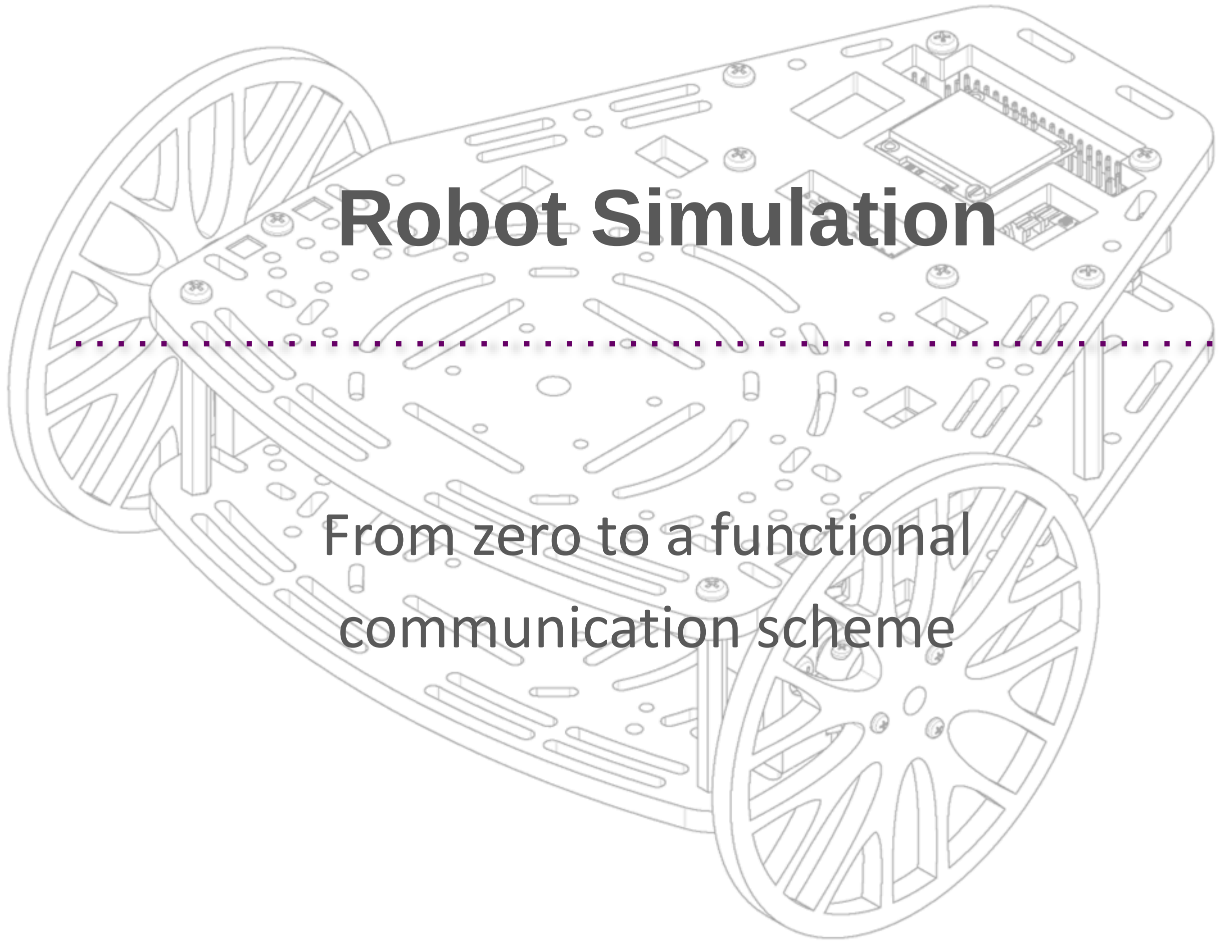


Robot Simulation

From zero to a functional communication scheme

Session 3: Robot Simulation.

I. Differential Drive Basics (1 hour approx.) *Video*
   - Differential Drive System (Nonholonomic)
   - Sensors and Actuators (Input and outputs) (Puzzlebot)
   - Robot Kinematics

II. ROS Visualisation Tools (45 min approx.) *Video*
   - Robot Model for Simulation
   - ROS Visualisation Interfaces
     - What is RVIZ and Gazebo
     - Comparison
   - RVIZ
     - Walk-trough RVIZ (Puzzlebot model)
   - Gazebo
     - -Connection between ROS and Gazebo
     - -ROS Control
     - -Walk-through Gazebo (Puzzlebot Model)

III. Activity (45 hours approx.) Live

IV. Create a basic World in Gazebo (30 min approx.) Video
   - Create world in Gazebo

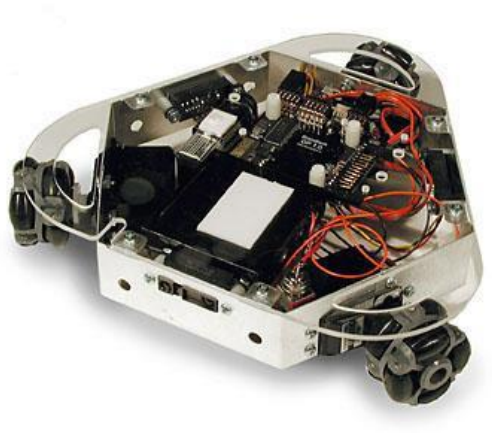
Requirements: Laptop, ROS preinstalled, Ubuntu preinstalled, Basic Knowledge of Python.

Differential Drive Basics

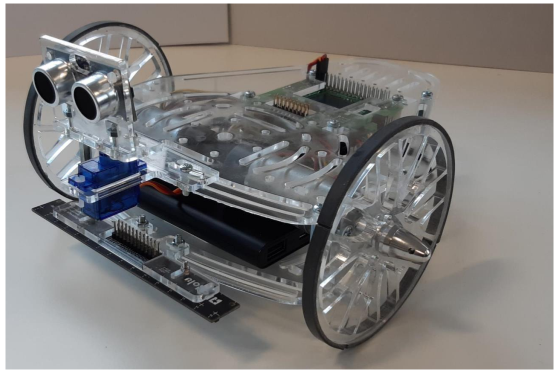
Mobile Robots

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

- In the following sessions we will focus on differential drive robots, also known as "differential wheeled robots".



Holonomic Robot
Acroname ©.



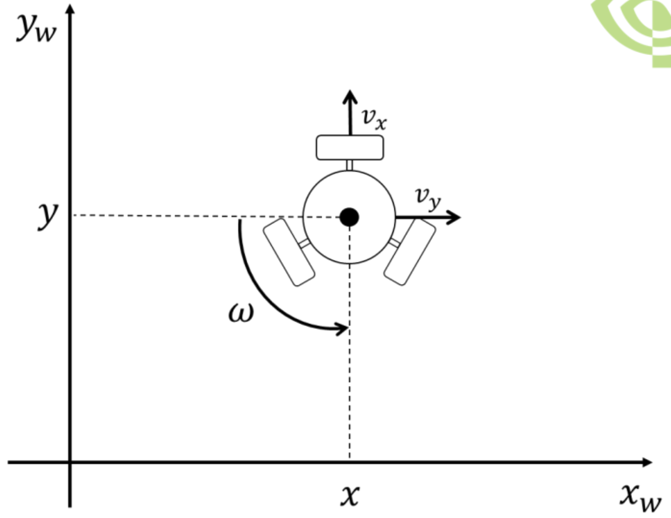
Differential-drive
Puzzlebot ©.

Holonomic vs. Nonholonomic Systems

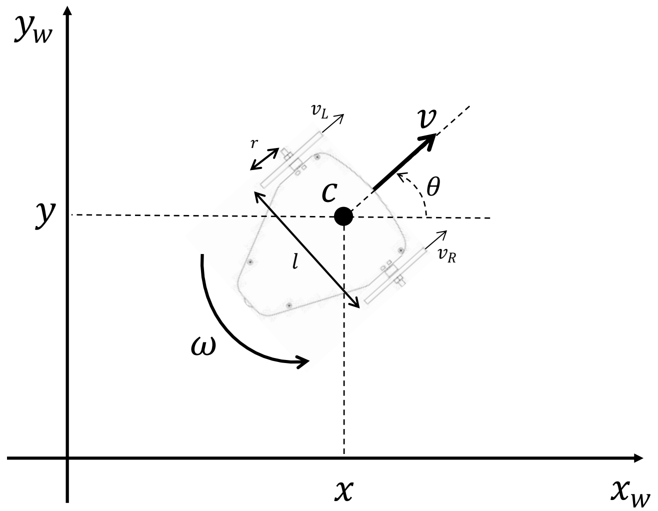
- Mobile robots can be classified as "Holonomic" or "Nonholonomic“.
- This classification depends on the relationship between controllable and total degrees of freedom of a robot.

- **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.



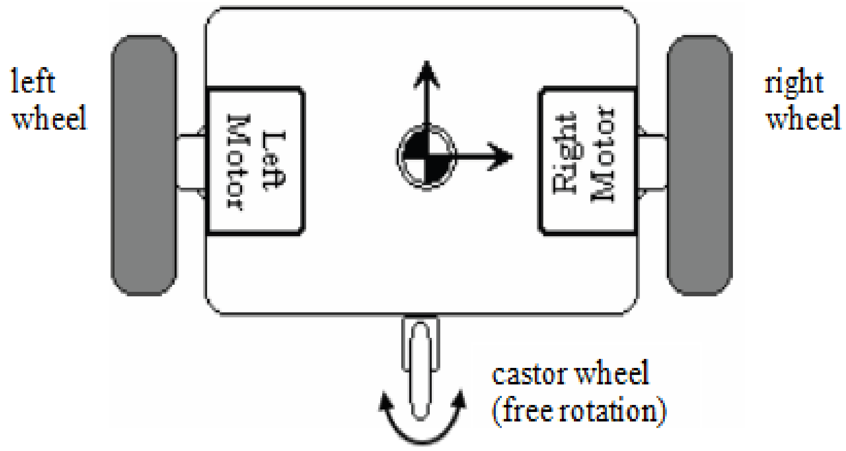
Holonomic Robot

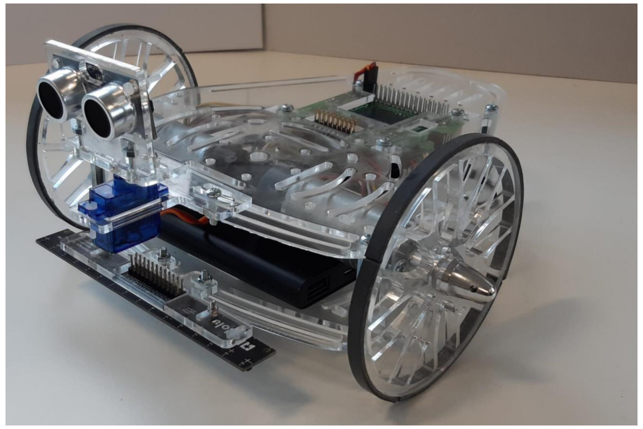


Nonholonomic Robot

Differential Drive

- Also known as differential wheeled robots.
- These are mobile robots whose movement is based on two separately driven wheels placed on either side of the robot body.
- It can thus change its direction by varying the relative rate of rotation of its wheels, thereby requiring no additional steering motion.





