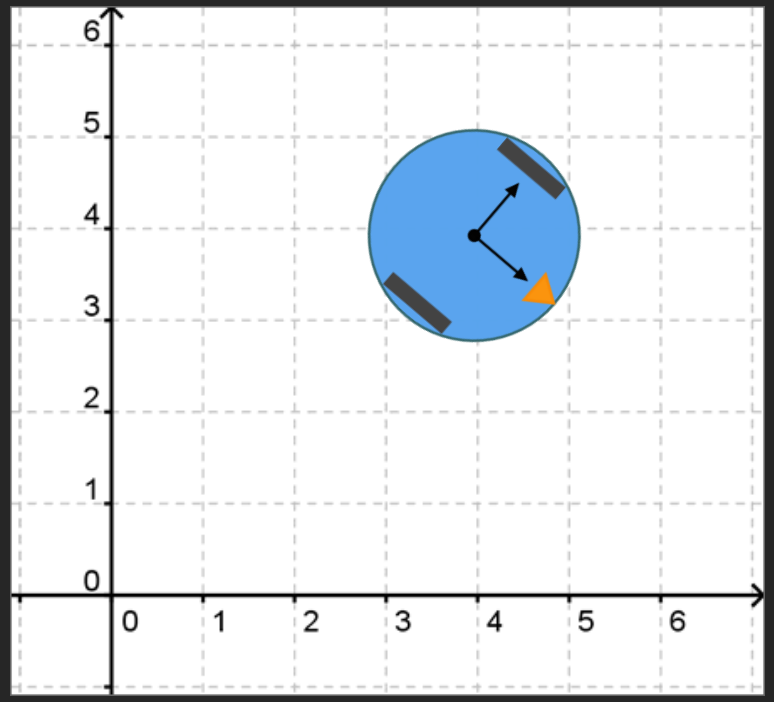


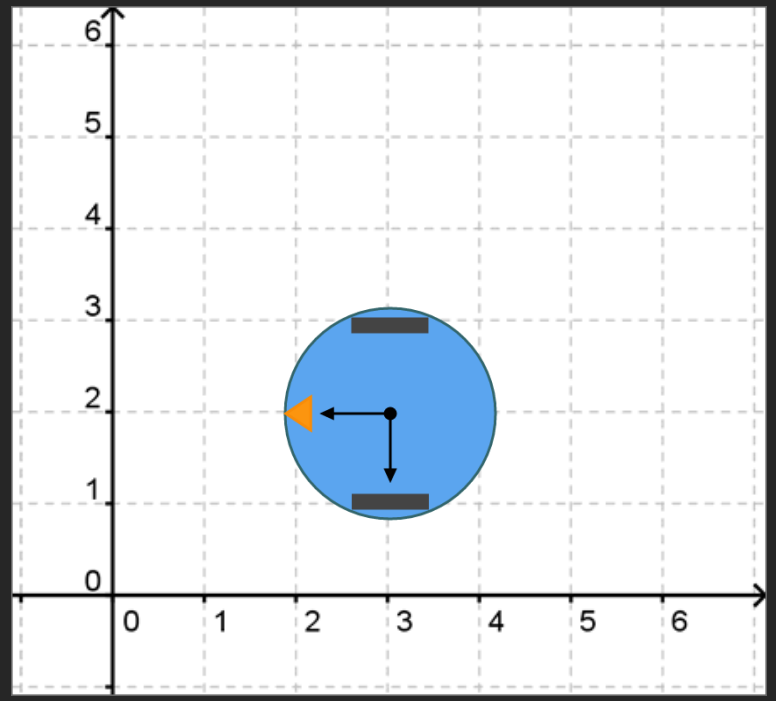


$$P_R = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix} = \begin{bmatrix} 5 \\ 2 \\ 0 \end{bmatrix}$$

$$P_R = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix} = \begin{bmatrix} 3 \\ 4 \\ \pi/2 \end{bmatrix}$$

DIFFERENTIAL ROBOT

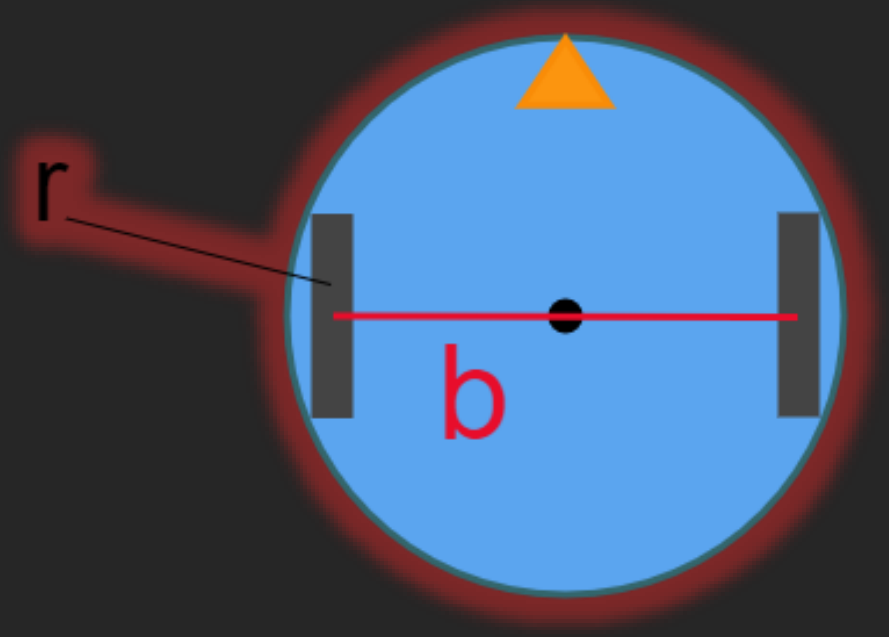




$$P_R = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix} = \begin{bmatrix} 4 \\ 4 \\ -\pi/4 \end{bmatrix}$$

$$P_R = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix} = \begin{bmatrix} 3 \\ 2 \\ -\pi \end{bmatrix} = \begin{bmatrix} 3 \\ 2 \\ \pi \end{bmatrix}$$

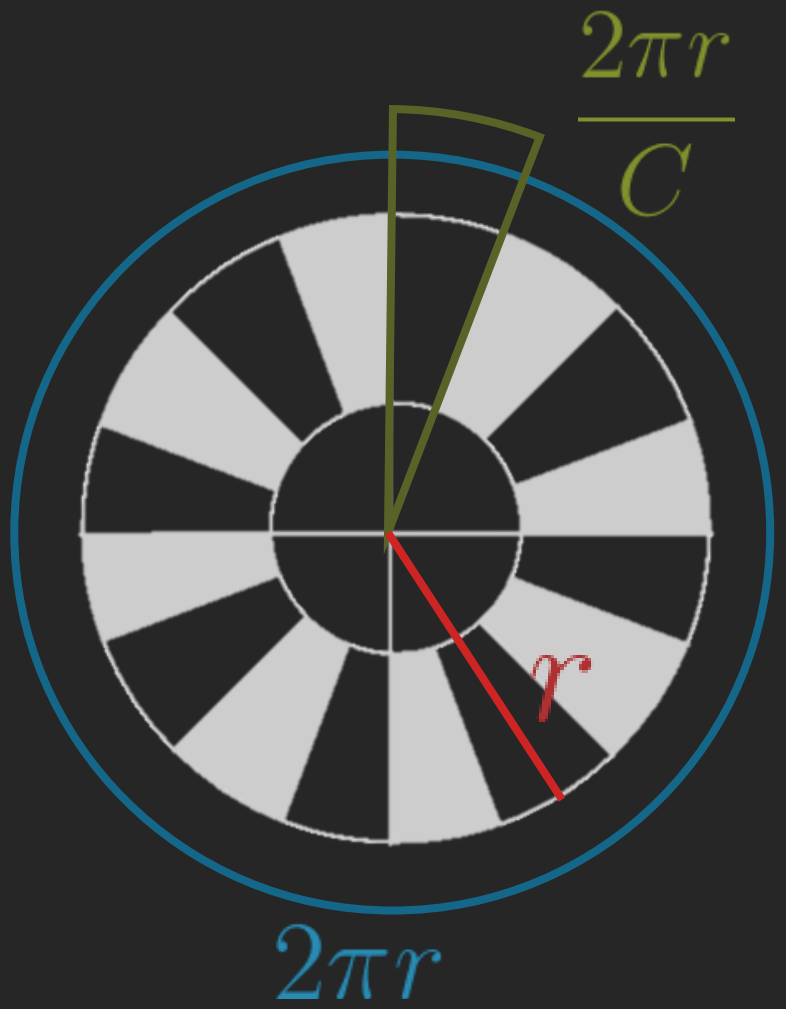
DIFFERENTIAL ROBOT



r Radius of wheels

b Wheel-to-wheel Distance

DIFFERENTIAL ROBOT



r Radius of wheels

$2\pi r$ Perimeter of wheels

C Encoder Resolution
(Pulses per revolution)

$$D = \frac{2\pi r}{C}$$ Linear wheel displacement
For 1 pulse

DIFFERENTIAL ROBOT

How can we know what is our position, given some particular movement?

Odometry typically uses encoders to measure wheel rotation and/or steering orientation*

Totally self-contained as it is always capable of providing the vehicle with an estimate of its position

The position error grows without bound unless an independent/external reference is used periodically to reduce the error

*Actually, that's only one of the ways to obtain odometry – there are other ways to have an estimation of the odometry by employing other technologies (e.g., LIDARs) and methods (e.g., SLAM).

