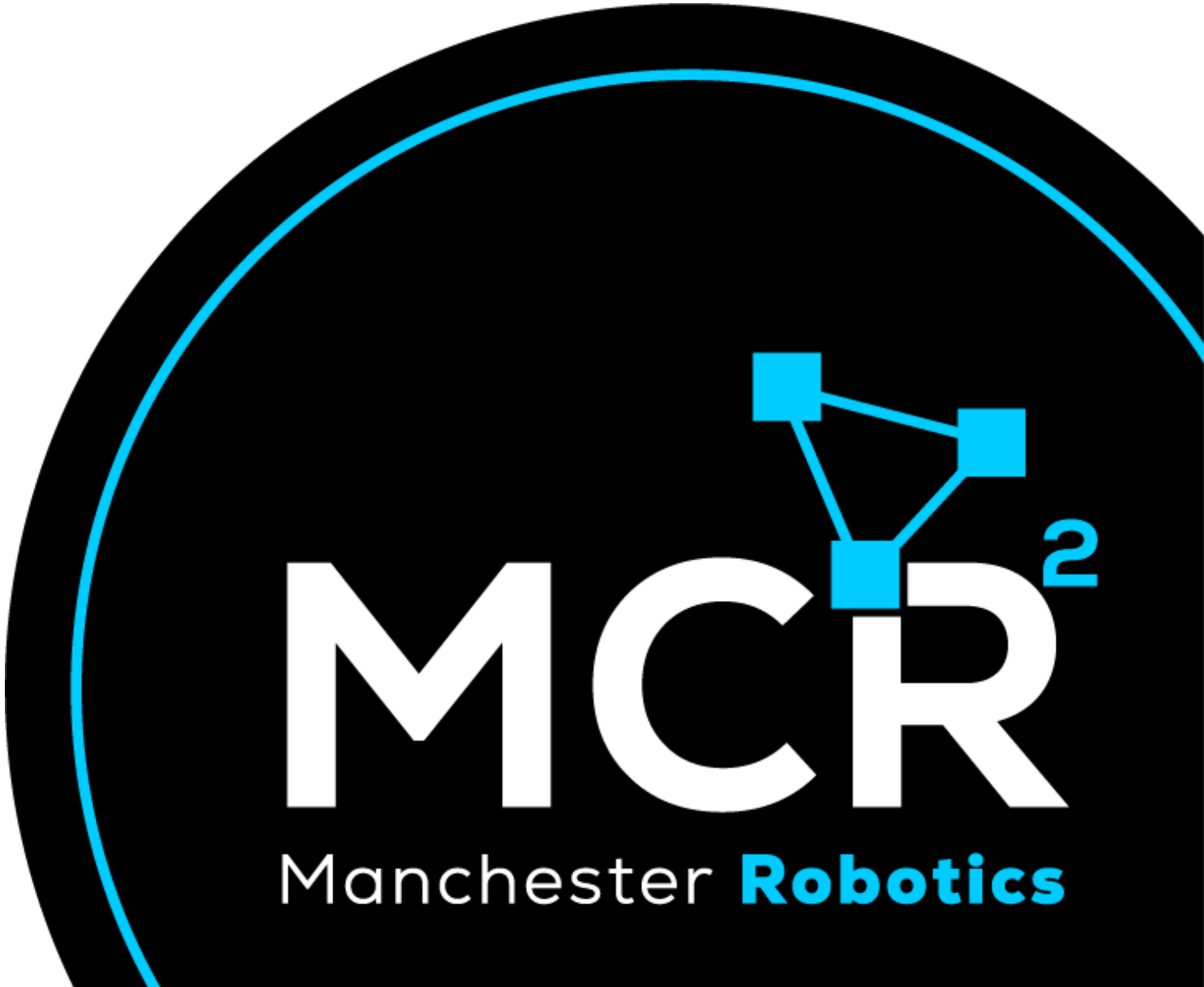
{Learn, Create, Innovate};

Open Loop Control

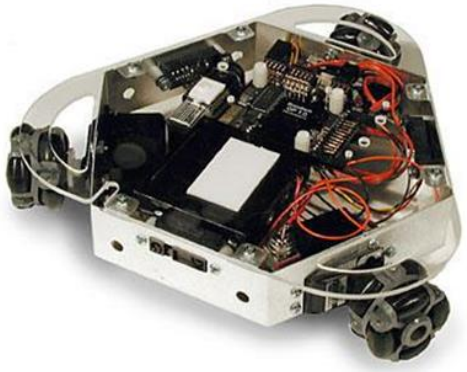
From zero to a functional robot.



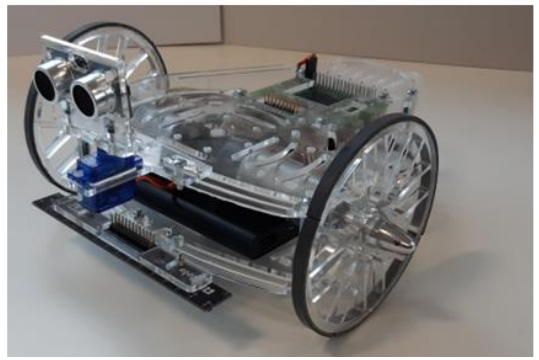
Mobile Robots

- There exists many types of wheeled robotic platforms
- Differential-Drive robots
- Omnidirectional robots
- Ackermann-steering robots
- and many others...

- In this course we will focus on differential drive robots, also known as “differential wheeled robots”.



Holonomic Robot
Acroname ©.

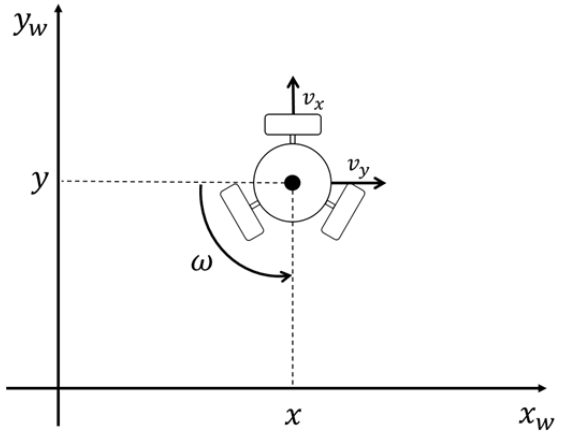


Differential-drive
Puzzlebot ©.

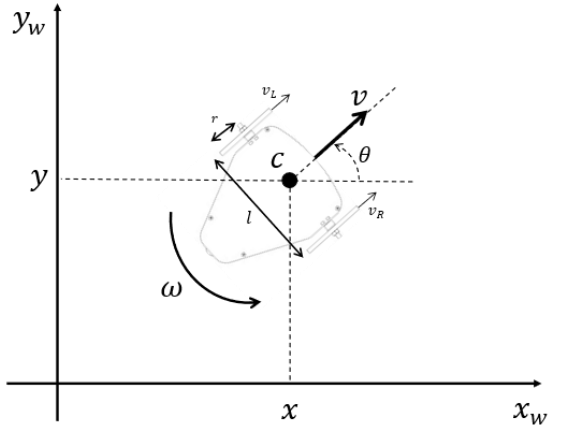
Holonomic vs. Nonholonomic Systems

Mobile robots can be classified as "Holonomic" or "Nonholonomic".

- This classification depends on the relationship between controllable and total degrees of freedom of a robot.
- **Holonomic Robots:** If the controllable degree of freedom is equal to total degrees of freedom, then the robot
- **Nonholonomic Robots:** If the controllable degree of freedom is less than the total degrees of freedom. Such systems are therefore called underactuated. Differential Drive Systems fall into this category.



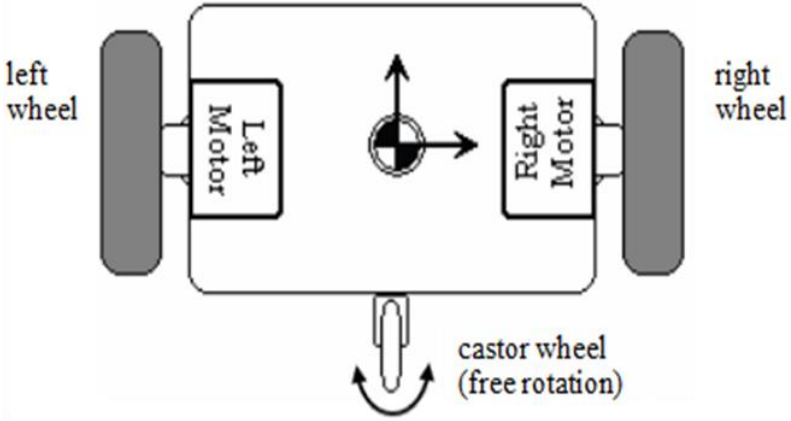
Holonomic Robot

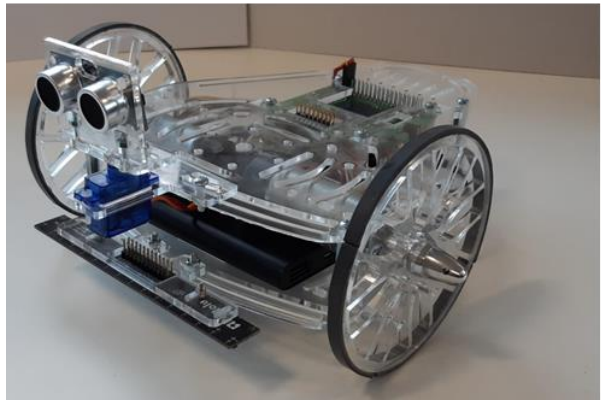


Nonholonomic Robot

Differential Drive

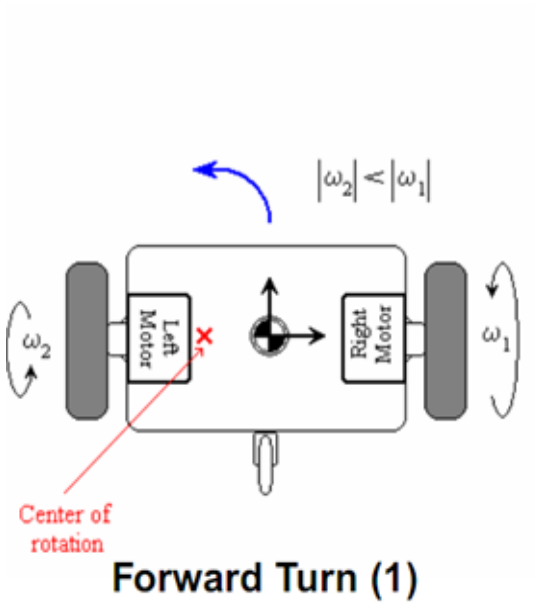
- These are mobile robots whose movement is based on two separately driven wheels placed on either side of the robot body.
- Two active wheels and one caster wheel (most common configuration)
- It can thus change its direction by varying the relative rate of rotation of its wheels, thereby requiring no additional steering motion.



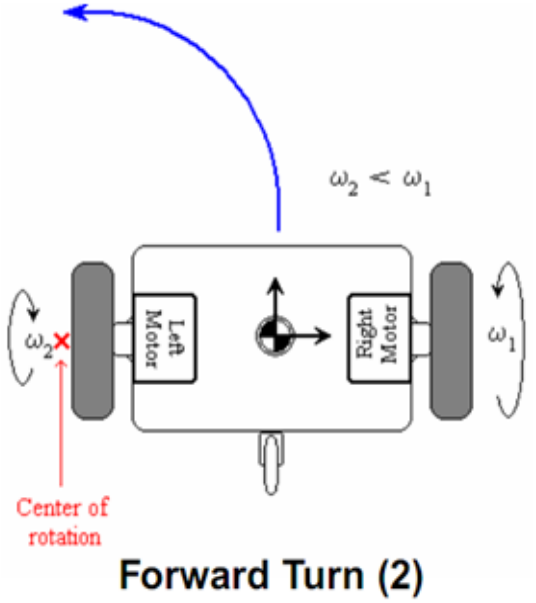


Differential drive robots

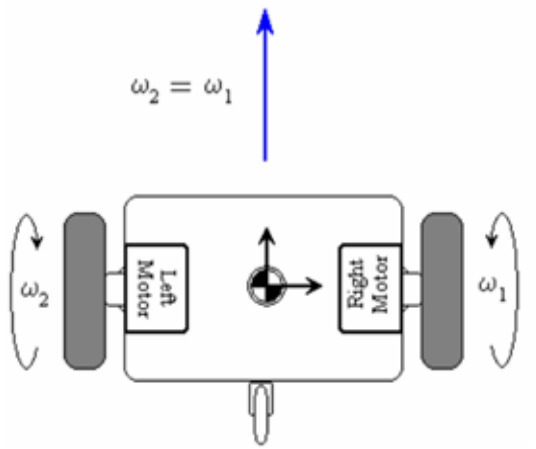
Differential Drive

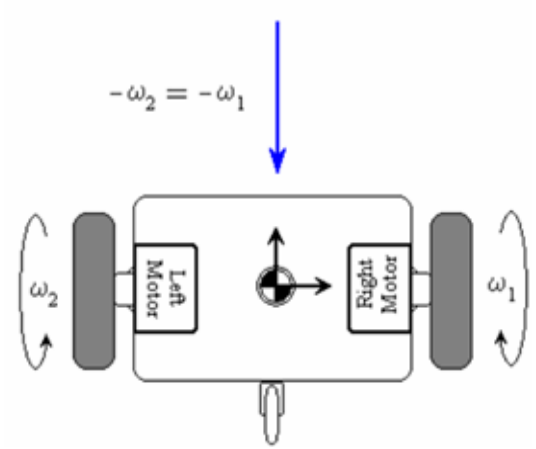


Forward Turn (1)