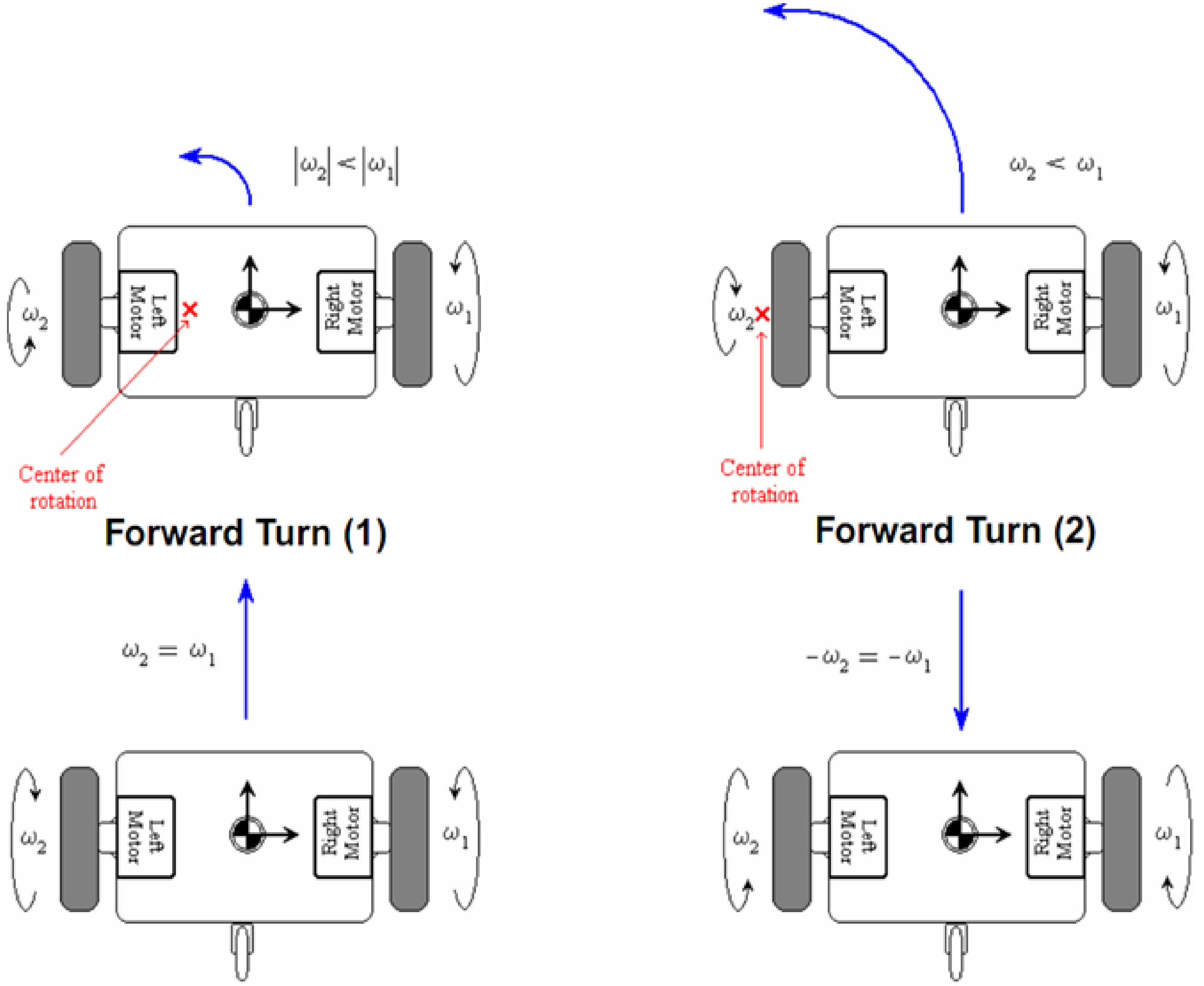
Differential-drive robots

Differential Drive

- As stated before, the robot will change direction, depending on the speed of the wheels at each side.

- When the two wheels are rotating at the same speed i.e., $\omega_1 = \omega_2$ or $-\omega_1 = -\omega_2$ the robot will drive forward or backward depending on the direction of rotation.

- The centre of rotation will vary depending on the difference of the wheel speeds. Being at the centre of the robot when $\omega_1 = -\omega_2$ or vice versa.



Robot Modelling (Kinematics)

- In engineering, a System is an entity that consists of interconnected components, built with a desired purpose. For this case, the system is said to be the differential drive robot.

- Systems can be modelled depending on their dynamical behavior. This models describe the behavior of a system using mathematical/computational concepts and language.
    - Outputs depend on the present and past values of the inputs.
    - Changes over time.
    - Sometimes called dynamic systems or sequential systems.
    - Mathematically described with differential or difference equations.

- **Dynamic Modelling**: Considers different time varying phenomena and the interaction between motions, forces and material properties.

- **Kinematic modelling:** Studies the motion of a robot or mechanism under a set of constraints, regardless of the forces that cause it. Represent the relationship between the robot motor speeds (inputs) and the robot state.

DC Brushed Motor with encoder

$$\xrightarrow{In} \boxed{\frac{K}{s^2 + 2\zeta\omega_n s + \omega_n^2}} \xrightarrow{Out}$$

2nd Order System

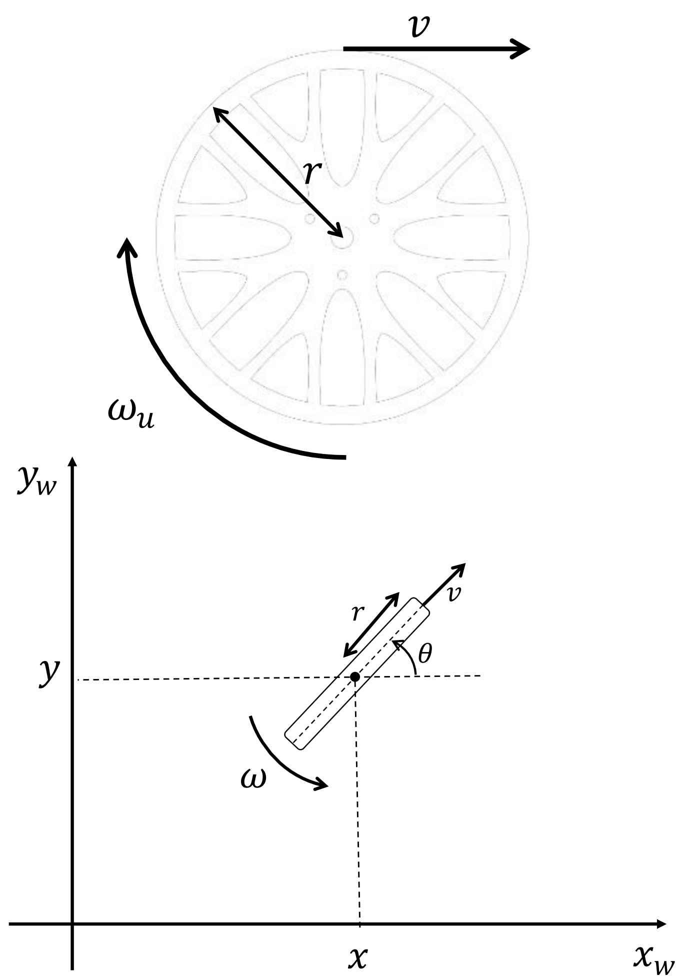
The Unicycle

- In order to derive the kinematics of a differential drive robot, it is instructive to first consider a simpler system; the unicycle.
- Ignoring balancing concerns, there are two action variables, i.e., direct inputs to the system in the x_w, y_w plane.

$$\begin{cases} \dot{x} = v \cdot \cos\theta \\ \dot{y} = v \cdot \sin\theta \\ \dot{\theta} = \omega \end{cases}$$

- where $v = \omega_u\, r$ is the linear velocity of the unicycle, ω_u is the wheel angular velocity, r is wheel radius whereas the second one is the steering velocity denoted by ω.
- It can be seen, that velocity component in the direction perpendicular to the direction of movement is always zero this is known as the non-slip condition

$$\dot{x} \cdot sin\theta - \dot{y} \cdot cos\theta = 0$$



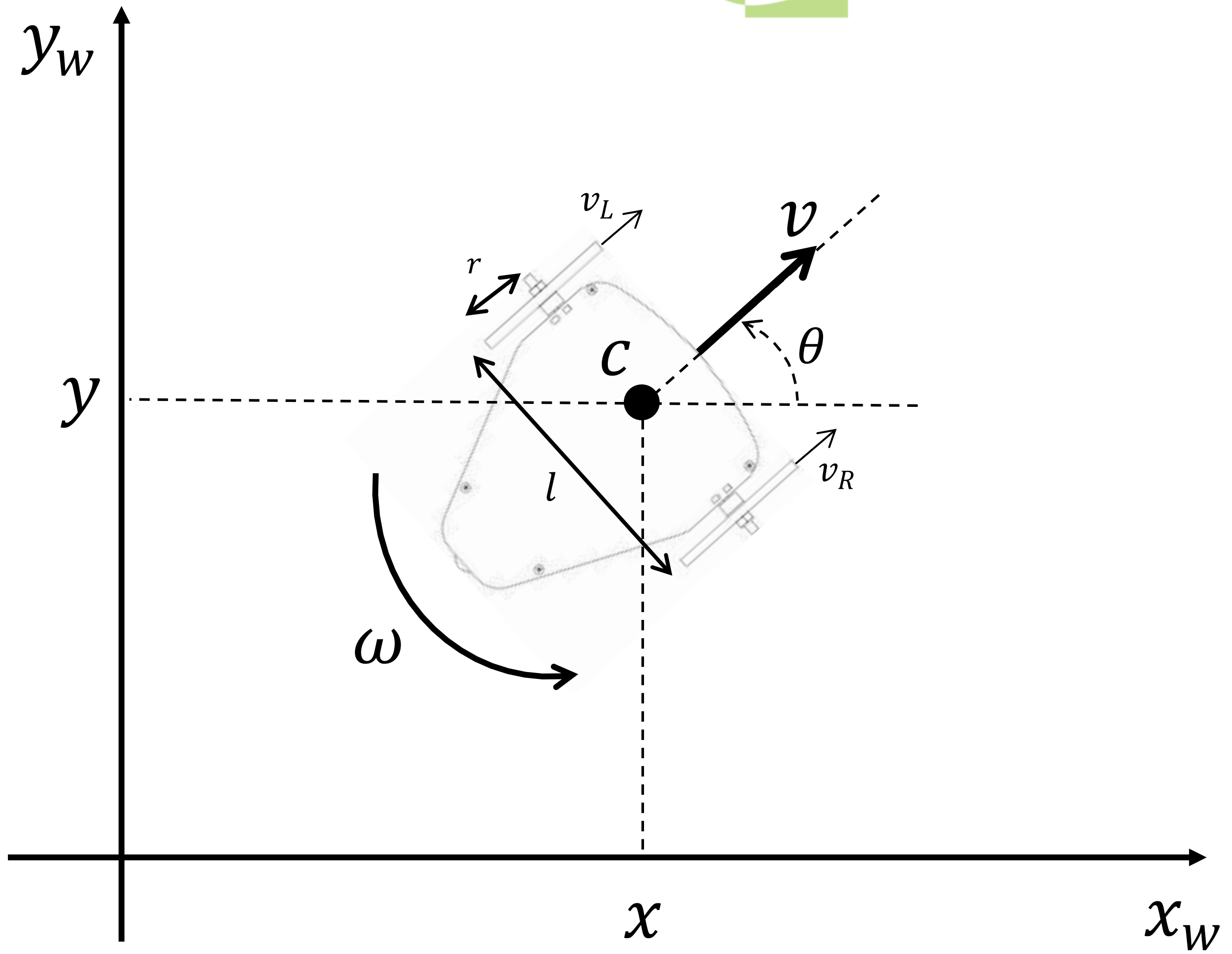
Unicycle side and top views

Differential Drive Kinematics

- For the case of a differential drive system, we can extend the idea of the Unicycle.
- To do this it is necessary to estimate the forward velocity v and the angular velocity ω.
- For this case, the resultant forward velocity v through C may be reasoned as an average of the two forward wheel velocities given by

$$v = \left(\frac{v_R + v_L}{2}\right) = r\left(\frac{\omega_R + \omega_L}{2}\right)$$

  - where r is the radius of the wheel and ω_R, ω_L are the angular velocities of the left and right wheels, respectively.