DIFFERENTIAL ROBOT

Displacement of the Wheels

$$D_L = \frac{2\pi r}{C_L} \times N_L$$

Displacement of Left Wheel

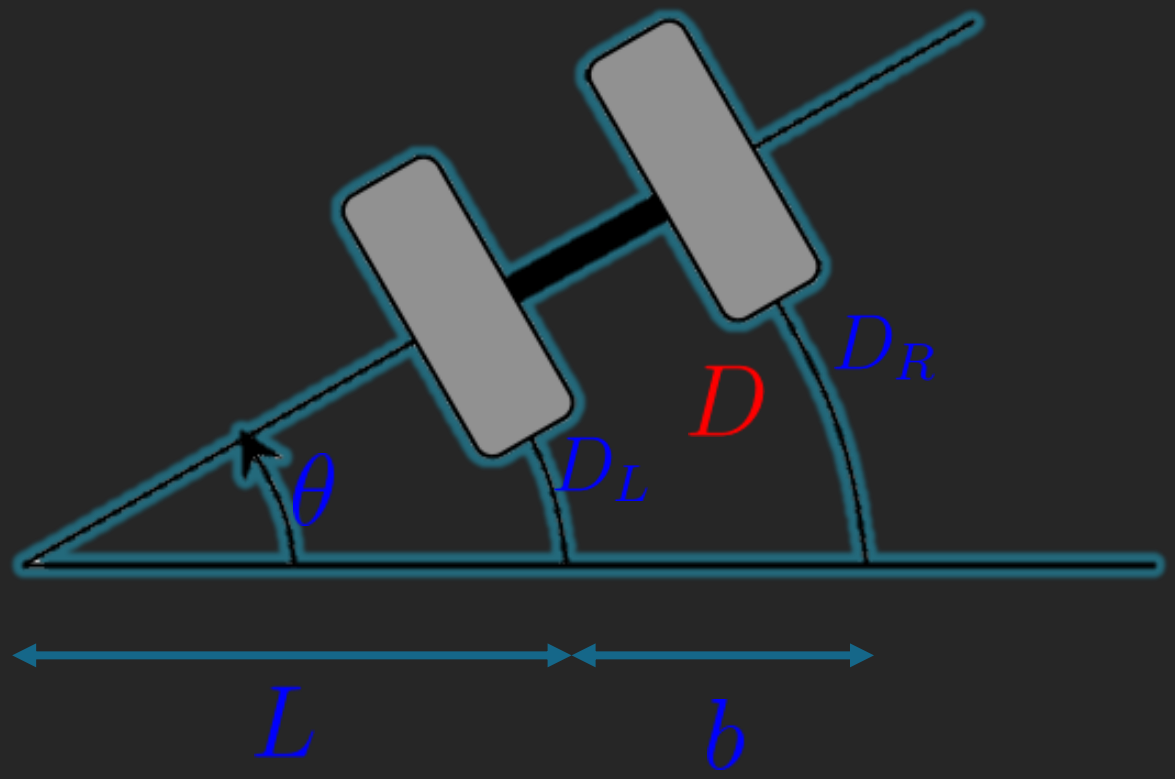
$$D_R = \frac{2\pi r}{C_R} \times N_R$$

Displacement of Right Wheel

$N_{L/R}$ Number of Pulses

DIFFERENTIAL ROBOT

Linear Displacement of the Robot



$$D = \frac{D_L + D_R}{2}$$

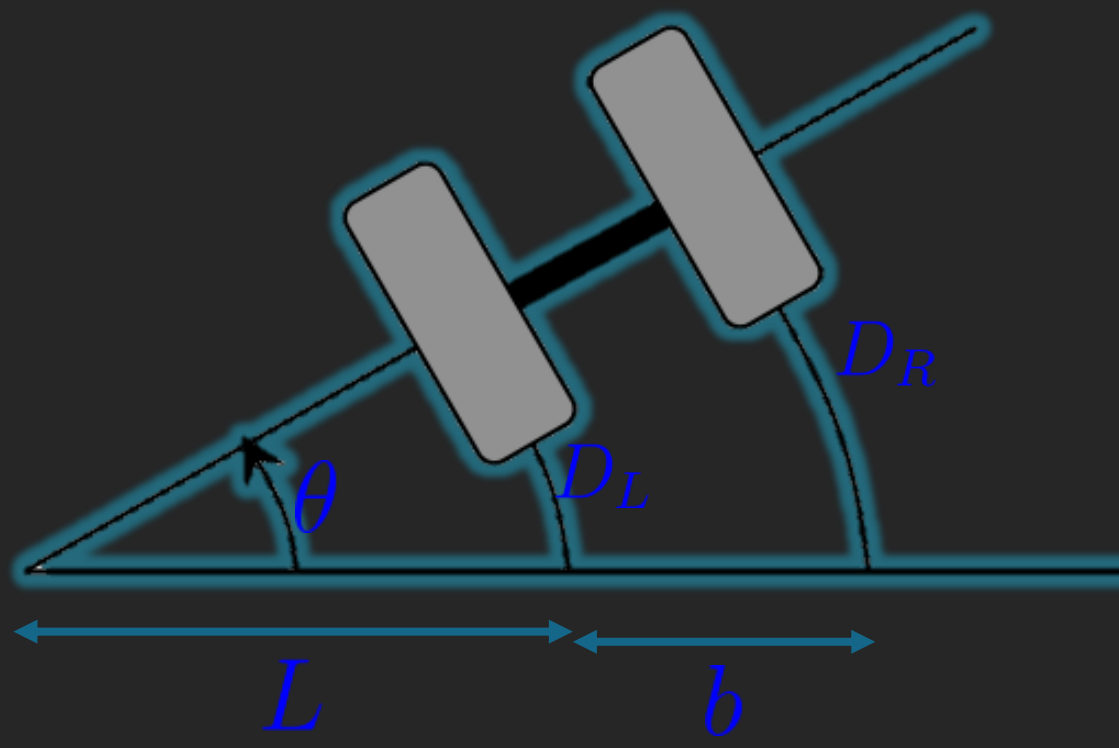
DIFFERENTIAL ROBOT

Change in Orientation

$$D_L = \theta L \Leftrightarrow L = \frac{D_L}{\theta}$$

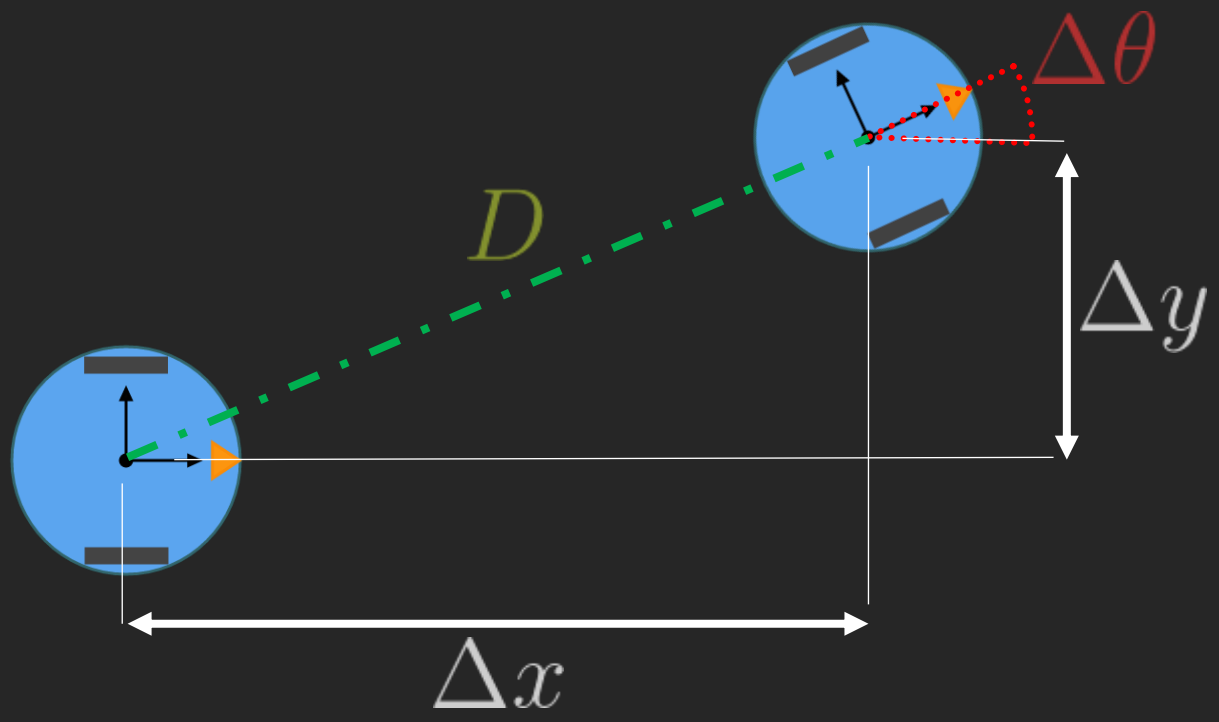
$$D_R = \theta(L + b) \Leftrightarrow \theta = \frac{D_R}{L + b}$$

$$\theta = \frac{D_R}{\frac{D_L}{\theta} + b} \Leftrightarrow \theta = \frac{D_R - D_L}{b}$$



DIFFERENTIAL ROBOT

Approximation!



$$P_{i+1} = \begin{bmatrix} x_{i+1} \\ y_{i+1} \\ \theta_{i+1} \end{bmatrix} = \begin{bmatrix} x_i + \Delta x \\ y_i + \Delta y \\ \theta_i + \Delta\theta \end{bmatrix}$$

$$\Delta\theta = \frac{\mathbf{D_R} - \mathbf{D_L}}{\mathbf{b}}$$

$$\Delta\mathbf{x} = \mathbf{D} \cdot \mathbf{cos}(\theta)$$

$$\Delta\mathbf{y} = \mathbf{D} \cdot \mathbf{sin}(\theta)$$

DIFFERENTIAL ROBOT

Algorithm Calculate Pose:

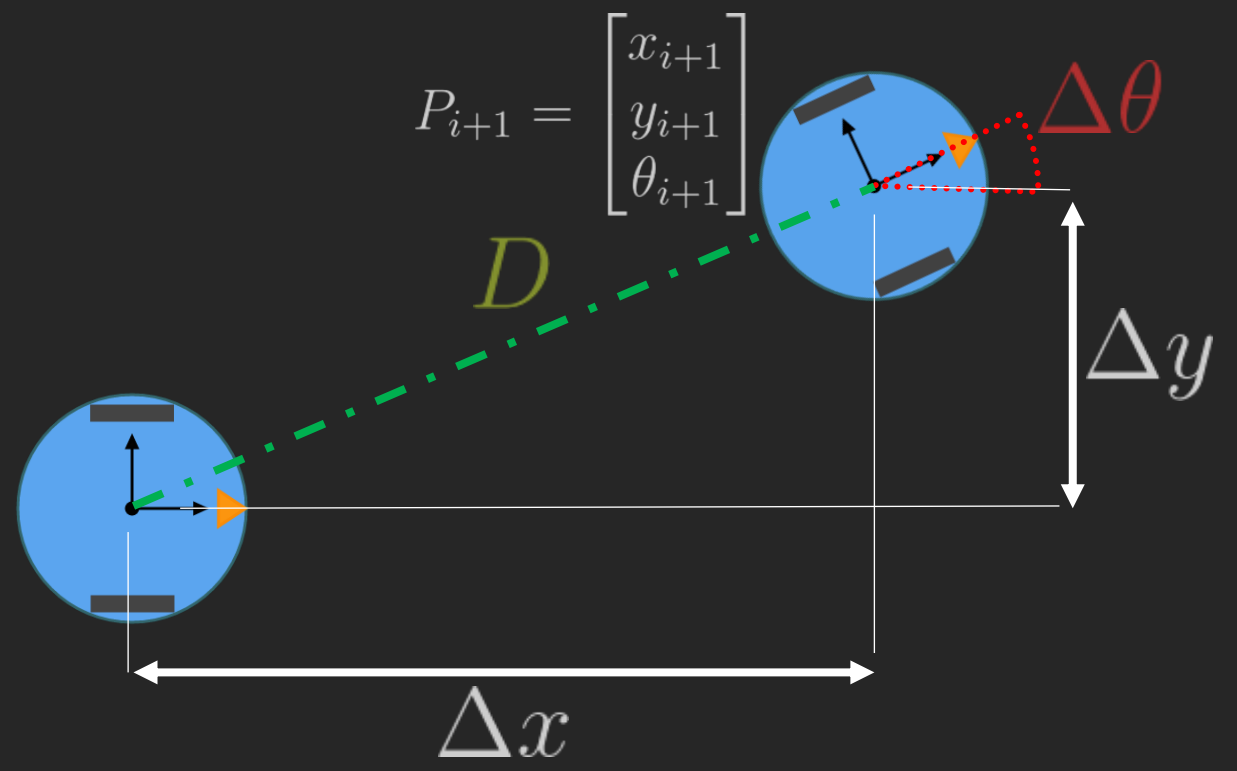
1. Get the number of encoder pulses since last iteration
2. Calc linear distance performed by each wheel
3. Calc linear distance performed by the robot
4. Calc the change in orientation
5. Update values in the following order