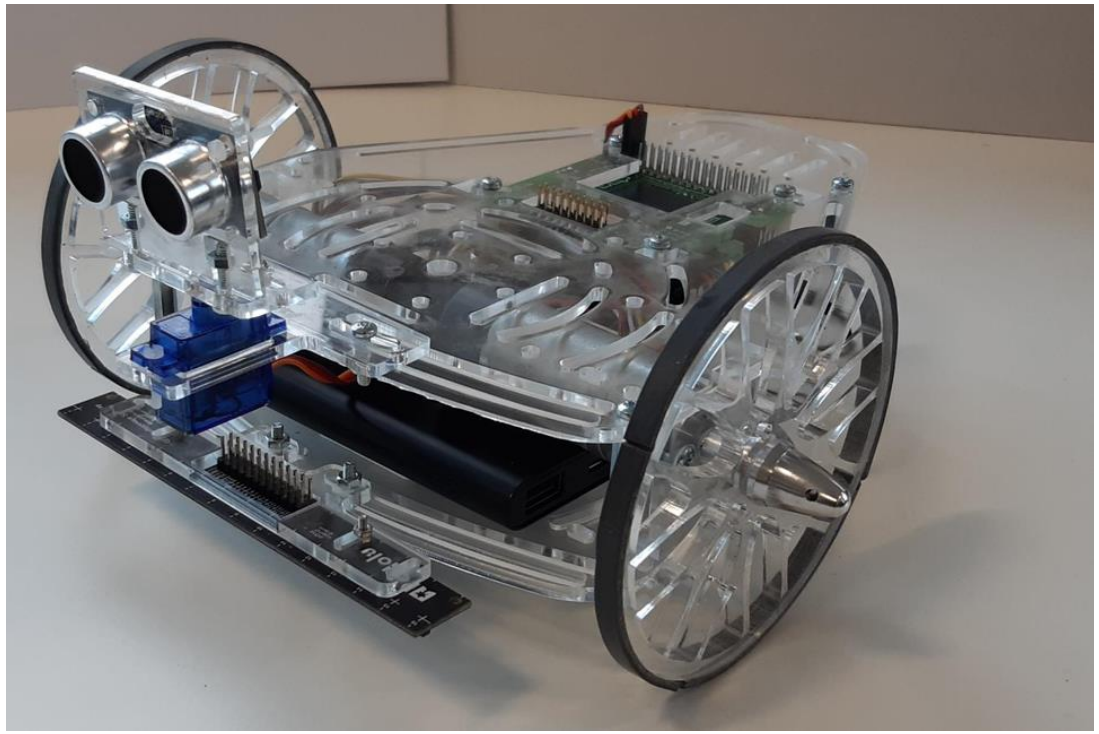
Forward Turn (2)





Sensors and Actuators for Differential Drive Robots

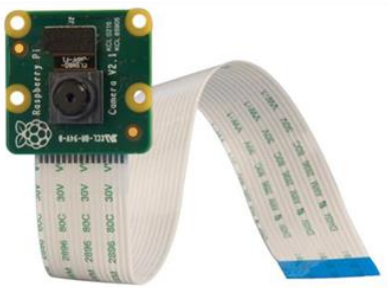
- Differential drive robots can perform different tasks autonomously.
- Each task requires a set of sensors and actuators that can be added to the robot to increase it level of autonomy.
- The minimum required sensors and actuators (motors, encoders, etc.), are the ones that allow us to control mobile robot movement and be localised (obtain its position and orientation), to correctly perform its functions.



Differential Drive Robot Sensors and Actuators.

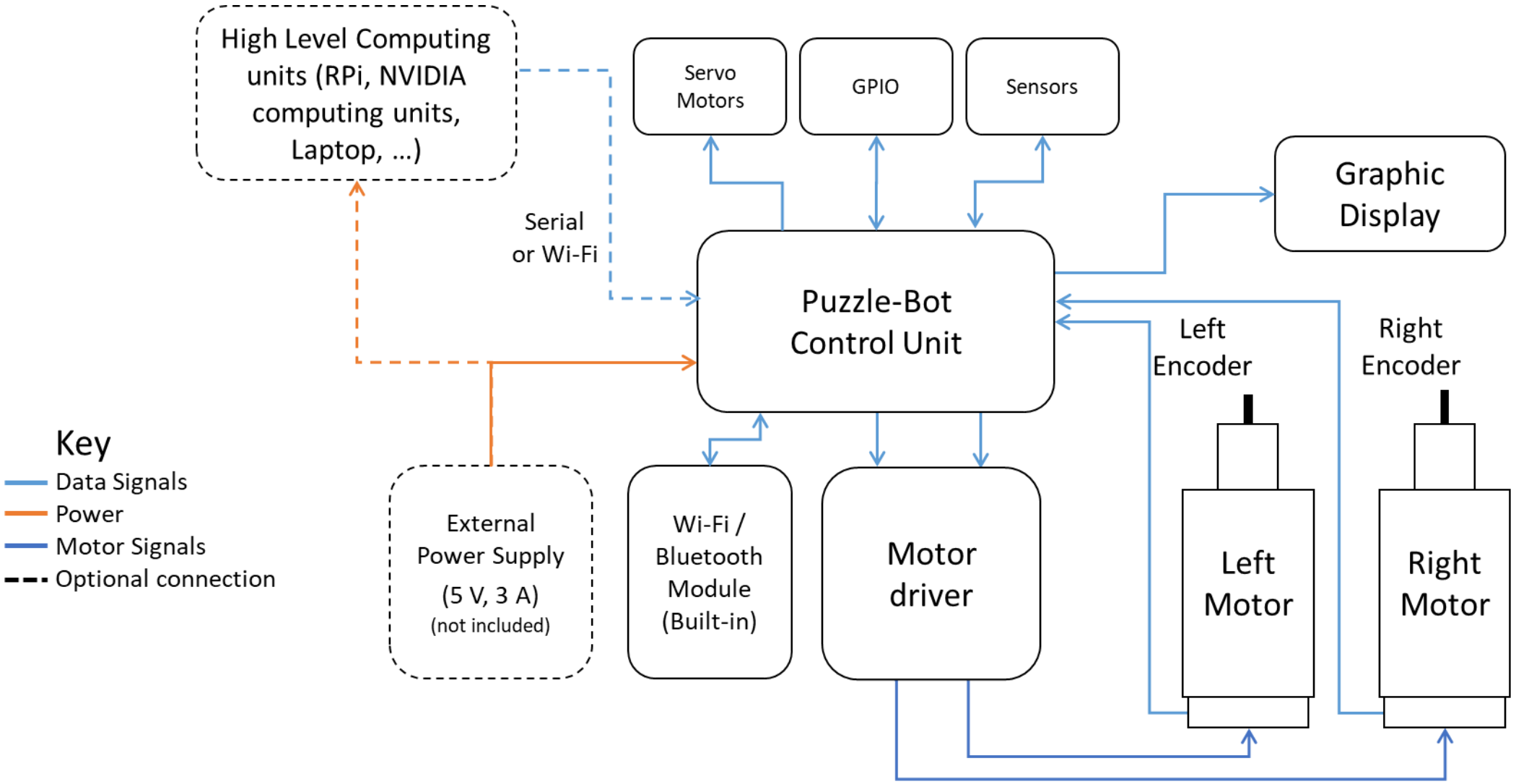
Differential Drive Sensors and Actuators

- For the case of the sensors, they can be classified as Exteroceptive and Proprioceptive.
- Exteroceptive: Used to measure the environment or the state of the environment, topology of the environment, temperature, etc. Some examples are Sonar, LiDAR, Light sensors, bumper sensors, magnetometers.
- Proprioceptive: Used to measure the state of the robot such as wheel position, velocity, acceleration, battery charge, etc. Some examples include, encoders, battery level, gyrometers, accelerometers.



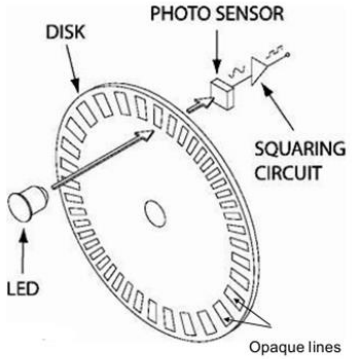
Sensors and Actuators.

Differential Drive Sensors and Actuators

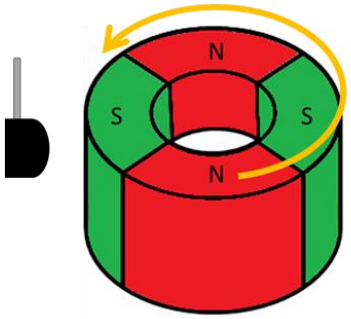


Encoders

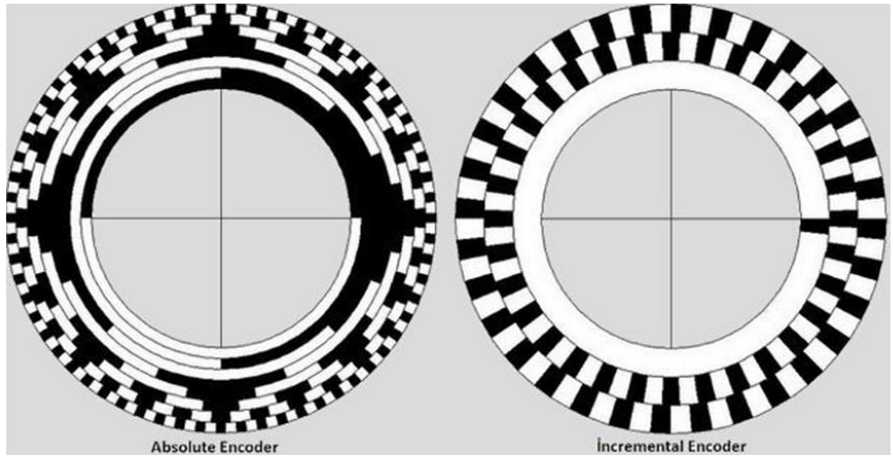
- Device that converts the angular position of a shaft (motor shaft) to an analogue or digital signal.
  - **Absolute:** Indicates the position of the shaft at all times, by producing a unique digital code for each angle (Angle transducers).
  - **Incremental:** Record the changes in position of the motor shaft with no indication or relation to any fixed position of the shaft.
- Encoders in mobile robots are considered proprioceptive sensors because they only acquire information about the robot itself, not the structure of the environment.