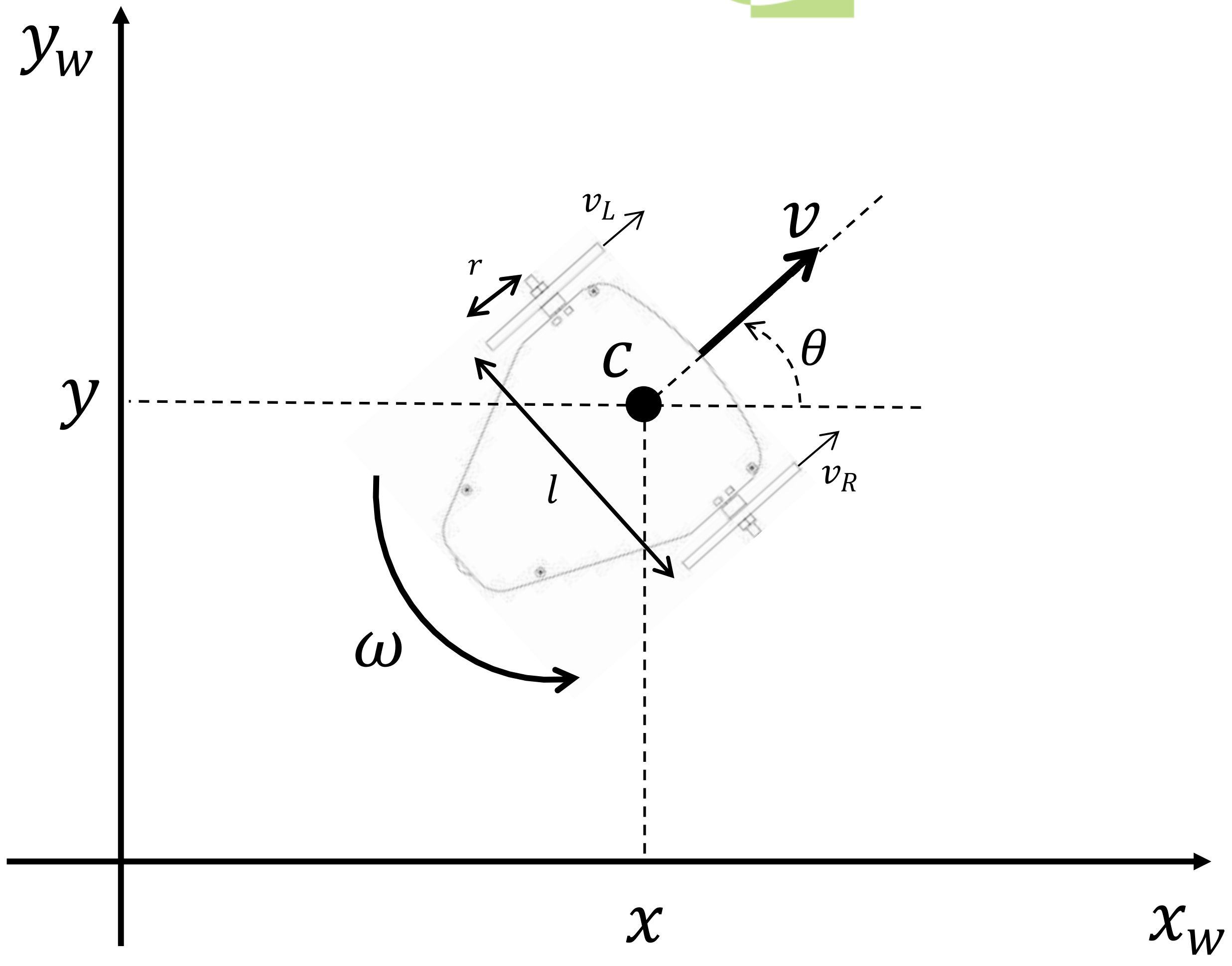
Differential Drive Robot
Puzzlebot ©.

Differential Drive Kinematics

- The resultant angular velocity ω (steering velocity), may also be reasoned as proportional to the difference between wheel velocities but inversely proportional to distance between the wheels, i.e.,

$$\omega = \left(\frac{v_R - v_L}{l}\right) = r\left(\frac{\omega_R - \omega_L}{l}\right)$$

  - where r is the radius of the wheel, l is the distance between wheels (wheelbase) and ω_R, ω_L are the angular velocities of the left and right wheels, respectively.



Differential Drive Robot
Puzzlebot ©.

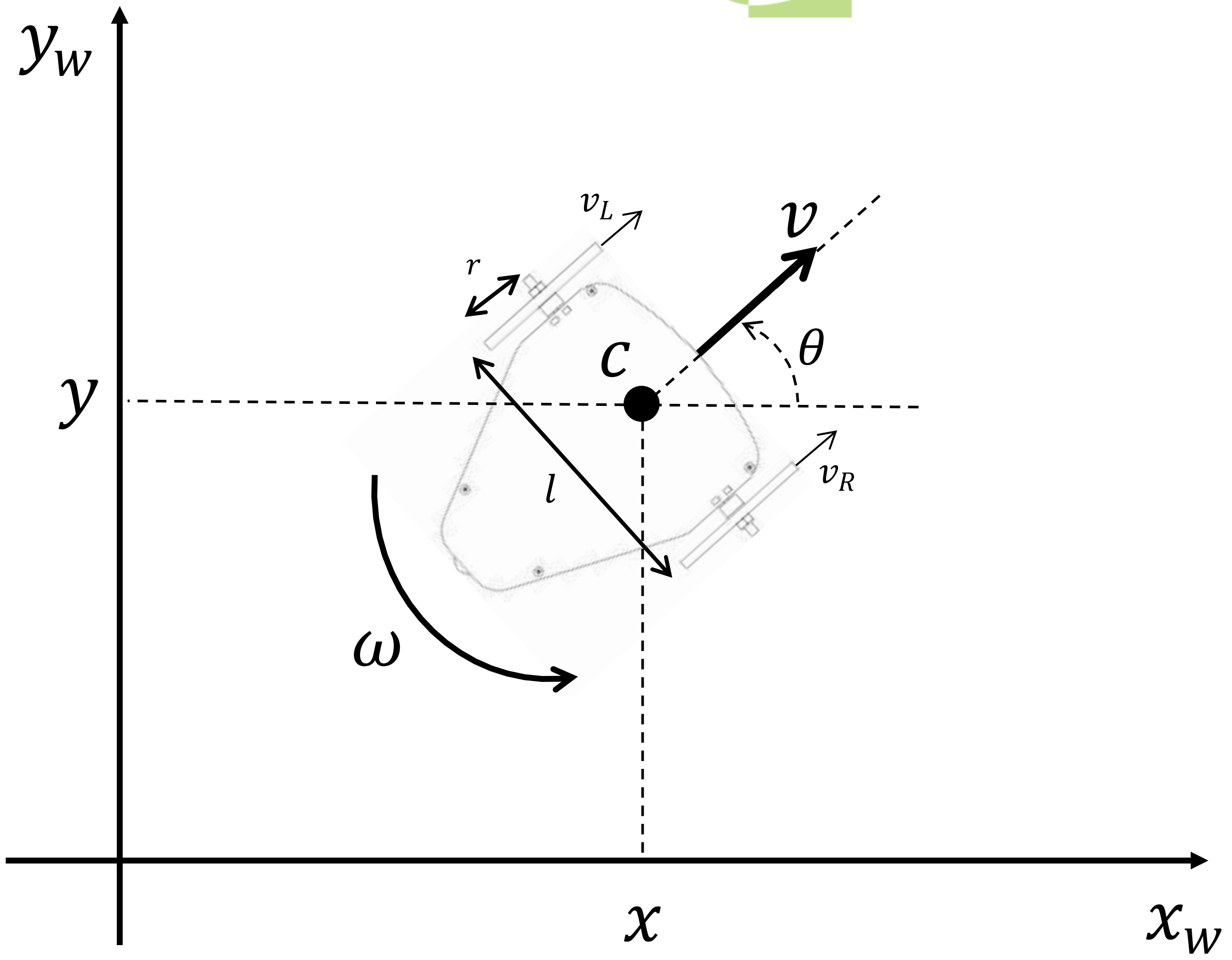
Differential Drive Kinematics

- Thus, just like the unicycle, the configuration transition equations may be given as

$$\begin{cases} \dot{x} = v \cdot cos\theta \\ \dot{y} = v \cdot sin\theta \\ \dot{\theta} = \omega \end{cases}$$

- Replacing v and ω we obtain

$$\begin{cases} \dot{x} = r\left(\frac{\omega_R + \omega_L}{2}\right) \cdot \cos\theta \\ \dot{y} = r\left(\frac{\omega_R + \omega_L}{2}\right) \cdot \sin\theta \\ \dot{\theta} = r\left(\frac{\omega_R - \omega_L}{l}\right) \end{cases}$$



Differential Drive Robot
Puzzlebot ©.

Differential Drive Kinematics

- One of the important parameters of a system is the *pose* or *posture* ($\mathbf{s}_k$), for this case it consists on the robot position and orientation with respect to a frame of reference (world frame).

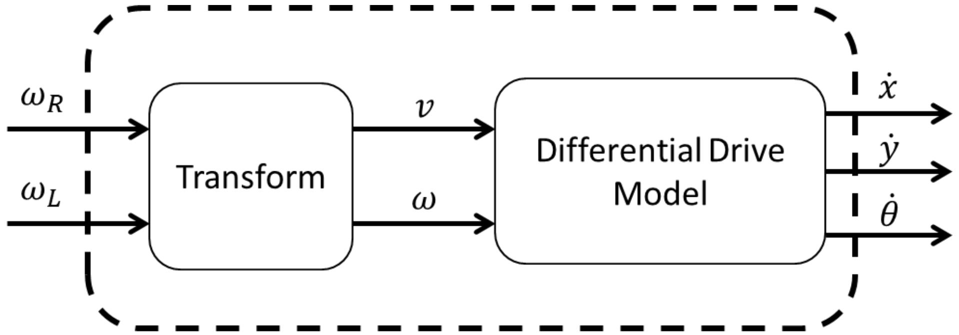
$$\mathbf{s} = [s_x \quad s_y \quad s_\theta]^T$$

- The kinematic model of a differential drive robot can then be stated in terms of the robot *pose* ($\mathbf{s}$), as follows

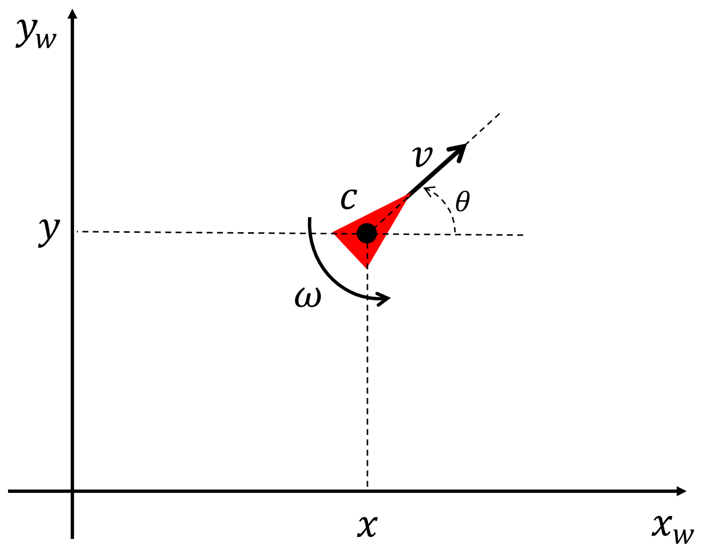
$$\frac{d}{dt}\begin{bmatrix} s_x \\ s_y \\ s_\theta \end{bmatrix} = \begin{bmatrix} \cos(s_\theta) & 0 \\ \sin(s_\theta) & 0 \\ 0 & 1 \end{bmatrix}\begin{bmatrix} v \\ \omega \end{bmatrix}$$

- where, the inputs to the system $\mathbf{u} = [v \quad \omega]^T$, are given by

$$\begin{bmatrix} v \\ \omega \end{bmatrix} = \begin{bmatrix} \frac{r}{2} & \frac{r}{2} \\ \frac{r}{l} & -\frac{r}{l} \end{bmatrix}\begin{bmatrix} \omega_R \\ \omega_L \end{bmatrix}$$



Kinematic Model



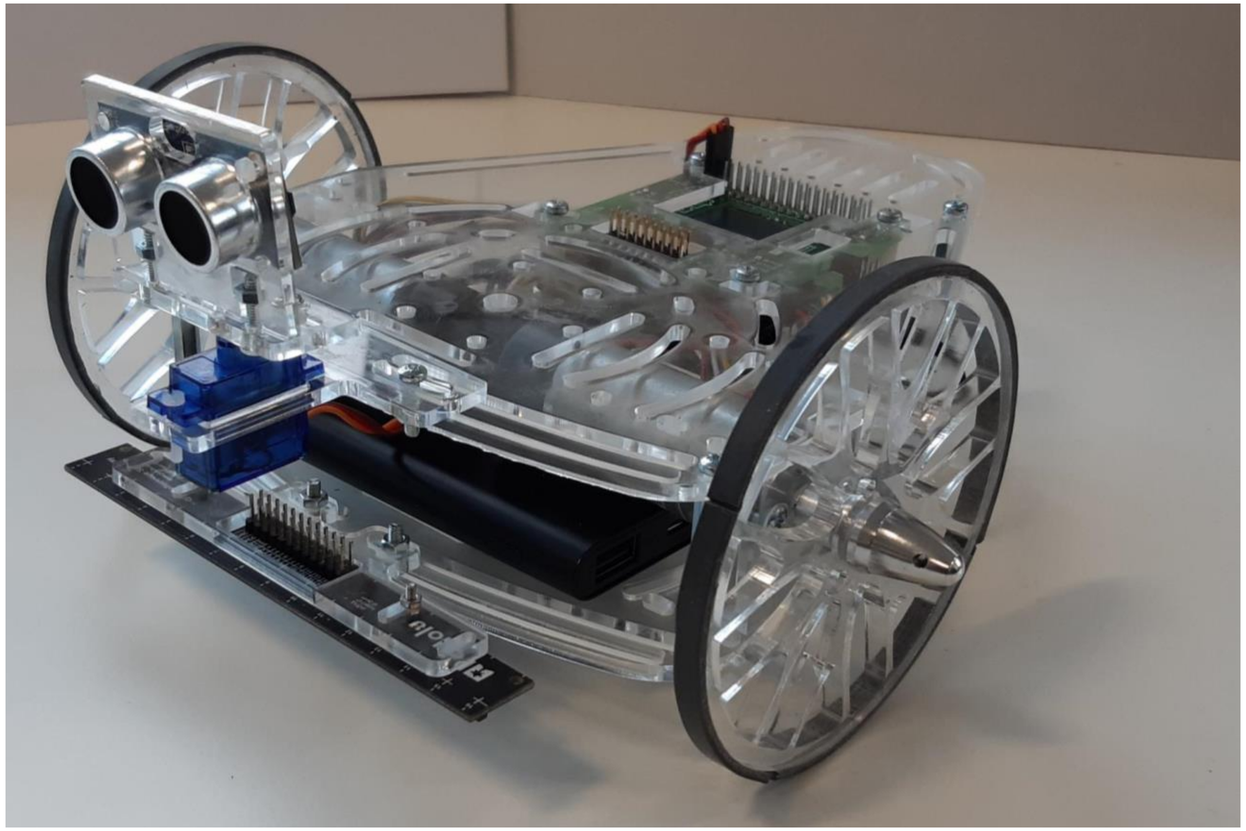
Differential Drive Robot Representation.