$N_{L/R}$

$D_{L/R}$

D

$\Delta\theta$



$$\theta_{i+1} = \theta_i + \frac{D_R - D_L}{b}$$

$$x_{i+1} = x_i + D \cdot cos(\theta)$$

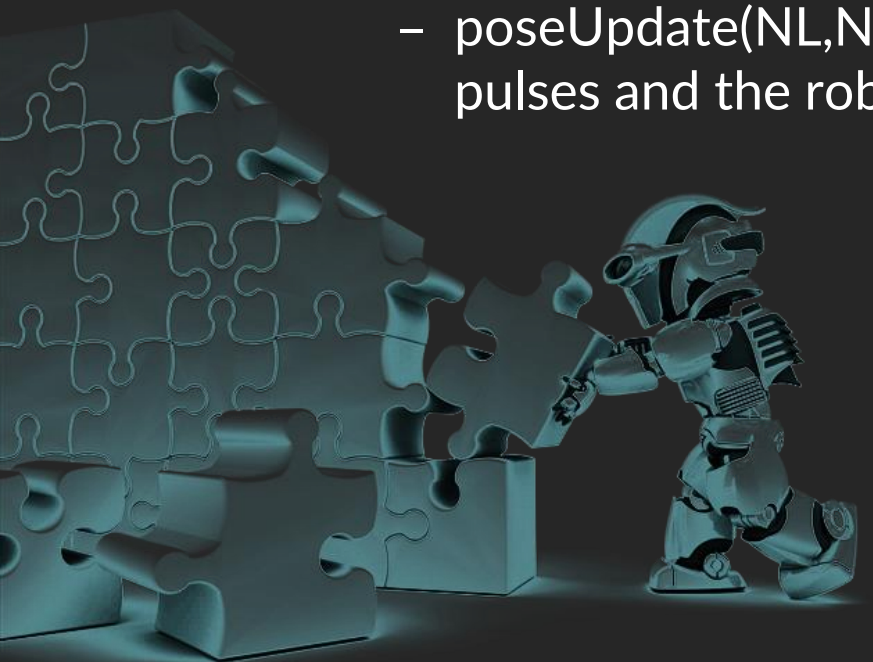
$$y_{i+1} = y_i + D \cdot sin(\theta)$$

KINEMATICS – CHALLENGE #1

- Follow the sequence previously established and find out where the robot will be after 10 steps, considering that:
  - The robot starts at the origin (0, 0), with $r = 5$ cm, $b = 20$ cm, $C = 100$
  - The encoders provide the following readings at each iteration:
    1. $N_{LT} = 0$ & $N_{RT} = 0$
    2. $N_{LT} = 100$ & $N_{RT} = 100$
    3. $N_{LT} = 300$ & $N_{RT} = 200$
    4. $N_{LT} = 500$ & $N_{RT} = 300$
    5. $N_{LT} = 700$ & $N_{RT} = 400$
    6. $N_{LT} = 900$ & $N_{RT} = 500$
    7. $N_{LT} = 1000$ & $N_{RT} = 600$
    8. $N_{LT} = 1100$ & $N_{RT} = 700$
    9. $N_{LT} = 1200$ & $N_{RT} = 800$
    10. $N_{LT} = 1200$ & $N_{RT} = 800$

KINEMATICS – CHALLENGE #1

- In Arduino, implement the algorithm to calculate the forward kinematics of the differential robot and evaluate it considering the given example. Verify your maths.
  - encUpdate(t): Get the current number of encoder pulses (load each element of the given 10 × 2 matrix, at each iteration t) and return its difference
  - poseUpdate(NL,NR,r,b,C): Returns new pose of the robot based on encoder pulses and the robot's physical properties



ARDUINO™

IS THERE ANY USE FOR THE INVERSE KINEMATICS IN THIS CASE?

Yep. Move the robot from point A to point B is always useful (even though you can benefit from ROS' navigation stack for that purpose)

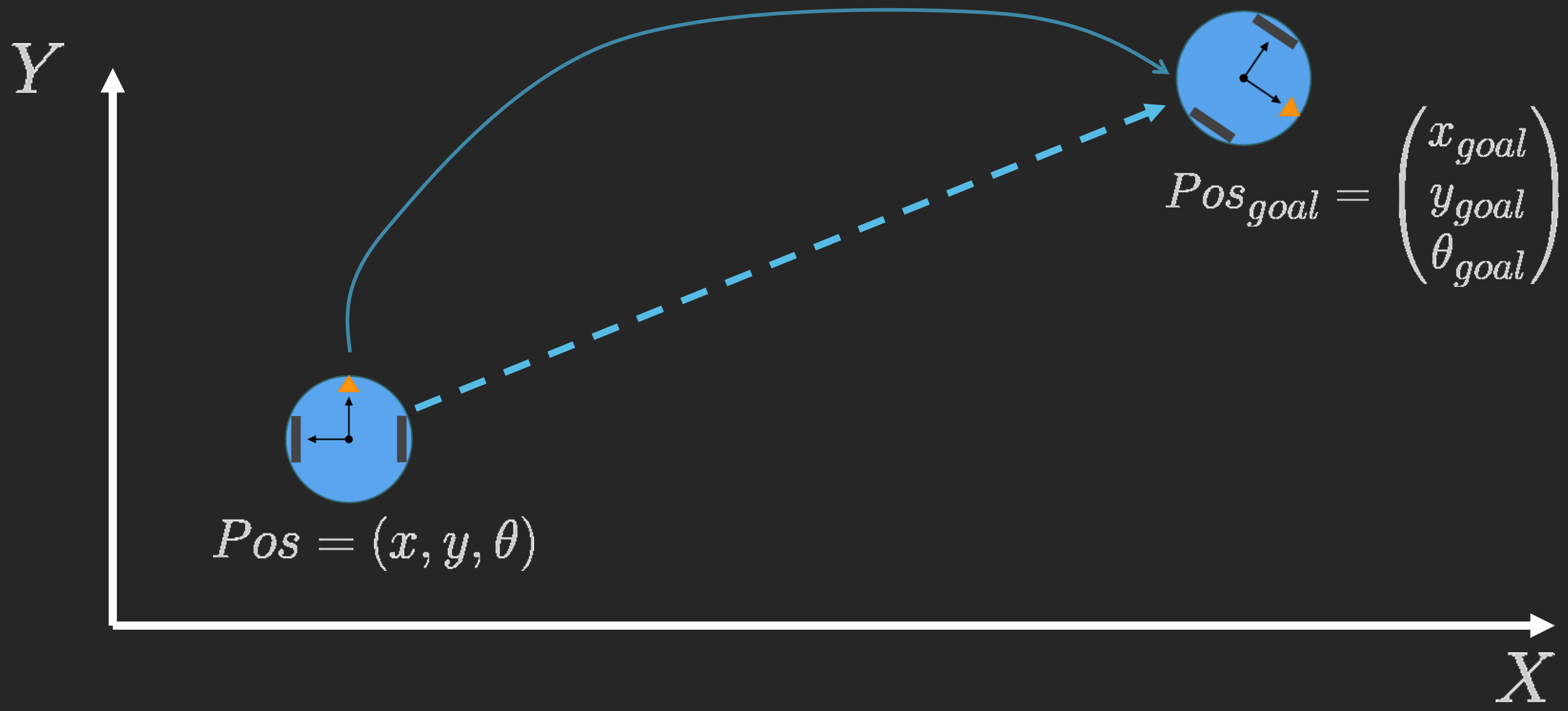
DIFFERENTIAL ROBOT

Inverse Kinematics: what wheels speeds do I need to perform the movements that I want?

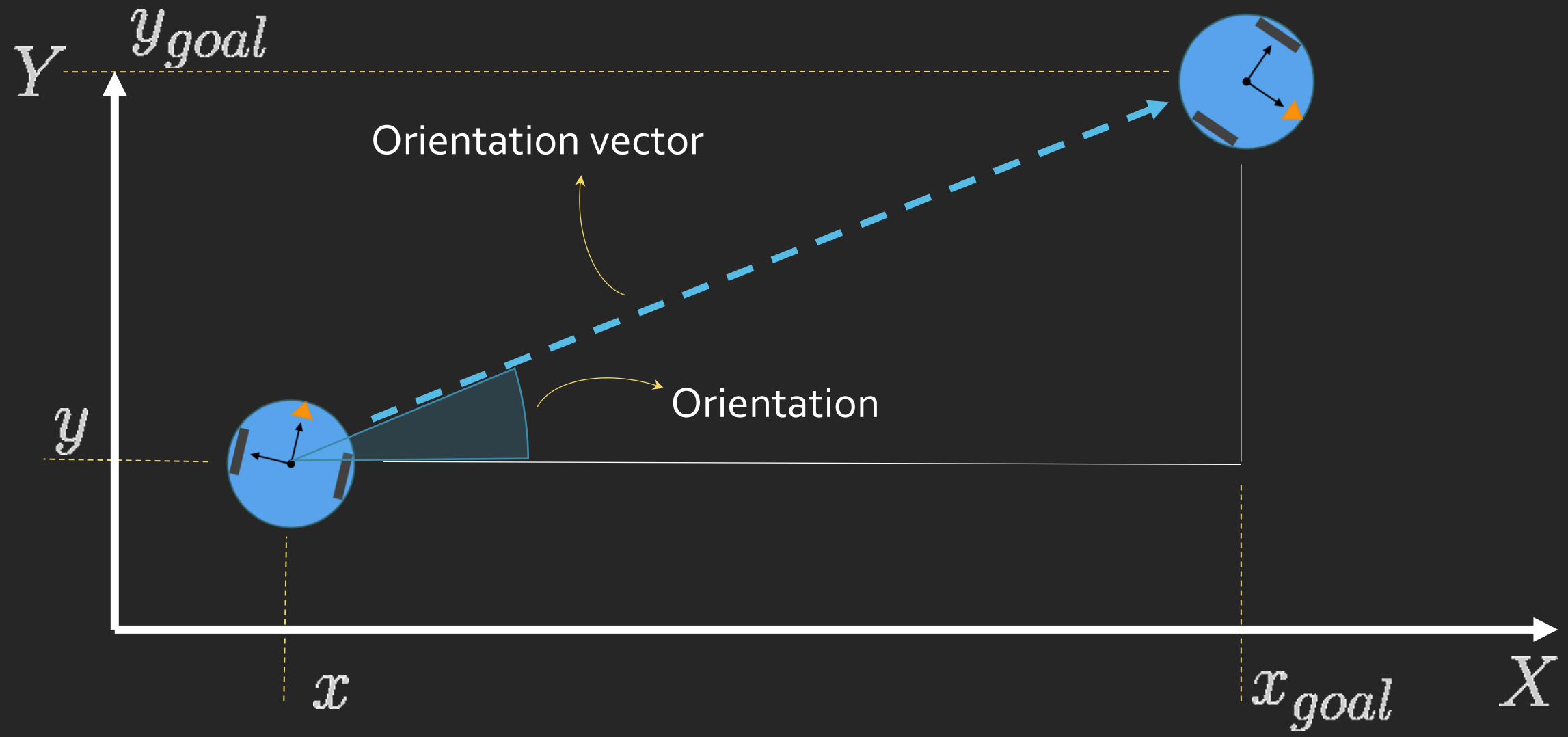
With Odometry, we can now estimate, at every instant, our **Position (x,y,θ).**

Now, considering our current Position, what movements do I need to perform in order to go to a **Goal Position**?