Optical Rotary Encoder

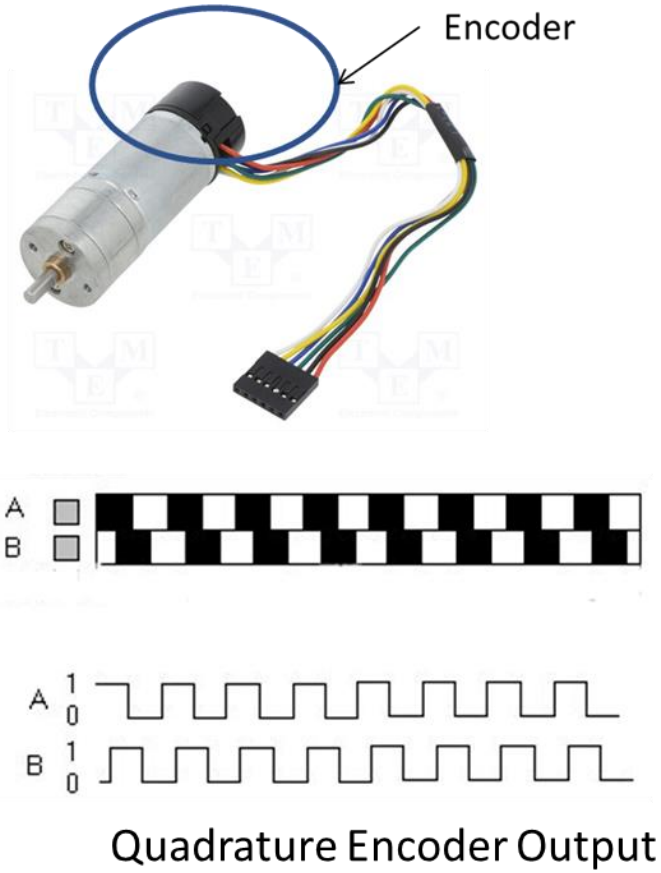


Magnetic Rotary Encoder



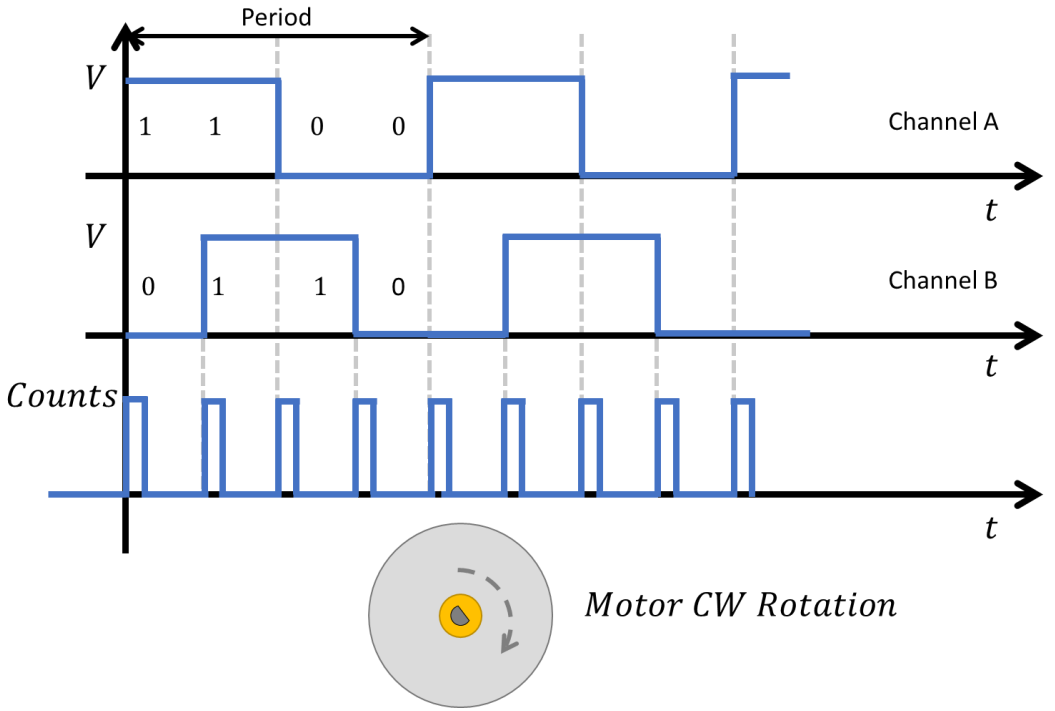
Encoders

- Incremental encoders, produce a series of electrical high-low pulses. These pulses, allows to obtain information such as the angular rotation of the shaft or the angular speed of the motor by counting the number of pulses that occur in a certain period (Δt) .
- In robotics, when an encoder is attached to the axle of each wheel in a differential-drive robot, it is possible to convert the number of pulses into useful information, such as the velocity or distance travelled by each wheel.
- With a single set of pulses (single channel / Channel A), it is impossible to know if the motor is rotating clockwise (CW) or counterclockwise (CCW).
- Therefore, a second line (dual channel / channel B) is attached, having its signal shifted by 90 electrical degrees (°e) with respect to channel A.



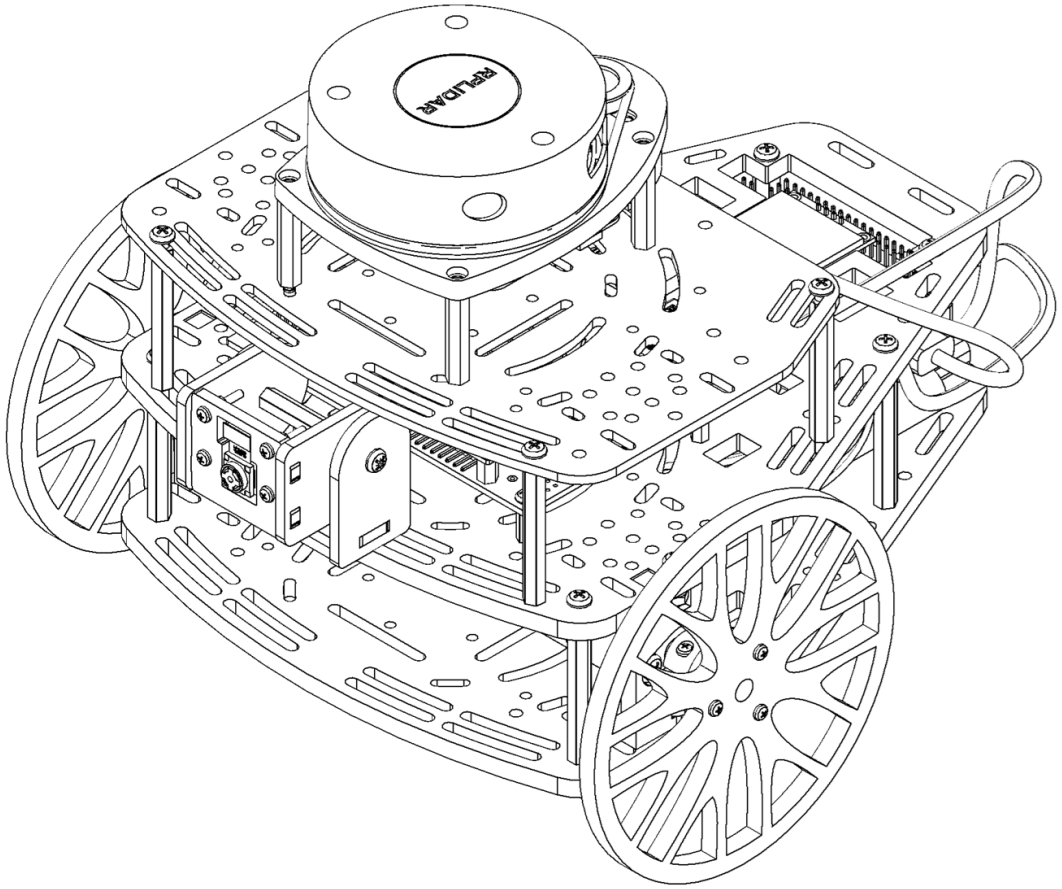
Encoders

- This phase shift allows to determine the direction of rotation.
- Depending on direction of rotation the signal of channel A is preceding channel B or vice versa.
- A simple estimation of the direction would be to verify the previous inputs (Channels A and B) and compare it with the actual inputs, the “code” will change dependant on the rotation direction.
- Counting the pulses of both channels, also leads to a more accurate angular velocity estimation.
- To count the pulses using a MCU, the most common methodology is to use interrupts so that no information is lost.



Encoders

- Puzzlebot motors use an incremental dual channel, quadrature encoders with 13 pulses per revolution, attached to the motor shafts before the reduction (35:1).
- The encoder is used to estimate the speed of the motors.
- Since the encoders can have a lot of noise, it is recommended to use a filter (low pass or band pass) to avoid having a noisy signal that affects the controller.



DC Motors

Introduction

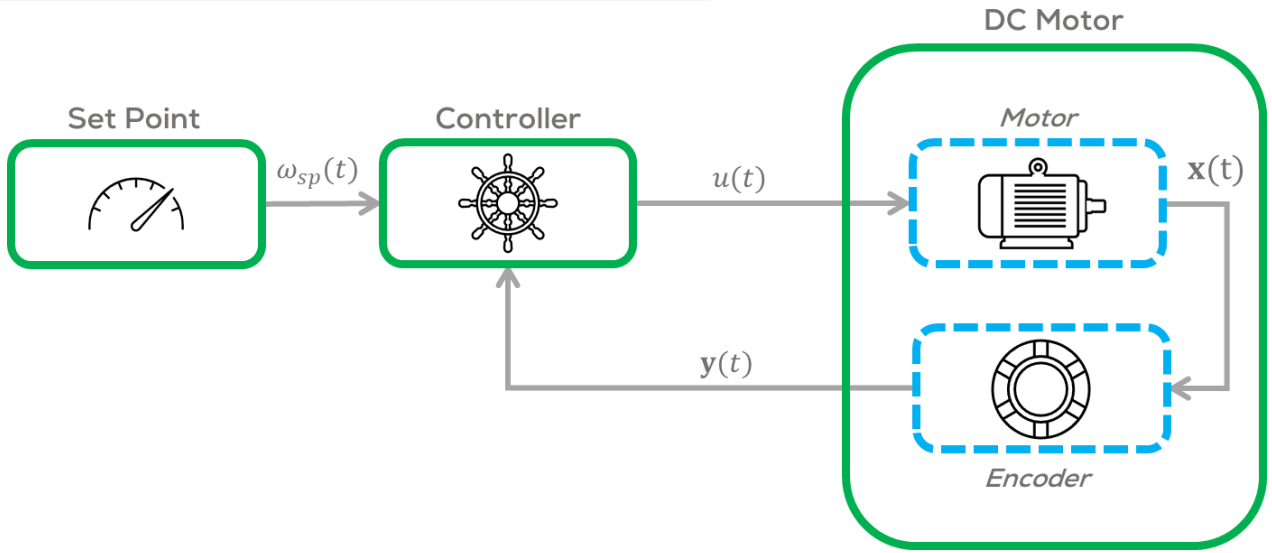
- A direct current (DC) motor is a type of electric machine that converts electrical energy into mechanical energy.
- DC motors take electrical power through direct current and convert this energy into mechanical rotation.
- This is done by using generated magnetic fields from the electrical currents, powering the movement of a rotor fixed within the output shaft.
- The output torque and speed depends upon both the electrical input and the design of the motor.

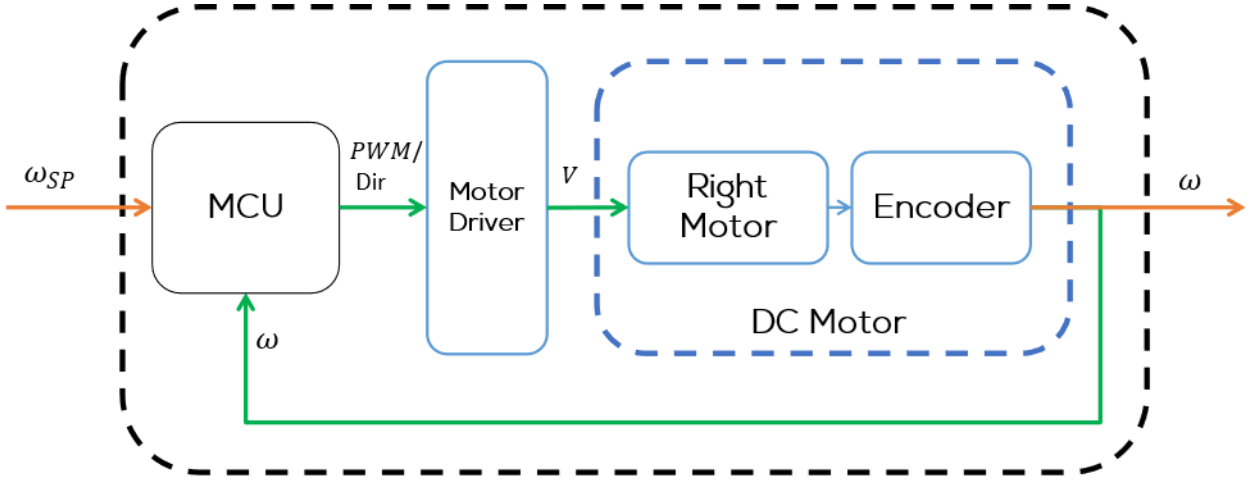
DC Brushed Motor with Encoder.

DC Motor Model Representation.