Differential Drive Sensors and Actuators

- As stated before, the kinematic model of a differential drive does not consider the different physical characteristics and forces of the robot.

- This type of model is used when testing high level robotic algorithms (control) such as path planning, dead reckoning localization, trajectory tracking, AI, etc.

- Differential drive robots in reality, have sensors and actuators (motors, encoders, etc.) that allow us to control them to reach the correct speed required to correctly perform its functions.

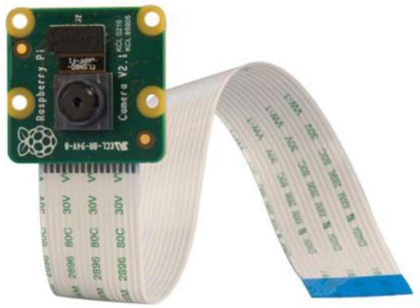


Differential Drive Robot Sensors and Actuators.

Differential Drive Sensors and Actuators

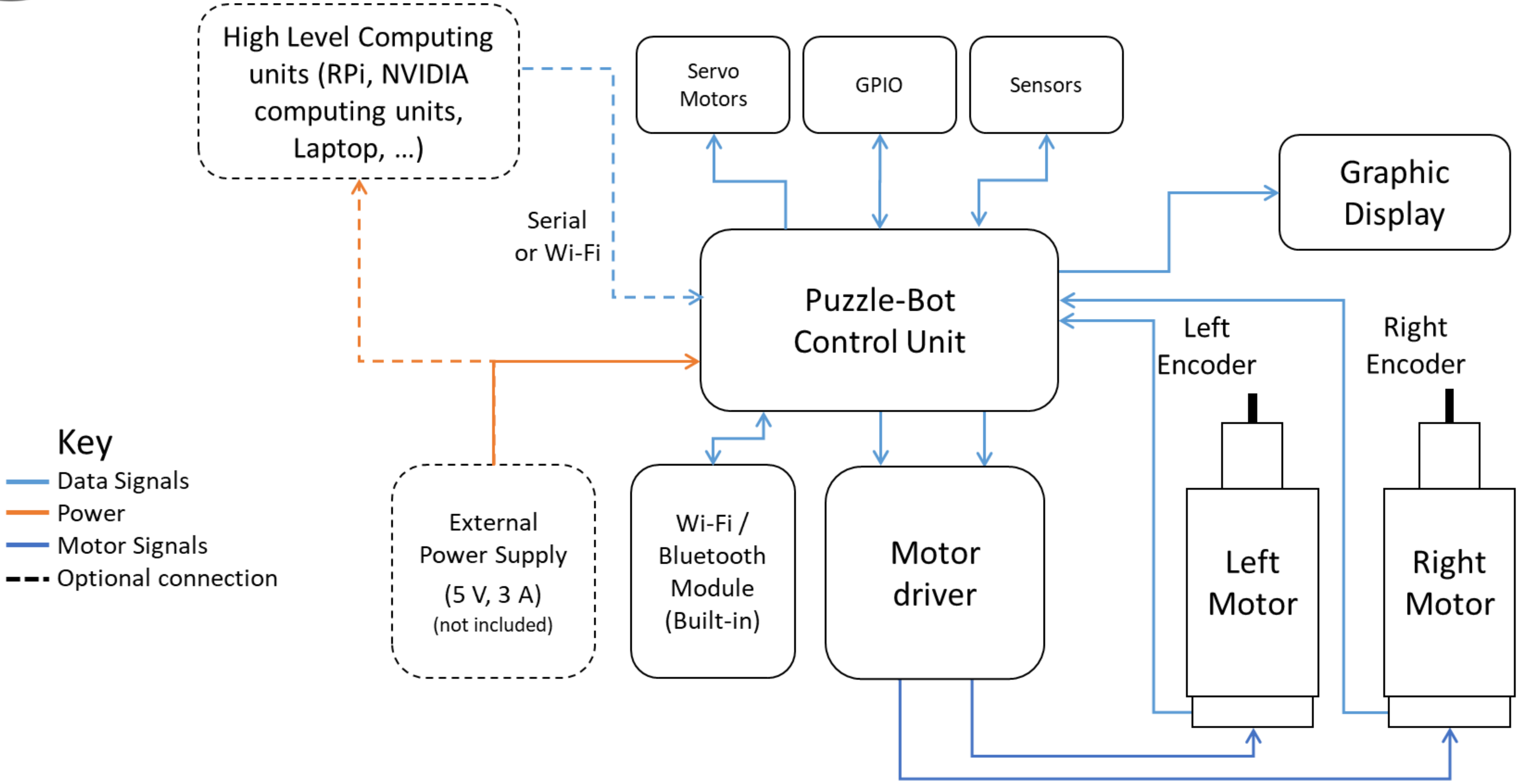
- For the case of the sensors, they can be classified as Exteroceptive and Proprioceptive.

- **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.



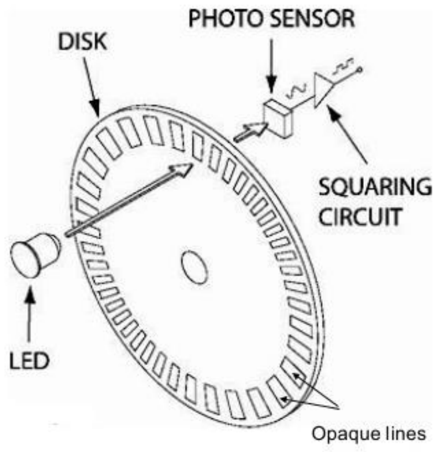
Sensors and Actuators.

Differential Drive Sensors and Actuators

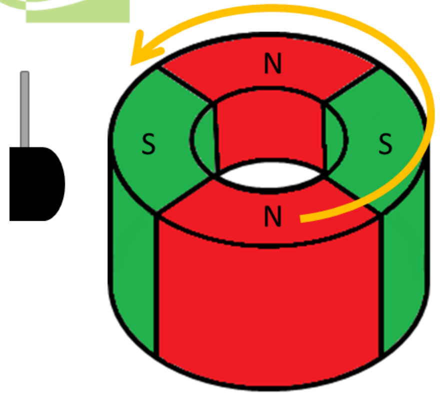


Encoders

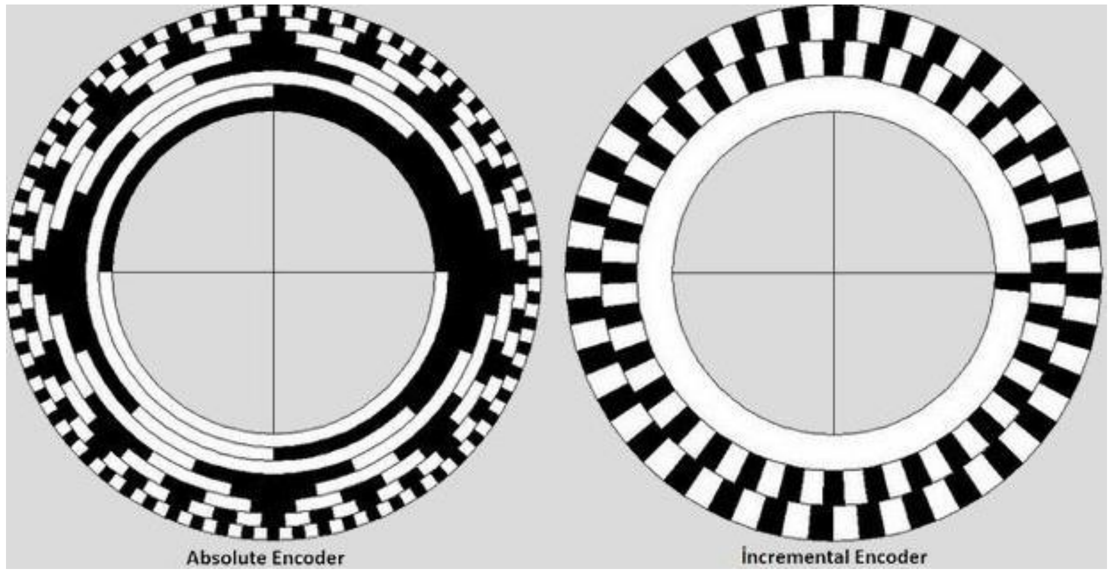
- Device that converts the angular position of a shaft (motor shaft) to an analogue or digital signal.
  - Absolute: Indicates the position of the shaft at all times by producing a unique digital code for each angle (Angle transducers).
  - Incremental: Record the changes in position of the motor shaft 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

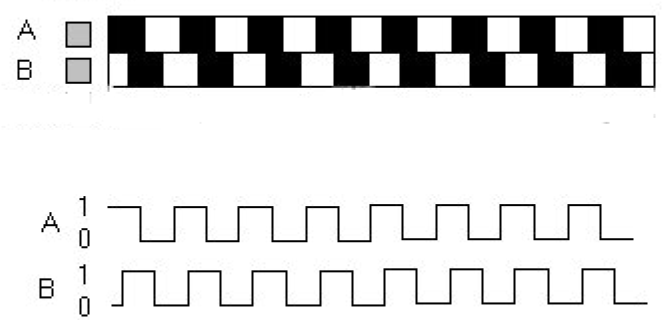


Encoders

- 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.

- PuzzleBot uses 2 incremental dual channel, quadrature encoders with 400 Pulses per revolution, attached to the motor shafts to estimate the speed of the motors.

- By counting the number of pulses that occur in a certain period of time (Δt) it is possible to estimate the velocity of the motor.

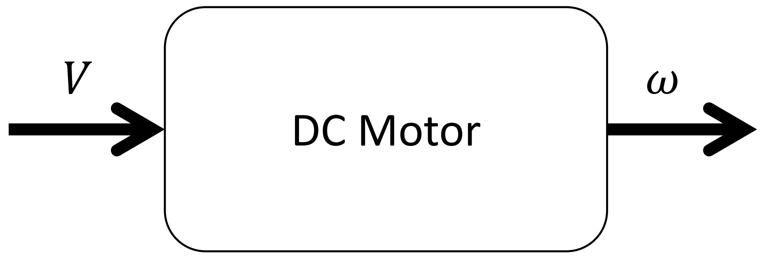


Quadrature Encoder Output

DC Motors

- 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.
- In control and robotics DC motors are usually modeled as second order systems. The input to this system is the voltage (V) and the output is the angular velocity (ω).