DIFFERENTIAL ROBOT



DIFFERENTIAL ROBOT

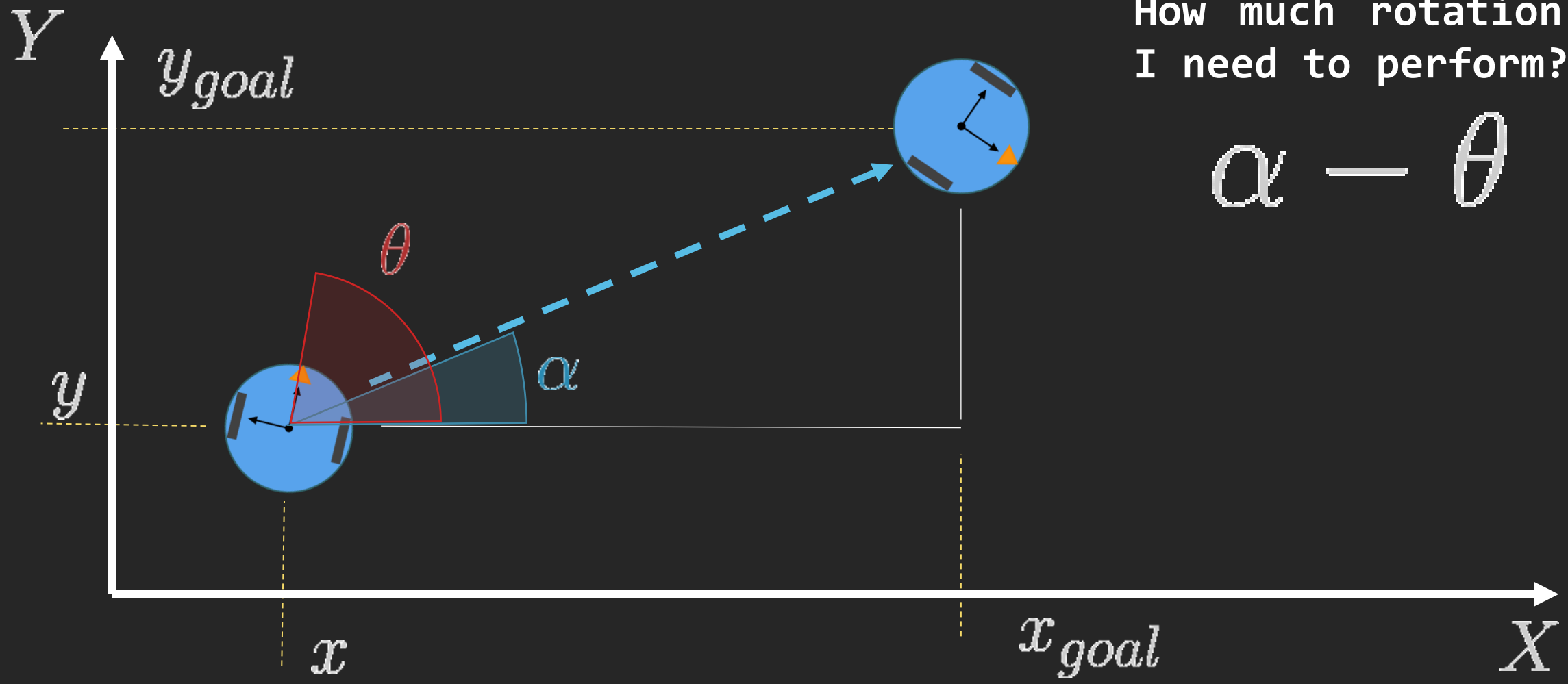


DIFFERENTIAL ROBOT

$$tan(\alpha) = \frac{opposite}{adjacent} \rightarrow tan(\alpha) = \frac{y_{goal} - y}{x_{goal} - x}$$

$$\rightarrow \alpha = \arctan 2 \left(\frac{y_{goal} - y}{x_{goal} - x} \right)$$

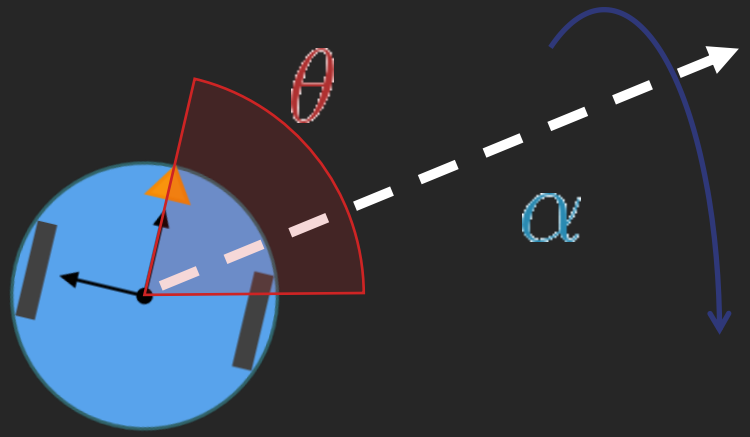
DIFFERENTIAL ROBOT



How much rotation do I need to perform?

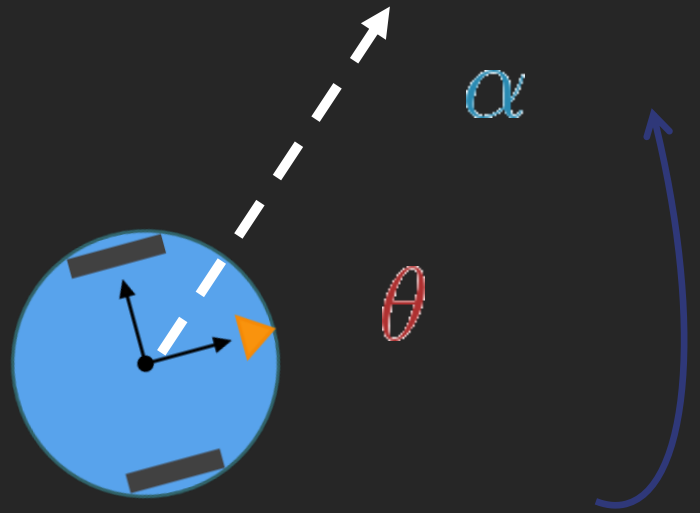
$$\alpha - \theta$$

DIFFERENTIAL ROBOT



$\theta > \alpha$

Negative Rotation



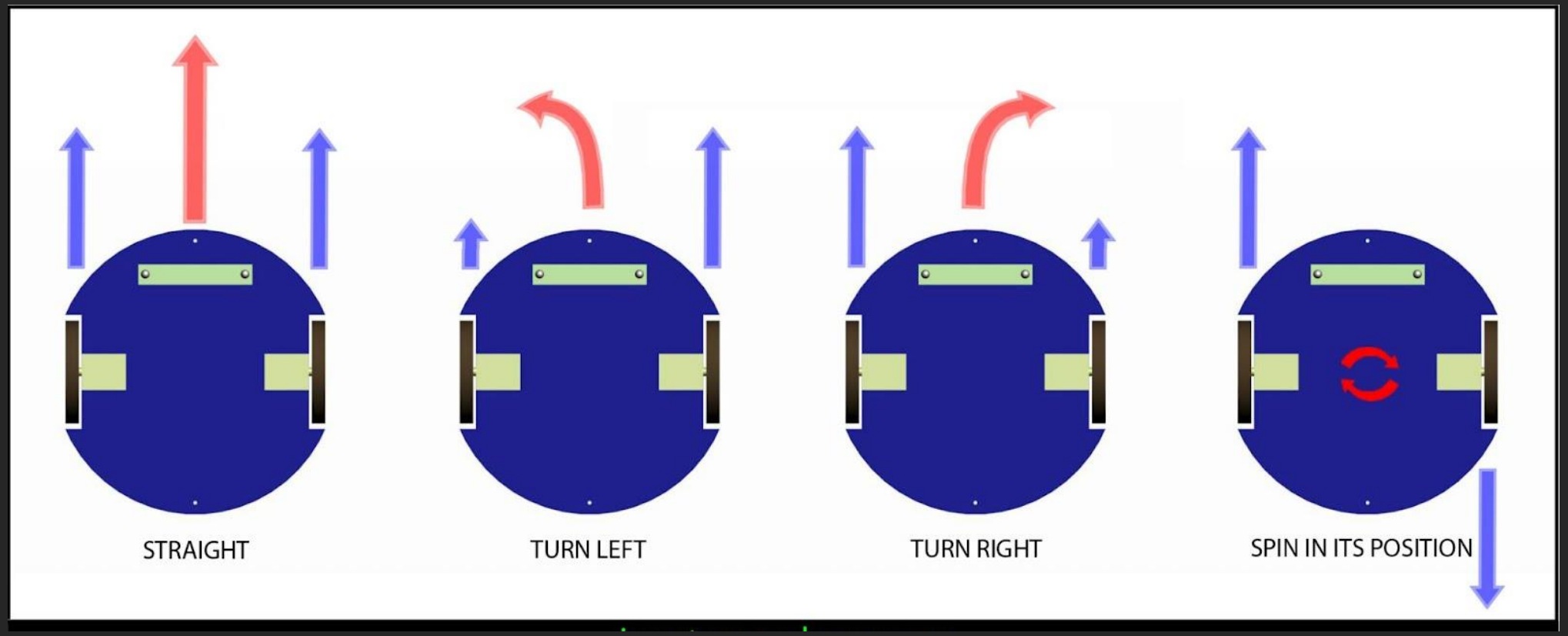
$\theta < \alpha$

Positive Rotation

How much rotation do I need to perform?

$$\alpha - \theta$$

DIFFERENTIAL ROBOT



We can control the **linear velocity** v and the **angular velocity** w of the robot.

DIFFERENTIAL ROBOT

Don’t forget about the sign!

$$\omega_{left} = \frac{v - \frac{b}{2} \cdot \omega}{r} \qquad \omega_{right} = \frac{v + \frac{b}{2} \cdot \omega}{r}$$

DIFFERENTIAL ROBOT

Algorithm Go To Pose:

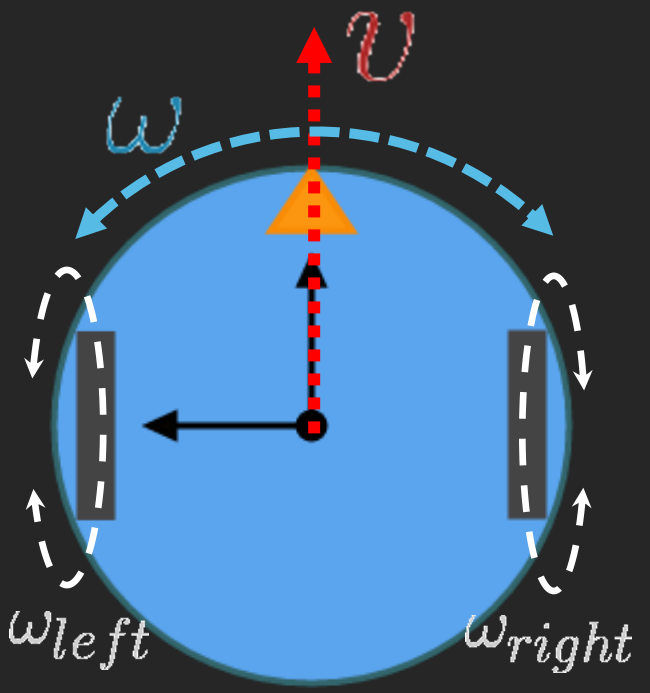
1. Calc the objective orientation
2. Calc the necessary rotation in order to put the robot in the goal's orientation
3. Limit rotations above π/2
4. Calc the necessary robot's linear and angular velocity
5. Calc the necessary wheels' angular velocities in order to perform the velocities on 4.