DC Motors

- In robotics, controllers are used to regulate the rotational speed, angular position or torque, required by the application.
- In robotics this is called low level control.
- For the case of a wheeled mobile robot is a common practice to implement a PID control to regulate the angular speed of the DC motors.
- The regulation of the angular speed or position of a motor, requires different stages.
  - Controller Stage
  - Power Stage (Driver)
  - Plant
  - Sensor Stage



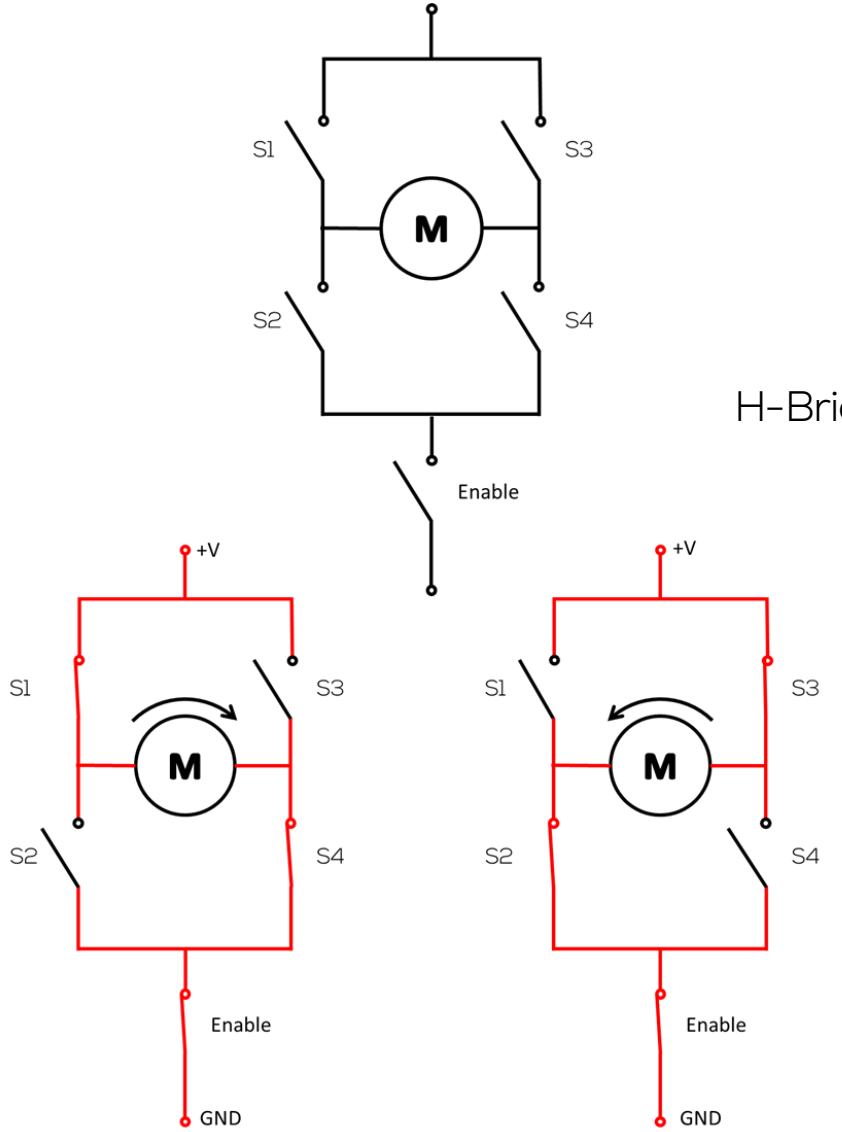


DC Motor Control Diagram*

Motor Driver

- H-bridge is an electronic circuit that switches the polarity of a voltage applied to a load.
- They work using a combination of switching components (mechanical switches, transistors, etc.), as shown in the diagram, to change the polarity to the load.
- H-Bridge Drivers are some of the most common motor drivers used in the control DC motors to run forwards or backwards.

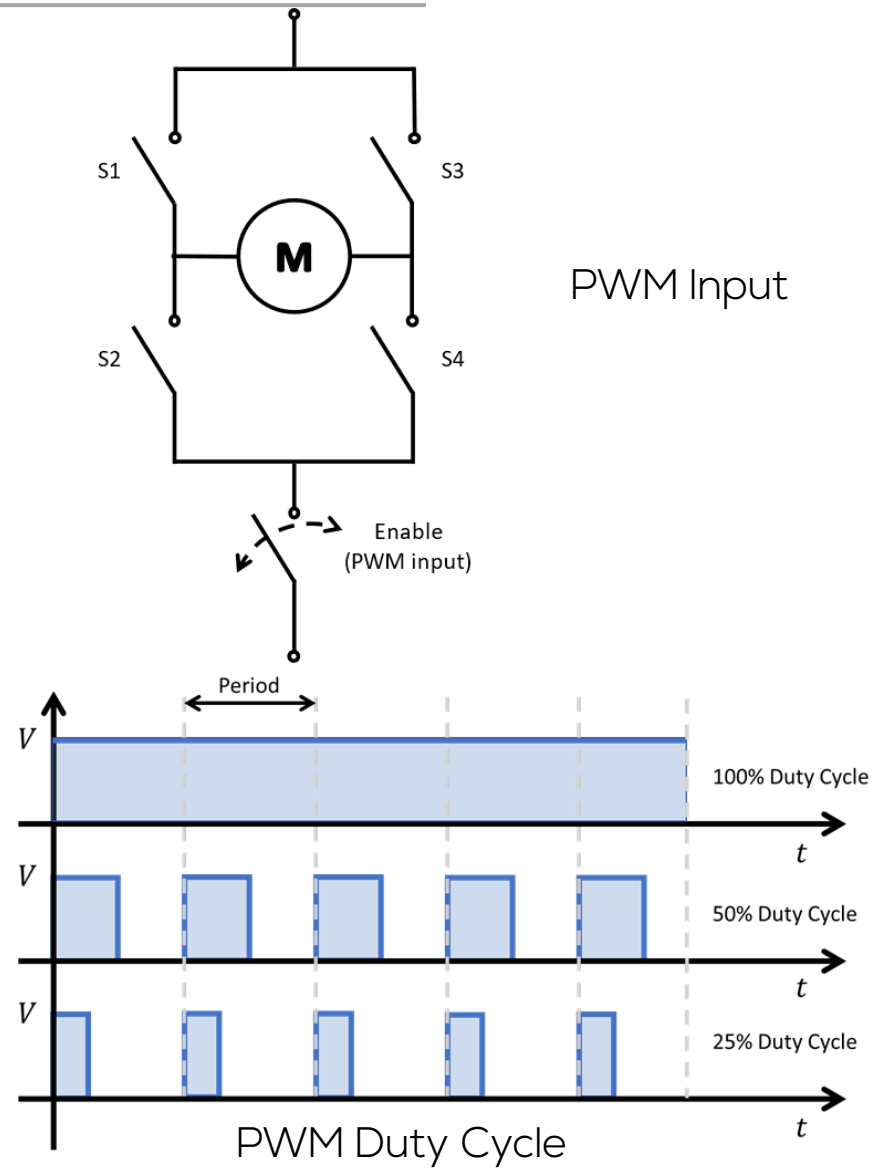
S1	S2	S3	S4	Motor
0	0	0	0	Motor Off
1	0	0	1	Right Turn
0	1	1	0	Left Turn
1	1	0	0	Short Circuit
0	0	1	1	Short Circuit



H-Bridge motor Driver Diagram

Motor Driver

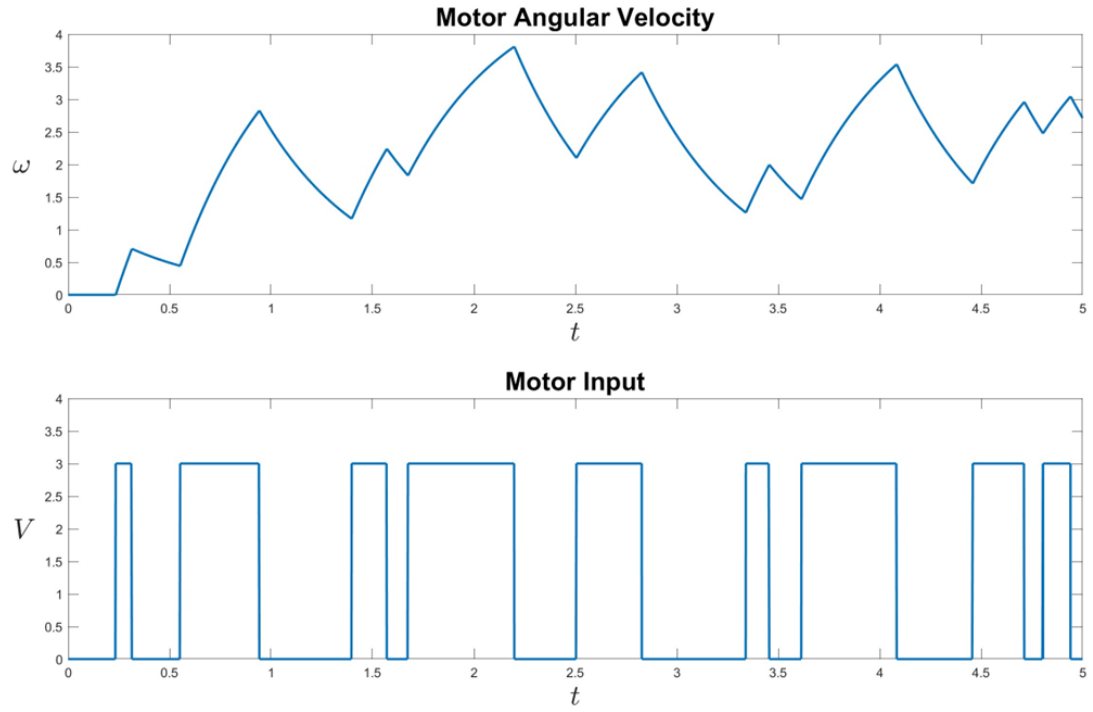
- Another capability of the motor driver is to regulate the angular speed of the motor.
- There are many ways to obtain this result, one of the most common one is to send a PWM (Pulse width modulated signal) to the enable pin of the H-Bridge.
- PWM (Pulse Width Modulation): Is a technique used in engineering to control the average power delivered by an electrical signal, by dividing it into discrete parts.
- In practice this is accomplished by rapidly turning the switch between the load and the source (enable switch), ON and OFF.



Motor Driver

- Given that the motor can be modelled as a second order systems, when applying a PWM voltage as an input, it is possible to observe the output behaviour as in the figure.

- This behaviour, can be used to control the power give to the motor and therefore controlling the motor angular speed.

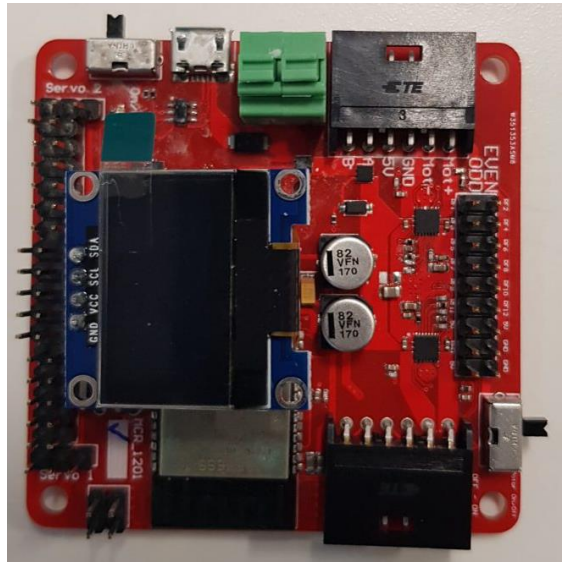


Motor Output/Input

MCU

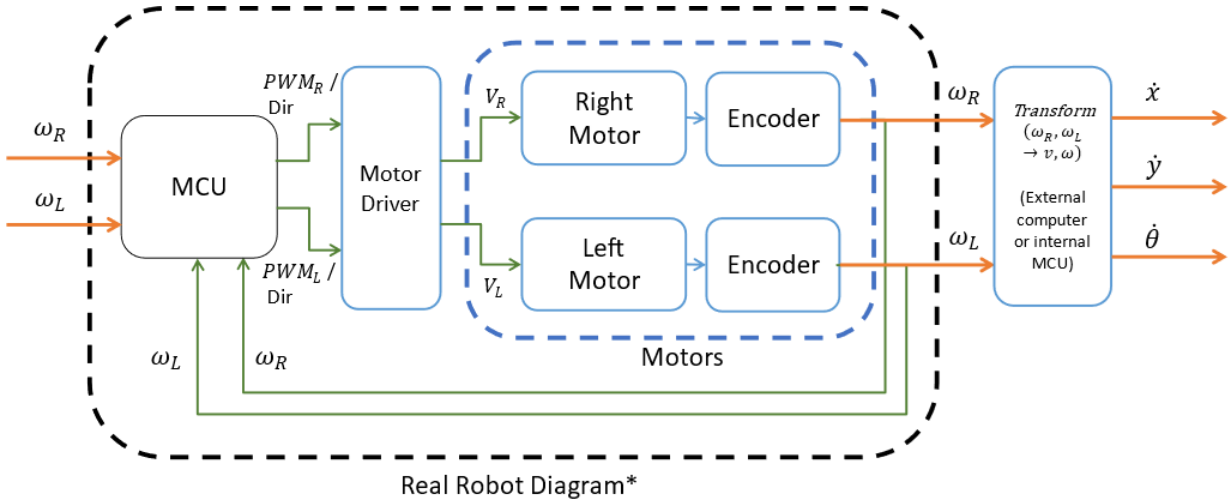
- A microcontroller is a compact integrated circuit. They are made to perform a specific operation in an embedded system.
- In robotics they are usually in charge of the low-level control of the robot, such as motors, and sensors or actuators that require a dedicated and fast controller to work.
- A typical microcontroller includes a processor, memory and input/output (I/O) peripherals on a single chip.