DC Brushed Motor with encoder

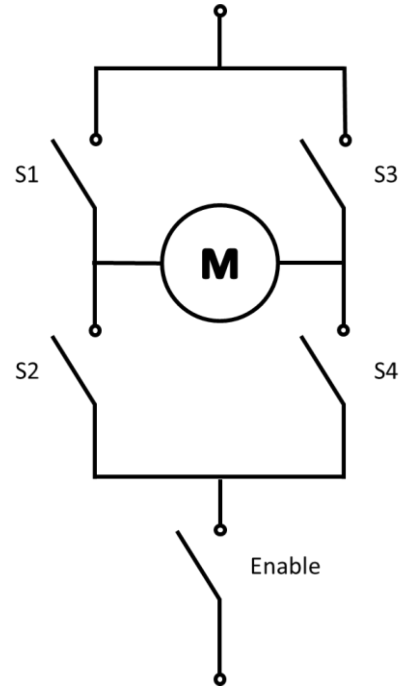


Motor Driver

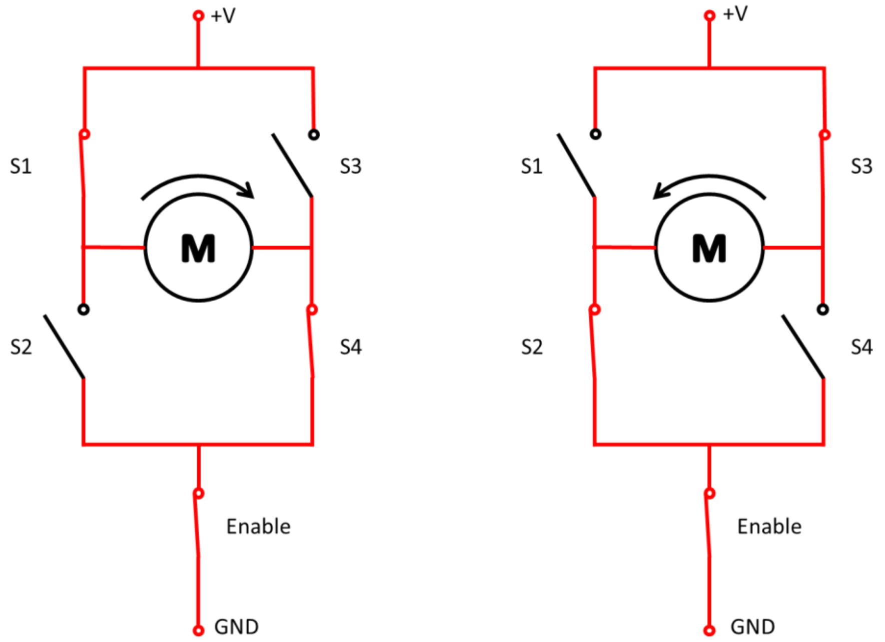
- A H-bridge is an electronic circuit that switches the polarity of a voltage applied to a load. They work using a combination of switching components (mechanical switches, transistors, etc.) as shown in the diagram to change the polarity to the load.

- H-Bridge Drivers are some of the most common motor drivers used in robotic applications such as 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

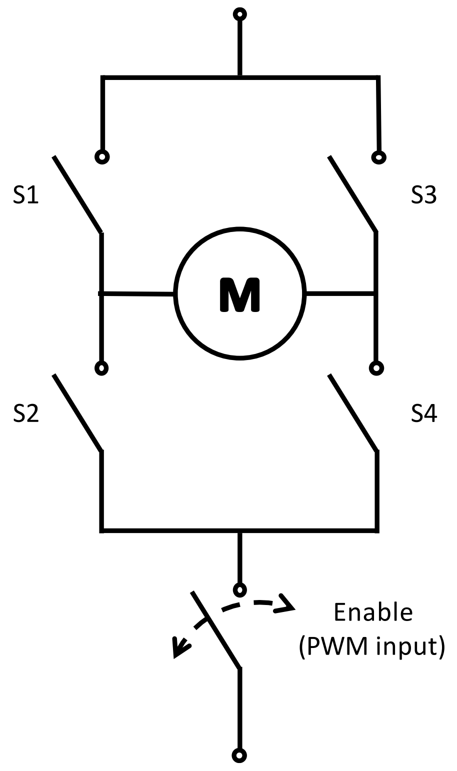
- Another capability of the motor driver is to regulate the angular speed of the motor.

- 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.

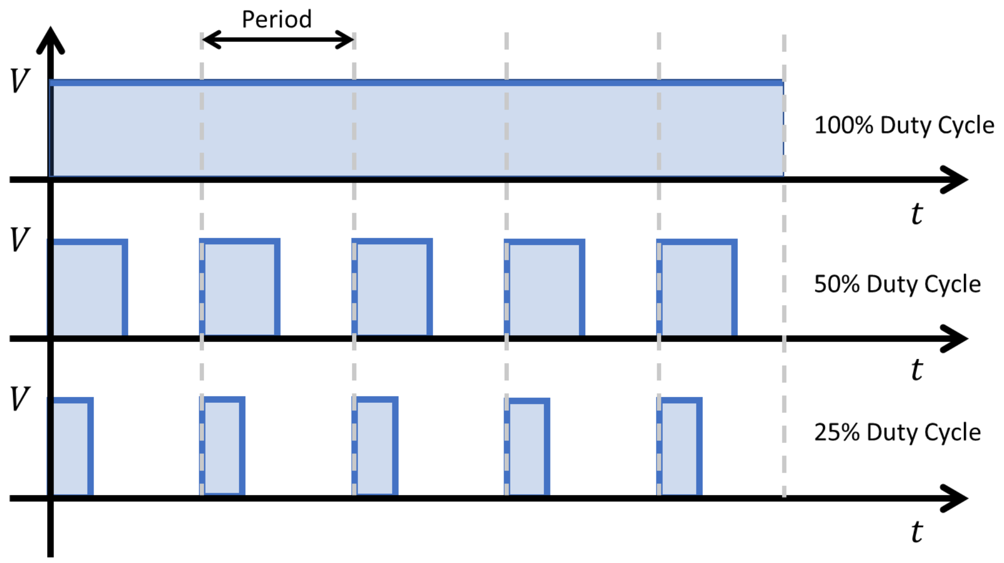
-

- 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.



PWM Input

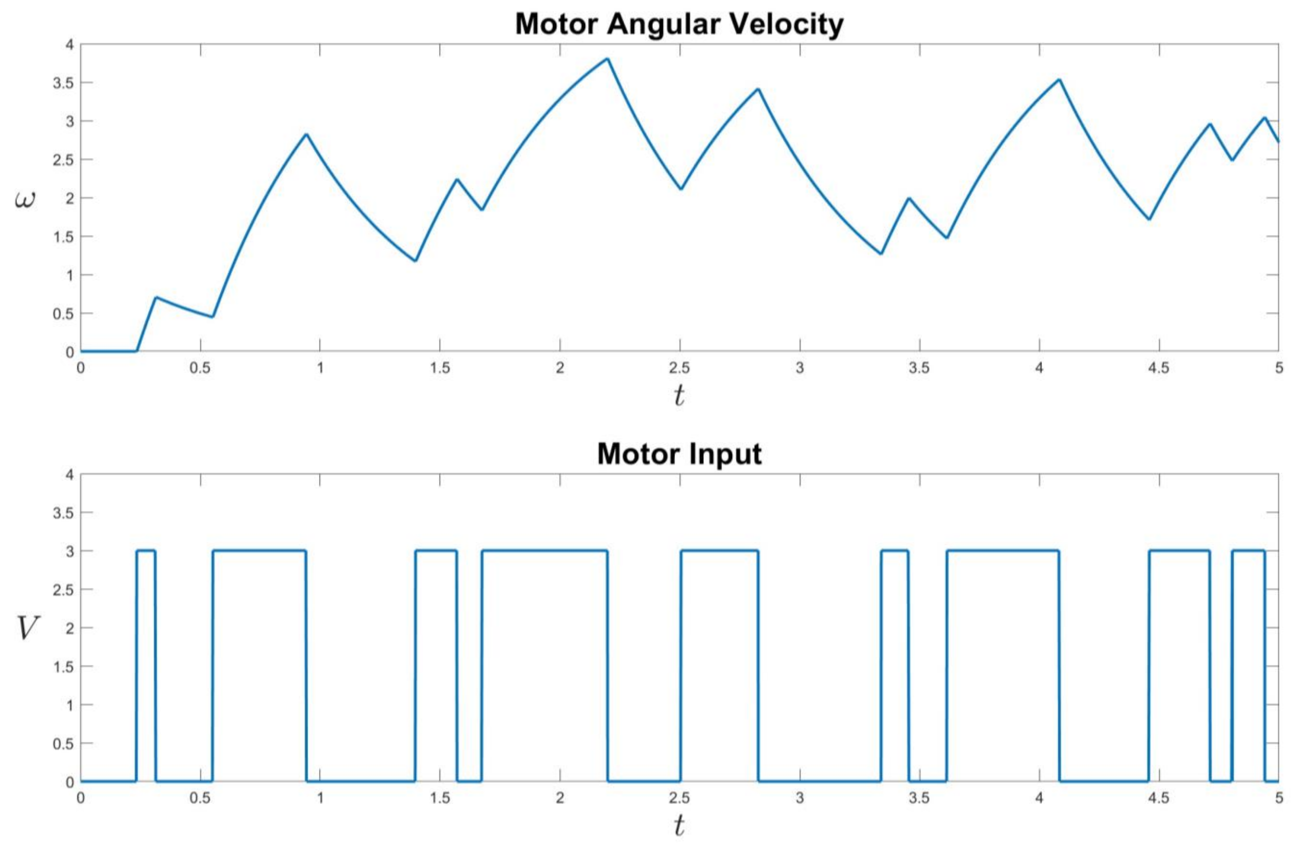


PWM Duty Cycle

Motor Driver

- Given that the motor can be modeled 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 behavior, can be used to control the power give to the motor and therefore controlling the motor angular speed.



Motor Input/Output

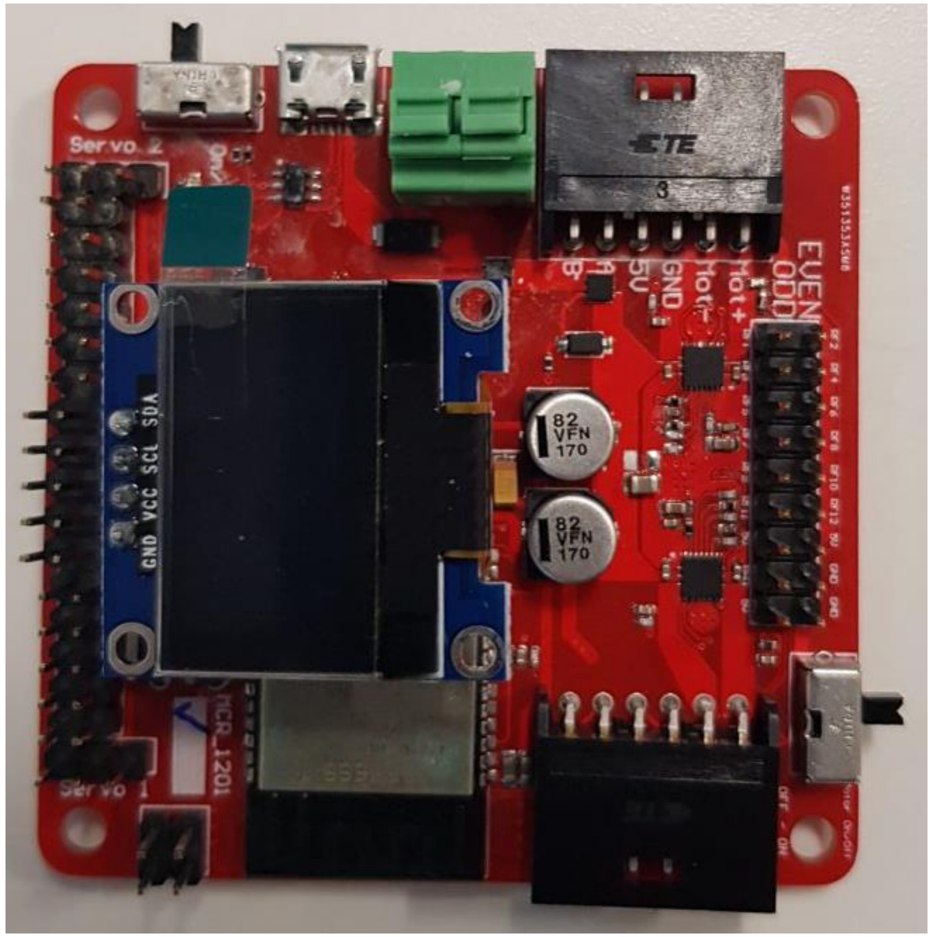
MCU

- 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.

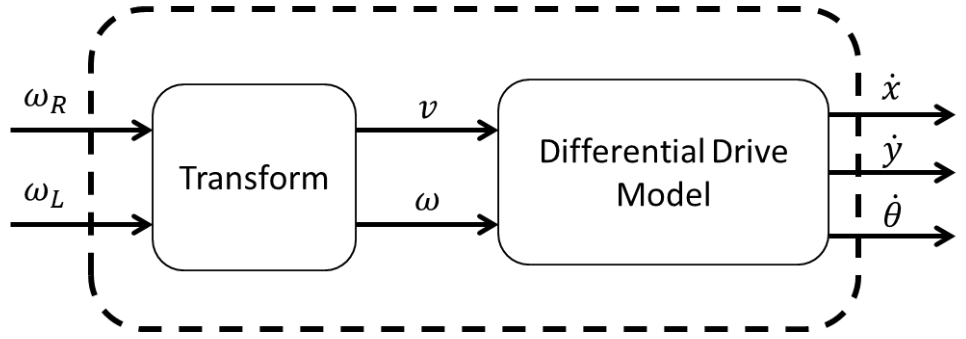
- A typical microcontroller includes a processor, memory and input/output (I/O) peripherals on a single chip.