α

$\alpha - \theta$

v, ω

$\omega_{left}, \omega_{right}$



DIFFERENTIAL ROBOT

Algorithm Calculate Pose 2:

1. Get the number of encoder pulses since last iteration
2. Calc real linear velocity and angular velocity

$$v_{i+1} = \frac{2\,\pi\,r}{C}\,\frac{N_R + N_L}{2}\,\frac{1}{\Delta t}$$

$$w_{i+1} = \frac{2\,\pi\,r}{C}\,\frac{N_R - N_L}{b}\,\frac{1}{\Delta t}$$

3. Calc change in position and orientation

$$\theta_{i+1} = arctan2(sin(\theta_i + w_{i+1}\Delta t), cos(\theta_i + w_{i+1}\Delta t))$$

$$x_{i+1} = x_i + v_{i+1}\cos(\theta_{i+1})\Delta t$$

$$y_{i+1} = y_i + v_{i+1}\sin(\theta_{i+1})\Delta t$$

DIFFERENTIAL ROBOT

Combining odometry and kinematics – first approach

1. Give the robot its initial Position P_{init} and a goal Position P_{goal}

2. Until robot does NOT reach the P_{goal}, do the following

3. Update Odometry

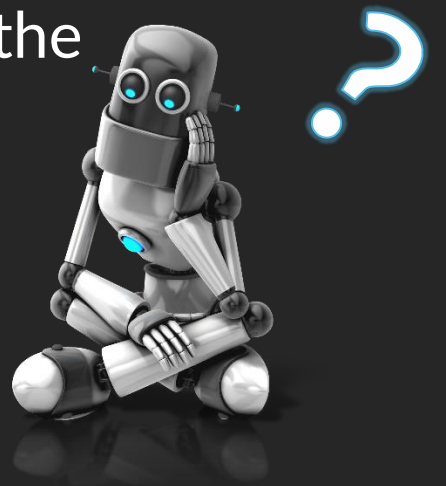
$$P = (x, y, \theta)$$

4. Update Kinematics according to Odometry and P_{goal}

DIFFERENTIAL ROBOT

Combining odometry and kinematics – second approach

1. Give the robot a desired linear and angular velocities, v^d, w^d
2. Calculate the desired angular velocities of the wheels,w_L^d, w_R^d
3. Run algorithm Calculate Pose 2
4. Calculate the real angular velocities of the wheels,w_L, w_R
5. Hum…and what now? How to make the robot move at the desired velocity?



KINEMATICS – CHALLENGE #2

- In Arduino, adjust challenge #1 in order to calculate the forward kinematics of the robot and evaluate it considering the encoder data given before. Besides the existing functions, add those ones:
  - cmd_vel(): Returns the desired linear and angular velocities (at this point, you can add constant values at will)
  - cmd_vel2wheels(V, W): Converts the linear and angular velocities to wheels angular velocities
  - poseUpdate(NL,NR,r,b,C): Besides returning the new position of the robot, also returns the real linear and angular velocities of the robot

THAT'S WHERE CONTROL COMES!

By control, we mean the science of **CONTROL THEORY**.

CONTROL

- Formal definition

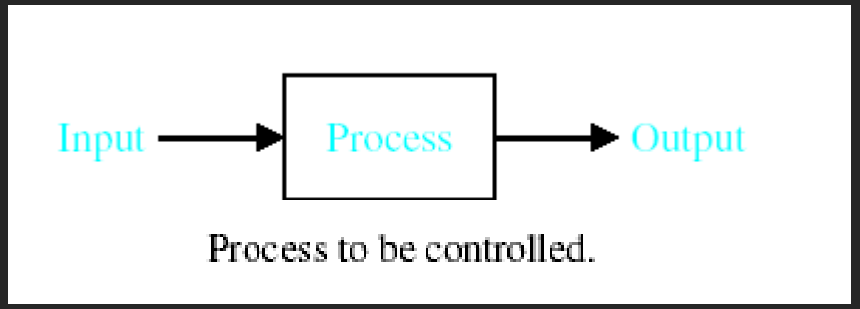
"interdisciplinary branch of engineering and computational mathematics that deals with the behavior of dynamical systems with inputs, and how their behavior is modified by feedback."

CONTROL

System – An interconnection of elements and devices for a desired purpose

Control System – An interconnection of components forming a system configuration that will provide a desired response.

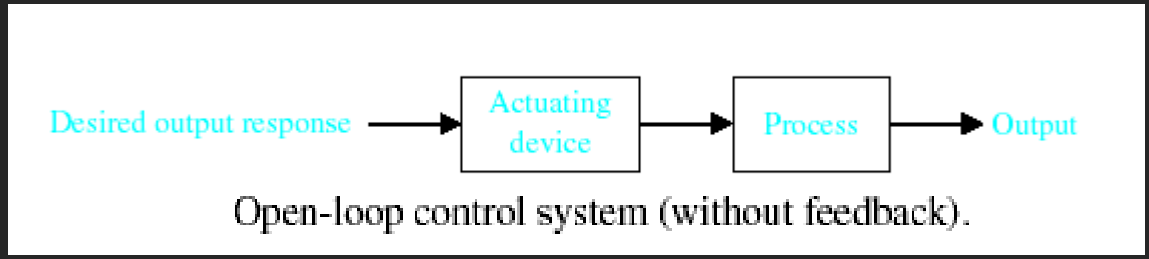
Process – The device, plant, or system under control. The input and output relationship represents the cause-and-effect relationship of the process.



Process to be controlled.

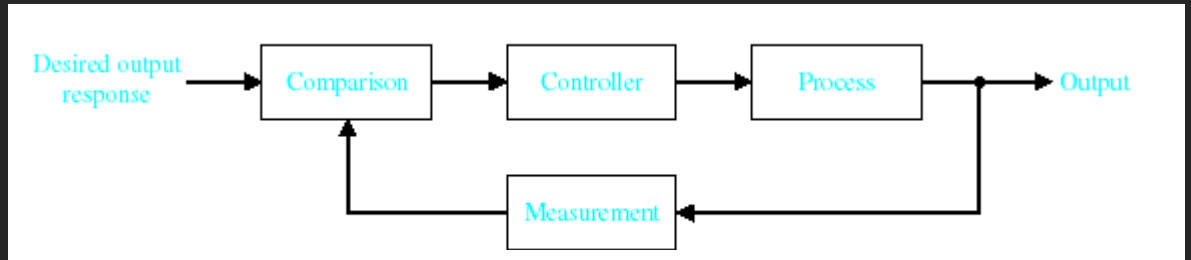
CONTROL

Open-Loop Control Systems utilize a controller or control actuator to obtain the desired response.



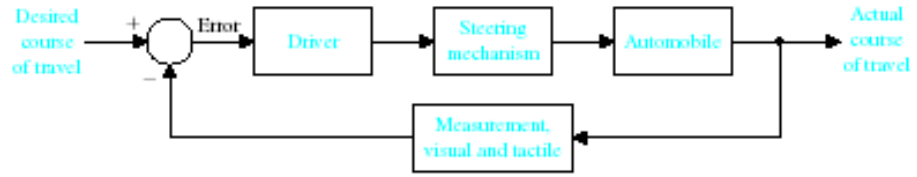
Open-loop control system (without feedback).

Closed-Loop Control Systems utilizes feedback to compare the actual output to the desired output response.

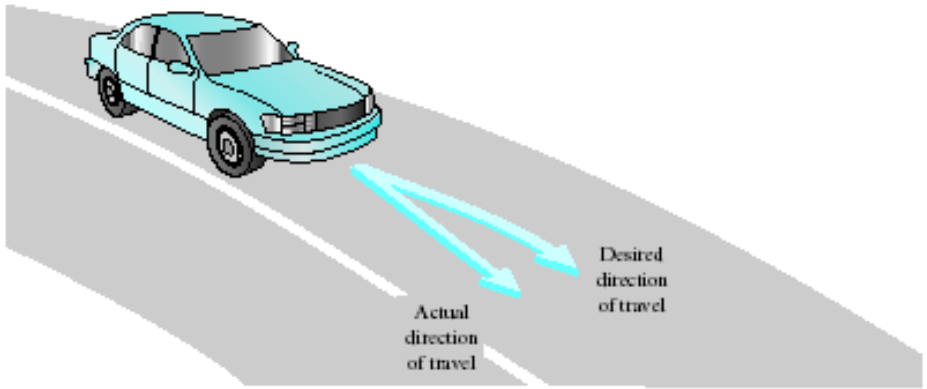


Closed-loop feedback control system (with feedback).

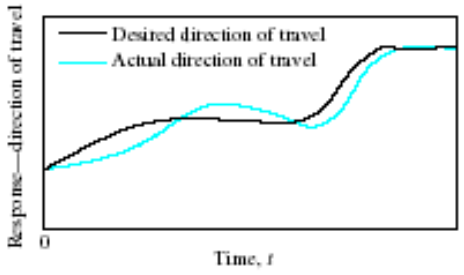
CONTROL



(a)



(b)



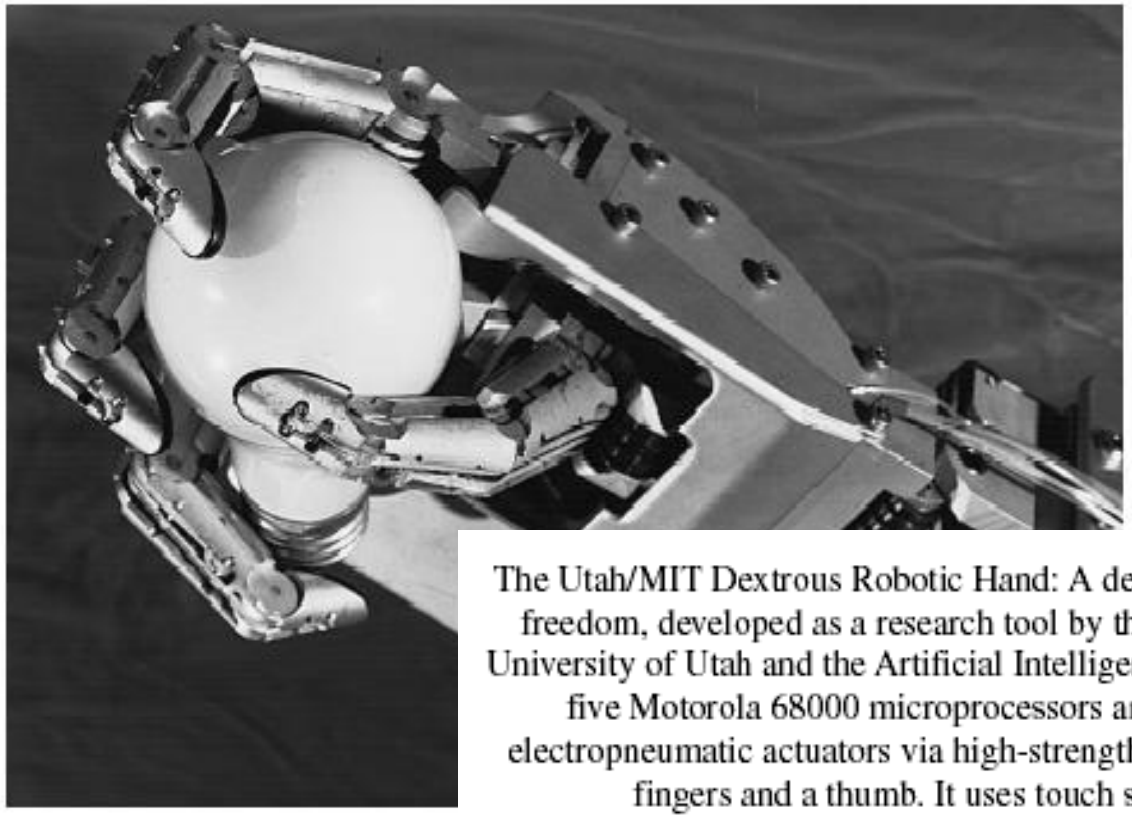
(c)