- For the case of the PuzzleBot:
  - ESP32-based Microcontroller
  - Xtensa dual-core 32-bit LX6 microprocessor
  - 520 KB of SRAM
  - WiFi & Bluetooth
  - DC-DC Converter
  - Motor Driver
  - 0.96” I2C LCD Display

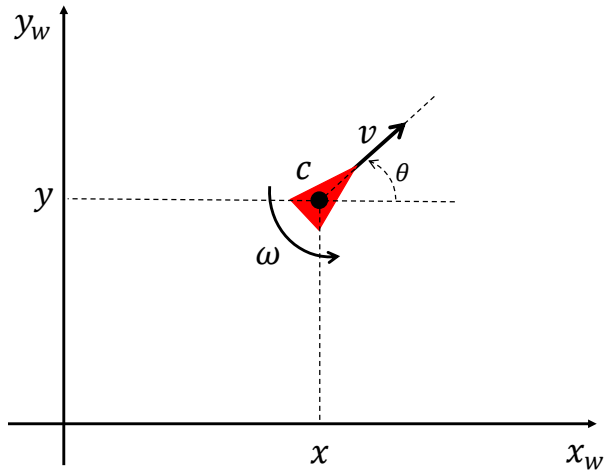


The Mobile Robot

- The real robot, requires the combination and usage of the previously shown sensors and actuators to work.
- For the real robot, a closed control loop for each of the motors is required, in order to reach the required velocities, set by the user.
- In robotics this is called low level control. For the case of a wheeled mobile robot is a common practice to implement a PID control.
- It can be observed that the inputs to the robot are the speed of the wheels $\mathrm{u(t)} = [\omega_r, \omega_l]$, and the outputs are the wheel velocities that can be transformed into robot velocities $\mathrm{y(t)} = [V, \omega]$.
- The inner loop controller and actuators will determine the behavior of the robot.



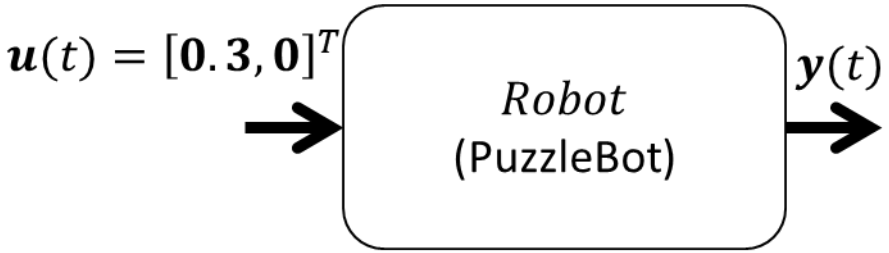
Real Robot Diagram*



Differential Drive Robot Representation.

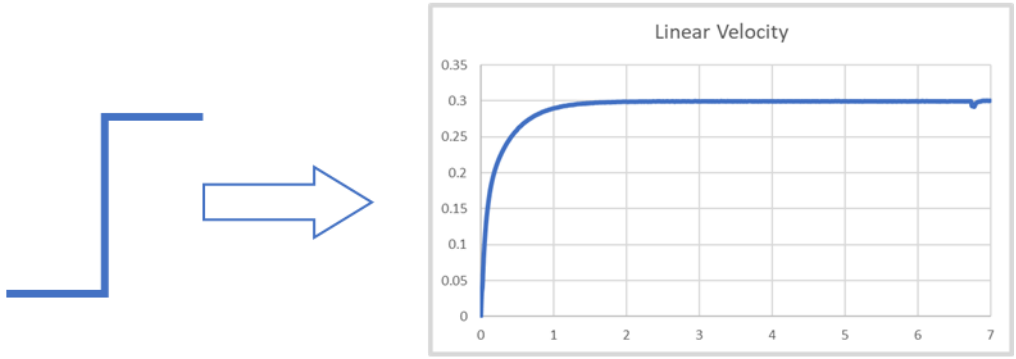
Open Loop Control

- Let the system (in this case the robot) to be represented as a black box and let only constant linear velocity of $v = 0.3\frac{m}{s}$ to be the input to the system (step input).



Black Box Model

- It can be observed that the output of the system matches the input reference at steady state. Therefore, the gain of the system is said to be 1. **This gain is only for the robot system for any other system the system's gain must be estimated.*



Input

Output

- A more in-depth analysis of this situation reveals that this is due to the inner controllers of the robot, allows the robot to follow the reference after some transient. The same thing happens with the angular speed.

Open Loop Control

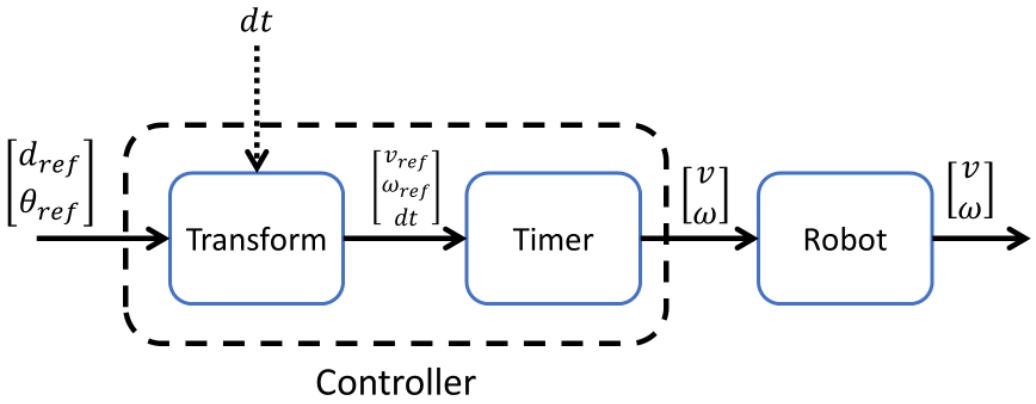
- Knowing that the system's output will follow the input after a transient, a question arises. Can we use this information to move the robot a certain distance?
- Since we know that the output follows the input, it is possible to approximate the distance and angle traveled by using the following formula.

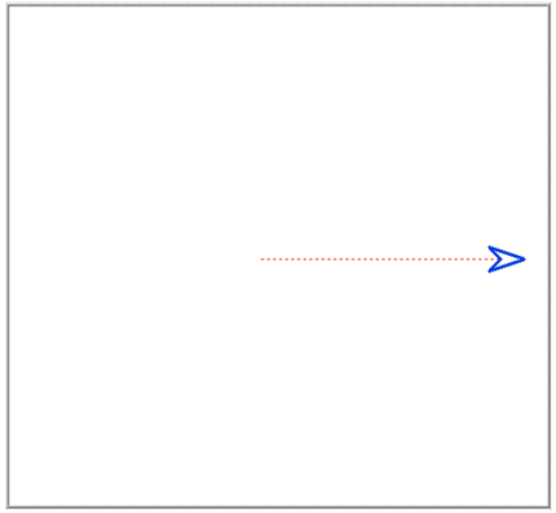
$$d = v \cdot dt = r\left(\frac{\omega_r + \omega_l}{2}\right) \cdot dt$$

$$\theta = \omega \cdot dt = r\left(\frac{\omega_r - \omega_l}{l}\right) \cdot dt$$

where dt is the time since the input was applied.

- Therefore, it is possible to use a timer to control the distance and angle travelled by the robot.
- This is called open loop control.

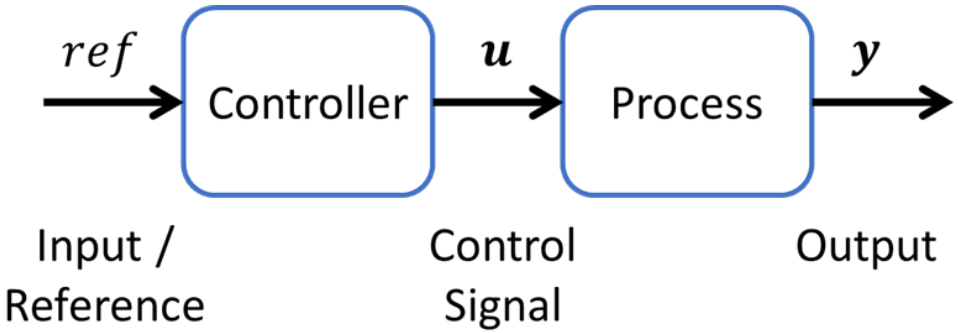


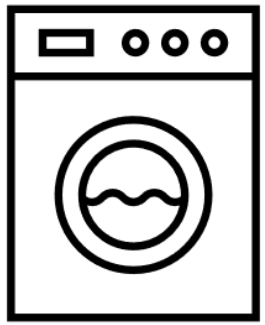


Open Loop Control

Open Loop Control

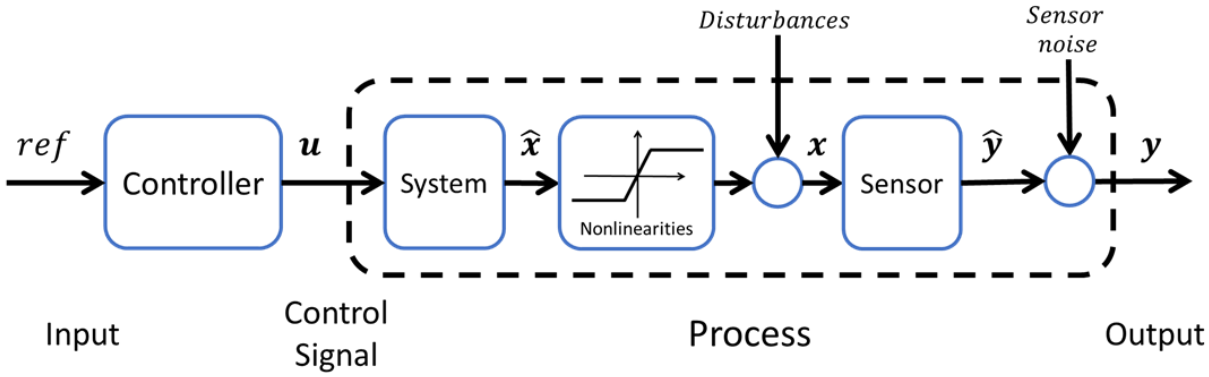
- Open Loop Control System is a system in which the control action is independent of the output of the system.
- In this type of control, the output is regulated by varying the input or reference.
- The output of the system is determined by the current state of the system and the inputs that are received from the controller.
- Some examples of this type of control systems are Windows, window blinds, washing machines, microwave ovens, hair drier, bread toaster, Door Lock System, some stepper motors, turning on/off lights, some remote-controlled applications, etc.

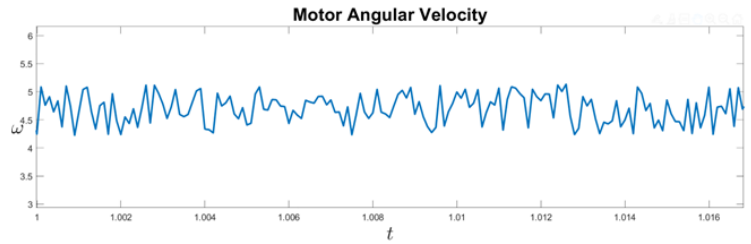




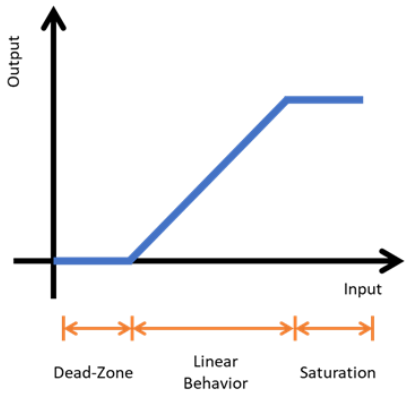
Open Loop Control

- Every process, present disturbances, nonlinearities and noise.
- Disturbances and noise come from the environment that surrounds the system.
  - On the differential drive robot, some disturbances can come from wheels slipping, obstacles on the environment, different types of terrain, etc.
  - On the other hand, the noise is present when reading the values of the motor encoders (Sensors).
- Nonlinearities are an intrinsic characteristic of a system in which the output does not linearly follow the input (output not proportional to the input).
  - For the case of the robot, the nonlinear behavior can be seen as a saturation and dead-zone in the motors and the robot itself.