- 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

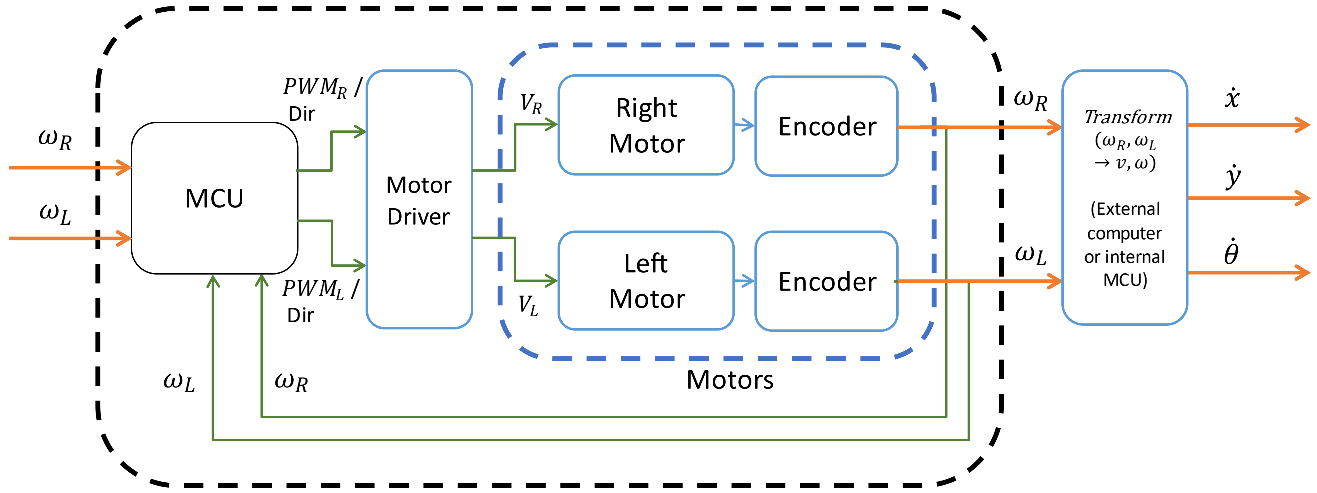


The Mobile Robot

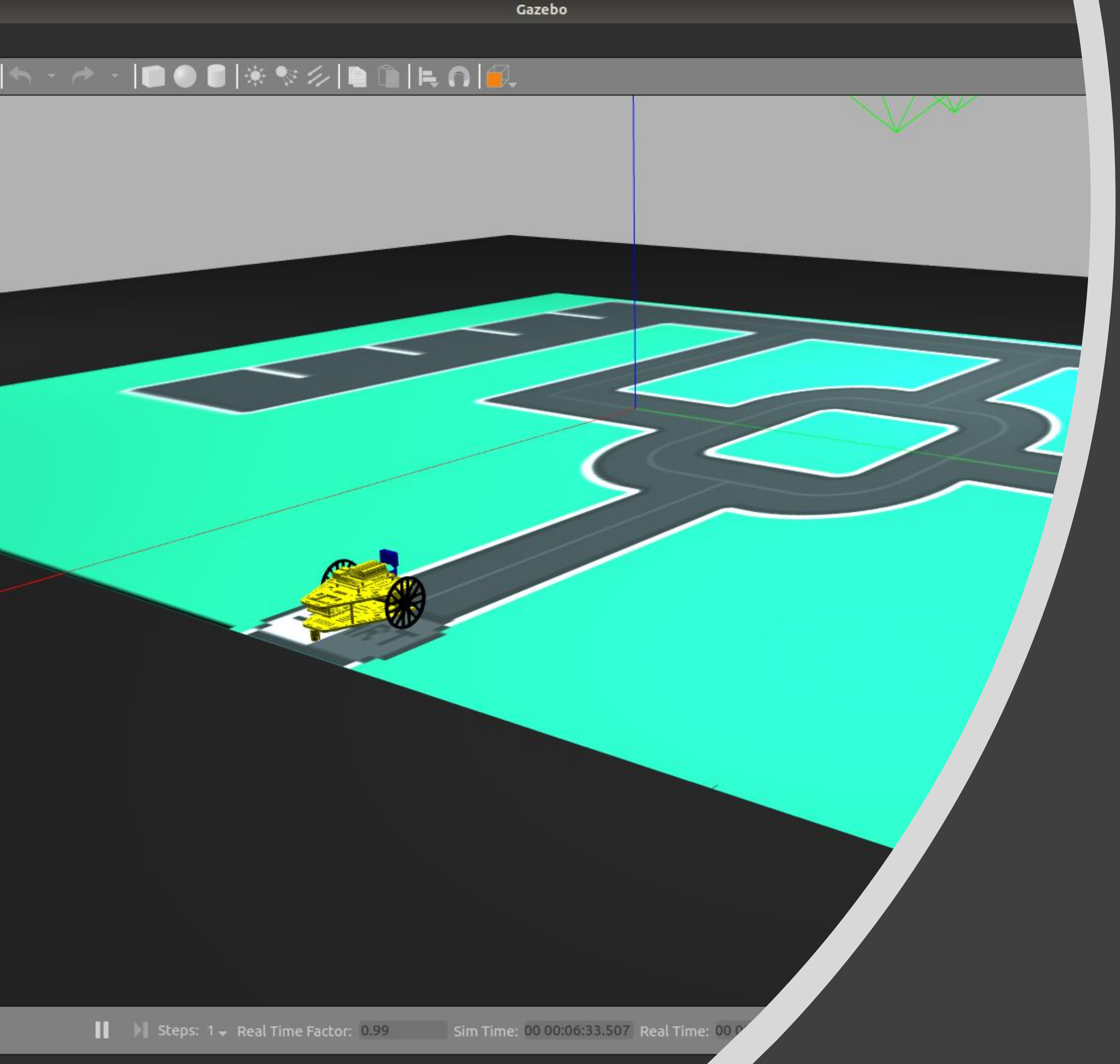
- The real robot, in comparison with the kinematic model, requires the usage of the previously shown sensors and actuators to work.
- For the real robot, a closed control loop for each of the motors is required, in order to reach the required velocities, set by the user.
- In robotics this is called 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 and outputs are the same, but the inner loop controller and actuators will determine how close the Real Robot will resemble the Kinematic Model.



Kinematic Model



Real Robot Diagram*



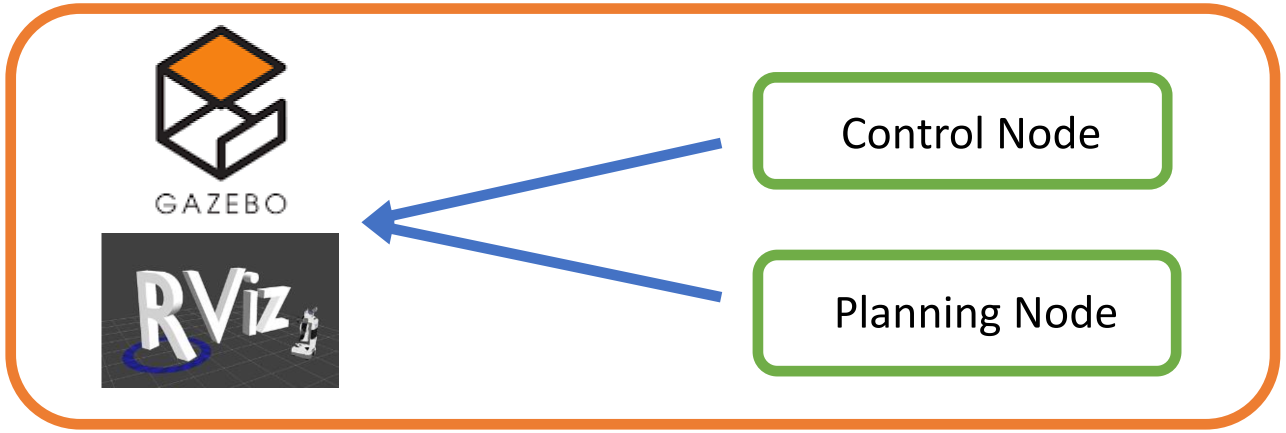
ROS visualisation and simulation tools

ROS tools

Visualization vs Simulation

In ROS, we can describe a robot and its behaviour. Some tools can help us to test our algorithms without the need of having a robot at hand.

The most common tools available are RVIZ and Gazebo, but they differ from each other. One will allow us to display information about my robot, and the other will enable simulation.



Gazebo and Rviz Comparison

Key features

RVIZ

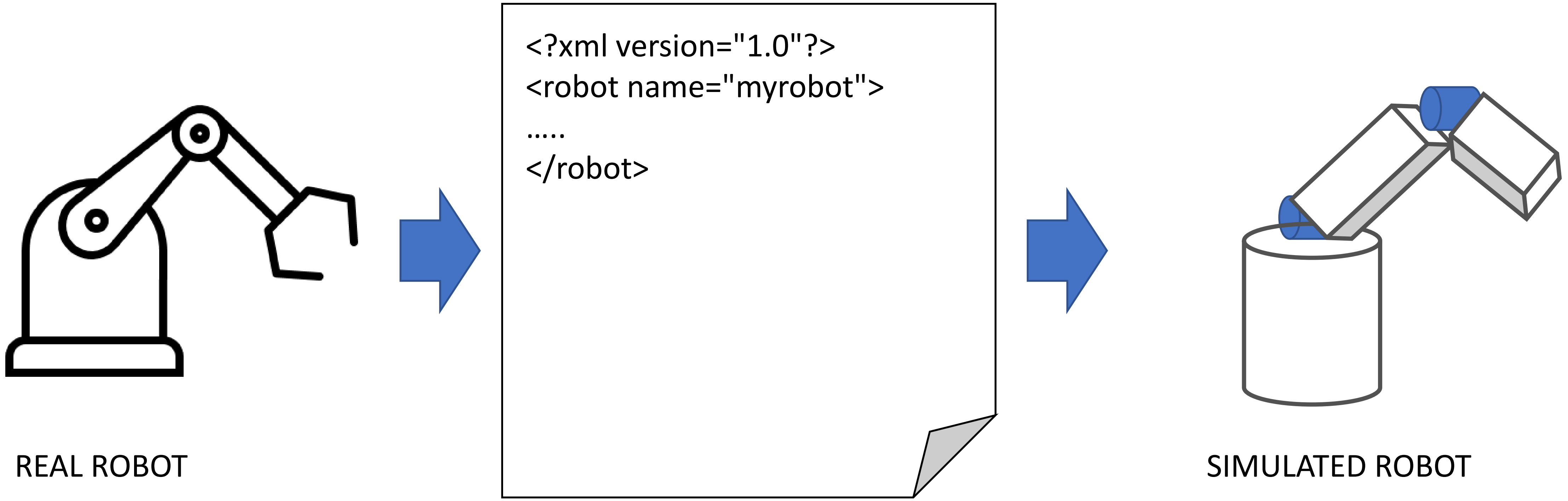
- 3D Visualisation tool
- Uses information to describe what the world around could be.
- It can display sensor information

Gazebo

- Robot simulator
- Interacts with the world as in reality (gravity, friction, etc)
- Can add plugins (describe sensor behaviours interacting with the world)

Robot modelling

Overview



```
<?xml version="1.0"?>
<robot name="myrobot">
.....
</robot>
```

URDF file

Gazebo and Rviz Comparison

Models: URDF vs SDF Model

URDF Model

(Universal Robot Description Format)

- ROS compatible (RVIZ)
- Gazebo compatible with tags
- Xacro compatible
- Describes only robots

SDF Model

(Simulation Description format)

- Gazebo compatible only
- Can describe robots, 3D objects and 3D worlds
- Can add plugins (describe sensor behaviours)

Conclusion: You probably need both

Robot modelling

ROS Puzzlebot files.