Automobile steering control system

The driver uses the difference between the actual and the desired direction of travel to generate a controlled adjustment of the steering wheel

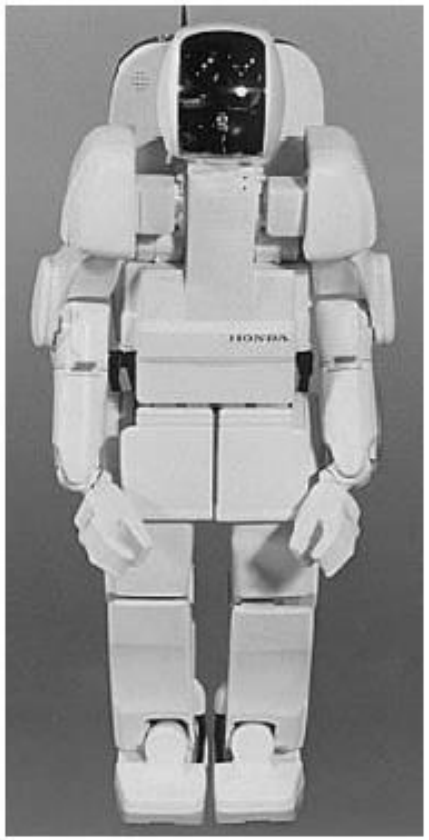
Typical direction-of-travel response

CONTROL



The Utah/MIT Dextrous Robotic Hand: A dextrous robotic hand having 18 degrees of freedom, developed as a research tool by the Center for Engineering Design at the University of Utah and the Artificial Intelligence Laboratory at MIT. It is controlled by five Motorola 68000 microprocessors and actuated by 36 high-performance electropneumatic actuators via high-strength polymeric tendons. The hand has three fingers and a thumb. It uses touch sensors and tendons for control. (Photograph by Michael Milochik. Courtesy of University of Utah.)

CONTROL



The Honda P3 humanoid robot. P3 walks, climbs stairs and turns corners.
Photo courtesy of American Honda Motor, Inc.

CONTROL

Boston Dynamics' ATLAS robot can walk, run and jump in many different ways, including back and front flip.

WHAT KIND OF CONTROL METHOD CAN WE USE TO CONTROL THE SPEED OF OUR ROBOT'S WHEELS?

The PID controller is one of the most widely used (alongside fuzzy logic) in the industry (e.g., robotic manipulators, AGVs, etc.) and mostly in our houses (e.g., AC, washing machine, etc.).
Let's introduce the PID controller.

PID

- More than half of the industrial controllers in use today utilize PID or modified PID control schemes (most of the time merged with fuzzy logic)

- When the mathematical model of the plant is not known and therefore analytical design methods cannot be used, PID controls prove to be quite useful

PID

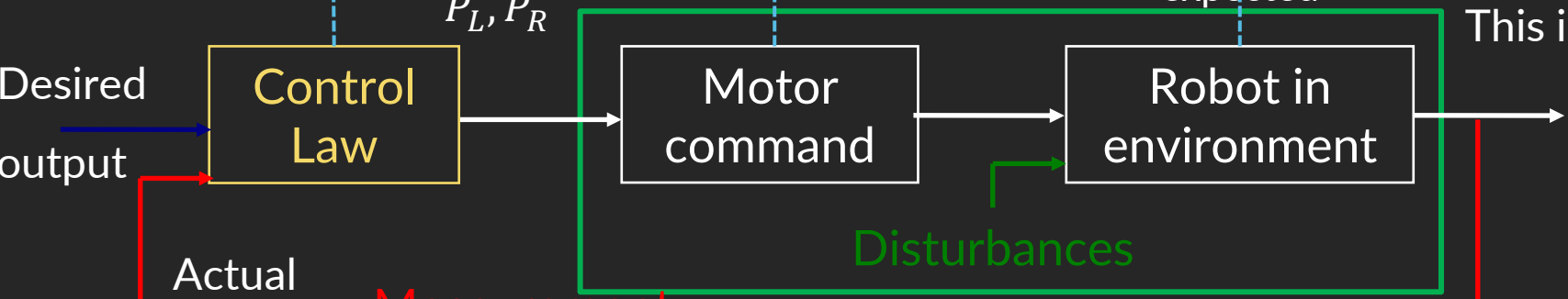
$$K_p\left(e(t) + \frac{1}{T_i}\int_0^t e(\tau)\,d\tau + T_d\frac{de(t)}{dt}\right)$$

The desired output can be our desired wheels velocity, as seen before w_L^d, w_R^d

The output of the controller feeds our robot's motors (power for each motor)

P_L, P_R

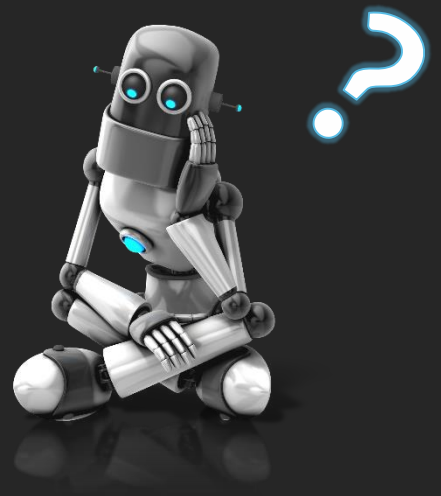
Even though the motors did receive controlled commands, the world is dynamic, and the robot may not behave exactly as expected



In our case, the measurements (feedback) is provided by the motor encoders, which can be translated into wheel angular velocity (as previously seen) w_L, w_R

PID

- But what is the meaning of K_p, K_i and K_d ?



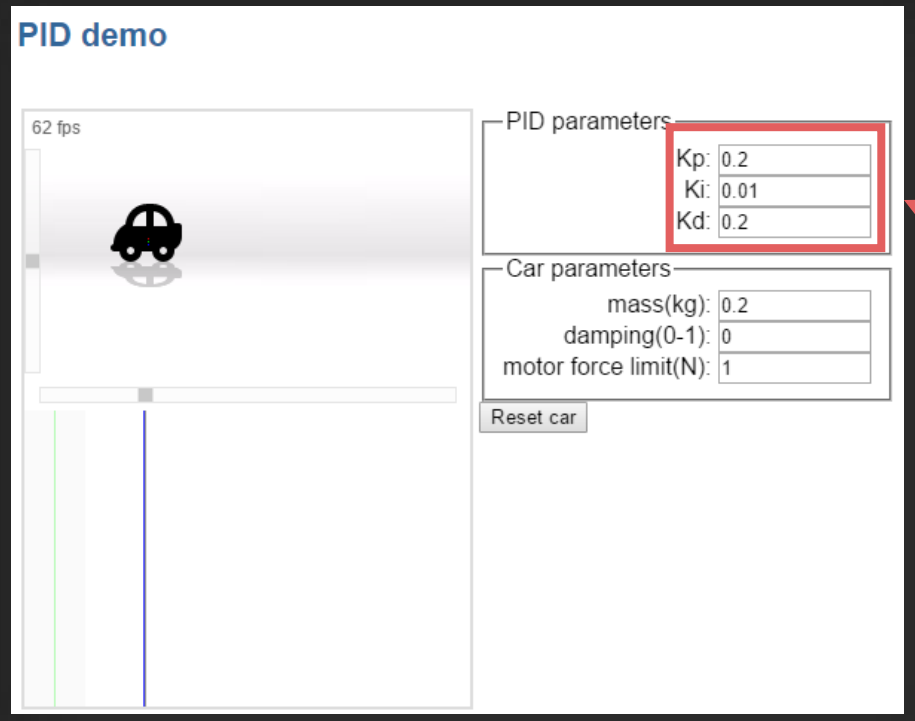
- There are 3 components in the PID
  - **P** = Proportional
  - **I** = Integral
  - **D** = Derivative

- K_p, K_i and K_d correspond to the gains of each component, respectivelly, necessary to reduce the proportional, integral and derivative errors.

PID

- Play this game and share your thoughts regarding K_p, K_i and K_d:

  https://sites.google.com/site/fpgaandco/pid-demo



Manipulate the gains of the controller

PID

Response	Rise Time	Overshoot	Settling Time	SS Error
K_P	Decrease	Increase	Small Change	Decrease
K_I	Decrease	Increase	Increase	Eliminate
K_D	Small Change	Decrease	Decrease	Small Change

PID

But how to tune these gains to obtain the best response possible?

Many different types of tuning rules have been proposed in the literature.

- ☞ Manual tuning on-site (with support of Ziegler-Nichols rules)
- ☞ On-line automatic tuning (e.g., PSO)
- ☞ Gain scheduling (e.g., hybrid approaches or simple heuristics to adapt the gains based on specific responses/changes in the plant)

PID

Ziegler-Nichols' Rules for Tuning PID Controllers

- Ziegler and Nichols proposed rules for determining values of the proportional gain K_p, integral time T_i, and derivative time T_d based on the transient response characteristics of a given plant.

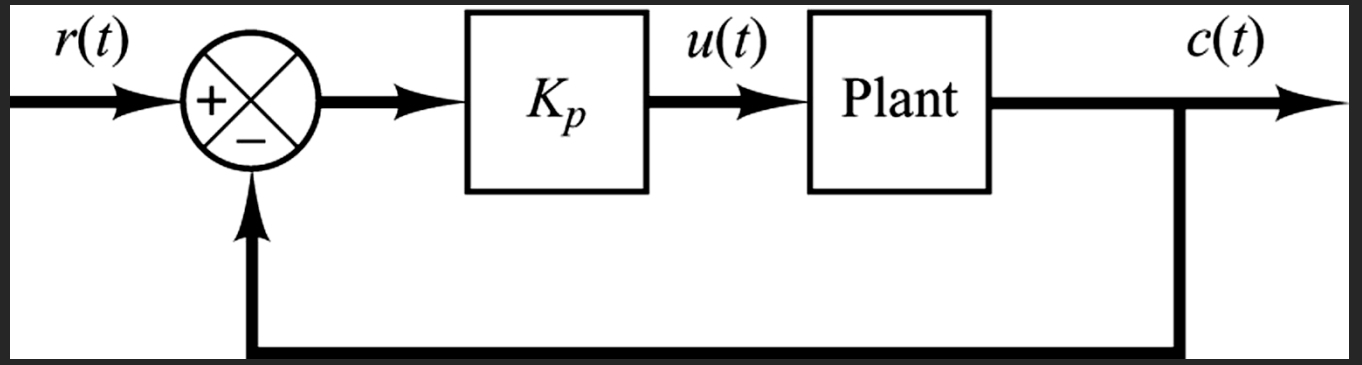
- Such determination of the parameters of PID controllers or tuning of PID controllers can be made by engineers on-site by experiments on the plant.