Noisy Signal (Angular velocity)

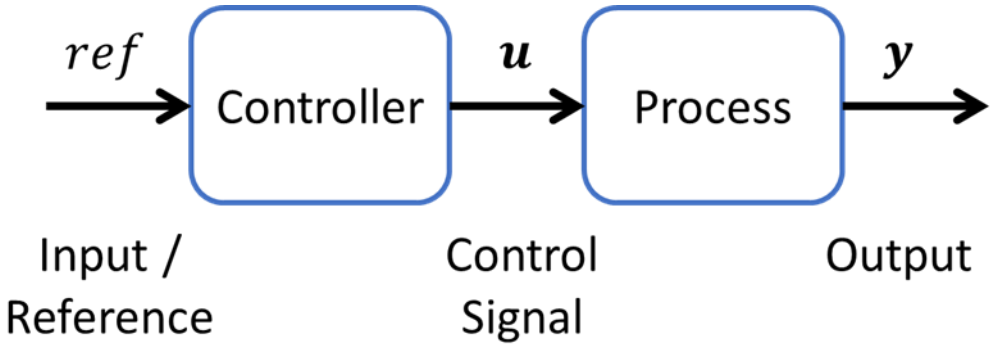


Nonlinear behavior

Open Loop Control

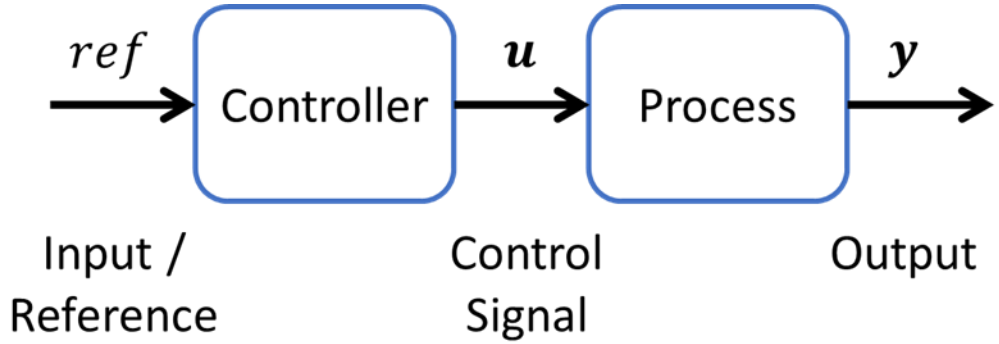
Open Loop Advantages:

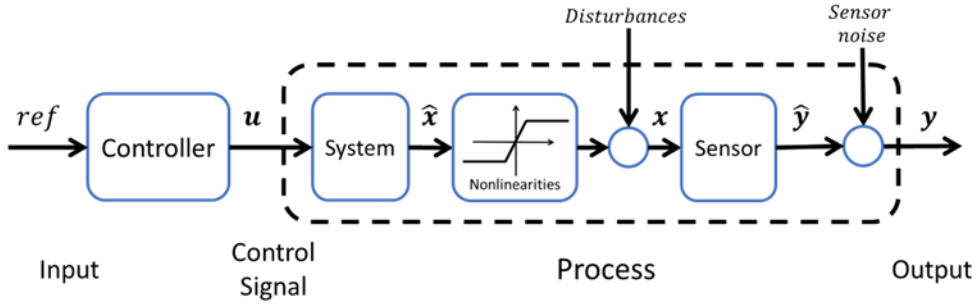
- Simple to design and implement, provided that the user has some experience with the system.
- The cost for design, implement and maintain is relatively low compared with other controllers.
- Maintenance is considered simple, no high technical level required (for most of the controllers).
- The behavior of the controller is quite stable, provided that the system is in a controlled environment, such that the process does not present big disturbances, or the process is not dangerous when unsupervised.



Open Loop Control

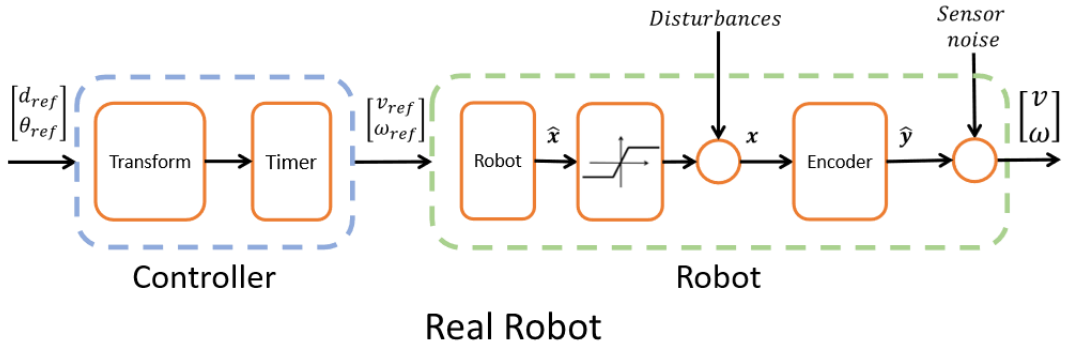
- Open Loop Disadvantages:
  - This type of control is not robust against disturbances (cannot correct the output in the presence of a disturbance).
  - An open-loop system has no self-regulation or control action over the output value.
  - Not reliable
  - The input depends on the experience of the user.
  - Each input determines a fixed operating point of the system.
  - Controller must be altered manually in case of an output disturbance or uncalibrated controller.
  - Requires re-calibration often.
  - Prone to errors in the output and control signal
  - Does not take into consideration changes on the process over time.
  - Also, there is no chance to correct the transition errors in open loop systems so there is more chance to occur errors.

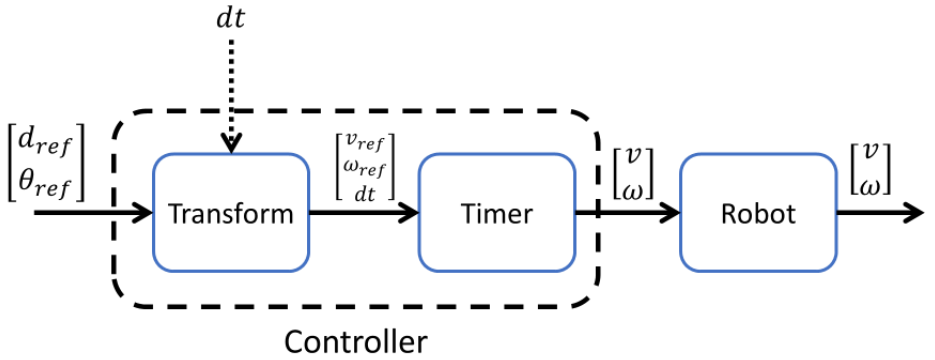




Open Loop Control: Mobile Robot

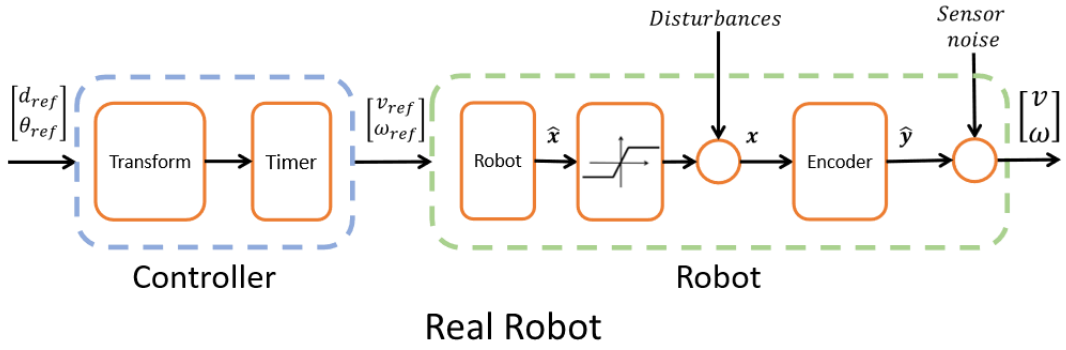
- As stated before, open loop control can be used to control de position of the robot.
- When designing this controller, the nonlinear behavior, perturbations and noise should be taken into consideration to make the controller work on the linear part of the robot and make it as robust as possible.
- In this case, the real mobile robot is a nonlinear system, due to the maximum and minimum velocities of the motors therefore it presents saturation and dead-zone. *Other nonlinear behaviour are present, but they are not relevant in this case.
- The linear behaviour is present within a region in which the actuators (motors) output are proportional to the input.



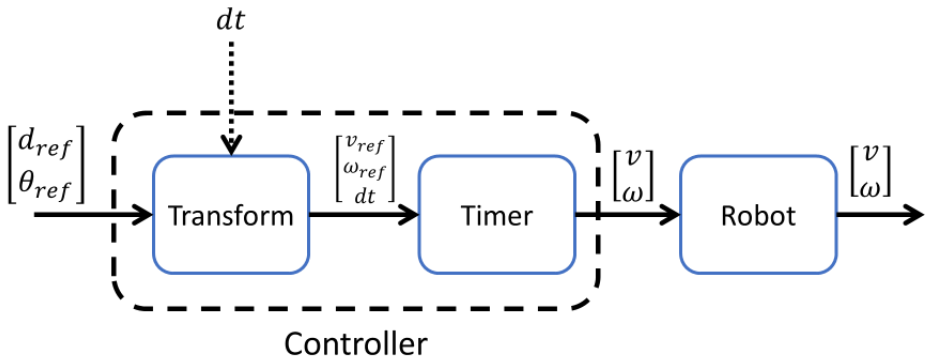


Open Loop Control: Mobile Robot

- Noise and perturbations can also be found in the robot.
- As said before, noise can be found in the encoder reading, the noise can be due to electromagnetic noise or radiation, mechanical unbalance of encoders, etc.
- Disturbances are due to the mechanical unbalance of the wheels, wheel slippage with the floor, defects of productions, changes in the environment, etc.
- Depending on the simulator, the dynamical behavior of the robot can be linear or nonlinear and can consider the noise and disturbances .



Real Robot

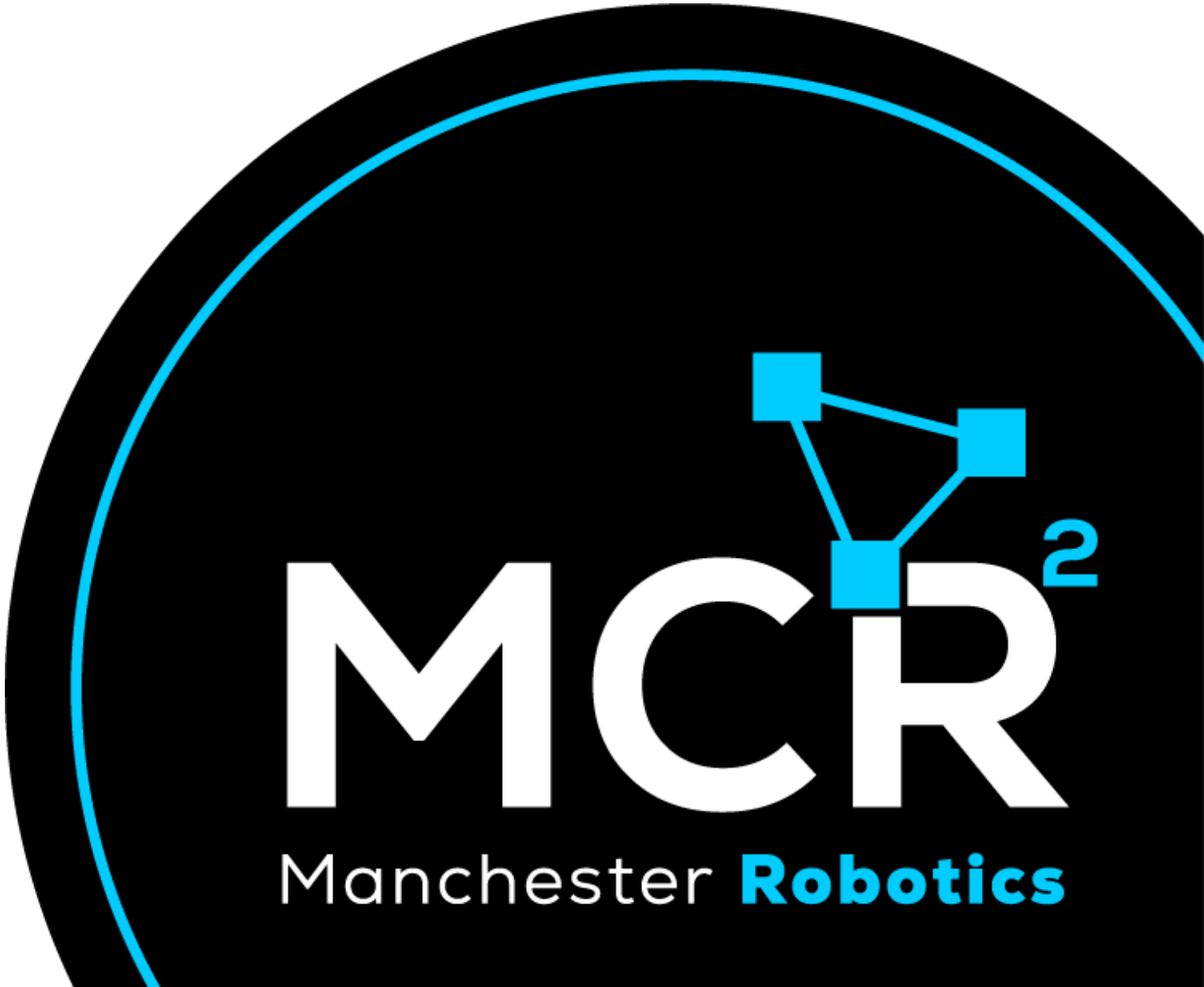


Simulated robot

{Learn, Create, Innovate};