Puzzlebot model files

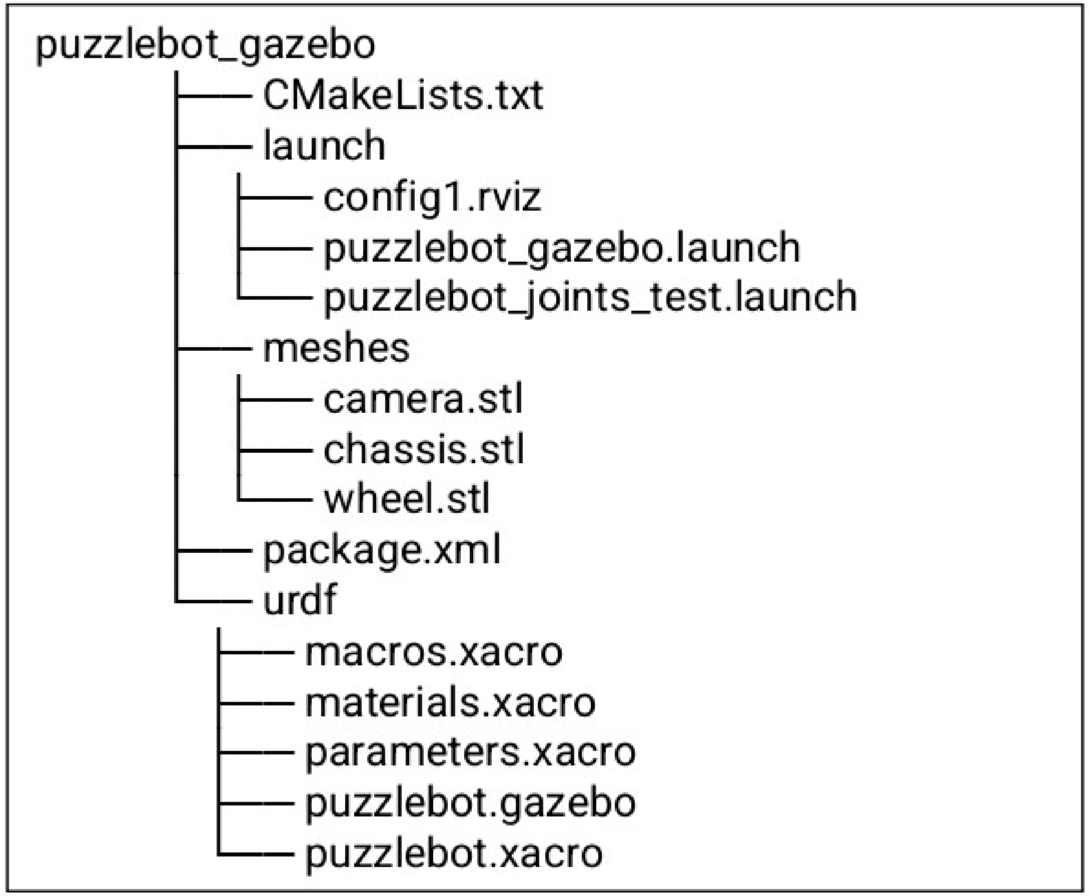
```
catkin_ws
    ├─ puzzlebot_gazebo – (or description) contains robot model description files.
    ├─ puzzlebot_world – contains files world description and models files
    ├─ puzzlebot_control – contains robot control files.
```

Robot modelling

ROS Puzzlebot files.

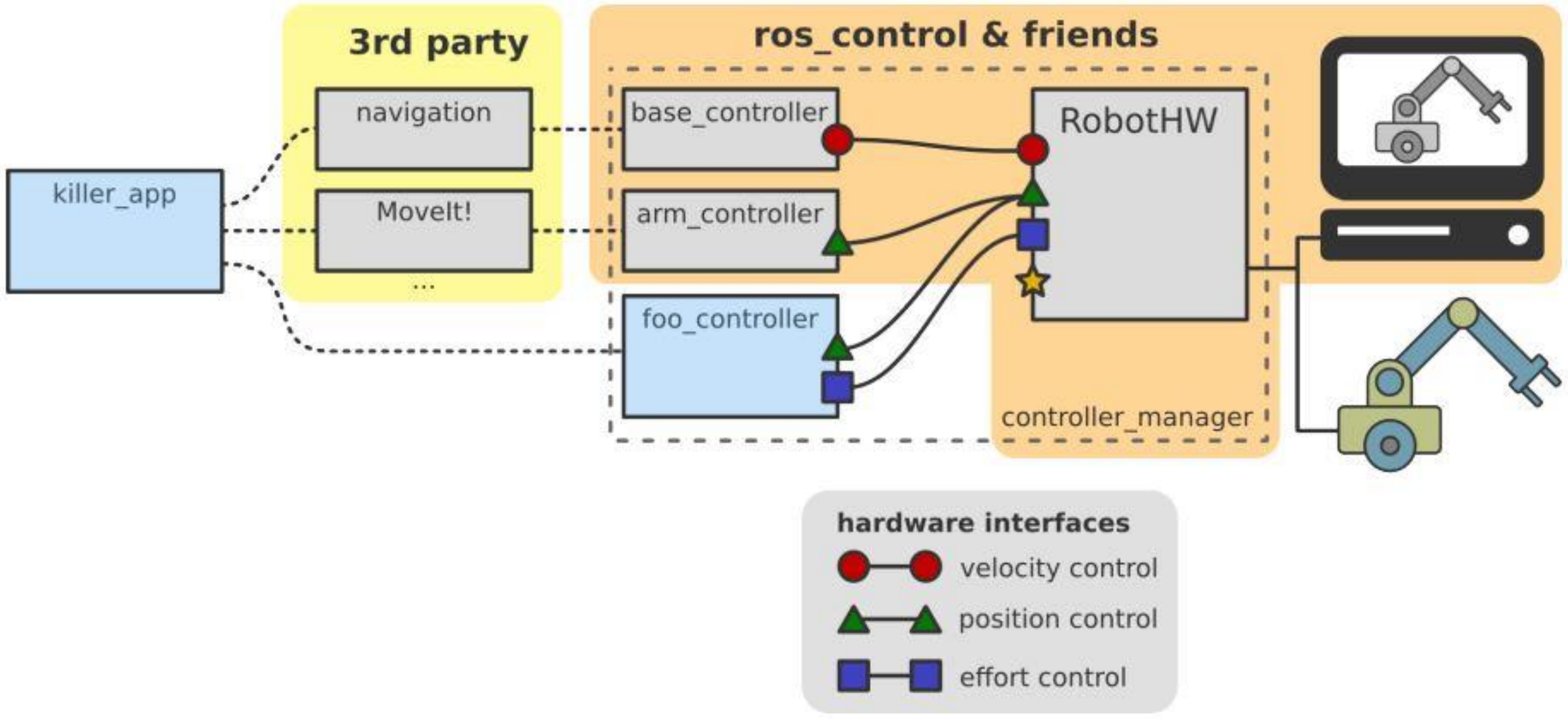
Puzzlebot model files

```
catkin_ws
    ├── puzzlebot_gazebo
    ├── puzzlebot_world
    ├── puzzlebot_control.
```



```
puzzlebot_gazebo
        ├── CMakeLists.txt
        ├── launch
        │     ├── config1.rviz
        │     ├── puzzlebot_gazebo.launch
        │     └── puzzlebot_joints_test.launch
        ├── meshes
        │     ├── camera.stl
        │     ├── chassis.stl
        │     └── wheel.stl
        ├── package.xml
        └── urdf
              ├── macros.xacro
              ├── materials.xacro
              ├── parameters.xacro
              ├── puzzlebot.gazebo
              └── puzzlebot.xacro
```

ROS Control
Overview



Gazebo

Ros Control files

Files required

```
puzzlebot_control
    ├── CMakeLists.txt
    ├── config
    │   ├── pid.yaml
    │   ├── puzzlebot_control.yaml
    │   └── puzzlebot_diff_control.yaml
    ├── include
    │   └── puzzlebot_control
    │       └── Shared.hpp
    ├── launch
    │   └── puzzlebot_control.launch
    ├── package.xml
    └── src
       └── puzzlebot_control_node.cpp
```

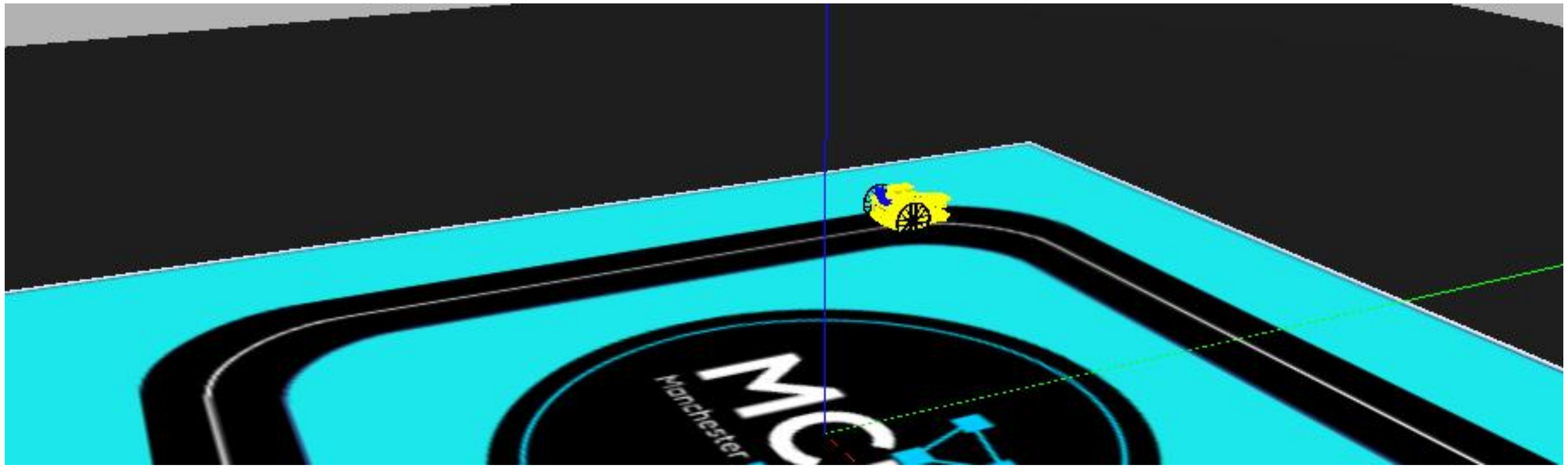
These files are here for reference only. This example could vary depending on the controller used.

Activity 1

Spawn Puzzlebot model in Gazebo.

- Download the folders From the GitHub repository for the lecture.
- Add files to your catkin workspace, and compile them.
- Use the launch file to launch your robot

```
roslaunch puzzlebot_world puzzlebot_tec_simple_world.launch
```

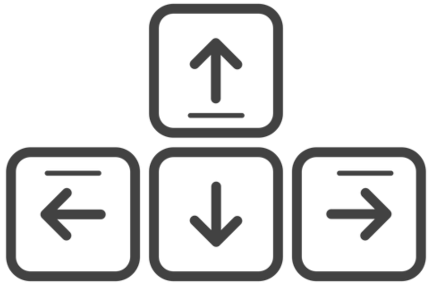


Activity 2

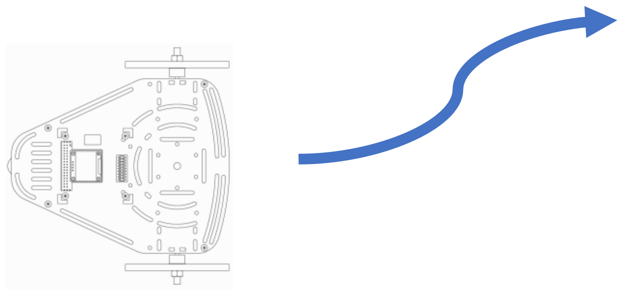
Teleoperate a PuzzleBot

- Install the teleop package using:

```
sudo apt-get install ros-noetic-teleop-twist-keyboard
```



Input in the terminal the commands to move the robot



See how the robot traverses accordingly the environment.