- We need to apply a series of fine tunings until an acceptable result is obtained.

PID

Ziegler-Nichols Method of Tuning Rule (one of them actually)

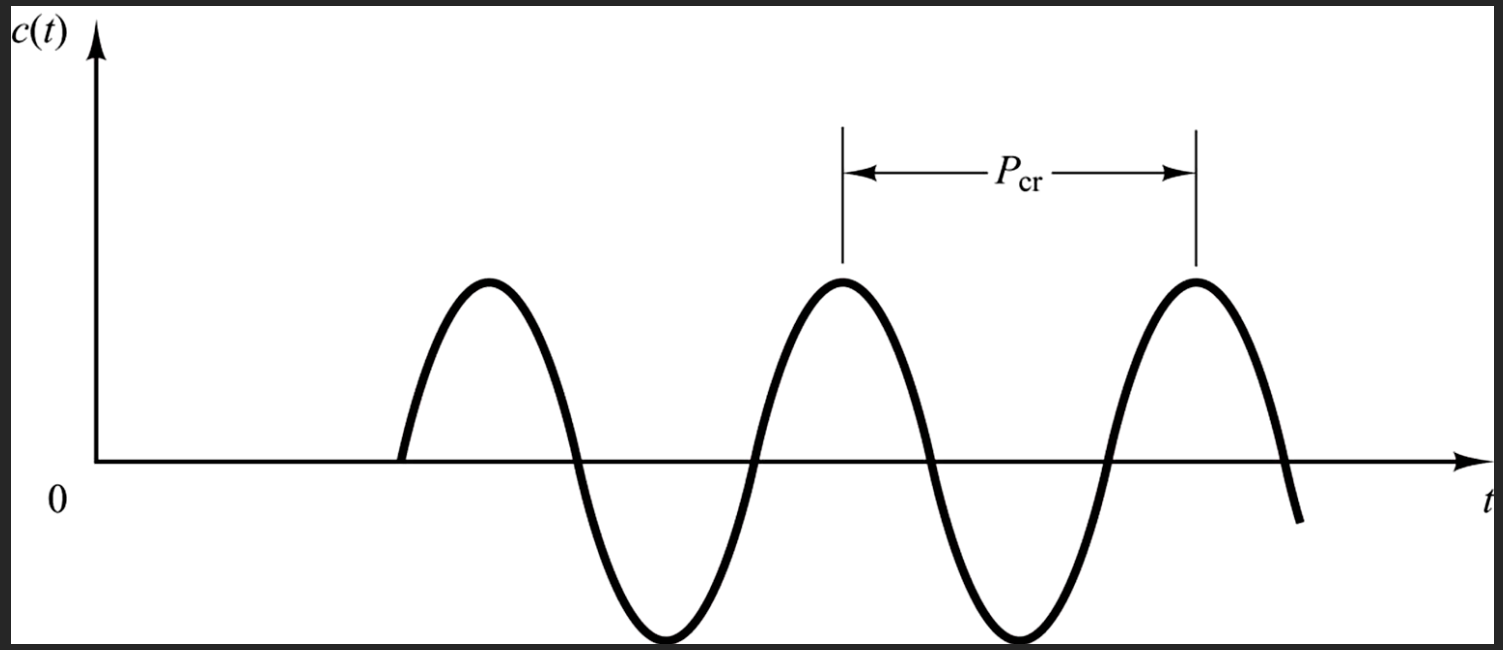
1. We first set $T_i = \infty$ and $T_d = 0$, which implies using only the proportional control action.



Closed-loop system with a proportional controller.

PID

2. We increase K_p from 0 to a critical value $K_{cr,}$ at which the output starts exhibiting sustained oscillations.



Sustained oscillation with critical period P_{cr} (P_{cr} is measured in seconds)

PID

3. Ziegler and Nichols suggested that we set the values of the parameters according to these formulas:

Type of Controller	K_p	T_i	T_d
P	$0.5K_{cr}$	∞	0
PI	$0.45K_{cr}$	$\frac{1}{1.2}P_{cr}$	0
PID	$0.6K_{cr}$	$0.5P_{cr}$	$0.125P_{cr}$

Ziegler–Nichols Tuning Rule Based on Critical Gain K_{cr} and Critical Period P_{cr}

PID

Disadvantages of Ziegler-Nichols method

- It can be time consuming if several trials are required and the process dynamics is slow
- Introducing oscillations into a process is, in many cases, not desirable (motors may suffer from current peaks)
- Will not work for an integrating process (streams comprised of gases, liquids, powders, slurries and melts do not naturally settle out at a steady-state operating level)
- Will not work for dead time dominant processes (dead time is the delay from when a controller output signal is issued, until when the measured process variable first begins to respond)
- Requires proper manipulation of the bias value by the controller or the person doing the tuning

Sometimes it is better to simply adopt a trial-and-error approach…

PID

Limitations of PID control

- **Tuning**: largely empirical (e.g., Ziegler–Nichols method), unless one can easily acquire data and apply an optimization method (harder than it seems)
- **Nonlinearities**: can only be solved by integrating with other approaches, such as gain scheduling, multi-point controller, fuzzy logic
- **Noise**: use low-pass filter (may cancel out D → PI)
- Often PI or PD is sufficient…

PID

Application in the control of the differential robotic system:

1. Give the robot a desired linear and angular velocities, v^d, w^d
2. Calculate the desired angular velocities of the wheels,w_L^d, w_R^d
3. Run algorithm Calculate Pose 2
4. Calculate the real angular velocities of the wheels,w_L, w_R
5. Calculate the proportional error:

$$\varepsilon_{P\,i+1} = \begin{bmatrix} w_L^d - w_L \\ w_R^d - w_R \end{bmatrix}$$

PID

6. Calculate the derivative error:

$$\varepsilon_{D\,i+1} = \varepsilon_{P\,i+1} - \varepsilon_{P\,i}$$

7. Calculate the integral error:

$$\varepsilon_{I\,i+1} = \varepsilon_{I\,i} + \varepsilon_{P\,i+1}$$

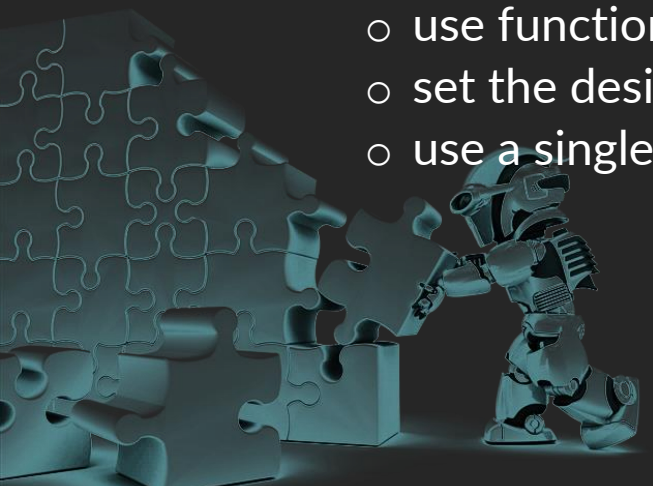
PID

8. Calculate the PID output and feed the motor driver with:

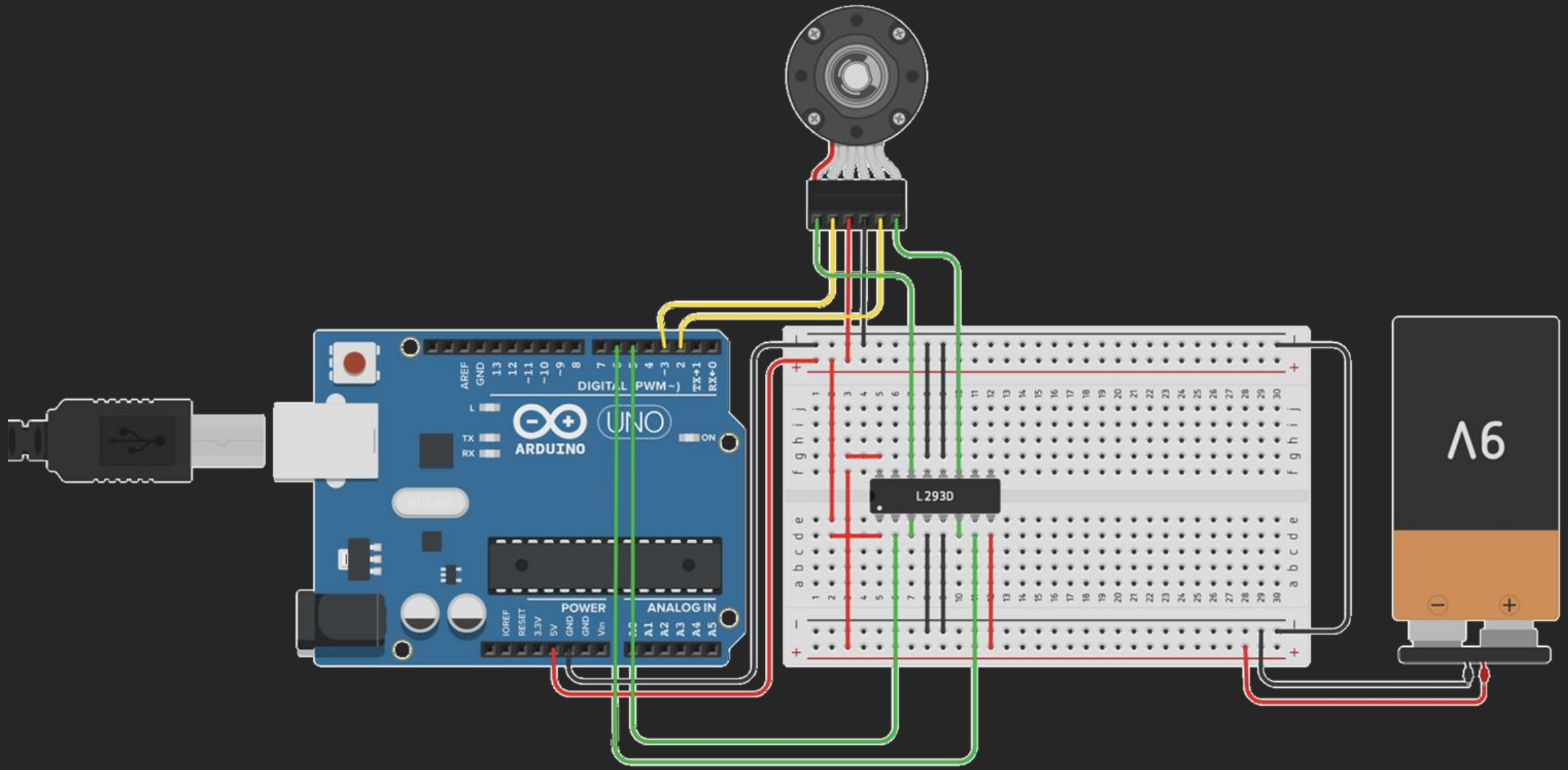
$$G_{i+1} = K_P \varepsilon_{P\,i+1} + K_I \varepsilon_{I\,i+1} \Delta t + K_D \frac{\varepsilon_{D\,i+1}}{\Delta t}$$