Gazebo

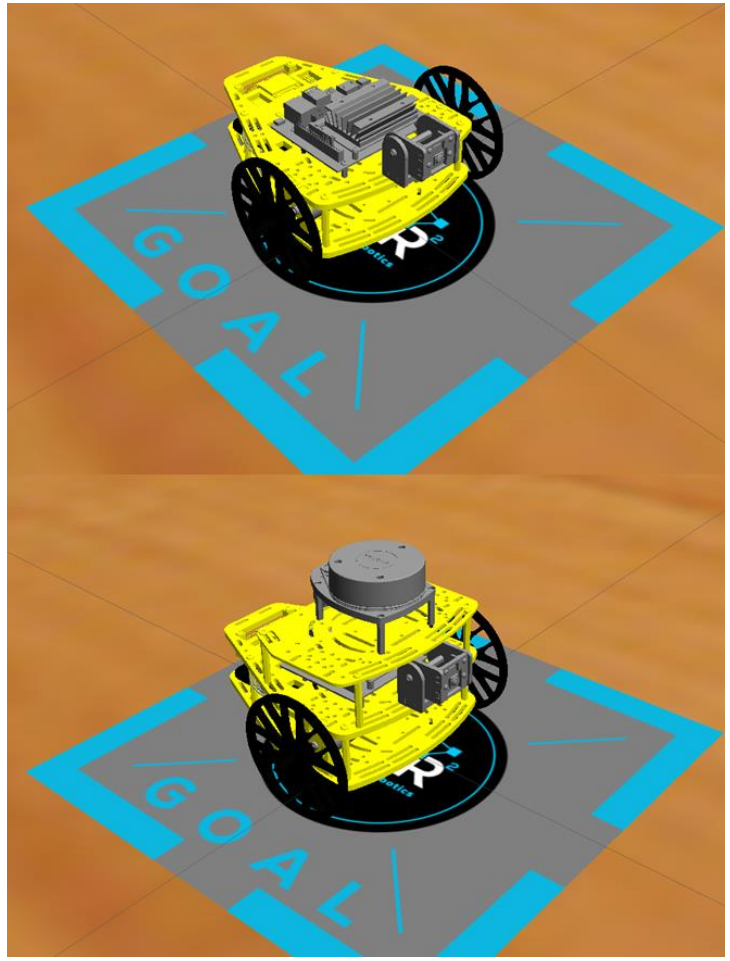
Activity 1: Puzzlebot Gazebo



Introduction

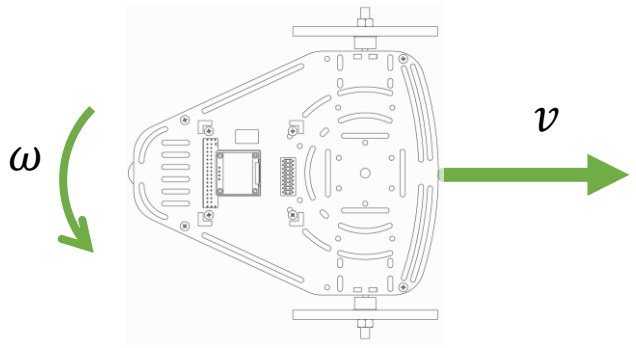
Introduction

- The following activity will help you build a simple controller for the Puzzlebot
- This activity consist of creating a node that sends commands to the real robot and the gazebo simulation and move in a straight line for a period of time.
- This activity will use Gazebo as a dynamical simulator.
- For more information about how to use the MCR2 Puzzlebot Gazebo Simulator, go to the file: "MCR2_Puzzlebot_Gazebo_Simulator"



Activity 1: Moving a Puzzlebot

Straight line



Drive the robot 2 meters in a straight line, turn and come back.

Objectives:

- Create a ROS node that sends commands to the robot gazebo simulation and move in a straight line for 2 meters.
- The Puzzlebot requires the commands to be published on a Twist() Message on the /puzzlebot/cmd_vel topic.

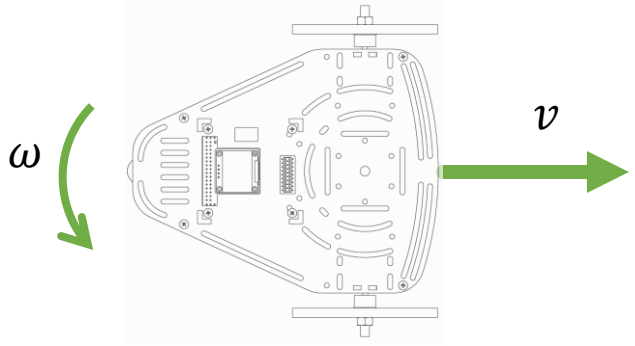
```
geometry_msgs/Twist

geometry_msgs/Vector3 linear
  float64 x
  float64 y
  float64 z
geometry_msgs/Vector3 angular
  float64 x
  float64 y
  float64 z
```

Activity 1: Moving a Puzzlebot

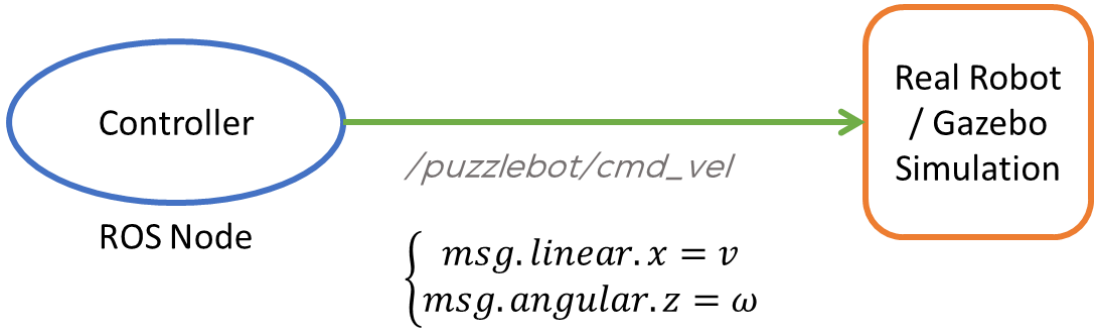
Information

- The Puzzlebot being a nonholonomic DDR, only accepts commands on the “linear x” and “angular z” parts of the message. Corresponding to linear velocity v and heading angular velocity ω.




```
geometry_msgs/Twist

geometry_msgs/Vector3 linear
  float64 x
geometry_msgs/Vector3 angular
  float64 z
```



Activity 1: Moving a Puzzlebot

Instructions

For this project two options are given to the students, using the template or making a package from scratch.

Template:

- Download the template from Activity 1, add it to your workspace and compile it using catkin_make.

Create a package:

- Make a new package called “robot_control”, with the following packages: rospy, std_msgs, tf2_ros, visualization_msgs tf_conversions, geometry_msgs, rosgraph_msgs

```
$ catkin_create_pkg robot_control rospy std_msgs tf2_ros
visualization_msgs tf_conversions geometry_msgs tf2_geometry_msgs
rosgraph_msgs
```

- Create a node called activity1.py inside the scripts folder

```
$ mkdir scripts && touch scripts/activity1.py
```

- Give executable permission to the file

```
$ cd ~/catkin_ws/src/markers/scripts/
$ sudo chmod +x activity1.py
```

- Modify the CMake file to include the newly created node to the

```
catkin_install_python(PROGRAMS scripts/activity1.py
  DESTINATION ${CATKIN_PACKAGE_BIN_DESTINATION})
```

Activity 1: Moving a Puzzlebot


```python
#!/usr/bin/env python
import rospy
import numpy as np
from geometry_msgs.msg import Twist
from rosgraph_msgs.msg import Clock
# Setup Variables to be used
first = True

# Declare the  process output message
controlOutput = Twist()
clock_msg = None

#Define the callback functions
def clock_callback(msg):
    global clock_msg
    clock_msg = msg

#Stop Condition
def stop():
  #Setup the stop message (can be the same as the control
message)
    controlOutput.linear.x = 0.0
    controlOutput.angular.z = 0.0
    control_pub.publish(controlOutput)
    print("Stopping")
```

```python
if __name__=='__main__':
    #Initialise and Setup node
    rospy.init_node("open_loop_controller")

    # Configure the Node
    loop_rate = rospy.Rate(rospy.get_param("~node_rate",10))
    rospy.on_shutdown(stop)

    #Setup de publishers
    control_pub = rospy.Publisher("/puzzlebot/cmd_vel", Twist, queue_size=1)

    #Setup Subscribers
    clock_sub = rospy.Subscriber('/clock', Clock, clock_callback, queue_size=1)

    print("The controller is Running")
```

Activity 1: Moving a Puzzlebot

```
try:

    while clock_msg == None:
        rospy.loginfo("Waiting for Gazebo")
        loop_rate.sleep()

    rospy.loginfo("Clock synchronized")
    rospy.sleep(0.5)

    #Run the node
    while not rospy.is_shutdown():

        if (first):
            controlOutput.linear.x = 0.0
            controlOutput.angular.z = 0.0
            control_pub.publish(controlOutput)
            rospy.loginfo("Motion Initiated")
            first = False
```

When connecting Gazebo with ROS, the user Nodes must wait until Gazebo Publishes the topic "/clock" to synchronize the Gazebo simulation time with ROS time.

Otherwise, the nodes might start without Gazebo and could cause instability.

"For calculations of time durations when using simulation time, clients should always wait until the first non-zero time value has been received before starting, because the first simulation time value from /clock topic may be a high value."

More information here.

Activity 1: Moving a Puzzlebot


```
else:
    # Move Forward
    controlOutput.linear.x = 0.3
    controlOutput.angular.z = 0.0
    control_pub.publish(controlOutput)
    rospy.sleep(4)  # Move for 2 seconds

    # Stop
    controlOutput.linear.x = 0.0
    controlOutput.angular.z = 0.0
    control_pub.publish(controlOutput)
    rospy.sleep(1)
    # Turn
    controlOutput.linear.x = 0.0
    controlOutput.angular.z = 1.57  # 90-degree turn
    control_pub.publish(controlOutput)
    rospy.sleep(2)

    # Stop
    controlOutput.linear.x = 0.0
    controlOutput.angular.z = 0.0
    control_pub.publish(controlOutput)
    rospy.sleep(1)
```

```
        # Move Forward
        controlOutput.linear.x = 0.3
        controlOutput.angular.z = 0.0
        control_pub.publish(controlOutput)
        rospy.sleep(4)  # Move for 2 seconds


        # Stop
        controlOutput.linear.x = 0.0
        controlOutput.angular.z = 0.0
        control_pub.publish(controlOutput)
        rospy.sleep(1)

        rospy.loginfo("Motion Complete")
        rospy.signal_shutdown("Square Completed")


    #Wait and repeat
    loop_rate.sleep()

except rospy.ROSInterruptException:
    pass #Initialise and Setup node
```

Activity
Some tips and tricks

- Use `rospy.get_time()` to get the current time.
- If the robot is not moving, check your topics with `rostopic echo` and `rostopic pub`
- Ensure your python file is executable: `sudo chmod +x <path_to_file>.py`
- Ensure you source your file: `source devel/setup.bash`

Activity 1: Results

- Save and recompile the project.
- Build the program using “catkin_make”

```
$ catkin_make
$ source devel/setup.bash
```

- Launch Gazebo in one terminal

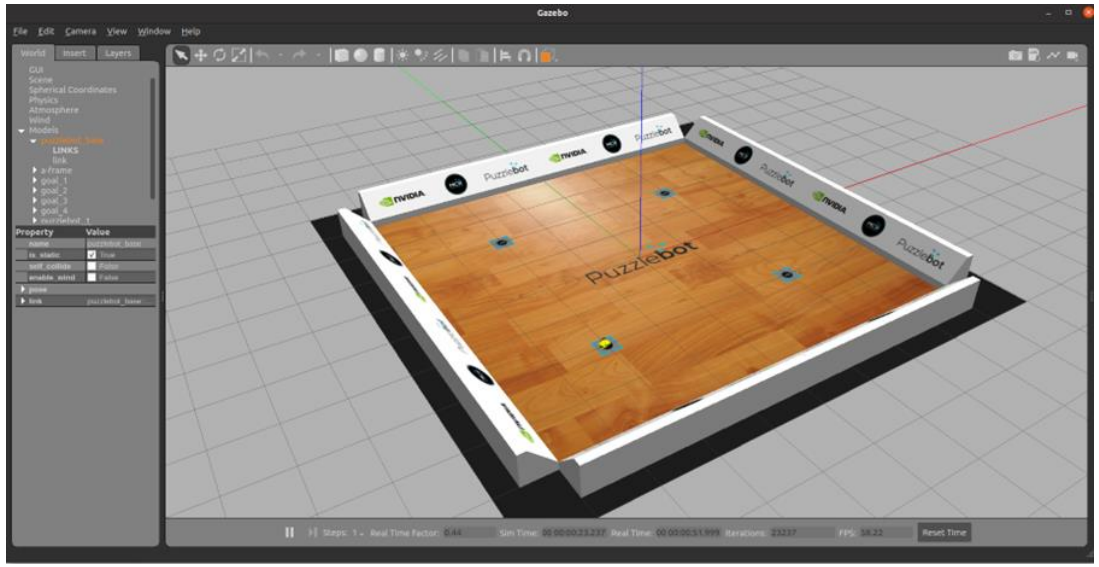
```
$ roslaunch puzzlebot_gazebo puzzlebot_gazebo.launch
```

- Open another terminal and run the Activity

```
$ rosrun robot_control activity1.py
```

- The user can observe or “echo” the “/motor_output” topic in a terminal or using the “rqt_plot”

- The robot should move forward, rotate 180 deg. and come back to the initial position.