Gazebo

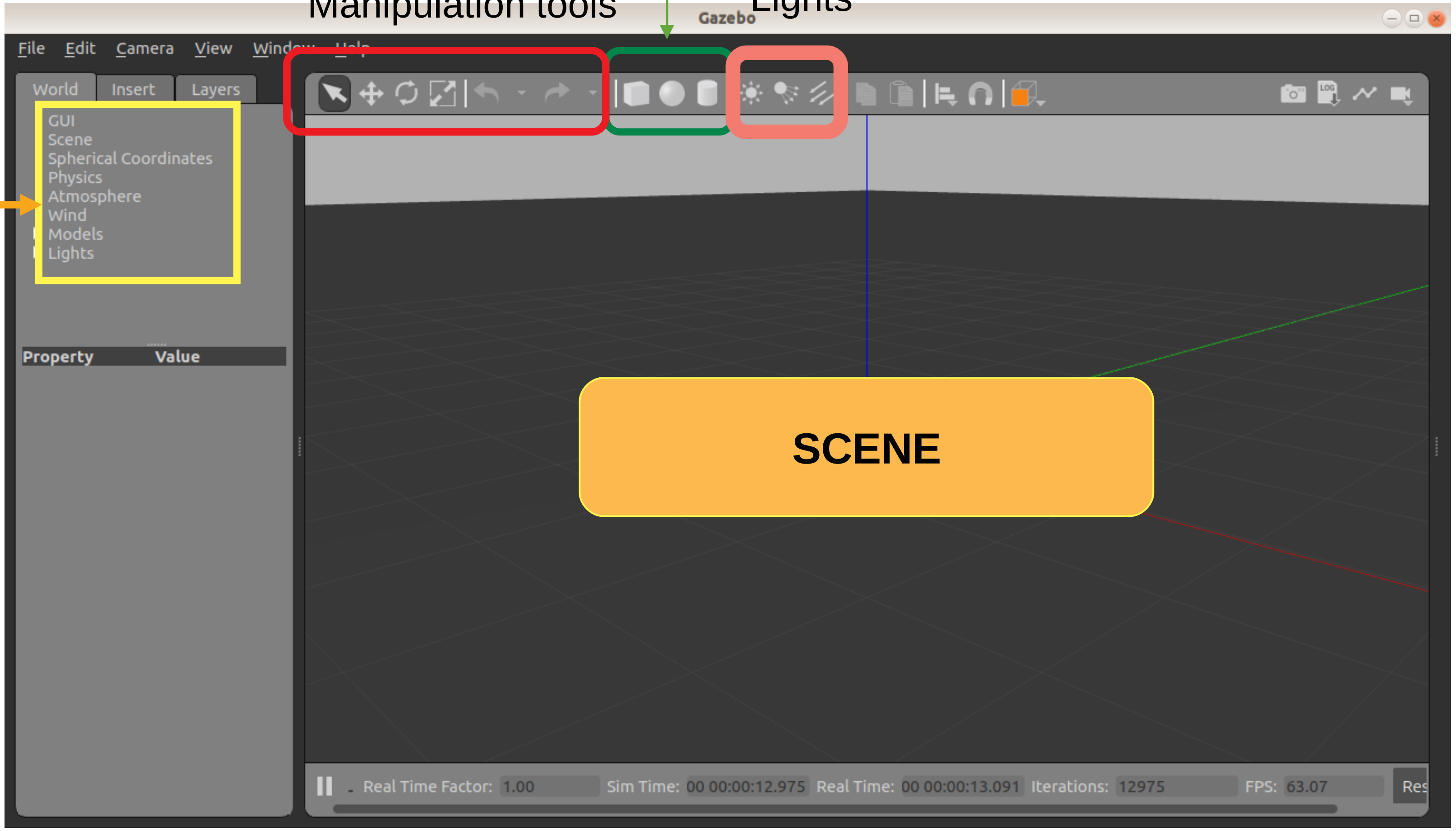
Interface Introduction

Create shapes

Manipulation tools

Lights

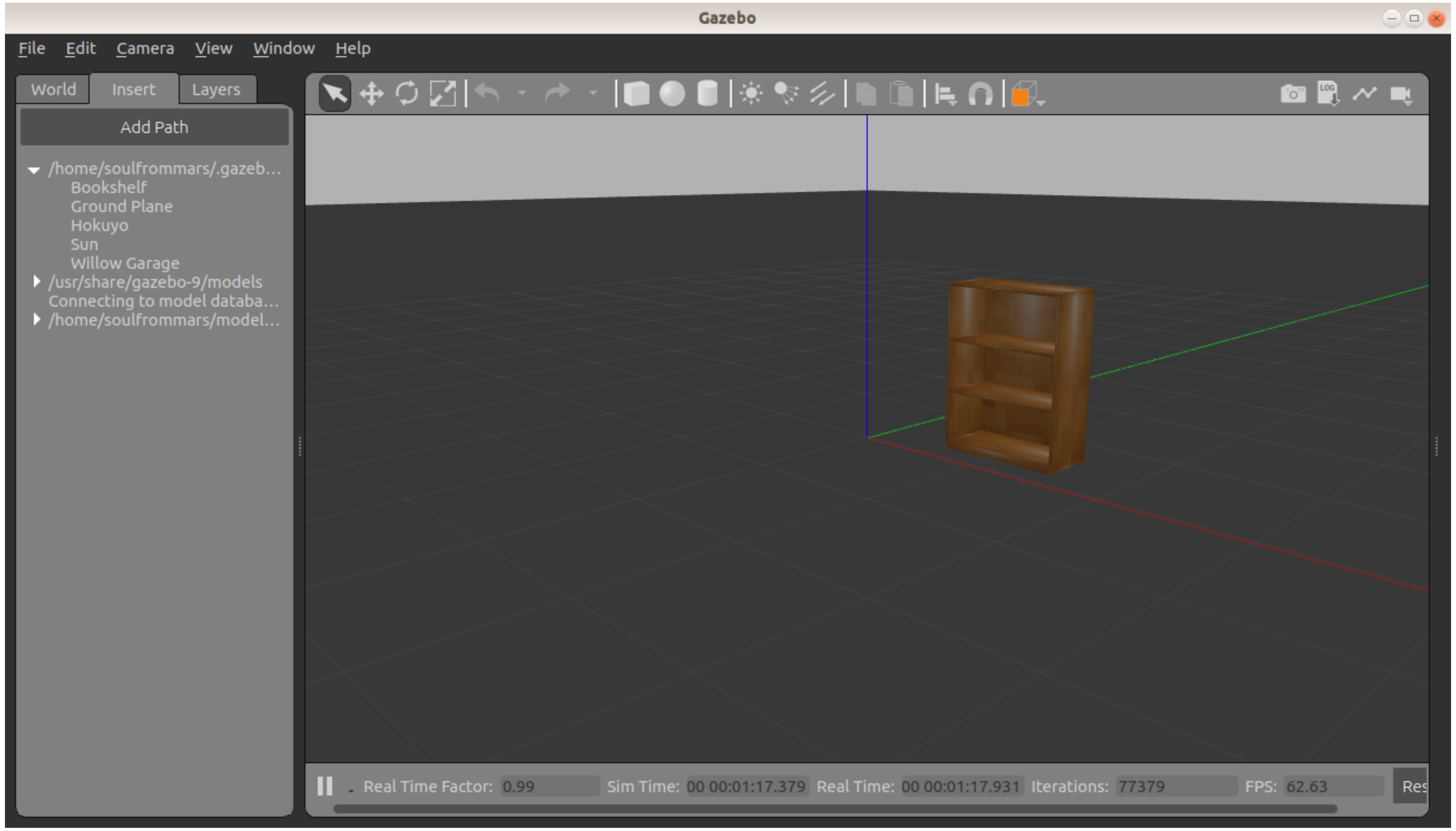
Elements in the world



More details in here

Gazebo

How to spawn a library model

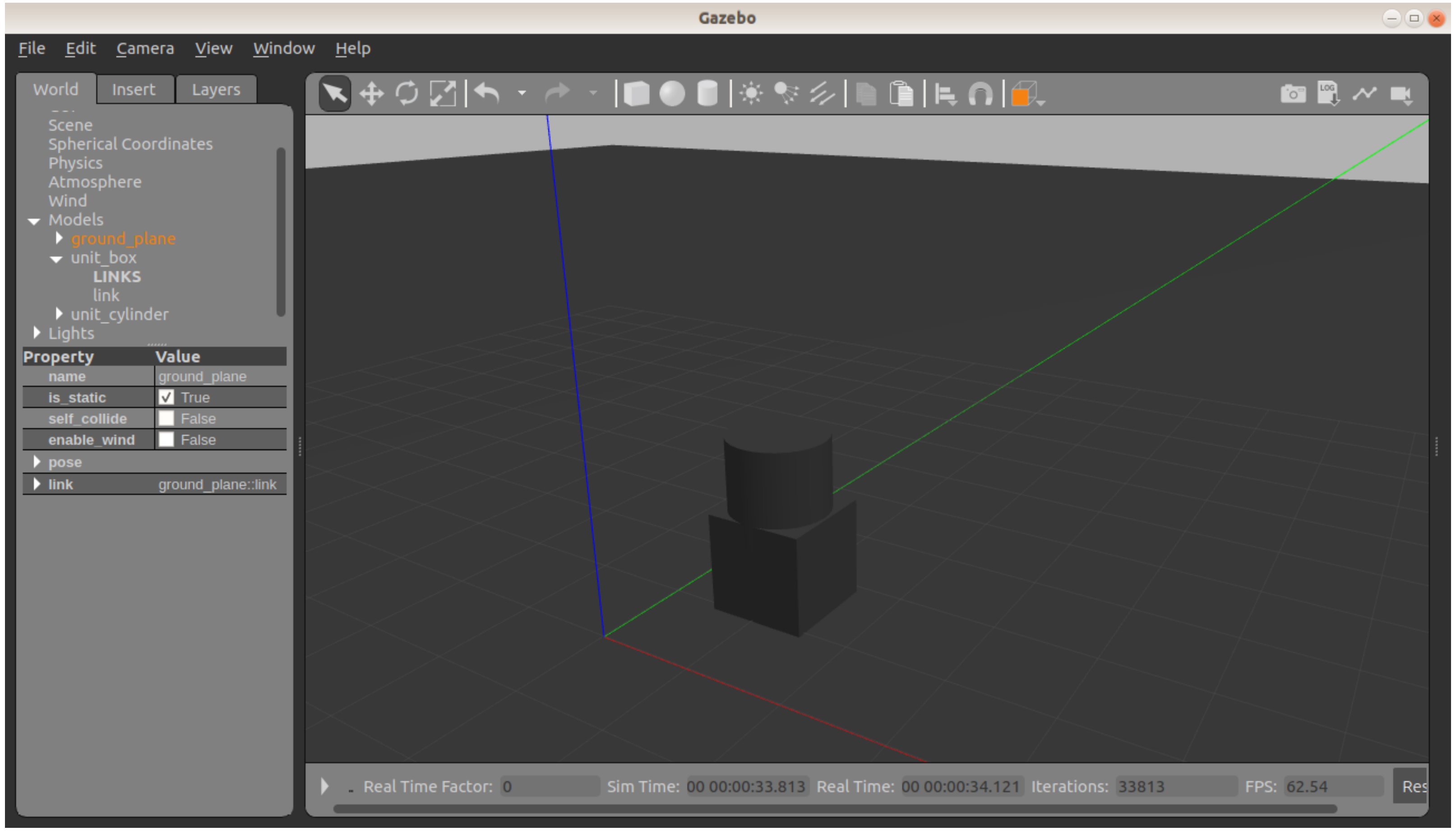

Gazebo
File Edit Camera View Window Help
World Insert Layers
Add Path
/home/soulfrommars/.gazeb...
Bookshelf
Ground Plane
Hokuyo
Sun
Willow Garage
/usr/share/gazebo-9/models
Connecting to model databa...
/home/soulfrommars/model...
Real Time Factor: 0.99 Sim Time: 00 00:01:17.379 Real Time: 00 00:01:17.931 Iterations: 77379 FPS: 62.63 Res

In the second tab, Objects from the gazebo Library are found.

Some other files can be added to a local library. Usually in home/.gazebo folder, but if required a model saved in the package can be loaded with a launch file.

Gazebo

How to create an object



Property	Value
name	ground_plane
is_static	✓ True
self_collide	False
enable_wind	False

Real Time Factor: 0 Sim Time: 00 00:00:33.813 Real Time: 00 00:00:34.121 Iterations: 33813 FPS: 62.54

It is possible to create models, links, or simple objects via the tools and Also the model editor.

The important part is to remember this is all saved in SDF (a gazebo format).

Also, the model editor cannot
Edit models or objects done Using URDF or Xacro.

Activity 3

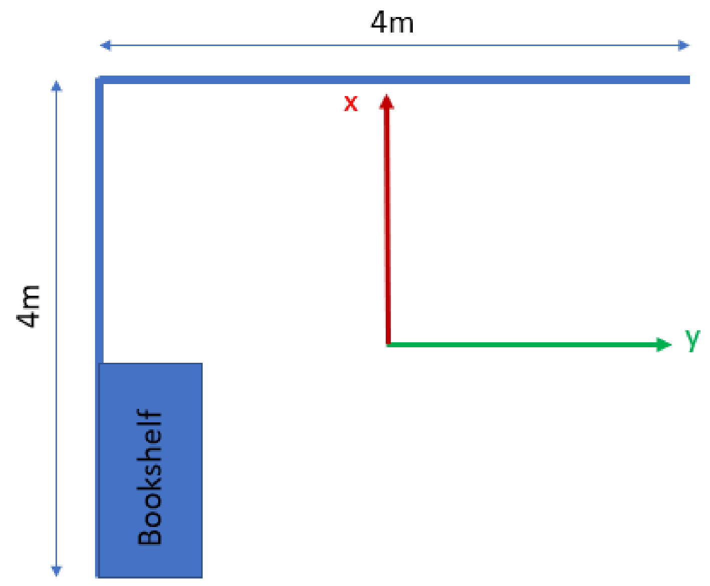
Spawn Puzzlebot model in your world.

- Open Gazebo using the command:

```
gazebo
```

- Using the interface, create a world with the information described in diagram 1.
- Save the world with the name “custom_room1.world”.
- Use the following command to spawn the puzzle in the world you saved.

```
roslaunch puzzlebot_world puzzlebot_tec_simple_world_edited.launch
```


4m
4m
x
y
Bookshelf