CONTROL – CHALLENGE #3

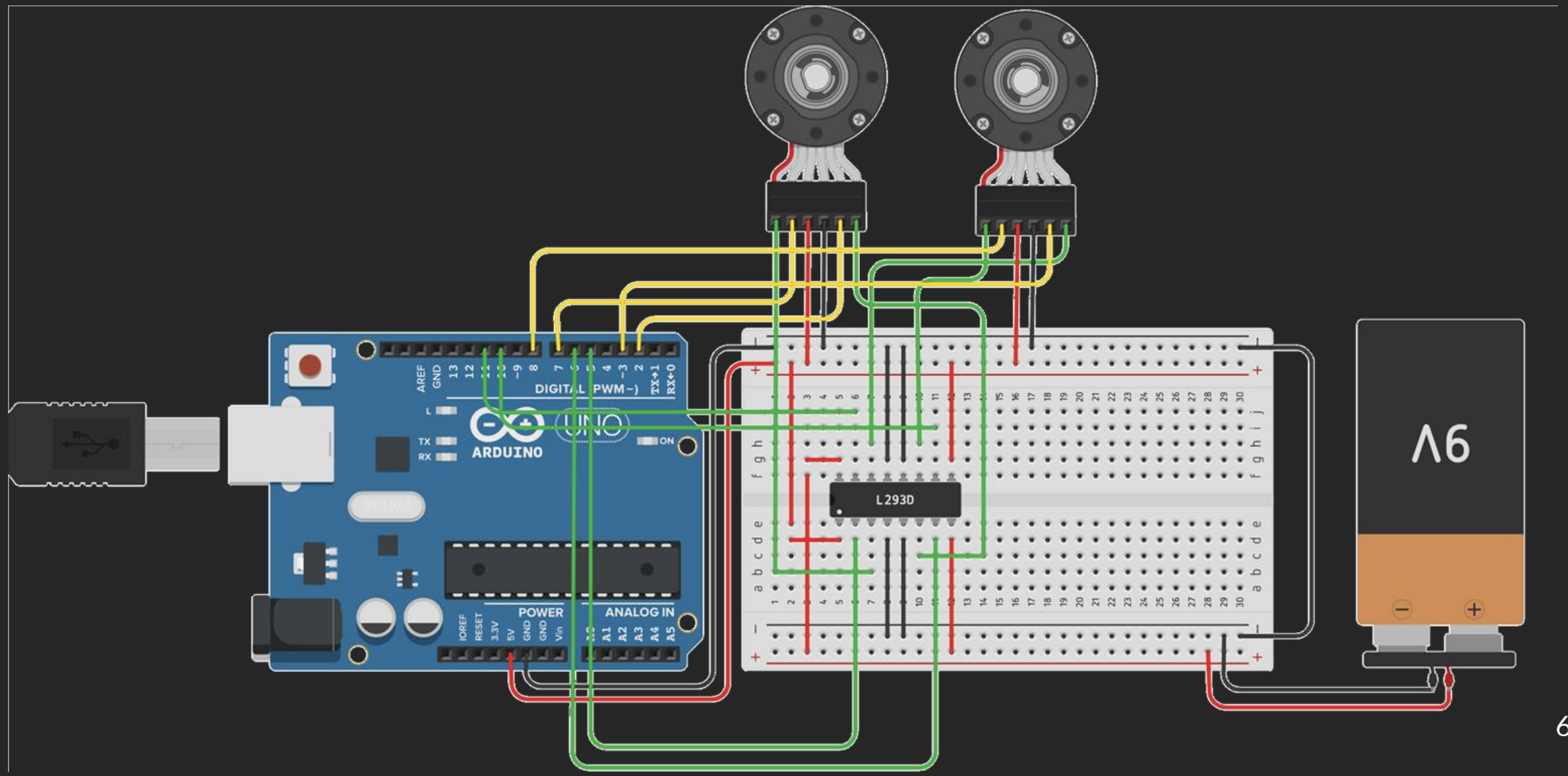
- In Arduino, implement a PID control for a DC motor equipped with an encoder (e.g. TinkerCAD: DC Motor with Encoder) with the following functions:
  - encFunc(): Increments an integer variable (e.g., long int) each time it enters this function; the callback of the function should be triggered by an attachInterrupt (e.g., using pin 3) or by using the *Encoder.h* library
  - pid_controller(): Runs the PID controller considering a given desired speed (it can be a constant value in an early phase and, afterwards, a potentiometer or a simple sequence of velocities) and the real speed calculated based on the difference between encoders' pulses at the desired sampling rate
  - TIPS:
    - use function millis() to control the sampling rate at 100 ms
    - set the desired speed between the whole range of possibilities, namely maximum speed
    - use a single encoder phase and control in a single direction to start with



CONTROL – CHALLENGE #3



CONTROL – CHALLENGE #3



CONTROL – CHALLENGE #3

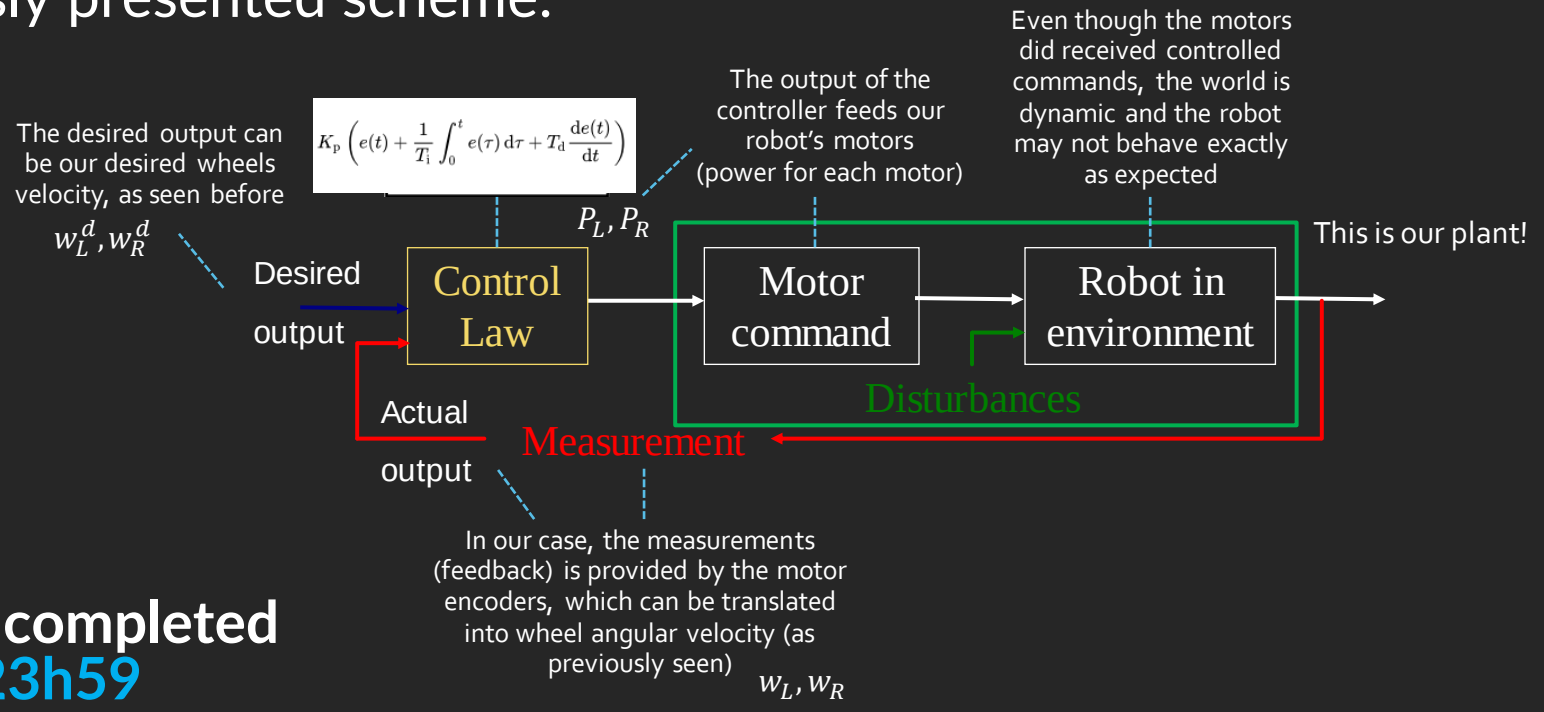
Just a small code to test your dual-motor connections!

```
#include<Encoder.h>
#define MLpos 5
#define MLneg 6
#define PLA 2
#define PLB 7
#define MRpos 10
#define MRneg 11
#define PRA 3
#define PRB 8

Encoder myEncL(PLA, PLB);
Encoder myEncR(PRA, PRB);

void setup()
{
  Serial.begin(9600);
  pinMode(LED_BUILTIN, OUTPUT);

  analogWrite(MLpos, 0);
  analogWrite(MLneg, 200);
  analogWrite(MRpos, 0);
  analogWrite(MRneg, 200);
}
```

```
void loop()
{
  unsigned long Tnow = millis();
  long NRt = 0, NR = 0;
  long NLt = 0, NL = 0;
  while(1){
    if (millis() - Tnow >= 1000){
      Tnow = millis();

      NL = NLt - myEncL.read();
      NLt = myEncL.read();
      Serial.println(NL);

      NR = NRt - myEncR.read();
      NRt = myEncR.read();
      Serial.println(NR);

    //delay(500);
    }
  }
}
```

CONTROL – Task

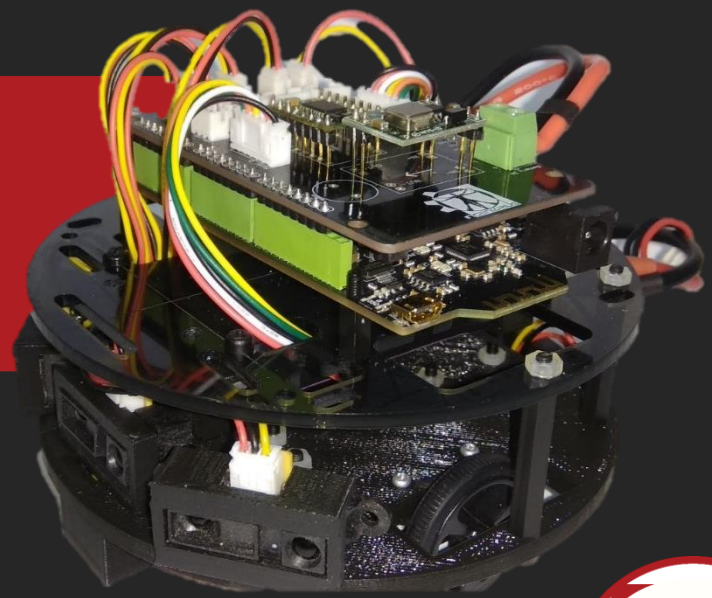
Adapt and merge the codes from challenge #2 and challenge #3 for the real hardware; i.e., for the motor driver, motor and encoder setup you will use in your robot. Control both speed and direction of both wheels considering a command velocity comprising linear and angular velocities, following the previously presented scheme:



- **This task should be completed until the 28th July, 23h59**

CRAFT #4

Thank you



Micael S. Couceiro micael@ingeniarius.pt

22/07/2024