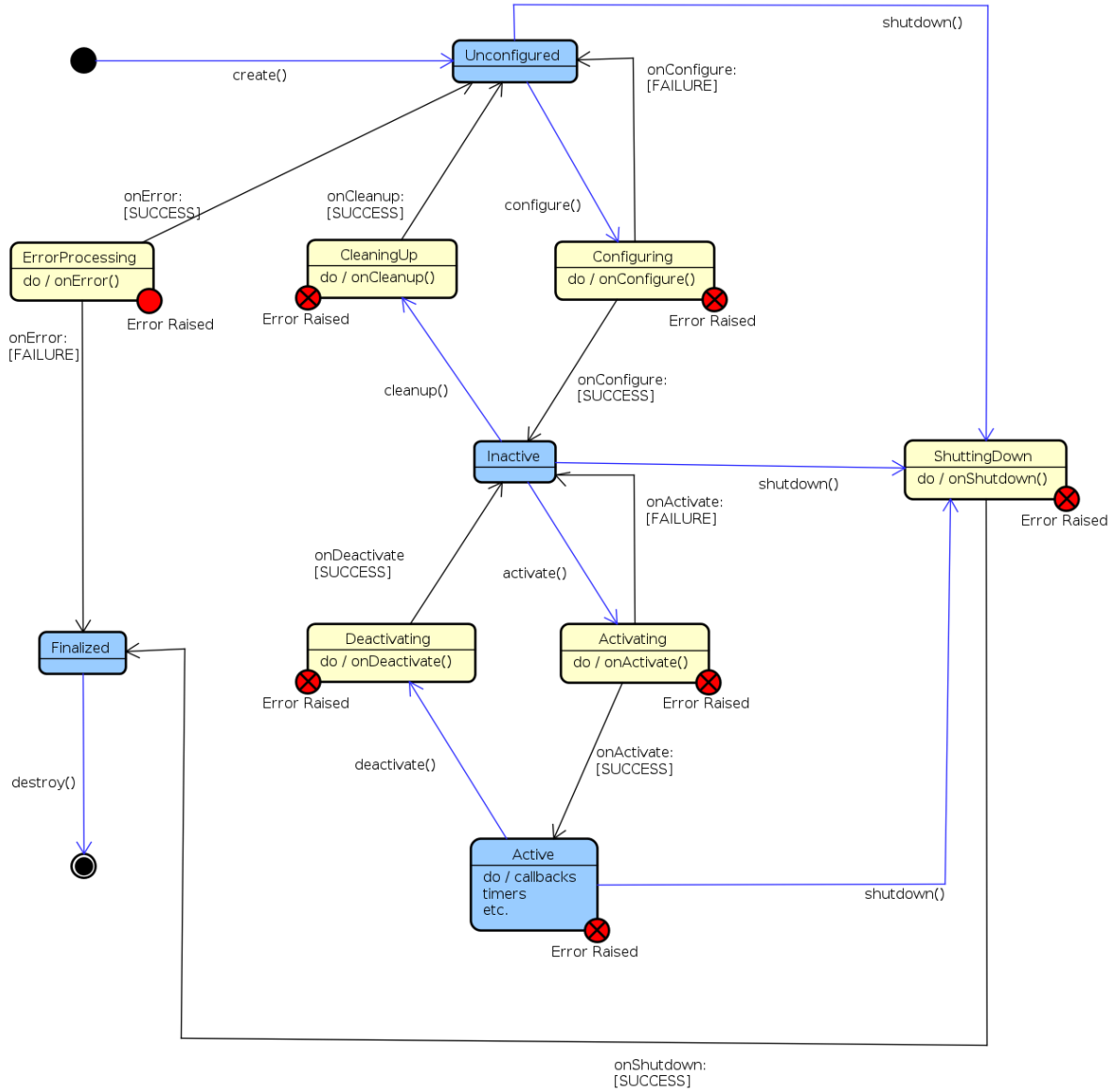
State Machines

Definition

- A state machine is a computational model that transitions between predefined states based on inputs or conditions.
- **Finite State Machines (FSM)** - Fixed number of states.

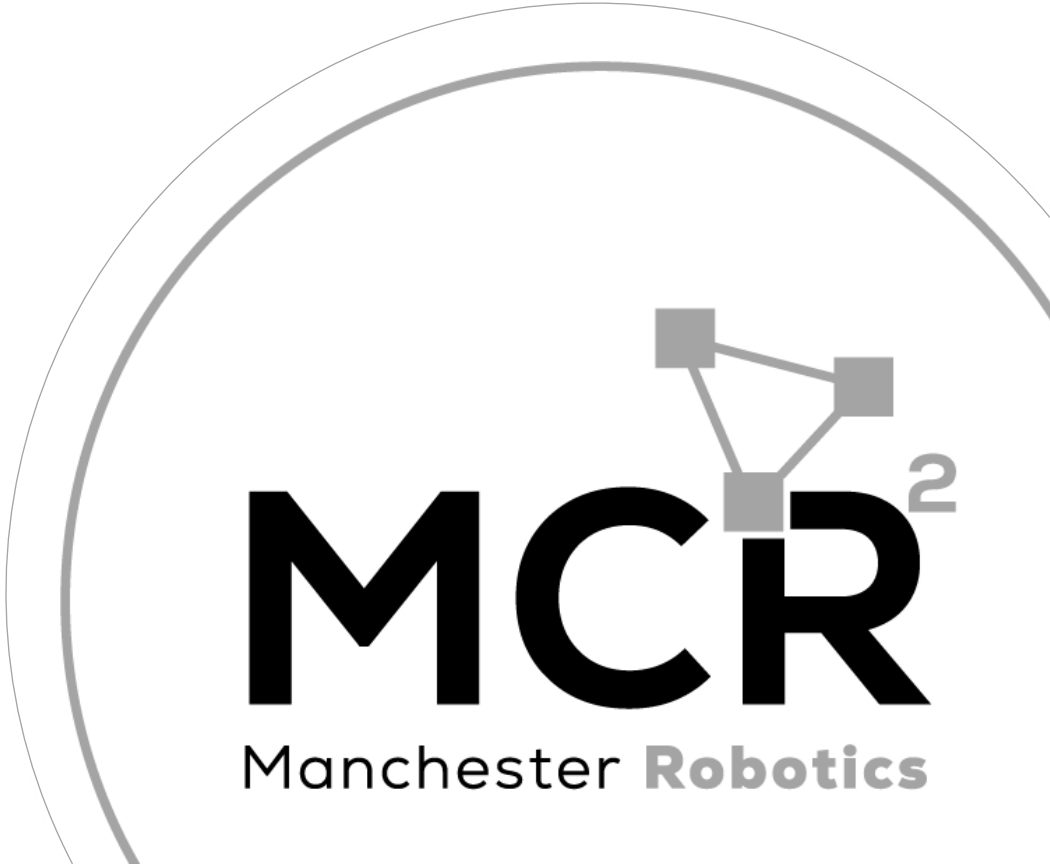
Why use state machines?

- Structured way to control robotic behaviour.
- Ensures modular, scalable, and readable code.



Thank you

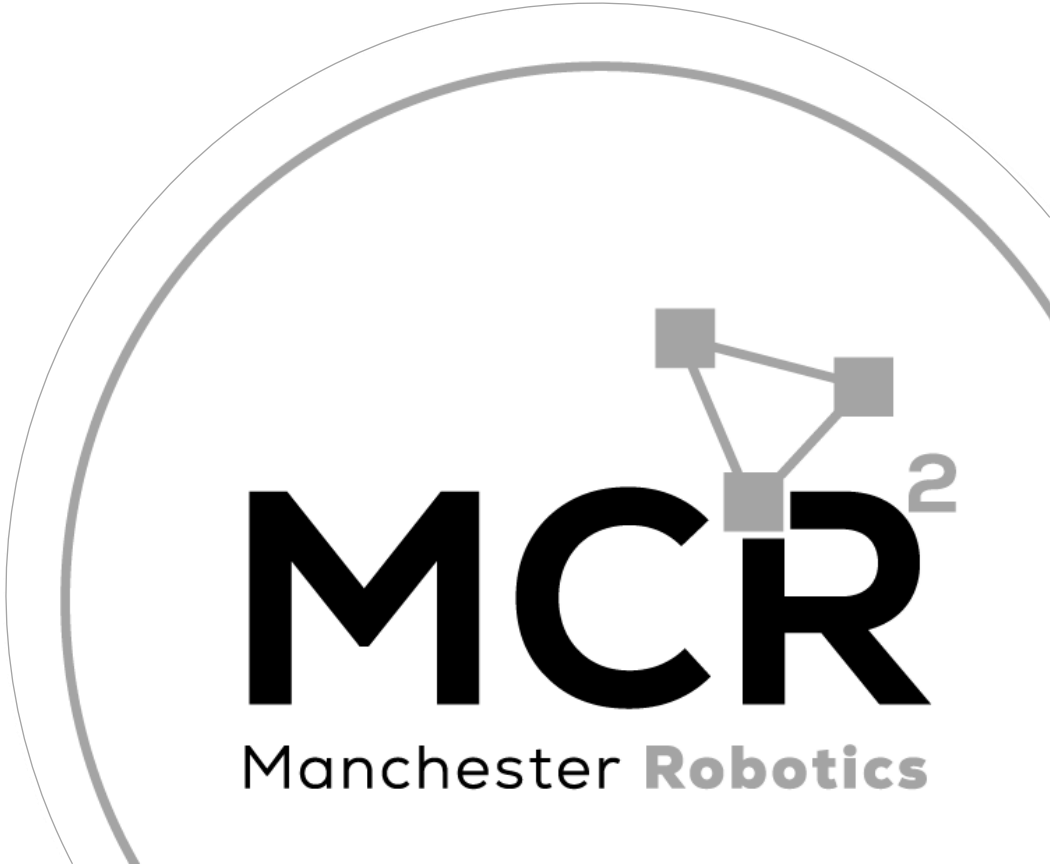
{Learn, Create, Innovate};



T&C

Terms and conditions

{Learn, Create, Innovate};



Terms and conditions

- *THE PIECES, IMAGES, VIDEOS, DOCUMENTATION, ETC. SHOWN HERE ARE FOR INFORMATIVE PURPOSES ONLY. THE DESIGN IS PROPRIETARY AND CONFIDENTIAL TO MANCHESTER ROBOTICS LTD. (MCR2). THE INFORMATION, CODE, SIMULATORS, DRAWINGS, VIDEOS PRESENTATIONS ETC. CONTAINED IN THIS PRESENTATION IS THE SOLE PROPERTY OF MANCHESTER ROBOTICS LTD. ANY REPRODUCTION, RESELL, REDISTRIBUTION OR USAGE IN PART OR AS A WHOLE WITHOUT THE WRITTEN PERMISSION OF MANCHESTER ROBOTICS LTD. IS STRICTLY PROHIBITED.*
- *THIS PRESENTATION MAY CONTAIN LINKS TO OTHER WEBSITES OR CONTENT BELONGING TO OR ORIGINATING FROM THIRD PARTIES OR LINKS TO WEBSITES AND FEATURES IN BANNERS OR OTHER ADVERTISING. SUCH EXTERNAL LINKS ARE NOT INVESTIGATED, MONITORED, OR CHECKED FOR ACCURACY, ADEQUACY, VALIDITY, RELIABILITY, AVAILABILITY OR COMPLETENESS BY US.*
- *WE DO NOT WARRANT, ENDORSE, GUARANTEE, OR ASSUME RESPONSIBILITY FOR THE ACCURACY OR RELIABILITY OF ANY INFORMATION OFFERED BY THIRD-PARTY WEBSITES LINKED THROUGH THE SITE OR ANY WEBSITE OR FEATURE LINKED IN ANY BANNER OR OTHER ADVERTISING.*